Oracle9*i*: Program with PL/SQL

Electronic Presentation

40054GC11
Production 1.1
October 2001
D34010

Authors

Nagavalli Pataballa
Priya Nathan

Technical Contributors and Reviewers

Anna Atkinson
Bryan Roberts
Caroline Pereda
Cesljas Zarco
Coley William
Daniel Gabel
Dr. Christoph Burandt
Hakan Lindfors
Helen Robertson
John Hoff
Lachlan Williams
Laszlo Czinkoczki
Laura Pezzini
Linda Boldt
Marco Verbeek
Natarajan Senthil
Priya Vennapusa
Roger Abuzalaf
Ruediger Steffan
Sarah Jones
Stefan Lindblad
Susan Dee

Publisher

Sheryl Domingue

Copyright © Oracle Corporation, 1999, 2000, 2001. All rights reserved.

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of the Education Products group of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Worldwide Education Services, Oracle Corporation, 500Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

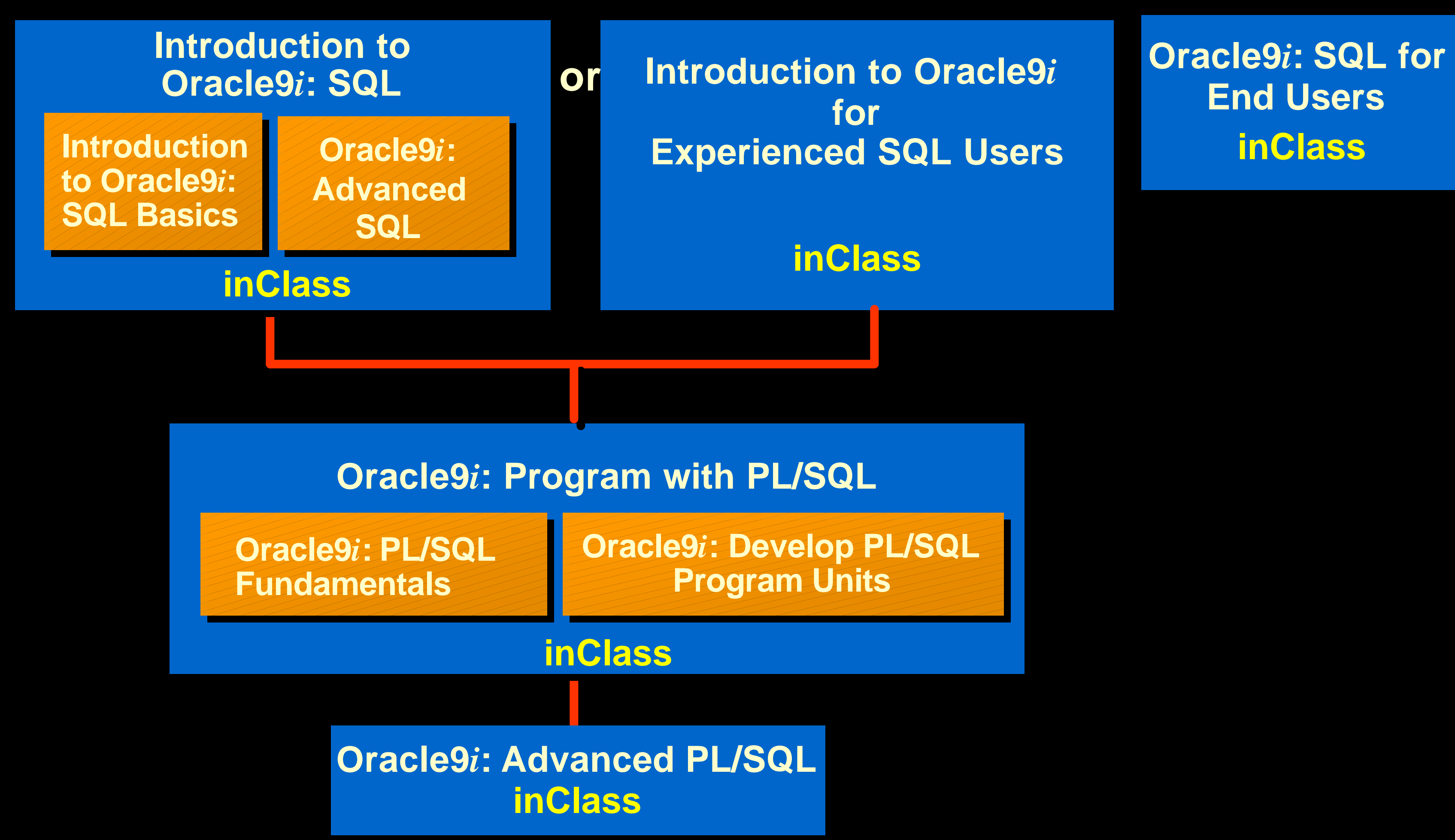
Oracle and all references to Oracle Products are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Curriculum Map

Copyright © Oracle Corporation, 2001. All rights reserved.

Languages Curriculum for Oracle9*i*



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

I

Overview of PL/SQL

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Course Objectives

After completing this course, you should be able to do the following:

- Describe the purpose of PL/SQL
- Describe the use of PL/SQL for the developer as well as the DBA
- Explain the benefits of PL/SQL
- Create, execute, and maintain procedures, functions, packages, and database triggers
- Manage PL/SQL subprograms and triggers
- Describe Oracle supplied packages
- Manipulate large objects (LOBs)

Copyright © Oracle Corporation, 2001. All rights reserved.

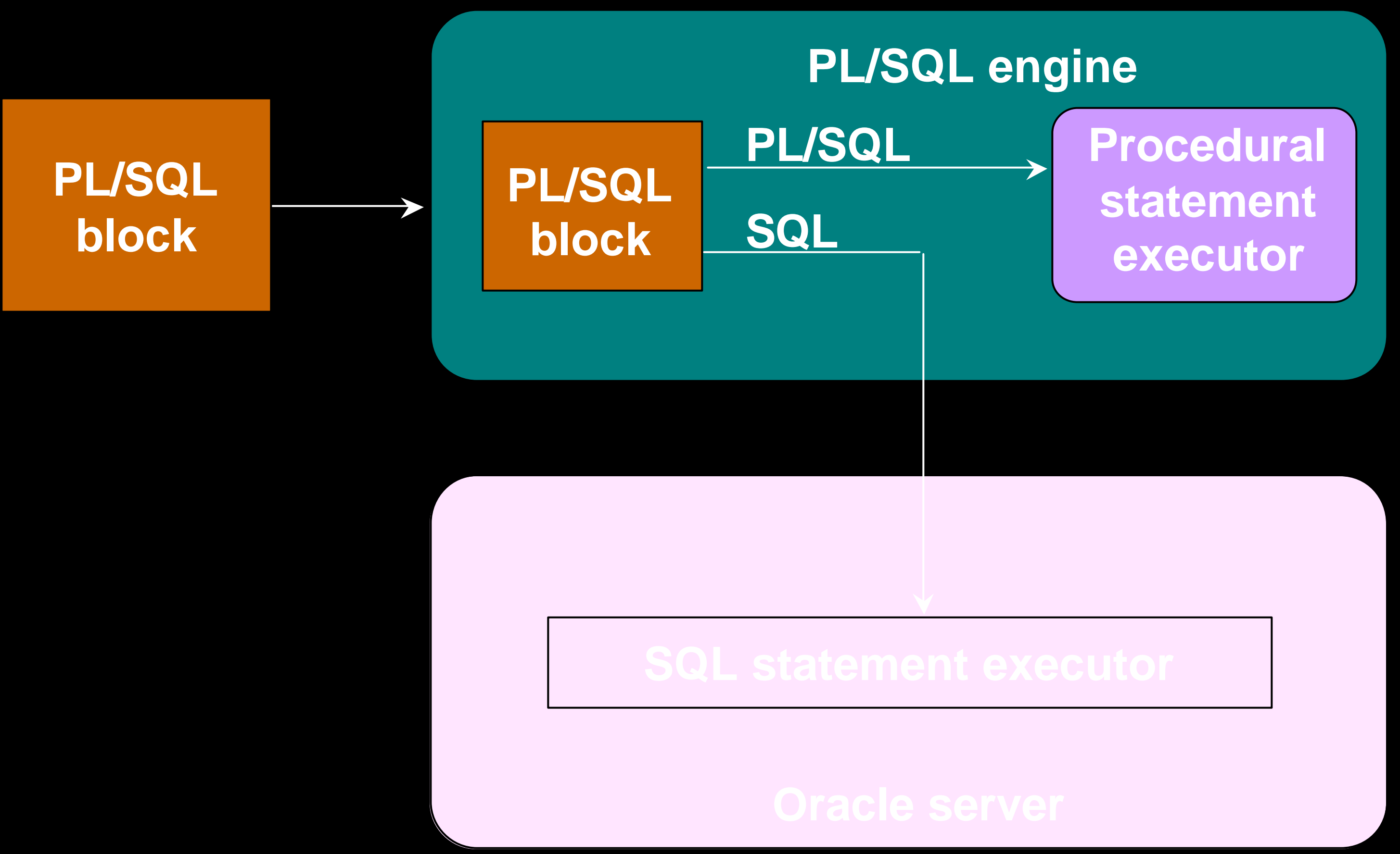
About PL/SQL

- 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.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

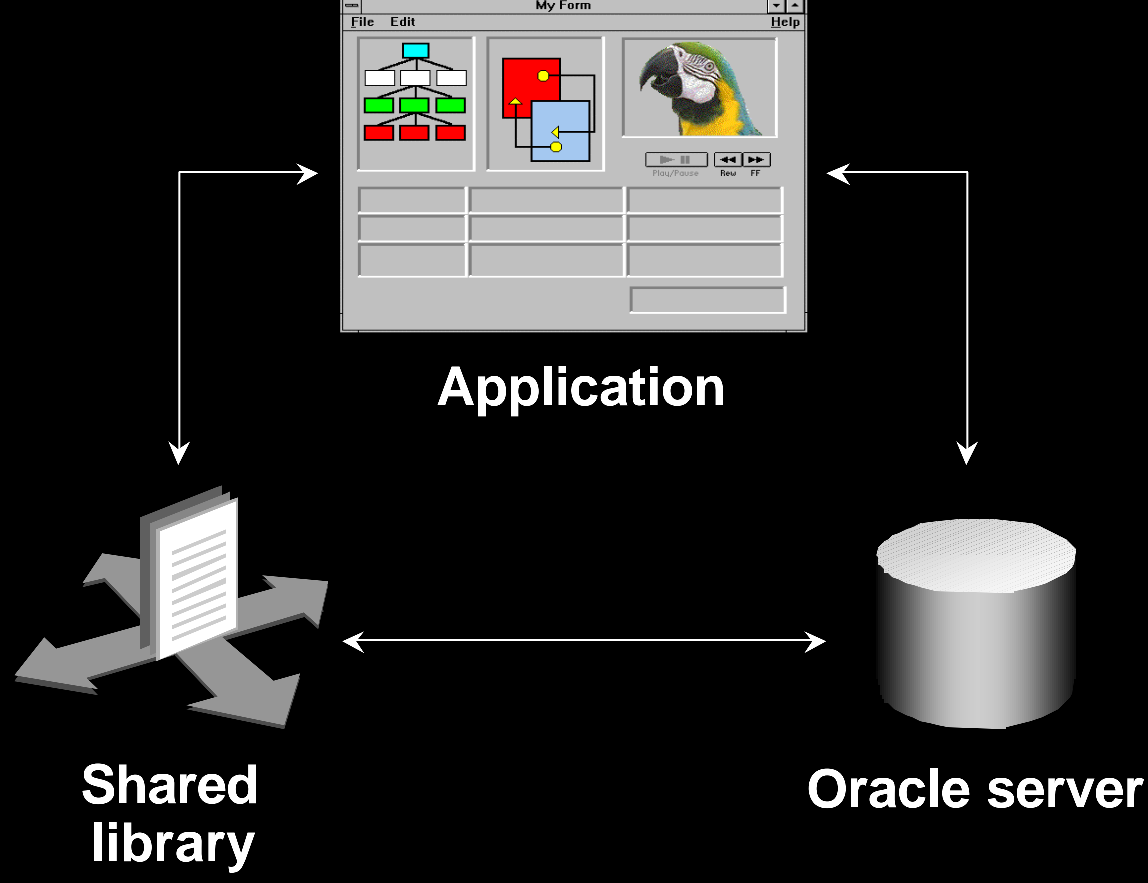
PL/SQL Environment



Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of PL/SQL

Integration

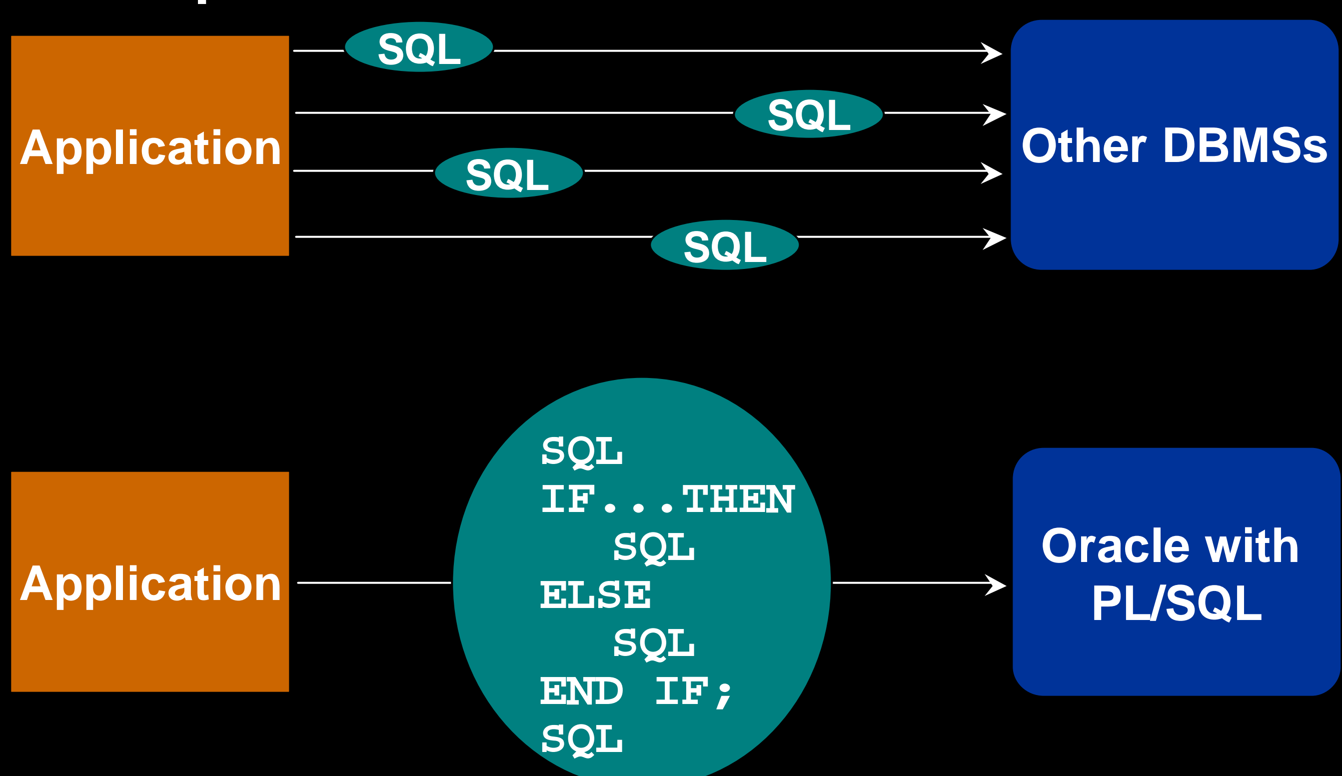


ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of PL/SQL

Improved performance



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of PL/SQL

Modularize program development

```
DECLARE
  • • •
BEGIN
  • • •
EXCEPTION
  • • •
END;
```


Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of PL/SQL

- **PL/SQL is portable.**
- **You can declare variables.**

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of PL/SQL

- You can program with procedural language control structures.
- PL/SQL can handle errors.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

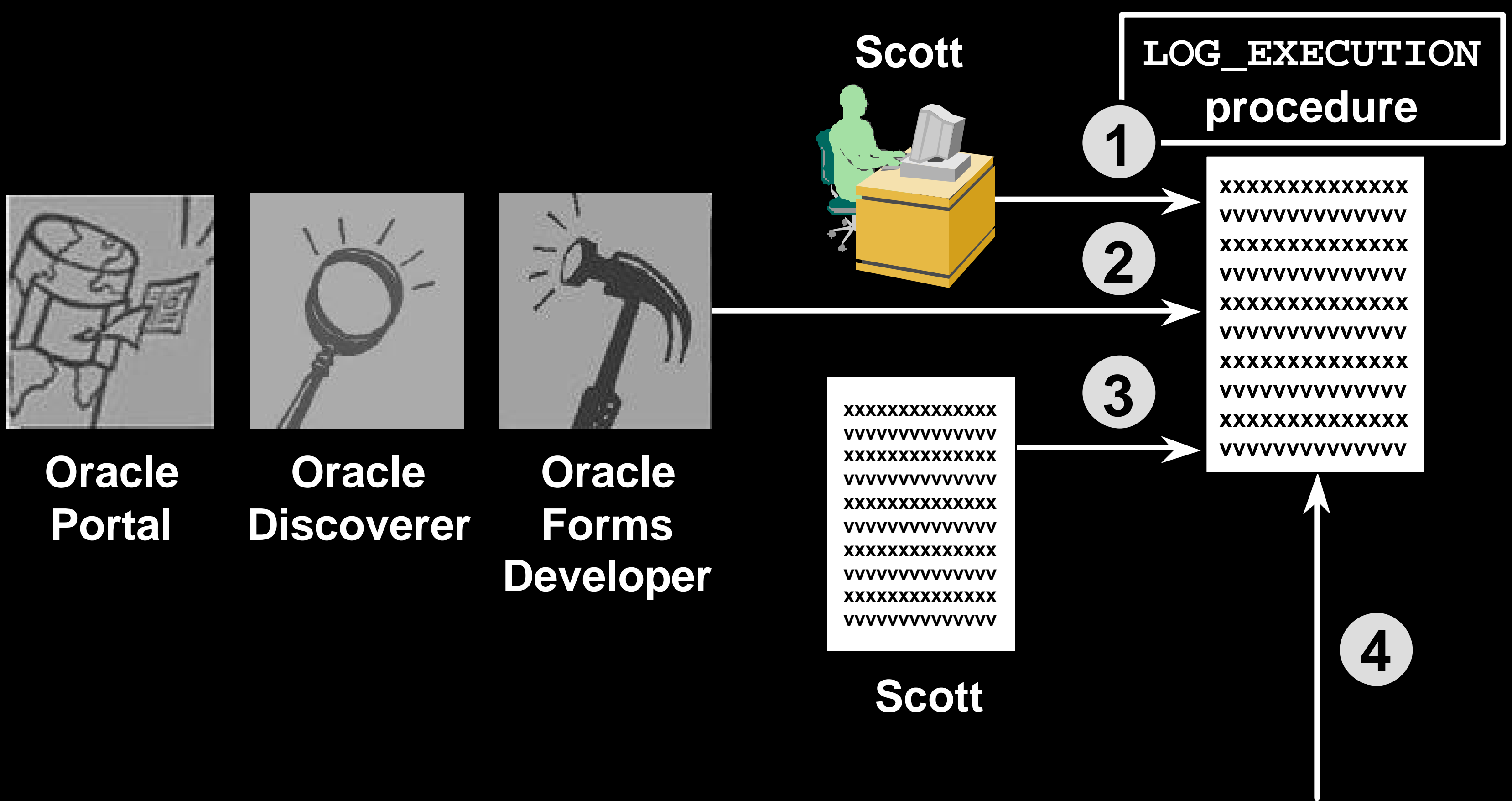
Benefits of Subprograms

- Easy maintenance
- Improved data security and integrity
- Improved performance
- Improved code clarity

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Stored Procedures and Functions



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

- 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.
- You can invoke subprograms from different environments.

Copyright © Oracle Corporation, 2001. All rights reserved.

1

Declaring Variables

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Recognize the basic PL/SQL block and its sections
- Describe the significance of variables in PL/SQL
- Declare PL/SQL variables
- Execute a PL/SQL block

Copyright © Oracle Corporation, 2001. All rights reserved.

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

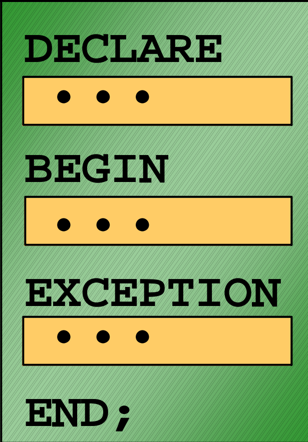
`END;` (Mandatory)

```
DECLARE
  • • •
BEGIN
  • • •
EXCEPTION
  • • •
END;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

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;
```



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Block Types

Anonymous

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Function

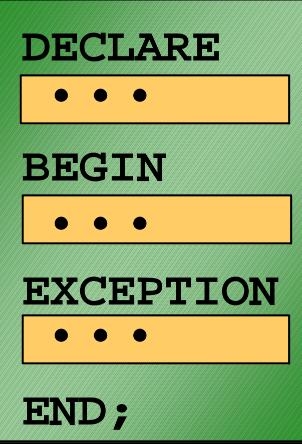
```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Program Constructs



Tools Constructs
Anonymous blocks
Application procedures or functions
Application packages
Application triggers
Object types

Database Server Constructs
Anonymous blocks
Stored procedures or functions
Stored packages
Database triggers
Object types

Copyright © Oracle Corporation, 2001. All rights reserved.

Use of Variables

Variables can be used for:

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Types of Variables

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using *i*SQL*Plus Variables Within PL/SQL Blocks

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

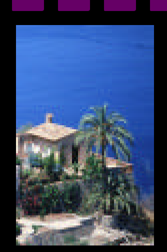
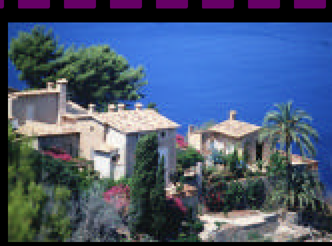
Types of Variables

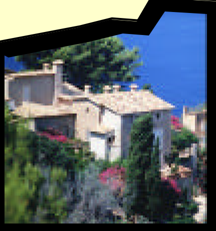
TRUE

25-JAN-01

"Four score and seven years ago our fathers brought forth upon this continent, a new nation, conceived in LIBERTY, and dedicated to the proposition that all men are created equal."

256120.08



Atlanta

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Guidelines for Declaring PL/SQL Variables

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

```
identifier := expr;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

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
  employee_id  NUMBER(6);
BEGIN
  SELECT     employee_id
  INTO       employee_id
  FROM       employees
  WHERE      last_name = 'Kochhar';
END;
/
```

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Variable Initialization and Keywords

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

Syntax:

```
identifier := expr;
```

Examples:

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

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Scalar Data Types

- Hold a single value
- Have no internal components

25-OCT-99

"Four score and seven years
ago our fathers brought
forth upon this continent, a
new nation, conceived in
LIBERTY, and dedicated to
the proposition that all men
are created equal."

TRUE

256120.08

Atlanta

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Base Scalar Data Types

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

Copyright © Oracle Corporation, 2001. All rights reserved.

Base Scalar Data Types

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

Copyright © Oracle Corporation, 2001. All rights reserved.

Scalar Variable Declarations

Examples:

```
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;
  ...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring Variables with the %TYPE Attribute

Syntax:

```
identifier     Table.column_name%TYPE;
```

Examples:

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

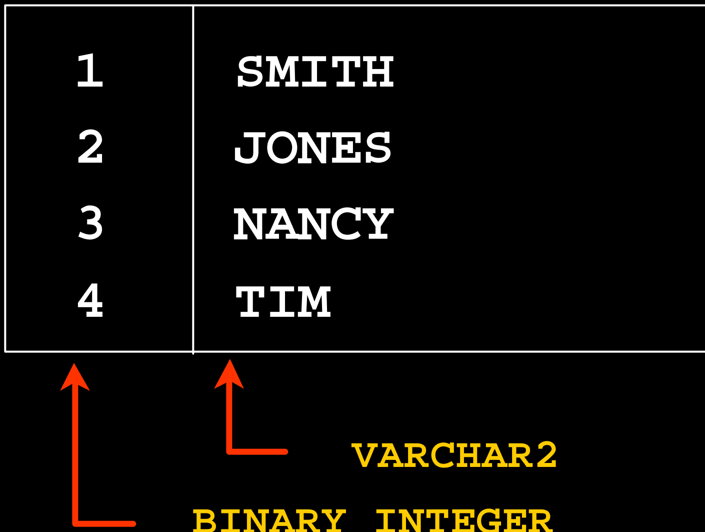
ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Composite Data Types

TRUE	23-DEC-98	ATLANTA	

PL/SQL table structure



1	SMITH
2	JONES
3	NANCY
4	TIM

PL/SQL table structure

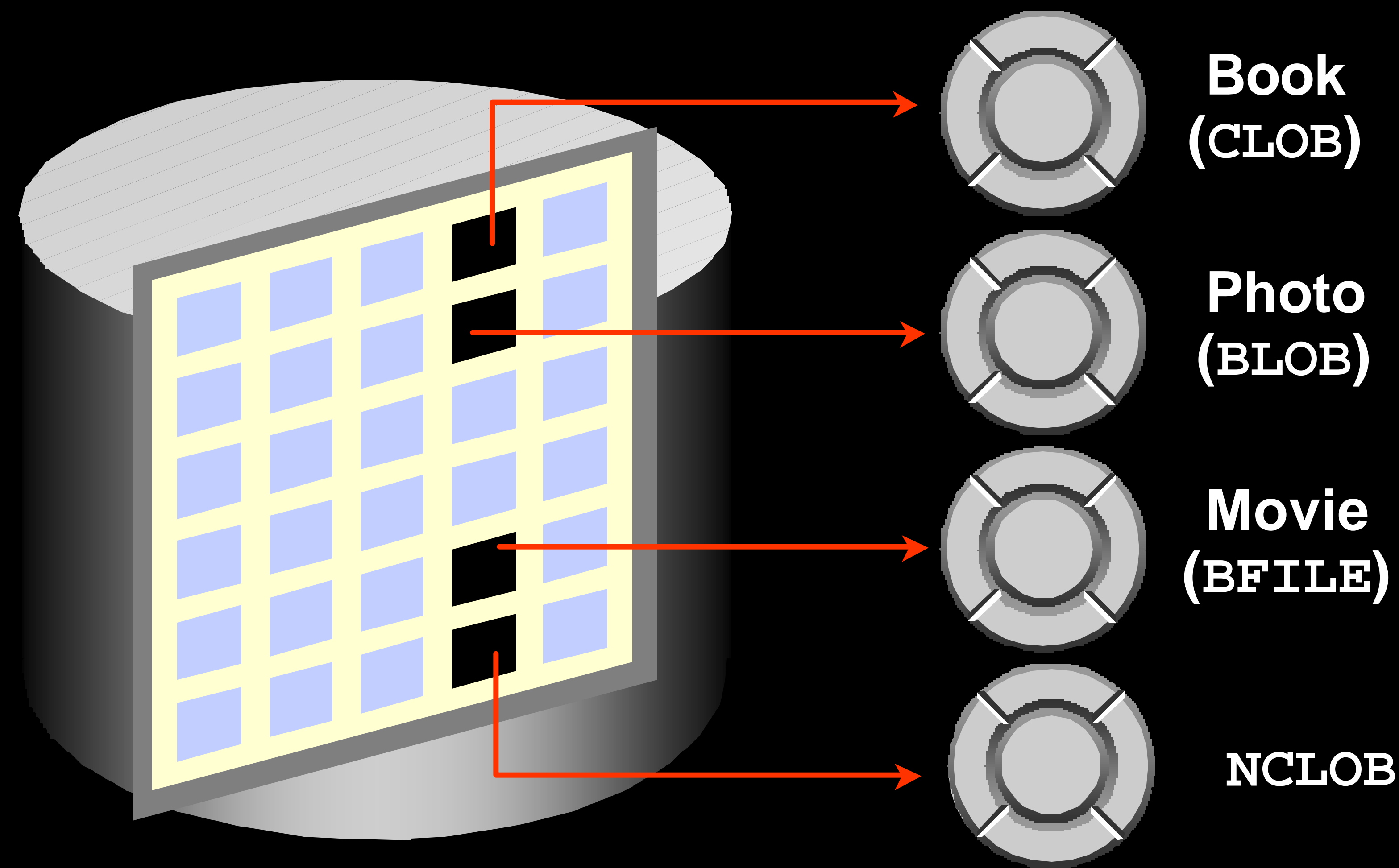
1	5000
2	2345
3	12
4	3456

NUMBER

BINARY_INTEGER

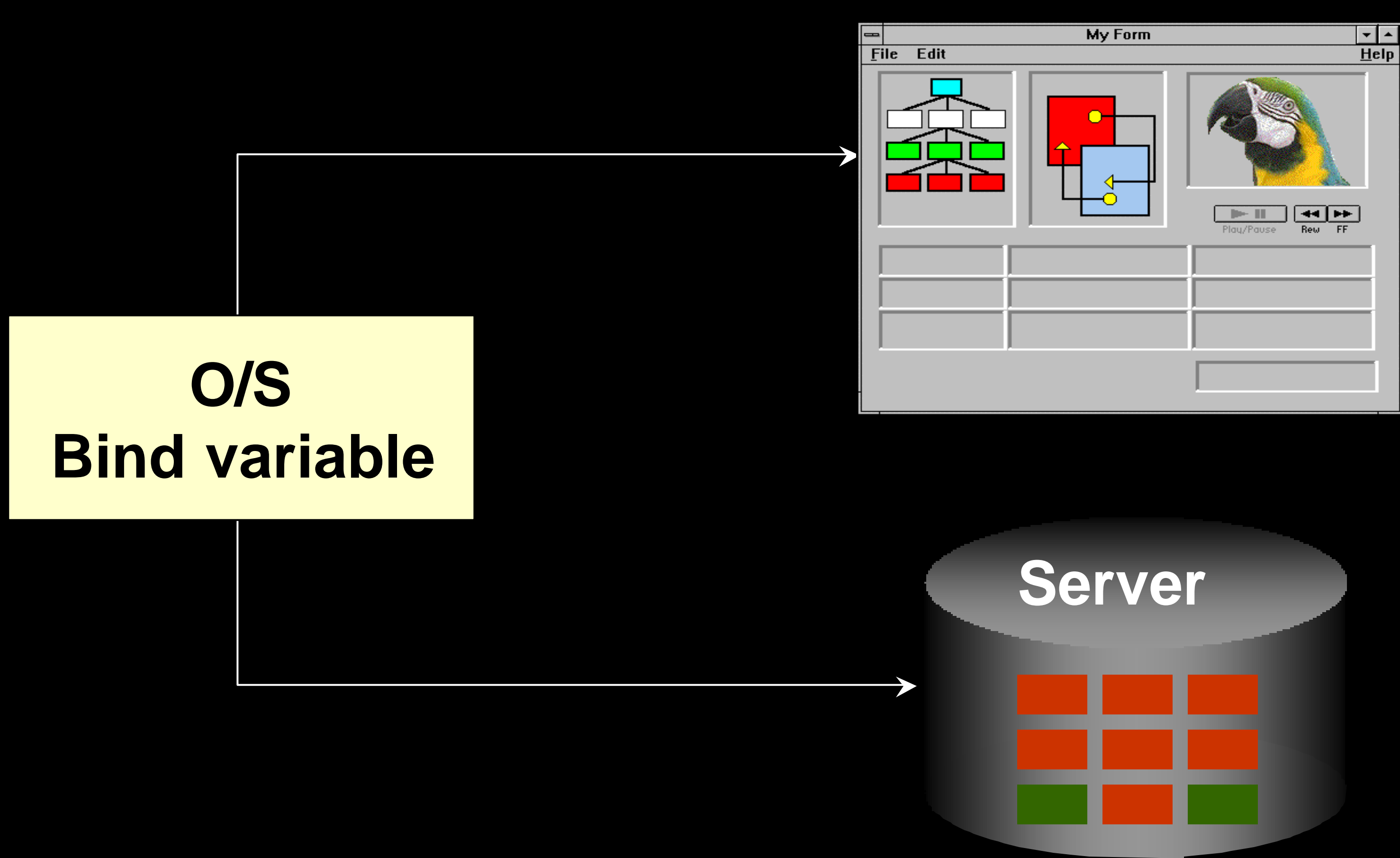
Copyright © Oracle Corporation, 2001. All rights reserved.

LOB Data Type Variables



Copyright © Oracle Corporation, 2001. All rights reserved.

Bind Variables



Copyright © Oracle Corporation, 2001. All rights reserved.

Using Bind Variables

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

Example:

```
VARIABLE      g_salary NUMBER
BEGIN
   SELECT     salary
   INTO       :g_salary
   FROM       employees
   WHERE      employee_id = 178;
END;
/
PRINT g_salary
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Referencing Non-PL/SQL Variables

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

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

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DBMS_OUTPUT.PUT_LINE

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

```
SET SERVEROUTPUT ON
DEFINE p_annual_sal = 60000
```

```
DECLARE
  v_sal NUMBER(9,2) := &p_annual_sal;
BEGIN
  v_sal := v_sal/12;
  DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' ||
                         TO_CHAR(v_sal));
END;
/
```

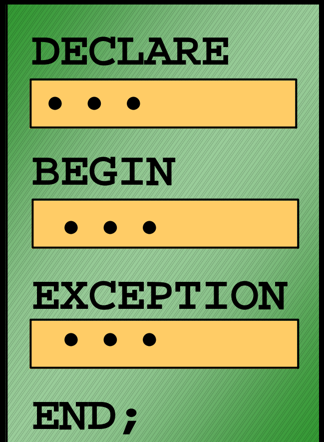
ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned that:

- PL/SQL blocks are composed of the following sections:
  - Declarative (optional)
  - Executable (required)
  - Exception handling (optional)
- A PL/SQL block can be an anonymous block, procedure, or function.



Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned that:

- PL/SQL identifiers:
  - Are defined in the declarative section
  - Can be of scalar, composite, reference, or `LOB` data type
  - Can be based on the structure of another variable or database object
  - Can be initialized
- Variables declared in an external environment such as *i*SQL*Plus are called host variables.
- Use `DBMS_OUTPUT.PUT_LINE` to display data from a PL/SQL block.

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 1 Overview

This practice covers the following topics:

- Determining validity of declarations
- Declaring a simple PL/SQL block
- Executing a simple PL/SQL block

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

2

Writing Executable Statements

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the significance of the executable section
- Use identifiers correctly
- Write statements in the executable section
- Describe the rules of nested blocks
- Execute and test a PL/SQL block
- Use coding conventions

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Block Syntax and Guidelines

- Statements can continue over several lines.
- Lexical units can be classified as:
  - Delimiters
  - Identifiers
  - Literals
  - Comments

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Identifiers

- Can contain up to 30 characters
- Must begin with an alphabetic character
- Can contain numerals, dollar signs, underscores, and number signs
- Cannot contain characters such as hyphens, slashes, and spaces
- Should not have the same name as a database table column name
- Should not be reserved words

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Block Syntax and Guidelines

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

```
v_name := 'Henderson';
```

  - Numbers can be simple values or scientific notation.
- A slash (/) runs the PL/SQL block in a script file or in some tools such as *i*SQL*PLUS.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Commenting Code

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

Example:

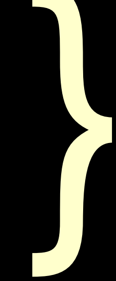
```
DECLARE
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := :g_monthly_sal * 12;
END;     -- This is the end of the block
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL Functions in PL/SQL

- Available in procedural statements:
  - Single-row number
  - Single-row character
  - Data type conversion
  - Date
  - Timestamp
  - `GREATEST` and `LEAST`
  - Miscellaneous functions



Same as in SQL

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

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL Functions in PL/SQL: 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);
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Data Type Conversion

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

```
DECLARE
   v_date DATE := TO_DATE('12-JAN-2001', 'DD-MON-YYYY');
BEGIN
   . . .
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Data Type Conversion

This statement produces a compilation error if the variable `v_date` is declared as a `DATE` data type.

```
v_date := 'January 13, 2001';
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Data Type Conversion

To correct the error, use the `TO_DATE` conversion function.

```
v_date := TO_DATE ('January 13, 2001',
                   'Month DD, YYYY');
```


Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

Nested Blocks and Variable Scope

- PL/SQL blocks 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 identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Nested Blocks and Variable Scope

Example:

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

Scope of *x*

Scope of *y*

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Identifier Scope

An identifier is visible in the regions where you can reference the identifier without having to qualify it:

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Qualify an Identifier

- The qualifier can be the label of an enclosing block.
- Qualify an identifier by using the block label prefix.

```
<<outer>>
   DECLARE
      birthdate DATE;
   BEGIN
         DECLARE
             birthdate DATE;
         BEGIN
            ...
            outer.birthdate :=
                            TO_DATE('03-AUG-1976',
                                     'DD-MON-YYYY');
         END;
    ....
    END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Determining Variable Scope

Class Exercise

```
<<outer>>
DECLARE
  v_sal      NUMBER(7,2) := 60000;
  v_comm     NUMBER(7,2) := v_sal * 0.20;
  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 * 0.30;
1 →
  END;
    v_message := 'SALESMAN'||v_message;
2 →
END;
```


Copyright © Oracle Corporation, 2001. All rights reserved.

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

} Same as in SQL

- Exponential operator (**)

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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 whether an employee number contains a value.

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Indenting Code

For clarity, indent each level of code.

Example:

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

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

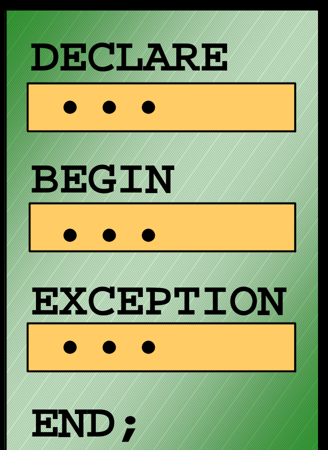
ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned that:

- PL/SQL block syntax and guidelines
- How to use identifiers correctly
- PL/SQL block structure: nesting blocks and scoping rules
- PL/SQL programming:
  - Functions
  - Data type conversions
  - Operators
  - Conventions and guidelines



Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 2 Overview

This practice covers the following topics:

- Reviewing scoping and nesting rules
- Developing and testing PL/SQL blocks

Copyright © Oracle Corporation, 2001. All rights reserved.

Interacting with the Oracle Server

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Write a successful `SELECT` statement in PL/SQL
- Write DML statements in PL/SQL
- Control transactions in PL/SQL
- Determine the outcome of SQL data manipulation language (DML) statements

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

`SELECT` Statements in PL/SQL

Retrieve data from the database with a `SELECT` statement.

Syntax:

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


Copyright © Oracle Corporation, 2001. All rights reserved.

SELECT Statements in PL/SQL

- The INTO clause is required.
- Queries must return one and only one row.

Example:

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Retrieving Data in PL/SQL

Retrieve the hire date and the salary for the specified employee.

Example:

```
DECLARE
  v_hire_date   employees.hire_date%TYPE;
  v_salary      employees.salary%TYPE;
BEGIN
  SELECT   hire_date, salary
  INTO     v_hire_date, v_salary
  FROM     employees
  WHERE    employee_id = 100;
  ...
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Retrieving Data in PL/SQL

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

Example:

```
SET SERVEROUTPUT ON
DECLARE
  v_sum_sal   NUMBER(10,2);
  v_deptno     NUMBER NOT NULL := 60;
BEGIN
  SELECT     SUM(salary)  -- group function
  INTO       v_sum_sal
  FROM       employees
  WHERE      department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum salary is ' ||
                        TO_CHAR(v_sum_sal));
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Naming Conventions

```
DECLARE
  hire_date       employees.hire_date%TYPE;
  sysdate         hire_date%TYPE;
  employee_id     employees.employee_id%TYPE := 176;
BEGIN
  SELECT      hire_date, sysdate
  INTO        hire_date, sysdate
  FROM        employees
  WHERE       employee_id = employee_id;
END;
/
```

```
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
```

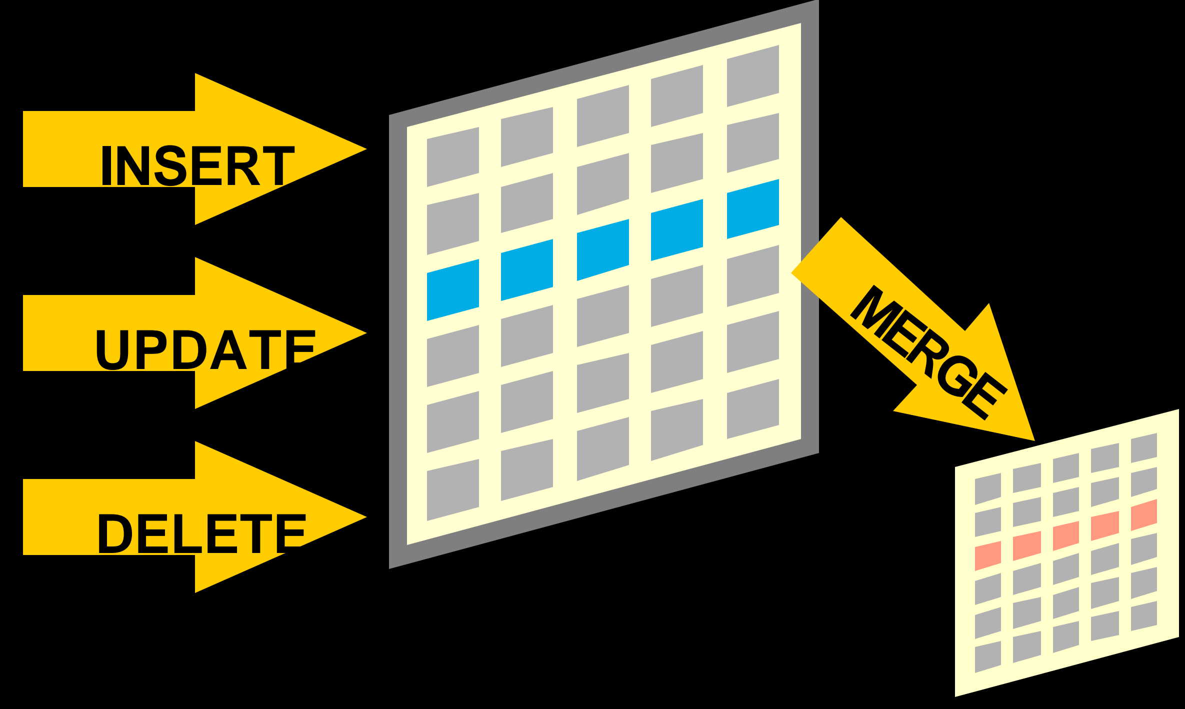
ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Manipulating Data Using PL/SQL

Make changes to database tables by using DML commands:

- `INSERT`
- `UPDATE`
- `DELETE`
- `MERGE`



Copyright © Oracle Corporation, 2001. All rights reserved.

Inserting Data

Add new employee information to the `EMPLOYEES` table.

Example:

```
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;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Updating Data

Increase the salary of all employees who are stock clerks.

