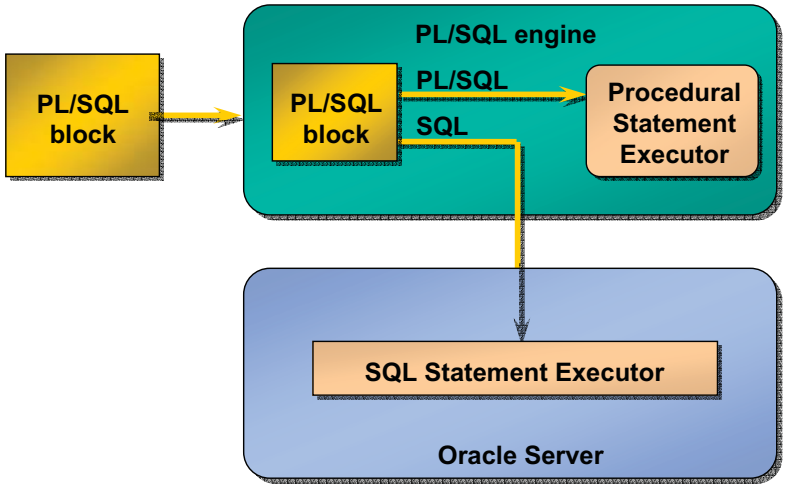
# Overview of PL/SQL

# About PL/SQL

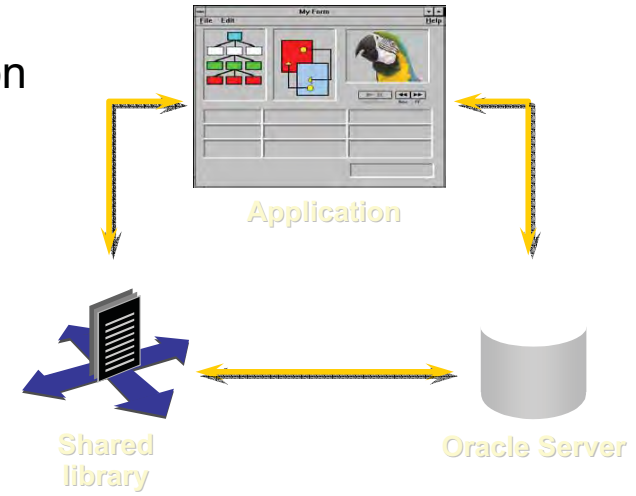
– PL/SQL is an extension to SQL with design features of programming languages.
– Data manipulation and query statements of SQL are included within procedural units of code.

