

---

# DATABASE PROGRAMMING WITH PL SQL 1 / 2

---

ORACLE ACADEMY



24 DE OCTUBRE DE 2022  
UNIVERSIDAD POLITECNICA DE AGUASCALIENTES  
Juan Carlos Herrera Hernández

# Contenido

Section 1 – Fundamentals.....	3
1-1 Introduction to PL/SQL.....	3
1-2 Benefits of PL/SQL.....	4
1-3 Creating PL/SQL Blocks .....	5
Section 2 – Defining Variables and Datatypes .....	8
2-1 Using Variables in PL/SQL .....	8
2-2 Recognizing PL/SQL Lexical Units.....	8
2-3 Recognizing Data Types .....	9
2-4 Using Scalar Data Types .....	10
2-5 Writing PL/SQL Executable Statements.....	10
2-6 Nested Blocks and Variable Scope.....	11
2-7 Good Programming Practices .....	12
Section 3 – Using SQL in PL/SQL .....	13
3-1 Review of SQL DML.....	13
3-2 Retrieving Data in PL/SQL .....	14
3-3 Manipulating Data in PL/SQL.....	14
3-4 Using Transaction Control Statements .....	15
Section 4 – Program Structures to Control Execution Flow.....	16
4-1 Conditional Control: IF Statements .....	16
4-2 Conditional Control: Case Statements.....	17
4-3 Iterative Control: Basic Loops .....	18
4-4 Iterative Control: While and For Loops.....	18
4-5 Iterative Control: Nested Loops.....	19
S e m e s t e r 1 , M I D T E R M.....	19
Section 5 – Using Cursors and Parameters .....	20
5-1 Introduction to Explicit Cursors .....	20
5-2 Using Explicit Cursor Attributes .....	21
5-3 Cursor FOR Loops.....	22
5-4 Cursors with Parameters .....	22
5-5 Using Cursors For Update .....	23
5-6 Using Multiple Cursors.....	24
Section 6 – Using Composite Datatypes .....	25
6-1 User-Defined Records .....	25
6-2 Indexing Tables of Records .....	26

Section 7 – Exception Handling.....	28
7-1 Handling Exceptions.....	28
7-2 Trapping Oracle Server Exceptions .....	28
7-3 Trapping User-Defined Exceptions .....	30
7-4 Recognizing the Scope of Exceptions.....	30
Section 8 – Using and Managing Procedures.....	32
8-1 Creating Procedures .....	32
8-2 Using Parameters in Procedures.....	33
8-3 Passing Parameters.....	33
Section 9 – Using and Managing Functions .....	35
9-1 Creating Functions .....	35
9-2 Using Functions in SQL Statements .....	36
9-3 Review of the Data Dictionary .....	37
9-4 Managing Procedures and Functions .....	37
9-5 Review of Object Privileges.....	38
9-6 Using Invoker's Rights and Autonomous Transactions.....	40
Semester 1, FINAL .....	40

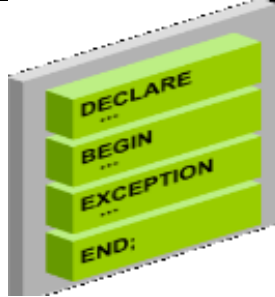
## Section 1 – Fundamentals

### 1-1 Introduction to PL/SQL

<b>Procedural Constructs</b>	Programming language features such as reusable/callable program units, modular blocks, cursors, constants, variables, assignment statements, conditional control statements, and loops.
<b>PL/SQL</b>	Oracle Corporations standard procedural language for relational databases which allows basic program logic and control flow to be combined with SQL statements.

MySQL	ORACLE
<pre>set @prom = (select avg(salary) from employees); select avg(salary) into @prom from employees; select @prom from dual;</pre>	<pre>define prom = (select avg(salary) from employees); select &amp;prom promedio from dual;</pre>
<pre>set @stmt = 'SELECT employee_id, salary FROM employees'; execute IMMEDIATE @stmt;</pre>	<pre>DEFINE colname=salary ; SELECT employee_id, &amp;colname FROM employees;</pre>

#### Procedural Language extension to SQL.

<pre>-- anonymous procedures set serveroutput on declare     prom number; begin     select avg(salary) into prom from employees;     dbms_output.put_line('promedio: '    prom); end; /</pre>	 <p>The basic unit in a PL/SQL program is a block.</p> <p>All PL/SQL programs consist of blocks as modules.</p>
---	---

<pre>set serveroutput on; declare     numero number := 5;     cadena varchar2(100);     i integer default 1; begin     loop         cadena := numero    ' x '                i    ' = '    (numero*i);         dbms_output.put_line(cadena);         EXIT WHEN i &gt;= 10;         i := i + 1;     end loop; end;</pre>	<pre>CREATE OR REPLACE PROCEDURE tabla(numero NUMBER) IS     cadena VARCHAR2(100); BEGIN     FOR i IN reverse 1..10 LOOP         cadena := numero                ' x '    i    ' = '    (numero*i);         dbms_output.put_line(cadena);     END LOOP; END; /</pre>
---	--

<pre>CREATE OR REPLACE FUNCTION factorial(m number) RETURN number IS     r number default 1;     n number := m; BEGIN     while (n &gt; 0) loop         r := r * n;         n := n - 1;     end loop;     return r; END; /</pre>	<pre>DECLARE     CURSOR cursor_employees IS         SELECT * FROM employees             where department_id = 60;     v_contador number := 0; BEGIN     dbms_output.put_line('No. LastName');     FOR r_emp in cursor_employees LOOP         v_contador := v_contador + 1 ;         dbms_output.put_line(v_contador    ' '                r_emp.last_name);     END LOOP; END;</pre>
--	--

```

DECLARE
CURSOR cursor_employees IS SELECT * FROM employees;
BEGIN
  FOR c_emp in cursor_employees LOOP
    IF c_emp.job_id= 'SA_REP' AND c_emp.hire_date<='05-Feb-2005' THEN
      UPDATE employees SET job_id= 'SR_SA_REP'
      WHERE employee_id= c_emp.employee_id;
    ELSIF c_emp.job_id= 'MK_REP' AND c_emp.hire_date<= '05-Feb-2005' THEN
      UPDATE employees SET job_id= 'SR_MK_REP'
      WHERE employee_id= c_emp.employee_id;
    ELSIF c_emp.job_id= 'ST_CLERK' AND c_emp.hire_date<='05-Feb-2005' THEN
      UPDATE employees SET job_id= 'SR_ST_CLRK'
      WHERE employee_id= c_emp.employee_id;
    END IF;
  END LOOP;
END;

```

## 1-2 Benefits of PL/SQL

<b>Portability</b>	The ability for PL/SQL programs to run anywhere an Oracle server runs.
<b>Blocks</b>	The basic unit of PL/SQL programs - also known as modules.
<b>Exceptions</b>	An error that occurs in the database or in a user's program during runtime.

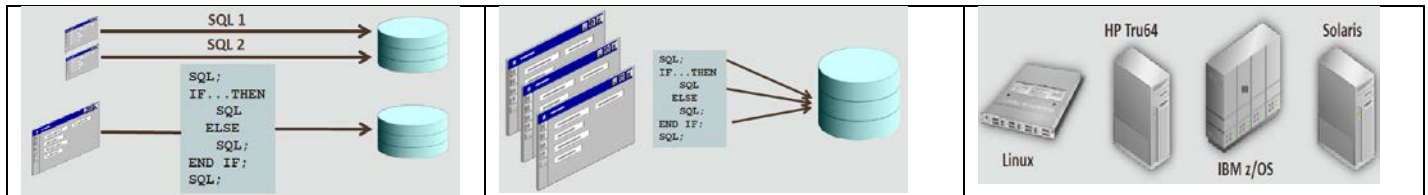
You can nest blocks inside other blocks to build powerful programs.

PL/SQL is integrated in Oracle tools, such as Oracle Forms Developer, Oracle Report Builder, and Application Express.





You can write portable program packages and create libraries that can be reused in different environments.

Exception Handling:

- If no data is found then...
- If too many rows are found then...
- If an invalid number is calculated then...



## PL/SQL in Oracle Products

Oracle Product	PL/SQL
	You can write PL/SQL code to manage application data or to manage the Oracle database itself. For example, you can write code for updating data (DML), creating data (DDL), generating reports, managing security, and so on.
	Using the Web Application Toolkit, you can create database-centric web applications written entirely or partially in PL/SQL.
	Using Forms Builder and Reports Developer, Oracle's client-side developer tools, you can build database-centric web applications and reports that include PL/SQL.
	Using a Web browser you can develop web applications that include PL/SQL.

### 1-3 Creating PL/SQL Blocks

Anonymous PL/SQL block	Unnamed blocks of code not stored in the database and do not exist after they are executed
Function	A program that computes and returns a single value
Subprograms	Named PL/SQL blocks that are stored in the database and can be declared as procedures or functions
Compiler	Software that checks and translates programs written in high-level programming languages into binary code to execute
Procedure	A program that performs an action, but does not have to return a value

Application Express	Browser-based, database-driven, application development environment.
SQL Workshop	A component of Application Express.
Application Builder	A component of Application Express.
SQL Developer	An IDE for database development and management.
JDeveloper	An IDE for Java-based development.
NetBeans	An IDE for Java, HTML5, PHP, and C++.

