PL/SQL

**Copyright © 2006,
Digitech Technologies**

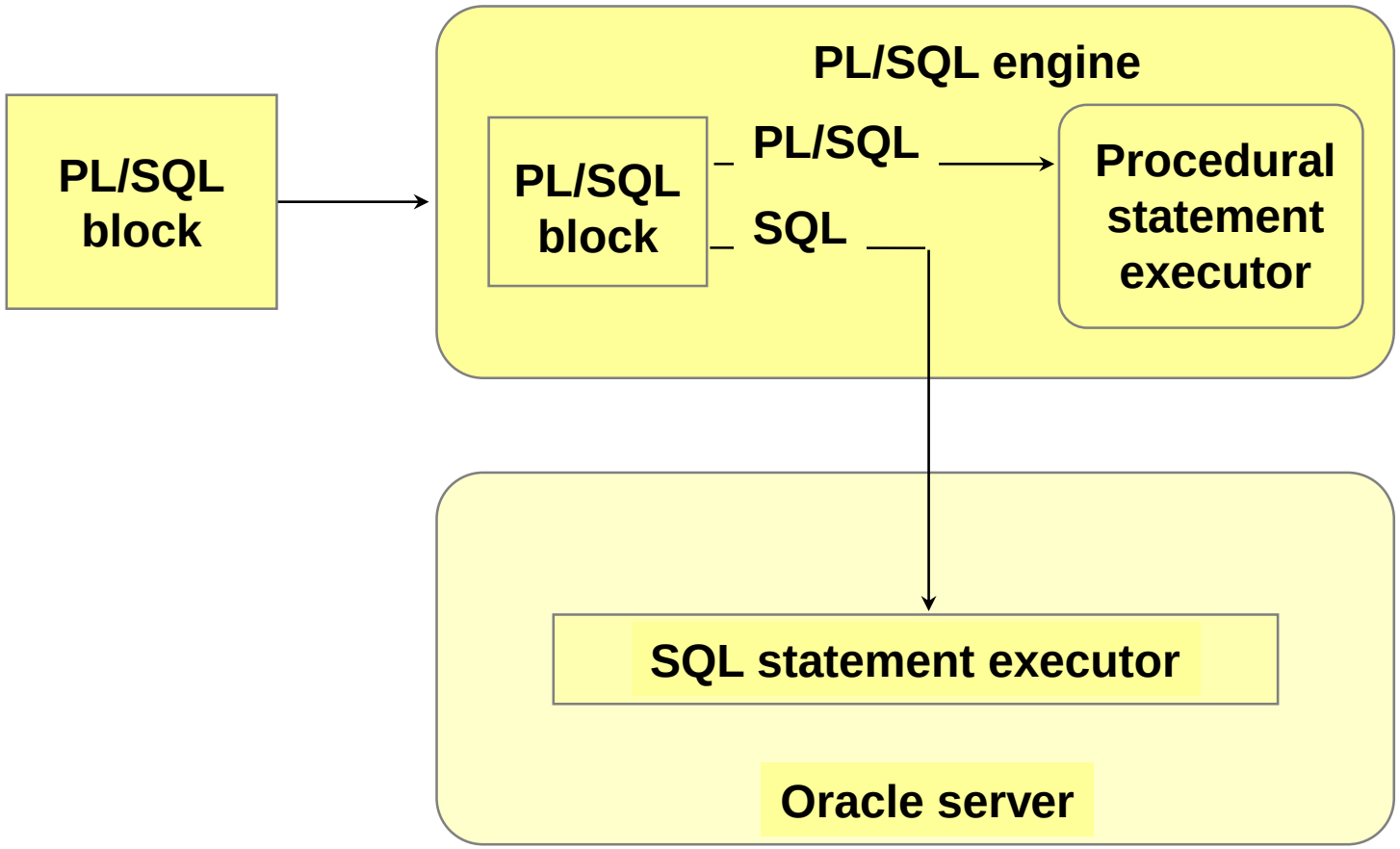
PL/SQL

PL/SQL is the procedural extension to SQL with design features of programming languages.

Data manipulation and query statements of SQL are included within procedural units of code.

