Writing executable statements

Inside PL/SQL block

- **Identifiers**: Identifiers are the names given to PL/SQL objects

Ex: v_empno, v_ename, " first Name "

Not recommended
Case sensitive
You can use space
You can use reserved word

- **Delimiters** : symbols that have special meaning

Ex: ; + *

- **Literals**: Any value that is assigned to a variable is a literal.

Ex: v_ename:='khaled' ; , v_empno:=10; v_flag:=true;

- **Comments**: used to describe you code

Ex:
```
--this code calculate sum of salaries
 /* this code calculate
 sum of salaries
 */
```

Delimiters are simple or compound symbols that have special meaning in PL/SQL.

Simple symbols

Symbol	Meaning
+	Addition operator
–	Subtraction/negation operator
*	Multiplication operator
/	Division operator
=	Equality operator
@	Remote access indicator
;	Statement terminator

Compound symbols

Symbol	Meaning
<>	Inequality operator
!=	Inequality operator
||	Concatenation operator
--	Single-line comment indicator
/*	Beginning comment delimiter
*/	Ending comment delimiter
:=	Assignment operator

Commenting Code

- Prefix single-line comments with two hyphens (--).
- Place multiple-line comments between the symbols /* and */.

Example:

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

SQL Functions in PL/SQL

- Available in procedural statements:
  - Single-row functions

```
Ex: v_ename:=substr(ename,1,5 );
     v_lname:= length( first_name );
     v_comm:= nvl( comm,0 );
     v_date:=add_months( hiredate,3 );
```

- Not available in procedural statements:
  - DECODE
  - Group functions

But you can use it in SQL statement inside PL/SQL

Starting in 11*g*:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  v_new_id := my_seq.NEXTVAL;
END;
/
```

Before 11*g*:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

Data Type Conversion

- Converts data to comparable data types
- Is of two types:
  - Implicit conversion
  - Explicit conversion
- Functions:
  - `TO_CHAR`
  - `TO_DATE`
  - `TO_NUMBER`
  - `TO_TIMESTAMP`

Nested Blocks

PL/SQL blocks can be nested.

- An executable section (`BEGIN … END`) can contain nested blocks.
- An exception section can contain nested blocks.

```
Declare
…
Begin
     …
     …
          declare
          ….
          begin
          …
         End;
End;
```

Nested Blocks

Example:

```
DECLARE
 v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
   v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
   DBMS_OUTPUT.PUT_LINE(v_inner_variable);
   DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
 DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

Programming Guidelines

Make code maintenance easier by:

- Documenting code with comments
- Developing a case convention for the code
- Developing naming conventions for identifiers and other objects
- Enhancing readability by indenting

Category	Case Convention	Examples
SQL statements	Uppercase	SELECT, INSERT
PL/SQL keywords	Uppercase	DECLARE, BEGIN, IF
Data types	Uppercase	VARCHAR2, BOOLEAN
Identifiers and parameters	Lowercase	v_sal, emp_cursor, g_sal, p_empno
Database tables and columns	Lowercase	employees, employee_id, department_id

Indenting Code

For clarity, indent each level of code.

```
BEGIN
  IF x=0 THEN
     y:=1;
  END IF;
END;
/
```

```
DECLARE
  deptno        NUMBER(4);
  location_id   NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    deptno,
          location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
...
END;
/
```

▶ Thank You