Anonymous Blocks	Procedure: Performs an action	Function: Computes and returns a value
[DECLARE] --variable declarations BEGIN -- statements [EXCEPTION] END;	PROCEDURE name IS --variable declarations BEGIN --statements [EXCEPTION] END;	FUNCTION name RETURN datatype IS --variable declaration(s) BEGIN --statements RETURN value; [EXCEPTION] END;

Section	Description	Inclusion
---------	-------------	-----------

Declarative (DECLARE)	Contains declarations of all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and exception sections.	Optional
Executable (BEGIN ... END;)	Contains SQL statements to retrieve data from the database and PL/SQL statements to manipulate data in the block. Must contain at least one statement.	Mandatory
Exception (EXCEPTION)	Specifies the actions to perform when errors and abnormal conditions arise in the executable section.	Optional

EXCEPTIONS
<pre> set SERVEROUTPUT ON; DECLARE     v_first_name VARCHAR2(25);     v_last_name  VARCHAR2(25); BEGIN     SELECT first_name, last_name     INTO v_first_name, v_last_name     FROM employees     WHERE last_name= 'King';     DBMS_OUTPUT.PUT_LINE ('The employee of the month is: '                               v_first_name   ' '    v_last_name   '.');  <b>EXCEPTION</b>     WHEN TOO_MANY_ROWS THEN         DBMS_OUTPUT.PUT_LINE ('Your select statement retrieved multiple rows.                                Consider using a cursor or changing the search criteria.');</pre>
<pre> END; /</pre>

<pre> CREATE OR REPLACE <b>PROCEDURE</b> print_date IS v_date VARCHAR2(30); BEGIN     SELECT TO_CHAR(SYSDATE, 'Mon DD, YYYY') INTO v_date FROM DUAL;     DBMS_OUTPUT.PUT_LINE(v_date); END;</pre>	<pre> begin     PRINT_DATE; end; /  call    print_date(); execute print_date();</pre>
---	---

<pre> CREATE OR REPLACE <b>FUNCTION</b> tomorrow(p_today IN DATE) RETURN DATE IS     v_tomorrow DATE; BEGIN     SELECT p_today+1 INTO v_tomorrow FROM DUAL;     RETURN v_tomorrow; END;</pre>	<pre> BEGIN     DBMS_OUTPUT.PUT_LINE(TOMORROW(SYSDATE)); END; /  SELECT TOMORROW(SYSDATE) FROM DUAL;</pre>
---	--

<pre> CREATE OR REPLACE <b>FUNCTION</b> factorial(m number) RETURN number IS     r number default 1;     n number := m; BEGIN     while (n &gt; 0) loop         r := r * n;         n := n - 1;     end loop;     return r; END; /</pre>	<pre> set SERVEROUTPUT ON BEGIN     DBMS_OUTPUT.PUT_LINE(factorial(5)); END; /  select factorial(5) from dual;</pre>
--	--



## Section 2 – Defining Variables and Datatypes

### 2-1 Using Variables in PL/SQL

<b>Variables</b>	Used for storage of data and manipulation of stored values.
<b>Parameters</b>	Values passed to a program by a user or by another program to customize the program.

**Identifier** [CONSTANT] **datatype** [NOT NULL] [:= *expr* | DEFAULT *expr*];

```
set SERVEROUTPUT ON
DECLARE
/* Declaracion de
   Variables */
   v_counter INTEGER := 0;
   v_contador number(3) DEFAULT 0;
   v_name VARCHAR2(20) := 'John';
   v_date Date default SYSDATE;
   c_pi constant number(5,4) := 3.1416;
   v_activo BOOLEAN := True;
BEGIN
   v_counter:= v_counter + 1;
   -- SELECT SYSDATE INTO v_date FROM DUAL;
   DBMS_OUTPUT.PUT_LINE(v_counter || ' ' || v_name || ' ' || v_date);
END;
/
```

### 2-2 Recognizing PL/SQL Lexical Units

<b>Literals</b>	An explicit numeric, character string, date, or Boolean value that is not represented by an identifier. 'UPA' != 'Upa'
<b>Delimiters</b>	Symbols that have special meaning to an Oracle database.
<b>Reserved words</b>	Words that have special meaning to an Oracle database and cannot be used as identifiers.
<b>Comments</b>	Describe the purpose and use of each code segment and are ignored by PL/SQL.
<b>Identifiers</b>	A name, up to 30 characters in length, given to a PL/SQL object. <b>Not sensitive</b> May include \$ (dollar sign), _ (underscore), or # (hashtag) . vCounter\$ = vcounter\$
<b>Lexical Units</b>	Building blocks of any PL/SQL block and are sequences of characters including letters, digits, tabs, returns, and symbols.

#### Partial List of Reserved Words

ALL	CREATE	FROM	MODIFY	SELECT
ALTER	DATE	GROUP	NOT	SYNONYM
AND	DEFAULT	HAVING	NULL	SYSDATE
ANY	DELETE	IN	NUMBER	TABLE
AS	DESC	INDEX	OR	THEN
ASC	DISTINCT	INSERT	ORDER	UPDATE
BETWEEN	DROP	INTEGER	RENAME	VALUES
CHAR	ELSE	INTO	ROW	VARCHAR2
COLUMN	EXISTS	IS	ROWID	VIEW
COMMENT	FOR	LIKE	ROWNUM	WHERE

## Delimiters

Symbol	Meaning	Symbol	Meaning
+	addition operator	<>	inequality operator
-	subtraction/negation operator	!=	inequality operator
*	multiplication operator		concatenation operator
/	division operator	--	single-line comment indicator
=	equality operator	/*	beginning comment delimiter
'	character string delimiter	*/	ending comment delimiter
;	statement terminator	**	exponent
		:=	assignment operator

## 2-3 Recognizing Data Types

<b>Object</b>	A schema object with a name, attributes, and methods.
<b>Scalar</b>	Hold a single value with no internal components.
<b>Composite</b>	Contain internal elements that are either scalar (record) or composite (record and table)
<b>Reference</b>	Hold values, called pointers, that point to a storage location.
<b>LOB</b>	Hold values, called locators, that specify the location of large objects (such as graphic images) that are stored out of line. (text, images, video, audio) up to 4GB
<b>BFILE</b>	Store large <b>binary files</b> outside of the database.
<b>BLOB</b>	Store large unstructured or structured <b>binary objects</b> .
<b>CLOB</b>	Store large blocks of <b>character</b> data in the database.
<b>NCLOB</b>	Store large blocks of single-byte or fixed width multi-byte NCHAR data in the database. National language character large object (NCLOB)

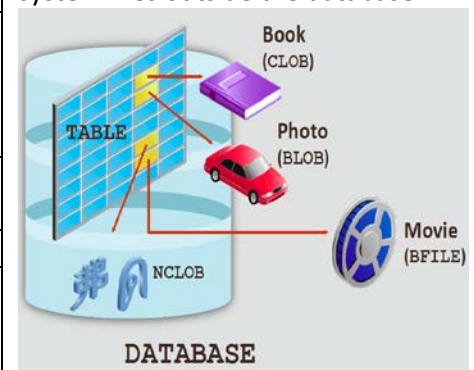
### PL/SQL supports five categories of data type

Data Type	Description
Scalar	Holds a single value with no internal elements. Character (char [1..32767], varchar2, long(2GB)) Number(number, pls_integer) Date(date, Timestamp) Boolean (True, False, Null)
Composite	Contains multiple internal elements that can be manipulated individually. Record(One row), Table, Varray RECORD v_emp_record employees%ROWTYPE; v_emp_record.first_name
Large Object (LOB)	Holds values called locators that specify the location of large objects (such as graphic images) that are stored out of line.
Reference	Holds values called pointers that point to a storage location.
Object	It is a schema object with a name, attributes, and methods. An object data type is similar to the class mechanism supported by C++ and Java.

### LOB Data Type

CLOB, BLOB, and NCLOB data is stored in the database, either inside or outside of the row.

BFILE data is stored in operating system files outside the database.



## 2-4 Using Scalar Data Types

<b>BOOLEAN</b>	A datatype that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.
<b>%TYPE</b>	Attribute used to declare a variable according to another previously declared variable or database column. PL/SQL determines the data type and size of the variable.

**Identifier table\_name.column\_name%TYPE;**  
**Identifier identifier%TYPE;**

```

set SERVEROUTPUT ON
DECLARE
    v_valid          BOOLEAN := True;
    v_id             employees.employee_id%TYPE default 100;
    v_last_name      VARCHAR2(25);
    v_salary         employees.salary%TYPE;
    v_new_salary     v_salary%TYPE;
BEGIN
    select last_name, salary into v_last_name, v_salary
    from employees where employee_id = v_id;
    IF v_valid THEN
        DBMS_OUTPUT.PUT_LINE(v_Last_name || ' ' || (v_salary+1));
    ELSE
        DBMS_OUTPUT.PUT_LINE('Test is FALSE');
    END IF;
END;
/

```

## 2-5 Writing PL/SQL Executable Statements

<b>Explicit conversion</b>	Converts values from one data type to another by using built-in functions.
<b>Implicit conversion</b>	Converts data types dynamically if they are mixed in a statement.

Character Functions:			Number Functions:			Date Functions:	
ASCII	LENGTH	RPAD	ABS	EXP	ROUND	ADD_MONTHS	MONTHS_BETWEEN
CHR	LOWER	RTRIM	ACOS	LN	SIGN	CURRENT_DATE	ROUND
CONCAT	LPAD	SUBSTR	ASIN	LOG	SIN	CURRENT_TIMESTAMP	SYSDATE
INITCAP	LTRIM	TRIM	ATAN	MOD	TAN	LAST_DAY	TRUNC
INSTR	REPLACE	UPPER	COS	POWER	TRUNC		

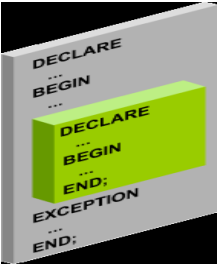
Implicit Conversions (It's not recommended)						Explicit Conversions	
	DATE	LONG	NUMBER	PLS_INTEGER	VARCHAR2	TO_NUMBER()	ROWIDTONCHAR()
DATE	N/A	X			X	TO_CHAR()	HEXTORAW()
LONG		N/A			X	TO_CLOB()	RAWTOHEX()
NUMBER		X	N/A	X	X	CHARTOROWID()	RAWTONHEX()
PLS_INTEGER		X	X	N/A	X	ROWIDTOCHAR()	TO_DATE()
VARCHAR2	X	X	X	X	N/A		

Operator	Operation
**	Exponentiation
+, -	Identity, negation
*, /	Multiplication, division
+, -,	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

<p>Statements can continue over several lines:</p> <pre>v_quote := 'The only thing that we can know is that we know            nothing and that is the highest flight of human reason.';</pre> <p>Numbers can be simple values or scientific notation: v_salary number := 2E4;</p> <pre>v_good_sal := v_sal BETWEEN 5000 AND 15000;</pre>	<pre>DECLARE   x VARCHAR2(20); BEGIN   x := '123' + '456' ;   DBMS_OUTPUT.PUT_LINE(x); END;</pre>
---	---

## 2-6 Nested Blocks and Variable Scope

<b>Block label</b>	A name given to a block of code which allows access to the variables that have scope, but are not visible.
<b>Variable scope</b>	Consists of all the blocks in which the variable is either local (the declaring block) or global (nested blocks within the declaring block) .
<b>Variable visibility</b>	The portion of the program where the variable can be accessed without using a qualifier.