**Copyright © 2006,
Digitech Technologies**

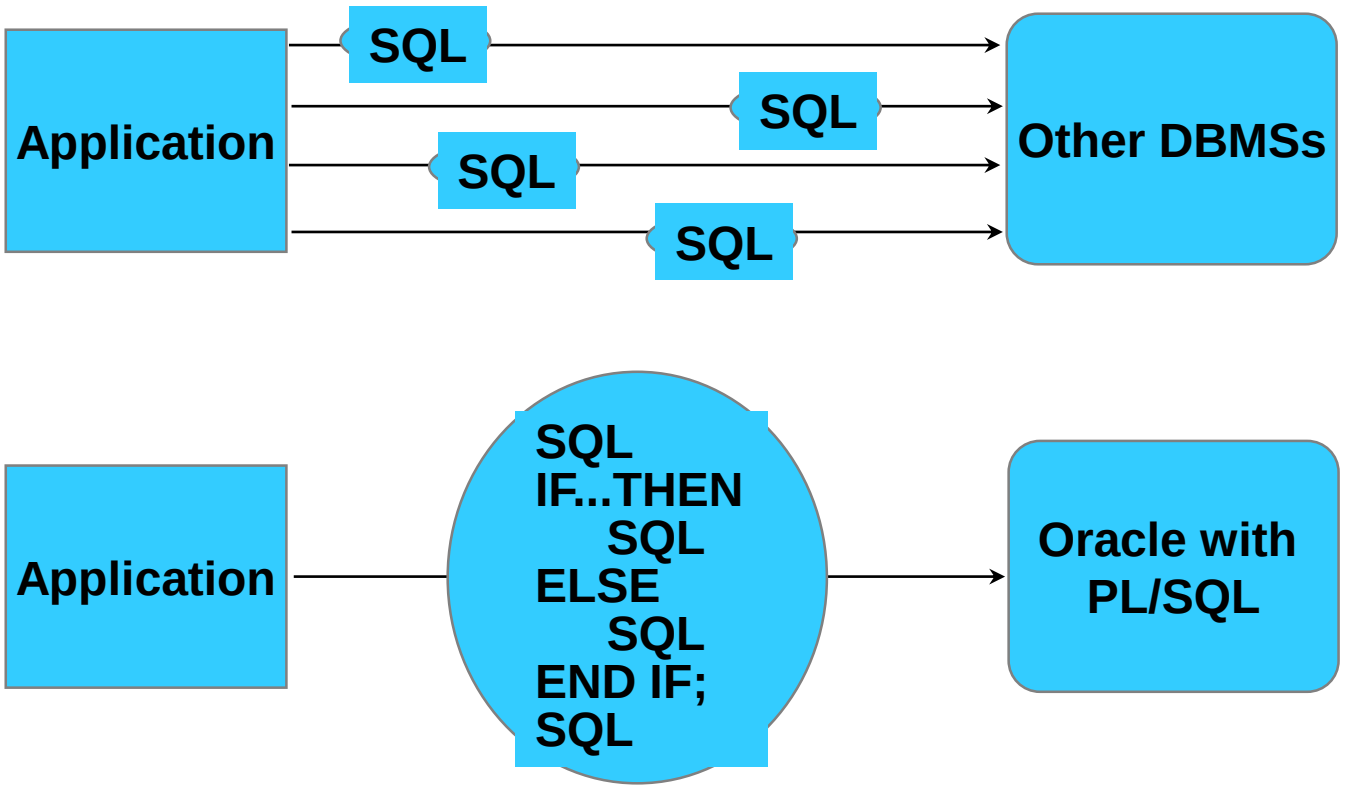
PL/SQL Environment



Copyright © 2006,
Digitech Technologies

Benefits of PL/SQL

Improved performance



Copyright © 2006,
Digitech Technologies

Benefits of PL/SQL

Modularized Program Development

Group logically related statements within blocks.

Nesting sub blocks inside larger blocks to build powerful programs.

Break down a complex problem into a set of manageable, well-defined, logical modules and implement the modules with blocks.

Place reusable PL/SQL code in libraries to be shared between Oracle Forms and Oracle Reports applications or store it in an Oracle server to make it accessible to any application that can interact with an Oracle database.

