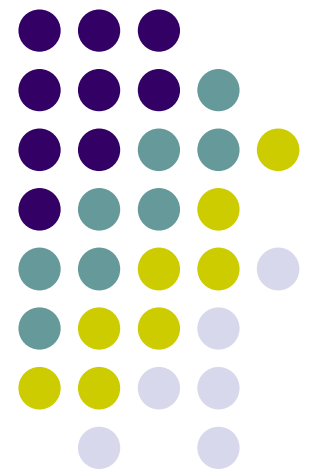


# Oracle PL/SQL Language

---

**CIS 331:  
Introduction to  
Database Systems**





# Topics:

- **Structure of a PL/SQL program**
- **Exceptions**
- **3-valued logic**
- **Loops (unconditional, while, for)**
- **Cursors**
- **Procedures**
- **Functions**
- **Triggers**

# Structure of a PL/SQL program



- PL/SQL code follows this scheme:

## **DECLARE**

- variables
- types
- local subprograms

## **BEGIN**

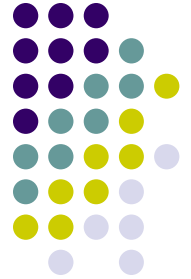
- procedural statements
- SQL statements

## **EXCEPTION**

- exception handling statements

## **END;**

# A simple PL/SQL program



```
SET SERVEROUTPUT ON      -- so your printouts will show up on the screen
```

```
DECLARE
```

```
    A NUMBER;
```

```
    B REAL;
```

```
    D VARCHAR2(16);
```

```
BEGIN
```

```
    A := 13.6;
```

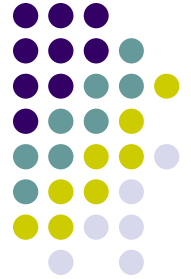
```
    B := 12;
```

```
    DBMS_OUTPUT.PUT_LINE('A / B = ' || A/B);
```

```
    D := 'I am a happy camper';
```

```
    DBMS_OUTPUT.PUT_LINE(D);
```

```
END;
```



# Exception handling

- An exception would break the execution of the program. PL/SQL provides means to handle this:

```
DECLARE
    A          NUMBER;
    B          REAL := 12;
    C          NUMBER := 0;
    MY_EXCEPTION EXCEPTION;
BEGIN
    A := 13.6;
    DBMS_OUTPUT.PUT_LINE('A / B = ' || A/B);
    -- C := 12;
    DBMS_OUTPUT.PUT_LINE('A / C = ' || A/C);

    RAISE MY_EXCEPTION;

EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('Ooooups. Division by zero!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Ooooups. Something went wrong!');
END;
```



# 3-valued logic

- PL/SQL's boolean variables support 3-valued logic. This means that a boolean variable can take values TRUE, FALSE or NULL.

```
DECLARE
    D BOOLEAN := null;
BEGIN
    IF D THEN
        DBMS_OUTPUT.PUT_LINE('D is true. ');
    ELSE
        DBMS_OUTPUT.PUT_LINE('D is false. ');
    END IF;

    IF NOT D THEN
        DBMS_OUTPUT.PUT_LINE('D is false. ');
    ELSE
        DBMS_OUTPUT.PUT_LINE('D is true. ');
    END IF;
END;
```



# Nulls

- This can be rectified by explicitly checking for nulls:

```
DECLARE
    D BOOLEAN := null;
BEGIN

    IF D THEN
        DBMS_OUTPUT.PUT_LINE('D is true. ');
    ELSIF D IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('D is null. ');
    ELSE
        DBMS_OUTPUT.PUT_LINE('D is false. ');
    END IF;

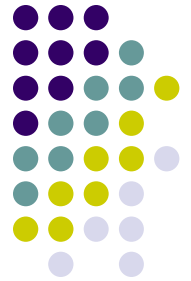
END;
```



# Loops, unconditional loop

- An example of an unconditional loop:

```
DECLARE
    i NUMBER := 1;
BEGIN
    LOOP
        EXIT WHEN i>10;
        DBMS_OUTPUT.PUT_LINE(i);
        i := i+1;
    END LOOP;
END;
```



# While loop

- An example of a while loop:

```
DECLARE
    i INTEGER := 1;
BEGIN
    WHILE i < 10 LOOP
        DBMS_OUTPUT.PUT_LINE(i);
        i := i+1;
    END LOOP;
END;
```