Example:

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Deleting Data

Delete rows that belong to department 10 from the `EMPLOYEES` table.

Example:

```
DECLARE
  v_deptno   employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM   employees
  WHERE         department_id = v_deptno;
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Merging Rows

Insert or update rows in the `COPY_EMP` table to match the `EMPLOYEES` table.

```
DECLARE
      v_empno employees.employee_id%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (e.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,
       . . .
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id, e.first_name, e.last_name,
          . . .,e.department_id);
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL Cursor Attributes

Delete rows that have the specified employee ID from the `EMPLOYEES` table. Print the number of rows deleted.

Example:

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_employee_id employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM  employees
  WHERE        employee_id = v_employee_id;
  :rows_deleted := (SQL%ROWCOUNT ||
                         ' row deleted.');
END;
/
PRINT rows_deleted
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Transaction Control 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.**

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned how to:

- Embed SQL in the PL/SQL block using `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `MERGE`
- Embed transaction control statements in a PL/SQL block `COMMIT`, `ROLLBACK`, and `SAVEPOINT`

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned that:

- There are two cursor types: implicit and explicit.
- Implicit cursor attributes are used to verify the outcome of DML statements:
  - `SQL%ROWCOUNT`
  - `SQL%FOUND`
  - `SQL%NOTFOUND`
  - `SQL%ISOPEN`
- Explicit cursors are defined by the programmer.

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 3 Overview

This practice covers creating a PL/SQL block to:

- Select data from a table
- Insert data into a table
- Update data in a table
- Delete a record from a table

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

4

Writing Control Structures

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

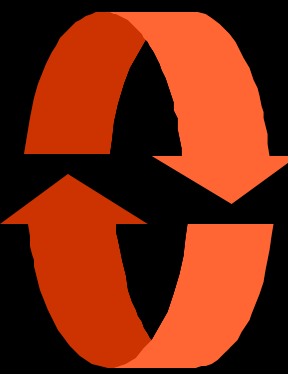
- Identify the uses and types of control structures
- Construct an `IF` statement
- Use `CASE` expressions
- Construct and identify different loop statements
- Use logic tables
- Control block flow using nested loops and labels

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling PL/SQL Flow of Execution

- You can change the logical execution 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`



Copyright © Oracle Corporation, 2001. All rights reserved.

IF Statements

Syntax:

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

If the employee name is Gietz, set the Manager ID to 102.

```
IF UPPER(v_last_name) = 'GIETZ' THEN
  v_mgr := 102;
END IF;
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Simple IF Statements

If the last name is Vargas:

- Set job ID to SA_REP
- Set department number to 80

```
. . .
IF v_ename      = 'Vargas' THEN
   v_job       := 'SA_REP';
   v_deptno    := 80;
END IF;
. . .
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Compound IF Statements

If the last name is Vargas and the salary is more than 6500:

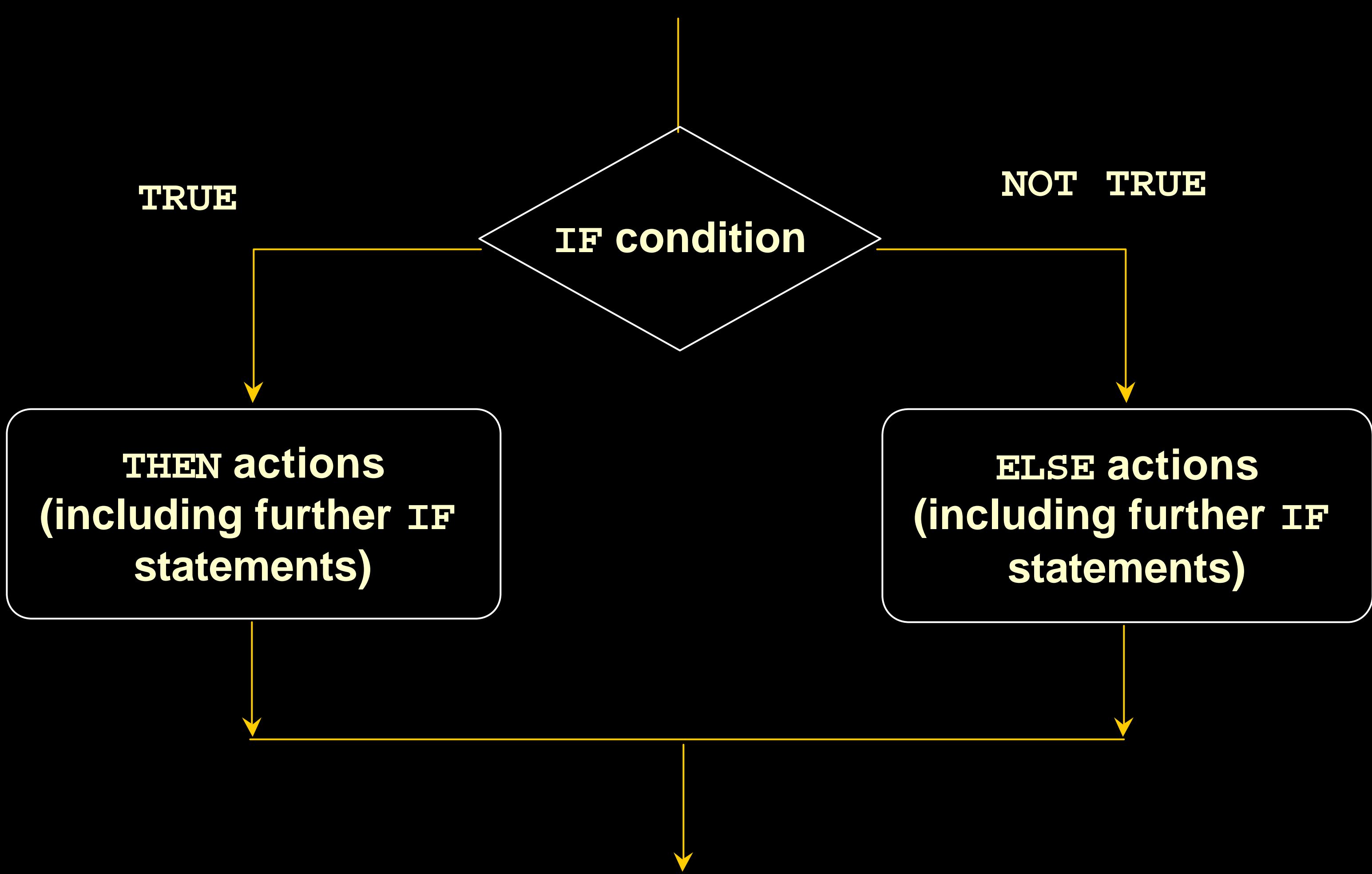
Set department number to 60.

```
. . .
IF v_ename  = 'Vargas' AND salary > 6500 THEN
   v_deptno := 60;
END IF;
. . .
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

IF-THEN-ELSE Statement Execution Flow



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

IF-THEN-ELSE Statements

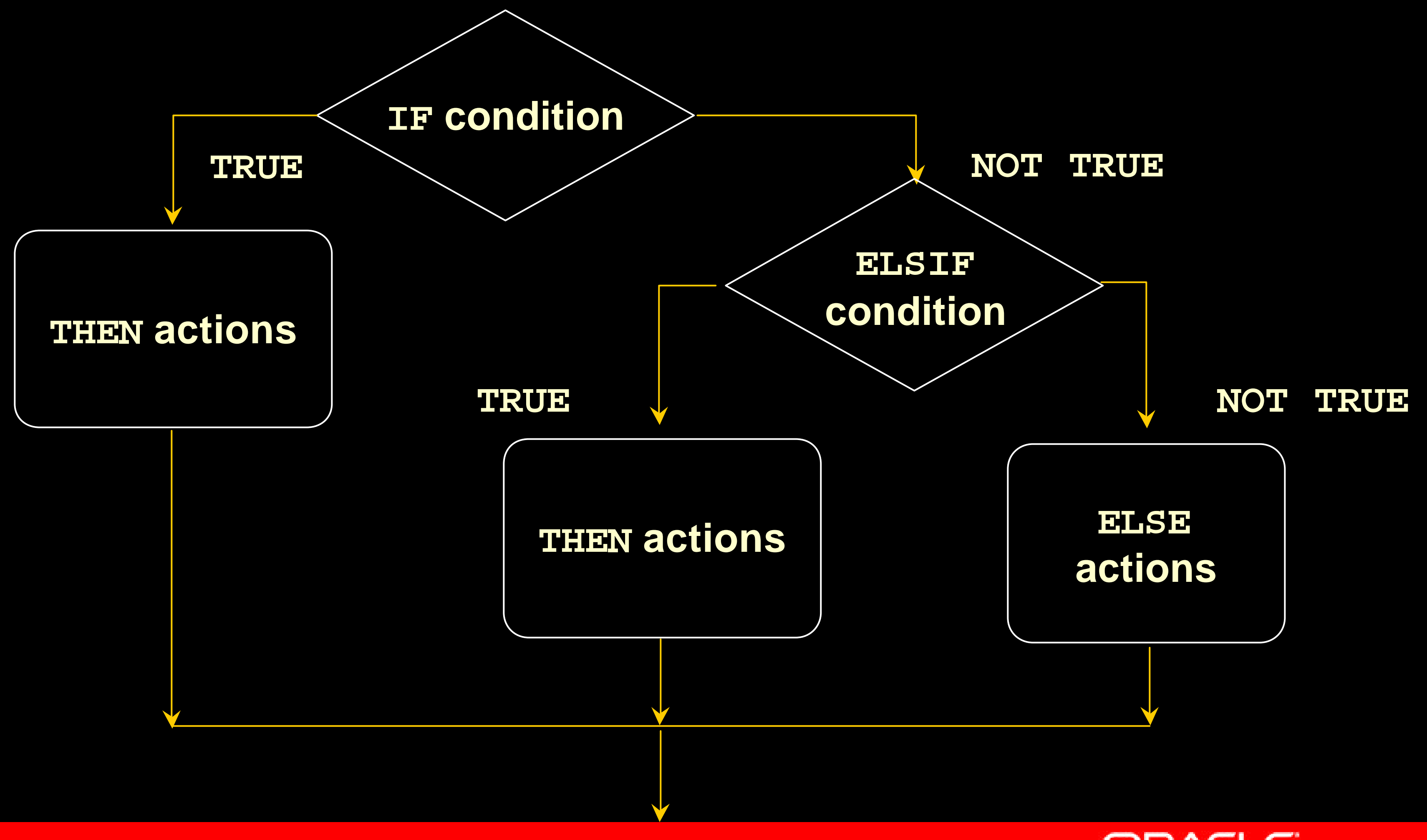
Set a Boolean flag to TRUE if the hire date is greater than five years; otherwise, set the Boolean flag to FALSE.

```
DECLARE
   v_hire_date  DATE := '12-Dec-1990';
   v_five_years BOOLEAN;
BEGIN
. . .
IF MONTHS_BETWEEN(SYSDATE,v_hire_date)/12 > 5 THEN
   v_five_years := TRUE;
ELSE
   v_five_years := FALSE;
END IF;
...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

IF-THEN-ELSIF Statement Execution Flow



Copyright © Oracle Corporation, 2001. All rights reserved.

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 := 0.2 * v_start;
ELSIF v_start >= 50 THEN
      v_start := 0.5 * v_start;
ELSE
      v_start := 0.1 * v_start;
END IF;
. . .
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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 expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1;]
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

CASE Expressions: Example

```
SET SERVEROUTPUT ON
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;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Handling Nulls

When working with nulls, you can 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.

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Boolean Conditions

What is the value of `V_FLAG` in each case?

```
v_flag := v_reorder_flag AND v_available_flag;
```

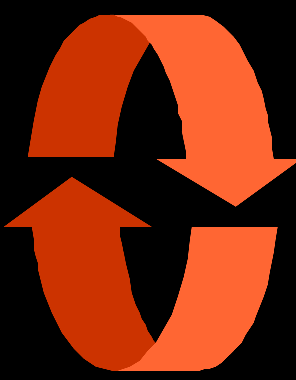
V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	?
TRUE	FALSE	?
NULL	TRUE	?
NULL	FALSE	?

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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



Copyright © Oracle Corporation, 2001. All rights reserved.

Basic Loops

Syntax:

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

```
condition     is a Boolean variable or
              expression (TRUE, FALSE, or NULL);
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Basic Loops

Example:

```
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;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

`WHILE` Loops

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`.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

WHILE Loops

Example:

```
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;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

FOR Loops

Syntax:

```
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 counter; it is declared implicitly.
- '`lower_bound .. upper_bound`' is required syntax.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

FOR Loops

Insert three new locations IDs for the country code of CA and the city of Montreal.

```
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;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

`FOR` Loops

Guidelines

- Reference the counter within the loop only; it is undefined outside the loop.
- Do *not* reference the counter as the target of an assignment.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

Copyright © Oracle Corporation, 2001. All rights reserved.

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 that references the label.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned to:

Change the logical flow of statements by using control structures.

- Conditional (`IF` statement)
- `CASE` Expressions
- Loops:
  - Basic loop
  - `FOR` loop
  - `WHILE` loop
- `EXIT` statements

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 4 Overview

This practice covers the following topics:

- Performing conditional actions using the `IF` statement
- Performing iterative steps using the loop structure

Copyright © Oracle Corporation, 2001. All rights reserved.

5

Working with Composite Data Types

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Create user-defined PL/SQL records
- Create a record with the `%ROWTYPE` attribute
- Create an `INDEX BY` table
- Create an `INDEX BY` table of records
- Describe the difference between records, tables, and tables of records

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Composite Data Types

- Are of two types:
  - PL/SQL RECORDS
  - PL/SQL Collections
    - INDEX BY Table
    - Nested Table
    - VARRAY
- Contain internal components
- Are reusable

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Records

- Must contain one or more components of any scalar, `RECORD`, or `INDEX BY` table data type, called fields
- Are similar in structure to records in a third generation language (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

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

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]
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a PL/SQL Record

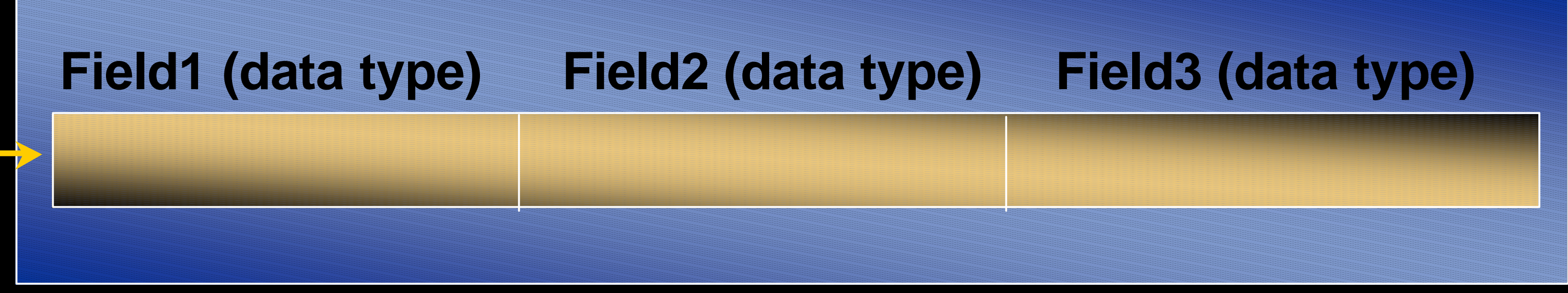
Declare variables to store the name, job, and salary of a new employee.

Example:

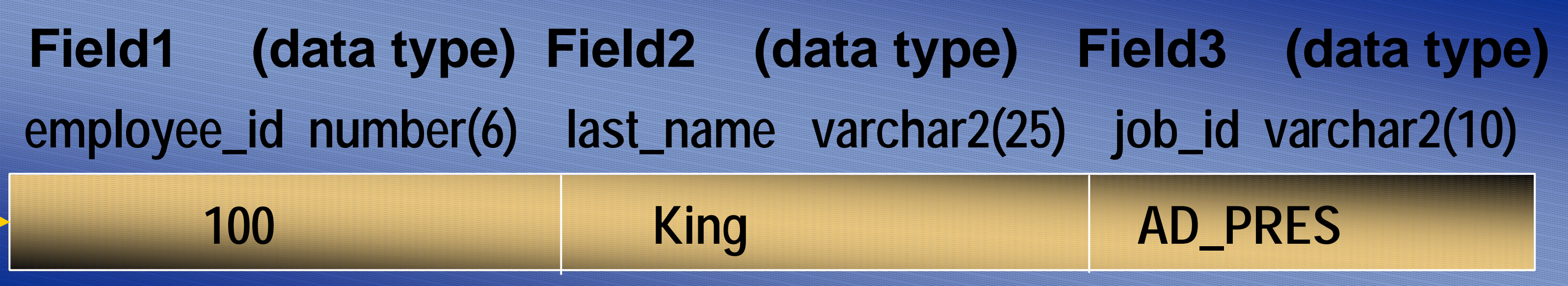
```
...
  TYPE emp_record_type IS RECORD
    (last_name   VARCHAR2(25),
     job_id      VARCHAR2(10),
     salary      NUMBER(8,2));
  emp_record     emp_record_type;
...
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Record Structure



Example:



Copyright © Oracle Corporation, 2001. All rights reserved.

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 data types from the columns of the table or view.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Advantages of Using `%ROWTYPE`

- The number and data types of the underlying database columns need not be known.
- The number and data types of the underlying database column may change at run time.
- The attribute is useful when retrieving a row with the `SELECT *` statement.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The %ROWTYPE Attribute

Examples:

Declare a variable to store the information about a department from the DEPARTMENTS table.

```
dept_record    departments%ROWTYPE;
```

Declare a variable to store the information about an employee from the EMPLOYEES table.

```
emp_record     employees%ROWTYPE;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

INDEX BY Tables

- Are composed of two components:
  - Primary key of data type BINARY_INTEGER
  - Column of scalar or record data type
- Can increase in size dynamically because they are unconstrained

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating an INDEX BY Table

Syntax:

```
TYPE type_name IS TABLE OF
     {column_type | variable%TYPE
     | table.column%TYPE} [NOT NULL]
     | table.%ROWTYPE
     [INDEX BY BINARY_INTEGER];
identifier   type_name;
```

Declare an INDEX BY table to store names.

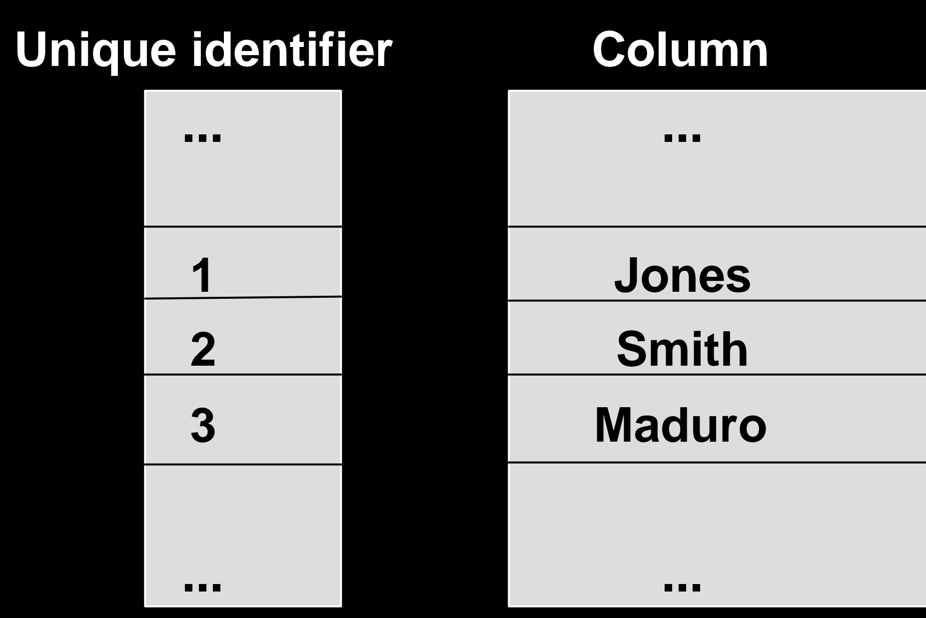
Example:

```
...
TYPE ename_table_type IS TABLE OF
                              employees.last_name%TYPE
  INDEX BY BINARY_INTEGER;
ename_table ename_table_type;
...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

INDEX BY Table Structure



Copyright © Oracle Corporation, 2001. All rights reserved.

Creating an INDEX BY Table

```
DECLARE
  TYPE ename_table_type IS TABLE OF
       employees.last_name%TYPE
       INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
       INDEX BY BINARY_INTEGER;
  ename_table         ename_table_type;
  hiredate_table      hiredate_table_type;
BEGIN
  ename_table(1)      := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
      INSERT INTO ...
    ...
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using `INDEX BY` Table Methods

The following methods make INDEX BY tables easier to use:

- `EXISTS`
- `COUNT`
- `FIRST` and `LAST`
- `PRIOR`
- `NEXT`
- `TRIM`
- `DELETE`

Copyright © Oracle Corporation, 2001. All rights reserved.

`INDEX BY` Table of Records

- Define a `TABLE` variable with a permitted PL/SQL data type.
- Declare a PL/SQL variable to hold department information.

Example:

```
DECLARE
  TYPE dept_table_type IS TABLE OF
       departments%ROWTYPE
       INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Example of INDEX BY Table of Records

```
SET SERVEROUTPUT ON
DECLARE
   TYPE emp_table_type is table of
      employees%ROWTYPE INDEX BY BINARY_INTEGER;
   my_emp_table  emp_table_type;
   v_count       NUMBER(3):= 104;
BEGIN
  FOR i IN 100..v_count
  LOOP
         SELECT * INTO my_emp_table(i) FROM employees
         WHERE employee_id = i;
  END LOOP;
  FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned to:

- Define and reference PL/SQL variables of composite data types:
  - PL/SQL records
  - `INDEX BY` tables
  - `INDEX BY` table of records
- Define a PL/SQL record by using the `%ROWTYPE` attribute

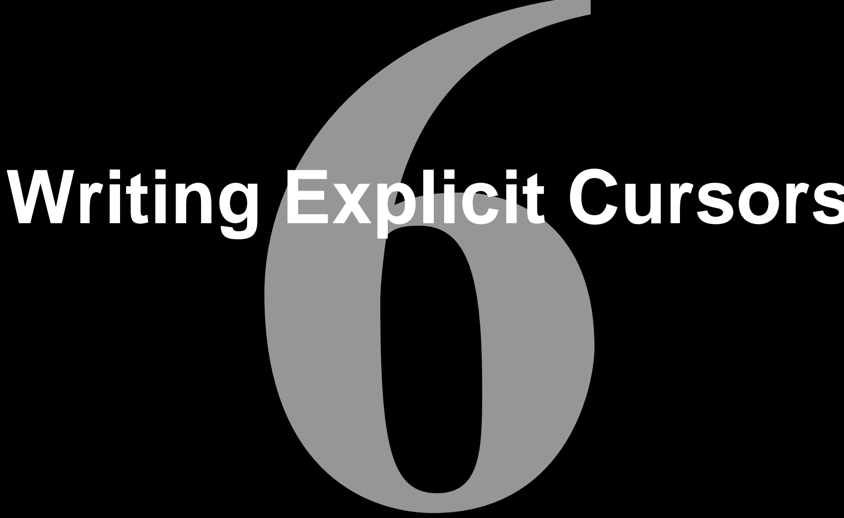
ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 5 Overview

This practice covers the following topics:

- Declaring `INDEX BY` tables
- Processing data by using `INDEX BY` tables
- Declaring a PL/SQL record
- Processing data by using a PL/SQL record

Copyright © Oracle Corporation, 2001. All rights reserved.



Writing Explicit Cursors

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Distinguish between an implicit and an explicit cursor
- Discuss when and why to use an explicit cursor
- Use a PL/SQL record variable
- Write a cursor `FOR` loop

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

About Cursors

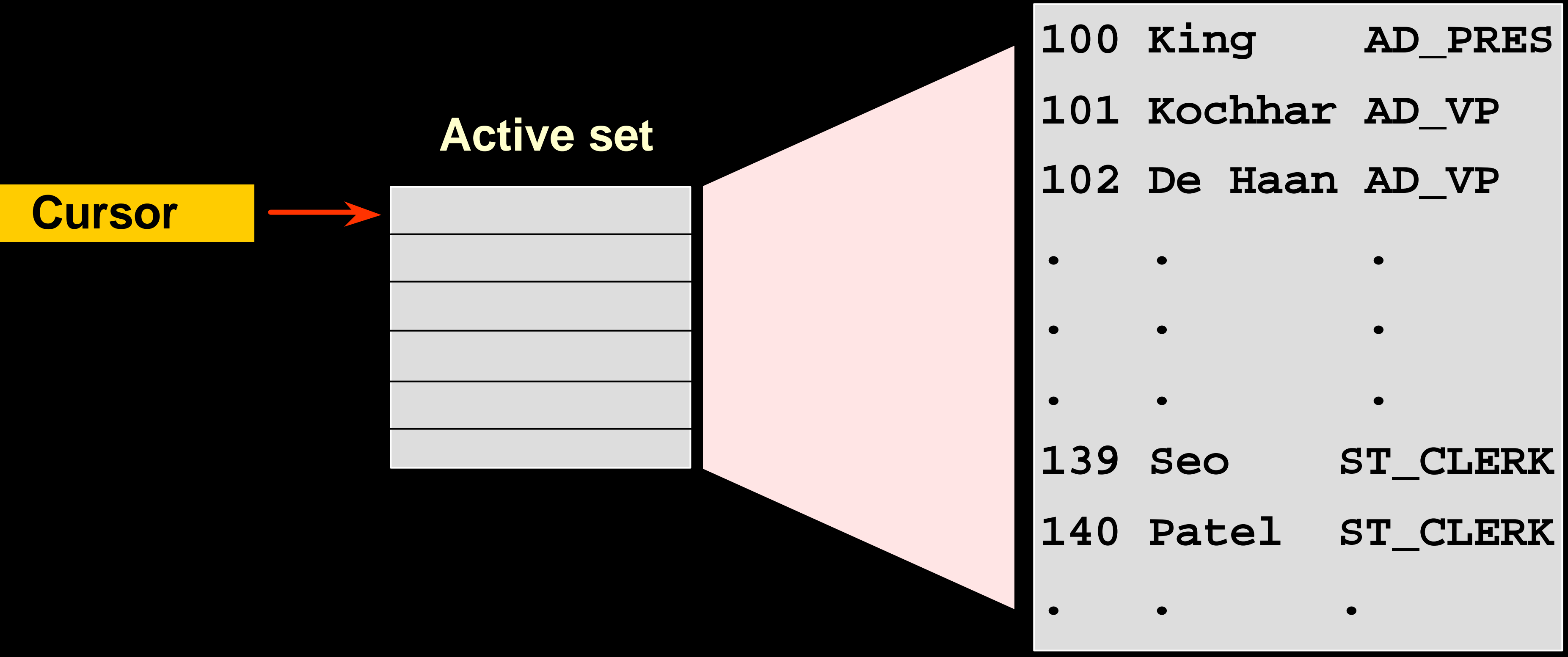
Every SQL statement executed by the Oracle Server has an individual cursor associated with it:

- Implicit cursors: Declared for all DML and PL/SQL `SELECT` statements
- Explicit cursors: Declared and named by the programmer

ORACLE

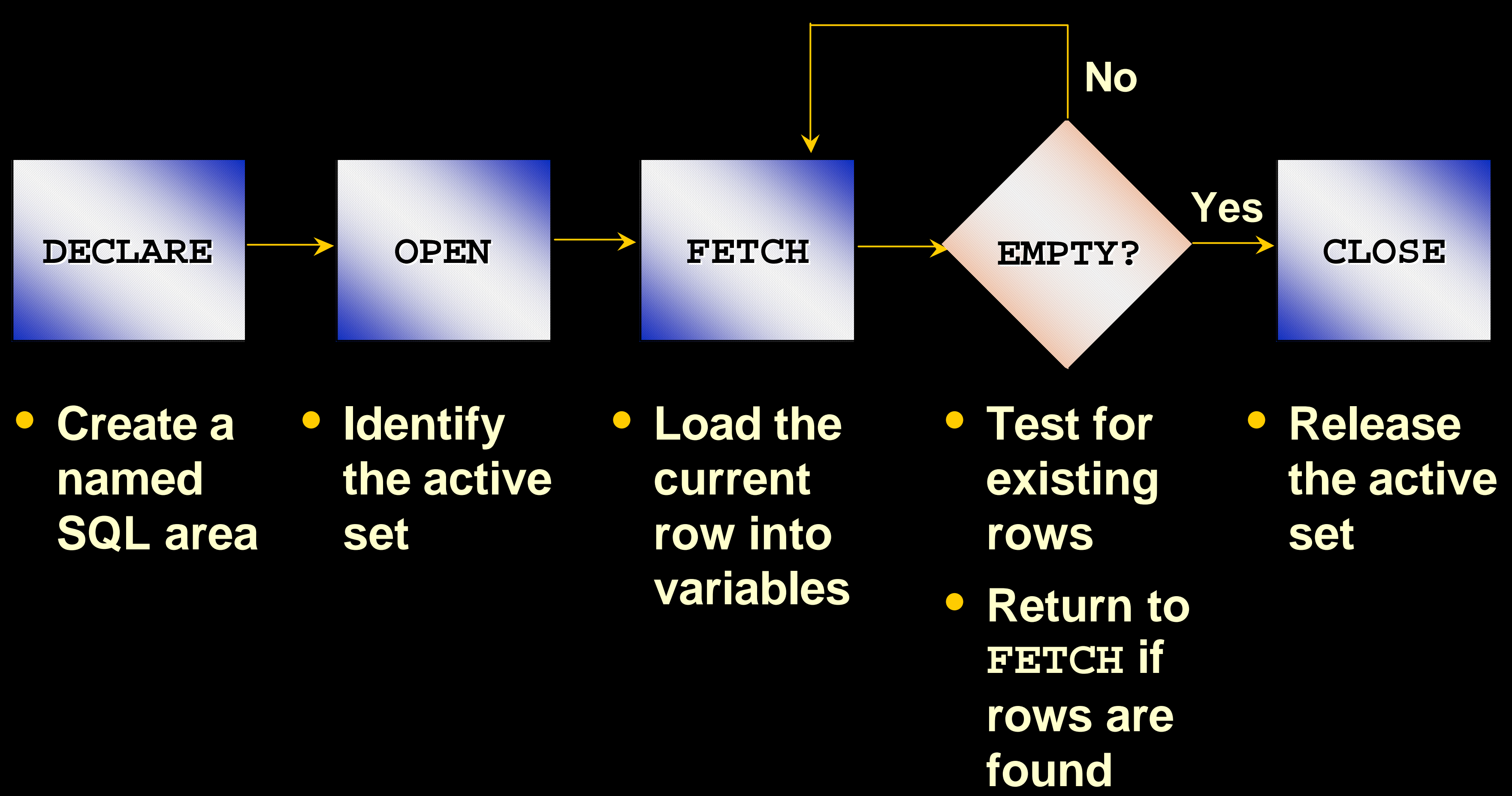
Copyright © Oracle Corporation, 2001. All rights reserved.

Explicit Cursor Functions



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Explicit Cursors

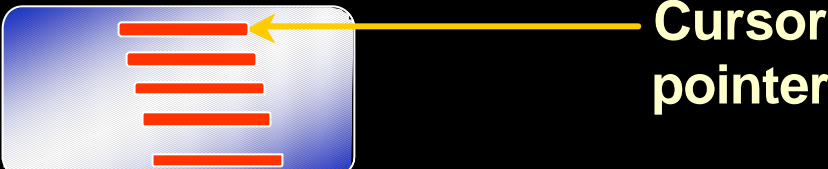


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Explicit Cursors

1. Open the cursor
2. Fetch a row
3. Close the Cursor

1. Open the cursor.



Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Explicit Cursors

1. Open the cursor
2. Fetch a row
3. Close the Cursor

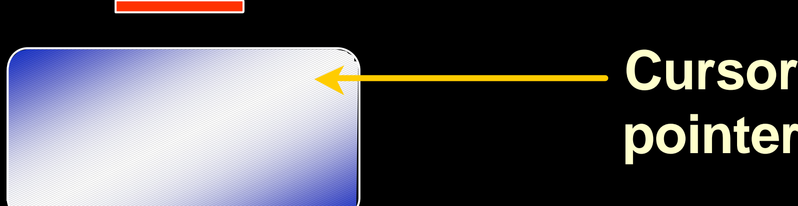
2. Fetch a row using the cursor.

Continue until empty.

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Explicit Cursors

1. Open the cursor
2. Fetch a row
3. Close the Cursor



Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring the Cursor

Syntax:

```
CURSOR cursor_name IS
     select_statement;
```

- **Do not include the `INTO` clause in the cursor declaration.**
- **If processing rows in a specific sequence is required, use the `ORDER BY` clause in the query.**

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring the Cursor

Example:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM   employees;

  CURSOR dept_cursor IS
    SELECT *
    FROM   departments
    WHERE  location_id = 170;
BEGIN
  ...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Opening the Cursor

Syntax:

```
OPEN  cursor_name;
```

- Open the cursor to execute the query and identify the active set.
- If the query returns no rows, no exception is raised.
- Use cursor attributes to test the outcome after a fetch.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Fetching Data from the Cursor

Syntax:

```
FETCH cursor_name INTO  [variable1, variable2, ...]
                        | record_name];
```

- Retrieve the current row values into variables.
- Include the same number of variables.
- Match each variable to correspond to the columns positionally.
- Test to see whether the cursor contains rows.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Fetching Data from the Cursor

Example:

```
LOOP
  FETCH emp_cursor INTO v_empno,v_ename;
  EXIT WHEN ...;
  ...
    -- Process the retrieved data
  …
END LOOP;
```


ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Closing the Cursor

Syntax:

```
CLOSE    cursor_name;
```

- **Close the cursor after completing the processing of the rows.**
- **Reopen the cursor, if required.**
- **Do not attempt to fetch data from a cursor after it has been closed.**

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Explicit Cursor Attributes

Obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The %ISOPEN Attribute

- Fetch rows only when the cursor is open.
- Use the %ISOPEN cursor attribute before performing a fetch to test whether the cursor is open.

Example:

```
IF NOT emp_cursor%ISOPEN THEN
    OPEN emp_cursor;
END IF;
LOOP
  FETCH emp_cursor...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Multiple Fetches

- Process several rows from an explicit cursor using a loop.
- Fetch a row with each iteration.
- Use explicit cursor attributes to test the success of each fetch.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The %NOTFOUND and %ROWCOUNT Attributes

- Use the %ROWCOUNT cursor attribute to retrieve an exact number of rows.
- Use the %NOTFOUND cursor attribute to determine when to exit the loop.

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Example

```
DECLARE
      v_empno  employees.employee_id%TYPE;
      v_ename  employees.last_name%TYPE;
      CURSOR emp_cursor IS
        SELECT employee_id, last_name
        FROM   employees;
    BEGIN
      OPEN emp_cursor;
      LOOP
        FETCH emp_cursor INTO v_empno, v_ename;
        EXIT WHEN emp_cursor%ROWCOUNT > 10 OR
                          emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(v_empno)
                              ||'  '|| v_ename);
      END LOOP;
      CLOSE emp_cursor;
END ;
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Cursors and Records

Process the rows of the active set by fetching values into a PL/SQL RECORD.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT  employee_id, last_name
    FROM    employees;
  emp_record   emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
  ...
```

emp_record

employee_id	last_name
100	King

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Cursor FOR Loops

Syntax:

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- The cursor FOR loop is a shortcut to process explicit cursors.
- Implicit open, fetch, exit, and close occur.
- The record is implicitly declared.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Cursor FOR Loops

Print a list of the employees who work for the sales department.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT last_name, department_id
    FROM   employees;
BEGIN
  FOR emp_record IN emp_cursor LOOP
         -- implicit open and implicit fetch occur
    IF emp_record.department_id = 80 THEN
      ...
  END LOOP; -- implicit close occurs
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Cursor FOR Loops Using Subqueries

No need to declare the cursor.

Example:

```
BEGIN
  FOR emp_record IN (SELECT last_name, department_id
                     FROM   employees) LOOP
        -- implicit open and implicit fetch occur
    IF emp_record.department_id = 80 THEN
      ...
  END LOOP; -- implicit close occurs
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson you should have learned to:

- Distinguish cursor types:
  - Implicit cursors: used for all DML statements and single-row queries
  - Explicit cursors: used for queries of zero, one, or more rows
- Manipulate explicit cursors
- Evaluate the cursor status by using cursor attributes
- Use cursor FOR loops

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 6 Overview

This practice covers the following topics:

- Declaring and using explicit cursors to query rows of a table
- Using a cursor `FOR` loop
- Applying cursor attributes to test the cursor status

Copyright © Oracle Corporation, 2001. All rights reserved.

7

Advanced Explicit Cursor Concepts

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Write a cursor that uses parameters
- Determine when a `FOR UPDATE` clause in a cursor is required
- Determine when to use the `WHERE CURRENT OF` clause
- Write a cursor that uses a subquery

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Cursors with Parameters

Syntax:

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- Pass parameter values to a cursor when the cursor is opened and the query is executed.
- Open an explicit cursor several times with a different active set each time.

```
OPEN  cursor_name(parameter_value,.....) ;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Cursors with Parameters

Pass the department number and job title to the `WHERE` clause, in the cursor `SELECT` statement.

```
DECLARE
  CURSOR emp_cursor
  (p_deptno NUMBER, p_job VARCHAR2) IS
     SELECT employee_id, last_name
     FROM   employees
     WHERE  department_id = p_deptno
     AND    job_id = p_job;
BEGIN
  OPEN emp_cursor (80, 'SA_REP');
  . . .
  CLOSE emp_cursor;
  OPEN emp_cursor (60, 'IT_PROG');
  . . .
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The `FOR UPDATE` Clause

Syntax:

```
SELECT ...
FROM         ...
FOR UPDATE [OF column_reference][NOWAIT];
```

- **Use explicit locking to deny access for the duration of a transaction.**
- **Lock the rows *before* the update or delete.**

Copyright © Oracle Corporation, 2001. All rights reserved.

The FOR UPDATE Clause

Retrieve the employees who work in department 80 and update their salary.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name, department_name
    FROM   employees,departments
    WHERE  employees.department_id =
           departments.department_id
    AND employees.department_id = 80
    FOR UPDATE OF salary NOWAIT;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The `WHERE CURRENT OF` Clause

Syntax:

```
WHERE CURRENT OF cursor ;
```

- Use cursors to update or delete the current row.
- Include the `FOR UPDATE` clause in the cursor query to lock the rows first.
- Use the `WHERE CURRENT OF` clause to reference the current row from an explicit cursor.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The WHERE CURRENT OF Clause

```
DECLARE
CURSOR sal_cursor IS
 SELECT e.department_id, employee_id, last_name, salary
 FROM   employees e, departments d
 WHERE  d.department_id = e.department_id
  and   d.department_id = 60
 FOR UPDATE OF salary NOWAIT;
 BEGIN
 FOR emp_record IN sal_cursor
LOOP
  IF emp_record.salary < 5000 THEN
       UPDATE employees
       SET    salary = emp_record.salary * 1.10
       WHERE CURRENT OF sal_cursor;
  END IF;
END LOOP;
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Cursors with Subqueries

Example:

```
DECLARE
  CURSOR my_cursor IS
    SELECT t1.department_id, t1.department_name,
           t2.staff
    FROM   departments t1, (SELECT department_id,
                                   COUNT(*) AS STAFF
                            FROM employees
                            GROUP BY department_id) t2
    WHERE t1.department_id = t2.department_id
    AND   t2.staff >= 3;
...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned to:

- Return different active sets using cursors with parameters.
- Define cursors with subqueries and correlated subqueries.
- Manipulate explicit cursors with commands using the:
  - `FOR UPDATE` clause
  - `WHERE CURRENT OF` clause

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 7 Overview

This practice covers the following topics:

- Declaring and using explicit cursors with parameters
- Using a `FOR UPDATE` cursor

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

8

Handling Exceptions

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Define PL/SQL exceptions
- Recognize unhandled exceptions
- List and use different types of PL/SQL exception handlers
- Trap unanticipated errors
- Describe the effect of exception propagation in nested blocks
- Customize PL/SQL exception messages

Copyright © Oracle Corporation, 2001. All rights reserved.