**Copyright © 2006,
Digitech Technologies**

Benefits of PL/SQL

PL/SQL is portable

We can declare variables

We can program with procedural language control structures.

Procedural Language Control Structures allow you to do the following:

- Execute a sequence of statements conditionally
- Execute a sequence of statements iteratively in a loop
- Process individually the rows returned by a multiple-row query with an explicit cursor

PL/SQL can handle errors.

The Error handling functionality in PL/SQL allows you to do the following:

- Process Oracle server errors with exception-handling routines
- Declare user-defined error conditions and process them with exception- handling routines

PL/SQL Block Structure

DECLARE (Optional)

Variables, cursors, user-defined exceptions

BEGIN (Mandatory)

- SQL statements
- PL/SQL statements

EXCEPTION (Optional)

Actions to perform when errors occur

END; (Mandatory)

**Copyright © 2006,
Digitech Technologies**

PL/SQL Block Structure

Executing Statements and PL/SQL Blocks

Place a semicolon (;) at the end of a SQL statement or PL/SQL control statement.

When the block is executed successfully, without unhandled errors or compile errors, the message output should be as follows:

PL/SQL procedure successfully completed.

Section keywords DECLARE, BEGIN, and EXCEPTION are not followed by semicolons.

END and all other PL/SQL statements require a semicolon to terminate the statement.

Note: In PL/SQL, an error is called an exception.

Copyright © 2006,
Digitech Technologies

PL/SQL Block Types

Anonymous PL/SQL Block	Named PL/SQL Blocks	
	Procedure	Function

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Copyright © 2006,
Digitech Technologies

Types of Variables

1. PL/SQL variables:
   - Scalar
   - Composite
   - Reference
   - LOB (large objects)

2. Non-PL/SQL variables:
   - Bind Variables
   - Host Variables

Copyright © 2006,
Digitech Technologies

PL/SQL Variables

Declare and initialize variables in the declaration section.

Assign new values to variables in the executable section.

Pass values into PL/SQL blocks through parameters.

View results through output variables.

**Copyright © 2006,
Digitech Technologies**

PL/SQL Variables

Declaration of a pl/sql variable

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

Ex:

```
DECLARE
  v_date          DATE;
  v_dno           NUMBER(2) NOT NULL := 5;
  v_location      VARCHAR2(13) := 'Hyderabad';
  c_incentive     CONSTANT NUMBER := 1000;
```

Copyright © 2006,
Digitech Technologies

PL/SQL Variables

- The first character must be a letter; the remaining characters can be letters, numbers, or special symbols(only # or $ or underscore(_))
- Initialize variables designated as NOT NULL and CONSTANT.
- Declare one identifier per line.
- Initialize identifiers by using the assignment operator (:=) or the DEFAULT reserved word.
- The names of the variables must not be longer than 30 characters.

Assignment:
identifier := *expression*;

**Copyright © 2006,
Digitech Technologies**

PL/SQL Variables: Naming Rules

Two variables can have the same name, provided they are in different blocks.

The variable names (identifiers) should not be the same as the table names or column names used in the block.

```
DECLARE
  empno         NUMBER(6);
BEGIN
  SELECT        empno
  INTO          empno
  FROM          emp
  WHERE         empno = 7369';
END;
/
```

Follow a naming convention for PL/SQL identifiers: e.g.: v_empno

Note: The names of local variables and formal parameters take precedence over the names of database tables.

The names of database table columns take precedence over the names of local variables.

Copyright © 2006, Digitech Technologies

Scalar Data Types

- **CHAR [(*maximum_length*)]**
- **VARCHAR2 (*maximum_length*)**
- **LONG**
- **NUMBER [(*precision, scale*)]**
- **BINARY_INTEGER**
- **PLS_INTEGER**
- **BOOLEAN**
- **DATE**

**Copyright © 2006,
Digitech Technologies**

Scalar Variable Declarations

```
DECLARE
  v_name          VARCHAR2(9);

  v_count         BINARY_INTEGER := 0;

  v_salary        NUMBER(9,2) := 0;

  v_shipdate      DATE := SYSDATE + 14;

  c_tax           CONSTANT NUMBER(3,2) := 6.75;

  v_isMale        BOOLEAN NOT NULL := TRUE;
```

Note: The life and scope of pl/sql variables is upto the end of the PL/SQL block.