 <p>&lt;&lt;outer&gt;&gt; optional Label with any name</p> <pre>DECLARE   v_outer_var VARCHAR2(20):='GLOBAL'; BEGIN   DECLARE     v_inner_var VARCHAR2(20):='LOCAL';   BEGIN     DBMS_OUTPUT.PUT_LINE(v_inner_var);     DBMS_OUTPUT.PUT_LINE(v_outer_var);   END;   DBMS_OUTPUT.PUT_LINE(v_outer_var); END;</pre>	<pre>&lt;&lt;outer&gt;&gt; DECLARE   v_father_name VARCHAR2(20):='Patrick';   v_date_of_birth DATE:='20-Apr-1972'; BEGIN   DECLARE     v_child_name VARCHAR2(20):='Mike';     v_date_of_birth DATE:='12-Dec-2002';   BEGIN     DBMS_OUTPUT.PUT_LINE('Father''s Name: '    v_father_name);     DBMS_OUTPUT.PUT_LINE('Date of Birth: '    outer.v_date_of_birth);     DBMS_OUTPUT.PUT_LINE('Child''s Name: '    v_child_name);     DBMS_OUTPUT.PUT_LINE('Date of Birth: '    v_date_of_birth);   END;   DBMS_OUTPUT.PUT_LINE('');   DBMS_OUTPUT.PUT_LINE('Date of Birth: '    v_date_of_birth); END;</pre>
---	--

## 2-7 Good Programming Practices

Category	Case Convention	Examples
SQL keywords	Uppercase	SELECT, INSERT
PL/SQL keywords	Uppercase	DECLARE, BEGIN, IF
Data types	Uppercase	VARCHAR2, BOOLEAN
Identifiers (variables, etc.)	Lowercase	v_salary, emp_cursor, c_tax_rate, p_empno
Tables and columns	Lowercase	employees, dept_id, salary, hire_date

Variables starting with v\_  
indent each level of code  
use %TYPE

Constants starting with c\_

Parameters starting with p\_

## Section 3 – Using SQL in PL/SQL

### 3-1 Review of SQL DML

<b>DELETE</b>	Statement used to remove existing rows in a table.
<b>INSERT</b>	Statement used to add new rows to a table.
<b>MERGE</b>	Statement used to INSERT and/or UPDATE a target table, based on matching values in a source table. UPSERT
<b>UPDATE</b>	Statement used to modify existing rows in a table.
<b>DDL</b>	When you create, change, or delete an object in a database.
<b>DML</b>	When you change data in an object (for example, by inserting or deleting rows).

#### ■ Insert Explicit

```
INSERT INTO employees (employee_id, first_name, last_name, email, hire_date, job_id, salary)
VALUES (305, 'Kareem', 'Naser', 'naserk@oracle.com', SYSDATE, 'SA_REP', NULL);
```

#### ■ Insert Implicit

```
INSERT INTO employees VALUES (
305, 'Kareem', 'Naser', 'naserk@oracle.com', '111-222-3333', SYSDATE, 'SA_REP', 7000, NULL, NULL, NULL, NULL);
```

DELETE FROM employees WHERE department_id= 80;	If the WHERE clause is omitted, ALL rows will be deleted
--	--

UPDATE employees SET salary = 11000, commission_pct= .3 WHERE employee_id= 176;	If the WHERE clause is omitted, ALL rows will be modified.
---	--

1.- CREATE TABLE bonuses ( employee_id NUMBER(6,0) NOT NULL, bonus NUMBER(8,2) DEFAULT 0);	2.- INSERT INTO bonuses(employee_id) SELECT employee_id FROM employees WHERE salary < 10000;
3.- MERGE INTO bonuses b USING employees e ON (b.employee_id= e.employee_id) WHEN MATCHED THEN UPDATE SET b.bonus= e.salary * .05; WHEN not MATCHED THEN INSERT VALUES(e.employee_id, e.bonus);	

### 3-2 Retrieving Data in PL/SQL

You cannot use DDL and DCL directly in PL/SQL, except to use Dynamic SQL “Execute Immediate” statement.

Handle Style	Description
DDL	CREATE TABLE, ALTER TABLE, DROP TABLE
DCL	GRANT, REVOKE

<pre>SELECT select_list INTO {variable_name [, variable_name]...       record_name} FROM table WHERE condition;</pre>	<pre>set SERVEROUTPUT ON DECLARE     v_id employees.employee_id%TYPE:= 100;     r_emp employees%ROWTYPE; BEGIN     SELECT * INTO r_emp     FROM employees     WHERE employee_id = v_id;     if SQL%FOUND THEN         DBMS_OUTPUT.PUT_LINE(r_emp.last_name  ' '  r_emp.salary);     End if; END;</pre>
---	--

### 3-3 Manipulating Data in PL/SQL

<b>Implicit cursors</b>	Defined automatically by Oracle for all SQL data manipulation statements, and for queries that return only one row. An implicit cursor is always automatically named “SQL”
<b>Explicit cursors</b>	Defined by the programmer for queries that return more than one row.
<b>MERGE</b>	Statement <u>selects</u> rows from one table to update and/or insert into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause.
<b>INSERT</b>	Statement adds new rows to the table.
<b>DELETE</b>	Statement removes rows from the table.
<b>UPDATE</b>	Statement modifies existing rows in the table.

#### Cursor Attributes for Implicit Cursors

Attribute	Description
SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement returned at least one row.
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement did not return even one row.
SQL%ROWCOUNT	An integer value that represents the number of rows affected by the most recent SQL statement.

```

set SERVEROUTPUT ON
DECLARE
    v_sal_increase employees.salary%TYPE:= 800;
BEGIN
    UPDATE copy_emp
    SET salary = salary + v_sal_increase
    WHERE job_id = 'ST_CLERK';
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
```

```

END;
```

### 3-4 Using Transaction Control Statements

<b>Transaction</b>	An inseparable list of database operations, which must be executed either in its entirety or not at all.
<b>ROLLBACK</b>	Used for discarding any changes that were made to the database after the last COMMIT.
<b>SAVEPOINT</b>	Used to mark an intermediate point in transaction processing.
<b>COMMIT</b>	Statement used to make database changes permanent.
<b>END</b>	Keyword used to signal the end of a PL/SQL block, not the end of a transaction.

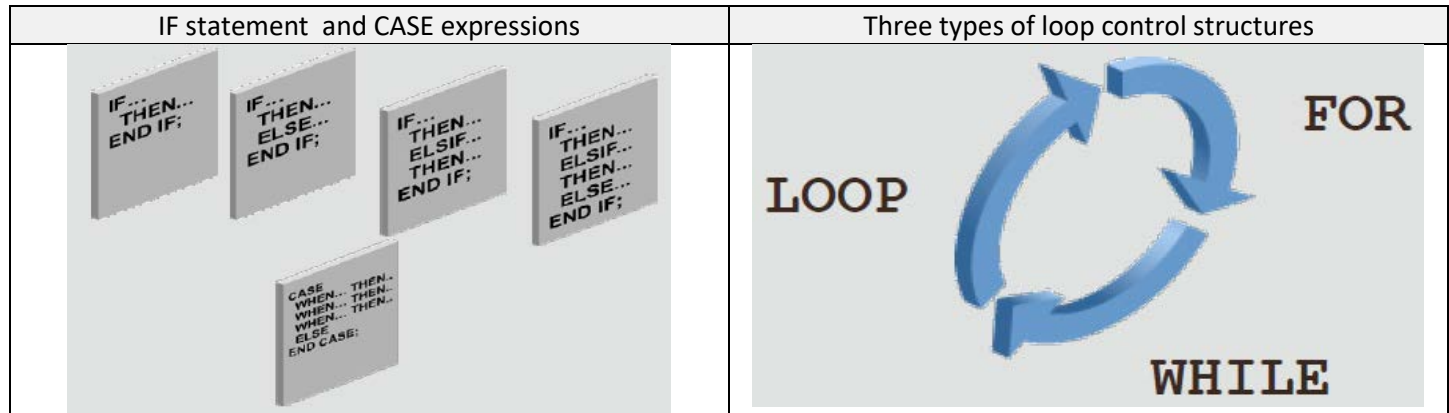
<pre> BEGIN     INSERT INTO pairtable VALUES (1, 2);     COMMIT; END;</pre>	<pre> BEGIN     INSERT INTO pairtable VALUES (7, 8);     SAVEPOINT my_sp_1;     INSERT INTO pairtable VALUES (9, 10);     SAVEPOINT my_sp_2;     INSERT INTO pairtable VALUES (11, 12);     ROLLBACK to my_sp_1;     INSERT INTO pairtable VALUES (13, 14);     COMMIT; END;</pre>
---	--



## Section 4 – Program Structures to Control Execution Flow

### 4-1 Conditional Control: IF Statements

<b>IF</b>	Statement that enables PL/SQL to perform actions selectively based on conditions.
<b>LOOP</b>	Control structures – Repetition statements that enable you to execute statements in a PL/SQL block repeatedly.
<b>Condition</b>	An expression with a TRUE or FALSE value that is used to make a decision.
<b>CASE</b>	An expression that determines a course of action based on conditions and can be used outside a PL/SQL block in a SQL statement.