# PL/SQL Environment



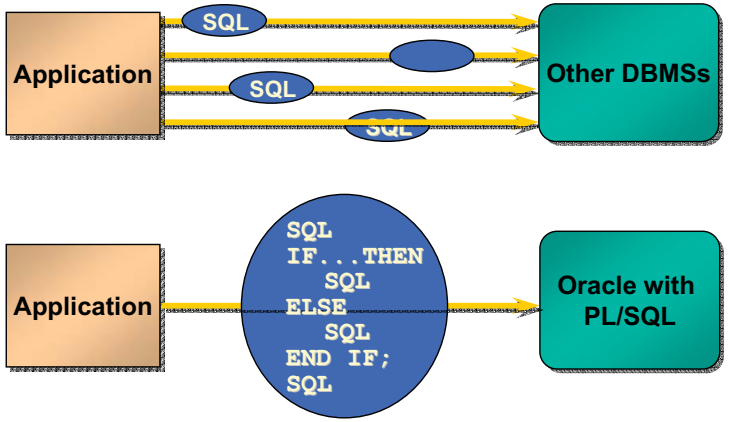

# Benefits of PL/SQL

- Integration



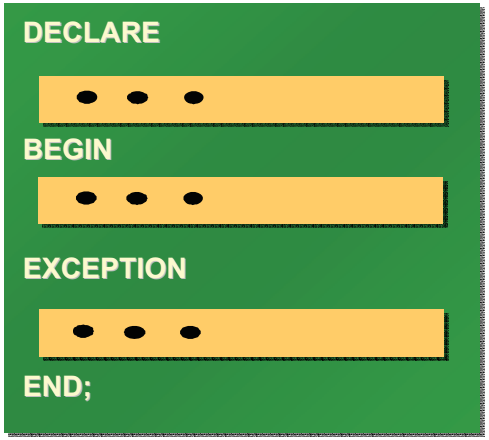
# Benefits of PL/SQL

- Improve performance




# Benefits of PL/SQL

- Modularize program development



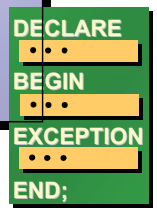

# Benefits of PL/SQL

- It is portable.
- You can declare identifiers.
- You can program with procedural language control structures.
- It can handle errors.

# Declaring Variables

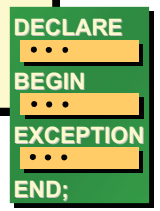
# PL/SQL Block Structure

- 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;
```

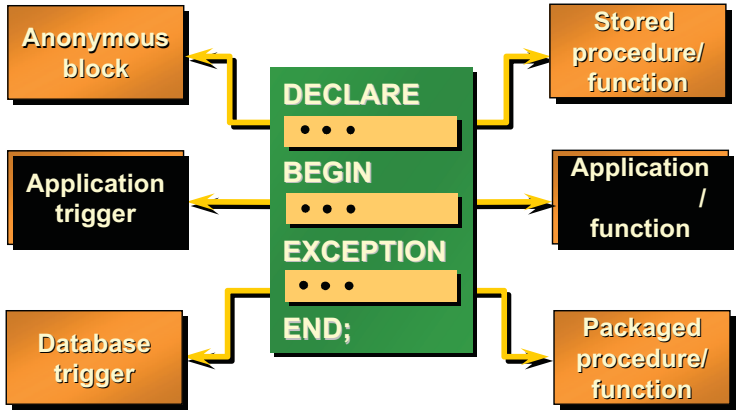
Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Function

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

END;
```

# Program Constructs



# Use of Variables

- Use variables for:
  - Temporary storage of data
  - Manipulation of stored values
  - Reusability
  - Ease of maintenance

# 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

# 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 and CONSTANT.
  - 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;
```

# 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

# Declaring Variables with the %TYPE Attribute

- Examples

```
...
  v_ename               emp.ename%TYPE;
  v_balance             NUMBER(7,2);
  v_min_balance         v_balance%TYPE := 10;
...
```

# Declaring Boolean Variables

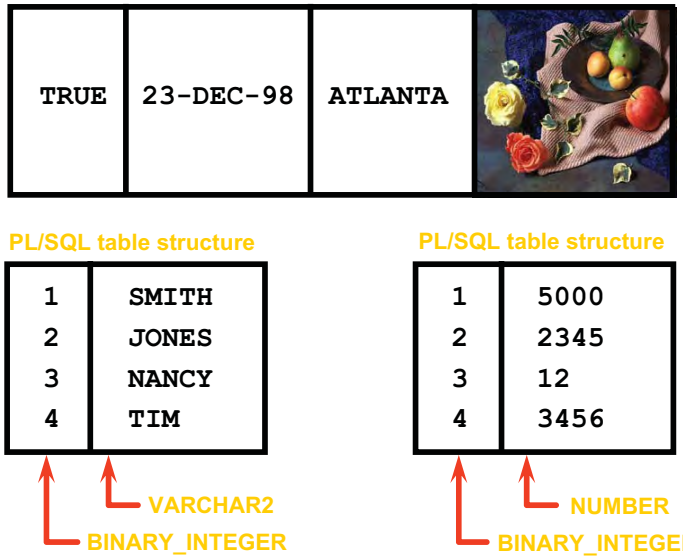
– Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.

– The variables are 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.

# PL/SQL Record Structure




# 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

# Writing Executable Statements

# Objectives

- After completing this lesson, you should be able to do the following:
  - Recognize the significance of the executable section
  - Write statements in the executable section
  - Describe the rules of nested blocks
  - Execute and test a PL/SQL block
  - Use coding conventions

# 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

# PL/SQL Block Syntax and Guidelines

– Literals
  - Character and date literals must be enclosed in single quotation marks.

