Database Systems

PL/ SQL

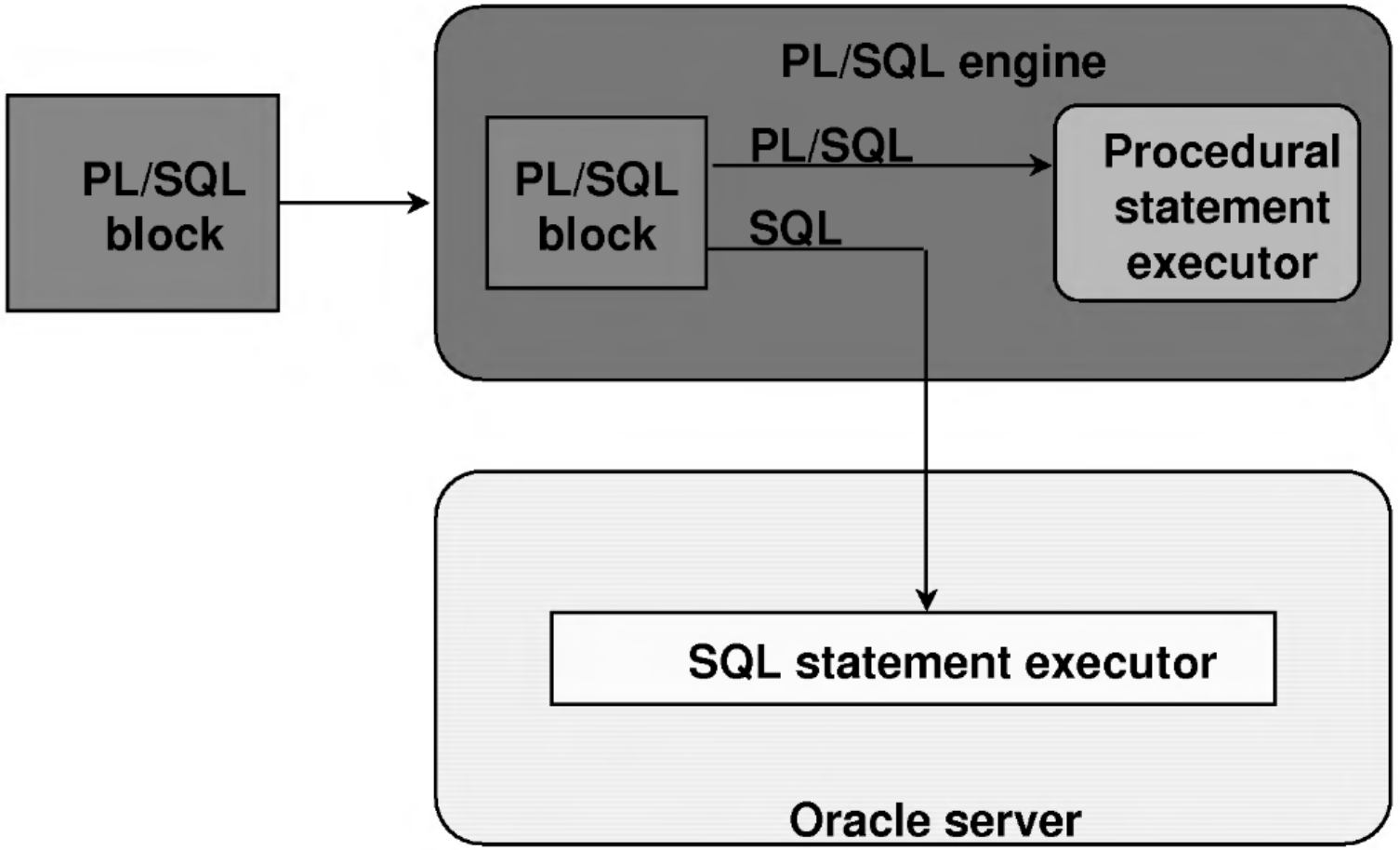
PL/SQL by Ivan Bayross

About 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.