Copyright © 2006,
Digitech Technologies

About Select statement within a PL/SQL Block

1. A select statement requires into clause.

syntax:

```
select <column_list> into <variable_list>
from <table_name>
where <condition(s)>;
```

2. A pre-defined exception named, NO_ DATA_FOUND is raised if select statement fails to return a value.

3. A pre-defined exception named, TOO_ MANY_ ROWS is raised if select statement returns more than one row values.

Note: This can be resolved by using cursors.

**Copyright © 2006,
Digitech Technologies**

About DBMS_OUTPUT.PUT_LINE

An Oracle-supplied packaged procedure

Must be enabled in SQL*Plus with **SET SERVEROUTPUT ON**

```
SET SERVEROUTPUT ON

DECLARE
  v_salary NUMBER(9,2) := &p_annualsalary;
BEGIN
  v_salary := v_salary/12;
  DBMS_OUTPUT.PUT_LINE ('Monthly salary is ' || v_salary);
END;
/
```

**Copyright © 2006,
Digitech Technologies**

About PL/SQL Block

Ex.
v_name := 'Smith';
v_birthdate := '12-JUN-1981' ;
v_salary := 18000;

- **Literals**
  - **–Character and date literals must be enclosed in single quotation marks.**
  - **–Numbers can be simple values or scientific notation.Ex. 3E5 means 3 * 10 to the power of 5 i.e 300000**
- **A slash (/) runs the PL/SQL block in a script file or in some tools such as SQL*PLUS.**

Copyright © 2006,
Digitech Technologies

Example of a Anonymous PL/SQl Block

```
DECLARE
        V_ENAME VARCHAR2(30);
        V_JOB VARCHAR2(30) ;
        V_SALARY NUMBER(10,2);
        V_DEPTNO NUMBER(3);
BEGIN
        SELECT   ENAME,JOB,SAL,DEPTNO
                INTO  V_ENAME,V_JOB,V_SALARY,V_DEPTNO
                FROM EMP WHERE EMPNO = &ENO;

        DBMS_OUTPUT.PUT_LINE(V_ENAME || ' , ' ||
                                        V_JOB || ' , ' ||
                                        V_SALARY || ' , ' ||
                                        V_DEPTNO);
END;
/
```

Copyright © 2006,
Digitech Technologies

Bind variables

Bind Variables are declared in the client environment (SQL*PLUS) which is also called as calling environment and they stay active until end of the session.

Syntax:

Declaration of a bind variable

VAR[IABLE] <variable_name> NUMBER | CHAR | CHAR (n) | VARCHAR2 (n)

Initializing a bind variable

> execute :bind_variable := value;

Copyright © 2006,
Digitech Technologies

Bind variables

```
>var  b_total number
>DECLARE
   v_empno emp.empno%type := &eno;
BEGIN
     select  sal + nvl(comm,0) into  :b_total from emp
    where  empno = v_empno;
END;
```

Note: Prefix bind variable with colon when referred within a PL/SQL block.

Displaying a bind variable

> print bind_variable

> print b_total

Copyright © 2006,
Digitech Technologies

Anchored Declaration: The %TYPE Attribute

To declare a variable according to:

- **A database column definition**
- **Another previously declared variable**

Prefix %TYPE with:

- **The database table and column**

  identifier **table_name.column_name%TYPE ;**

- **The previously declared variable name**

  identifier **identifier_name%TYPE ;**

Copyright © 2006,
Digitech Technologies

Anchored Declaration: The %TYPE Attribute

Examples:

```
...
  v_ename          emp.ename%TYPE;

  v_bal            NUMBER(7,2);

  v_min_bal        v_bal%TYPE := 1000;
...
```

**Copyright © 2006,
Digitech Technologies**

Using %TYPE variables in PL/SQL Block

```
declare
  v_ename emp.ename%type;
  v_job emp.job%type;
  v_sal emp.sal%type;
  begin
     select ename, job, sal into v_ename, v_job, v_sal
     from emp where empno = &eno;
     dbms_output.put_line(v_ename  ||'  , '|| v_job||'  ,  '|| v_sal);
  end;
```

Copyright © 2006,
Digitech Technologies

Anchored Declaration: The %ROWTYPE Attribute

A pl/sql variable of %ROWTYPE can hold entire row of a table.