<pre>IF condition THEN   statements; [ELSIF condition THEN   statements;] [ELSIF condition THEN   statements;] [ELSE   statements;] END IF;</pre>	<pre>set SERVEROUTPUT ON DECLARE   v_myage NUMBER := 10; BEGIN   IF v_myage &gt; 0 AND v_myage&lt; 11 THEN     DBMS_OUTPUT.PUT_LINE('I am a child');   ELSIF v_myage&lt; 20 THEN     DBMS_OUTPUT.PUT_LINE('I am young');   ELSIF v_myage&lt; 30 THEN     DBMS_OUTPUT.PUT_LINE('I am in my twenties');   ELSIF v_myage&lt; 40 THEN     DBMS_OUTPUT.PUT_LINE('I am in my thirties');   ELSE     DBMS_OUTPUT.PUT_LINE('I am mature');   END IF; END;</pre>
---	---

## 4-2 Conditional Control: Case Statements

<b>Logic Tables</b>	Shows the results of all possible combinations of two conditions.
<b>CASE statement</b>	A block of code that performs actions based on conditional tests.
<b>CASE expression</b>	An expression that selects a result and returns it into a variable.

Logic Tables											
AND				OR				NOT			
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	FALSE	NULL	TRUE	FALSE		
TRUE	TRUE	Ex. FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE		
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE		
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL		

CASE Statements	
<pre> DECLARE   v_age NUMBER := 10;   v_txt varchar2(50); BEGIN   CASE v_age     WHEN 0 THEN v_txt := 'unborn';     WHEN 10 THEN v_txt := 'teenager';     ELSE v_txt := 'I do not know';   END CASE;   DBMS_OUTPUT.PUT_LINE(V_TXT); END;</pre>	<pre> DECLARE   v_age NUMBER := 10;   v_txt varchar2(50); BEGIN   CASE     WHEN v_age &lt; 11 THEN v_txt := 'child';     WHEN v_age &lt; 20 THEN v_txt := 'young';     WHEN v_age &lt; 30 THEN v_txt := 'twenties';     ELSE v_txt := 'I am mature';   END CASE;   DBMS_OUTPUT.PUT_LINE(V_TXT); END;</pre>

CASE Expression Syntax	
<pre> variable_name:= CASE selector   WHEN expression1 THEN result1   WHEN expression2 THEN result2   ...   WHEN expressionN THEN resultN   [ELSE resultN+1] END;</pre>	<pre> variable_name:= CASE   WHEN search_condition1 THEN result1   WHEN search_condition2 THEN result2   ...   WHEN search_conditionN THEN resultN   [ELSE resultN+1] END;</pre>
<pre> DECLARE   v_grade CHAR(1) := 'A';   v_appraisal VARCHAR2(20); BEGIN   v_appraisal:=   CASE v_grade     WHEN 'A' THEN 'Excellent'     WHEN 'B' THEN 'Very Good'     ELSE 'No such grade'   END;   DBMS_OUTPUT.PUT_LINE(v_appraisal); END;</pre>	<pre> DECLARE   v_grade CHAR(1) := 'A';   v_appraisal VARCHAR2(20); BEGIN   v_appraisal :=   CASE     WHEN v_grade = 'A' THEN 'Excellent'     WHEN v_grade IN ('B','C') THEN 'Good'     ELSE 'No such grade'   END;   DBMS_OUTPUT.PUT_LINE (v_appraisal); END;</pre>

## 4-3 Iterative Control: Basic Loops

<b>Basic Loop</b>	Encloses a sequence of statements between the keywords LOOP and END LOOP and must execute at least once.
<b>EXIT</b>	Statement to terminate a loop.

Without the EXIT statement, the loop would never end (an infinite loop)		
<pre> BEGIN   LOOP     statements;     EXIT [WHEN condition];   END LOOP; END;</pre>	<pre> DECLARE   v_counter NUMBER(2) := 1; BEGIN   LOOP     DBMS_OUTPUT.PUT_LINE(v_counter);     v_counter := v_counter + 1;     EXIT WHEN v_counter &gt; 5;   END LOOP; END;</pre>	<pre> DECLARE   v_counter NUMBER := 1; BEGIN   LOOP     DBMS_OUTPUT.PUT_LINE(v_counter);     v_counter := v_counter + 1;     IF v_counter &gt; 5 THEN EXIT;   END IF;   END LOOP; END;</pre>

## 4-4 Iterative Control: While and For Loops

<b>WHILE Loop</b>	Repeats a sequence of statements until the controlling condition is no longer TRUE.
<b>FOR Loop</b>	Repeats a sequence of statements until a set number of iterations have been completed.

<pre> WHILE condition LOOP   statement1;   statement2;   ... END LOOP;</pre>	<pre> FOR counter IN [REVERSE] lower..upper LOOP   statement1;   statement2;   ... END LOOP;</pre>
<pre> DECLARE   v_counter NUMBER(2) := 1; BEGIN   WHILE v_counter &lt; 5 LOOP     DBMS_OUTPUT.PUT_LINE(v_counter);     v_counter := v_counter + 1;   END LOOP; END;</pre>	<pre> DECLARE   v_limit NUMBER(2) := 5; BEGIN   FOR i IN 1..v_limit LOOP     DBMS_OUTPUT.PUT_LINE(i);   END LOOP; END;</pre>

#### 4-5 Iterative Control: Nested Loops

```
Declare
  r varchar(50);
BEGIN
  FOR i IN 1..3 LOOP
    FOR j IN REVERSE 1..5 LOOP
      r := i || ' X ' || j || ' = ' || i*j;
      DBMS_OUTPUT.PUT_LINE(r);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP;
END;
```

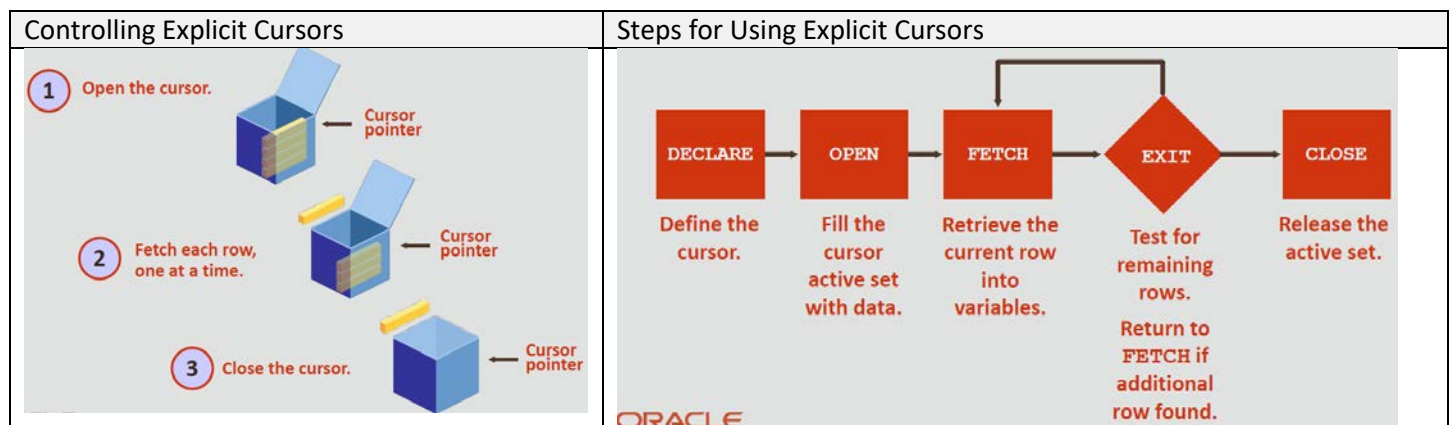
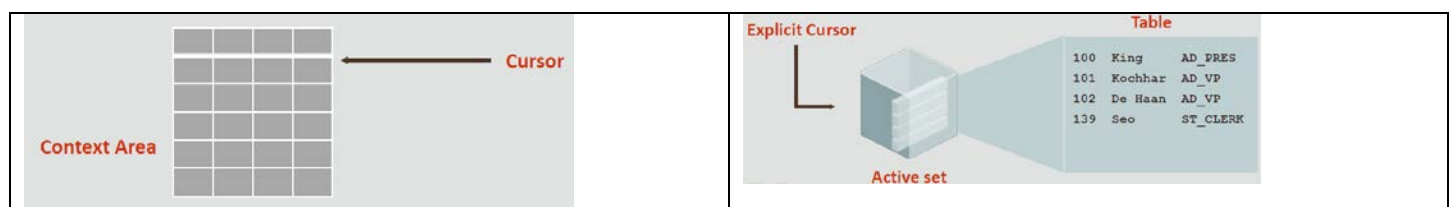
```
DECLARE
  i PLS_INTEGER:= 0;
  j PLS_INTEGER:= 5;
  v_r varchar2(50);
BEGIN
  <<outer_loop>>    -- Label
  LOOP
    i := i + 1;
    j := 5;
    EXIT WHEN i> 3;
    <<inner_loop>>  -- Label
    LOOP
      v_r := i || ' X ' || j || ' = ' || i*j;
      DBMS_OUTPUT.PUT_LINE(v_r);
      j := j-1;
      EXIT WHEN j = 0;
    END LOOP inner_loop;
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP -- outer_loop;
END;
```

Semester1, MIDTERM

## Section 5 – Using Cursors and Parameters

### 5-1 Introduction to Explicit Cursors

<b>Implicit Cursor</b>	Defined automatically by Oracle for all SQL DML statements, and for SELECT statements that return only one row
<b>Explicit Cursor</b>	Declared by the programmer for queries that return more than one row
<b>Cursor</b>	A label for a context area or a pointer to the context area
<b>Context Area</b>	An allocated memory area used to store the data processed by a SQL statement
<b>Active set</b>	The set of rows returned by a multiple row query in an explicit cursor operation
<b>OPEN</b>	Statement that executes the query associated with the cursor, identifies the active set, and positions the cursor pointer to the first row
<b>FETCH</b>	Statement that retrieves the current row and advances the cursor to the next row either until there are no more rows or until a specified condition is met
<b>CLOSE</b>	Disables a cursor, releases the context area, and undefines the active set



```
DECLARE
CURSOR cur_depts IS
  SELECT department_id, department_name
  FROM departments;
BEGIN
  FOR c_dep in cur_depts LOOP
    DBMS_OUTPUT.PUT_LINE(c_dep.department_id || ' ' || c_dep.department_name);
  END LOOP;
END;
```

```

DECLARE
  CURSOR cur_depts IS
    SELECT department_id, department_name FROM departments;
  v_department_id departments.department_id%TYPE;
  v_department_name departments.department_name%TYPE;
BEGIN
  OPEN cur_depts;
  LOOP
    FETCH cur_depts INTO v_department_id, v_department_name;    -- 2 variables
    EXIT WHEN cur_depts%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_department_id||' '||v_department_name);
  END LOOP;
  CLOSE cur_depts;
END;

```

```

DECLARE
  CURSOR cur_depts_emps IS
    SELECT department_name, COUNT(*) AS how_many
    FROM departments d, employees e
    WHERE d.department_id = e.department_id
    GROUP BY d.department_name
    HAVING COUNT(*) > 1;
...