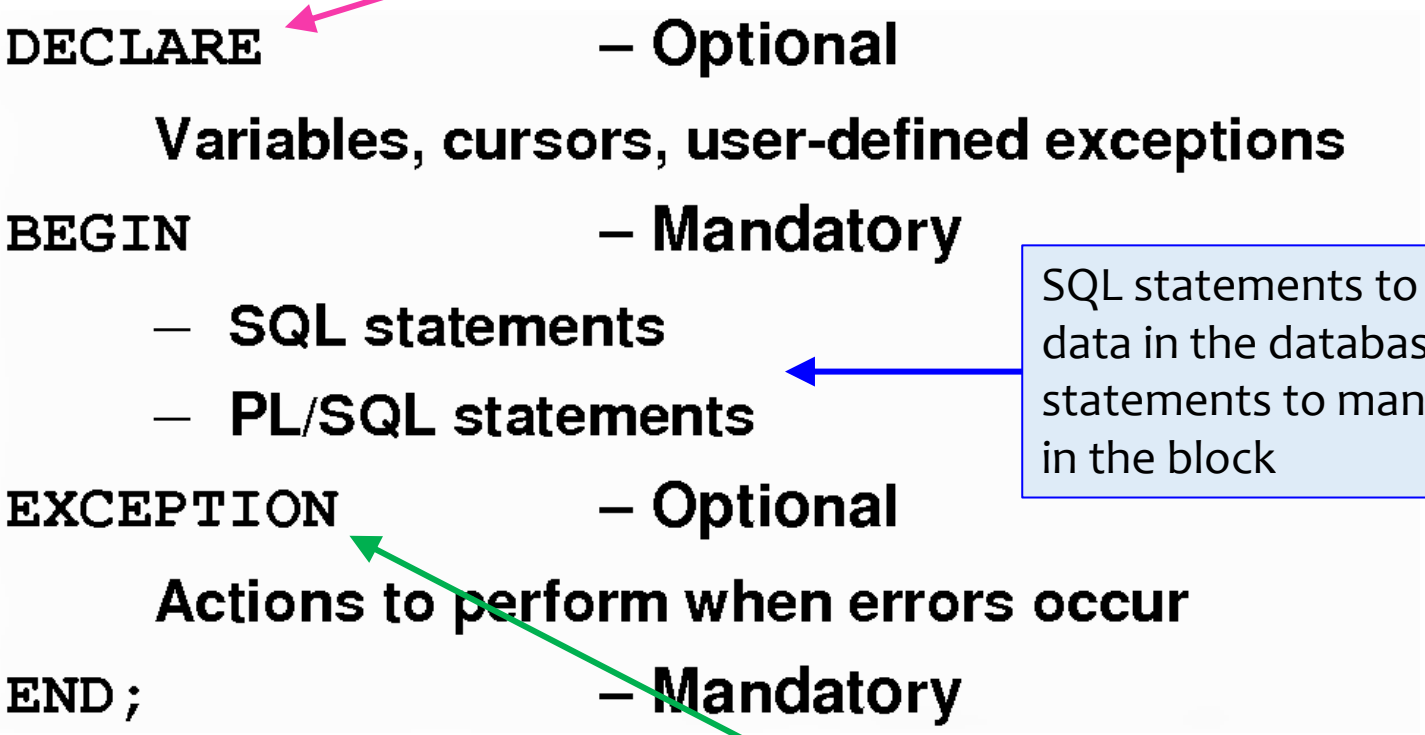
PL/SQL Environment



Key Points

- PL/SQL is an extension to SQL.
- Blocks of PL/SQL code are passed to and processed by a PL/SQL engine.
- Benefits of PL/SQL:
  - Integration
  - Improved performance
  - Portability
  - Modularity of program development
- Subprograms are named PL/SQL blocks
  - Declared as either procedures or functions.
  - Invoke subprograms from different environments.