Syntax:

Identifier table_name%rowtype;

```
v_erec      emp%ROWTYPE;

v_drec      dept%ROWTYPE;
```

**Copyright © 2006,
Digitech Technologies**

Using %ROWTYPE variables in PL/SQL Block

```
declare
 v_erec emp%rowtype;
begin

 select * into erec from emp where empno = &eno;
 dbms_output.put_line(v_erec.empno || `,' ||
                                v_erec.ename ||' ,'||
                                v_erec.job ||' , '||
                                v_erec.sal);
end;
```

Copyright © 2006,
Digitech Technologies

Boolean Variables

Only the values TRUE, FALSE, and NULL can be assigned

to a Boolean variable.

The variables are compared by the logical operators AND, OR, and NOT.

The variables always yield TRUE, FALSE, or NULL.

Arithmetic, character, and date expressions can be used to return a Boolean value.

Copyright © 2006,
Digitech Technologies

Boolean Variables

```
DECLARE
 v_flag BOOLEAN := FALSE;
BEGIN
  if v_flag then
    dbms_output.put_line('Hello');
  else
     dbms_output.put_line('Welcome');
  end if;
  v_flag := TRUE;
  if v_flag then
    dbms_output.put_line('Hello');
  else
     dbms_output.put_line('Welcome');
  end if;
END;
```

Copyright © 2006,
Digitech Technologies

Host Variables

Host variables are the variables declared in the host language. SQL statements embedded within the source code of host language can refer these host variables.

Note : To reference host variables, you must prefix the host variables with a colon (:) to distinguish them from declared PL/SQL variables.

Note: C source code that contains embedded SQL needs to be initially compiled with PRO* C.

sample.pc -> pro*c -> sample.c -> vc++ -> sample.exe

Copyright © 2006,
Digitech Technologies

Commenting Code

Prefix single-line comments with two dashes (--).

Place multiple-line comments between the symbols /* and */.

```
> var  b_msalary

> execute :b_msalary := 5000;

> DECLARE
  v_annualsal  NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_annualsal  :=  :b_msalary * 12;
  dbms_output.put_line ('Annual Salary : ' || v_annualsal);
END;    -- This is the end of the block
```

Copyright © 2006,
Digitech Technologies

Nested Blocks and Variable Scope

- PL/SQL blocks can be nested wherever an executable statement is allowed.
- A nested block becomes a statement.
- An exception section can contain nested blocks.
- The scope of an identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.
- References to an identifier are resolved according to its scope and visibility

Copyright © 2006,
Digitech Technologies

Nested Blocks and Variable Scope

Example

```
DECLARE
  x  BINARY_INTEGER;
BEGIN
                                          Scope of x
   DECLARE
     y  NUMBER;
   BEGIN
      y :=  x;                            Scope of y
   END

END;
```

**Copyright © 2006,
Digitech Technologies**

Identifier Scope

An identifier is visible in the regions where you can reference the identifier without having to qualify it:

- A block can look *up* to the enclosing block.
- A block cannot look *down* to enclosed blocks.

Note: An identifier is visible in the block in which it is declared and in all nested sub- blocks.

If the block does not find the identifier declared locally, it looks *up* to the declarative section of the enclosing (or parent) blocks.

The block never looks *down* to enclosed (or child) blocks or sideways to sibling blocks.