```
v_ename := 'Henderson';
```

  - Numbers can be simple values or scientific notation.

– A PL/SQL block is terminated by a slash ( / ) on a line by itself.

# Commenting Code

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

– Place multi-line comments between the symbols /* and */.

- Example

```
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := &p_monthly_sal * 12;
END; -- This is the end of the block
```

# SQL Functions in PL/SQL

– Available in procedural statements:
  - Single-row number
  - Single-row character
  - Datatype conversion
  - Date

**Same as in SQL**

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

# 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);
```

# Datatype 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;
```

# Datatype Conversion

This statement produces a compilation error if the variable v_date is declared as datatype DATE.

```
v_date := 'January 13, 1998';
```

To correct the error, use the TO_DATE conversion function.

```
v_date := TO_DATE ('January 13, 1998',
                   'Month DD, YYYY');
```

# Nested Blocks and Variable Scope

– Statements 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 object is the region of the program that can refer to the object.

# Nested Blocks and Variable Scope

• An identifier is visible in the regions in which you can reference the unqualified identifier:
  – A block can look up to the enclosing block.
  – A block cannot look down to enclosed blocks.

# Nested Blocks and Variable Scope

Example

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

Scope of *x*

Scope of *y*

# Operators in PL/SQL

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

**Same as in SQL**

# Operators in PL/SQL

- Examples
  - Increment the counter for a loop.

```
v_count      := v_count + 1;
```

  - Set the value of a Boolean flag.

```
v_equal      := (v_n1 = v_n2);
```

  - Validate an employee number if it contains a value.

```
v_valid      := (v_empno IS NOT NULL);
```

# Code Naming Conventions

- Avoid ambiguity:
  - The names of local variables and formal parameters take precedence over the names of database tables.
  - The names of columns take precedence over the names of local variables.

# Interacting with the Oracle Server

# 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;
```

# SELECT Statements in PL/SQL

- The INTO clause is required.
- Example

```
DECLARE
  v_deptno    NUMBER(2);
  v_loc       VARCHAR2(15);
BEGIN
  SELECT      deptno, loc
  INTO        v_deptno, v_loc
  FROM        dept
  WHERE       dname = 'SALES';
...
END;
```

# Retrieving Data in PL/SQL

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

```
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;
```

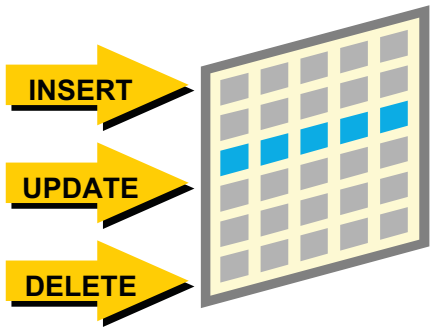
# Retrieving Data in PL/SQL

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

```
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




# Inserting Data

- Add new employee information to the emp table.
- Example

```
BEGIN
    INSERT INTO   emp(empno, ename, job, deptno)
    VALUES        (empno_sequence.NEXTVAL, 'HARDING',
                  'CLERK', 10);
END;
```

# Updating Data

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

```
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.
- Example

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

# Naming Conventions

- Use a naming convention to avoid ambiguity in the WHERE clause.
- Database columns and identifiers should have distinct names.
- Syntax errors can arise because PL/SQL checks the database first for a column in the table.