PL/SQL Block Structure



Contains all variables, constants, cursors, and user- defined exceptions that are referenced in the executable and declarative sections

SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block

The actions to perform when errors and abnormal conditions arise in the executable section

Executing Statements and PL/SQL Blocks

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

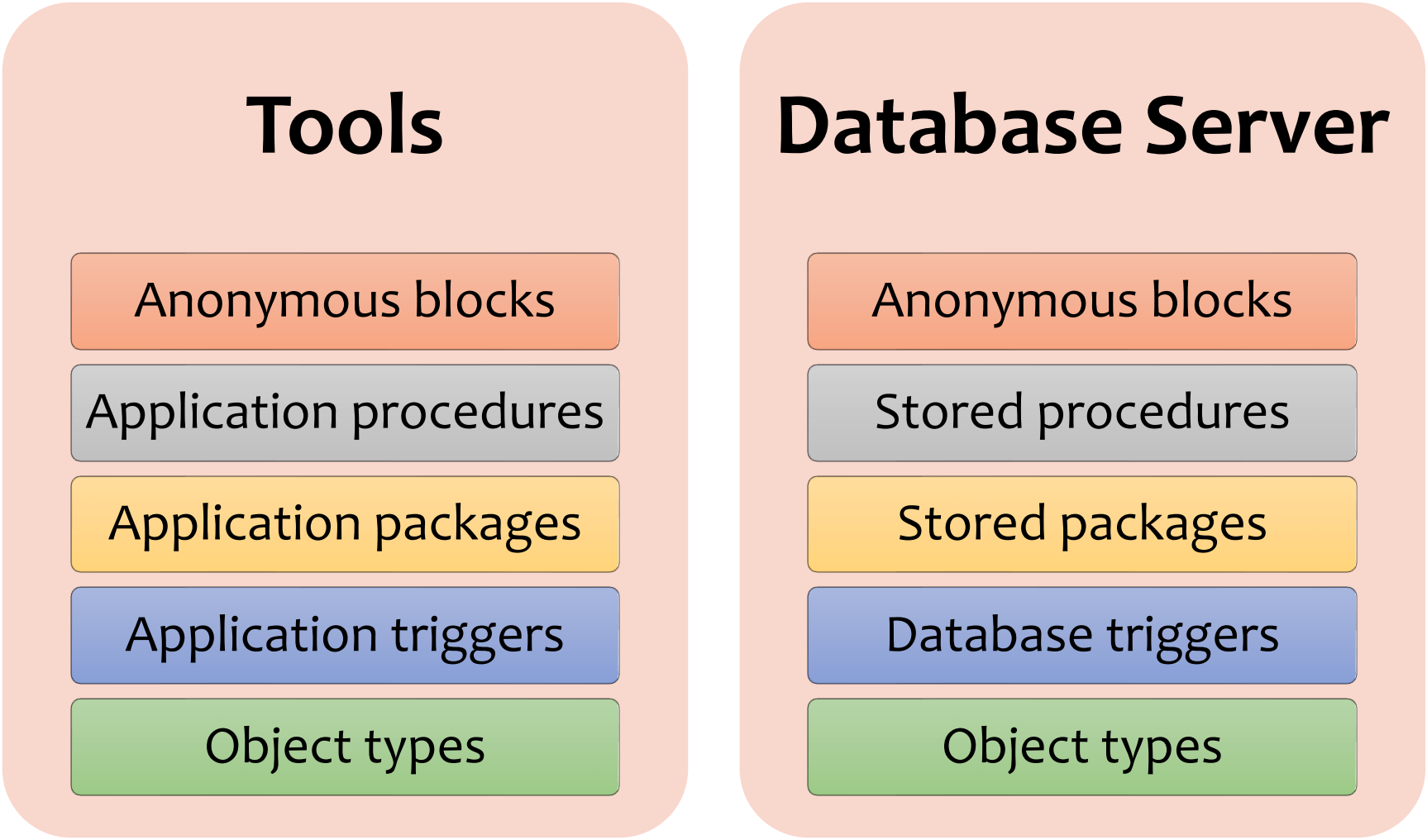
Block Types

```
[DECLARE]


BEGIN
   --statements

[EXCEPTION]

END;
```

```
PROCEDURE name
IS

BEGIN
   --statements

[EXCEPTION]

END;
```

