DATABASE MANAGEMENT SYSTEMS

Subject Teacher: Zartasha Baloch

INTRODUCTION TO PL/SQL

Lecture # 29, 30, 31 & 32

Disclaimer: The material used in this presentation to deliver the lecture i.e., definitions/text and pictures/graphs etc. does not solely belong to the author/presenter. The presenter has gathered this lecture material from various sources on web/textbooks. Following sources are especially acknowledged:

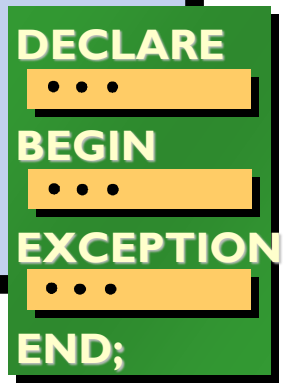
1. Connolly, Thomas M., and Carolyn E. Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.
2. *https://www.tutorialspoint.com/plsql/index.htm*
3. *https://www.oracle.com/database/technologies/appdev/plsql.html*
4. Greenberg, Nancy, and Instructor Guide PriyaNathan. "Introduction to Oracle9i: SQL." ORACLE, USA (2001).

ABOUT PL/SQL

- PL/SQL is an extension to SQL with design features of programming languages.
- Data manipulation and query statements of SQL are included within procedural units of code.

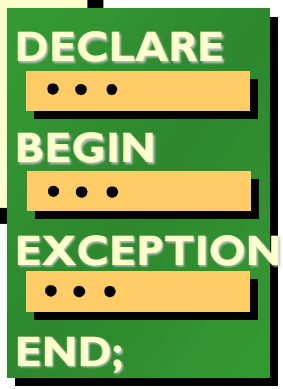
PL/SQL BLOCK STRUCTURE

- DECLARE – Optional
  - Variables, cursors, user-defined exceptions
- BEGIN – Mandatory
  - SQL statements
  - PL/SQL statements
- EXCEPTION – Optional
  - Actions to perform when errors occur
- END; – Mandatory



PL/SQL BLOCK STRUCTURE

```
DECLARE
  v_variable  VARCHAR2(5);
BEGIN
  SELECT      column_name
    INTO      v_variable
    FROM      table_name;
EXCEPTION
  WHEN exception_name THEN
  ...
END;
```



BLOCK TYPES

- Anonymous

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

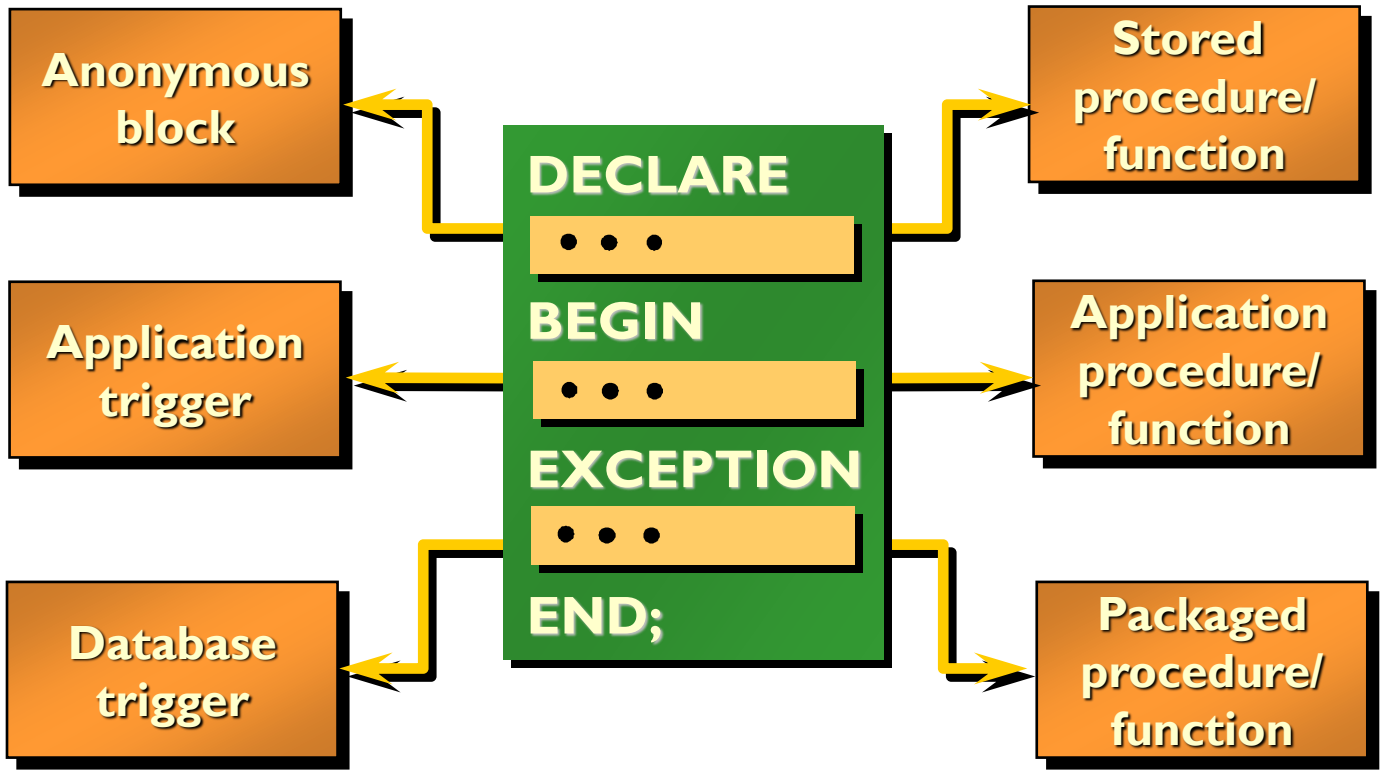
Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Function