```

## 5-2 Using Explicit Cursor Attributes

Attribute	Type	Description
<b>Record</b>		A composite data type in PL/SQL, consisting of a number of fields each with their own name and data type
<b>%ROWTYPE</b>		Declares a record with the same fields as the cursor on which it is based
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open.
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch did not return a row.
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returned a row; opposite of %NOTFOUND.
%ROWCOUNT	Number	Evaluates to the total number of rows FETCHed so far.

```

set SERVEROUTPUT ON
DECLARE
  CURSOR cur_emps IS
    SELECT * FROM employees
    WHERE department_id= 60;
  r_emp cur_emps%ROWTYPE;
BEGIN
  OPEN cur_emps;
  LOOP
    FETCH cur_emps INTO r_emp;
    EXIT WHEN cur_emps%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(
      r_emp.employee_id||' - '||
      r_emp.last_name);
  END LOOP;
  CLOSE cur_emps;
END;

```

```

DECLARE
  CURSOR cur_emps_dept IS
    SELECT last_name, department_name
    FROM employees e, departments d
    WHERE e.department_id=d.department_id;
  r_emp_dep cur_emps_dept%ROWTYPE;
BEGIN
  OPEN cur_emps_dept;
  LOOP
    FETCH cur_emps_dept INTO r_emp_dep;
    EXIT WHEN cur_emps_dept%NOTFOUND OR
      cur_emps_dept%ROWCOUNT> 10;
    DBMS_OUTPUT.PUT_LINE(
      r_emp_dep.last_name||' - '||
      r_emp_dep.department_name);
  END LOOP;
  CLOSE cur_emps_dept;
END;

```

```

IF NOT cur_emps%ISOPEN THEN
    OPEN cur_emps;
END IF;
LOOP
    FETCH cur_emps...

```

### 5-3 Cursor FOR Loops

<b>Cursor FOR loop</b>	Automates standard cursor-handling operations such as OPEN, FETCH, %NOTFOUND, and CLOSE so that they do not need to be coded explicitly
------------------------	---

<pre> FOR record_name IN cursor_name LOOP     statement1;     statement2;     . . . END LOOP;  ■ Without declare the cursor BEGIN     FOR r_emp IN (SELECT * FROM employees                   WHERE department_id= 50)     LOOP         DBMS_OUTPUT.PUT_LINE(r_emp.last_name);     END LOOP; END; </pre>	<pre> DECLARE CURSOR cur_dep IS     SELECT department_id, department_name     FROM departments     ORDER BY department_id; BEGIN     FOR r_dep IN cur_dep LOOP         EXIT WHEN cur_dep%ROWCOUNT &gt; 5;         DBMS_OUTPUT.PUT_LINE(r_dep.department_id                                  ' '    r_dep.department_name);     END LOOP; END; </pre>
--	--

### 5-4 Cursors with Parameters

<pre> CURSOR cursor_name [(parameter_name datatype, ...)] IS     select_statement; </pre>	<pre> OPEN cursor_name(parameter_value1,                   parameter_value2, ...);  FOR r_emp IN cur_emp(60) LOOP </pre>
---	--

<pre> DECLARE CURSOR cursor_employees( p_dep number) IS     SELECT * FROM employees     where department_id = p_dep; v_contador number := 0; BEGIN     dbms_output.put_line('No. LastName');     FOR r_emp in cursor_employees(90) LOOP         v_contador := v_contador +1 ;         dbms_output.put_line(v_contador    ' '                                  r_emp.last_name);     END LOOP; END; </pre>	<pre> DECLARE CURSOR cur_emp(p_dep integer) IS     SELECT * FROM employees     WHERE department_id= p_dep; r_emp cur_emp%ROWTYPE; BEGIN     OPEN cur_emp(90);     LOOP         FETCH cur_emp INTO r_emp;         EXIT WHEN cur_emp%NOTFOUND;         DBMS_OUTPUT.PUT_LINE(             r_emp.employee_id   ' - '                r_emp.last_name);     END LOOP;     CLOSE cur_emp; END; </pre>
---	--

<pre> CREATE OR REPLACE PROCEDURE pr_Emp(p_dep NUMBER) IS CURSOR cursor_employees(<b>p_dep number</b>) IS   SELECT * FROM employees   where department_id = p_dep; v_contador number := 0; BEGIN   dbms_output.put_line('No. LastName');   FOR r_emp in cursor_employees(<b>p_dep</b>) LOOP     v_contador := v_contador +1 ;     dbms_output.put_line(v_contador    ' '                            r_emp.last_name);   END LOOP; END; / call pr_Emp(90); </pre>	<pre> CREATE PROCEDURE pr_Emp2(p_dep NUMBER) IS CURSOR cursor_employees IS   SELECT * FROM employees   where department_id = p_dep; v_contador number := 0; BEGIN   dbms_output.put_line('No. LastName');   FOR r_emp in cursor_employees LOOP     v_contador := v_contador +1 ;     dbms_output.put_line(v_contador                            ' '    r_emp.last_name);   END LOOP; END; / call pr_Emp2(90);      -- little aesthetic. </pre>
--	--

## 5-5 Using Cursors For Update

<b>FOR UPDATE</b>	Declares that each row is locked as it is being fetched so other users cannot modify the rows while the cursor is open
<b>NOWAIT</b>	A keyword used to tell the Oracle server not to wait if the requested rows have already been locked by another user

```

CURSOR cursor_name IS
SELECT... FROM...
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];

```

n = number of seconds to wait and check whether the rows are unlocked.

If the cursor is based on a join of two tables, we may want to lock the rows of one table but not the other  
To do this, we specify **any column** of the table we want to lock.

It also allows us to modify the rows ourselves using a ... WHERE CURRENT OF *cursor-name*

```

DECLARE
CURSOR cur_eds IS
  SELECT employee_id, salary, department_name
  FROM my_employees e, my_departments d
  WHERE e.department_id = d.department_id
  FOR UPDATE OF salary NOWAIT;
BEGIN
  FOR r_eds IN cur_eds LOOP
    UPDATE my_employees
    SET salary = r_eds.salary * 1.1
    WHERE CURRENT OF cur_eds;
  END LOOP;
END;

```



## 5-6 Using Multiple Cursors

Explain the need for using multiple cursors to produce multi-level reports

```
set SERVEROUTPUT ON
DECLARE
  CURSOR cur_dep IS
    SELECT * FROM departments;
  CURSOR cur_emp (p_dep NUMBER) IS
    SELECT * FROM employees WHERE department_id = p_dep;
BEGIN
  FOR r_dep IN cur_dep LOOP
    DBMS_OUTPUT.PUT_LINE(upper(r_dep.department_name));
    FOR r_emp IN cur_emp (r_dep.department_id) LOOP
      DBMS_OUTPUT.PUT_LINE(r_emp.last_name);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(' ');
  END LOOP;
END;
```

## Section 6 – Using Composite Datatypes

### 6-1 User-Defined Records

<b>PL/SQL record</b>	a composite data type consisting of a group of related data items stored as fields, each with its own name and data type
----------------------	--

```
TYPE type_name IS RECORD
  (field_declaration[,field_declaration]..);

identifier type_name;
```

<pre>set SERVEROUTPUT ON DECLARE   r_emp      employees%ROWTYPE;   r_emp_c    r_emp%ROWTYPE; BEGIN   SELECT * INTO r_emp   FROM employees   WHERE employee_id = 100;   r_emp_c := r_emp;   r_emp_c.salary:=r_emp.salary* 1.2;   DBMS_OUTPUT.PUT_LINE(r_emp.last_name       ':Old Salary='  r_emp.salary       ' New Salary='  r_emp_c.salary); END; /</pre>	<pre>set SERVEROUTPUT ON DECLARE   TYPE person_type IS RECORD(     employee_id employees.employee_id%type,     last_name   employees.last_name%TYPE,     dep_id      departments.department_id%type,     dep_name    departments.department_name%TYPE);   r_per person_type; Begin   SELECT e.employee_id,   e.last_name,          d.department_id, d.department_name          INTO r_per   FROM employees e JOIN departments d   ON e.department_id = d.department_id   WHERE employee_id = 200;   DBMS_OUTPUT.PUT_LINE(r_per.employee_id  ' '        r_per.last_name    ' is in the '        r_per.dep_name      ' department.');</pre>
---	---

#### Visibility and Scope of Types and Records

What will be displayed by each of the PUT\_LINES?