Handling Exceptions with PL/SQL

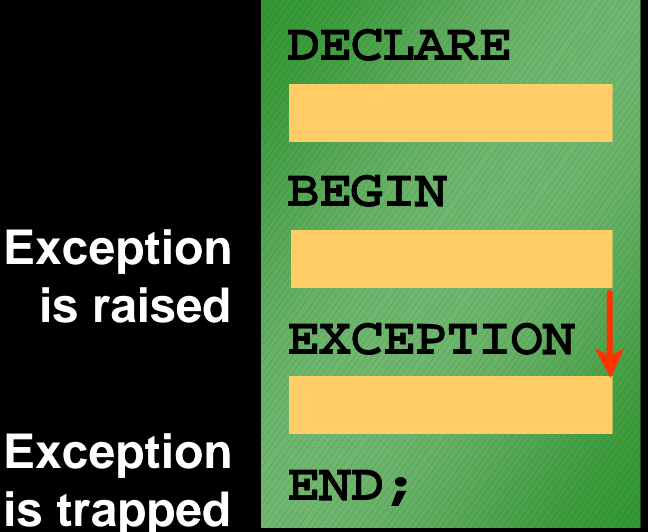
- An exception is an identifier in PL/SQL that is raised during execution.
- How is it raised?
  - An Oracle error occurs.
  - You raise it explicitly.
- How do you handle it?
  - Trap it with a handler.
  - Propagate it to the calling environment.

ORACLE

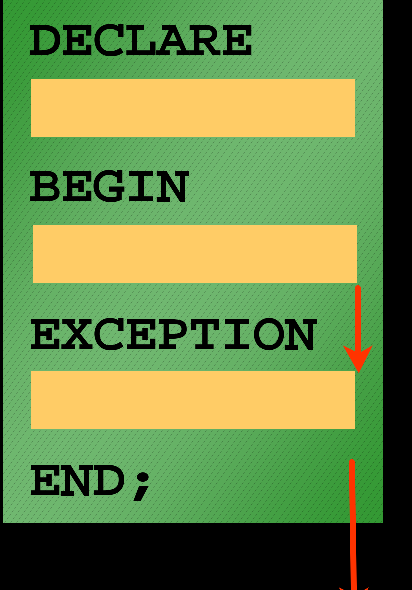
Copyright © Oracle Corporation, 2001. All rights reserved.

Handling Exceptions

Trap the exception



Propagate the exception



Copyright © Oracle Corporation, 2001. All rights reserved.

Exception Types

- Predefined Oracle Server
- Nonpredefined Oracle Server

- User-defined   Explicitly raised

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Trapping Exceptions

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trapping Exceptions Guidelines

- The `EXCEPTION` keyword starts exception-handling section.
- Several exception handlers are allowed.
- Only one handler is processed before leaving the block.
- `WHEN OTHERS` is the last clause.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trapping Predefined Oracle Server Errors

- Reference the standard name in the exception-handling routine.
- Sample predefined exceptions:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Predefined Exceptions

Syntax:

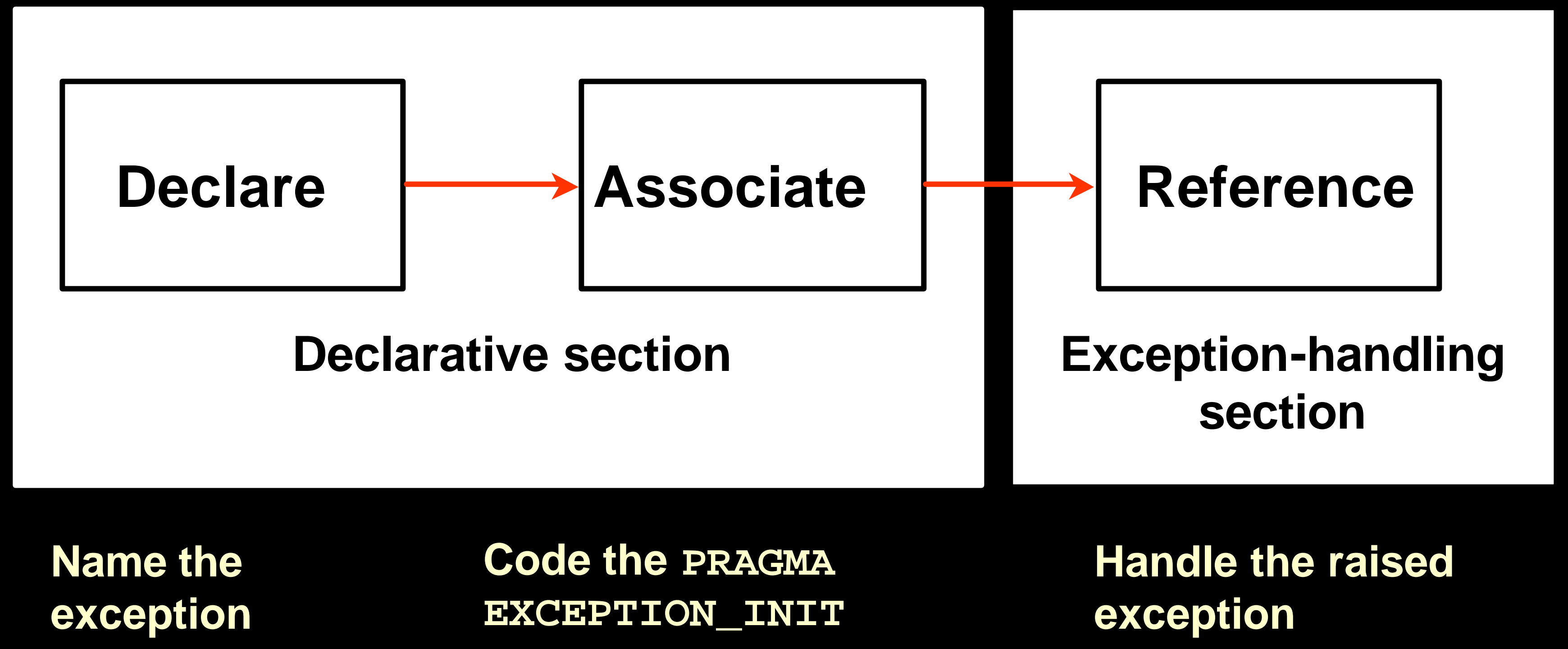
```
BEGIN
. . .
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;

  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trapping Nonpredefined Oracle Server Errors



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Nonpredefined Error

Trap for Oracle server error number –2292, an integrity constraint violation.

```
DEFINE p_deptno = 10
DECLARE
  e_emps_remaining EXCEPTION;
  PRAGMA EXCEPTION_INIT
    (e_emps_remaining, -2292);
BEGIN
  DELETE FROM departments
  WHERE  department_id = &p_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining  THEN
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(&p_deptno) || '.  Employees exist. ');
END;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Functions for Trapping Exceptions

- `SQLCODE`: Returns the numeric value for the error code
- `SQLERRM`: Returns the message associated with the error number

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Functions for Trapping Exceptions

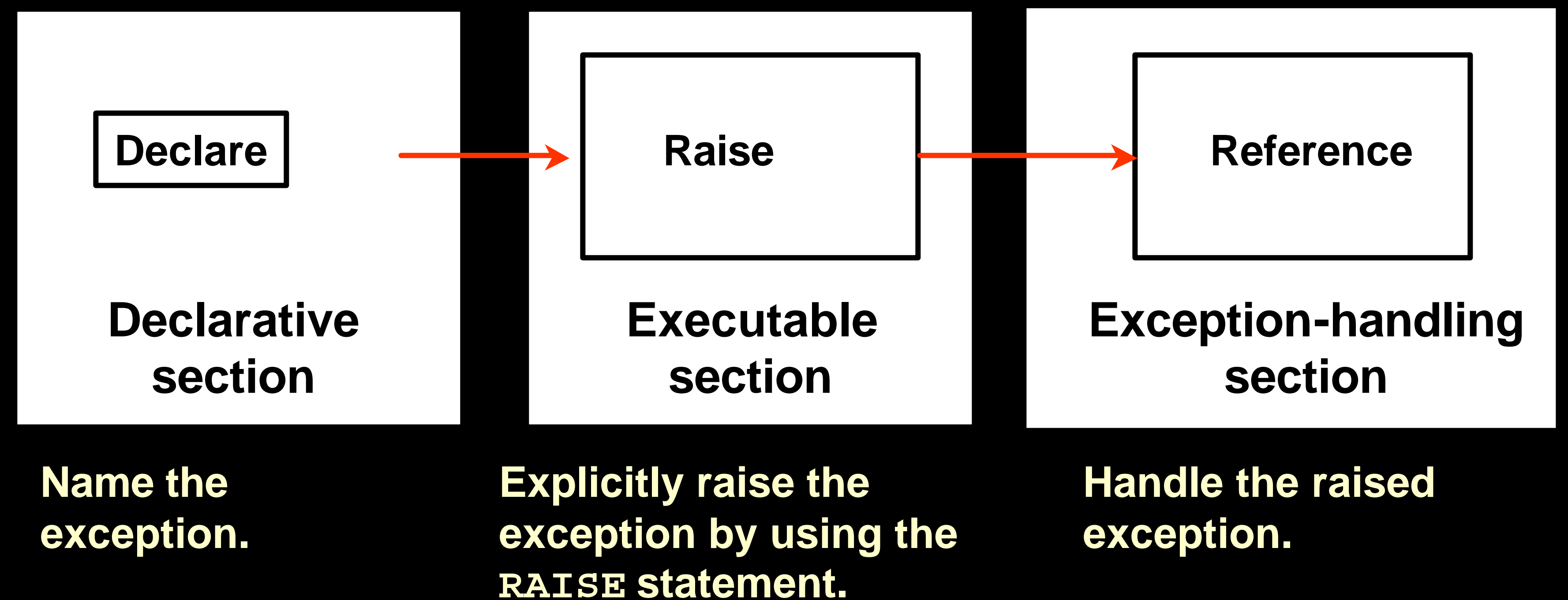
Example:

```
DECLARE
  v_error_code       NUMBER;
  v_error_message    VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code := SQLCODE ;
    v_error_message := SQLERRM ;
    INSERT INTO errors
    VALUES(v_error_code, v_error_message);
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trapping User-Defined Exceptions



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

User-Defined Exceptions

Example:

```
DEFINE p_department_desc = 'Information Technology '
DEFINE P_department_number = 300
```

```
DECLARE
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE      departments
  SET         department_name = '&p_department_desc'
  WHERE       department_id = &p_department_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

1. `e_invalid_department EXCEPTION;`
2. `RAISE e_invalid_department;`
3. `e_invalid_department`

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Calling Environments

*i*SQL*Plus	Displays error number and message to screen
Procedure Builder	Displays error number and message to screen
Oracle Developer Forms	Accesses error number and message in a trigger by means of the `ERROR_CODE` and `ERROR_TEXT` packaged functions
Precompiler application	Accesses exception number through the `SQLCA` data structure
An enclosing PL/SQL block	Traps exception in exception-handling routine of enclosing block

Copyright © Oracle Corporation, 2001. All rights reserved.

Propagating Exceptions

Subblocks can handle an exception or pass the exception to the enclosing block.

```
DECLARE
  . . .
  e_no_rows        exception;
  e_integrity      exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
     SELECT ...
     UPDATE ...
     IF SQL%NOTFOUND THEN
       RAISE e_no_rows;
     END IF;
    END;
  END LOOP;
EXCEPTION
  WHEN e_integrity THEN ...
  WHEN e_no_rows THEN ...
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The RAISE_APPLICATION_ERROR Procedure

Syntax:

```
raise_application_error (error_number,
                  message[, {TRUE | FALSE}]);
```

- You can use this procedure to issue user-defined error messages from stored subprograms.
- You can report errors to your application and avoid returning unhandled exceptions.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The RAISE_APPLICATION_ERROR Procedure

- Used in two different places:
  - Executable section
  - Exception section
- Returns error conditions to the user in a manner consistent with other Oracle server errors

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

RAISE_APPLICATION_ERROR

Executable section:

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
   ...
```

Exception section:

```
...
EXCEPTION
     WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR (-20201,
         'Manager is not a valid employee.');
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned that:

- Exception types:
  - Predefined Oracle server error
  - Nonpredefined Oracle server error
  - User-defined error
- Exception trapping
- Exception handling:
  - Trap the exception within the PL/SQL block.
  - Propagate the exception.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 8 Overview

This practice covers the following topics:

- Handling named exceptions
- Creating and invoking user-defined exceptions

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Procedures

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Distinguish anonymous PL/SQL blocks from named PL/SQL blocks (subprograms)
- Describe subprograms
- List the benefits of using subprograms
- List the different environments from which subprograms can be invoked

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

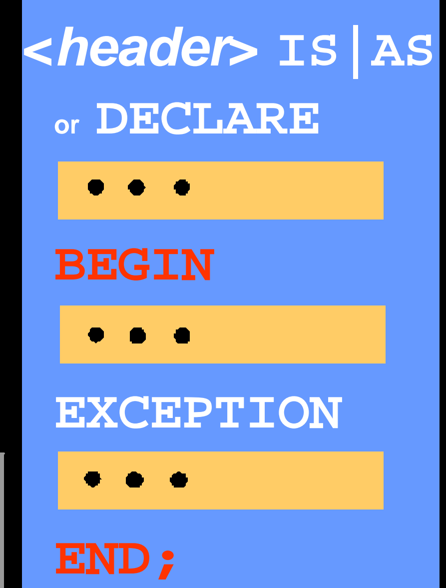
Objectives

After completing this lesson, you should be able to do the following:

- Describe PL/SQL blocks and subprograms
- Describe the uses of procedures
- Create procedures
- Differentiate between formal and actual parameters
- List the features of different parameter modes
- Create procedures with parameters
- Invoke a procedure
- Handle exceptions in procedures
- Remove a procedure

Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Program Constructs



Tools Constructs
Anonymous blocks
Application procedures or functions
Application packages
Application triggers
Object types

Database Server Constructs
Anonymous blocks
Stored procedures or functions
Stored packages
Database triggers
Object types

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of Subprograms

A subprogram:

- Is a named PL/SQL block that can accept parameters and be invoked from a calling environment
- Is of two types:
  - A procedure that performs an action
  - A function that computes a value
- Is based on standard PL/SQL block structure
- Provides modularity, reusability, extensibility, and maintainability
- Provides easy maintenance, improved data security and integrity, improved performance, and improved code clarity

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Block Structure for Anonymous PL/SQL Blocks

`DECLARE` (optional)
Declare PL/SQL objects to be used within this block

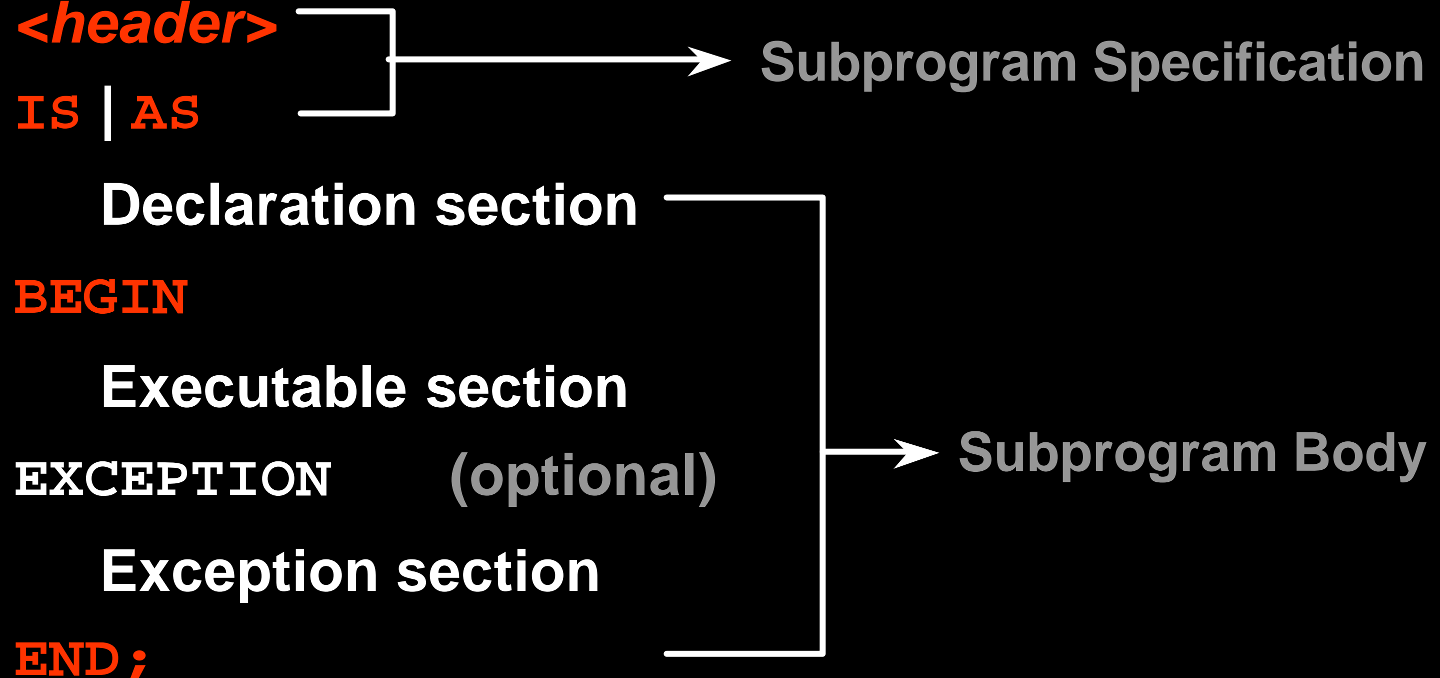
`BEGIN` (mandatory)
Define the executable statements

`EXCEPTION` (optional)
Define the actions that take place if an error or exception arises

`END;` (mandatory)

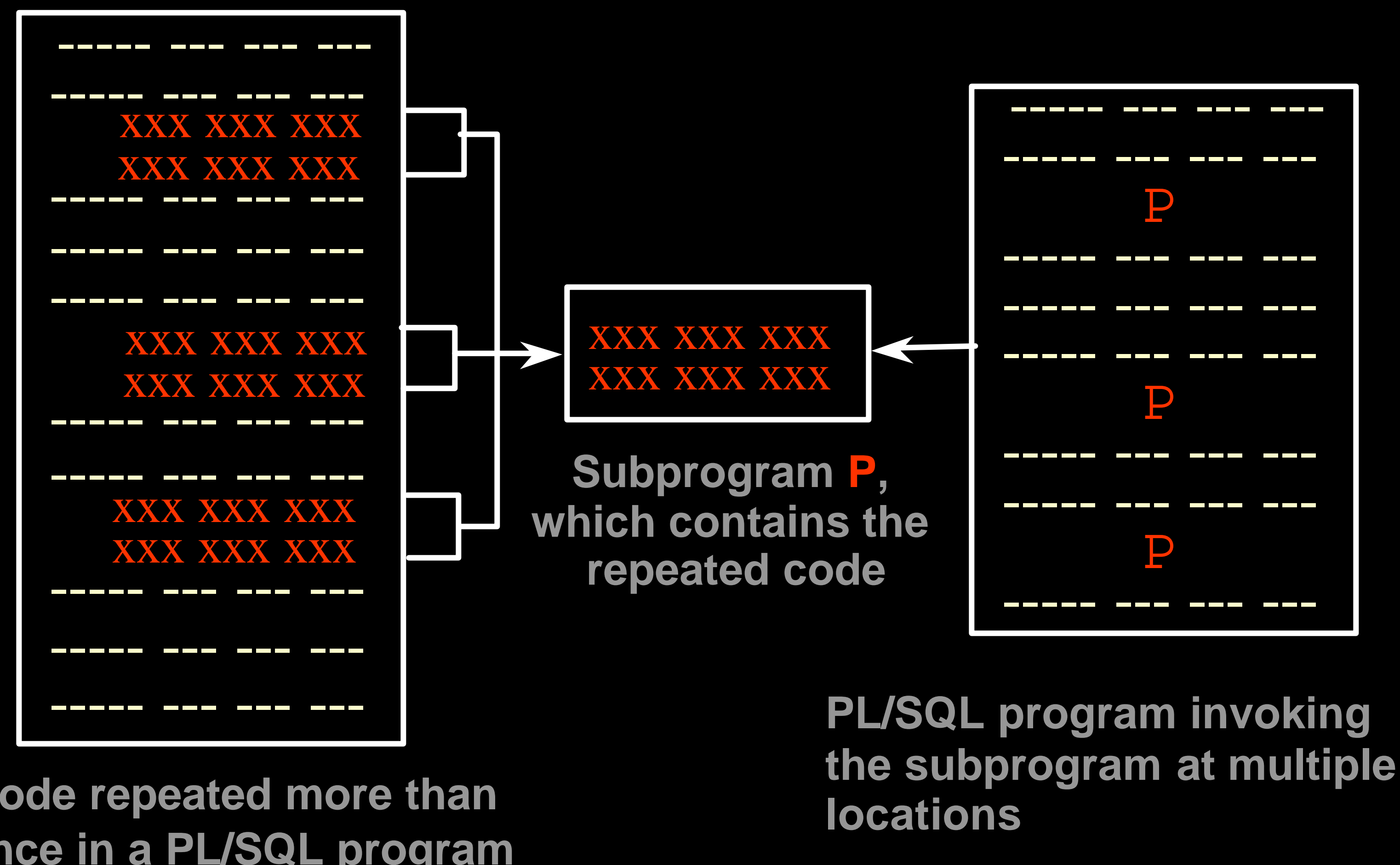
Copyright © Oracle Corporation, 2001. All rights reserved.

Block Structure for PL/SQL Subprograms



Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Subprograms



Copyright © Oracle Corporation, 2001. All rights reserved.

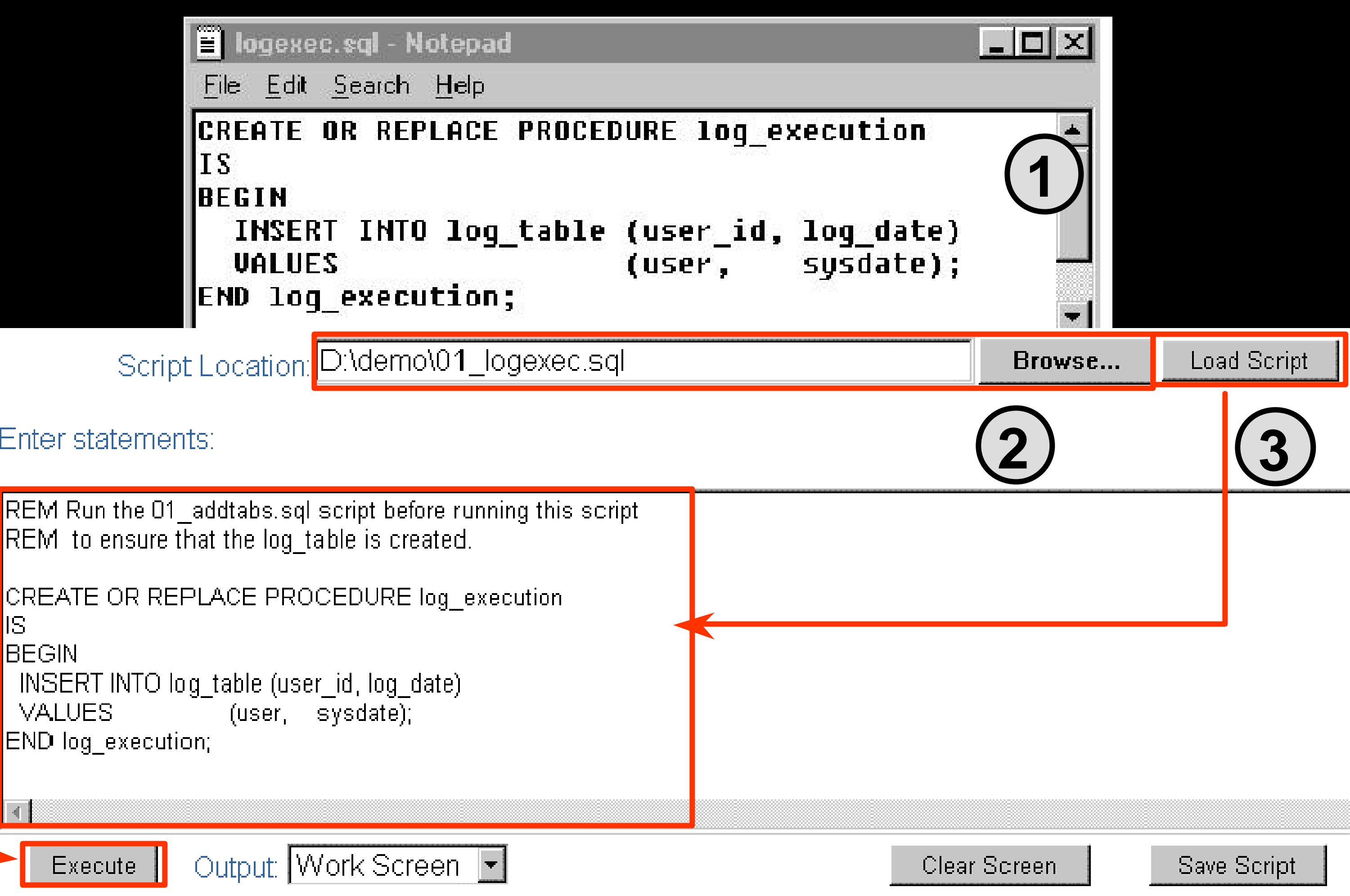
Benefits of Subprograms

- Easy maintenance
- Improved data security and integrity
- Improved performance
- Improved code clarity

ORACLE

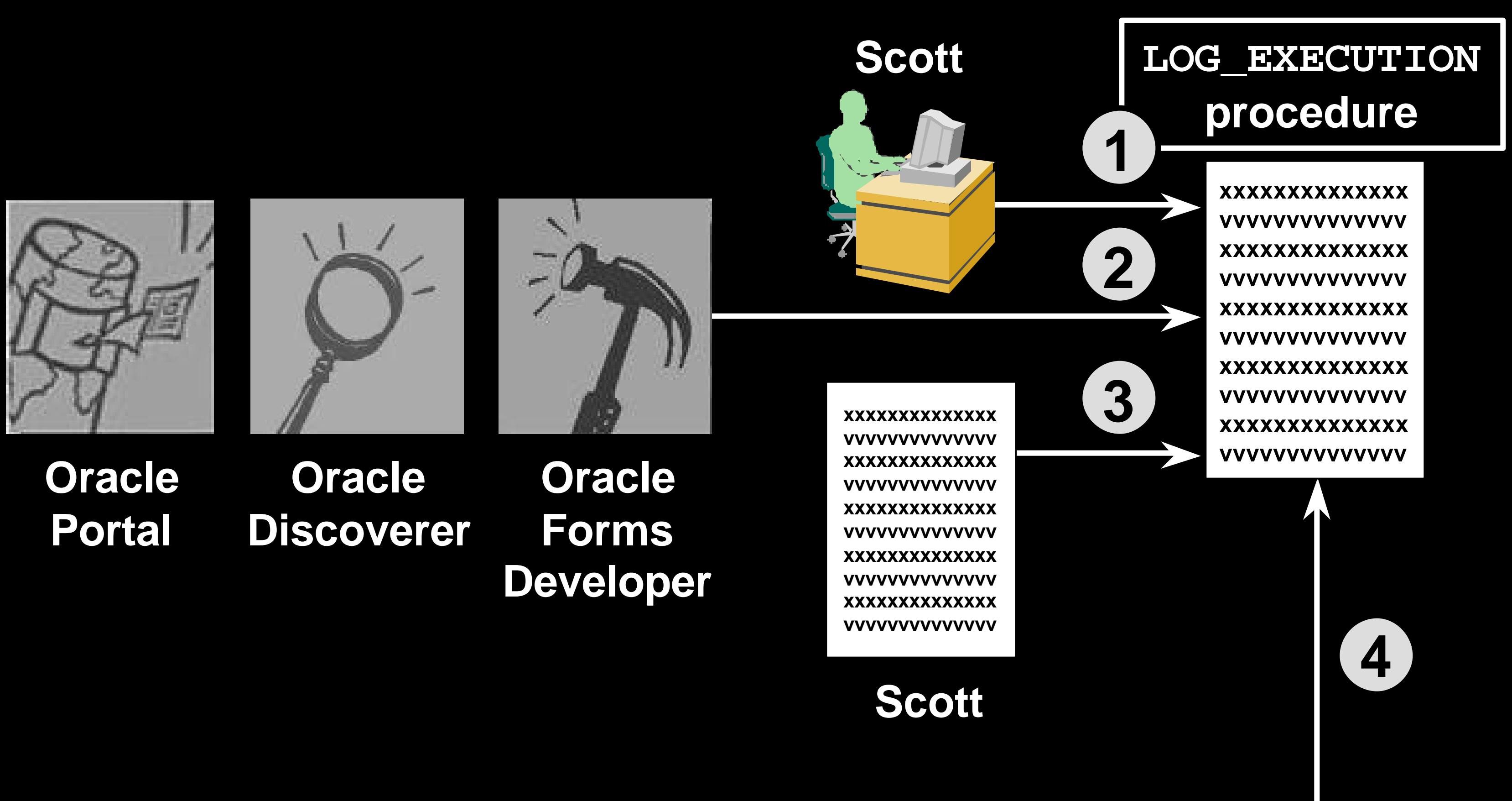
Copyright © Oracle Corporation, 2001. All rights reserved.

Developing Subprograms by Using *i*SQL*Plus



Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Stored Procedures and Functions



Copyright © Oracle Corporation, 2001. All rights reserved.

What Is a Procedure?

- A procedure is a type of subprogram that performs an action.
- A procedure can be stored in the database, as a schema object, for repeated execution.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Creating Procedures

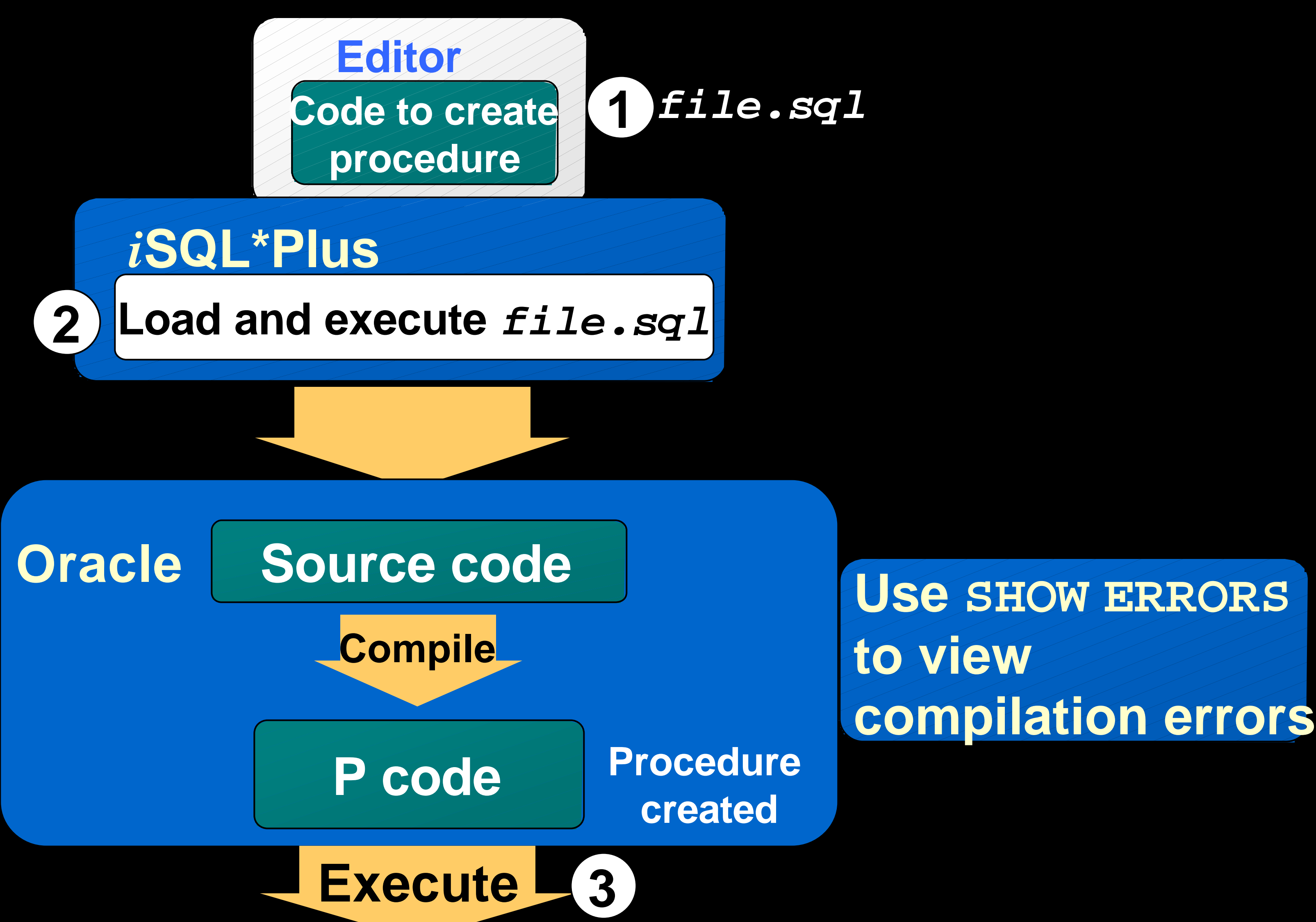
```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

- The `REPLACE` option indicates that if the procedure exists, it will be dropped and replaced with the new version created by the statement.
- PL/SQL block starts with either `BEGIN` or the declaration of local variables and ends with either `END` or `END` *procedure_name.*

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Developing Procedures



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Formal Versus Actual Parameters

- **Formal parameters: variables declared in the parameter list of a subprogram specification**

  Example:

```
CREATE PROCEDURE raise_sal(p_id NUMBER, p_amount NUMBER)
...
END raise_sal;
```

- **Actual parameters: variables or expressions referenced in the parameter list of a subprogram call**

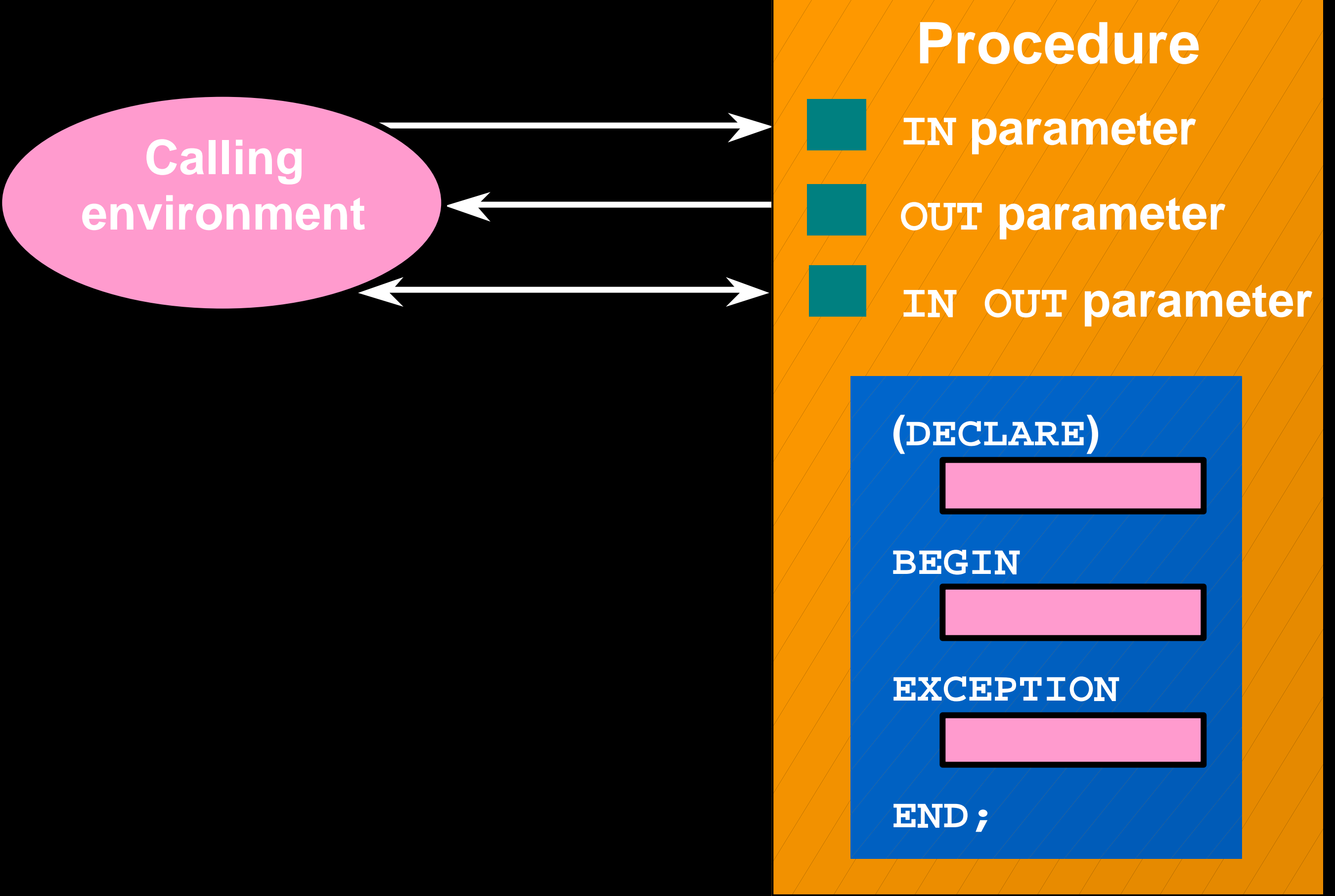
  Example:

```
raise_sal(v_id, 2000)
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Procedural Parameter Modes



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

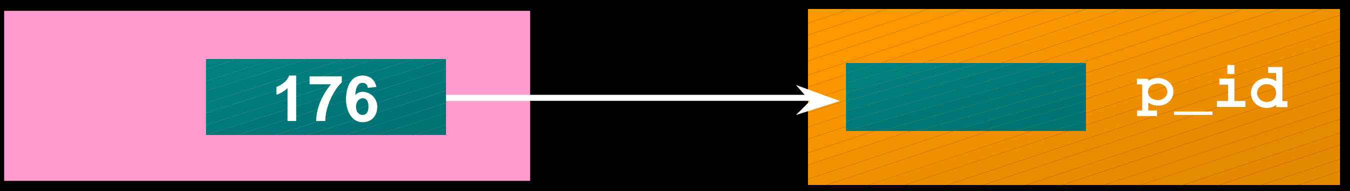
Creating Procedures with Parameters

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

IN Parameters: Example

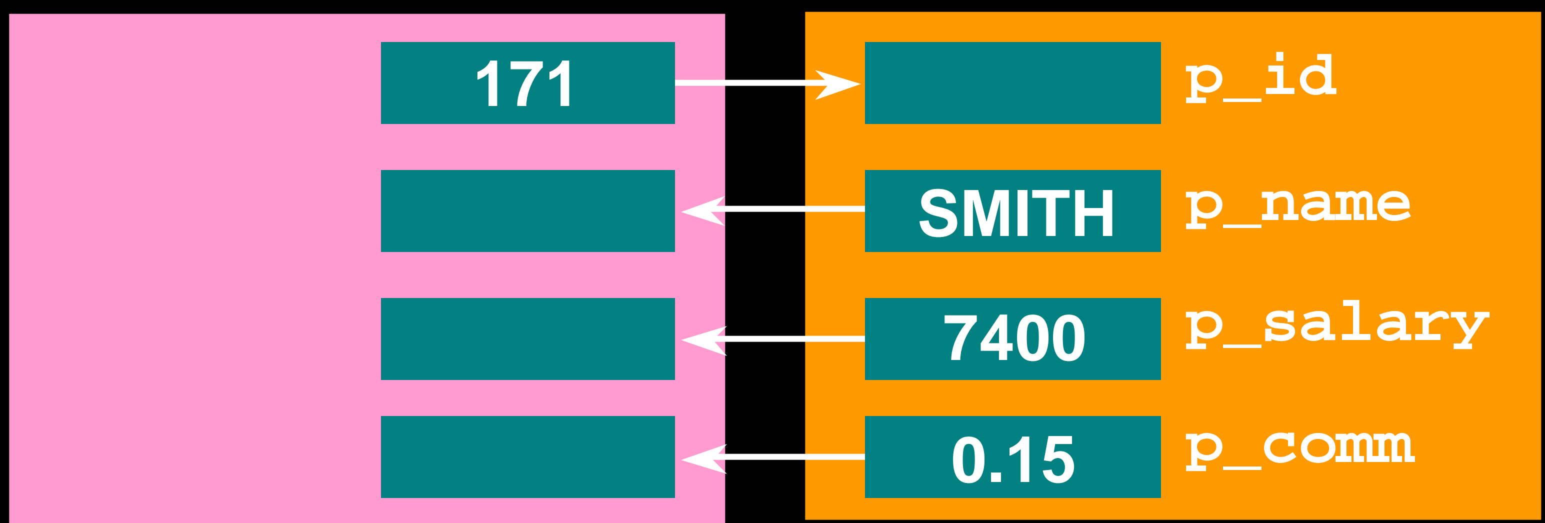



```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id IN employees.employee_id%TYPE)
IS
BEGIN
  UPDATE employees
  SET    salary = salary * 1.10
  WHERE  employee_id = p_id;
END raise_salary;
/
```

```
Procedure created.
```


Copyright © Oracle Corporation, 2001. All rights reserved.

OUT Parameters: Example



Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

OUT Parameters: Example

emp_query.sql

```
CREATE OR REPLACE PROCEDURE query_emp
  (p_id       IN   employees.employee_id%TYPE,
   p_name     OUT  employees.last_name%TYPE,
   p_salary   OUT  employees.salary%TYPE,
   p_comm     OUT  employees.commission_pct%TYPE)
IS
BEGIN
  SELECT   last_name, salary, commission_pct
   INTO    p_name, p_salary, p_comm
   FROM    employees
   WHERE   employee_id = p_id;
END query_emp;
/
```

```
Procedure created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Viewing OUT Parameters

- Load and run the `emp_query.sql` script file to create the `QUERY_EMP` procedure.
- Declare host variables, execute the `QUERY_EMP` procedure, and print the value of the global `G_NAME` variable.

```
VARIABLE g_name      VARCHAR2(25)
VARIABLE g_sal       NUMBER
VARIABLE g_comm      NUMBER

EXECUTE query_emp(171, :g_name, :g_sal, :g_comm)

PRINT g_name
```

```
PL/SQL procedure successfully completed.
```

G_NAME
Smith

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

IN OUT Parameters

Calling environment

FORMAT_PHONE procedure

'8006330575'

'(800)633-0575'

p_phone_no

```
CREATE OR REPLACE PROCEDURE format_phone
  (p_phone_no IN OUT VARCHAR2)
IS
BEGIN
  p_phone_no := '(' || SUBSTR(p_phone_no,1,3) ||
                ')' || SUBSTR(p_phone_no,4,3) ||
                '-' || SUBSTR(p_phone_no,7);
END format_phone;
/
```

```
Procedure created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Viewing IN OUT Parameters

```
VARIABLE g_phone_no VARCHAR2(15)
BEGIN
  :g_phone_no := '8006330575';
END;
/
PRINT g_phone_no
EXECUTE format_phone (:g_phone_no)
PRINT g_phone_no
```

PL/SQL procedure successfully completed.

G_PHONE_NO
8006330575

PL/SQL procedure successfully completed.

G_PHONE_NO
(800)633-0575

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Methods for Passing Parameters

- Positional: List actual parameters in the same order as formal parameters.
- Named: List actual parameters in arbitrary order by associating each with its corresponding formal parameter.
- Combination: List some of the actual parameters as positional and some as named.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DEFAULT Option for Parameters

```
CREATE OR REPLACE PROCEDURE add_dept
  (p_name   IN departments.department_name%TYPE
                                  DEFAULT 'unknown',
   p_loc    IN departments.location_id%TYPE
                                  DEFAULT 1700)
IS
BEGIN
  INSERT INTO departments(department_id,
              department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
/
```

```
Procedure created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Examples of Passing Parameters

```
BEGIN
  add_dept;
  add_dept ('TRAINING', 2500);
  add_dept ( p_loc => 2400, p_name =>'EDUCATION');
  add_dept ( p_loc => 1200) ;
END;
/
SELECT department_id, department_name, location_id
FROM departments;
```

PL/SQL procedure successfully completed.