Copyright © 2006,
Digitech Technologies

Qualify an Identifier

- The qualifier can be the label of an enclosing block.
- Qualify an identifier by using the block label prefix.

```
<<outer>>
DECLARE
 i integer := 5;
 BEGIN
 <<inner>>
    DECLARE
     i integer := 2;
     j integer;
      BEGIN
         j := inner.i ** outer.i ;
        dbms_output.put_line(j);
      END;
  END;
```

Copyright © 2006,
Digitech Technologies

Operators in PL/SQL

- Logical operators : AND , OR , NOT
- Arithmetic operators : + , - , * , /
- Relational operators : < , <= , > , >= , <> or != , =
- Concatenation : ||
- Exponential operator : **

Note: Parentheses can be used to control order of operations

**Copyright © 2006,
Digitech Technologies**

SQL Statements in PL/SQL

- Extract a row of data from the database by using the SELECT command.
- Make changes to rows in the database by using DML commands.
- Control a transaction with the COMMIT, ROLLBACK, or SAVEPOINT command.
- Determine DML outcome with implicit cursor attributes.

Copyright © 2006,
Digitech Technologies

SQL Statements in PL/SQL

About PL/SQL blocks containing DML statements and transaction control commands

- The keyword END signals the end of a PL/SQL block, not the end of a transaction. Just as a block can span multiple transactions, a transaction can span multiple blocks.

- PL/SQL does not directly support data definition language (DDL) statements, such as CREATE TABLE, ALTER TABLE or DROP TABLE.

- PL/SQL does not support data control language (DCL) statements, such as GRANT or REVOKE.

Copyright © 2006,
Digitech Technologies

SELECT Statements in PL/SQL

SELECT ***select_list*** **INTO** ***variable_name*** **[,** ***variable_name*****]...**
FROM ***table***
[WHERE ***condition*****];**

```
SET SERVEROUTPUT ON
DECLARE
  v_payroll     emp.sal%type;
  v_deptno        emp.deptno%type := &dno;
BEGIN
  SELECT        SUM(sal) + nvl(sum(comm),0)
  INTO          v_payroll
  FROM          emp
  WHERE         deptno = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The payroll of dept:' || v_deptno||'= '  ||
                                                        v_payroll);
END;
/
```

Copyright © 2006,
Digitech Technologies

Manipulating Data Using PL/SQL

Make changes to database tables by using DML commands:

- INSERT
- UPDATE
- DELETE

Inserting Data

```
BEGIN
  INSERT INTO emp  (empno,ename,job,sal,deptno)
  VALUES (emp_seq.NEXTVAL, 'Ravi Kumar', 'Manager', 18000,10);
 -- COMMIT;
END;
```

Copyright © 2006,
Digitech Technologies

Updating Data

```
DECLARE
  v_sal_inc       emp.sal%TYPE := 2500;
BEGIN
  UPDATE          emp
  SET             sal = sal + v_sal_inc
  WHERE           upper(job) = 'MANAGER' ;
  -- COMMIT ;
END;
/
```

Note: PL/SQL variable assignments always use **:=** and SQL column assignments always use **=**

Copyright © 2006,
Digitech Technologies

Deleting Data

```
DECLARE
  v_deptno  emp.deptno%TYPE := &dno;
BEGIN
  DELETE FROM   emp
  WHERE  deptno = v_deptno;
  -- COMMIT;
END;
/
```

Note: The WHERE clause is used to determine which rows are affected. If no rows are modified, no error occurs, unlike the SELECT statement in PL/SQL.

Copyright © 2006,
Digitech Technologies

PL/SQL CONSTRUCTS

There are two types of constructs:

1. Decision-making elements

2.Iterative controls (Loops)

Decision-making elements

1. Simple if statement:

```
IF <CONDITION> THEN
  _____;
  _____;
END IF;
```

**Copyright © 2006,
Digitech Technologies**

Decision-making elements

2. If else statement:

```
IF <CONDITION> THEN
        _____;
        _____;
ELSE
        _____;
        _____;
END IF;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

3.Nested if statement:

```
a. IF <condition1>  THEN
        IF <condition2> THEN
            ----------------------;
            ----------------------;
        END IF;
   END IF;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

4. Nested if statement with else clauses

```
IF <condition1>  THEN
        IF <condition2> THEN
                ----------------------;
                ----------------------;
        ELSE
                ----------------------;
                ----------------------;
        END IF;
ELSE
        ----------------------;
        ----------------------;
END IF;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

Ex.