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

END;
```

Program Constructs



Use of Variables

- Temporary storage of data
- Manipulation of stored values
- Reusability
- Ease of maintenance

- 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.

More on variables

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

- PL/SQL does not have input or output capability of its own.
- Reference substitution variables within a PL/SQL block with a preceding ampersand.
- iSQL*Plus host (or "bind") variables can be used to pass run time values out of the PL/SQL block back to the iSQL*Plus environment.

Declaring PL/SQL Variables

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

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

Variable Initialization and Keywords

- Assignment operator (: =)
- default keyword
- NOT NULL constraint

```
identifier := expr;
```

```
v_hiredate := '01-JAN-2001';
```

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

If there is a single quotation mark in the string, use a single quotation mark twice

Scalar Data Type

- CHAR [(maximum_length)]
- VARCHAR2 (maxlmum_length)
- LONG
- LONG RAW
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

How to Declare The Variable

```
DECLARE
  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;
  ...
```

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

Syntax:

identifier Table.column_name%TYPE;

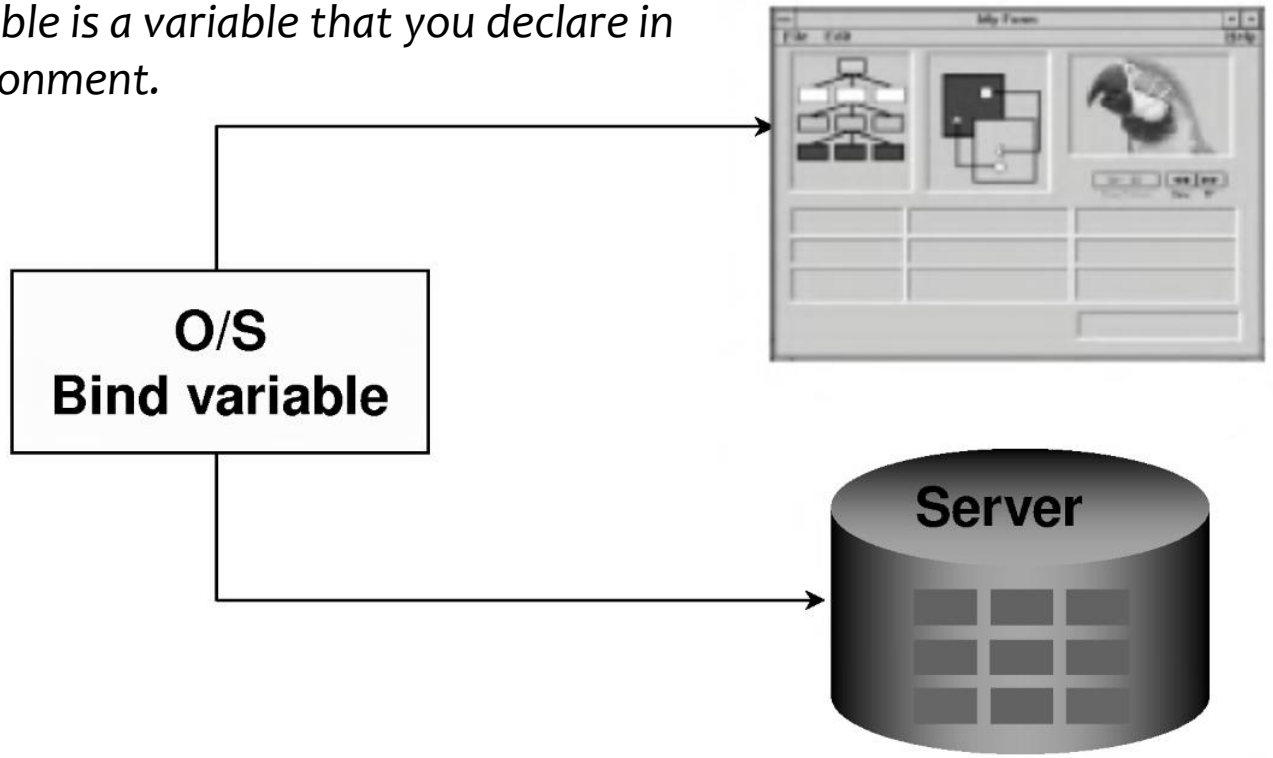
```
...
  v_name              employees.last_name%TYPE;
  v_balance           NUMBER(7,2);
  v_min_balance       v_balance%TYPE := 10;
...
```

Declaring 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.

Bind Variables

A bind variable is a variable that you declare in a host environment.



VARIABLE return_code NUMBER
VARIABLE return_msg VARCHAR2(30)

These variables should be preceded by a colon. E.g. **: return_code**

DBMS_OUTPUT.PUT_LINE

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