<pre>DECLARE -- outer block   TYPE employee_type IS RECORD(     first_name employees.first_name%TYPE:= 'Amy');   r_emp_outer employee_type; BEGIN   DBMS_OUTPUT.PUT_LINE(r_emp_outer.first_name);   DECLARE -- inner block     r_emp_inner employee_type;   BEGIN     r_emp_outer.first_name:= 'Clara';     DBMS_OUTPUT.PUT_LINE(r_emp_outer.first_name         ' and '    r_emp_inner.first_name);   END;   DBMS_OUTPUT.PUT_LINE(r_emp_outer.first_name); END;</pre>
---

## 6-2 Indexing Tables of Records

Save in memory

<b>Collection</b>	A set of occurrences of the same kind of data
<b>INDEX BY TABLE</b>	A collection which is based on a <b>single field</b> or column; for example, on the last_name column of EMPLOYEES
<b>INDEX BY TABLE OF RECORDS</b>	A collection which is based on a composite <b>record</b> type; for example, on the whole DEPARTMENTS row

Populating an INDEX BY Table	methods
<pre>DECLARE   TYPE type_name IS TABLE OF DATA_TYPE     INDEX BY PRIMARY_KEY_DATA_TYPE;   identifier type_name; BEGIN   FOR record IN (SELECT column FROM table)   LOOP     identifier(primary_key) := record.column;   END LOOP; END;</pre>	<pre>EXISTS  PRIOR COUNT  NEXT FIRST   DELETE LAST    TRIM</pre>

<pre>DECLARE   TYPE t_Last_name IS TABLE OF employees.Last_name%TYPE     INDEX BY BINARY_INTEGER;   v_Last_name_tab t_Last_name; BEGIN   FOR emp_rec IN (SELECT employee_id, Last_name FROM employees) LOOP     v_Last_name_tab(emp_rec.employee_id) := emp_rec.Last_name;   END LOOP;   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab.COUNT);      -- 20   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab.FIRST);      -- 100   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab.next(107));  -- 124   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab.prior(124)); -- 107   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab.last);       -- 206   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab(100));       -- King   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab(206));       -- Gientz   v_Last_name_tab.DELETE(201);   DBMS_OUTPUT.PUT_LINE(v_Last_name_tab.next(200));  -- 202   IF v_Last_name_tab.EXISTS(200) then     DBMS_OUTPUT.PUT_LINE(v_Last_name_tab(200));     -- Whalen   end if; END;</pre>
---

## INDEX BY Table of Records

```
DECLARE
  TYPE t_emp IS TABLE OF employees%ROWTYPE
    INDEX BY BINARY_INTEGER;
  v_count BINARY_INTEGER := 0;
  i BINARY_INTEGER := 0;
  v_emp_tab t_emp;
BEGIN
  FOR r_emp IN (SELECT * FROM employees order by employee_id) LOOP
    v_count := v_count + 1;
    v_emp_tab(v_count) := r_emp;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(v_emp_tab.COUNT);
  DBMS_OUTPUT.PUT_LINE(v_emp_tab(1).Last_name);
  i := v_emp_tab.Last;
  DBMS_OUTPUT.PUT_LINE(v_emp_tab(i).Last_name);
  DBMS_OUTPUT.PUT_LINE(v_emp_tab(i).salary);
END;
```

## Section 7 – Exception Handling

### 7-1 Handling Exceptions

<b>Exception Handler</b>	Code that defines the recovery actions to be performed when execution-time errors occur.
<b>Exception</b>	Occurs when an error is discovered during the execution of a program that disrupts the normal operation of the program.

```
DECLARE
    v_first_name VARCHAR2(25);
    v_last_name  VARCHAR2(25);
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM employees
    WHERE last_name = 'King ';
    DBMS_OUTPUT.PUT_LINE ('The employee of the month is: ' ||
                           v_first_name || ' ' || v_last_name || '.');
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE ('You select multiple rows.');
```

### 7-2 Trapping Oracle Server Exceptions

<b>Predefined Oracle Server Errors</b>	Each of these has a predefined name. For example, if the error ORA-01403 occurs when no rows are retrieved from the database in a SELECT statement, then PL/SQL raises the predefined exception-name NO_DATA_FOUND.
<b>Non-predefined Oracle Server Errors</b>	Each of these has a standard Oracle error number (ORA-nnnnn) and error message, but not a predefined name. We declare our own names for these so that we can reference these names in the exception section.
<b>PRAGMA EXCEPTION_INIT</b>	Tells the compiler to associate an exception name with an Oracle error number. That allows you to refer to any Oracle Server exception by name and to write a specific handler for it.
<b>SQLCODE</b>	Returns the numeric value for the error code (You can assign it to a NUMBER variable.)
<b>SQLERRM</b>	Returns character data containing the message associated with the error number

SQLCODE Value	Description
0	No exception encountered
1	User defined exception
+100	NO_DATA_FOUND exception
Negative number	Another Oracle Server error number

```
begin
  INSERT INTO departments (department_id, department_name) VALUES (280, NULL);
end;
```

**Error report -**

**ORA-01400** : cannot insert NULL into (DEPARTMENTS.DEPARTMENT\_NAME)

\*Cause: An attempt was made to insert NULL into previously listed objects.

\*Action: These objects cannot accept NULL values.

### Functions for Trapping Exceptions

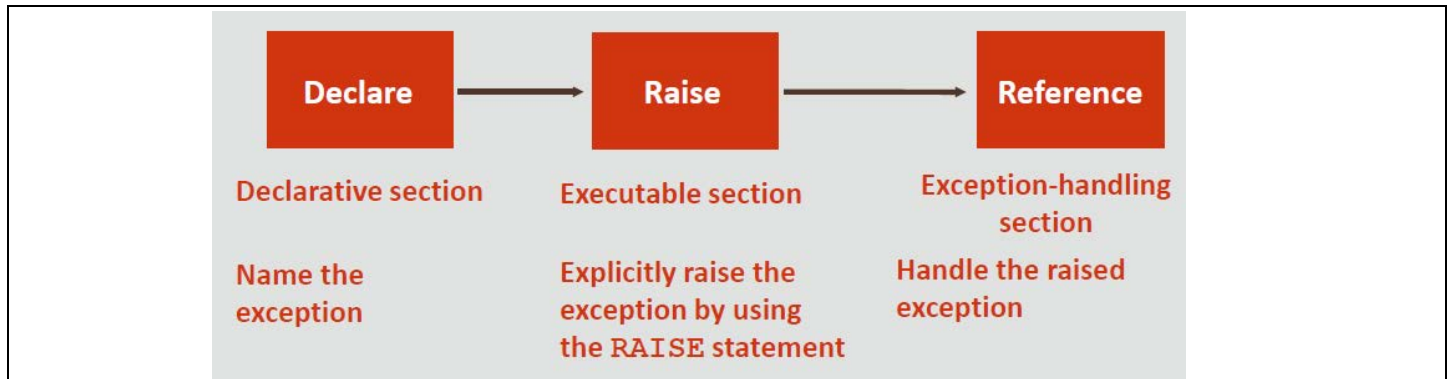
```
DECLARE
  v_error_code    NUMBER;
  v_error_message VARCHAR2(255);
BEGIN
  INSERT INTO departments (department_id, department_name) VALUES (280, NULL);
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code :=    SQLCODE;
    v_error_message:= SQLERRM;
    DBMS_OUTPUT.PUT_LINE ('Error No.    ' || v_error_code );
    DBMS_OUTPUT.PUT_LINE ('Descripcion ' || v_error_message );
END;
```

### Non-predefined Oracle Server Errors

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments (department_id, department_name) VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep THEN
    DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

### 7-3 Trapping User-Defined Exceptions

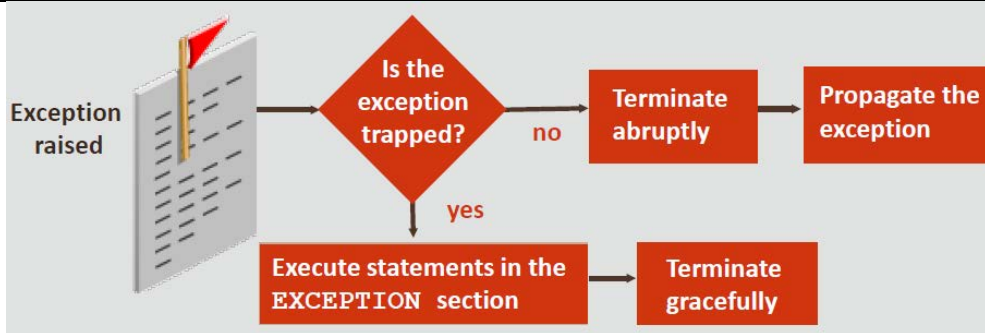
<b>RAISE_APPLICATION_ERROR</b>	A procedure used to return user-defined error messages from stored subprograms.
<b>RAISE</b>	Use this statement to raise a named exception.
<b>user-defined errors</b>	These errors are not automatically raised by the Oracle Server, but are defined by the



```
DECLARE
  v_last_name employees.last_name%TYPE := 'Silly Name';
  e_name EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_name, -20999); -- [-20000 .. -20999]
BEGIN
  DELETE FROM employees
  WHERE last_name= v_last_name;
  IF SQL%ROWCOUNT = 0 THEN
    RAISE e_name;
    RAISE_APPLICATION_ERROR(-20999, 'Invalid last name');
  ELSE
    DBMS_OUTPUT.PUT_LINE(v_last_name||' deleted');
  END IF;
EXCEPTION
  WHEN e_name THEN
    DBMS_OUTPUT.PUT_LINE('Valid last names are: ');
    FOR c1 IN (SELECT DISTINCT last_name FROM employees) LOOP
      DBMS_OUTPUT.PUT_LINE(c1.last_name);
    END LOOP;
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error deleting from employees');
END;
```

### 7-4 Recognizing the Scope of Exceptions

<b>Propagation</b>	The inner block terminates unsuccessfully, and PL/SQL passes the exception to the outer block.
<b>Exception Visibility</b>	The portion of the program where the exception can be accessed without using a qualifier.
<b>Exception scope</b>	The portion of a program in which the exception is declared and is accessible.



```

set SERVEROUTPUT ON
DECLARE
  v_last_name employees.last_name%TYPE;
BEGIN
  BEGIN
    SELECT last_name INTO v_last_name
    FROM employees WHERE job_id = 'AD_PRES'; -- AD_PRES  IT_PROG;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Message 4');
END;

```

```

set SERVEROUTPUT ON
DECLARE
  e_myexcep EXCEPTION;
BEGIN
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN e_myexcep THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;

```



## Section 8 – Using and Managing Procedures

### 8-1 Creating Procedures

<b>Subprograms</b>	Named PL/SQL blocks that are compiled and stored in the database.
<b>IS or AS</b>	Indicates the DECLARE section of a subprogram.
<b>Anonymous Blocks</b>	Unnamed executable PL/SQL blocks that cannot be reused or stored in the database for later use.
<b>Procedures</b>	Named PL/SQL blocks that can accept parameters and are compiled and stored in the database.