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

END;
```

PROGRAM CONSTRUCTS



HANDLING VARIABLES IN PL/SQL

- 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.

TYPES OF VARIABLES

- PL/SQL variables:
  - Scalar
  - Composite
  - Reference
  - LOB (large objects)
- Non-PL/SQL variables: Bind and host variables

DECLARING PL/SQL VARIABLES

Syntax

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

Examples

```
Declare
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

DECLARING PL/SQL VARIABLES

- Guidelines
  - Follow naming conventions.
  - Initialize variables designated as NOT NULL.
  - Initialize identifiers by using the assignment operator (:=) or the DEFAULT reserved word.
  - Declare at most one identifier per line.

NAMING RULES

- Two variables can have the same name, provided they are in different blocks.
- The variable name (identifier) should not be the same as the name of table columns used in the block.

```
DECLARE
   empno  NUMBER(4);
BEGIN
   SELECT     empno
   INTO       empno
   FROM       emp
   WHERE      ename = 'SMITH';
END;
```

Adopt a naming convention for PL/SQL identifiers: for example, v_empno

ASSIGNING VALUES TO VARIABLES

Syntax

- *identifier* := *expr*;

Examples

Set a predefined hiredate for new employees.

```
v_hiredate := '31-DEC-98';
```

Set the employee name to “Maduro.”

```
v_ename := 'Maduro';
```

VARIABLE INITIALIZATION AND KEYWORDS

- Using:
  - Assignment operator (:=)
  - DEFAULT keyword
  - NOT NULL constraint

BASE SCALAR DATATYPES

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

SCALAR VARIABLE DECLARATIONS

- Examples

```
v_job           VARCHAR2(9);
v_count         BINARY_INTEGER := 0;
v_total_sal     NUMBER(9,2) := 0;
v_orderdate     DATE := SYSDATE + 7;
c_tax_rate      CONSTANT NUMBER(3,2) := 8.25;
v_valid         BOOLEAN NOT NULL := TRUE;
hours_worked    INTEGER DEFAULT 40;
```

DECLARING BOOLEAN VARIABLES

- Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.
- The variables are connected 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.

- Example
  - V_bool BOOLEAN := FALSE;

THE %TYPE ATTRIBUTE

- Declare a variable according to:
  - A database column definition
  - Another previously declared variable
- Prefix %TYPE with:
  - The database table and column
  - The previously declared variable name

THE %TYPE ATTRIBUTE

```
SQL> DECLARE
name VARCHAR2(20) := 'JoHn SmItH';
upper_name name%TYPE; -- inherits data type and default value
lower_name name%TYPE; -- inherits data type and default value
init_name name%TYPE; -- inherits data type and default value
BEGIN
DBMS_OUTPUT.PUT_LINE ('name: ' || name);
DBMS_OUTPUT.PUT_LINE ('upper_name: ' || UPPER(name));
DBMS_OUTPUT.PUT_LINE ('lower_name: ' || LOWER(name));
DBMS_OUTPUT.PUT_LINE ('init_name: ' || INITCAP(name));
END;
/
```

OUTPUT

name: JoHn SmItH
upper_name: JOHN SMITH
lower_name: john smith
init_name: John Smith
PL/SQL procedure successfully completed.