```
SET SERVEROUTPUT ON
declare
v_sal number(9,2) := 6000;
BEGIN
v_sal := v_sal/12;
DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' || TO_CHAR(v_sal));
END;
/
```

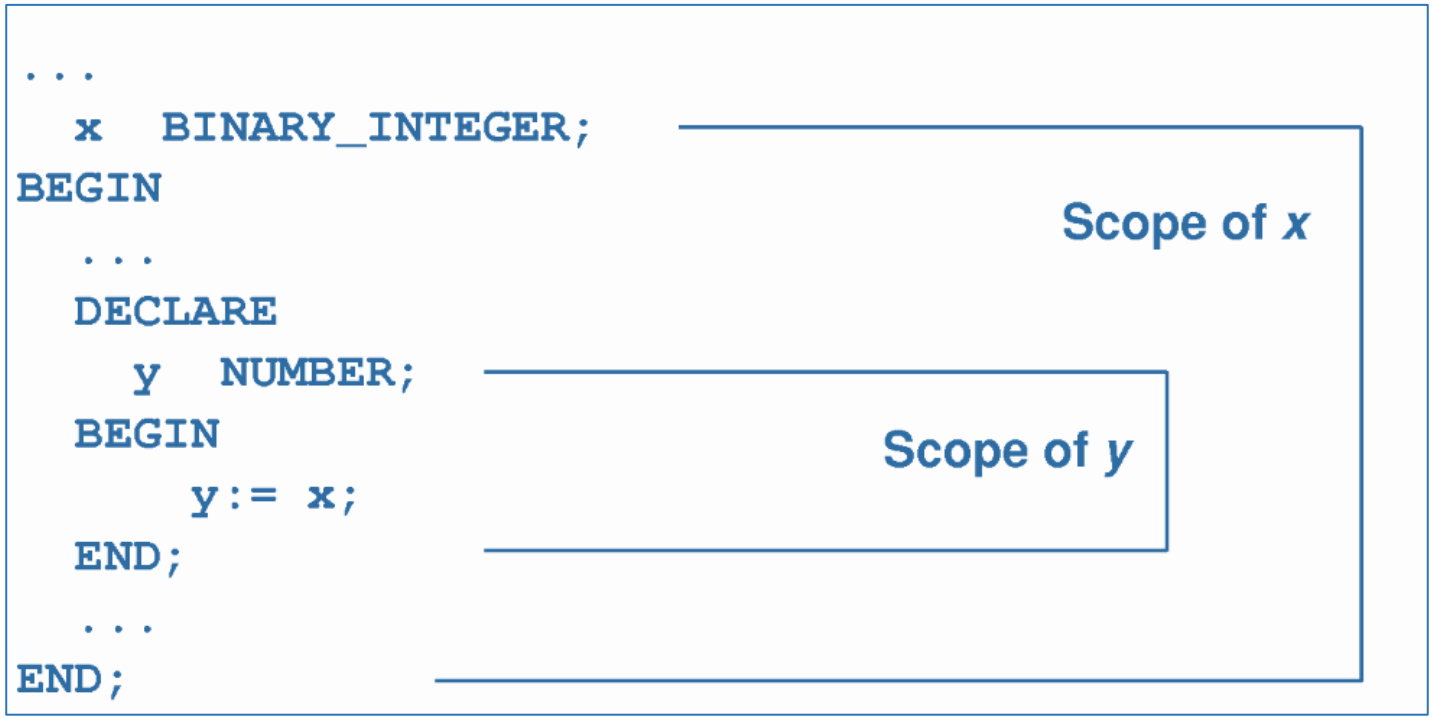
Operators

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

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

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

Nested Blocks and Variable Scope



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

Exercise