Anonymous Blocks	Subprograms
Unnamed PL/SQL blocks	Named PL/SQL blocks
Compiled on every execution	Compiled only once, when created
Not stored in the database	Stored in the database
Cannot be invoked by other applications	They are named and therefore can be invoked by other applications
Do not return values	Subprograms called functions must return values
Cannot take parameters	Can take parameters

Two types of subprograms:

Functions: Must return a single value using the RETURN statement

Procedures: Does not return values.

The keyword DECLARE is replaced by CREATE PROCEDURE procedure-name IS|AS

In anonymous blocks, DECLARE states, "this is the start of a block".

Functions can be called from SQL, procedures can not.

Functions are considered expressions, procedures are not.

Parameters are Mode defaults to IN

Procedures and functions can both return data as OUT and IN OUT parameters.

Shows up in USER\_OBJECTS as an object type of PROCEDURE

More details in USER\_PROCEEDURES

Detailed PL/SQL code in USER\_SOURCE

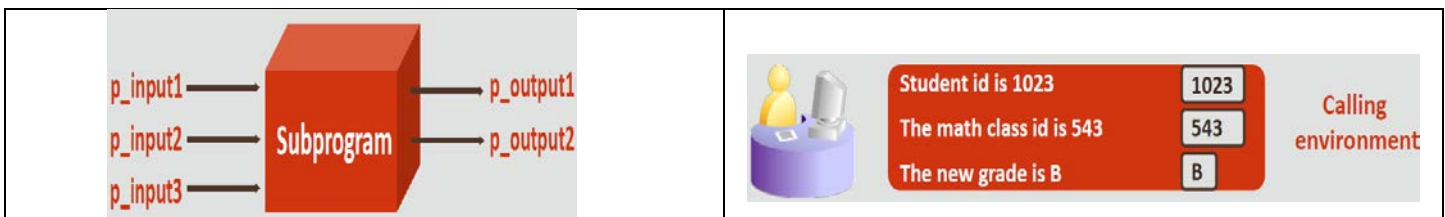
<ul style="list-style-type: none"><li>• <b>Anonymous blocks</b></li></ul> <pre>DECLARE (Optional)     Variables, cursors, etc.; BEGIN (Mandatory)     SQL and PL/SQL statements; EXCEPTION (Optional)     WHEN exception-handling actions; END; (Mandatory)</pre>	<pre>CREATE [OR REPLACE] PROCEDURE procedure_name     [(parameter1 [mode]datatype1,       parameter2 [mode]datatype2, ...)] IS   AS     [local_variable_declarations; ...] BEGIN     -- actions; [EXCEPTION] END [procedure_name];</pre>
---	--

When a subprogram is CREATED, the source code is stored in the database even if compilation errors occurred.

<pre> CREATE OR REPLACE PROCEDURE subproc ... END subproc;  CREATE OR REPLACE PROCEDURE mainproc ... IS BEGIN ...     subproc(...); ... END mainproc; </pre>	<pre> CREATE OR REPLACE PROCEDURE mainproc ... IS     PROCEDURE subproc (...) IS BEGIN         ...     END subproc; BEGIN     ...     subproc(...); ... END mainproc; </pre>
--	--

## 8-2 Using Parameters in Procedures

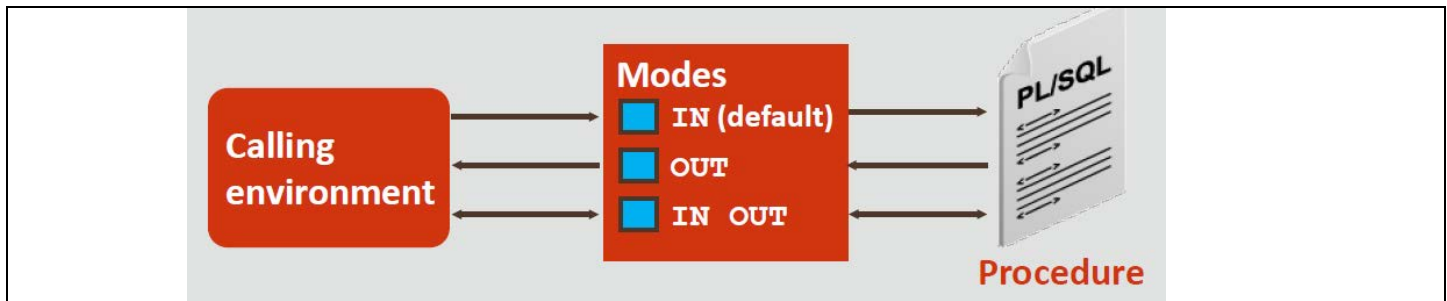
<b>Parameters</b>	Pass or communicate data between the caller and subprogram.
<b>Argument</b>	The actual value assigned to a parameter.
<b>Formal Parameters</b>	A parameter name declared in the procedure heading. Example: p_emp_id Notice that the formal parameter data types do <b>not have sizes</b> .
<b>Actual Parameters</b>	Can be literal values, variables, or expressions that are sent to the parameter list of a called subprogram. Example: raise_sal(v_emp_id, v_raise + 100);



<pre> CREATE OR REPLACE PROCEDURE change_grade( p_student_id IN NUMBER, p_class_id   IN NUMBER, p_grade      IN VARCHAR2) IS BEGIN     UPDATE grade_table SET grade = p_grade     WHERE student_id = p_student_id AND            class_id = p_class_id; END; </pre>	<pre> BEGIN change_grade(1023, 543, 'B'); END;  EXECUTE change_grade(1023, 543, 'B');  CALL change_grade(1023, 543, 'B'); </pre> <p>You must enter the arguments in the same order as they are declared in the procedure.</p>
---	---

## 8-3 Passing Parameters

<b>IN Parameter</b>	Provides values for a subprogram to process (default)
<b>OUT Parameter</b>	Returns a value to the caller
<b>IN OUT Parameter</b>	Supplies an input value, which may be returned as a modified value
<b>three ways of passing parameters</b>	
<b>Positional Notation</b>	Lists the actual parameters in the same order as the formal parameters
<b>Named Notation</b>	Lists the actual parameters in arbitrary order and uses the association operator ( '='>' which is an equal and an arrow together) to associate a named formal parameter with its actual parameter
<b>Combination Notation</b>	Lists some of the actual parameters as positional (no special operator) and some as named (with the => operator)



IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, constant, expression, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

<pre>CREATE OR REPLACE PROCEDURE query_emp(   p_id   IN   number default 100,   p_name OUT employees.last_name%TYPE,   p_sal  OUT employees.salary%TYPE) IS BEGIN   SELECT last_name, salary INTO          p_name,    p_sal   FROM employees   WHERE employee_id= p_id; END query_emp;</pre>	<pre>DECLARE   v_name  employees.last_name%TYPE;   v_sal   employees.salary%TYPE; BEGIN   -- query_emp(178, v_name, v_sal); -- positional   query_emp(p_id =&gt;178, p_sal=&gt;v_sal, p_name=&gt;v_name);   -- query_emp(178, p_sal =&gt;v_sal, p_name=&gt;v_name);   -- query_emp( p_sal=&gt;v_sal, p_name=&gt;v_name );   DBMS_OUTPUT.PUT_LINE('Name: '    v_name);   DBMS_OUTPUT.PUT_LINE('Salary: '   v_sal); END;</pre>
--	--

<pre>set SERVEROUTPUT ON CREATE OR REPLACE PROCEDURE format_phone (p_phone IN OUT VARCHAR2) IS BEGIN   p_phone:= '('    SUBSTR(p_phone, 1, 3)                ')'    SUBSTR(p_phone, 4, 3)                '-'    SUBSTR(p_phone, 7); END format_phone;</pre>	<pre>DECLARE   a_phone VARCHAR2(13); BEGIN   a_phone := '8006330575' ;   format_phone(a_phone);   DBMS_OUTPUT.PUT_LINE(a_phone); END;</pre> <p>(800)633-0575</p>
---	--

## Section 9 – Using and Managing Functions

### 9-1 Creating Functions

**Stored Functions** A named PL/SQL block that can accept optional IN parameters and must return a single output.

Procedures	Functions
Execute as a PL/SQL statement	Invoked as part of an expression
Do not contain RETURN clause in the header	Must contain a RETURN clause in the header
May return values (if any) in output parameters (not required)	Must return a single value
	Must contain at least one RETURN statement

Avoid the following within functions:

- Any kind of DML or DDL
- COMMIT or ROLLBACK
- Altering global variables

<pre>CREATE [OR REPLACE] FUNCTION function_name [(parameter1 [mode1] datatype1, ...)] RETURN datatype IS AS [local_variable_declarations; ...] BEGIN --actions; RETURN expression; [EXCEPTION] END [function_name];</pre>	<pre>CREATE OR REPLACE FUNCTION get_sal (p_id IN employees.employee_id%TYPE) RETURN NUMBER IS v_sal employees.salary%TYPE:= 0; BEGIN SELECT salary INTO v_sal FROM employees WHERE employee_id = p_id; RETURN v_sal; EXCEPTION WHEN NO_DATA_FOUND THEN RETURN NULL; END get_sal; /</pre>
---	--