THE %TYPE ATTRIBUTE

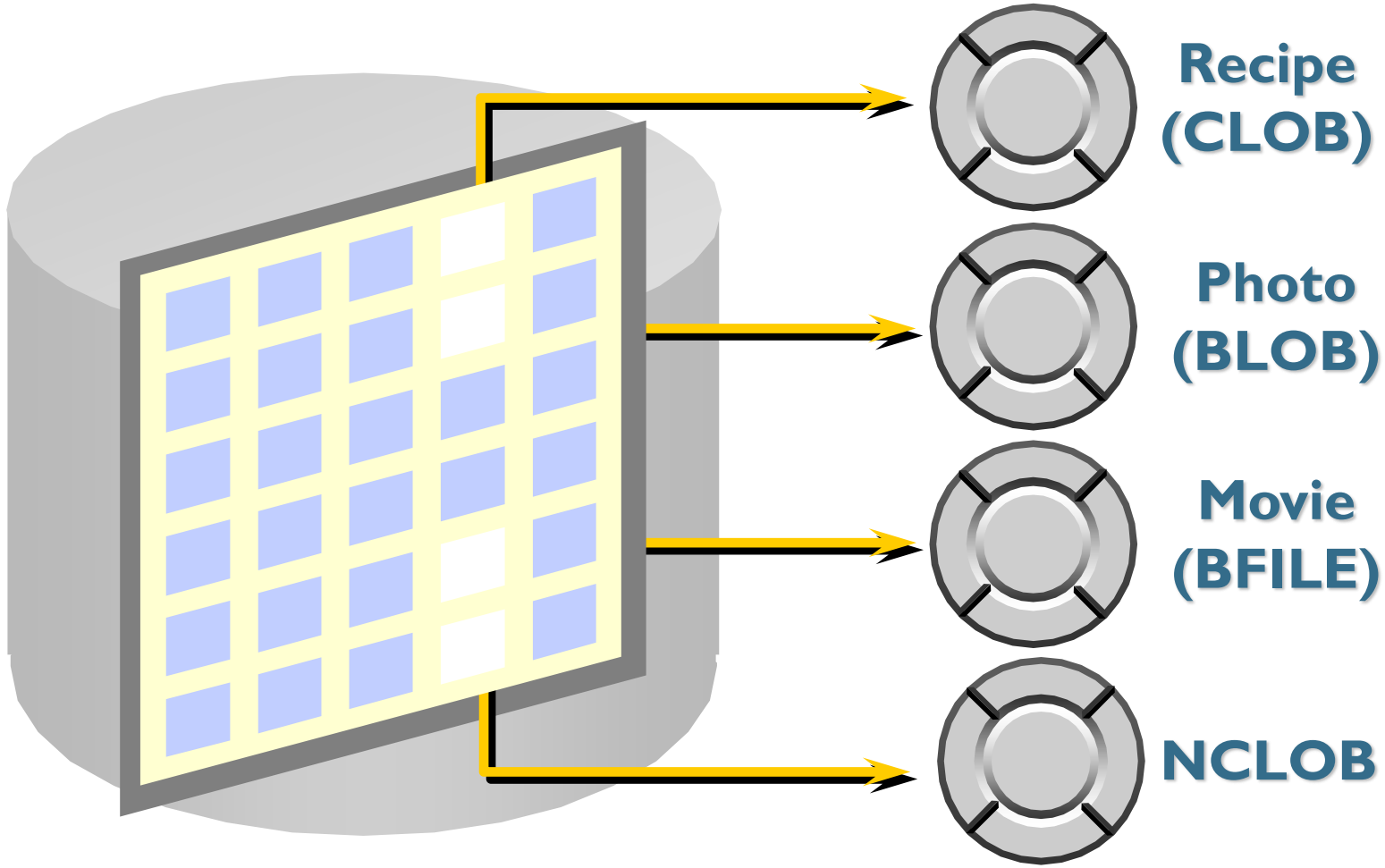
```
SQL> DECLARE
 v_empid employees.empid%TYPE;
v_deptid employees.deptid%TYPE;
v_deptname employees.deptname%TYPE;
```