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
40	Human Resources	2400
...		
290	TRAINING	2500
300	EDUCATION	2400
310	unknown	1200

31 rows selected.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring Subprograms

leave_emp2.sql

```
CREATE OR REPLACE PROCEDURE leave_emp2
  (p_id  IN  employees.employee_id%TYPE)
IS
  PROCEDURE log_exec
  IS
  BEGIN
    INSERT INTO log_table (user_id, log_date)
    VALUES (USER, SYSDATE);
  END log_exec;
BEGIN
  DELETE FROM employees
  WHERE employee_id = p_id;
  log_exec;
END leave_emp2;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking a Procedure from an Anonymous PL/SQL Block

```
DECLARE
  v_id NUMBER := 163;
BEGIN
  raise_salary(v_id);     --invoke procedure
  COMMIT;
...
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking a Procedure from Another Procedure

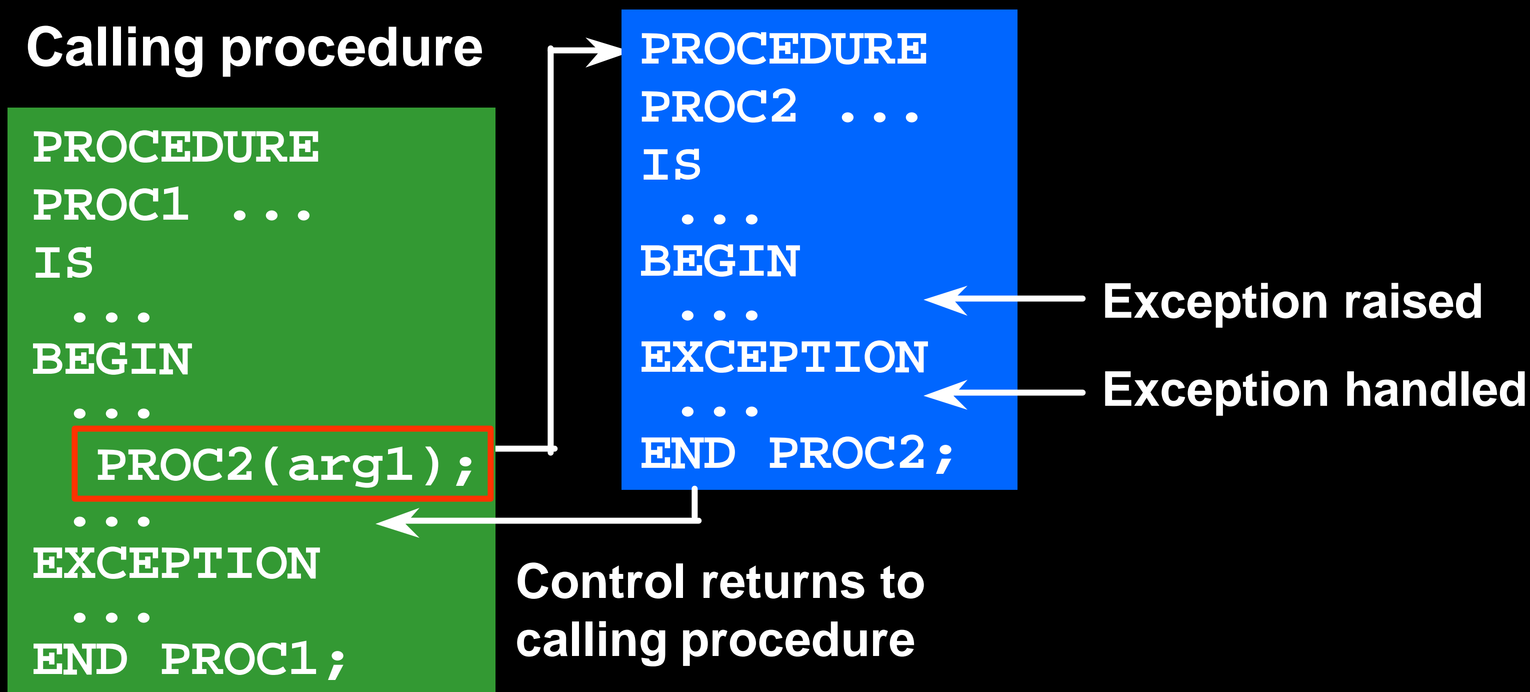
process_emps.sql

```
CREATE OR REPLACE PROCEDURE process_emps
IS
   CURSOR emp_cursor IS
    SELECT employee_id
    FROM   employees;
BEGIN
   FOR emp_rec IN emp_cursor
   LOOP
    raise_salary(emp_rec.employee_id);
   END LOOP;
   COMMIT;
END process_emps;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Handled Exceptions



Copyright © Oracle Corporation, 2001. All rights reserved.

Handled Exceptions

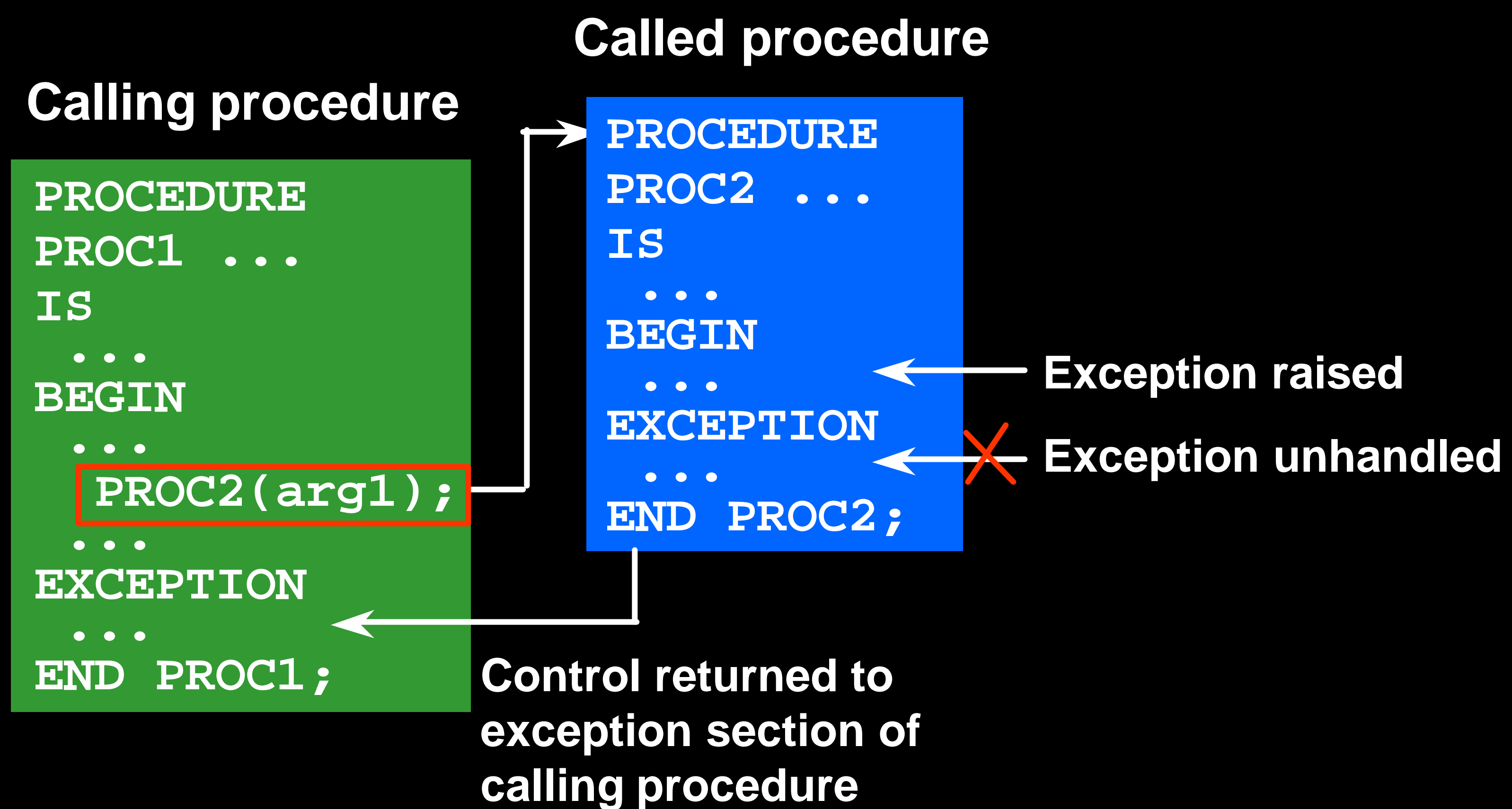
```
CREATE PROCEDURE p2_ins_dept(p_locid NUMBER) IS
 v_did NUMBER(4);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Procedure p2_ins_dept started');
 INSERT INTO departments VALUES (5, 'Dept 5', 145, p_locid);
 SELECT department_id INTO v_did FROM employees
  WHERE employee_id = 999;
END;
```

```
CREATE PROCEDURE p1_ins_loc(p_lid NUMBER, p_city VARCHAR2)
IS
 v_city VARCHAR2(30); v_dname VARCHAR2(30);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Main Procedure p1_ins_loc');
 INSERT INTO locations (location_id, city) VALUES (p_lid, p_city);
 SELECT city INTO v_city FROM locations WHERE location_id = p_lid;
 DBMS_OUTPUT.PUT_LINE('Inserted city '||v_city);
 DBMS_OUTPUT.PUT_LINE('Invoking the procedure p2_ins_dept ...');
 p2_ins_dept(p_lid);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('No such dept/loc for any employee');
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Unhandled Exceptions



Copyright © Oracle Corporation, 2001. All rights reserved.

Unhandled Exceptions

```
CREATE PROCEDURE p2_noexcep(p_locid NUMBER) IS
 v_did NUMBER(4);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Procedure p2_noexcep started');
 INSERT INTO departments VALUES (6, 'Dept 6', 145, p_locid);
 SELECT department_id INTO v_did FROM employees
  WHERE employee_id = 999;
END;
```

```
CREATE PROCEDURE p1_noexcep(p_lid NUMBER, p_city VARCHAR2)
IS
 v_city VARCHAR2(30); v_dname VARCHAR2(30);
BEGIN
 DBMS_OUTPUT.PUT_LINE(' Main Procedure p1_noexcep');
 INSERT INTO locations (location_id, city) VALUES (p_lid, p_city);
 SELECT city INTO v_city FROM locations WHERE location_id = p_lid;
 DBMS_OUTPUT.PUT_LINE('Inserted new city '||v_city);
 DBMS_OUTPUT.PUT_LINE('Invoking the procedure p2_noexcep ...');
 p2_noexcep(p_lid);
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Procedures

Drop a procedure stored in the database.

Syntax:

```
DROP PROCEDURE procedure_name
```

Example:

```
DROP PROCEDURE raise_salary;
```

```
Procedure dropped.
```


Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

Summary

In this lesson, you should have learned that:

- A procedure is a subprogram that performs an action.
- You create procedures by using the `CREATE PROCEDURE` command.
- You can compile and save a procedure in the database.
- Parameters are used to pass data from the calling environment to the procedure.
- There are three parameter modes: `IN`, `OUT`, and `IN OUT`.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

- Local subprograms are programs that are defined within the declaration section of another program.
- Procedures can be invoked from any tool or language that supports PL/SQL.
- You should be aware of the effect of handled and unhandled exceptions on transactions and calling procedures.
- You can remove procedures from the database by using the `DROP PROCEDURE` command.
- Procedures can serve as building blocks for an application.

ORACLE

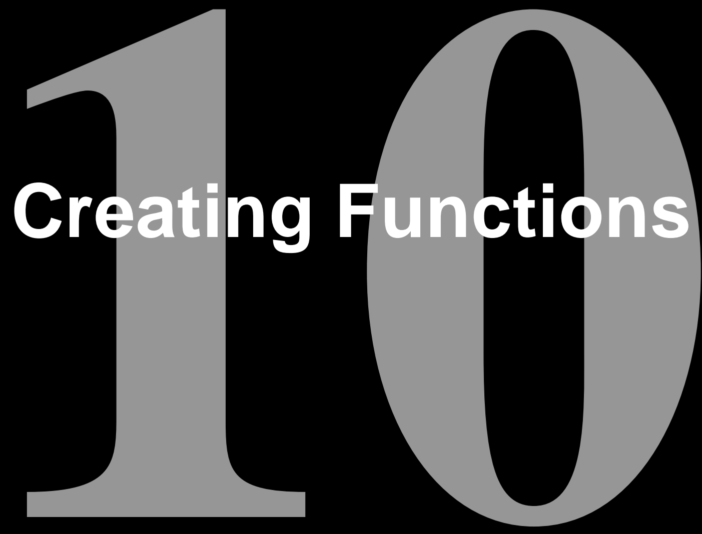
Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 9 Overview

This practice covers the following topics:

- Creating stored procedures to:
  - Insert new rows into a table, using the supplied parameter values
  - Update data in a table for rows matching with the supplied parameter values
  - Delete rows from a table that match the supplied parameter values
  - Query a table and retrieve data based on supplied parameter values
- Handling exceptions in procedures
- Compiling and invoking procedures

Copyright © Oracle Corporation, 2001. All rights reserved.



Creating Functions

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the uses of functions
- Create stored functions
- Invoke a function
- Remove a function
- Differentiate between a procedure and a function

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of Stored Functions

- A function is a named PL/SQL block that returns a value.
- A function can be stored in the database as a schema object for repeated execution.
- A function is called as part of an expression.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Creating Functions

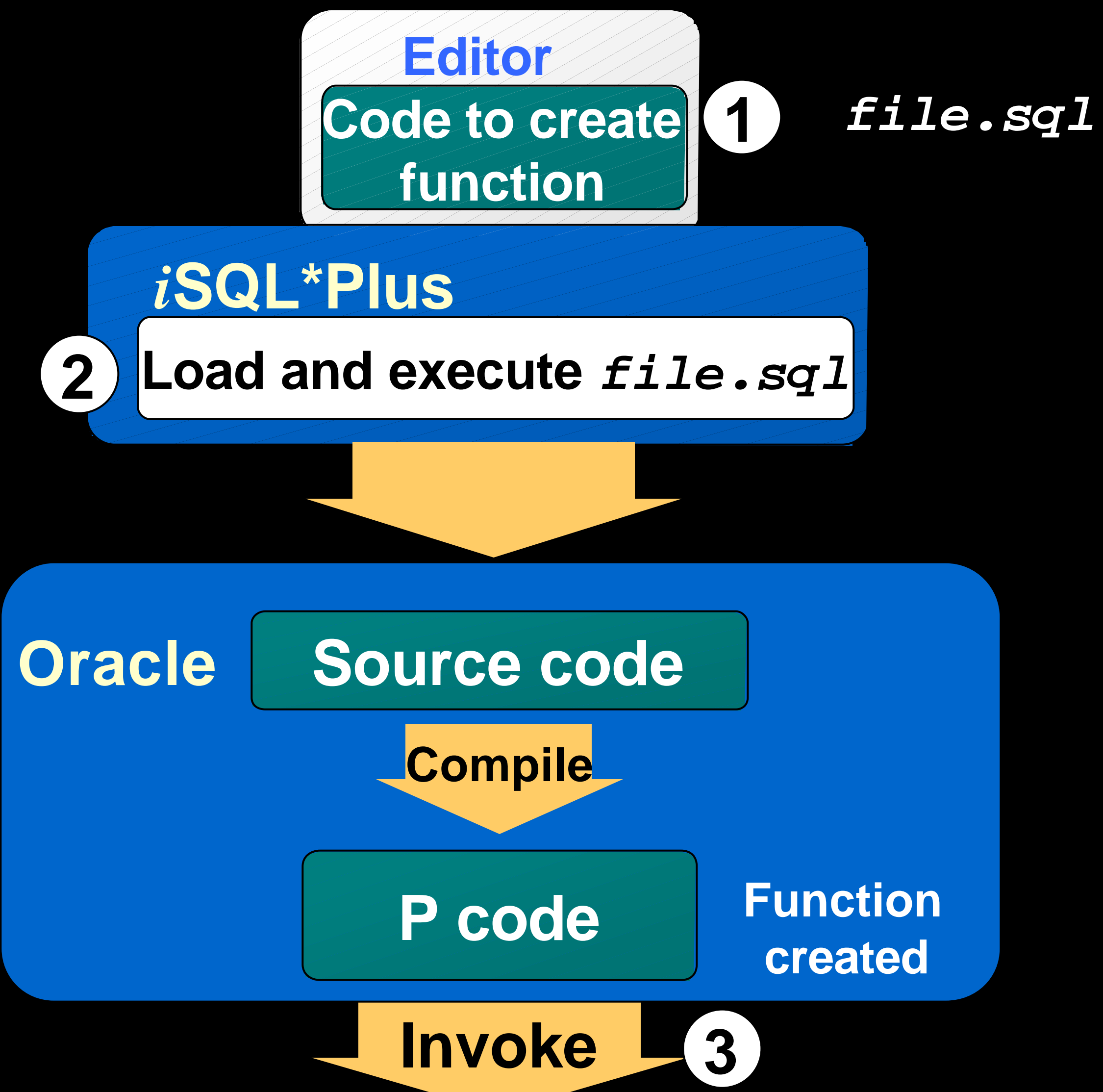
```
CREATE [OR REPLACE] FUNCTION function_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
PL/SQL Block;
```

The PL/SQL block must have at least one `RETURN` statement.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Function



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Stored Function by Using *i*SQL*Plus

1. Enter the text of the `CREATE FUNCTION` statement in an editor and save it as a SQL script file.
2. Run the script file to store the source code and compile the function.
3. Use `SHOW ERRORS` to see compilation errors.
4. When successfully compiled, invoke the function.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Stored Function by Using *i*SQL*Plus: Example

get_salary.sql

```
CREATE OR REPLACE FUNCTION get_sal
      (p_id   IN employees.employee_id%TYPE)
      RETURN NUMBER
IS
      v_salary employees.salary%TYPE :=0;
BEGIN
      SELECT salary
      INTO   v_salary
      FROM   employees
      WHERE  employee_id = p_id;
     RETURN v_salary;
END get_sal;
/
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

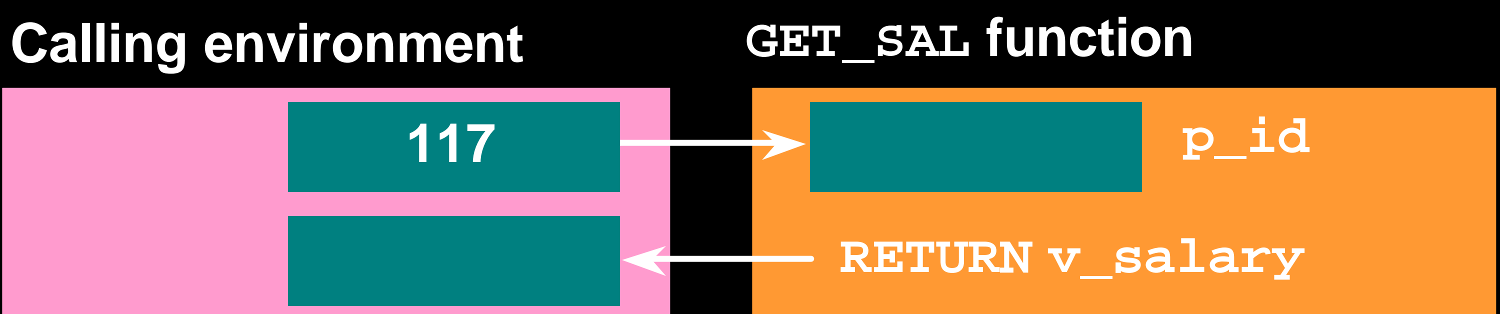
Executing Functions

- Invoke a function as part of a PL/SQL expression.
- Create a variable to hold the returned value.
- Execute the function. The variable will be populated by the value returned through a `RETURN` statement.

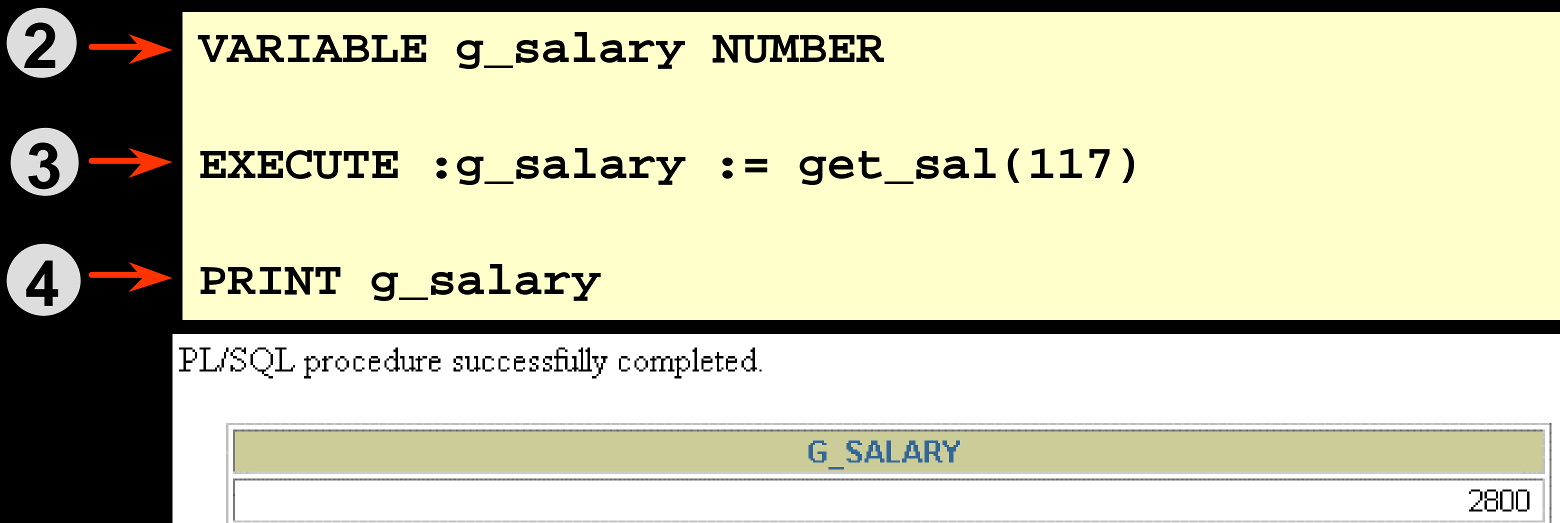
ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Executing Functions: Example



1. Load and run the `get_salary.sql` file to create the function



 Copyright © Oracle Corporation, 2001. All rights reserved.

Advantages of User-Defined Functions in SQL Expressions

- Extend SQL where activities are too complex, too awkward, or unavailable with SQL
- Can increase efficiency when used in the `WHERE` clause to filter data, as opposed to filtering the data in the application
- Can manipulate character strings

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Functions in SQL Expressions: Example

```
CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
 RETURN NUMBER IS
BEGIN
   RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;
```

Function created.

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
108	Greenberg	12000	960
109	Faviet	9000	720
110	Chen	8200	656
111	Sciarra	7700	616
112	Urman	7800	624
113	Popp	6900	552

6 rows selected.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Locations to Call User-Defined Functions

- Select list of a `SELECT` command
- Condition of the `WHERE` and `HAVING` clauses
- `CONNECT BY`, `START WITH`, `ORDER BY`, and `GROUP BY` clauses
- `VALUES` clause of the `INSERT` command
- `SET` clause of the `UPDATE` command

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Calling Functions from SQL Expressions

To be callable from SQL expressions, a user-defined function must:

- Be a stored function
- Accept only `IN` parameters
- Accept only valid SQL data types, not PL/SQL specific types, as parameters
- Return data types that are valid SQL data types, not PL/SQL specific types

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Calling Functions from SQL Expressions

- Functions called from SQL expressions cannot contain DML statements.
- Functions called from `UPDATE/DELETE` statements on a table T cannot contain DML on the same table T.
- Functions called from an `UPDATE` or a `DELETE` statement on a table T cannot query the same table.
- Functions called from SQL statements cannot contain statements that end the transactions.
- Calls to subprograms that break the previous restriction are not allowed in the function.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Calling from SQL

```
CREATE OR REPLACE FUNCTION dml_call_sql (p_sal NUMBER)
   RETURN NUMBER IS
BEGIN
   INSERT INTO employees(employee_id, last_name, email,
                         hire_date, job_id, salary)
         VALUES(1, 'employee 1', 'emp1@company.com',
                    SYSDATE, 'SA_MAN', 1000);
   RETURN (p_sal + 100);
END;
/
```

```
Function created.
```

```
UPDATE employees SET salary = dml_call_sql(2000)
 WHERE employee_id = 170;
```

```
UPDATE employees SET salary = dml_call_sql(2000)
                     *
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "PLSQL.DML_CALL_SQL", line 4
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Functions

Drop a stored function.

Syntax:

```
DROP FUNCTION function_name
```

Example:

```
DROP FUNCTION get_sal;
```

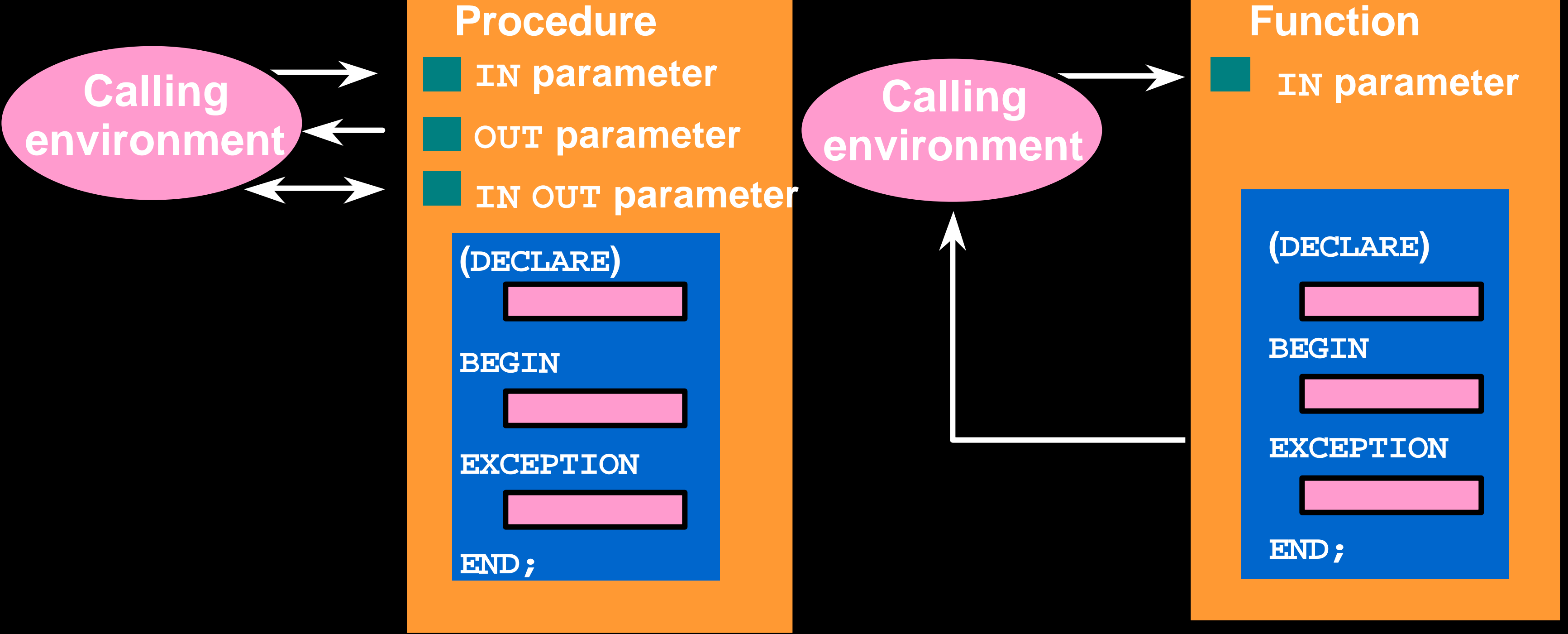
```
Function dropped.
```

- All the privileges granted on a function are revoked when the function is dropped.
- The `CREATE OR REPLACE` syntax is equivalent to dropping a function and recreating it. Privileges granted on the function remain the same when this syntax is used.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Procedure or Function?



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Comparing Procedures and Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain `RETURN` clause in the header	Must contain a `RETURN` clause in the header
Can return none, one, or many values	Must return a single value
Can contain a `RETURN` statement	Must contain at least one `RETURN` statement

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of Stored Procedures and Functions

- Improved performance
- Easy maintenance
- Improved data security and integrity
- Improved code clarity

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned that:

- A function is a named PL/SQL block that must return a value.
- A function is created by using the `CREATE FUNCTION` syntax.
- A function is invoked as part of an expression.
- A function stored in the database can be called in SQL statements.
- A function can be removed from the database by using the `DROP FUNCTION` syntax.
- Generally, you use a procedure to perform an action and a function to compute a value.

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 10 Overview

This practice covers the following topics:

- Creating stored functions
  - To query a database table and return specific values
  - To be used in a SQL statement
  - To insert a new row, with specified parameter values, into a database table
  - Using default parameter values
- Invoking a stored function from a SQL statement
- Invoking a stored function from a stored procedure

Copyright © Oracle Corporation, 2001. All rights reserved.

11

Managing Subprograms

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

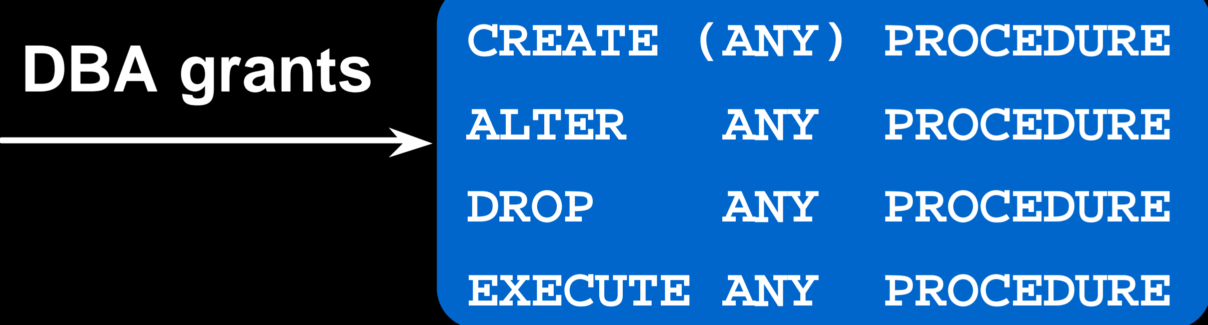
After completing this lesson, you should be able to do the following:

- Contrast system privileges with object privileges
- Contrast invokers rights with definers rights
- Identify views in the data dictionary to manage stored objects
- Describe how to debug subprograms by using the `DBMS_OUTPUT` package

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Required Privileges



To be able to refer and access objects from a different schema in a subprogram, you must be granted access to the referred objects explicitly, not through a role.

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

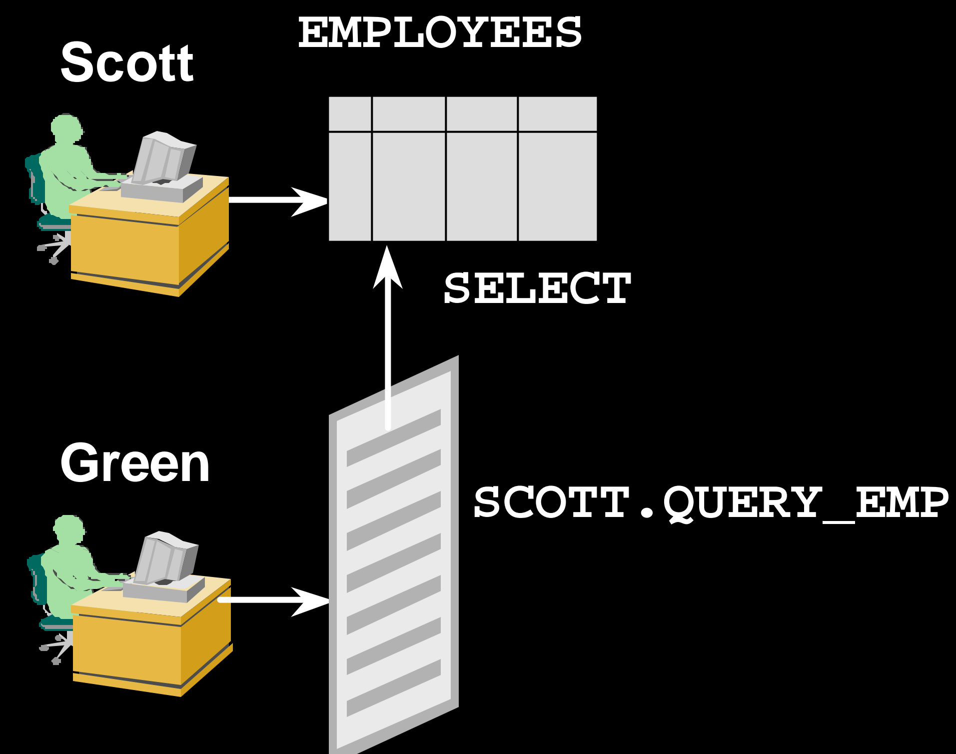
Granting Access to Data

Direct access:

```
GRANT SELECT
 ON   employees
 TO   scott;
Grant Succeeded.
```

Indirect access:

```
GRANT EXECUTE
 ON   query_emp
 TO   green;
Grant Succeeded.
```



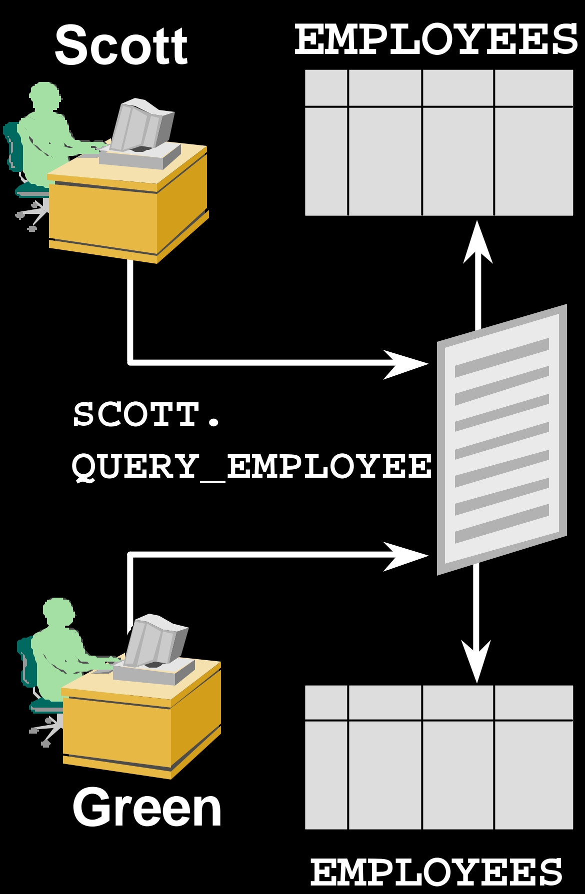
The procedure executes with the privileges of the owner (default).

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Using Invoker's-Rights

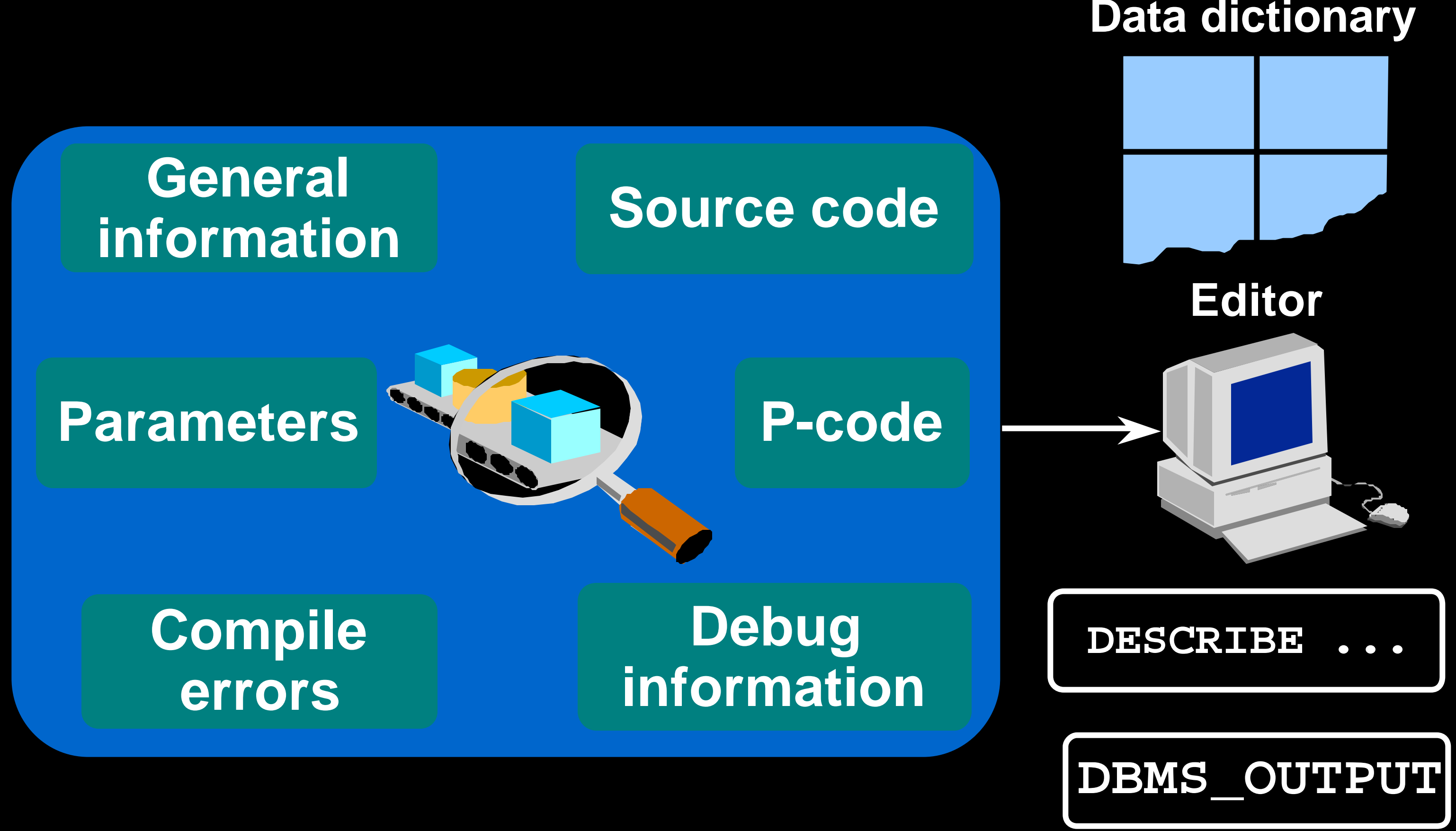
The procedure executes with the privileges of the user.