```
<<outer>>
DECLARE
  V_SAL          NUMBER(7,2)  := 60000;
  V_COMM         NUMBER(7,2)  := V_SAL * .20;
  V_MESSAGE      VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
        V_SAL            NUMBER(7,2)  := 50000;
        V_COMM           NUMBER(7,2)  := 0;
        V_TOTAL_COMP     NUMBER(7,2)  := V_SAL + V_COMM;
  BEGIN
        V_MESSAGE := 'CLERK not'||V_MESSAGE;
        outer.V_COMM := V_SAL *.30
  END;
    V_MESSAGE := 'SALESMAN'||V_MESSAGE;
END;
```

1 → V_MESSAGE, V_COMM, OUTER.V_COMM ??

2 → V_TOTAL_COMP, V_COMM, V_MESSAGE ??

Order of Operations

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

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.

select Statements in PL/SQL

- Retrieve data from the database with a select statement.

This is mandatory

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

Queries must return one and only one row.

1. Specify the **same number of variables** in the **INTO clause** as database columns in the SELECT clause.

2. Be sure that **they correspond positionally** and that their data types are compatible.

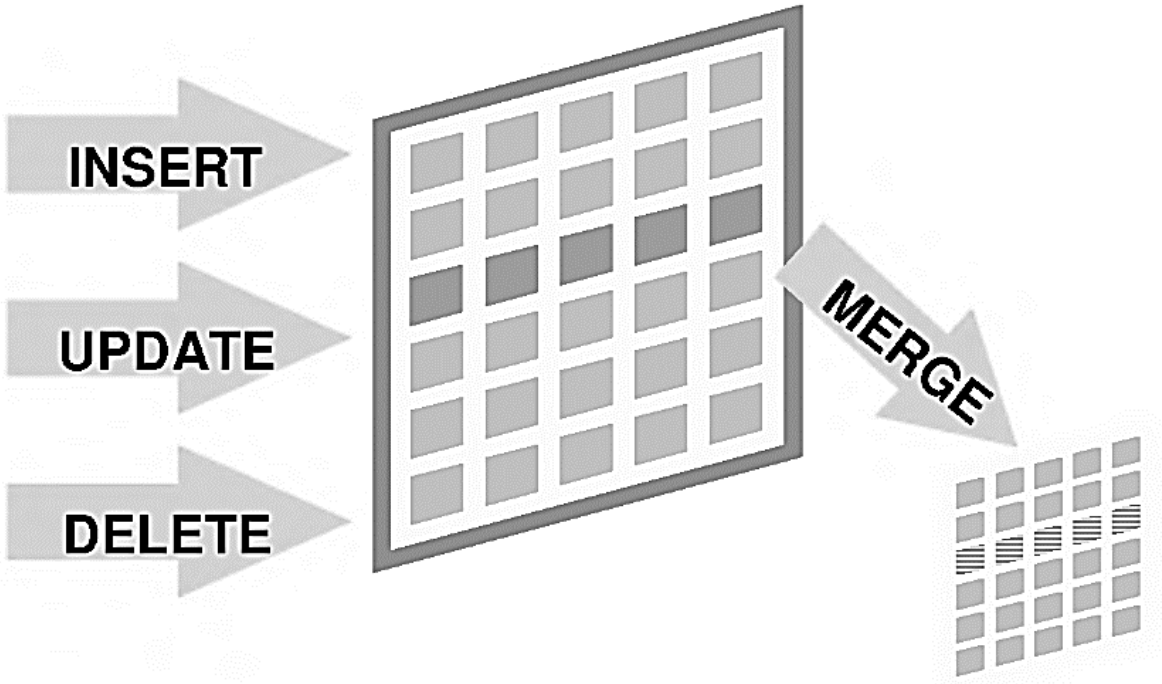
3. Use group functions, such as SUM, in a SQL statement, because group functions apply to groups of rows in a table.

Naming Conventions

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

Manipulating Data Using PL/SQL

- Make changes to database tables by using DML commands:



Inserting Data

- Add new employee information to the employees table.