%ROWTYPE

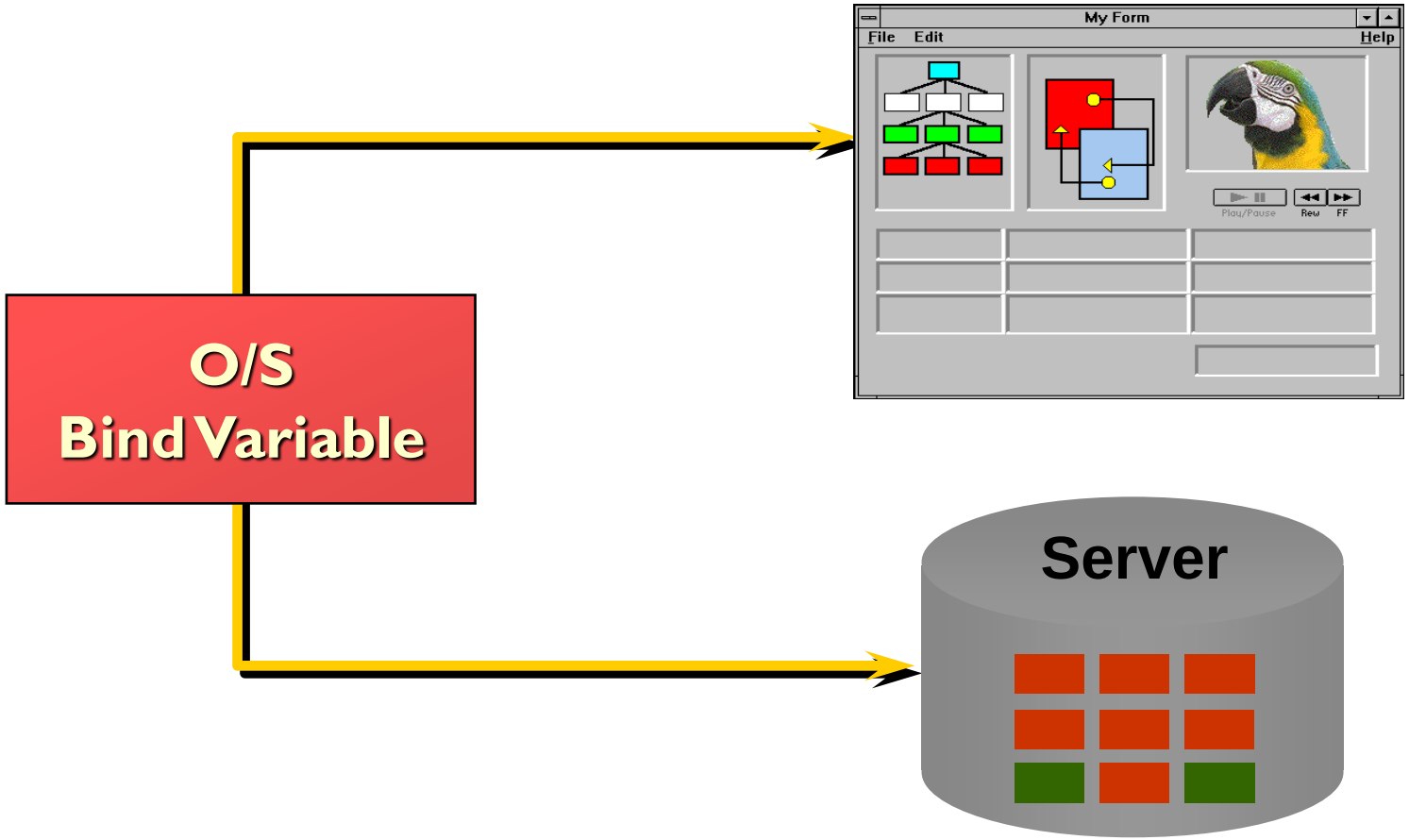
- Example

  emprec employees%ROWTYPE;

LOB DATATYPE VARIABLES



BIND VARIABLES



REFERENCING NON-PL/SQL VARIABLES

- Store the annual salary into a SQL*Plus host variable.

```
:g_monthly_sal := v_sal / 12;
```

  - Reference non-PL/SQL variables as host variables.
  - Prefix the references with a colon (:).

USING BIND VARIABLES

- To reference a bind variable in PL/SQL, you must prefix its name with a colon (:).