# For loop

- An example of a for loop:

```
BEGIN
  FOR k IN 1..10 LOOP
    -- note that k was not declared
    DBMS_OUTPUT.PUT_LINE(k);
  END LOOP;

END;
```

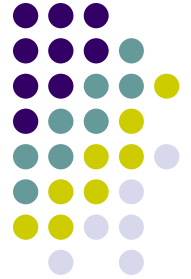


# Cursors

- So far PL/SQL looked like a regular (although primitive) programming language. Let us introduce some special data types:

```
DECLARE
  i INTEGER;
  j i%TYPE;
  name1 VARCHAR2(32);
  name2 students.last_name%TYPE;
  student_rec students%ROWTYPE;           -- sort of like struct in C
  CURSOR s1 IS                             -- and now the plot thickens:
    SELECT first_name, last_name
    FROM students;
  student_name s1%ROWTYPE;
BEGIN
  OPEN s1;
  LOOP
    FETCH s1 INTO student_name;
    EXIT WHEN s1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(student_name.first_name || ' ' ||
student_name.last_name);
  END LOOP;
  CLOSE s1;
END;
```

# More cursors



```
DECLARE

    CURSOR p1
    IS
    SELECT first_name, last_name
        FROM professors;

    professor_name p1%ROWTYPE;

BEGIN
    FOR professor_name IN p1 LOOP
        DBMS_OUTPUT.PUT_LINE (professor_name.first_name
            || ' ' || professor_name.last_name);
    END LOOP;
END;
```

# Cursors with parameters



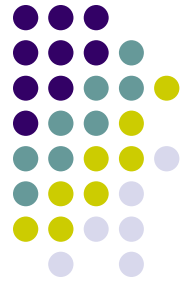
```
DECLARE
    CURSOR class_cursor (class_name VARCHAR2) IS
    SELECT c.code, c.name
    FROM classes c, (select code from departments) b
        WHERE name like '%' || class_name || '%'
        AND c.department = b.code
    ;
    class_row class_cursor%ROWTYPE;
BEGIN
    OPEN class_cursor ('Comp');
    LOOP
        FETCH class_cursor INTO class_row;
        EXIT WHEN class_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Code: ' || class_row.code || ' Name: ' ||
            class_row.name);
    END LOOP;
    CLOSE class_cursor;

END;
```



# Program units

- Typical PL/SQL program units are:
  - procedures
  - functions
  - database triggers
  - packages



# Procedures

- A sample procedure – print a name for a given SSN:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE PROCEDURE names
  (ssnum IN VARCHAR) IS

  CURSOR p1 IS
  SELECT first_name, last_name
     FROM professors
        WHERE ssn = ssnum;

  professor_name p1%ROWTYPE;

BEGIN
  FOR professor_name IN p1 LOOP
    DBMS_OUTPUT.PUT_LINE(professor_name.first_name || ' ' ||
      professor_name.last_name);
  END LOOP;
END;
```



# Calls and errors

- Which can then be called by

```
EXECUTE names ('2345789078');
```

- **SHO ERR** (abbrev. from show errors) - useful for debugging



# Functions

- A sample function – return someone's age in years given his or her birthday:

```
CREATE OR REPLACE FUNCTION age
  (dob IN DATE)
  RETURN NUMBER
  IS

  current_age NUMBER;

BEGIN
  current_age := TRUNC((SYSDATE - dob) / 365.25);
  RETURN current_age;
END;
```

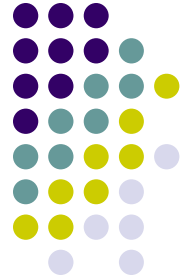
# Functions



- Which can then be used in SQL\*Plus client as any other function:

```
SELECT sysdate()  
      FROM dual  
;
```

```
SELECT age(to_date('08-12-1978', 'mm-dd-yyyy'))  
      AS age  
      FROM dual  
;
```



# Triggers

- Syntax of Oracle triggers (taken from the Oracle PL/SQL manual):

```
CREATE OR REPLACE TRIGGER trigger_name
    [BEFORE | AFTER] [INSERT | UPDATE | DELETE]
    ON table_name
    [FOR EACH ROW] [WHEN condition]
    BEGIN
        --
        -- trigger body
        --
    END;
```