```
BEGIN
        INSERT INTO employees
        (employee_id, first_name, last_name, email, hire_date, job_id, salary)
        VALUES
        (employees_seq.NEXTVAL, 'Ruth', 'Cores', 'RCORES', sysdate, 'AD_ASST', 4000);
END;
```

Updating Data

- Increase the salary of all employees who are stock clerks.

```
DECLARE
        v_sal_increase      employees.salary%TYPE := 800;
BEGIN
        UPDATE employees
        SET salary = salary + v_sal_increase
        WHERE job_id = 'ST_CLERK';
END;
```

PL/SQL variable assignments always use : =, and SQL column assignments always use =.

If no rows are modified, no error occurs, unlike the SELECT statement in PL/SQL.

Deleting Data

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

```
DECLARE
        v_deptno employees.department_id%TYPE :
BEGIN
        DELETE FROM employees
        WHERE department_id = v_deptno;
END;
```

Merging Rows

- **Inserts or updates rows** in **one table**, using data **from another table**.
- Each row is inserted or updated in the target table, depending upon an **equijoin condition**.
- Example

Example

```
DECLARE
        v_empno EMPLOYEES.EMPLOYEE_ID%TYPE := 100;
BEGIN
        MERGE INTO copy_emp c
                USING employees e
                ON (c.employee_id = v_empno)
        WHEN MATCHED THEN
                UPDATE SET
                        c.first_name = e.first_name,
                        c.last_name = e.last_name,
                        c.email = e.email,
                        c.phone_number = e.phone_number,
                        c.hire_date = e.hire_date,
                        c.job_id = e.job_id,
                        c.salary = e.salary,
                        c.commission_pct = e.commission_pct,
                        c.manager_id = e.manager_id,
                        c.department_id = e.department_id
        WHEN NOT MATCHED THEN
                INSERT VALUES(e.employee_id,
                e.first_name, e.last_name,  e.email,
                e.phone_number, e.hire_date, e.job_id,  e.salary,
                e.commission_pct, e.manager_id,  e.department_id);
END;
```

Naming Convensions

Identifier	Naming Convention	Example
Variable	v_name	v_sal
Constant	c_name	c_company_name
Cursor	name_cursor	emp_cursor
Exception	e_name	e_too_many
Table type	name_table_type	amount_table_type
Table	name_table	countries
Record type	name_record_type	emp_record_type
Record	name_record	customer_record
*i*SQL*Plus substitution variable (also referred to as substitution parameter)	p_name	p_sal
*i*SQL*Plus host or bind variable	g_name	g_year_sal

SQL Cursor

- A cursor is a private SQL work area.
- There are two types of cursors:
  - Implicit cursors
  - Explicit cursors
- The Oracle server uses implicit cursors to parse and execute your SQL statements.
- Explicit cursors are explicitly declared by the programmer.

SQL Cursor Attributes

- Using SQL cursor attributes, you can test the outcome of your SQL statements.

SQL%ROWCOUNT	Number of rows affected by the most recent SQL statement (an integer value)
SQL%FOUND	Boolean attribute that evaluates to true if the most recent SQL statement affects one or more rows
SQL%NOTFOUND	Boolean attribute that evaluates to true if the most recent SQL statement does not affect any rows
SQL%ISOPEN	Always evaluates to false because PL/SQL closes implicit cursors immediately after they are executed

SQL Cursor Attributes

- Delete rows that have the specified employee ID from the employees table. Print the number of rows deleted.

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
        v_employee_id employees.employee_id%TYPE
BEGIN
        DELETE FROM employees
        WHERE employee_id = v_employee_id;
        :rows_deleted := (SQL%ROWCOUNT || ' row deleted.');
END;
/
PRINT rows_deleted
```