Example

```
VARIABLE g_salary NUMBER
DECLARE
   v_sal      emp.sal%TYPE;
BEGIN
   SELECT     sal
   INTO       v_sal
   FROM       emp
   WHERE      empno = 7369;
  :g_salary          := v_sal;
END;
/
```

DBMS_OUTPUT.PUT_LINE

- An Oracle-supplied packaged procedure
- An alternative for displaying data from a PL/SQL block
- Must be enabled in SQL*Plus with
- SET SERVEROUTPUT ON

PL/SQL BLOCK SYNTAX AND GUIDELINES

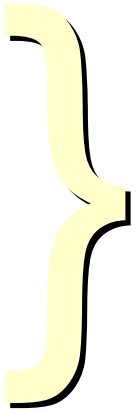
- Statements can continue over several lines.
- Lexical units can be separated by:
  - Spaces
  - Delimiters
  - Identifiers
  - Literals
  - Comments

PL/SQL BLOCK SYNTAX AND GUIDELINES

- Identifiers
  - Can contain up to 30 characters
  - Cannot contain reserved words unless enclosed in double quotation marks
  - Must begin with an alphabetic character
  - Should not have the same name as a database table column name

SQL FUNCTIONS IN PL/SQL

- Available:
  - Single-row number
  - Single-row character
  - Datatype conversion
  - Date
- Not available:
  - DECODE
  - Group functions



Same as in SQL

PL/SQL FUNCTIONS

Examples

- Build the mailing list for a company.

```
v_mailing_address := v_name||CHR(10)||
                     v_address||CHR(10)||v_state||
                     CHR(10)||v_zip;
```

- Convert the employee name to lowercase.

```
v_ename           := LOWER(v_ename);
```

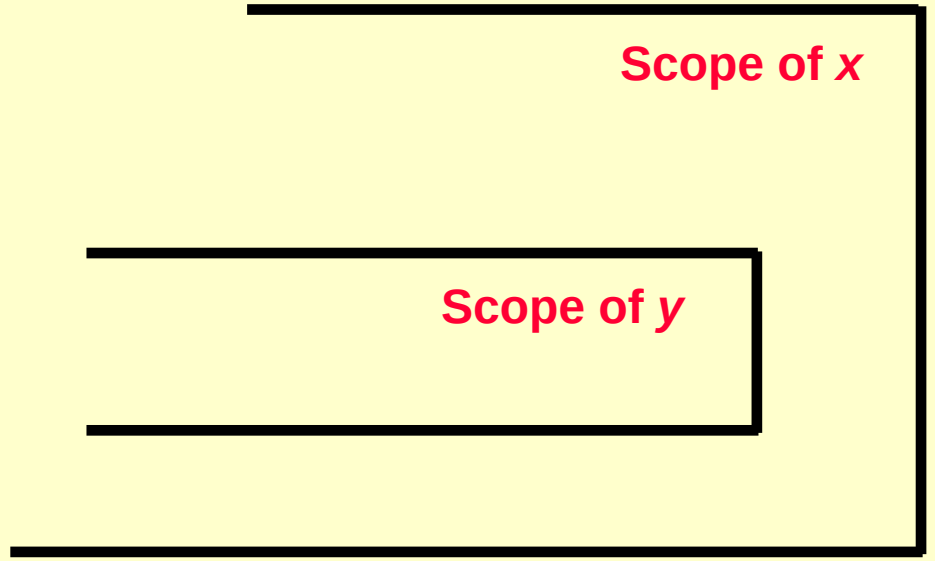
DATA TYPE CONVERSION

- Convert data to comparable datatypes.
- Mixed datatypes can result in an error and affect performance.
- Conversion functions:
  - TO_CHAR
  - TO_DATE
  - TO_NUMBER

```
DECLARE
    v_date VARCHAR2(15);
BEGIN
    SELECT TO_CHAR(hiredate,
                   'MON. DD, YYYY')
    INTO   v_date
    FROM   emp
    WHERE  empno = 7839;
END;
```

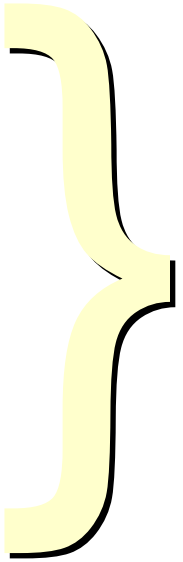
NESTED BLOCKS AND VARIABLE SCOPE

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
    ...
  END;
  ...
END;
```



OPERATORS IN PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations
- Exponential operator (**)



Same as in SQL

INDENTING CODE

- For clarity, indent each level of code.

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

```
DECLARE
   v_deptno    NUMBER(2);
   v_location      VARCHAR2(13);
BEGIN
   SELECT   deptno,
            loc
   INTO     v_deptno,
            v_location
   FROM     dept
   WHERE    dname = 'SALES';
...
END;
```

SQL STATEMENTS IN PL/SQL

- Extract a row of data from the database by using the SELECT command. Only a single set of values can be returned.
- 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 cursors.

SELECT STATEMENTS IN PL/SQL

- Retrieve data from the database with SELECT.
- Syntax

```
SELECT select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
WHERE   condition;
```

RETRIEVING DATA IN PL/SQL

- Retrieve the order date and the ship date for the specified order.