```
CREATE PROCEDURE query_employee
(p_id IN employees.employee_id%TYPE,
 p_name OUT employees.last_name%TYPE,
 p_salary OUT employees.salary%TYPE,
 p_comm OUT
   employees.commission_pct%TYPE)
AUTHID CURRENT_USER
IS
BEGIN
  SELECT last_name, salary,
         commission_pct
   INTO  p_name, p_salary, p_comm
   FROM  employees
   WHERE employee_id=p_id;
END query_employee;
/
```



Copyright © Oracle Corporation, 2001. All rights reserved.

Managing Stored PL/SQL Objects



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

USER_OBJECTS

Column	Column Description
OBJECT_NAME	Name of the object
OBJECT_ID	Internal identifier for the object
OBJECT_TYPE	Type of object, for example, TABLE, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER
CREATED	Date when the object was created
LAST_DDL_TIME	Date when the object was last modified
TIMESTAMP	Date and time when the object was last recompiled
STATUS	VALID or INVALID

*Abridged column list

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

List All Procedures and Functions

```
SELECT object_name, object_type
FROM   user_objects
WHERE object_type in ('PROCEDURE','FUNCTION')
ORDER BY object_name;
```

OBJECT_NAME	OBJECT_TYPE
ADD_DEPT	PROCEDURE
ADD_JOB	PROCEDURE
ADD_JOB_HISTORY	PROCEDURE
ANNUAL_COMP	FUNCTION
DEL_JOB	PROCEDURE
DML_CALL_SQL	FUNCTION
…	
TAX	FUNCTION
UPD_JOB	PROCEDURE
VALID_DEPTID	FUNCTION

24 rows selected.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

USER_SOURCE Data Dictionary View

Column	Column Description
NAME	Name of the object
TYPE	Type of object, for example, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY
LINE	Line number of the source code
TEXT	Text of the source code line

Copyright © Oracle Corporation, 2001. All rights reserved.

List the Code of Procedures and Functions

```
SELECT  text
FROM  user_source
WHERE  name = 'QUERY_EMPLOYEE'
ORDER BY  line;
```

TEXT
PROCEDURE query_employee
(p_id IN employees.employee_id%TYPE, p_name OUT employees.last_name%TYPE,
p_salary OUT employees.salary%TYPE, p_comm OUT employees.commission_pct%TYPE)
AUTHID CURRENT_USER
IS
BEGIN
SELECT last_name, salary, commission_pct
INTO p_name,p_salary,p_comm
FROM employees
WHERE employee_id=p_id;
END query_employee;

11 rows selected.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

USER_ERRORS

Column	Column Description
NAME	Name of the object
TYPE	Type of object, for example, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER
SEQUENCE	Sequence number, for ordering
LINE	Line number of the source code at which the error occurs
POSITION	Position in the line at which the error occurs
TEXT	Text of the error message

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Detecting Compilation Errors: Example

```
CREATE OR REPLACE PROCEDURE log_execution
IS
BEGIN
INPUT INTO log_table (user_id, log_date)
                                        -- wrong
VALUES (USER, SYSDATE);
END;
/
```

```
Warning: Procedure created with compilation errors.
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

List Compilation Errors by Using USER_ERRORS

```
SELECT line || '/' || position POS, text
FROM   user_errors
WHERE  name = 'LOG_EXECUTION'
ORDER BY line;
```

POS	TEXT
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of the following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignored.
6/1	PLS-00103: Encountered the symbol "END"

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

List Compilation Errors by Using SHOW ERRORS

```
SHOW ERRORS PROCEDURE log_execution
```

Errors for PROCEDURE LOG_EXECUTION:

LINE/COL	ERROR
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of th e following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignore d.
6/1	PLS-00103: Encountered the symbol "END"

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DESCRIBE in *i*SQL*Plus

```
DESCRIBE query_employee
DESCRIBE add_dept
DESCRIBE tax
```

PROCEDURE **QUERY_EMPLOYEE**

Argument Name	Type	In/Out	Default?
P_ID	NUMBER(6)	IN	
P_NAME	VARCHAR2(25)	OUT	
P_SALARY	NUMBER(8,2)	OUT	
P_COMM	NUMBER(2,2)	OUT	

PROCEDURE ADD_DEPT

Argument Name	Type	In/Out	Default?
P_NAME	VARCHAR2(30)	IN	DEFAULT
P_LOC	NUMBER(4)	IN	DEFAULT

FUNCTION TAX RETURNS NUMBER

Argument Name	Type	In/Out	Default?
P_VALUE	NUMBER	IN	

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

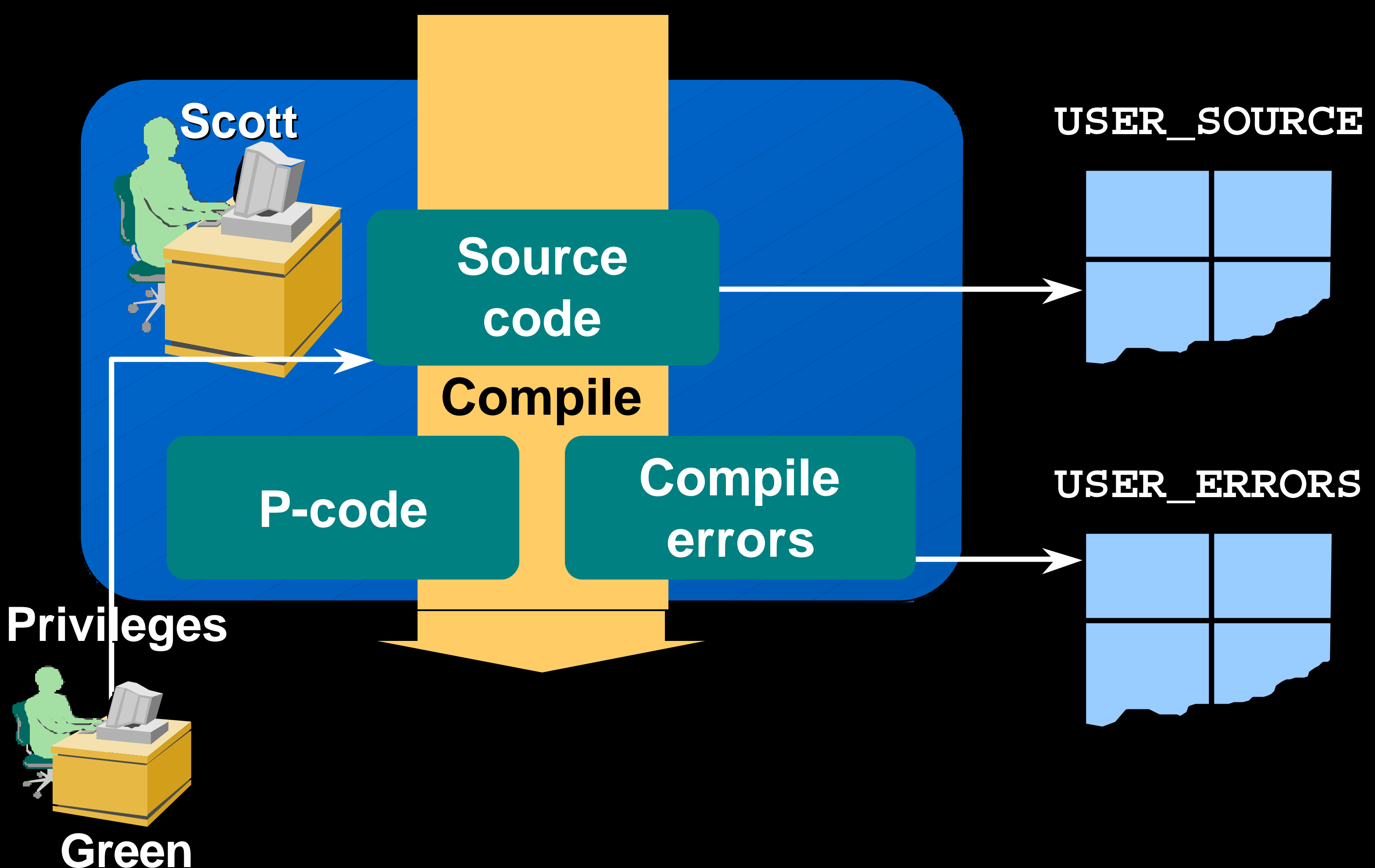
Debugging PL/SQL Program Units

- The `DBMS_OUTPUT` package:
  - Accumulates information into a buffer
  - Allows retrieval of the information from the buffer
- Autonomous procedure calls (for example, writing the output to a log table)
- Software that uses `DBMS_DEBUG`
  - Procedure Builder
  - Third-party debugging software

ORACLE

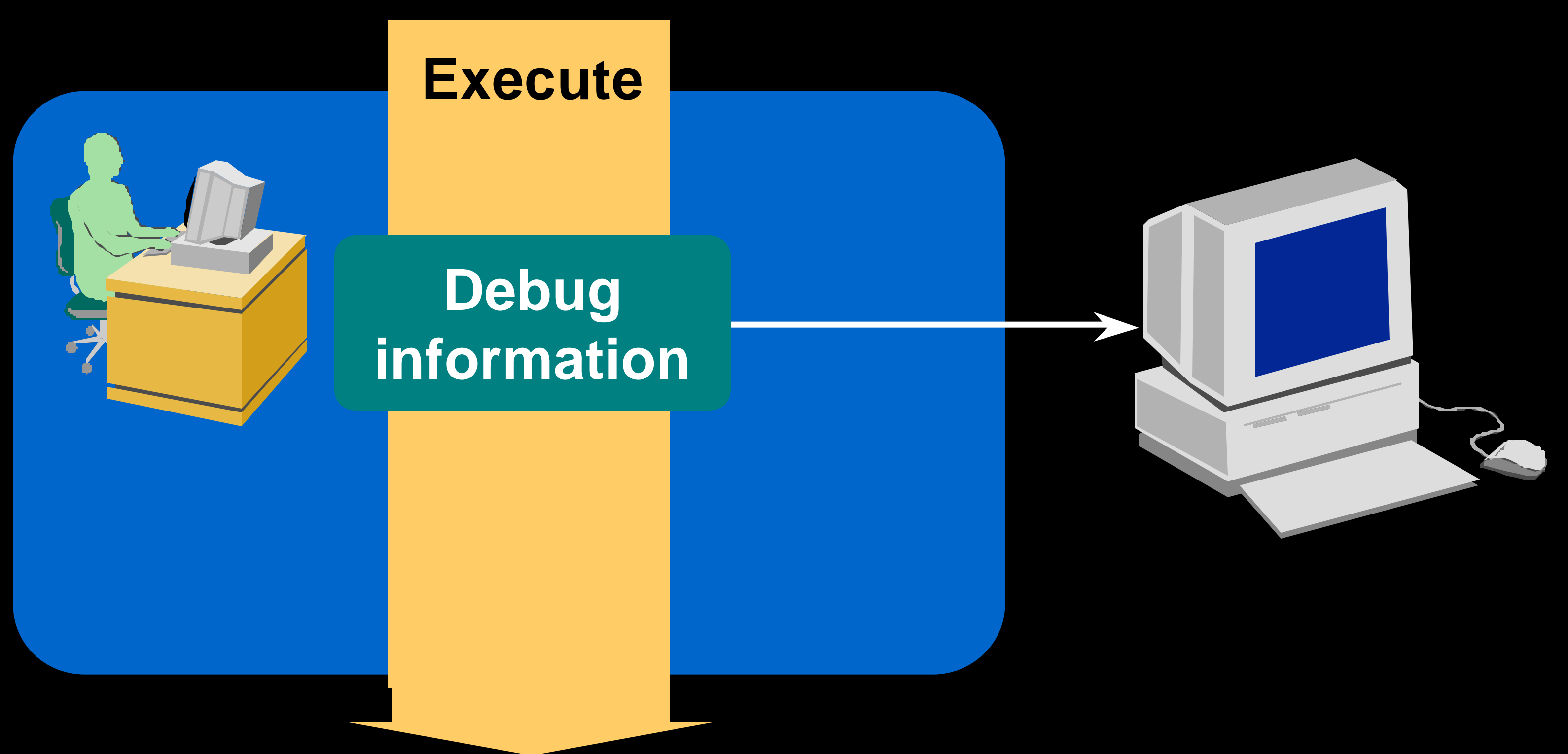
Copyright © Oracle Corporation, 2001. All rights reserved.

Summary



Copyright © Oracle Corporation, 2001. All rights reserved.

Summary



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 11 Overview

This practice covers the following topics:

- Re-creating the source file for a procedure
- Re-creating the source file for a function

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Packages

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe packages and list their possible components
- Create a package to group together related variables, cursors, constants, exceptions, procedures, and functions
- Designate a package construct as either public or private
- Invoke a package construct
- Describe a use for a bodiless package

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of Packages

Packages:

- Group logically related PL/SQL types, items, and subprograms
- Consist of two parts:
  - Specification
  - Body
- Cannot be invoked, parameterized, or nested
- Allow the Oracle server to read multiple objects into memory at once

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Components of a Package

Package specification

Procedure A declaration

Public variable

Public procedure

Package body

Procedure B definition

Private variable

Private procedure

Procedure A definition

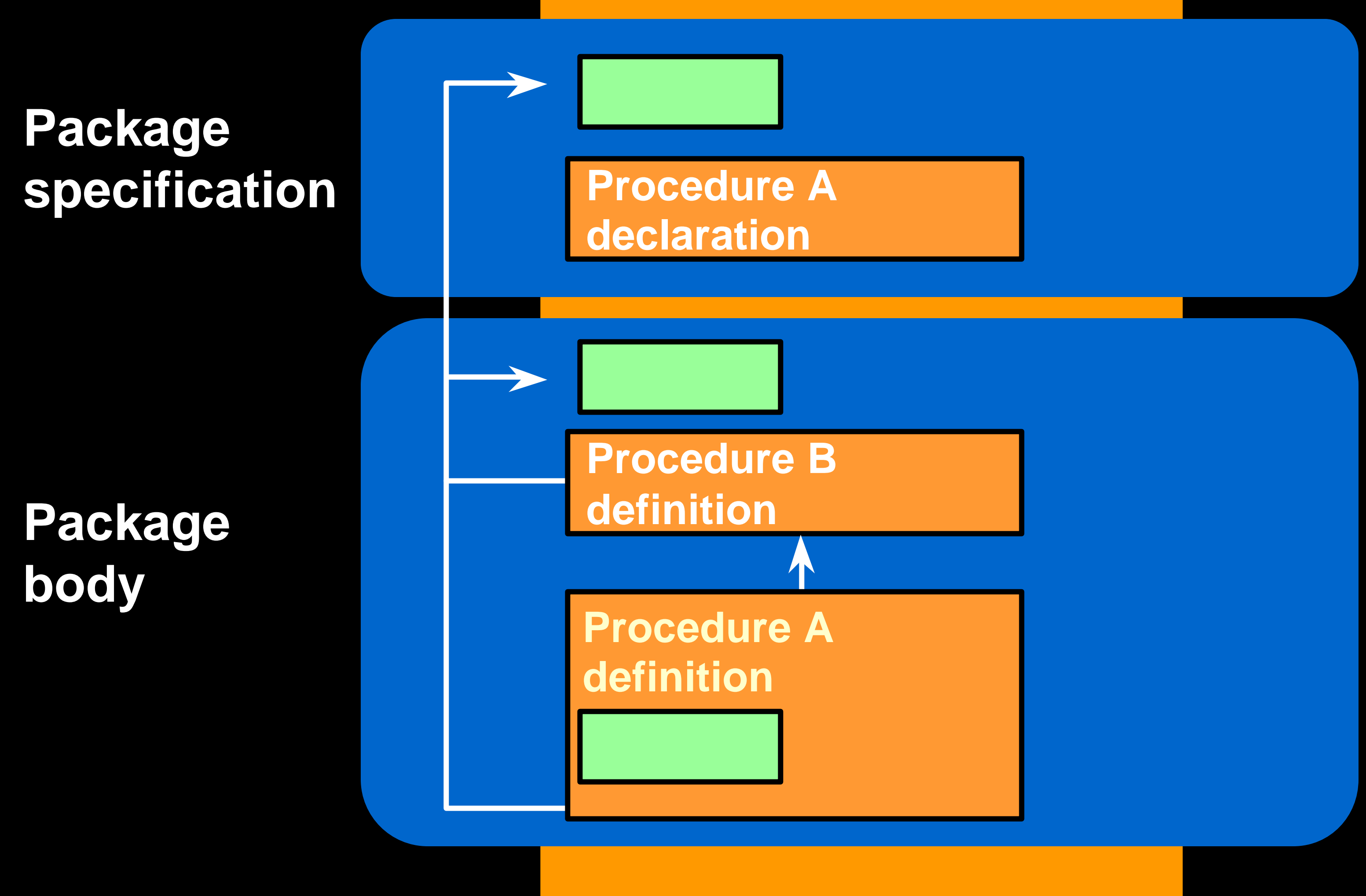
Public procedure

Local variable

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

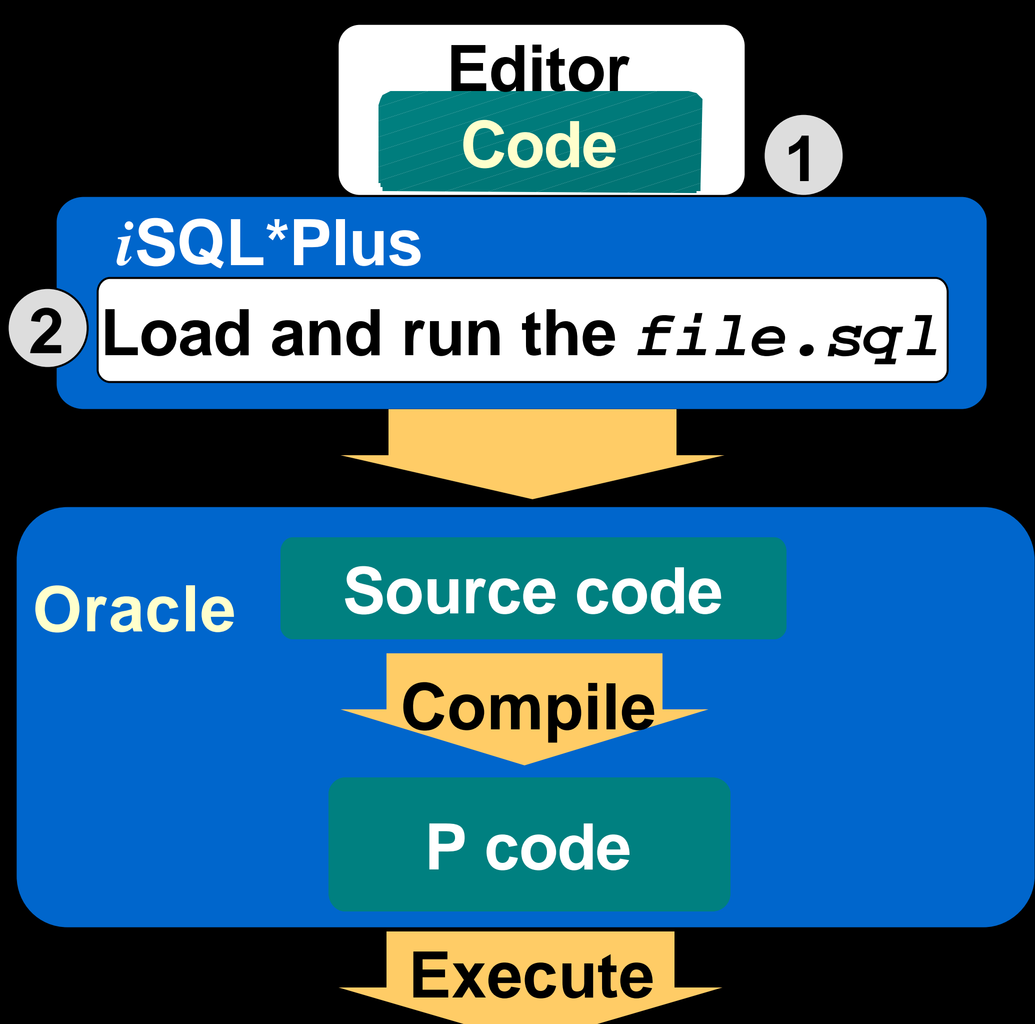
Referencing Package Objects



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Developing a Package



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Developing a Package

- Saving the text of the `CREATE PACKAGE` statement in two different SQL files facilitates later modifications to the package.
- A package specification can exist without a package body, but a package body cannot exist without a package specification.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating the Package Specification

Syntax:

```
CREATE [OR REPLACE] PACKAGE package_name
IS|AS
    public type and item declarations
    subprogram specifications
END package_name;
```

- The `REPLACE` option drops and recreates the package specification.
- Variables declared in the package specification are initialized to `NULL` by default.
- All the constructs declared in a package specification are visible to users who are granted privileges on the package.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring Public Constructs

COMM_PACKAGE package

Package specification

G_COMM

1

RESET_COMM procedure declaration

2

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Package Specification: Example

```
CREATE OR REPLACE PACKAGE comm_package IS
  g_comm NUMBER := 0.10;  --initialized to 0.10
  PROCEDURE reset_comm
  (p_comm   IN  NUMBER);
END comm_package;
/
```

```
Package created.
```

- `G_COMM` is a global variable and is initialized to 0.10.
- `RESET_COMM` is a public procedure that is implemented in the package body.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating the Package Body

Syntax:

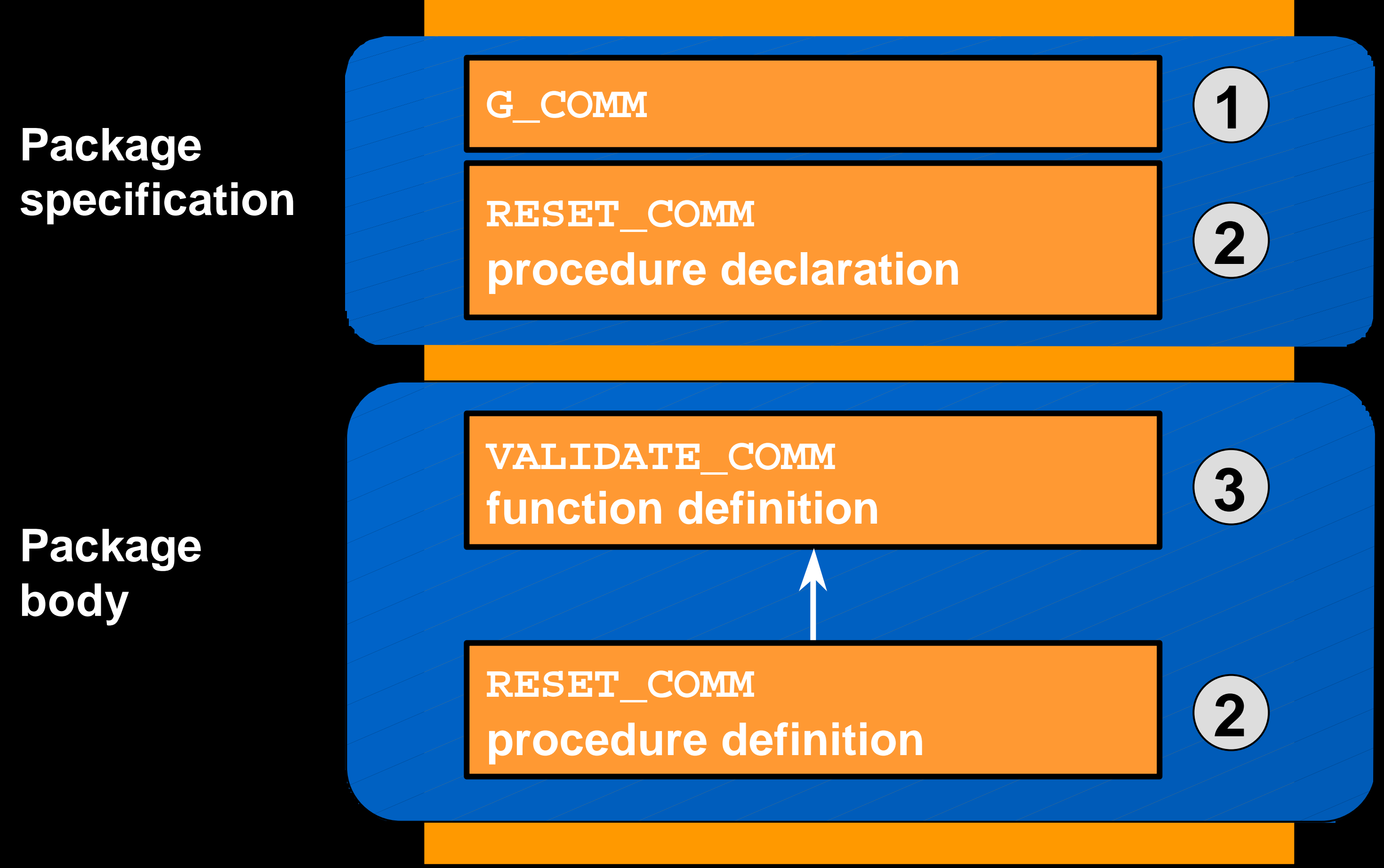
```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS|AS
    private type and item declarations
    subprogram bodies
END package_name;
```

- **The `REPLACE` option drops and recreates the package body.**
- **Identifiers defined only in the package body are private constructs. These are not visible outside the package body.**
- **All private constructs must be declared before they are used in the public constructs.**

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Public and Private Constructs



Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Package Body: Example

comm_pack.sql

```
CREATE OR REPLACE PACKAGE BODY comm_package
IS
   FUNCTION  validate_comm (p_comm IN NUMBER)
    RETURN BOOLEAN
   IS
     v_max_comm    NUMBER;
   BEGIN
     SELECT    MAX(commission_pct)
      INTO     v_max_comm
      FROM     employees;
     IF   p_comm > v_max_comm THEN RETURN(FALSE);
     ELSE   RETURN(TRUE);
     END IF;
   END validate_comm;
...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Package Body: Example

comm_pack.sql

```
  PROCEDURE  reset_comm (p_comm   IN  NUMBER)
  IS
  BEGIN
   IF  validate_comm(p_comm)
    THEN   g_comm:=p_comm;  --reset global variable
   ELSE
    RAISE_APPLICATION_ERROR(-20210,'Invalid commission');
   END IF;
  END reset_comm;
END comm_package;
/
```

```
Package body created.
```


Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Package Constructs

Example 1: Invoke a function from a procedure within the same package.

```
CREATE OR REPLACE PACKAGE BODY comm_package IS
   . . .
 PROCEDURE reset_comm
  (p_comm   IN  NUMBER)
 IS
 BEGIN
  IF validate_comm(p_comm)
  THEN g_comm := p_comm;
  ELSE
    RAISE_APPLICATION_ERROR
         (-20210, 'Invalid commission');
  END IF;
 END reset_comm;
END comm_package;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Package Constructs

Example 2: Invoke a package procedure from *i*SQL*Plus.

```
EXECUTE comm_package.reset_comm(0.15)
```

Example 3: Invoke a package procedure in a different schema.

```
EXECUTE scott.comm_package.reset_comm(0.15)
```

Example 4: Invoke a package procedure in a remote database.

```
EXECUTE comm_package.reset_comm@ny(0.15)
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring a Bodiless Package

```
CREATE OR REPLACE PACKAGE global_consts IS
  mile_2_kilo    CONSTANT  NUMBER  :=  1.6093;
  kilo_2_mile    CONSTANT  NUMBER  :=  0.6214;
  yard_2_meter   CONSTANT  NUMBER  :=  0.9144;
  meter_2_yard   CONSTANT  NUMBER  :=  1.0936;
END global_consts;
/

EXECUTE DBMS_OUTPUT.PUT_LINE('20 miles = '||20*
    global_consts.mile_2_kilo||' km')
```

```
Package created.
20 miles = 32.186 km
PL/SQL procedure successfully completed.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Referencing a Public Variable from a Stand-Alone Procedure

Example:

```
CREATE OR REPLACE PROCEDURE meter_to_yard
            (p_meter IN NUMBER, p_yard OUT NUMBER)
IS
BEGIN
  p_yard := p_meter * global_consts.meter_2_yard;
END meter_to_yard;
/
VARIABLE yard NUMBER
EXECUTE  meter_to_yard (1, :yard)
PRINT yard
```

```
Procedure created.
PL/SQL procedure successfully completed.
```

YARD
1.0936

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Packages

To remove the package specification and the body, use the following syntax:

```
DROP PACKAGE package_name;
```

To remove the package body, use the following syntax:

```
DROP PACKAGE BODY package_name;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Guidelines for Developing Packages

- Construct packages for general use.
- Define the package specification before the body.
- The package specification should contain only those constructs that you want to be public.
- Place items in the declaration part of the package body when you must maintain them throughout a session or across transactions.
- Changes to the package specification require recompilation of each referencing subprogram.
- The package specification should contain as few constructs as possible.

Copyright © Oracle Corporation, 2001. All rights reserved.

Advantages of Packages

- Modularity: Encapsulate related constructs.
- Easier application design: Code and compile specification and body separately.
- Hiding information:
  - Only the declarations in the package specification are visible and accessible to applications.
  - Private constructs in the package body are hidden and inaccessible.
  - All coding is hidden in the package body.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Advantages of Packages

- Added functionality: Persistency of variables and cursors
- Better performance:
  - The entire package is loaded into memory when the package is first referenced.
  - There is only one copy in memory for all users.
  - The dependency hierarchy is simplified.
- Overloading: Multiple subprograms of the same name

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Improve organization, management, security, and performance by using packages
- Group related procedures and functions together in a package
- Change a package body without affecting a package specification
- Grant security access to the entire package

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Hide the source code from users
- Load the entire package into memory on the first call
- Reduce disk access for subsequent calls
- Provide identifiers for the user session

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

Command	Task
`CREATE [OR REPLACE] PACKAGE`	Create (or modify) an existing package specification
`CREATE [OR REPLACE] PACKAGE BODY`	Create (or modify) an existing package body
`DROP PACKAGE`	Remove both the package specification and the package body
`DROP PACKAGE BODY`	Remove the package body only

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 12 Overview

This practice covers the following topics:

- Creating packages
- Invoking package program units

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

More Package Concepts

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Write packages that use the overloading feature
- Describe errors with mutually referential subprograms
- Initialize variables with a one-time-only procedure
- Identify persistent states

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Overloading

- Enables you to use the same name for different subprograms inside a PL/SQL block, a subprogram, or a package
- Requires the formal parameters of the subprograms to differ in number, order, or data type family
- Enables you to build more flexibility because a user or application is not restricted by the specific data type or number of formal parameters

Note: Only local or packaged subprograms can be overloaded. You cannot overload stand-alone subprograms.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Overloading: Example

over_pack.sql

```
CREATE OR REPLACE PACKAGE over_pack
IS
  PROCEDURE add_dept
   (p_deptno IN departments.department_id%TYPE,
    p_name IN departments.department_name%TYPE
                                 DEFAULT 'unknown',
    p_loc IN departments.location_id%TYPE DEFAULT 0);
  PROCEDURE add_dept
   (p_name IN departments.department_name%TYPE
                                 DEFAULT 'unknown',
    p_loc  IN departments.location_id%TYPE DEFAULT 0);
END over_pack;
/
```

```
Package created.
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Overloading: Example

over_pack_body.sql

```
CREATE OR REPLACE PACKAGE BODY over_pack  IS
 PROCEDURE add_dept
 (p_deptno IN departments.department_id%TYPE,
  p_name IN departments.department_name%TYPE DEFAULT 'unknown',
  p_loc  IN departments.location_id%TYPE DEFAULT 0)
 IS
 BEGIN
  INSERT INTO departments (department_id,
                           department_name, location_id)
  VALUES  (p_deptno, p_name, p_loc);
 END add_dept;
 PROCEDURE add_dept
 (p_name IN departments.department_name%TYPE DEFAULT 'unknown',
  p_loc  IN departments.location_id%TYPE DEFAULT 0)
 IS
 BEGIN
  INSERT INTO departments (department_id,
                           department_name, location_id)
  VALUES  (departments_seq.NEXTVAL, p_name, p_loc);
 END add_dept;
END over_pack;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Overloading: Example

- Most built-in functions are overloaded.
- For example, see the `TO_CHAR` function of the `STANDARD` package.

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, P2 VARCHAR2) RETURN VARCHAR2;
```

- If you redeclare a built-in subprogram in a PL/SQL program, your local declaration overrides the global declaration.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Forward Declarations

You must declare identifiers before referencing them.

```
CREATE OR REPLACE PACKAGE BODY forward_pack
IS
  PROCEDURE award_bonus(. . .)
  IS
  BEGIN
   calc_rating(. . .);        --illegal reference

  END;

  PROCEDURE calc_rating(. . .)
  IS
  BEGIN
    ...
  END;

END forward_pack;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Forward Declarations

```
CREATE OR REPLACE PACKAGE BODY forward_pack
IS

 PROCEDURE calc_rating(. . .);     -- forward declaration

 PROCEDURE award_bonus(. . .)
 IS                                -- subprograms defined
 BEGIN                             -- in alphabetical order
  calc_rating(. . .);
 . . .
 END;

 PROCEDURE calc_rating(. . .)
 IS
 BEGIN
  . . .
 END;

END forward_pack;
/
```


Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a One-Time-Only Procedure

```
CREATE OR REPLACE PACKAGE taxes
IS
    tax   NUMBER;
  ...  -- declare all public procedures/functions
END  taxes;
/
```

```
CREATE OR REPLACE PACKAGE BODY taxes
IS
  ... -- declare all private variables
  ... -- define public/private procedures/functions
BEGIN
   SELECT   rate_value
   INTO     tax
   FROM     tax_rates
   WHERE    rate_name = 'TAX';
END taxes;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Package Functions Used in SQL

A function called from:

- A query or DML statement can not end the current transaction, create or roll back to a savepoint, or `ALTER` the system or session.
- A query statement or a parallelized DML statement can not execute a DML statement or modify the database.
- A DML statement can not read or modify the particular table being modified by that DML statement.

Note: Calls to subprograms that break the above restrictions are not allowed.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

User Defined Package: `taxes_pack`

```
CREATE OR REPLACE PACKAGE taxes_pack
IS
   FUNCTION tax (p_value IN NUMBER) RETURN NUMBER;
END taxes_pack;
/
```

```
Package created.
```

```
CREATE OR REPLACE PACKAGE BODY taxes_pack
IS
   FUNCTION tax (p_value IN NUMBER) RETURN NUMBER
   IS
    v_rate NUMBER := 0.08;
   BEGIN
     RETURN (p_value * v_rate);
   END tax;
END taxes_pack;
/
```

```
Package body created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking a User-Defined Package Function from a SQL Statement

```
SELECT taxes_pack.tax(salary), salary, last_name
FROM   employees;
```

TAXES_PACK.TAX(SALARY)	SALARY	LAST_NAME
1920	24000	King
1360	17000	Kochhar
1360	17000	De Haan
720	9000	Hunold
480	6000	Ernst
422.4	5280	Austin
422.4	5280	Pataballa
369.6	4620	Lorentz
960	12000	Greenberg

...

109 rows selected.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Persistent State of Package Variables: Example

```
CREATE OR REPLACE PACKAGE comm_package IS
  g_comm  NUMBER := 10;          --initialized to 10
  PROCEDURE reset_comm (p_comm   IN   NUMBER);
END comm_package;
/
```

```
CREATE OR REPLACE PACKAGE BODY comm_package IS
   FUNCTION  validate_comm (p_comm  IN  NUMBER)
             RETURN BOOLEAN
   IS v_max_comm    NUMBER;
   BEGIN
    ...      --  validates commission to be less than maximum
             --   commission in the table
   END validate_comm;
   PROCEDURE  reset_comm (p_comm   IN  NUMBER)
   IS BEGIN
     ...     -- calls validate_comm with specified value
   END reset_comm;
END comm_package;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Persistent State of Package Variables

Time	Scott	Jones
9:00	`EXECUTE comm_package.reset_comm (0.25)` max_comm=0.4 > 0.25 g_comm = 0.25	
9:30		`INSERT INTO employees (last_name, commission_pct) VALUES ('Madonna', 0.8);` max_comm=0.8
9:35		`EXECUTE comm_package.reset_comm(0.5)` max_comm=0.8 > 0.5 g_comm = 0.5

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Persistent State of Package Variables

Time	Scott	Jones
9:00	EXECUTE comm_package.reset_comm (0.25) max_comm=0.4 > 0.25 g_comm = 0.25	
9:30		INSERT INTO employees (last_name, commission_pct) VALUES ('Madonna', 0.8); max_comm=0.8
9:35		EXECUTE comm_package.reset_comm(0.5) max_comm=0.8 > 0.5 g_comm = 0.5
10:00	EXECUTE comm_package.reset_comm (0.6) max_comm=0.4 < 0.6 INVALID	
11:00		ROLLBACK;
11:01		EXIT

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Persistent State of Package Variables

Time	Scott	Jones
9:00	EXECUTE comm_package.reset_comm (0.25) max_comm=0.4 > 0.25 g_comm = 0.25	
9:30		INSERT INTO employees (last_name, commission_pct) VALUES ('Madonna', 0.8); max_comm=0.8
9:35		EXECUTE comm_package.reset_comm(0.5) max_comm=0.8 > 0.5 g_comm = 0.5
10:00	EXECUTE comm_package.reset_comm (0.6) max_comm=0.4 < 0.6 INVALID	
11:00		ROLLBACK;
11:01		EXIT
11:45		Logged In again. g_comm = 10, max_comm=0.4
12:00	VALID →	EXECUTE comm_package.reset_comm(0.25)

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling the Persistent State of a Package Cursor

Example:

```
CREATE OR REPLACE PACKAGE pack_cur
IS
 CURSOR c1 IS  SELECT employee_id
               FROM   employees
               ORDER BY employee_id DESC;
 PROCEDURE proc1_3rows;
 PROCEDURE proc4_6rows;
END pack_cur;
/
```

```
Package created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling the Persistent State of a Package Cursor

```
CREATE OR REPLACE PACKAGE BODY pack_cur  IS
  v_empno NUMBER;
  PROCEDURE proc1_3rows IS
  BEGIN
    OPEN c1;
    LOOP
     FETCH c1 INTO v_empno;
     DBMS_OUTPUT.PUT_LINE('Id :' ||(v_empno));
     EXIT WHEN c1%ROWCOUNT >= 3;
    END LOOP;
  END proc1_3rows;
  PROCEDURE proc4_6rows IS
  BEGIN
    LOOP
     FETCH c1 INTO v_empno;
     DBMS_OUTPUT.PUT_LINE('Id :' ||(v_empno));
     EXIT WHEN c1%ROWCOUNT >= 6;
    END LOOP;
    CLOSE c1;
  END proc4_6rows;
END pack_cur;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Executing PACK_CUR

```
SET SERVEROUTPUT ON
EXECUTE pack_cur.proc1_3rows
EXECUTE pack_cur.proc4_6rows
```

```
Id :208
Id :207
Id :206
PL/SQL procedure successfully completed.
Id :205
Id :204
Id :203
PL/SQL procedure successfully completed.
```


Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Tables and Records in Packages

```
CREATE OR REPLACE PACKAGE emp_package IS
  TYPE emp_table_type IS TABLE OF employees%ROWTYPE
     INDEX BY BINARY_INTEGER;
  PROCEDURE read_emp_table
              (p_emp_table OUT emp_table_type);
END emp_package;
/
```

```
CREATE OR REPLACE PACKAGE BODY emp_package IS
  PROCEDURE read_emp_table
              (p_emp_table OUT emp_table_type) IS
  i BINARY_INTEGER := 0;
  BEGIN
    FOR emp_record IN (SELECT * FROM employees)
    LOOP
      p_emp_table(i) := emp_record;
      i:= i+1;
    END LOOP;
  END read_emp_table;
END emp_package;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Overload subprograms
- Use forward referencing
- Use one-time-only procedures
- Describe the purity level of package functions
- Identify the persistent state of packaged objects

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 13 Overview

This practice covers the following topics:

- Using overloaded subprograms
- Creating a one-time-only procedure

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle Supplied Packages

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Write dynamic SQL statements using `DBMS_SQL` and `EXECUTE IMMEDIATE`
- Describe the use and application of some Oracle server-supplied packages:
  - `DBMS_DDL`
  - `DBMS_JOB`
  - `DBMS_OUTPUT`
  - `UTL_FILE`
  - `UTL_HTTP` and `UTL_TCP`

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Supplied Packages

Oracle-supplied packages:

- Are provided with the Oracle server
- Extend the functionality of the database
- Enable access to certain SQL features normally restricted for PL/SQL

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Native Dynamic SQL

Dynamic SQL:

- Is a SQL statement that contains variables that can change during runtime
- Is a SQL statement with placeholders and is stored as a character string
- Enables general-purpose code to be written
- Enables data-definition, data-control, or session-control statements to be written and executed from PL/SQL
- Is written using either `DBMS_SQL` or native dynamic SQL

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Execution Flow

SQL statements go through various stages:

- Parse
- Bind
- Execute
- Fetch

Note: Some stages may be skipped.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the DBMS_SQL Package

The DBMS_SQL package is used to write dynamic SQL in stored procedures and to parse DDL statements. Some of the procedures and functions of the package include:

- OPEN_CURSOR
- PARSE
- BIND_VARIABLE
- EXECUTE
- FETCH_ROWS
- CLOSE_CURSOR

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using DBMS_SQL

```
CREATE OR REPLACE PROCEDURE delete_all_rows
   (p_tab_name IN VARCHAR2, p_rows_del OUT NUMBER)
 IS
   cursor_name    INTEGER;
 BEGIN
   cursor_name := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(cursor_name, 'DELETE FROM '||p_tab_name,
                  DBMS_SQL.NATIVE );
   p_rows_del := DBMS_SQL.EXECUTE (cursor_name);
   DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
/
```

Use dynamic SQL to delete rows

```
VARIABLE deleted NUMBER
EXECUTE  delete_all_rows('employees', :deleted)
PRINT deleted
```

PL/SQL procedure successfully completed.

DELETED
109

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the EXECUTE IMMEDIATE Statement

Use the EXECUTE IMMEDIATE statement for native dynamic SQL with better performance.

```
EXECUTE IMMEDIATE dynamic_string
 [INTO {define_variable
     [, define_variable] ... | record}]
 [USING [IN|OUT|IN OUT] bind_argument
     [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- **INTO is used for single-row queries and specifies the variables or records into which column values are retrieved.**
- **USING is used to hold all bind arguments. The default parameter mode is IN.**

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Dynamic SQL Using `EXECUTE IMMEDIATE`

```
CREATE PROCEDURE del_rows
  (p_table_name  IN  VARCHAR2,
   p_rows_deld   OUT NUMBER)
IS
BEGIN
  EXECUTE IMMEDIATE 'delete from '||p_table_name;
  p_rows_deld := SQL%ROWCOUNT;
END;
/
```

Procedure created.

```
VARIABLE deleted NUMBER
EXECUTE del_rows('test_employees',:deleted)
PRINT deleted
```

PL/SQL procedure successfully completed.

DELETED
109

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the DBMS_DDL Package

The DBMS_DDL Package:

- Provides access to some SQL DDL statements from stored procedures
- Includes some procedures:
  - ALTER_COMPILE (object_type, owner, object_name)

```
DBMS_DDL.ALTER_COMPILE('PROCEDURE','A_USER','QUERY_EMP')
```

  - ANALYZE_OBJECT (object_type, owner, name, method)

```
DBMS_DDL.ANALYZE_OBJECT('TABLE','A_USER','JOBS','COMPUTE')
```

Note: This package runs with the privileges of calling user, rather than the package owner SYS.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using `DBMS_JOB` for Scheduling

`DBMS_JOB` Enables the scheduling and execution of PL/SQL programs:

- Submitting jobs
- Executing jobs
- Changing execution parameters of jobs
- Removing jobs
- Suspending Jobs

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DBMS_JOB Subprograms

Available subprograms include:

- SUBMIT
- REMOVE
- CHANGE
- WHAT
- NEXT_DATE
- INTERVAL
- BROKEN
- RUN

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Submitting Jobs

You can submit jobs by using `DBMS_JOB.SUBMIT`.