# 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.

# 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

| | |
|---|---|
| **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

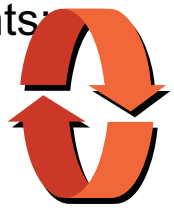
- Delete rows that have the specified order number from the ITEM table. Print the number of rows deleted.
- Example

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_ordid  NUMBER := 605;
BEGIN
  DELETE FROM  item
  WHERE        ordid = v_ordid;
  :rows_deleted := (SQL%ROWCOUNT ||
                        ' rows deleted.');
END;
/
PRINT rows_deleted
```

# Writing Control Structures

# 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 Osborne.

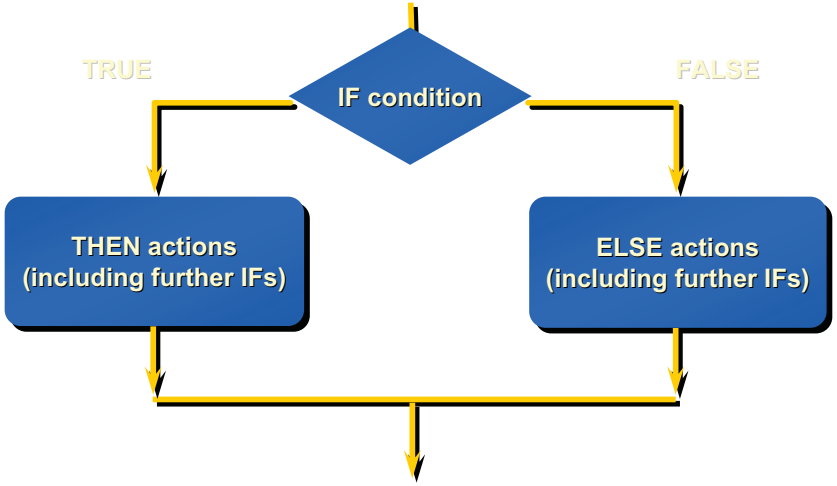
```
IF v_ename = 'OSBORNE' THEN
  v_mgr := 22;
END IF;
```

# 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.
- Example

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

# IF-THEN-ELSE Statement Execution Flow

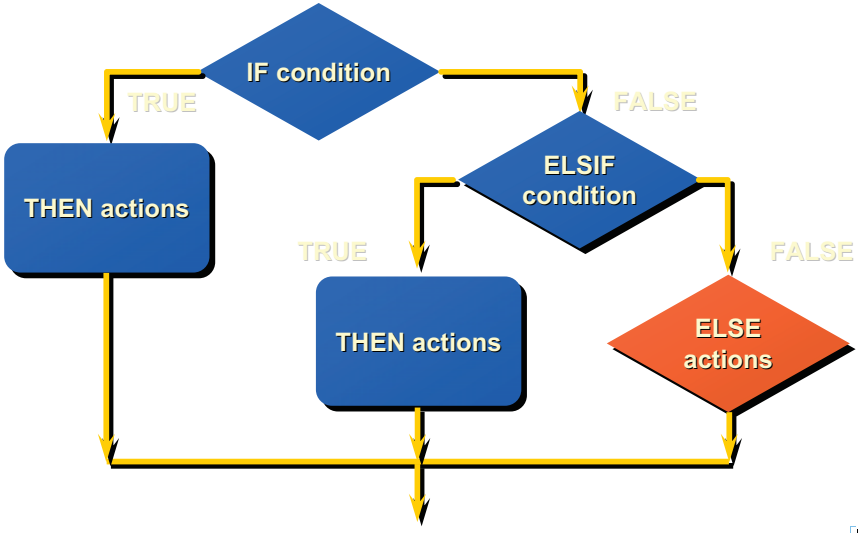



# IF-THEN-ELSE Statements

- 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 Statement Execution Flow




# IF-THEN-ELSIF Statements

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

```
. . .
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 |