```
DECLARE
  v_orderdate   ord.orderdate%TYPE;
  v_shipdate    ord.shipdate%TYPE;
BEGIN
  SELECT   orderdate, shipdate
  INTO     v_orderdate, v_shipdate
  FROM     ord
  WHERE    id = 620;
       ...
END;
```

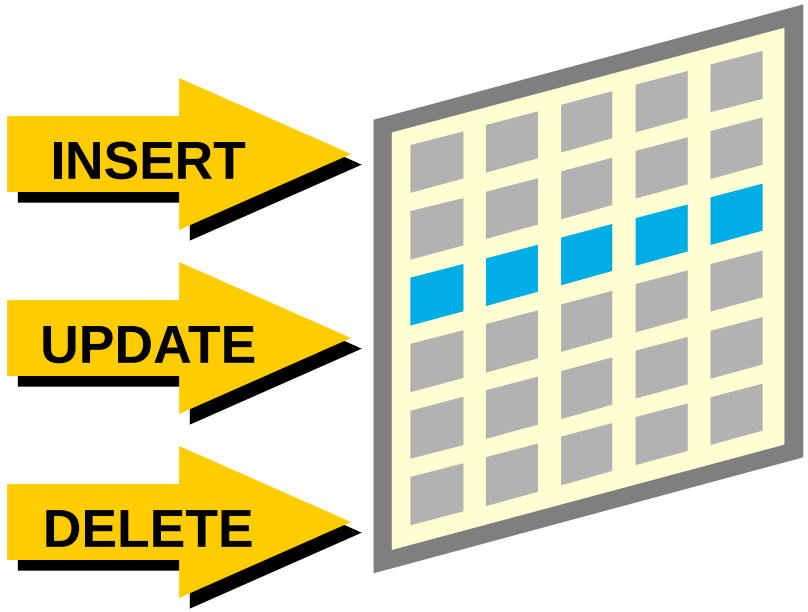
RETRIEVING DATA IN PL/SQL

- Return the sum of the salaries for all employees in the specified department.

```
DECLARE
   v_sum_sal    emp.sal%TYPE;
   v_deptno     NUMBER NOT NULL := 10;
BEGIN
   SELECT       SUM(sal)  -- group function
   INTO         v_sum_sal
   FROM         emp
   WHERE        deptno = v_deptno;
END;
```

MANIPULATING DATA USING PL/SQL

- Make changes to database tables by using DML commands:
  - INSERT
  - UPDATE
  - DELETE



UPDATING DATA

- Increase the salary of all employees in the emp table who are Analysts.

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE      emp
  SET         sal = sal + v_sal_increase
  WHERE       job = 'ANALYST';
END;
```

DELETING DATA

- Delete rows that belong to department 10 from the emp table.

```
DECLARE
  v_deptno    emp.deptno%TYPE := 10;
BEGIN
  DELETE FROM    emp
  WHERE          deptno = v_deptno;
END;
```

NAMING CONVENTIONS

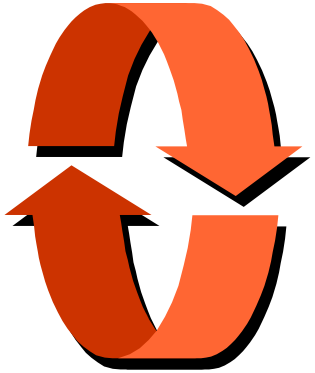
```
DECLARE
   orderdate      ord.orderdate%TYPE;
   shipdate            ord.shipdate%TYPE;
   ordid      ord.ordid%TYPE := 601;
BEGIN
   SELECT orderdate, shipdate
   INTO   orderdate, shipdate
   FROM   ord
   WHERE  ordid = ordid;
END;
SQL> /
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested
number of rows
ORA-06512: at line 6
```

COMMIT AND ROLLBACK STATEMENTS

- Initiate a transaction with the first DML command to follow a COMMIT or ROLLBACK.
- Use COMMIT and ROLLBACK SQL statements to terminate a transaction explicitly.