Available parameters include:

- `JOB OUT BINARY_INTEGER`
- `WHAT IN VARCHAR2`
- `NEXT_DATE IN DATE DEFAULT SYSDATE`
- `INTERVAL IN VARCHAR2 DEFAULT 'NULL'`
- `NO_PARSE IN BOOLEAN DEFAULT FALSE`

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Submitting Jobs

Use `DBMS_JOB.SUBMIT` to place a job to be executed in the job queue.

```
VARIABLE jobno NUMBER
BEGIN
  DBMS_JOB.SUBMIT (
   job   => :jobno,
   what => 'OVER_PACK.ADD_DEPT(''EDUCATION'',2710);',
   next_date =>  TRUNC(SYSDATE + 1),
   interval  =>  'TRUNC(SYSDATE + 1)'
  );
  COMMIT;
END;
/
PRINT jobno
```

PL/SQL procedure successfully completed.

JOBNO
1

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Changing Job Characteristics

- `DBMS_JOB.CHANGE`: Changes the `WHAT`, `NEXT_DATE`, and `INTERVAL` parameters
- `DBMS_JOB.INTERVAL`: Changes the `INTERVAL` parameter
- `DBMS_JOB.NEXT_DATE`: Changes the next execution date
- `DBMS_JOB.WHAT`: Changes the `WHAT` parameter

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Running, Removing, and Breaking Jobs

- `DBMS_JOB.RUN`: Runs a submitted job immediately
- `DBMS_JOB.REMOVE`: Removes a submitted job from the job queue
- `DBMS_JOB.BROKEN`: Marks a submitted job as broken, and a broken job will not run

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Viewing Information on Submitted Jobs

- Use the DBA_JOBS dictionary view to see the status of submitted jobs.

```
SELECT job, log_user, next_date, next_sec,
       broken, what
FROM DBA_JOBS;
```

JOB	LOG_USER	NEXT_DATE	NEXT_SEC	B	WHAT
1	PLSQL	28-SEP-01	06:00:00	N	OVER_PACK.ADD_DEPT('EDUCATION',2710);

- Use the DBA_JOBS_RUNNING dictionary view to display jobs that are currently running.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the DBMS_OUTPUT Package

The DBMS_OUTPUT package enables you to output messages from PL/SQL blocks. Available procedures include:

- PUT
- NEW_LINE
- PUT_LINE
- GET_LINE
- GET_LINES
- ENABLE/DISABLE

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Interacting with Operating System Files

- `UTL_FILE` Oracle-supplied package:
  - Provides text file I/O capabilities
  - Is available with version 7.3 and later
- The `DBMS_LOB` Oracle-supplied package:
  - Provides read-only operations on external `BFILES`
  - Is available with version 8 and later
  - Enables read and write operations on internal `LOBS`

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

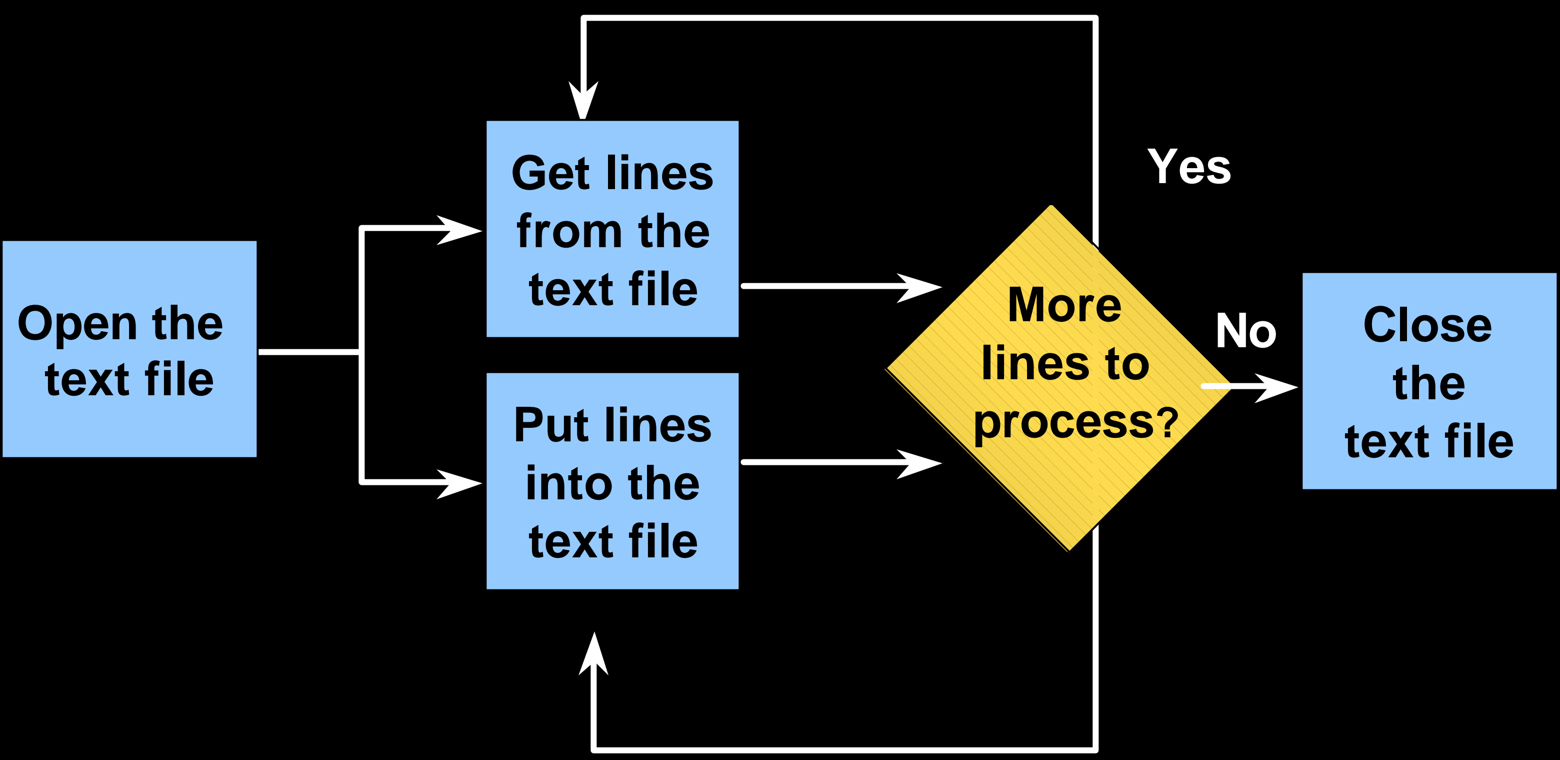
What Is the UTL_FILE Package?

- Extends I/O to text files within PL/SQL
- Provides security for directories on the server through the init.ora file
- Is similar to standard operating system I/O
  - Open files
  - Get text
  - Put text
  - Close files
  - Use the exceptions specific to the UTL_FILE package

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

File Processing Using the UTL_FILE Package



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

UTL_FILE Procedures and Functions

- Function FOPEN
- Function IS_OPEN
- Procedure GET_LINE
- Procedure PUT, PUT_LINE, PUTF
- Procedure NEW_LINE
- Procedure FFLUSH
- Procedure FCLOSE, FCLOSE_ALL

Copyright © Oracle Corporation, 2001. All rights reserved.

Exceptions Specific to the `UTL_FILE` Package

- `INVALID_PATH`
- `INVALID_MODE`
- `INVALID_FILEHANDLE`
- `INVALID_OPERATION`
- `READ_ERROR`
- `WRITE_ERROR`
- `INTERNAL_ERROR`

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The FOPEN and IS_OPEN Functions

```
FUNCTION FOPEN
(location IN VARCHAR2,
 filename IN VARCHAR2,
 open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

```
FUNCTION IS_OPEN
(file_handle IN FILE_TYPE)
RETURN BOOLEAN;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using UTL_FILE

sal_status.sql

```
CREATE OR REPLACE PROCEDURE sal_status
(p_filedir IN VARCHAR2,  p_filename IN VARCHAR2)
IS
 v_filehandle UTL_FILE.FILE_TYPE;
 CURSOR emp_info IS
   SELECT last_name, salary, department_id
    FROM employees
    ORDER BY department_id;
 v_newdeptno employees.department_id%TYPE;
 v_olddeptno employees.department_id%TYPE := 0;
BEGIN
 v_filehandle := UTL_FILE.FOPEN (p_filedir, p_filename,'w');
 UTL_FILE.PUTF (v_filehandle,'SALARY REPORT: GENERATED ON
                              %s\n', SYSDATE);
 UTL_FILE.NEW_LINE (v_filehandle);
 FOR v_emp_rec IN emp_info LOOP
  v_newdeptno := v_emp_rec.department_id;
...
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using UTL_FILE

sal_status.sql

```
    ...
    IF v_newdeptno <> v_olddeptno THEN
      UTL_FILE.PUTF (v_filehandle, 'DEPARTMENT: %s\n',
                      v_emp_rec.department_id);
    END IF;
    UTL_FILE.PUTF (v_filehandle,'  EMPLOYEE: %s earns: %s\n',
                     v_emp_rec.last_name, v_emp_rec.salary);
    v_olddeptno := v_newdeptno;
  END LOOP;
  UTL_FILE.PUT_LINE (v_filehandle, '*** END OF REPORT ***');
  UTL_FILE.FCLOSE (v_filehandle);
EXCEPTION
  WHEN UTL_FILE.INVALID_FILEHANDLE THEN
    RAISE_APPLICATION_ERROR (-20001, 'Invalid File.');
  WHEN UTL_FILE.WRITE_ERROR THEN
   RAISE_APPLICATION_ERROR (-20002, 'Unable to write to
                                     file');
END sal_status;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The UTL_HTTP Package

The UTL_HTTP package:

- Enables HTTP callouts from PL/SQL and SQL to access data on the Internet
- Contains the functions REQUEST and REQUEST_PIECES which take the URL of a site as a parameter, contact that site, and return the data obtained from that site
- Requires a proxy parameter to be specified in the above functions, if the client is behind a firewall
- Raises INIT_FAILED or REQUEST_FAILED exceptions if HTTP call fails
- Reports an HTML error message if specified URL is not accessible

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the UTL_HTTP Package

```sql
SELECT UTL_HTTP.REQUEST('http://www.oracle.com',
                        'edu-proxy.us.oracle.com')
FROM DUAL;
```

UTL_HTTP.REQUEST('HTTP://WWW.ORACLE.COM','EDU-PROXY.US.ORACLE.COM')

```
<html> <head> <title>Oracle Corporation</title> <meta name="description" content="Oracle Corporation provides the software that powers the
Internet. F or more information about Oracle, please call 650/506-7000."> <meta name="keywords" content="Oracle, Oracle Corporation, Oracle
Corp,Oracle8i, Oracle 9i, 8i, 9i"> <script language="JavaScript" src="http://www.oracle.com/admin/jscripts/lib.js"> </script> </head> <body
bgcolor="#FFFFFF" text="#000000 " link="#000000" vlink="#FF0000"> <!--Start Header--> <center> <table border=0 cellspacing=0 cellpadding=3
width=850 align="center"> <tr> <td align= center valign="middle"> <div align="right"><a
href="http://www.oracle.com/elog/trackurl?d=http://my.oracle.com&di=872609" target="_top"><img src="per sonalize.gif" width="50" height="50"
border="0" alt="My Oracle"></a> <a href="/products/index.html?content.html" target="_top"><img src="products .gif" width="50"
height="50" border="0" alt="Products"></a>  <a href="http://oraclestore.oracle.com/" target="_top"><img src="store.gif" width=" 50"
height="50" border="0" alt="Store"></a></div></td> <td align=center valign="middle" width="34%"> <div align="center"><a href="/"
target="_top"><i mg src="logo.gif" width="175" height="28" border=0 alt="Oracle.com" align="middle"></a></div></td> <td align=center
valign="middle"> <div align="lef t"><a href="http://otn.oracle.com/software/"><img src="download.gif" width="50" height="50" border="0"
alt="Download"></a> <a href="/corporate/co ntact/index.html?content.html" target="_top"><img src="contact.gif" width="50" height="50"
border="0" alt="Contact"></a> <a href="/pls/use/use_qu
ery_html.show_query_form?p_person_id=100&amp;p_location_array=&amp;p_doc_location_array=&amp;p_keyword_array=&amp;p_value_array="><img
src="search.gif " width="50" height="50" border="0" alt="Search"></a></div> </td></tr></table> <!--End Header--> <table border=0 cellspacing=0
cellpadding=0 width=" 850"> <tr><td align="center" width="100%"> <table
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the UTL_TCP Package

The UTL_TCP Package:

- Enables PL/SQL applications to communicate with external TCP/IP-based servers using TCP/IP
- Contains functions to open and close connections, to read or write binary or text data to or from a service on an open connection
- Requires remote host and port as well as local host and port as arguments to its functions
- Raises exceptions if the buffer size is too small, when no more data is available to read from a connection, when a generic network error occurs, or when bad arguments are passed to a function call

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle-Supplied Packages

Other Oracle-supplied packages include:

- DBMS_ALERT
- DBMS_APPLICATION_INFO
- DBMS_DESCRIBE
- DBMS_LOCK
- DBMS_SESSION
- DBMS_SHARED_POOL
- DBMS_TRANSACTION
- DBMS_UTILITY

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Take advantage of the preconfigured packages that are provided by Oracle
- Create packages by using the `catproc.sql` script
- Create packages individually.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 14 Overview

This practice covers using:

- `DBMS_SQL` for dynamic SQL
- `DBMS_DDL` to analyze a table
- `DBMS_JOB` to schedule a task
- `UTL_FILE` to generate text reports

Copyright © Oracle Corporation, 2001. All rights reserved.

Manipulating Large Objects

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

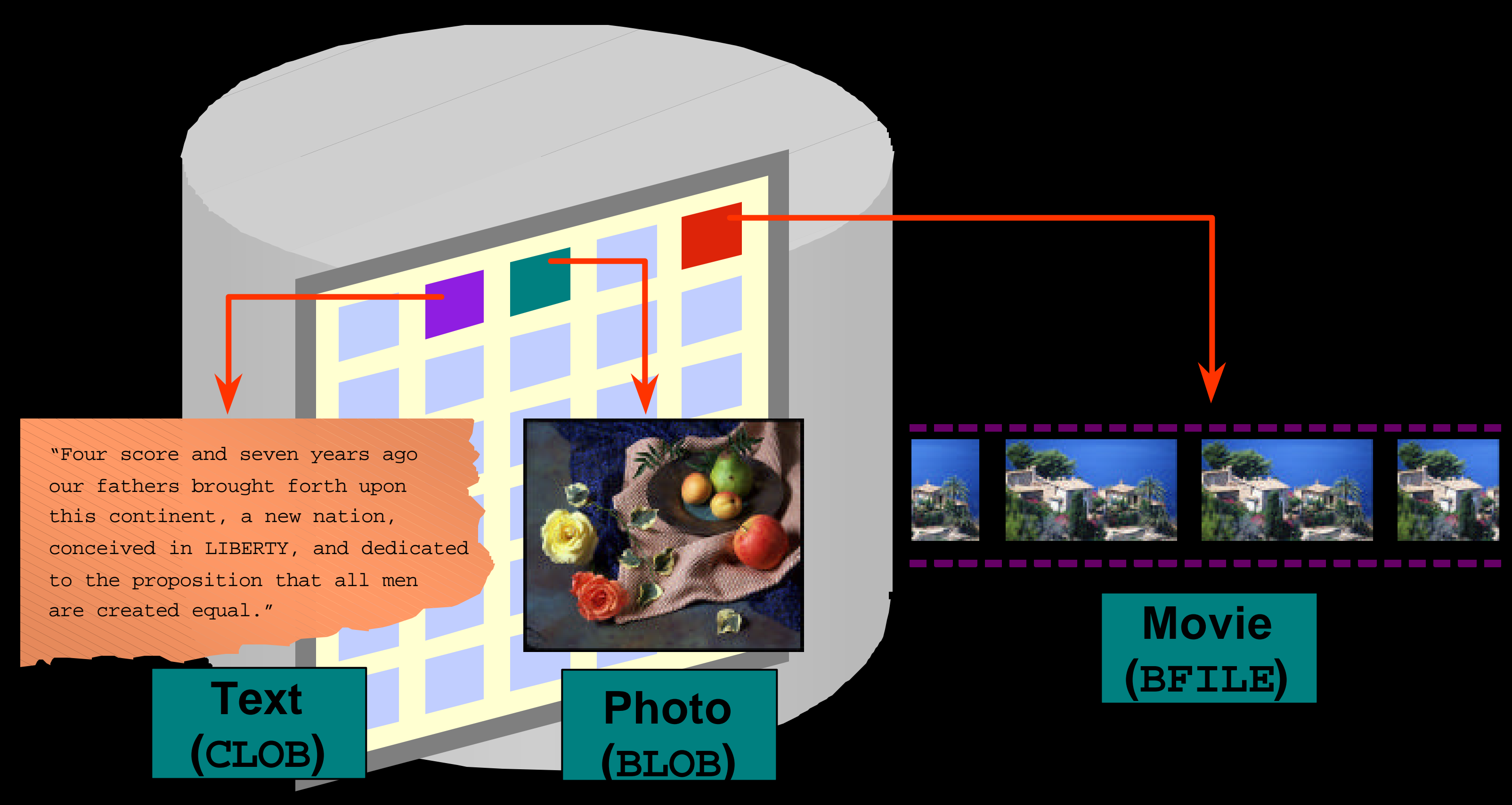
- Compare and contrast `LONG` and large object (`LOB`) data types
- Create and maintain `LOB` data types
- Differentiate between internal and external `LOB`s
- Use the `DBMS_LOB` PL/SQL package
- Describe the use of temporary `LOB`s

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is a LOB?

LOBs are used to store large unstructured data such as text, graphic images, films, and sound waveforms.



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Contrasting LONG and LOB Data Types

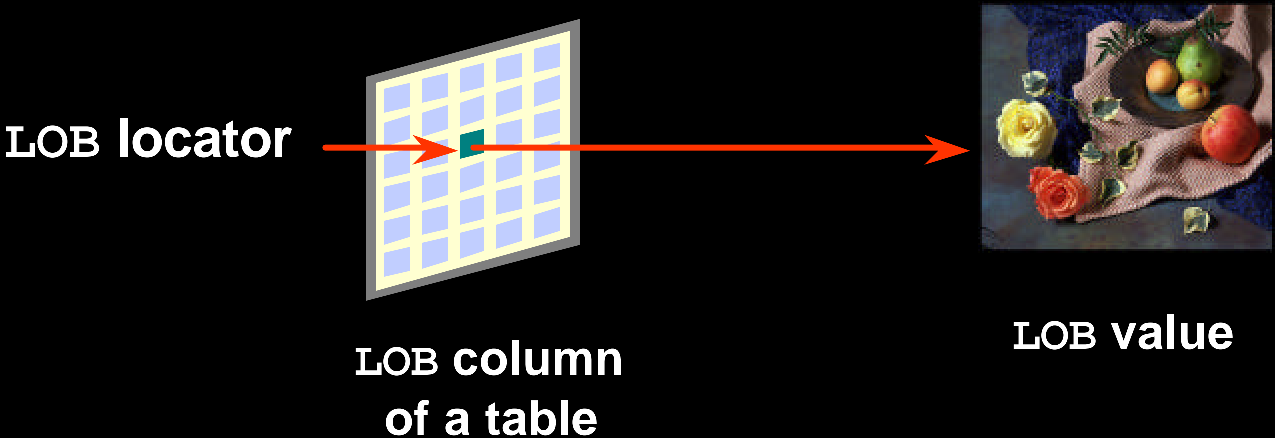
LONG and LONG RAW	LOB
Single LONG column per table	Multiple LOB columns per table
Up to 2 GB	Up to 4 GB
SELECT returns data	SELECT returns locator
Data stored in-line	Data stored in-line or out-of-line
Sequential access to data	Random access to data

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Anatomy of a LOB

The LOB column stores a locator to the LOB's value.

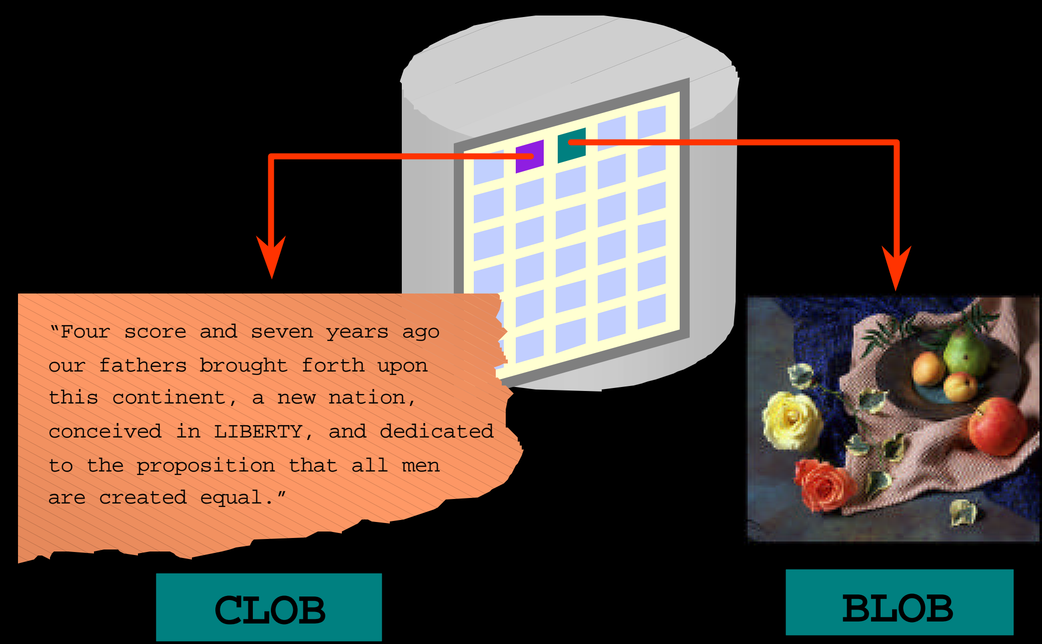


ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Internal LOBs

The LOB value is stored in the database.



Copyright © Oracle Corporation, 2001. All rights reserved.

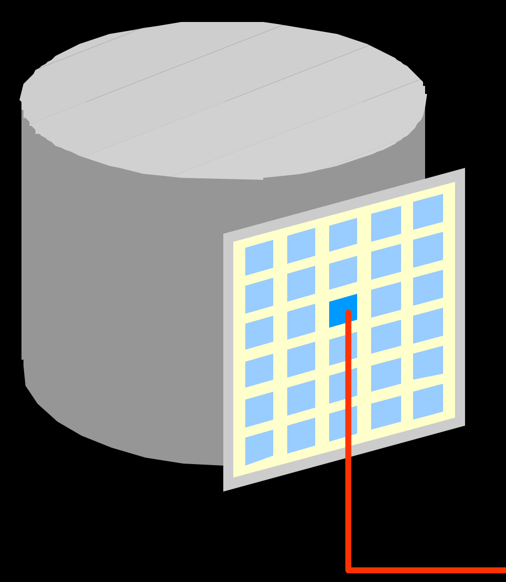
Managing Internal LOBs

- To interact fully with LOB, file-like interfaces are provided in:
  – PL/SQL package DBMS_LOB
  – Oracle Call Interface (OCI)
  – Oracle Objects for object linking and embedding (OLE)
  – Pro*C/C++ and Pro*COBOL precompilers
  – JDBC
- The Oracle server provides some support for LOB management through SQL.

ORACLE

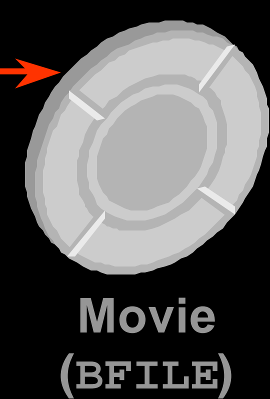
Copyright © Oracle Corporation, 2001. All rights reserved.

What Are BFILEs?



The BFILE data type supports an external or file-based large object as:

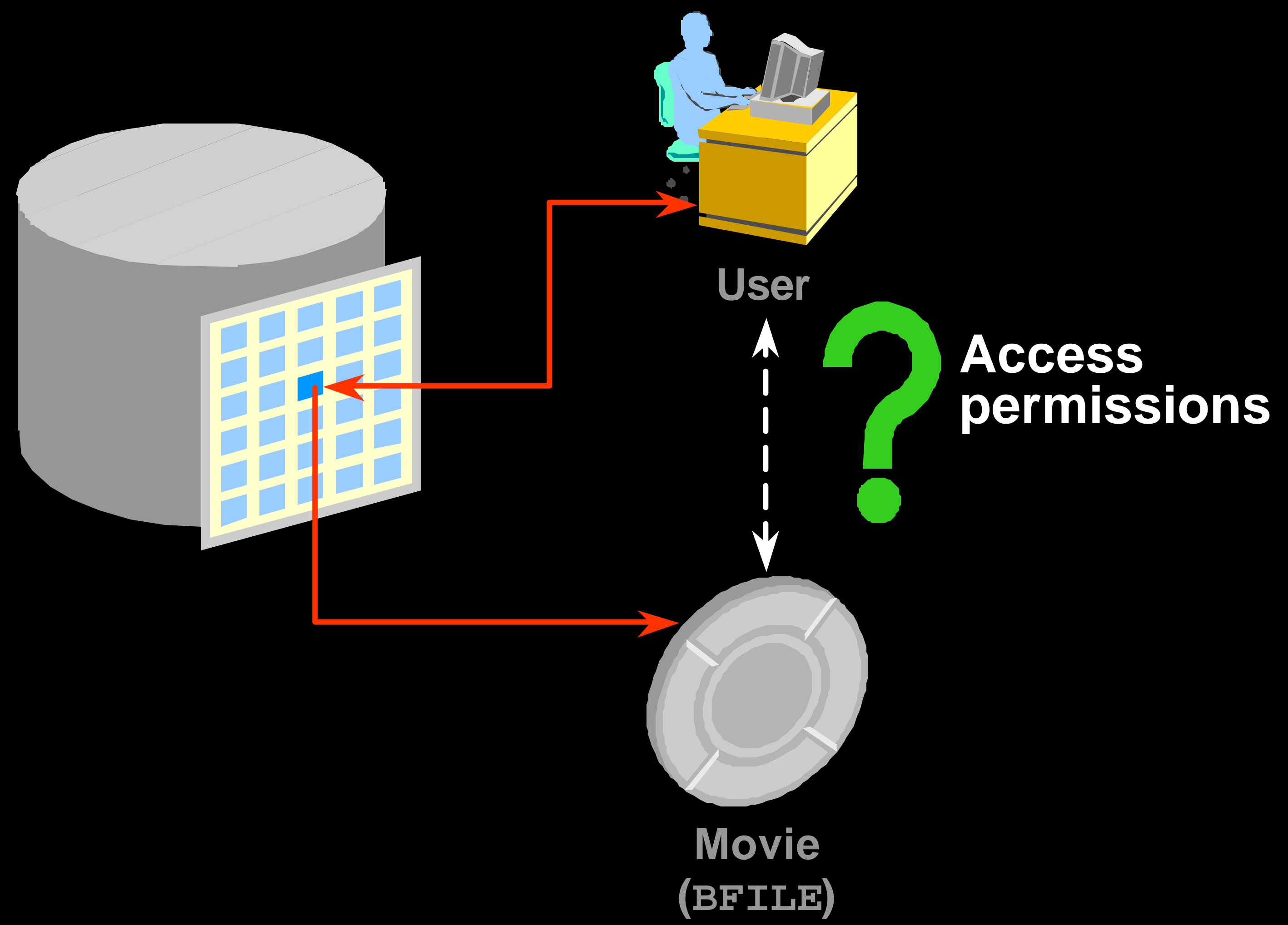
- Attributes in an object type
- Column values in a table



Movie
(BFILE)

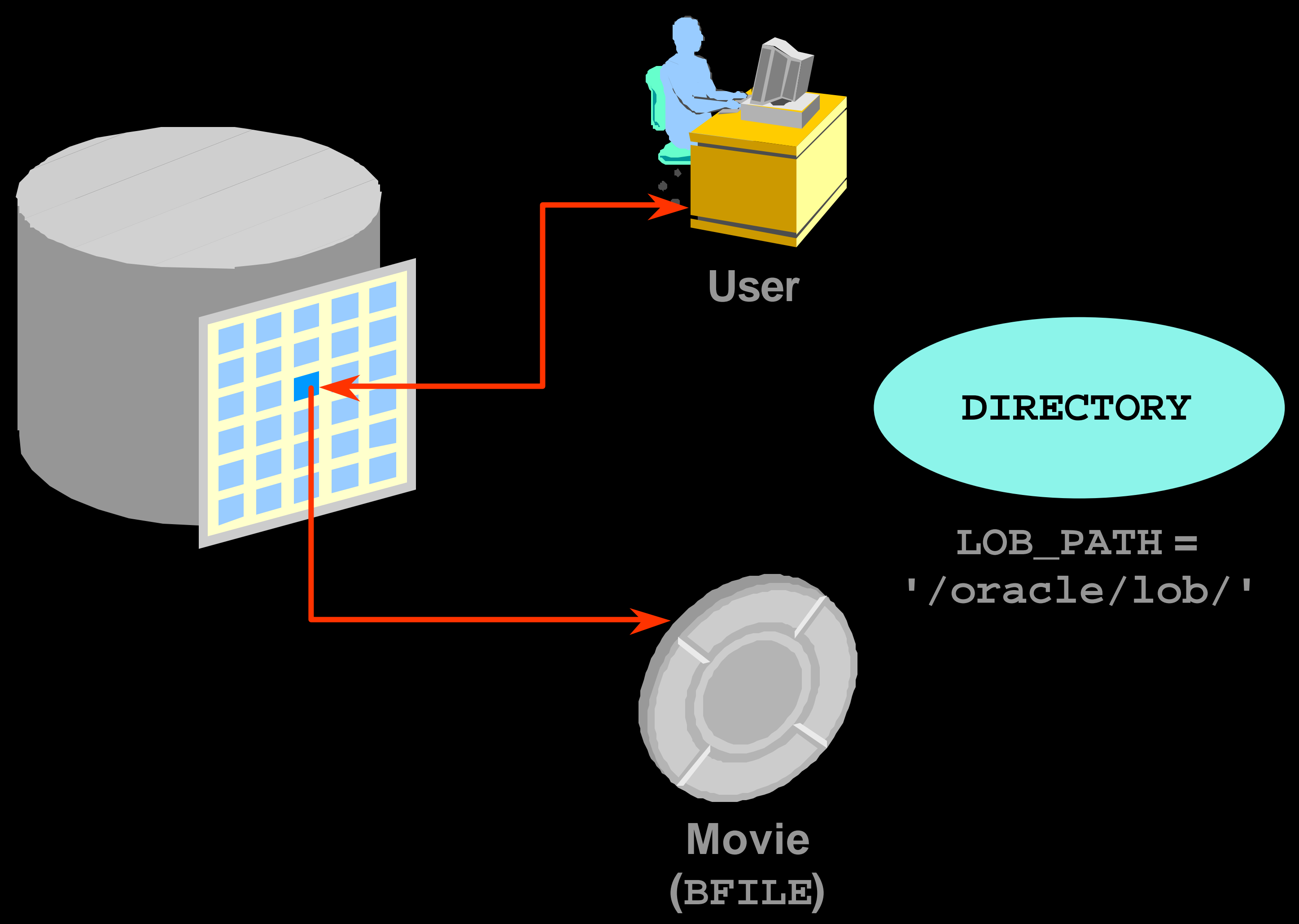
Copyright © Oracle Corporation, 2001. All rights reserved.

Securing BFILES



Copyright © Oracle Corporation, 2001. All rights reserved.

A New Database Object: DIRECTORY



Copyright © Oracle Corporation, 2001. All rights reserved.

Guidelines for Creating DIRECTORY Objects

- Do not create DIRECTORY objects on paths with database files.
- Limit the number of people who are given the following system privileges:
  - CREATE ANY DIRECTORY
  - DROP ANY DIRECTORY
- All DIRECTORY objects are owned by SYS.
- Create directory paths and properly set permissions before using the DIRECTORY object so that the Oracle server can read the file.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Managing BFILES

- Create an OS directory and supply files.
- Create an Oracle table with a column that holds the BFILE data type.
- Create a DIRECTORY object.
- Grant privileges to read the DIRECTORY object to users.
- Insert rows into the table by using the BFILENAME function.
- Declare and initialize a LOB locator in a program.
- Read the BFILE.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Preparing to Use BFILES

- Create or modify an Oracle table with a column that holds the BFILE data type.

```
ALTER TABLE employees
  ADD emp_video BFILE;
```

- Create a DIRECTORY object by using the CREATE DIRECTORY command.

```
CREATE DIRECTORY dir_name
  AS os_path;
```

- Grant privileges to read the DIRECTORY object to users.

```
GRANT READ ON DIRECTORY dir_name TO
user|role|PUBLIC;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

The `BFILENAME` Function

Use the `BFILENAME` function to initialize a `BFILE` column.

```
FUNCTION BFILENAME (directory_alias IN VARCHAR2,
                    filename IN VARCHAR2)
RETURN BFILE;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Loading BFILES

```
CREATE OR REPLACE PROCEDURE load_emp_bfile
     (p_file_loc IN VARCHAR2) IS
  v_file      BFILE;
  v_filename  VARCHAR2(16);
  CURSOR emp_cursor IS
    SELECT first_name FROM employees
    WHERE department_id = 60 FOR UPDATE;
BEGIN
  FOR emp_record IN emp_cursor LOOP
    v_filename := emp_record.first_name || '.bmp';
    v_file := BFILENAME(p_file_loc, v_filename);
    DBMS_LOB.FILEOPEN(v_file);
    UPDATE employees SET emp_video = v_file
      WHERE CURRENT OF emp_cursor;
    DBMS_OUTPUT.PUT_LINE('LOADED FILE: '||v_filename
      || ' SIZE: ' || DBMS_LOB.GETLENGTH(v_file));
    DBMS_LOB.FILECLOSE(v_file);
  END LOOP;
END load_emp_bfile;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Loading BFILES

Use the DBMS_LOB.FILEEXISTS function to vefiry that the file exists in the operating system. The function returns 0 if the file does not exist, and returns 1 if the file does exist.

```
CREATE OR REPLACE PROCEDURE load_emp_bfile
(p_file_loc IN VARCHAR2)
IS
  v_file        BFILE;   v_filename    VARCHAR2(16);
  v_file_exists BOOLEAN;
  CURSOR emp_cursor IS ...
BEGIN
  FOR emp_record IN emp_cursor LOOP
   v_filename := emp_record.first_name || '.bmp';
   v_file := BFILENAME  (p_file_loc, v_filename);
   v_file_exists := (DBMS_LOB.FILEEXISTS(v_file) = 1);
    IF v_file_exists THEN
      DBMS_LOB.FILEOPEN (v_file); ...
```


Copyright © Oracle Corporation, 2001. All rights reserved.

Migrating from LONG to LOB

The Oracle9*i* server allows migration of LONG columns to LOB columns.

- Data migration consists of the procedure to move existing tables containing LONG columns to use LOBs.

```
ALTER TABLE [<schema>.] <table_name>
    MODIFY (<long_col_name> {CLOB | BLOB | NCLOB}
```

- Application migration consists of changing existing LONG applications for using LOBs.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Migrating From LONG to LOB

- Implicit conversion: LONG (LONG RAW) or a VARCHAR2(RAW) variable to a CLOB (BLOB) variable, and vice versa
- Explicit conversion:
  - TO_CLOB() converts LONG, VARCHAR2, and CHAR to CLOB
  - TO_BLOB() converts LONG RAW and RAW to BLOB
- Function and Procedure Parameter Passing:
  - CLOBs and BLOBs as actual parameters
  - VARCHAR2, LONG, RAW, and LONG RAW are formal parameters, and vice versa
- LOB data is acceptable in most of the SQL and PL/SQL operators and built-in functions

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The DBMS_LOB Package

- Working with LOB often requires the use of the Oracle-supplied package DBMS_LOB.
- DBMS_LOB provides routines to access and manipulate internal and external LOBs.
- Oracle9*i* enables retrieving LOB data directly using SQL, without using any special LOB API.
- In PL/SQL you can define a VARCHAR2 for a CLOB and a RAW for BLOB.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The DBMS_LOB Package

- **Modify LOB values:**

  APPEND, COPY, ERASE, TRIM, WRITE, LOADFROMFILE

- **Read or examine LOB values:**

  GETLENGTH, INSTR, READ, SUBSTR

- **Specific to BFILES:**

  FILECLOSE, FILECLOSEALL, FILEEXISTS, FILEGETNAME, FILEISOPEN, FILEOPEN

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

The DBMS_LOB Package

- NULL parameters get NULL returns.
- Offsets:
  - BLOB, BFILE: Measured in bytes
  - CLOB, NCLOB: Measured in characters
- There are no negative values for parameters.

Copyright © Oracle Corporation, 2001. All rights reserved.