<p>■ Invoking a Function as an Expression in a SQL Statement</p> <pre>SELECT get_sal(employee_id) from employees;</pre> <p>DECLARE v_sal employees.salary%type;</p> <p>BEGIN</p> <p>■ as Part of a PL/SQL Expression</p> <pre>v_sal:= get_sal(100);</pre> <p>■ as a Parameter in Another Subprogram</p> <pre>DBMS_OUTPUT.PUT_LINE(get_sal(100));</pre> <p>END;</p>
--

RETURN true or false	If existsDepto( ) then
<pre> CREATE OR REPLACE FUNCTION valid_dept   (p_dept_no departments.department_id%TYPE) RETURN BOOLEAN IS v_valid VARCHAR2(1); BEGIN   SELECT 'x' INTO v_valid     FROM departments     WHERE department_id = p_dept_no;   RETURN true; EXCEPTION   WHEN NO_DATA_FOUND THEN RETURN(false);   WHEN OTHERS THEN NULL; END;</pre>	<pre> BEGIN   IF valid_dept(v_departmentid) THEN     insert into employees values(...);   ELSE     DBMS_OUTPUT.PUT_LINE       ('Depto invalido');   END IF; END;</pre>

<pre> CREATE OR REPLACE FUNCTION esBisiesto   (p_ano IN NUMBER) RETURN boolean IS   b_yn boolean := false; BEGIN   IF MOD ( p_ano, 400 ) = 0 OR     MOD ( p_ano, 4 ) = 0 AND     MOD ( p_ano, 100 ) != 0 THEN     b_yn := true;   END IF;   RETURN b_yn; END;</pre>	<pre> ■ leap-year  declare   v_ano number := 2024; begin   if isbisiesto(v_ano) = 1 then     if isbisiesto(v_ano) then       DBMS_OUTPUT.PUT_LINE('Es bisiesto');     else       DBMS_OUTPUT.PUT_LINE('No bisiesto');     end if;   end if; end; / SELECT ISBISIESTO(2024) FROM DUAL; ORA-00902: invalid datatype</pre>
---	---

## 9-2 Using Functions in SQL Statements

**User Defined Function** A function created by the PL/SQL programmer that can be used anywhere there is a value or function.

If the SQL statement processes many rows in a table, the function executes once for each row processed by the SQL statement. Functions called from a SELECT statement cannot contain DML statements.

The SYS.STANDARD package has functions like UPPER and LOWER. Imagine the different ways you can spell the last name: "Taylor," "TAYLOR," "taylor," or "TAYlor."

<pre> SELECT * FROM employees WHERE UPPER(last_name) = UPPER('TAYlor');</pre>
---

<pre> CREATE OR REPLACE FUNCTION tax   (p_value IN NUMBER) RETURN NUMBER IS BEGIN   RETURN (p_value * 0.08); END tax;</pre>	<pre> SELECT employee_id, tax(salary) FROM employees WHERE tax(salary) &gt; (SELECT MAX(tax(salary))                      FROM employees                      WHERE department_id= 20) ORDER BY tax(salary) DESC;</pre>
---	---

9-3 Review of the Data Dictionary

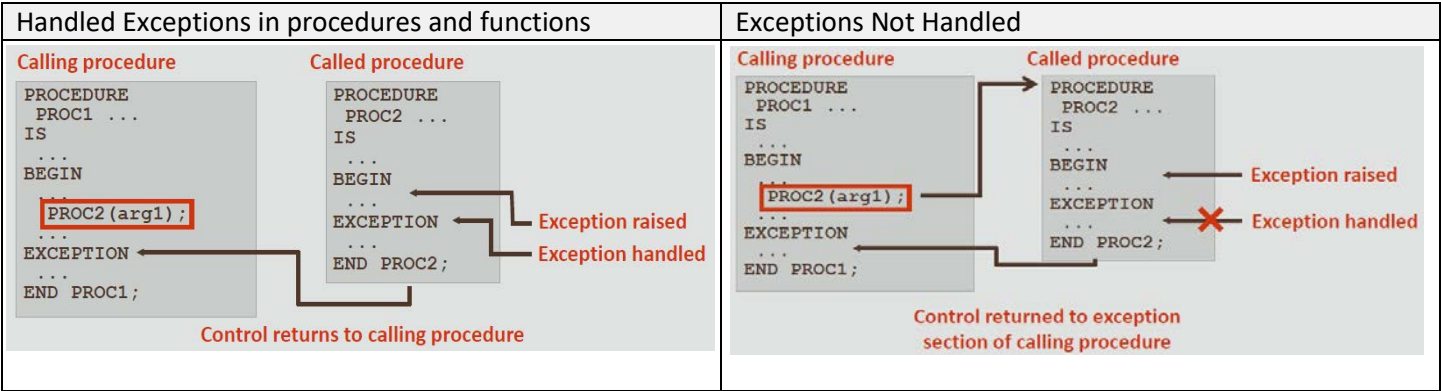
<b>Data Dictionary</b>	A catalog of all database objects contained in an Oracle database.
<b>three types of Data Dictionary:</b>	
<b>DBA_* tables</b>	Contain information about everything in the database, no matter who owns them.
<b>ALL_* tables</b>	Contain information about objects which you have privileges to use.
<b>USER_* tables</b>	Contain information about objects you own. User_tables, User_indexes, User_sequences, User_Objects

```
brief description of each table:
SELECT * FROM DICT[IONARY]
SELECT * FROM DICT WHERE table_name LIKE 'USER%IND%';

SELECT object_type, COUNT(*) FROM USER_OBJECTS
GROUP BY object_type;
```

9-4 Managing Procedures and Functions

<b>USER_OBJECTS</b>	The dictionary table that contains the names and types of procedures and functions that you own.
<b>USER_SOURCE</b>	The dictionary table that contains source code for all of the subprograms that you own.
<b>ALL_SOURCE</b>	The dictionary table that contains source code for subprograms that are owned by others who have granted you the EXECUTE privilege.



With and without Exception
<pre> CREATE OR REPLACE PROCEDURE add_department(     p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS BEGIN     INSERT INTO DEPARTMENTS (department_id,         department_name, manager_id, location_id)         VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);     DBMS_OUTPUT.PUT_LINE('Added Dept: '    p_name); EXCEPTION     WHEN OTHERS THEN         DBMS_OUTPUT.PUT_LINE('Error adding dept: '    p_name); END; </pre>
<pre> BEGIN     add_department('Media', 100, 1800);     add_department('Editing', 99, 1800);     add_department('Advertising', 101, 1800); END; </pre>

**DROP PROCEDURE** *procedure\_name*

**DROP FUNCTION** *function\_name*

<pre> SELECT object_name, OBJECT_TYPE FROM USER_OBJECTS WHERE object_type= 'FUNCTION'; </pre>	<pre> SELECT * FROM USER_SOURCE WHERE name = 'TAX' ORDER BY line; </pre>	<pre> SELECT * FROM ALL_SOURCE WHERE OWNER = 'ESQUEMAS' AND NAME = 'TAX'; </pre>
---	--	--

## 9-5 Review of Object Privileges

<b>EXECUTE privilege</b>	Allows the grantee to invoke and execute a PL/SQL subprogram. owner->creator Grantee ->beneficiario, concesionario granted -> concedido
<b>INDEX privilege</b>	Allows the grantee to create indexes on the table.
<b>Object privilege</b>	Allows the use of a specific database object, such as a table, a view, or a PL/SQL procedure, by one or more database users.
<b>ALTER privilege</b>	Allows the grantee to ALTER the table.
<b>REFERENCES privilege</b>	Allows the grantee to check for the existence of rows in a table using foreign key constraints.

Object Privilege	Table	View	Sequence	Procedure
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X			
INSERT	X	X		
REFERENCES	X	X		
SELECT	X	X	X	
UPDATE	X	X		

<b>GRANT</b> <i>object_priv</i> [( <i>columns</i> )] <b>ON</b> <i>object</i> <b>TO</b> { <i>user</i>   <i>role</i>   <b>PUBLIC</b> } [ <b>WITH GRANT OPTION</b> ];	<b>REVOKE</b> <i>object_priv</i> [( <i>columns</i> )] <b>ON</b> <i>object</i> <b>FROM</b> { <i>user</i>   <i>role</i>   <b>PUBLIC</b> };
---	--

Syntax	Definition
object_priv	The privilege to be granted.
columns	Limits UPDATE privilege to a specific column (optional).
object	The object name on which the privilege(s) are granted.
User   role	Identifies the user/roles to whom the privilege is granted.
PUBLIC	Grants the named privilege(s) to all users.
WITH GRANT OPTION	Allows grantee to grant object privileges to others.

GRANT INSERT, UPDATE ON employees <b>TO</b> HR, SUSAN;	REVOKE INSERT, UPDATE ON employees <b>FROM</b> HR, SUSAN;
GRANT SELECT ON departments <b>TO</b> PUBLIC;	REVOKE SELECT ON departments <b>FROM</b> PUBLIC;
GRANT EXECUTE ON add_dept <b>to</b> SUSAN;	REVOKE EXECUTE ON add_dept <b>FROM</b> SUSAN;

<p>■ SUSAN does not need GRANT INSERT on departments</p> <pre>CREATE OR REPLACE PROCEDURE add_dept ... IS BEGIN ... INSERT INTO DEPARTMENTS ... ; ... END; GRANT EXECUTE ON add_dept TO SUSAN;</pre>
--

<p>BILL owns the STUDENTS and GRADES tables. HANNAH needs to create a procedure that JIEP needs to invoke:</p> <p>(Hannah)</p> <pre>CREATE OR REPLACE PROCEDURE student_proc... IS BEGIN   SELECT ...FROM bill.students JOIN bill.grades...;   UPDATE bill.students...; ... END;</pre> <p>(Jiep)</p> <pre>BEGIN hannah.student_proc(...); END;</pre> <p>■ Who needs which privileges on which objects?</p>
--



## 9-6 Using Invoker's Rights and Autonomous Transactions

S e m e s t e r 1 , F I N A L