CONTROLLING PL/SQL FLOW OF EXECUTION

- You can change the logical flow of statements using conditional IF statements and loop control structures.
- Conditional IF statements:
  - IF-THEN-END IF
  - IF-THEN-ELSE-END IF
  - IF-THEN-ELSIF-END IF



IF STATEMENTS

Syntax

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Simple IF statement:

Set the manager ID to 22 if the employee name is King.

```
IF v_ename = 'KING' THEN
  v_mgr := 22;
END IF;
```

SIMPLE IF STATEMENTS

- Set the job title to Salesman, the department number to 35, and the commission to 20% of the current salary if the last name is Miller.

```
. . .
IF v_ename = 'MILLER' THEN
  v_job := 'SALESMAN';
  v_deptno := 35;
  v_new_comm := sal * 0.20;
END IF;
. . .
```

IF-THEN-ELSE STATEMENTS

- Set a flag for orders where there are fewer than five days between order date and ship date.
- Example

```
...
IF v_shipdate - v_orderdate < 5 THEN
  v_ship_flag := 'Acceptable';
ELSE
  v_ship_flag := 'Unacceptable';
END IF;
...
```

IF-THEN-ELSIF STATEMENTS

- For a given value, calculate a percentage of that value based on a condition.

```
. . .
IF v_start > 100 THEN
  v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
  v_start := .5 * v_start;
ELSE
  v_start := .1 * v_start;
END IF;
. . .
```

BUILDING LOGICAL CONDITIONS

- You can handle null values with the IS NULL operator.
- Any arithmetic expression containing a null value evaluates to NULL.
- Concatenated expressions with null values treat null values as an empty string.

LOGIC TABLES

- Build a simple Boolean condition with a comparison operator.

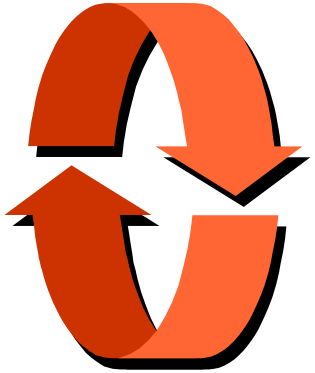
AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

ITERATIVE CONTROL: LOOP STATEMENTS

- Loops repeat a statement or sequence of statements multiple times.
- There are three loop types:
  - Basic loop
  - FOR loop
  - WHILE loop



BASIC LOOP

```
LOOP                                    -- delimiter
  statement1;                           -- statements
  . . .
  EXIT [WHEN condition];                -- EXIT statement
END LOOP;                               -- delimiter
```

where: *condition* is a Boolean variable or expression (TRUE, FALSE, or NULL);

BASIC LOOP

```
DECLARE
  v_ordid     item.ordid%TYPE := 601;
  v_counter   NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

FOR LOOP

```
FOR counter in [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- Use a FOR loop to shortcut the test for the number of iterations.
- Do not declare the index; it is declared implicitly.

FOR LOOP

- Insert the first 10 new line items for order number 601.

```
DECLARE
  v_ordid     item.ordid%TYPE := 601;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, i);
  END LOOP;
END;
```

FOR LOOP

```
DECLARE
  x NUMBER := 100;
BEGIN
  FOR i IN 1..10 LOOP
    IF MOD(i,2) = 0 THEN      -- i is even
      INSERT INTO temp VALUES (i, x, 'i is even');
    ELSE
      INSERT INTO temp VALUES (i, x, 'i is odd');
    END IF;
    x := x + 100;
  END LOOP;
  COMMIT;
END;
```

OUTPUT

```
SQL> SELECT * FROM temp ORDER BY col1;

NUM_COL1  NUM_COL2  CHAR_COL
--------- --------  ---------
       1       100  i is odd
       2       200  i is even
       3       300  i is odd
       4       400  i is even
       5       500  i is odd
       6       600  i is even
       7       700  i is odd
       8       800  i is even
       9       900  i is odd
      10      1000  i is even
```

WHILE LOOP

- Syntax

```
WHILE condition LOOP
   statement1;
   statement2;
   . . .
END LOOP;
```

Condition is evaluated at the beginning of each iteration.

Use the WHILE loop to repeat statements while a condition is TRUE.

- You would use a WHILE LOOP statement when you are unsure of how many times you want the loop body to execute.
- Since the WHILE condition is evaluated before entering the loop, it is possible that the loop body may not execute even once.