DBMS_LOB.READ and DBMS_LOB.WRITE

```
PROCEDURE READ (
  lobsrc IN BFILE|BLOB|CLOB ,
  amount IN OUT BINARY_INTEGER,
  offset IN INTEGER,
  buffer OUT RAW|VARCHAR2 )
```

```
PROCEDURE WRITE (
  lobdst IN OUT BLOB|CLOB,
  amount IN OUT BINARY_INTEGER,
  offset IN INTEGER := 1,
  buffer IN RAW|VARCHAR2 )  -- RAW for BLOB
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Adding LOB Columns to a Table

```
ALTER TABLE employees ADD
  (resume       CLOB,
   picture      BLOB);
```

```
Table altered.
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Populating LOB Columns

Insert a row into a table with LOB columns:

```
INSERT INTO employees (employee_id, first_name,
    last_name, email, hire_date, job_id,
    salary, resume, picture)
VALUES (405, 'Marvin', 'Ellis', 'MELLIS', SYSDATE,
     'AD_ASST', 4000, EMPTY_CLOB(),NULL);
```

```
1 row created.
```

Initialize a LOB column using the EMPTY_BLOB() function:

```
UPDATE employees
SET resume = 'Date of Birth: 8 February 1951',
    picture = EMPTY_BLOB()
WHERE employee_id = 405;
```

```
1 row updated.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Updating LOB by Using SQL

UPDATE CLOB column

```
UPDATE employees
SET resume = 'Date of Birth: 1 June 1956'
WHERE employee_id = 170;
```

```
1 row updated.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Updating LOB by Using DBMS_LOB in PL/SQL

```
DECLARE
  lobloc CLOB;          -- serves as the LOB locator
  text   VARCHAR2(32767):='Resigned: 5 August 2000';
  amount NUMBER ;       -- amount to be written
  offset INTEGER;       -- where to start writing
BEGIN
  SELECT resume INTO lobloc
  FROM   employees
  WHERE  employee_id = 405 FOR UPDATE;
  offset := DBMS_LOB.GETLENGTH(lobloc) + 2;
  amount := length(text);
  DBMS_LOB.WRITE (lobloc, amount, offset, text );
  text   := ' Resigned: 30 September 2000';
  SELECT resume INTO lobloc
  FROM   employees
  WHERE  employee_id = 170 FOR UPDATE;
  amount := length(text);
  DBMS_LOB.WRITEAPPEND(lobloc, amount, text);
  COMMIT;
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Selecting CLOB Values by Using SQL

```
SELECT employee_id, last_name , resume -- CLOB
FROM employees
WHERE employee_id IN (405, 170);
```

EMPLOYEE_ID	LAST_NAME	RESUME
170	Fox	Date of Birth: 1 June 1956 Resigned = 30 September 2000
405	Ellis	Date of Birth: 8 February 1951 Resigned = 5 August 2000

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Selecting CLOB Values by Using DBMS_LOB

- DBMS_LOB.SUBSTR(lob_column, no_of_chars, starting)
- DBMS_LOB.INSTR (lob_column, pattern)

```
SELECT DBMS_LOB.SUBSTR (resume, 5, 18),
       DBMS_LOB.INSTR (resume,' = ')
FROM   employees
WHERE  employee_id IN (170, 405);
```

DBMS_LOB.SUBSTR(RESUME,5,18)	DBMS_LOB.INSTR(RESUME,'=')
June	36
Febru	40

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Selecting CLOB Values in PL/SQL

```
DECLARE
  text VARCHAR2(4001);
BEGIN
 SELECT resume INTO text
 FROM employees
 WHERE employee_id = 170;
 DBMS_OUTPUT.PUT_LINE('text is: '|| text);
END;
/
```

```
text is: Date of Birth: 1 June 1956 Resigned = 30 September 2000
PL/SQL procedure successfully completed.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing LOBS

Delete a row containing LOBS:

```
DELETE
FROM   employees
WHERE employee_id = 405;
```

```
1 row deleted.
```

Disassociate a LOB value from a row:

```
UPDATE employees
SET resume = EMPTY_CLOB()
WHERE employee_id = 170;
```

```
1 row updated.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Temporary LOBs

- Temporary LOBs:
  - Provide an interface to support creation of LOBs that act like local variables
  - Can be BLOBs, CLOBs, or NCLOBs
  - Are not associated with a specific table
  - Are created using DBMS_LOB.CREATETEMPORARY procedure
  - Use DBMS_LOB routines
- The lifetime of a temporary LOB is a session.
- Temporary LOBs are useful for transforming data in permanent internal LOBs.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Temporary LOB

PL/SQL procedure to create and test a temporary LOB:

```
CREATE OR REPLACE PROCEDURE IsTempLOBOpen
    (p_lob_loc IN OUT BLOB, p_retval OUT INTEGER)
IS
BEGIN
  -- create a temporary LOB
  DBMS_LOB.CREATETEMPORARY (p_lob_loc, TRUE);
  -- see if the LOB is open: returns 1 if open
  p_retval := DBMS_LOB.ISOPEN (p_lob_loc);
  DBMS_OUTPUT.PUT_LINE ('The file returned a value
                          ....' || p_retval);
  -- free the temporary LOB
  DBMS_LOB.FREETEMPORARY (p_lob_loc);
END;
/
```

```
Procedure created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Identify four built-in types for large objects: BLOB, CLOB, NCLOB, and BFILE
- Describe how LOBs replace LONG and LONG RAW
- Describe two storage options for LOBs:
  - The Oracle server (internal LOBs)
  - External host files (external LOBs)
- Use the DBMS_LOB PL/SQL package to provide routines for LOB management
- Use temporary LOBs in a session

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 15 Overview

This practice covers the following topics:

- Creating object types, using the new data types `CLOB` and `BLOB`
- Creating a table with `LOB` data types as columns
- Using the `DBMS_LOB` package to populate and interact with the `LOB` data

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

16

Creating Database Triggers

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe different types of triggers
- Describe database triggers and their use
- Create database triggers
- Describe database trigger firing rules
- Remove database triggers

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Types of Triggers

A trigger:

- Is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or the database
- Executes implicitly whenever a particular event takes place
- Can be either:
  - Application trigger: Fires whenever an event occurs with a particular application
  - Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

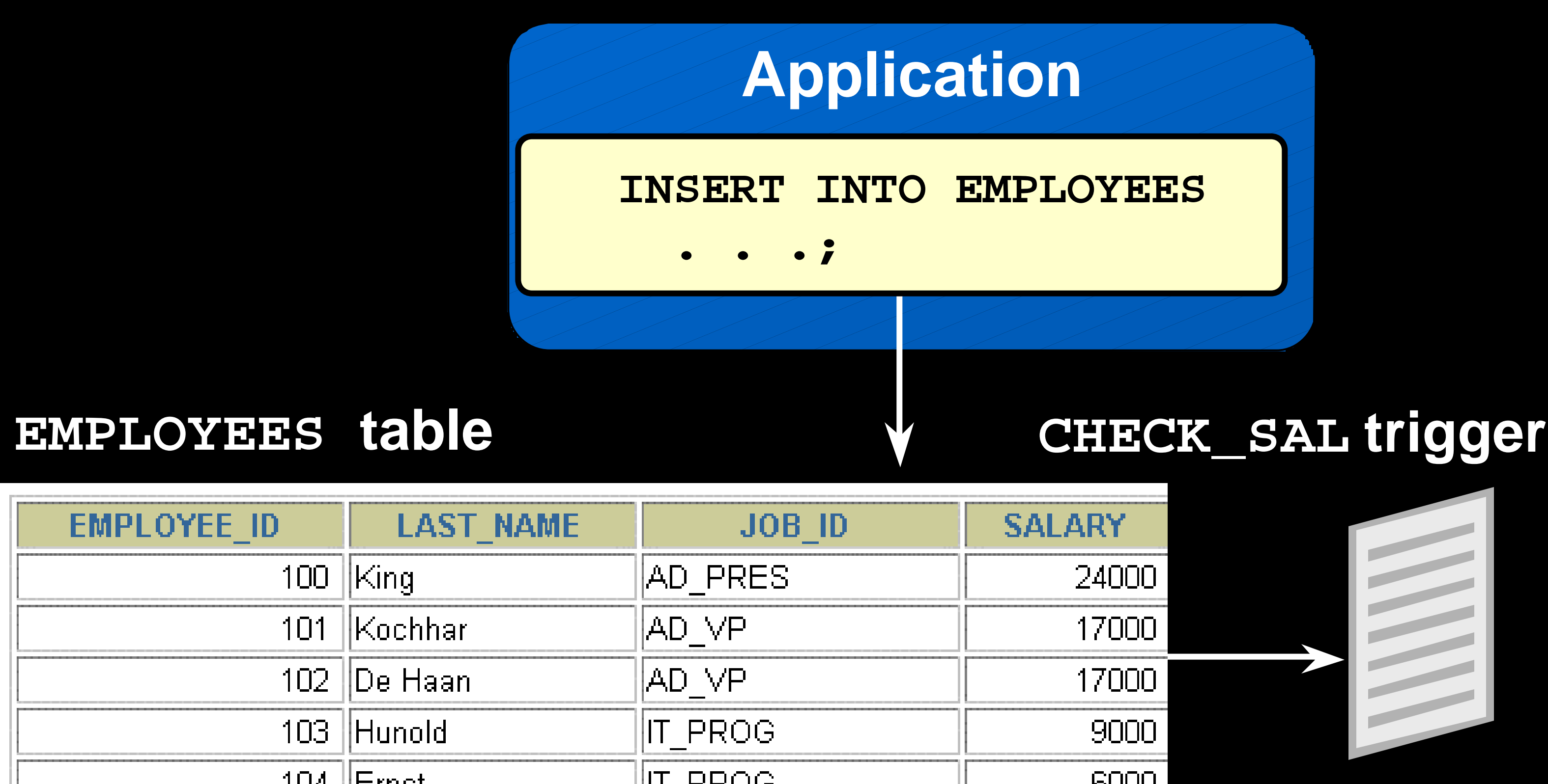
Guidelines for Designing Triggers

- Design triggers to:
  - Perform related actions
  - Centralize global operations
- Do not design triggers:
  - Where functionality is already built into the Oracle server
  - That duplicate other triggers
- Create stored procedures and invoke them in a trigger, if the PL/SQL code is very lengthy.
- The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Database Trigger: Example



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Creating DML Triggers

A triggering statement contains:

- Trigger timing
  - For table: `BEFORE`, `AFTER`
  - For view: `INSTEAD OF`
- Triggering event: `INSERT`, `UPDATE`, or `DELETE`
- Table name: On table, view
- Trigger type: Row or statement
- `WHEN` clause: Restricting condition
- Trigger body: PL/SQL block

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Trigger timing: When should the trigger fire?

- **`BEFORE`: Execute the trigger body before the triggering DML event on a table.**
- **`AFTER`: Execute the trigger body after the triggering DML event on a table.**
- **`INSTEAD OF`: Execute the trigger body instead of the triggering statement. This is used for views that are not otherwise modifiable.**

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Triggering user event: Which DML statement causes the trigger to execute? You can use any of the following:

- `INSERT`
- `UPDATE`
- `DELETE`

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Trigger type: Should the trigger body execute for each row the statement affects or only once?

- Statement: The trigger body executes once for the triggering event. This is the default. A statement trigger fires once, even if no rows are affected at all.
- Row: The trigger body executes once for each row affected by the triggering event. A row trigger is not executed if the triggering event affects no rows.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Trigger Components

Trigger body: What action should the trigger perform?

The trigger body is a PL/SQL block or a call to a procedure.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Firing Sequence

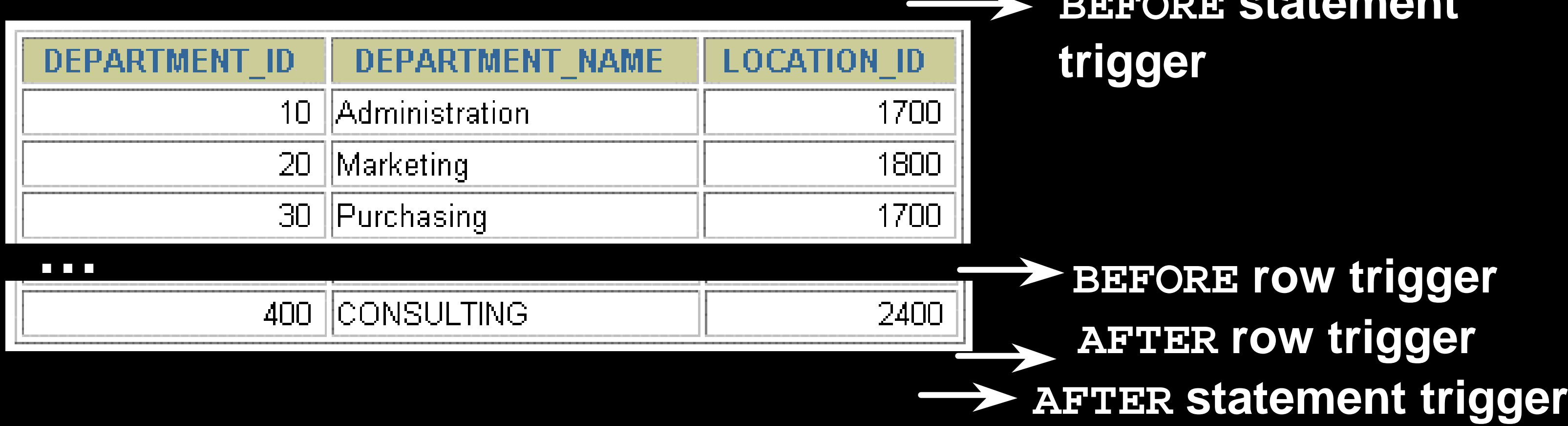
Use the following firing sequence for a trigger on a table, when a single row is manipulated:

DML statement

```
INSERT INTO departments (department_id,
                         department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```

```
1 row created.
```

Triggering action



DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
...		
400	CONSULTING	2400

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Firing Sequence

Use the following firing sequence for a trigger on a table, when many rows are manipulated:

```
UPDATE employees
    SET salary = salary * 1.1
    WHERE department_id = 30;
```

6 rows updated.

→ BEFORE statement trigger

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
114	Raphaely	30
115	Khoo	30
116	Baida	30
117	Tobias	30
118	Himuro	30
119	Colmenares	30

→ BEFORE row trigger
→ AFTER row trigger
...
→ BEFORE row trigger
→ AFTER row trigger
...

→ AFTER statement trigger

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Creating DML Statement Triggers

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
   timing
      event1 [OR event2 OR event3]
         ON table_name
trigger_body
```

Note: Trigger names must be unique with respect to other triggers in the same schema.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating DML Statement Triggers

Example:

```
CREATE OR REPLACE TRIGGER secure_emp
 BEFORE INSERT ON employees
 BEGIN
  IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
     (TO_CHAR(SYSDATE,'HH24:MI')
              NOT BETWEEN '08:00' AND '18:00')
   THEN RAISE_APPLICATION_ERROR (-20500,'You may
              insert into EMPLOYEES table only
                     during business hours.');
  END IF;
END;
/
```

```
Trigger created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Testing SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
            first_name, email, hire_date,
            job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
        'IT_PROG', 4500, 60);
```

```
INSERT INTO employees (employee_id, last_name, first_name, email,
            *
ERROR at line 1:
ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "PLSQL.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Conditional Predicates

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
 IF (TO_CHAR (SYSDATE,'DY') IN ('SAT','SUN')) OR
    (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '08' AND '18')
 THEN
   IF  DELETING THEN
     RAISE_APPLICATION_ERROR (-20502,'You may delete  from
            EMPLOYEES table only during business hours.');
   ELSIF  INSERTING THEN
     RAISE_APPLICATION_ERROR (-20500,'You may insert into
            EMPLOYEES table only during business hours.');
   ELSIF   UPDATING ('SALARY') THEN
     RAISE_APPLICATION_ERROR (-20503,'You may update
                 SALARY only during business hours.');
   ELSE
     RAISE_APPLICATION_ERROR (-20504,'You may update
            EMPLOYEES table only during normal hours.');
   END IF;
  END IF;
END;
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a DML Row Trigger

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
  timing
    event1 [OR event2 OR event3]
      ON table_name
  [REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW
  [WHEN (condition)]
trigger_body
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Creating DML Row Triggers

```
CREATE OR REPLACE TRIGGER restrict_salary
  BEFORE  INSERT OR UPDATE OF salary ON employees
    FOR EACH ROW
    BEGIN
      IF  NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
          AND :NEW.salary > 15000
      THEN
        RAISE_APPLICATION_ERROR (-20202,'Employee
                                 cannot earn this amount');
     END IF;
END;
/
```

```
Trigger created.
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Using OLD and NEW Qualifiers

```
CREATE OR REPLACE TRIGGER audit_emp_values
 AFTER DELETE OR INSERT OR UPDATE ON employees
 FOR EACH ROW
BEGIN
  INSERT INTO audit_emp_table (user_name, timestamp,
     id, old_last_name, new_last_name, old_title,
     new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
      :OLD.last_name, :NEW.last_name, :OLD.job_id,
      :NEW.job_id, :OLD.salary, :NEW.salary );
END;
/
```

```
Trigger created.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using OLD and NEW Qualifiers: Example Using Audit_Emp_Table

```
INSERT INTO employees
        (employee_id, last_name, job_id, salary, ...)
 VALUES (999, 'Temp emp', 'SA_REP', 1000, ...);

UPDATE employees
 SET salary = 2000, last_name = 'Smith'
 WHERE employee_id = 999;
```

```
1 row created.
1 row updated.
```

```
SELECT user_name, timestamp, ... FROM audit_emp_table
```

USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA_REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

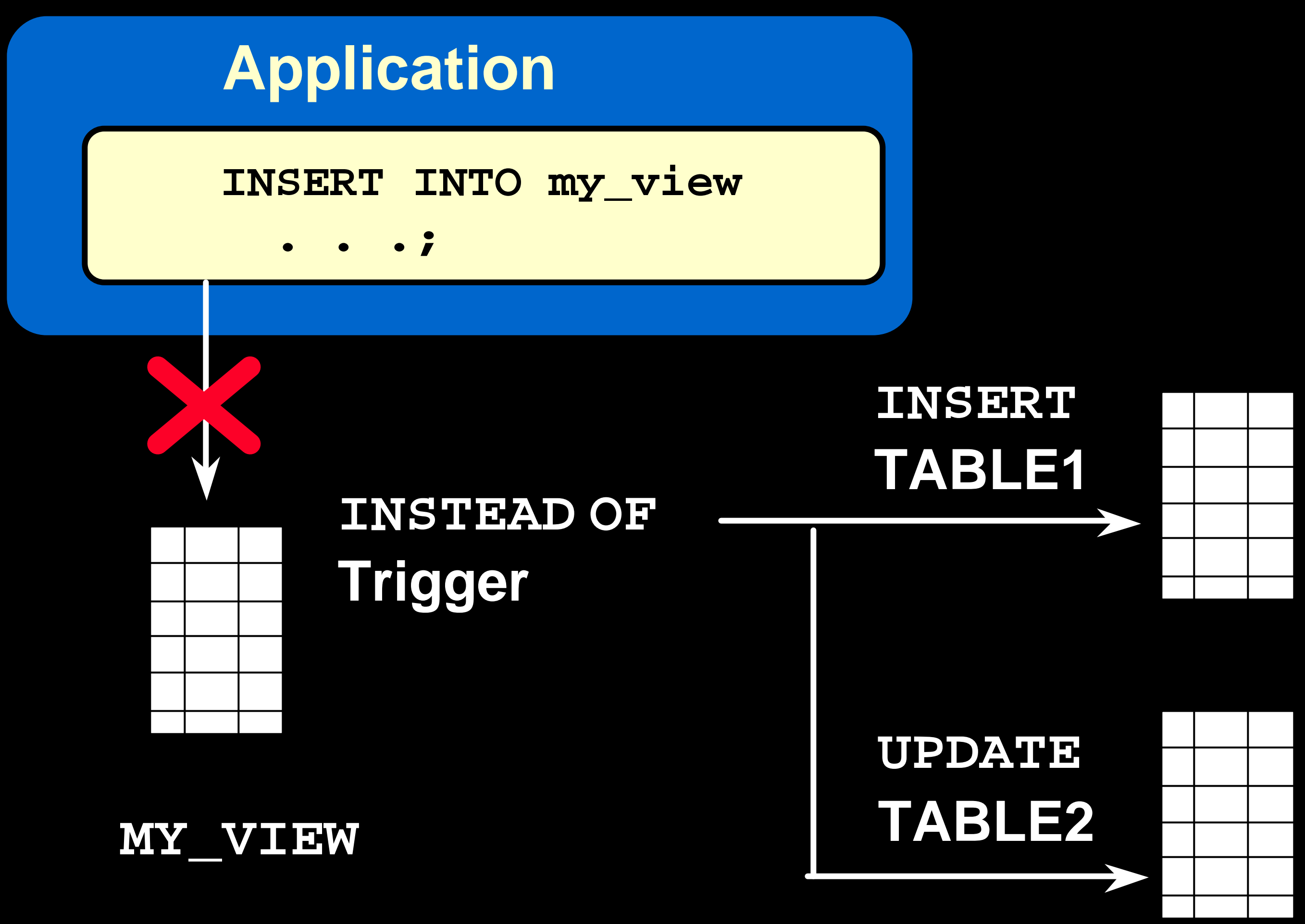
Restricting a Row Trigger

```
CREATE OR REPLACE TRIGGER derive_commission_pct
  BEFORE INSERT OR UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.job_id = 'SA_REP')
BEGIN
  IF  INSERTING
     THEN :NEW.commission_pct := 0;
  ELSIF :OLD.commission_pct IS NULL
     THEN :NEW.commission_pct := 0;
  ELSE
    :NEW.commission_pct := :OLD.commission_pct + 0.05;
  END IF;
END;
/
```

```
Trigger created.
```


Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

INSTEAD OF Triggers



 Copyright © Oracle Corporation, 2001. All rights reserved.

Creating an INSTEAD OF Trigger

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
  INSTEAD OF
    event1 [OR event2 OR event3]
      ON view_name
   [REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
trigger_body
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating an INSTEAD OF Trigger

INSERT into EMP_DETAILS that is based on EMPLOYEES and DEPARTMENTS tables

1

```
INSERT INTO emp_details(employee_id, ... )
VALUES(9001,'ABBOTT',3000,10,'abbott.mail.com','HR_MAN');
```

INSTEAD OF INSERT into EMP_DETAILS

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	EMAIL	JOB_
100	King	90	SKING	AD_PRE
101	Kochhar	90	NKOCHHAR	AD_VP
102	De Haan	90	LDEHAAN	AD_VP

...

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating an INSTEAD OF Trigger

INSERT into EMP_DETAILS that is based on EMPLOYEES and DEPARTMENTS tables

1

```
INSERT INTO emp_details(employee_id, ... )
VALUES(9001,'ABBOTT',3000,10,'abbott.mail.com','HR_MAN');
```

INSTEAD OF INSERT into EMP_DETAILS

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	EMAIL	JOB_
100	King	90	SKING	AD_PRE
101	Kochhar	90	NKOCHHAR	AD_VP
102	De Haan	90	LDEHAAN	AD_VP

...

2 INSERT into NEW_EMPS

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID	EMA
100	King	24000	90	SKING
101	Kochhar	17000	90	NKOCHH
102	De Haan	17000	90	LDEHAA
...				
9001	ABBOTT	3000	10	abbott.m

3 UPDATE NEW_DEPTS

DEPARTMENT_ID	DEPARTMENT_NAME	TOT_DEPT_SA
10	Administration	940
20	Marketing	1900
30	Purchasing	3012
40	Human Resources	650

...

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

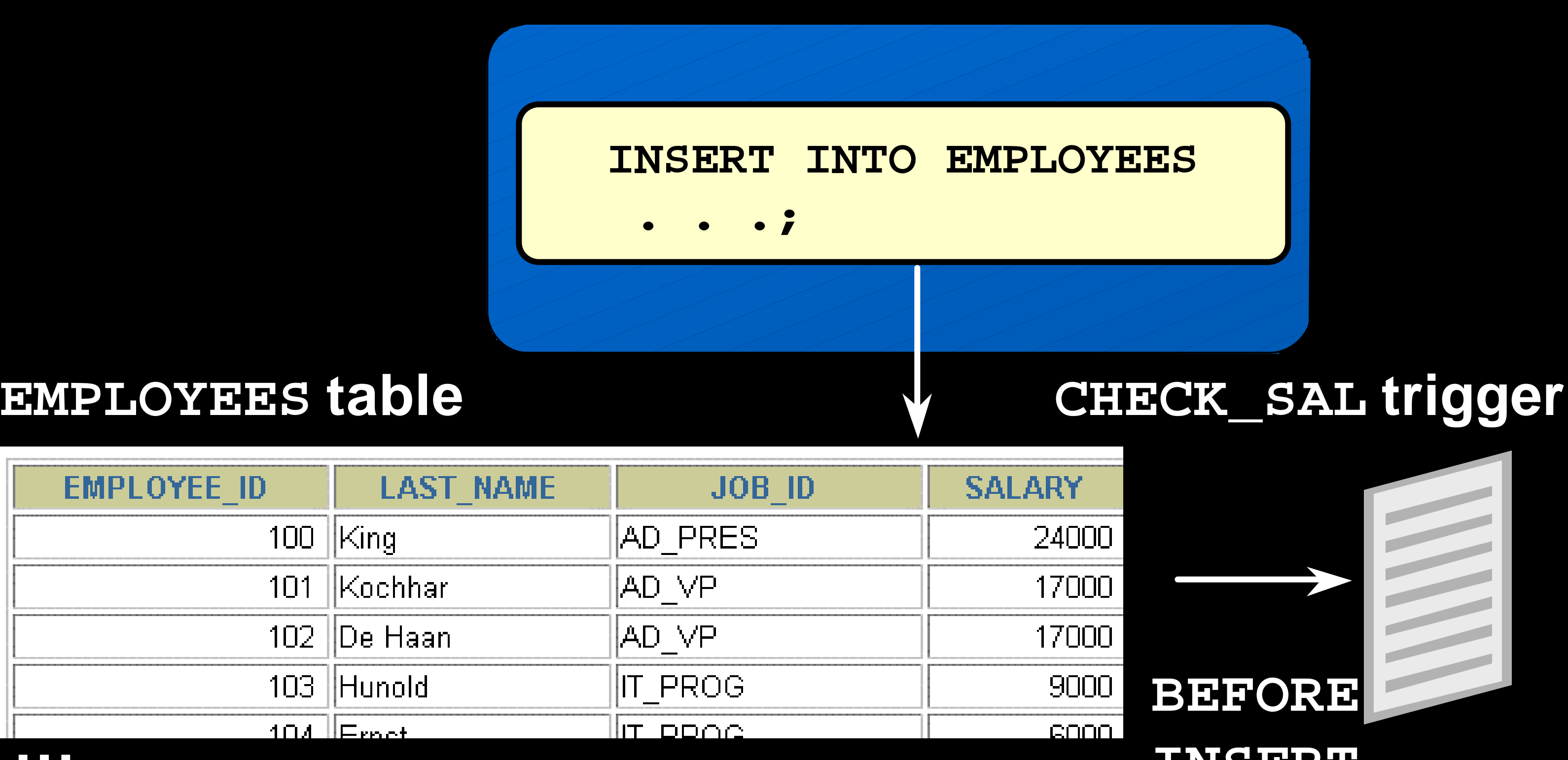
Differentiating Between Database Triggers and Stored Procedures

Triggers	Procedures
Defined with CREATE TRIGGER	Defined with CREATE PROCEDURE
Data dictionary contains source code in USER_TRIGGERS	Data dictionary contains source code in USER_SOURCE
Implicitly invoked	Explicitly invoked
COMMIT, SAVEPOINT, and ROLLBACK are not allowed	COMMIT, SAVEPOINT, and ROLLBACK are allowed

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Differentiating Between Database Triggers and Form Builder Triggers



EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000

...

Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

Managing Triggers

Disable or reenable a database trigger:

```
ALTER TRIGGER trigger_name  DISABLE | ENABLE
```

Disable or reenable all triggers for a table:

```
ALTER TABLE table_name   DISABLE | ENABLE  ALL TRIGGERS
```

Recompile a trigger for a table:

```
ALTER TRIGGER trigger_name COMPILE
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

`DROP TRIGGER` Syntax

To remove a trigger from the database, use the `DROP TRIGGER` syntax:

```
DROP TRIGGER trigger_name;
```

Example:

```
DROP TRIGGER secure_emp;
```

```
Trigger dropped.
```

Note: All triggers on a table are dropped when the table is dropped.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trigger Test Cases

- Test each triggering data operation, as well as nontriggering data operations.
- Test each case of the `WHEN` clause.
- Cause the trigger to fire directly from a basic data operation, as well as indirectly from a procedure.
- Test the effect of the trigger upon other triggers.
- Test the effect of other triggers upon the trigger.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trigger Execution Model and Constraint Checking

1. Execute all `BEFORE STATEMENT` triggers.
2. Loop for each row affected:
   a. Execute all `BEFORE ROW` triggers.
   b. Execute all `AFTER ROW` triggers.
3. Execute the DML statement and perform integrity constraint checking.
4. Execute all `AFTER STATEMENT` triggers.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Trigger Execution Model and Constraint Checking: Example

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Integrity constraint violation error
```

```
CREATE OR REPLACE TRIGGER constr_emp_trig
 AFTER UPDATE ON employees
  FOR EACH ROW
BEGIN
  INSERT INTO departments
    VALUES (999, 'dept999', 140, 2400);
END;
/
```

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Successful after trigger is fired
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

A Sample Demonstration for Triggers Using Package Constructs

DML into
EMPLOYEES table

AUDIT_EMP_TRIG
FOR EACH ROW
Increment variables

AUDIT_EMP_TAB
AFTER STATEMENT
Copy and then reset variables

VAR_PACK package

AUDIT_TABLE

1 2 3 4

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

After Row and After Statement Triggers

```
CREATE OR REPLACE TRIGGER audit_emp_trig
AFTER    UPDATE or INSERT or DELETE on EMPLOYEES
FOR EACH ROW
BEGIN
  IF      DELETING    THEN  var_pack.set_g_del(1);
  ELSIF   INSERTING   THEN  var_pack.set_g_ins(1);
  ELSIF   UPDATING ('SALARY')
                     THEN  var_pack.set_g_up_sal(1);
  ELSE    var_pack.set_g_upd(1);
  END IF;
END audit_emp_trig;
/
```

```
CREATE OR REPLACE TRIGGER audit_emp_tab
AFTER    UPDATE or INSERT or DELETE on employees
BEGIN
  audit_emp;
END audit_emp_tab;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Demonstration: VAR_PACK Package Specification

var_pack.sql

```
CREATE OR REPLACE PACKAGE var_pack
IS
-- these functions are used to return the
-- values of package variables
  FUNCTION g_del RETURN NUMBER;
  FUNCTION g_ins RETURN NUMBER;
  FUNCTION g_upd RETURN NUMBER;
  FUNCTION g_up_sal RETURN NUMBER;
-- these procedures are used to modify the
-- values of the package variables
  PROCEDURE set_g_del    (p_val  IN  NUMBER);
  PROCEDURE set_g_ins    (p_val  IN  NUMBER);
  PROCEDURE set_g_upd    (p_val  IN  NUMBER);
  PROCEDURE set_g_up_sal (p_val  IN  NUMBER);
END var_pack;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Demonstration: Using the AUDIT_EMP Procedure

```
CREATE OR REPLACE PROCEDURE audit_emp  IS
  v_del     NUMBER   := var_pack.g_del;
  v_ins     NUMBER   := var_pack.g_ins;
  v_upd     NUMBER   := var_pack.g_upd;
  v_up_sal  NUMBER   := var_pack.g_up_sal;
BEGIN
  IF  v_del + v_ins + v_upd != 0  THEN
    UPDATE audit_table SET
      del = del + v_del, ins = ins + v_ins,
      upd = upd + v_upd
    WHERE user_name=USER AND tablename='EMPLOYEES'
    AND   column_name IS NULL;
  END IF;
  IF v_up_sal != 0   THEN
    UPDATE audit_table SET upd = upd + v_up_sal
    WHERE user_name=USER AND tablename='EMPLOYEES'
    AND   column_name = 'SALARY';
  END IF;
-- resetting global variables in package VAR_PACK
  var_pack.set_g_del (0); var_pack.set_g_ins (0);
  var_pack.set_g_upd (0); var_pack.set_g_up_sal (0);
END audit_emp;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

Procedure

xxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvvv
xxxxxxxxxxxxxxxxxx

Package

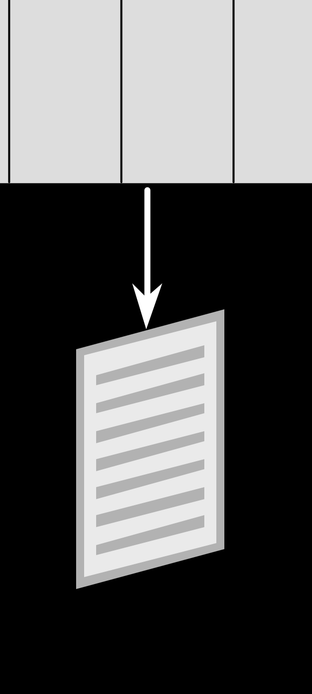
Procedure A declaration

Procedure B definition

Procedure A definition

Local variable

Trigger



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 16 Overview

This practice covers the following topics:

- Creating statement and row triggers
- Creating advanced triggers to add to the capabilities of the Oracle database

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

More Trigger Concepts

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Create additional database triggers
- Explain the rules governing triggers
- Implement triggers

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Database Triggers

- Triggering user event:
  - `CREATE`, `ALTER`, or `DROP`
  - Logging on or off
- Triggering database or system event:
  - Shutting down or starting up the database
  - A specific error (or any error) being raised

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Triggers on DDL Statements

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
   timing
      [ddl_event1 [OR ddl_event2 OR ...]]
         ON {DATABASE|SCHEMA}
trigger_body
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Triggers on System Events

```
CREATE [OR REPLACE] TRIGGER trigger_name
   timing
      [database_event1 [OR database_event2 OR ...]]
         ON {DATABASE|SCHEMA}
trigger_body
```


ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

LOGON and LOGOFF Trigger Example

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON  ON  SCHEMA
BEGIN
 INSERT INTO log_trig_table(user_id, log_date, action)
 VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF  ON  SCHEMA
BEGIN
 INSERT INTO log_trig_table(user_id, log_date, action)
 VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

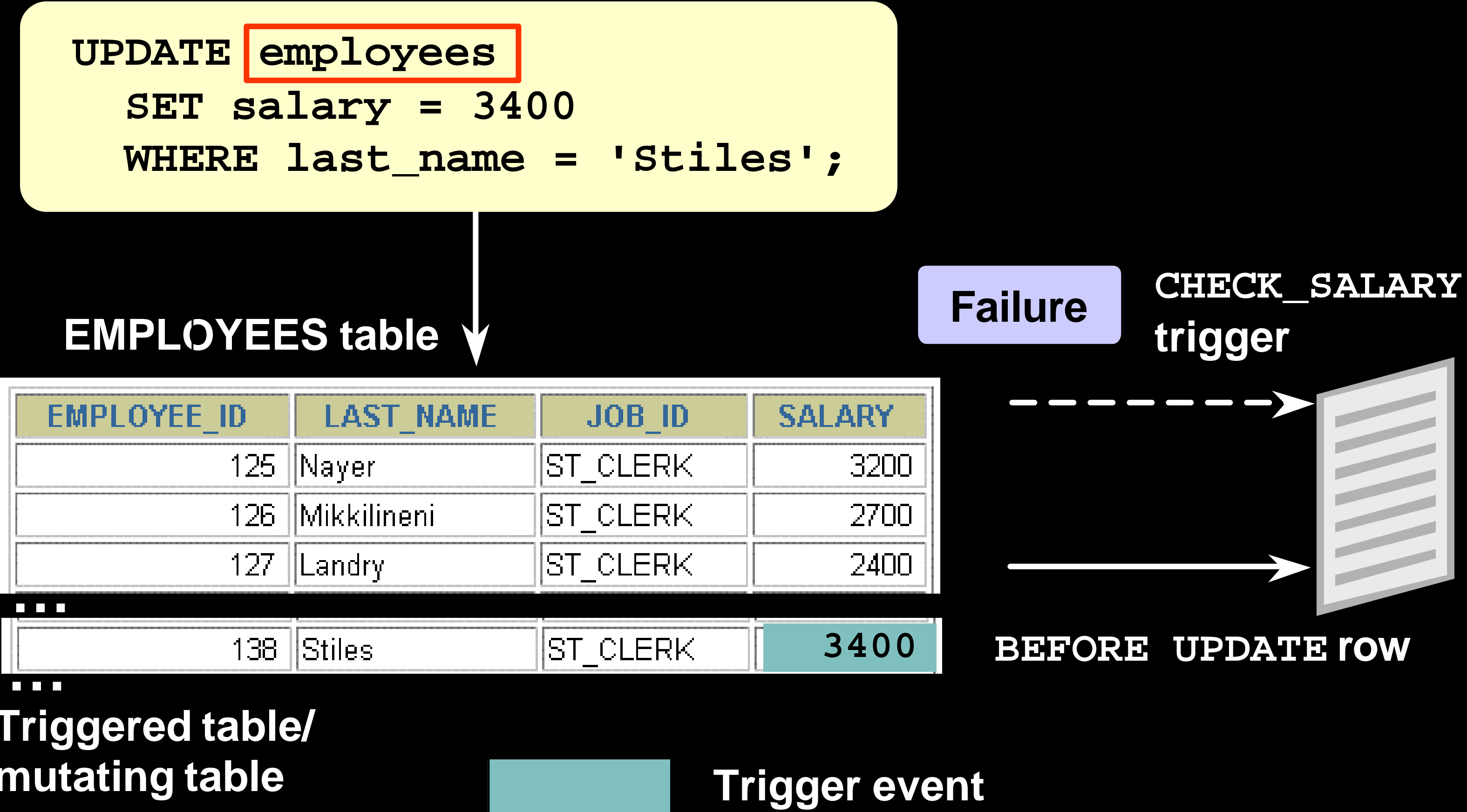
CALL Statements

```
CREATE [OR REPLACE] TRIGGER trigger_name
  timing
    event1 [OR event2 OR event3]
      ON table_name
  [REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
  [WHEN condition]
CALL procedure_name
```

```
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
  CALL log_execution
/
```


Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

Reading Data from a Mutating Table



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Mutating Table: Example

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  v_minsalary employees.salary%TYPE;
  v_maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
   INTO  v_minsalary, v_maxsalary
   FROM  employees
   WHERE job_id = :NEW.job_id;
  IF :NEW.salary < v_minsalary OR
     :NEW.salary > v_maxsalary THEN
     RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Mutating Table: Example

```
UPDATE employees
 SET salary = 3400
 WHERE last_name = 'Stiles';
```

```
UPDATE employees
     *
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "PLSQL.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'PLSQL.CHECK_SALARY'
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Implementing Triggers

You can use trigger for:

- Security
- Auditing
- Data integrity
- Referential integrity
- Table replication
- Computing derived data automatically
- Event logging

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Security Within the Server

```
GRANT SELECT, INSERT, UPDATE, DELETE
 ON    employees
 TO    clerk;              -- database role
GRANT clerk TO scott;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling Security with a Database Trigger

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
  v_dummy VARCHAR2(1);
BEGIN
 IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT','SUN'))
  THEN RAISE_APPLICATION_ERROR (-20506,'You may only
        change data during normal business hours.');
 END IF;
 SELECT COUNT(*) INTO v_dummy FROM holiday
 WHERE holiday_date = TRUNC (SYSDATE);
 IF v_dummy > 0 THEN RAISE_APPLICATION_ERROR(-20507,
           'You may not change data on a holiday.');
 END IF;
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the Server Facility to Audit Data Operations

```
AUDIT INSERT, UPDATE, DELETE
   ON  departments
   BY ACCESS
WHENEVER SUCCESSFUL;
```

```
Audit succeeded.
```

The Oracle server stores the audit information in a data dictionary table or operating system file.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Auditing by Using a Trigger

```
CREATE OR REPLACE TRIGGER audit_emp_values
 AFTER DELETE OR INSERT OR UPDATE ON employees
 FOR EACH ROW
BEGIN
 IF (audit_emp_package.g_reason IS NULL) THEN
    RAISE_APPLICATION_ERROR (-20059, 'Specify a reason
     for the data operation through the procedure SET_REASON
     of the AUDIT_EMP_PACKAGE before proceeding.');
 ELSE
    INSERT INTO audit_emp_table (user_name, timestamp, id,
      old_last_name, new_last_name, old_title, new_title,
      old_salary, new_salary, comments)
    VALUES (USER, SYSDATE, :OLD.employee_id, :OLD.last_name,
      :NEW.last_name, :OLD.job_id, :NEW.job_id, :OLD.salary,
      :NEW.salary, audit_emp_package.g_reason);
 END IF;
END;
```

```
CREATE OR REPLACE TRIGGER cleanup_audit_emp
 AFTER INSERT OR UPDATE OR DELETE ON employees
BEGIN
  audit_emp_package.g_reason := NULL;
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Enforcing Data Integrity Within the Server

```
ALTER TABLE employees ADD
  CONSTRAINT ck_salary CHECK (salary >= 500);
```

```
Table altered.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Protecting Data Integrity with a Trigger

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20508,
      'Do not decrease salary.');
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Enforcing Referential Integrity Within the Server

```
ALTER TABLE employees
  ADD CONSTRAINT emp_deptno_fk
  FOREIGN KEY (department_id)
     REFERENCES departments(department_id)
ON DELETE CASCADE;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Protecting Referential Integrity with a Trigger

```
CREATE OR REPLACE TRIGGER cascade_updates
 AFTER UPDATE OF department_id ON departments
 FOR EACH ROW
BEGIN
  UPDATE employees
   SET employees.department_id=:NEW.department_id
   WHERE employees.department_id=:OLD.department_id;
  UPDATE job_history
   SET department_id=:NEW.department_id
   WHERE department_id=:OLD.department_id;
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Replicating a Table Within the Server

```
CREATE SNAPSHOT emp_copy AS
  SELECT * FROM employees@ny;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Replicating a Table with a Trigger

```
CREATE OR REPLACE TRIGGER emp_replica
 BEFORE INSERT OR UPDATE ON employees
 FOR EACH ROW
BEGIN /*Only proceed if user initiates a data operation,
        NOT through the cascading trigger.*/
   IF INSERTING THEN
     IF :NEW.flag IS NULL THEN
       INSERT INTO employees@sf
       VALUES(:new.employee_id, :new.last_name,..., 'B');
       :NEW.flag := 'A';
     END IF;
     ELSE   /* Updating. */
      IF :NEW.flag = :OLD.flag THEN
        UPDATE employees@sf
        SET ename = :NEW.last_name, ...,
            flag = :NEW.flag
        WHERE employee_id = :NEW.employee_id;
      END IF;
      IF :OLD.flag = 'A' THEN :NEW.flag := 'B';
      ELSE :NEW.flag := 'A';
      END IF;
    END IF;
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Computing Derived Data Within the Server

```
UPDATE departments
 SET total_sal=(SELECT SUM(salary)
                FROM employees
                WHERE employees.department_id =
                     departments.department_id);
```


Copyright © Oracle Corporation, 2001. All rights reserved.

Computing Derived Values with a Trigger

```
CREATE OR REPLACE PROCEDURE increment_salary
  (p_id      IN departments.department_id%TYPE,
   p_salary IN departments.total_sal%TYPE)
IS
BEGIN
   UPDATE departments
   SET    total_sal = NVL (total_sal, 0)+ p_salary
   WHERE  department_id = p_id;
END increment_salary;
```

```
CREATE OR REPLACE TRIGGER compute_salary
AFTER INSERT OR UPDATE OF salary OR DELETE ON employees
FOR EACH ROW
BEGIN
 IF DELETING THEN
    increment_salary(:OLD.department_id,(-1*:OLD.salary));
 ELSIF  UPDATING THEN
 increment_salary(:NEW.department_id,(:NEW.salary-:OLD.salary))
 ELSE increment_salary(:NEW.department_id,:NEW.salary);--INSERT
 END IF;
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Logging Events with a Trigger

```
CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF quantity_on_hand, reorder_point
 ON inventories  FOR EACH ROW
 DECLARE
 v_descrip product_descriptions.product_description%TYPE;
 v_msg_text VARCHAR2(2000);
 stat_send  number(1);
BEGIN
  IF  :NEW.quantity_on_hand <= :NEW.reorder_point THEN
   SELECT product_description INTO v_descrip
   FROM product_descriptions
   WHERE product_id = :NEW.product_id;
   v_msg_text := 'ALERT: INVENTORY LOW ORDER:'||CHR(10)||
   ...'Yours,' ||CHR(10) ||user || '.'|| CHR(10)|| CHR(10);
  ELSIF
   :OLD.quantity_on_hand < :NEW.quantity_on_hand THEN NULL;
  ELSE
   v_msg_text := 'Product #'||... CHR(10);
  END IF;
  DBMS_PIPE.PACK_MESSAGE(v_msg_text);
  stat_send := DBMS_PIPE.SEND_MESSAGE('INV_PIPE');
END;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of Database Triggers

- Improved data security:
  - Provide enhanced and complex security checks
  - Provide enhanced and complex auditing
- Improved data integrity:
  - Enforce dynamic data integrity constraints
  - Enforce complex referential integrity constraints
  - Ensure that related operations are performed together implicitly

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Managing Triggers

The following system privileges are required to manage triggers:

- The `CREATE/ALTER/DROP (ANY) TRIGGER` privilege enables you to create a trigger in any schema
- The `ADMINISTER DATABASE TRIGGER` privilege enables you to create a trigger on `DATABASE`
- The `EXECUTE` privilege (if your trigger refers to any objects that are not in your schema)

Note: Statements in the trigger body operate under the privilege of the trigger owner, not the trigger user.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Viewing Trigger Information

You can view the following trigger information:

- `USER_OBJECTS` data dictionary view: object information
- `USER_TRIGGERS` data dictionary view: the text of the trigger
- `USER_ERRORS` data dictionary view: PL/SQL syntax errors (compilation errors) of the trigger

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using USER_TRIGGERS*

Column	Column Description
TRIGGER_NAME	Name of the trigger
TRIGGER_TYPE	The type is BEFORE, AFTER, INSTEAD OF
TRIGGERING_EVENT	The DML operation firing the trigger
TABLE_NAME	Name of the database table
REFERENCING_NAMES	Name used for :OLD and :NEW
WHEN_CLAUSE	The when_clause used
STATUS	The status of the trigger
TRIGGER_BODY	The action to take

* Abridged column list

Copyright © Oracle Corporation, 2001. All rights reserved.