Controlling PL/SQL Flow of Execution

- Change the logical execution of statements using
  - conditional if statements and
    - IF-THEN-END IF
    - IF-THEN-ELSE-END IF
    - IF-THEN-ELSIF-END IF
  - CASE
  - Loop control structures.
    - Basic loop
    - for loop
    - WHILE lOOp

if Statements

- Syntax

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

- When possible, use the ELS IF clause instead of nesting IF statements.
- If the action in the ELSE clause consists purely of another IF statement, it is more convenient to use the ELS IF clause.
- Any arithmetic expression containing null values evaluates to null.

case Expressions

- A case expression selects a result and returns it.
- To select the result, the case expression uses an expression whose value is used to select one of several alternatives.

```
CASE selector
        WHEN expression 1 THEN result 1
        WHEN expression 2 THEN result 2
        …
        WHEN expression N THEN result N
        [ELSE resultN+1;]
END;
```

Example

```
SET SERVEROUTPUT ON
DEFINE p_grade = a
DECLARE
        v_grade CHAR(1) := UPPER('&p_grade' ) ;
        v_appraisal VARCHAR2(20);
BEGIN
        v_appraisal :=
                CASE v_grade
                        WHEN 'A' THEN 'Excellent'
                        WHEN 'B' THEN 'Very Good'
                        WHEN 'C' THEN 'Good'
                        ELSE 'No such grade'
                END;
        DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal ' || v_appraisal);
END;
```

Handling Nulls

- Avoid some common mistakes by keeping in mind the following rules:
  - Simple comparisons involving nulls always yield NULL.
  - Applying the logical operator not to a null yields NULL.
  - In conditional control statements,
    - if the condition yields null, its associated sequence of statements is not executed.

Logic Tables

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

- Basic loop
- for loop
- WHILE lOOp

Basic Loops

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

```
DECLARE
  v_country_id    locations.country_id%TYPE := 'CA';
  v_location_id   locations.location_id%TYPE;
  v_counter       NUMBER(2) := 1;
  v_city          locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_location_id FROM locations
  WHERE country_id = v_country_id;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + v_counter),v_city, v_country_id );
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
```

WHILE LOOPS

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

Condition is evaluated at the beginning of each iteration.

```
DECLARE
  v_country_id      locations.country_id%TYPE := 'CA';
  v_location_id     locations.location_id%TYPE;
  v_city            locations.city%TYPE := 'Montreal';
  v_counter         NUMBER  := 1;
BEGIN
  SELECT MAX(location_id) INTO v_location_id FROM locations
  WHERE country_id = v_country_id;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + v_counter), v_city, v_country_id );
    v_counter := v_counter + 1;
  END LOOP;
END;
```

FOR LOOPS

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

```
DECLARE
  v_country_id    locations.country_id%TYPE := 'CA';
  v_location_id   locations.location_id%TYPE;
  v_city          locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_location_id
    FROM locations
    WHERE country_id = v_country_id;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + i), v_city, v_country_id );
  END LOOP;
END;
```

Guidelines While Using Loops

- Use the basic loop when the statements inside the loop must execute at least once.
- Use the while loop if the condition has to be evaluated at the start of each iteration.
- Use a for loop if the number of iterations is known.

Thank you