```
Declare
        v_empno      emp.empno%type := &enum;
        v_desig    emp.job%type;
        v_years   number(2);
        v_incr     emp.sal%type;
Begin
        select job, round( (sysdate-hiredate)/365)
                into v_desig, v_years from emp where empno = v_empno;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

```
        if upper(trim(v_desig)) = upper('manager') then
                if v_years > 10  then
                        v_incr  := 10000;
                else
                        v_incr := 8000;
                end if;
        else
                v_incr := 4000;
        end if;

        update emp set sal=sal+incr where empno = v_empno;
        commit;
end;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

4. IF ... ELSIF Ladder

```
IF  <CONDITION1> THEN
        _____;
        _____;
ELSIF <CONDITION2> THEN
        _____;
        _____;
ELSIF <CONDITION3> THEN
        _____;
        _____;
ELSE
        _____;
        _____;
END IF;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

```
Declare
        v_sal emp.sal%type;
        v_eno emp.empno%type := &eno;
begin
        select sal into v_sal from emp
                where empno = v_eno;
```

Copyright © 2006,
Digitech Technologies

Decision-making elements

```
if v_sal < 10000 then
        v_sal := v_sal + 4000;
elsif v_sal < 20000 then
        v_sal := v_sal + 3000;
elsif v_sal < 30000 then
        v_sal := v_sal + 2000;
else
        v_sal := v_sal + 1000;
end if;
update emp set sal = v_sal where empno = v_eno;
end;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

1. Unconditional loop

```
LOOP
        _____;
        _____;
END LOOP;
```

To terminate the loop

```
LOOP
        ----------;
        ----------;
        IF  <CONDITION> THEN
                EXIT;
        END IF;
END LOOP;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

Instead of using the IF condition to exit the Loop we can use WHEN condition also

```
LOOP
        ----------;
        ----------;
        EXIT WHEN  <CONDITION>;
END LOOP;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

Ex.

```
Declare
        v_count number(3) := 1;
Begin
        Loop
                dbms_output.put_line(v_count);
                v_count := v_count + 1;
                exit when v_count = 10;
        End loop;
End;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

2. While Loop

```
WHILE <CONDITION>
LOOP
        ----------------------;
        ----------------------;
END LOOP;
```

**Copyright © 2006,
Digitech Technologies**

Iterative Controls (Loops)

Ex.

```
Declare
        v_count number(3) := 1;
Begin
        while v_count <=10
        Loop
                dbms_output.put_line(v_count );
                v_count := v_count + 1;
        End Loop;
End;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

3. For Loop

```
FOR <LOOP_COUNTER> IN [REVERSE] LOWER_LIMIT..UPPER_LIMIT
LOOP
        ----------------------;
        ----------------------;
END LOOP;
```

**Copyright © 2006,
Digitech Technologies**

Iterative Controls (Loops)

About For Loop :

1. The loop_counter is implicitly declared as Integer type
2. The loop_counter implicitly increments for each iteration
3. The loop_counter behaves like constant within the loop.

   i.e. the loop counter cannot be assigned or modified.
4. The life & scope of a loop counter is limited to the loop.

   i.e. it cannot be accessed outside the loop.
5. The lower-limit should be always less than upper-limit. If not, it leads to error. If they are equal, loop iterates only once.
6. If REVERSE option is used , control initially takes the upper-limit into loop counter and decrements for each iteration until loop counter reaches lower-limit.

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

Ex.

```
begin
        for i in 1..10
        loop
                dbms_output.put_line(i) ;
        end loop;
end;
```

```
begin
        for i in reverse 1..10
        loop
                dbms_output.put_line(i);
        end loop;
end;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

Example of a nested loop

```
begin
 for i in 1..10  loop
          for j in 1..10 loop
                    dbms_output.put( rpad(i*j,5) );
          end loop;
  dbms_output.put_line(' ');
 end loop;
end;
```

Copyright © 2006,
Digitech Technologies

Iterative Controls (Loops)

Ex.

```
begin
        for i in 1..10
        loop
          if (mod(i,2) = 0 )    then
                  NULL;
           else
                  dbms_output.put_line(i);
         end if;
        end loop;
end;
```

Copyright © 2006,
Digitech Technologies