WHILE LOOP

- Let's look at a WHILE LOOP example in Oracle:

```
WHILE monthly_value <= 4000
LOOP
    monthly_value := daily_value * 31;
END LOOP;
```

In this WHILE LOOP example, the loop would terminate once the monthly_value exceeded 4000 as specified by:

WHILE monthly_value <= 4000

The WHILE LOOP will continue while monthly_value <= 4000. And once monthly_value is > 4000, the loop will terminate.

NESTED LOOPS AND LABELS

- Nest loops to multiple levels.
- Use labels to distinguish between blocks and loops.
- Exit the outer loop with the EXIT statement referencing the label.

NESTED LOOPS AND LABELS

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
```

NESTED LOOPS IN NESTED BLOCKS

```
DECLARE
  x NUMBER := 0;
  counter NUMBER := 0;
BEGIN
  FOR i IN 1..3 LOOP
    x := x + 1000;
    counter := counter + 1;
    INSERT INTO temp VALUES (x, counter, 'in OUTER loop');
    /* start an inner block */
    DECLARE
      x NUMBER := 0;  -- this is a local version of x
    BEGIN
      FOR i IN 1..4 LOOP
        x := x + 1;  -- this increments the local x
        counter := counter + 1;
        INSERT INTO temp VALUES (x, counter, 'inner loop');
      END LOOP;
    END;
  END LOOP;
  COMMIT;
END;
```

Output Table

SQL> SELECT * FROM temp ORDER BY col2;

X	counter	CHAR_COL
1000	1	in OUTER loop
1	2	inner loop
2	3	inner loop
3	4	inner loop
4	5	inner loop
2000	6	in OUTER loop
1	7	inner loop
2	8	inner loop
3	9	inner loop
4	10	inner loop
3000	11	in OUTER loop
1	12	inner loop
2	13	inner loop
3	14	inner loop
4	15	inner loop

COMPOSITE DATATYPES

- Types:
  - PL/SQL RECORDS
  - PL/SQL TABLES
- Contain internal components
- Are reusable

PL/SQL RECORDS

- Must contain one or more components of any scalar, RECORD, or PL/SQL TABLE datatype, called fields
- Are similar in structure to records in a 3GL
- Are not the same as rows in a database table
- Treat a collection of fields as a logical unit
- Are convenient for fetching a row of data from a table for processing

CREATING A PL/SQL RECORD

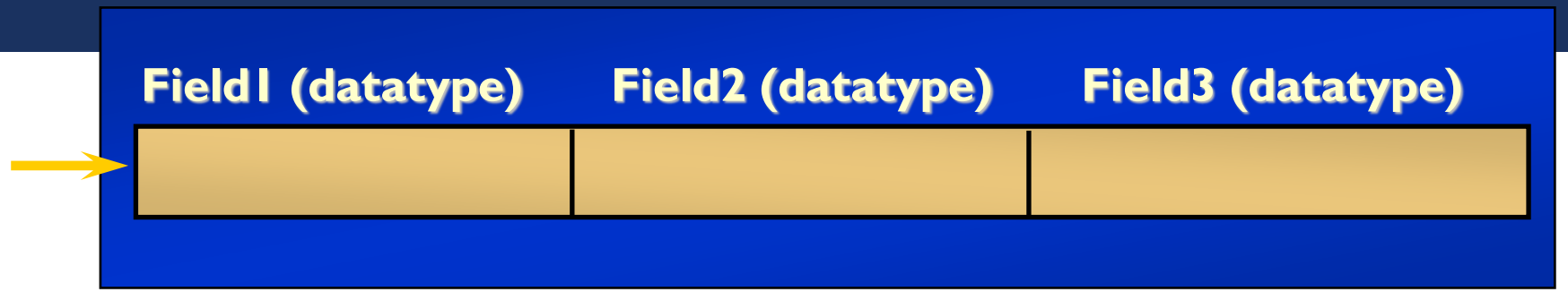
- Syntax

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

- Where *field_declaration* is

```
field_name {field_type | variable%TYPE
    | table.column%TYPE | table%ROWTYPE}
    [[NOT NULL] {:= | DEFAULT} expr]
```

PL/SQL RECORD STRUCTURE



Example

THE %ROWTYPE ATTRIBUTE

- Declare a variable according to a collection of columns in a database table or view.
- Prefix %ROWTYPE with the database table.
- Fields in the record take their names and datatypes from the columns of the table or view.

THE %ROWTYPE ATTRIBUTE

- Examples
- Declare a variable to store the same information about a department as it is stored in the DEPT table.

```
dept_record     dept%ROWTYPE;
```

- Declare a variable to store the same information about an employee as it is stored in the EMP table.

```
emp_record      emp%ROWTYPE;
```