Listing the Code of Triggers

```
SELECT trigger_name, trigger_type, triggering_event,
         table_name, referencing_names,
         status, trigger_body
FROM   user_triggers
WHERE  trigger_name = 'RESTRICT_SALARY';
```

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_NAME	REFERENCING_NAMES	WHEN_CLAUS	STATUS	TRIGGER_BODY
RESTRICT_SALARY	BEFORE EACH ROW	INSERT OR UPDATE	EMPLOYEES	REFERENCING NEW AS NEW OLD AS OLD		ENABLED	BEGIN IF NOT (:NEW.JOB_ ID IN ('AD_PRES ', 'AD_VP')) AND :NE W.SAL

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Use advanced database triggers
- List mutating and constraining rules for triggers
- Describe the real-world application of triggers
- Manage triggers
- View trigger information

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 17 Overview

This practice covers creating advanced triggers to add to the capabilities of the Oracle database.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Managing Dependencies

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Track procedural dependencies
- Predict the effect of changing a database object upon stored procedures and functions
- Manage procedural dependencies

ORACLE

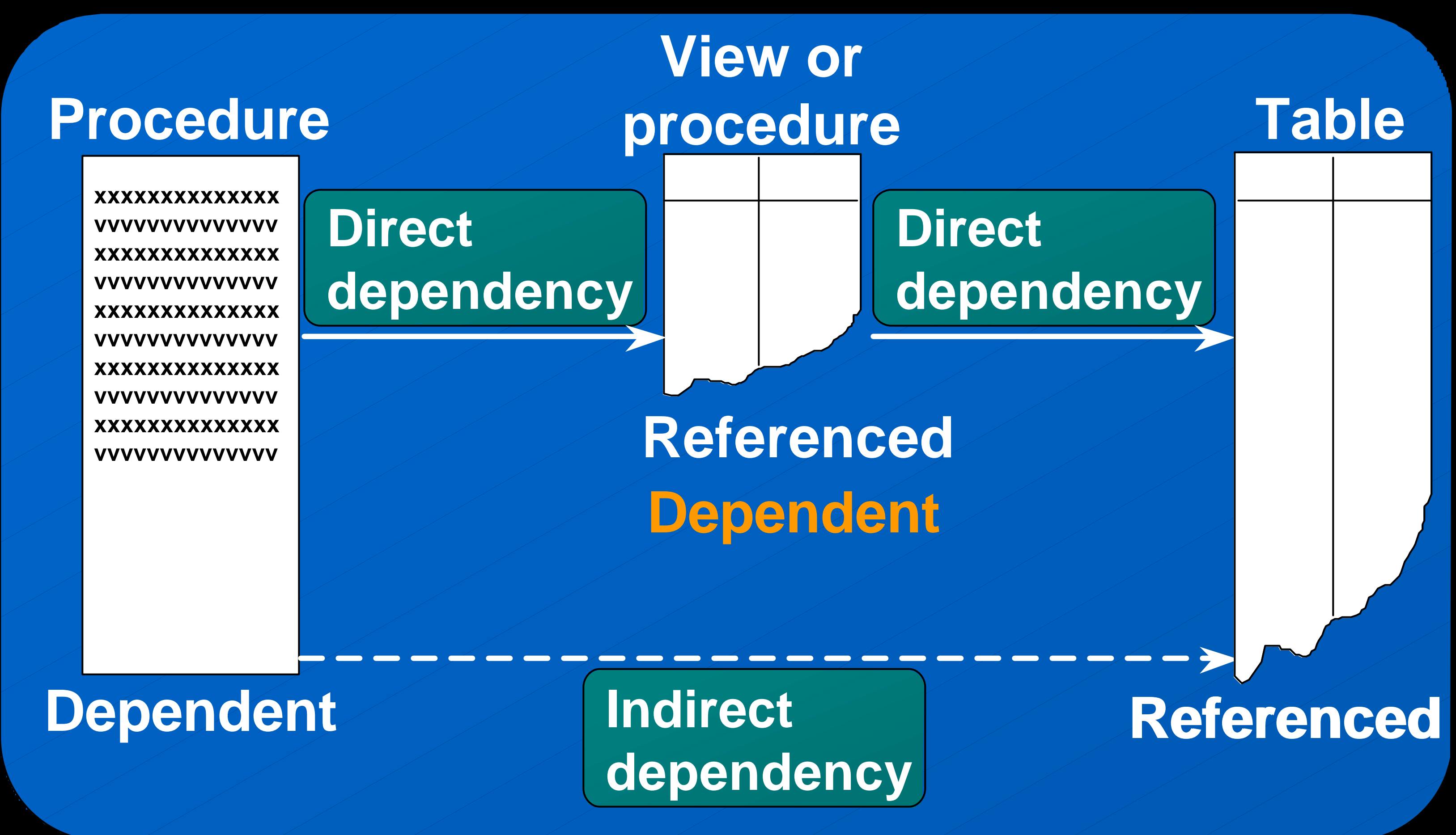
Copyright © Oracle Corporation, 2001. All rights reserved.

Understanding Dependencies

Dependent Objects	Referenced Objects
Table	Function
View	Package Specification
Database Trigger	Procedure
Procedure	Sequence
Function	Synonym
Package Body	Table
Package Specification	View
User-Defined Object and Collection Types	User-Defined Object and Collection Types

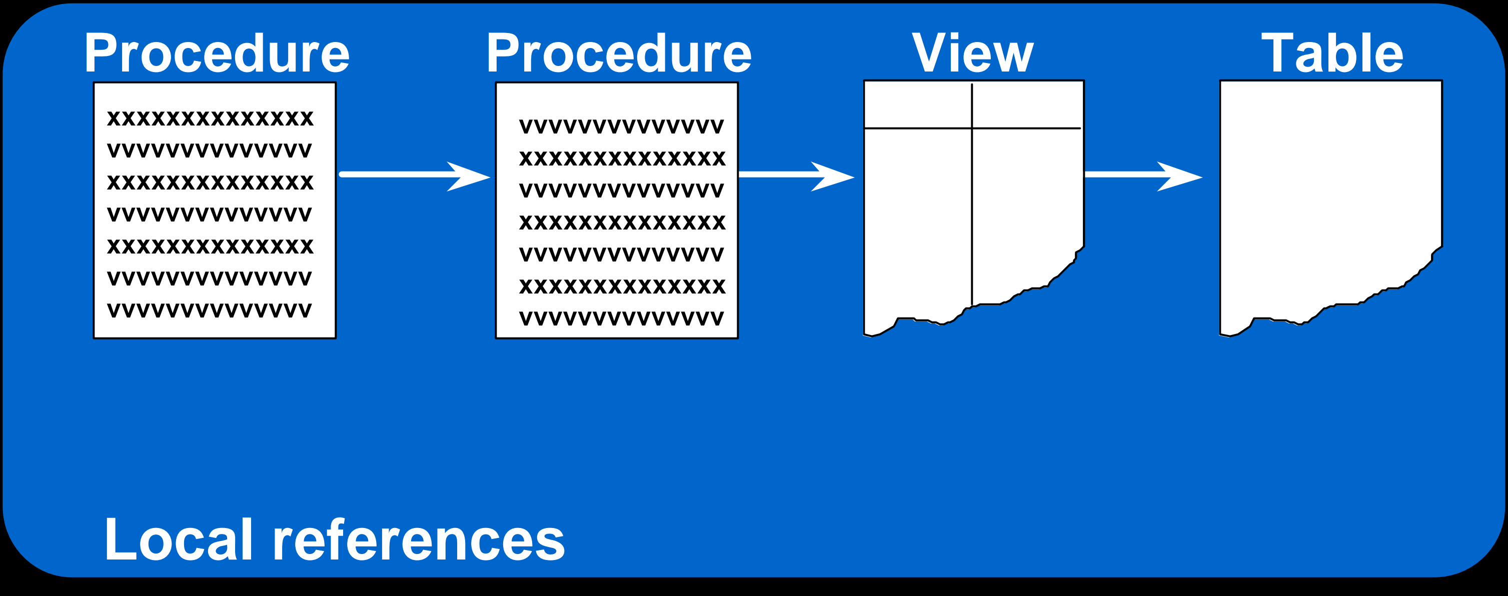
Copyright © Oracle Corporation, 2001. All rights reserved.

Dependencies



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

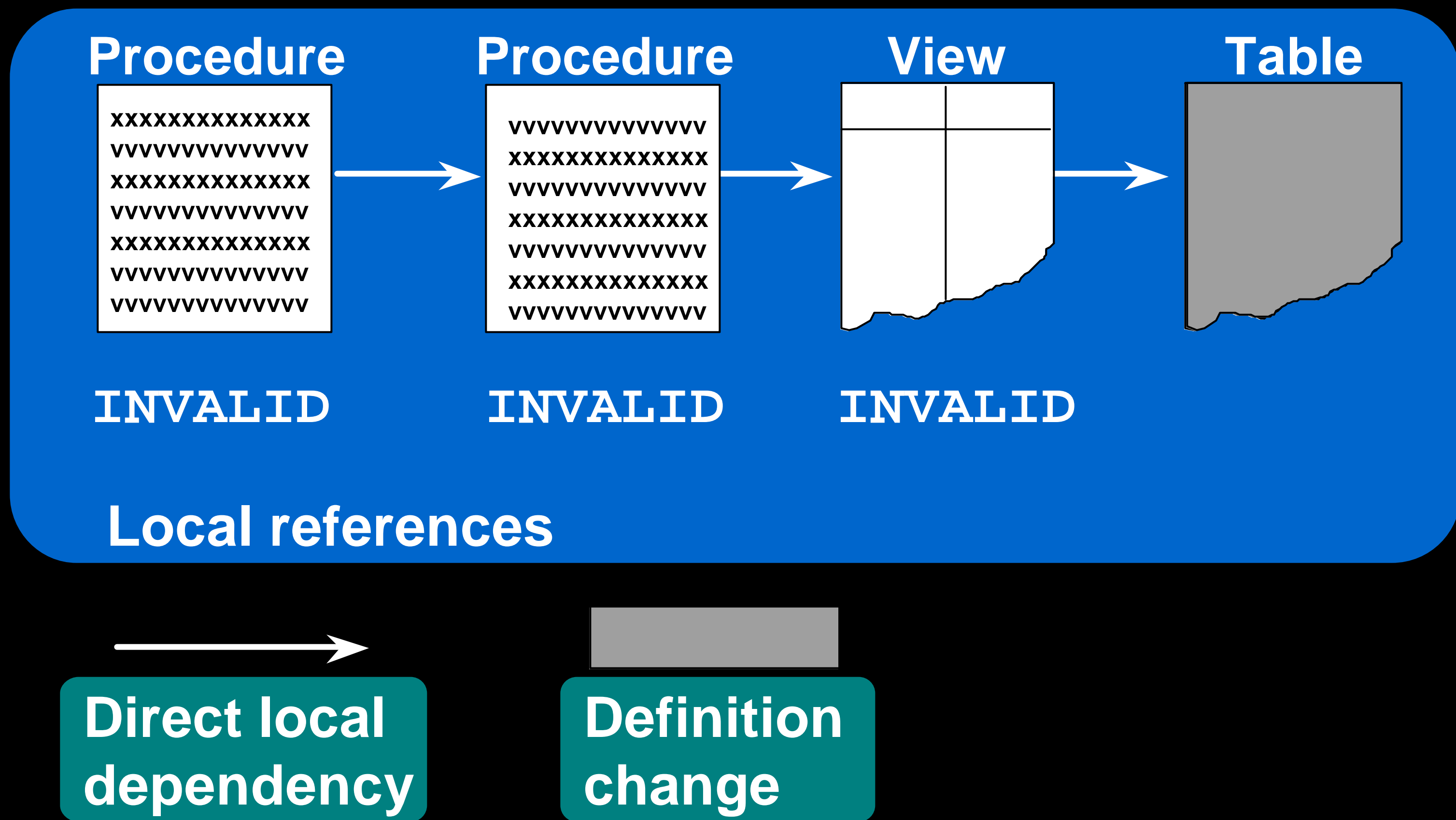
Local Dependencies



Direct local dependency

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Local Dependencies

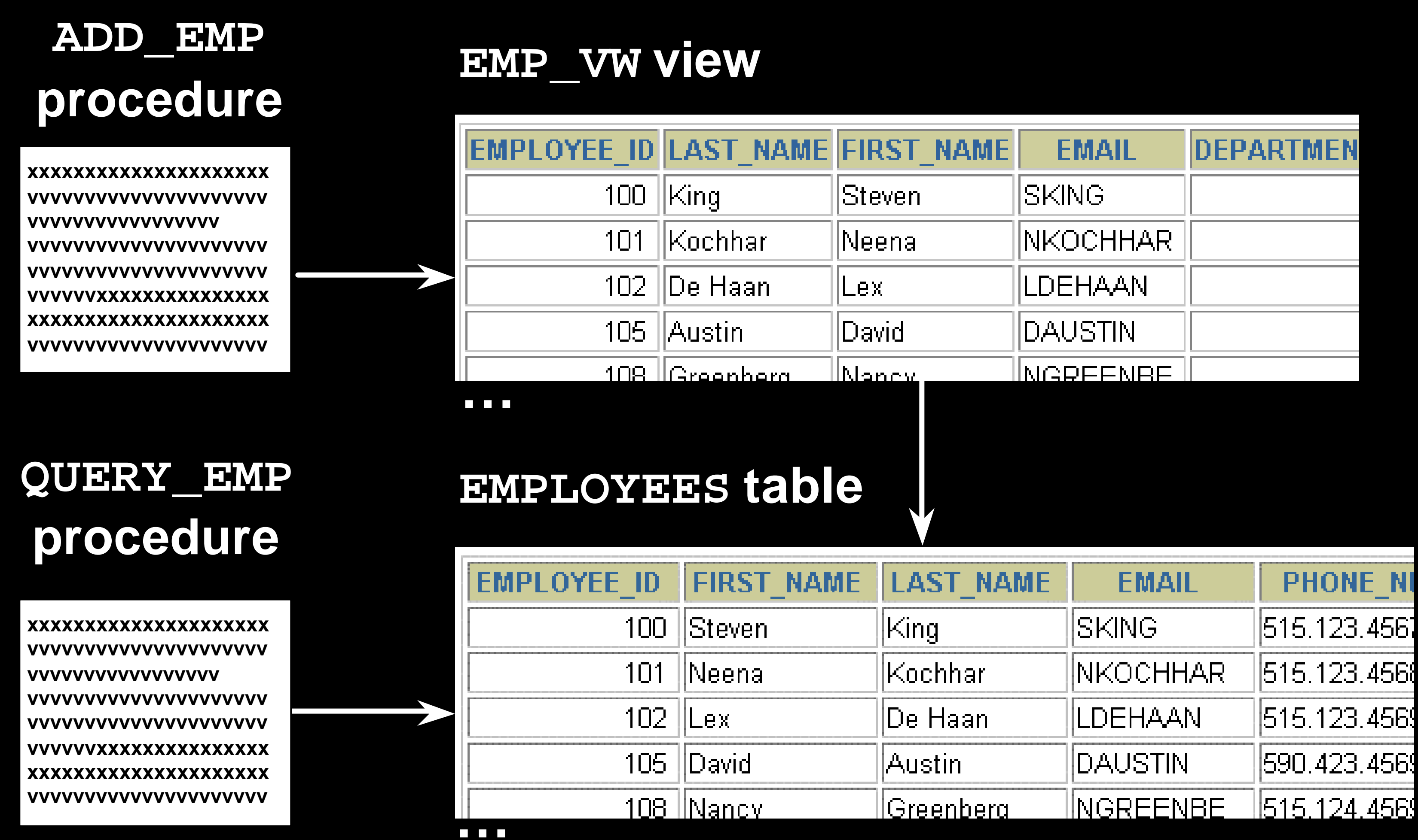


The Oracle server implicitly recompiles any `INVALID` object when the object is next called.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

A Scenario of Local Dependencies



EMPLOYEE_ID	LAST_NAME	FIRST_NAME	EMAIL	DEPARTMEN
100	King	Steven	SKING	
101	Kochhar	Neena	NKOCHHAR	
102	De Haan	Lex	LDEHAAN	
105	Austin	David	DAUSTIN	
108	Greenberg	Nancy	NGREENBE	

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NU
100	Steven	King	SKING	515.123.4567
101	Neena	Kochhar	NKOCHHAR	515.123.4568
102	Lex	De Haan	LDEHAAN	515.123.4569
105	David	Austin	DAUSTIN	590.423.4569
108	Nancy	Greenberg	NGREENBE	515.124.4569

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Displaying Direct Dependencies by Using `USER_DEPENDENCIES`

```
SELECT name, type, referenced_name, referenced_type
FROM   user_dependencies
WHERE  referenced_name IN ('EMPLOYEES','EMP_VW' );
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_T
EMP_DETAILS_VIEW	VIEW	EMPLOYEES	TABLE
...			
EMP_VW	VIEW	EMPLOYEES	TABLE
...			
QUERY_EMP	PROCEDURE	EMPLOYEES	TABLE
ADD_EMP	PROCEDURE	EMP_VW	VIEW

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Displaying Direct and Indirect Dependencies

1. Run the script `utldtree.sql` that creates the objects that enable you to display the direct and indirect dependencies.
2. Execute the `DEPTREE_FILL` procedure.

```
EXECUTE deptree_fill('TABLE','SCOTT','EMPLOYEES')
```

```
PL/SQL procedure successfully completed.
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Displaying Dependencies

DEPTREE View

```
SELECT   nested_level, type, name
FROM     deptree
ORDER BY seq#;
```

NESTED_LEVEL	TYPE	NAME
0	TABLE	EMPLOYEES
1	VIEW	EMP_DETAILS_VIEW

...

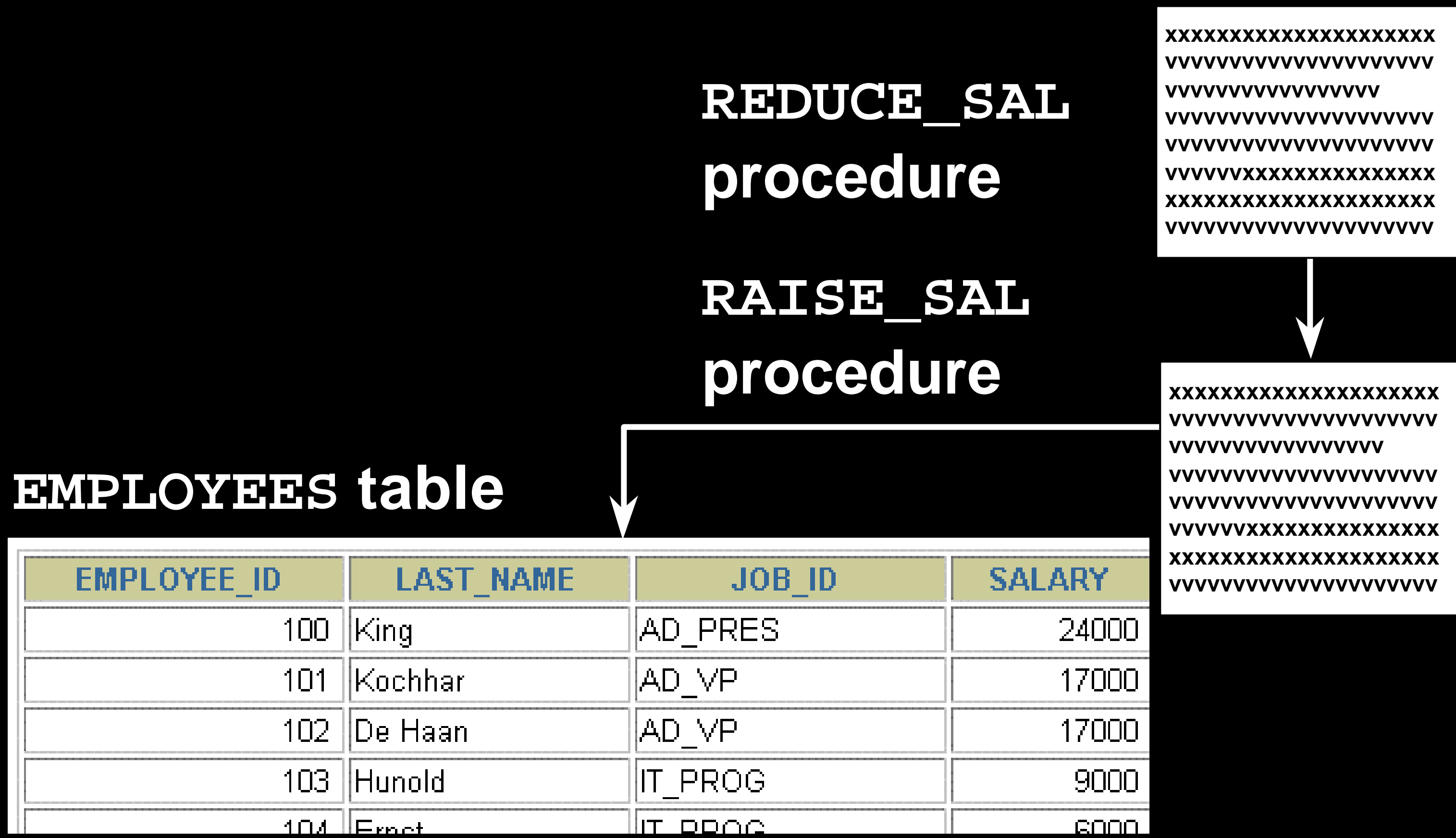
NESTED_LEVEL	TYPE	NAME
1	TRIGGER	CHECK_SALARY
1	VIEW	EMP_VW
2	PROCEDURE	ADD_EMP
1	PACKAGE	MGR_CONSTRAINTS_PKG
2	TRIGGER	CHECK_PRES_TITLE

...

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Another Scenario of Local Dependencies



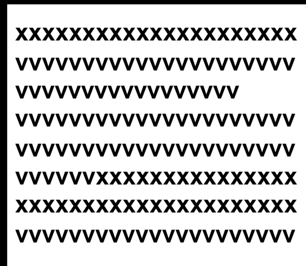
EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000

...

Copyright © Oracle Corporation, 2001. All rights reserved.

A Scenario of Local Naming Dependencies

`QUERY_EMP` **procedure**



`EMPLOYEES` **public synonym**

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
[illegible]	[illegible]	[illegible]	[illegible]

...

`EMPLOYEES` **table**

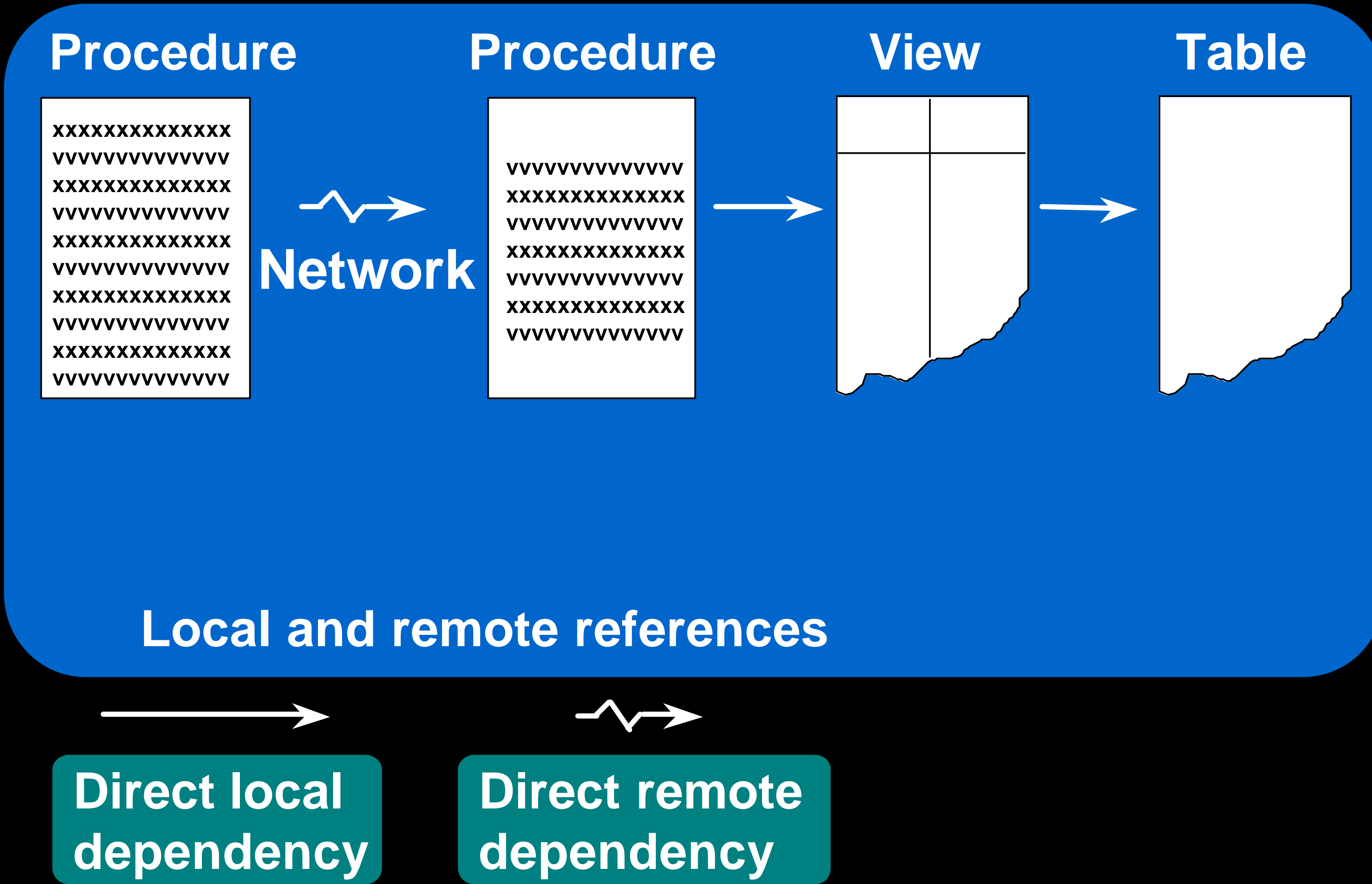
EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
[illegible]	[illegible]	[illegible]	[illegible]

...

ORACLE

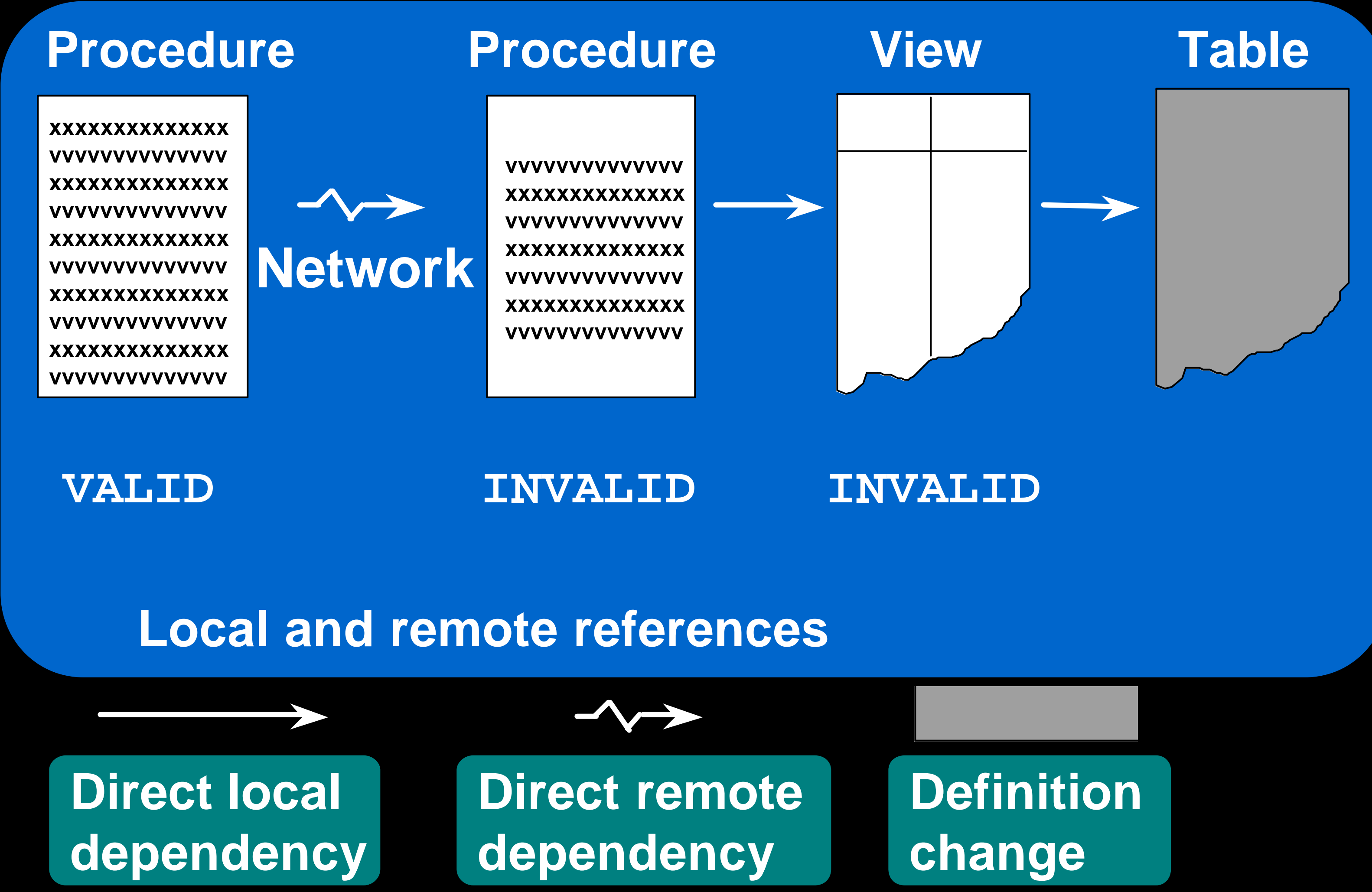
Copyright © Oracle Corporation, 2001. All rights reserved.

Understanding Remote Dependencies



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Understanding Remote Dependencies



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Concepts of Remote Dependencies

Remote dependencies are governed by the mode chosen by the user:

- `TIMESTAMP` checking
- `SIGNATURE` checking

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

REMOTE_DEPENDENCIES_MODE Parameter

Setting REMOTE_DEPENDENCIES_MODE:

- As an init.ora parameter

```
REMOTE_DEPENDENCIES_MODE = value
```

- At the system level

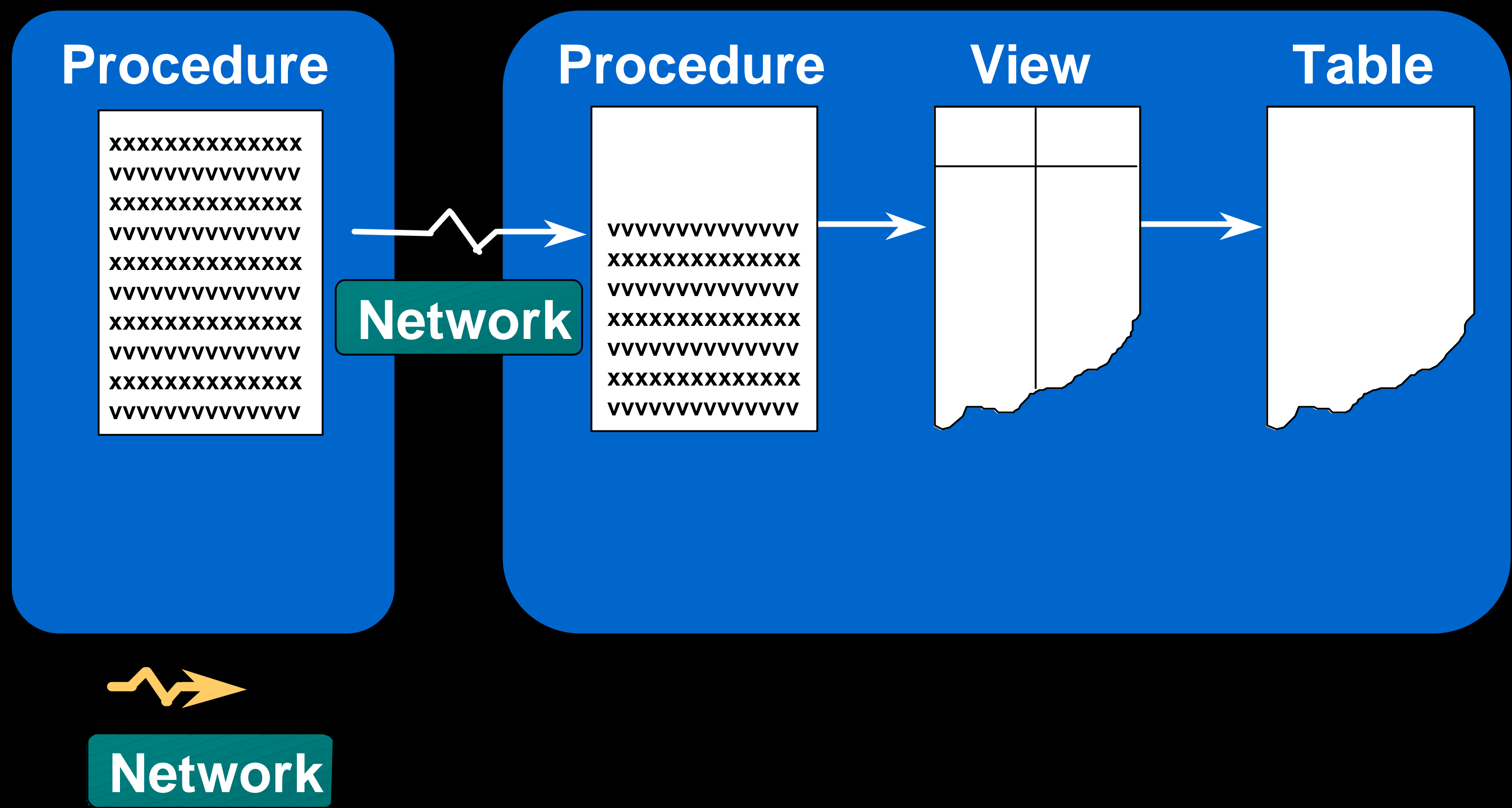
```
ALTER SYSTEM SET
REMOTE_DEPENDENCIES_MODE = value
```

- At the session level

```
ALTER SESSION SET
REMOTE_DEPENDENCIES_MODE = value
```

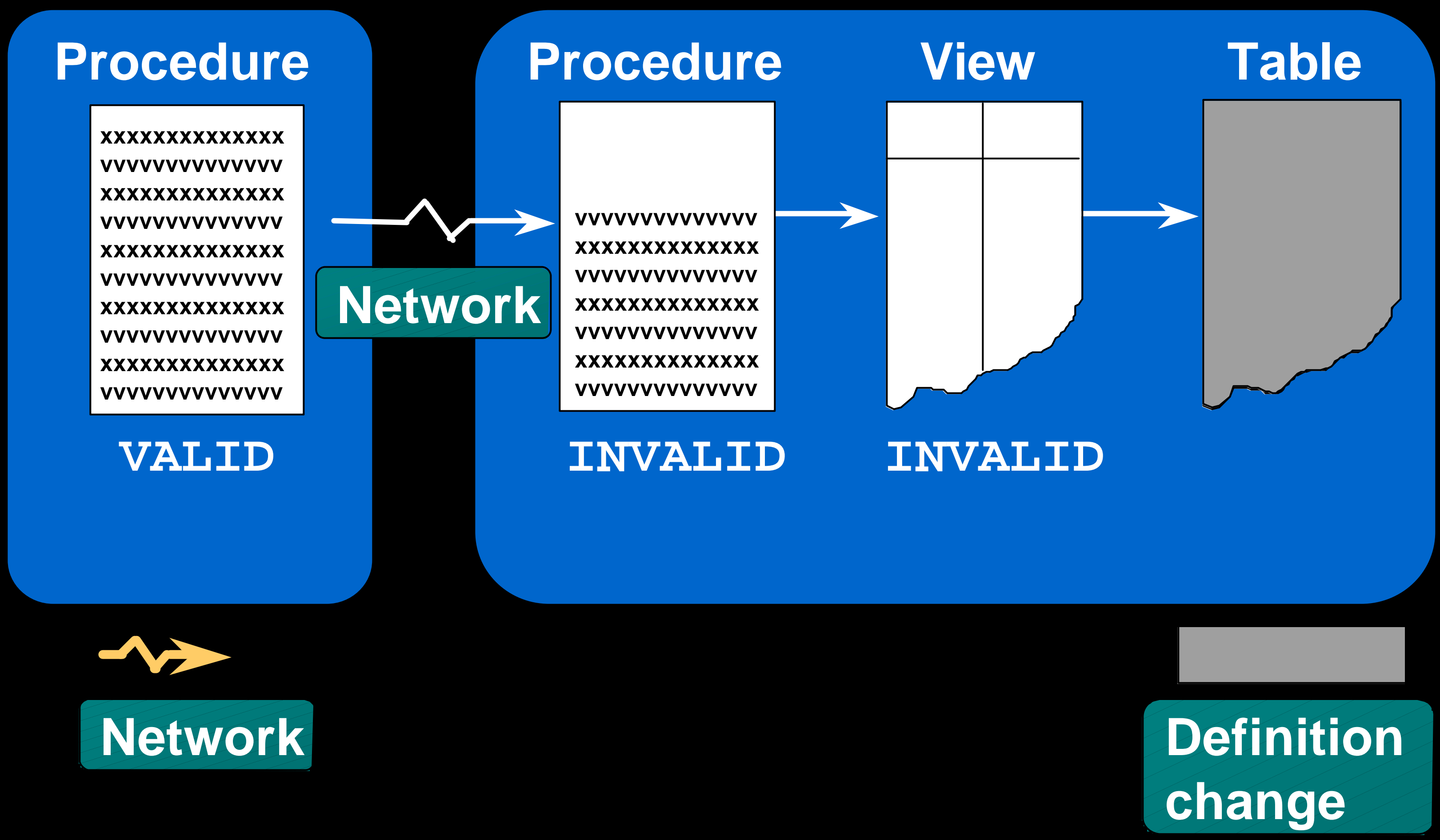
Copyright © Oracle Corporation, 2001. All rights reserved.

Remote Dependencies and Time Stamp Mode



Copyright © Oracle Corporation, 2001. All rights reserved.

Remote Dependencies and Time Stamp Mode



Copyright © Oracle Corporation, 2001. All rights reserved.

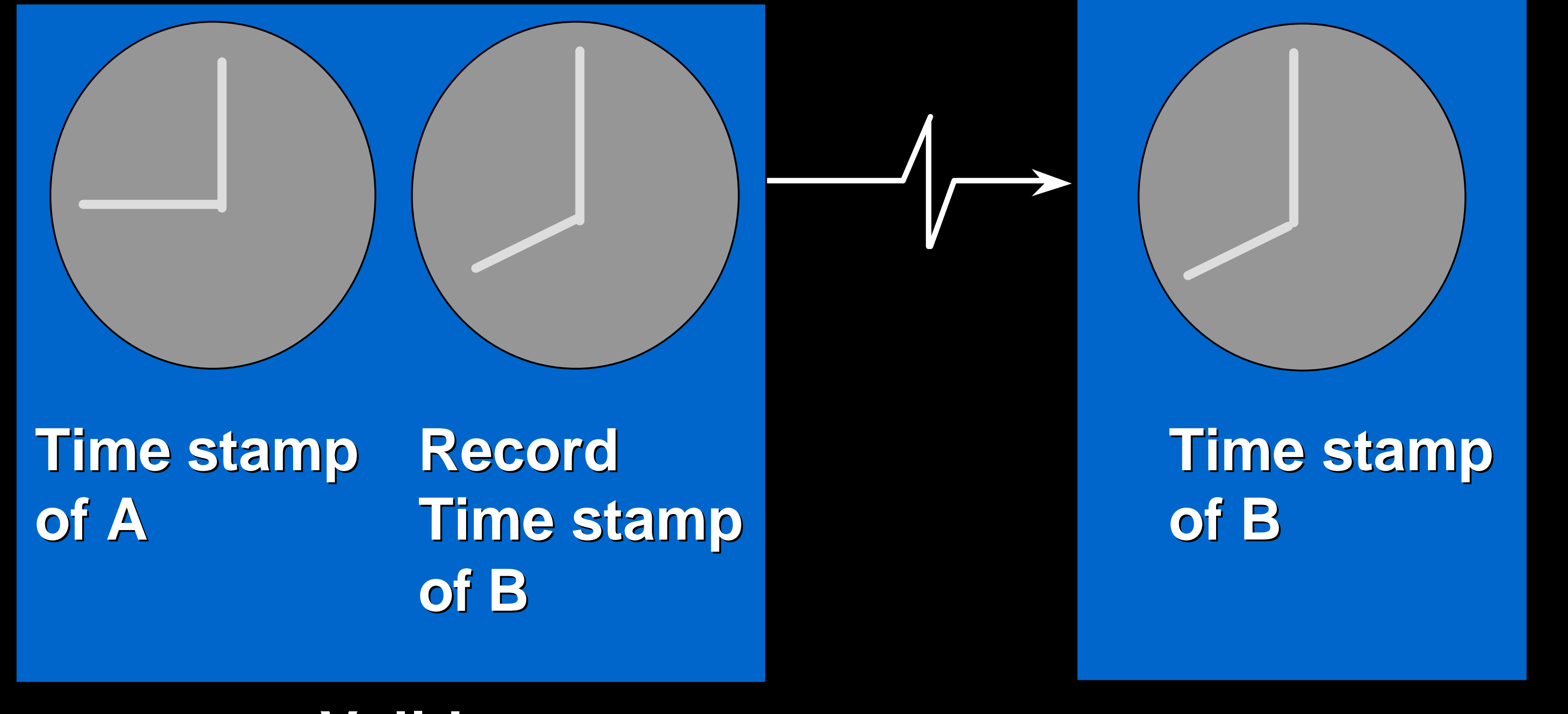
Remote Procedure B Compiles at 8:00 a.m.

Remote procedure B

Valid

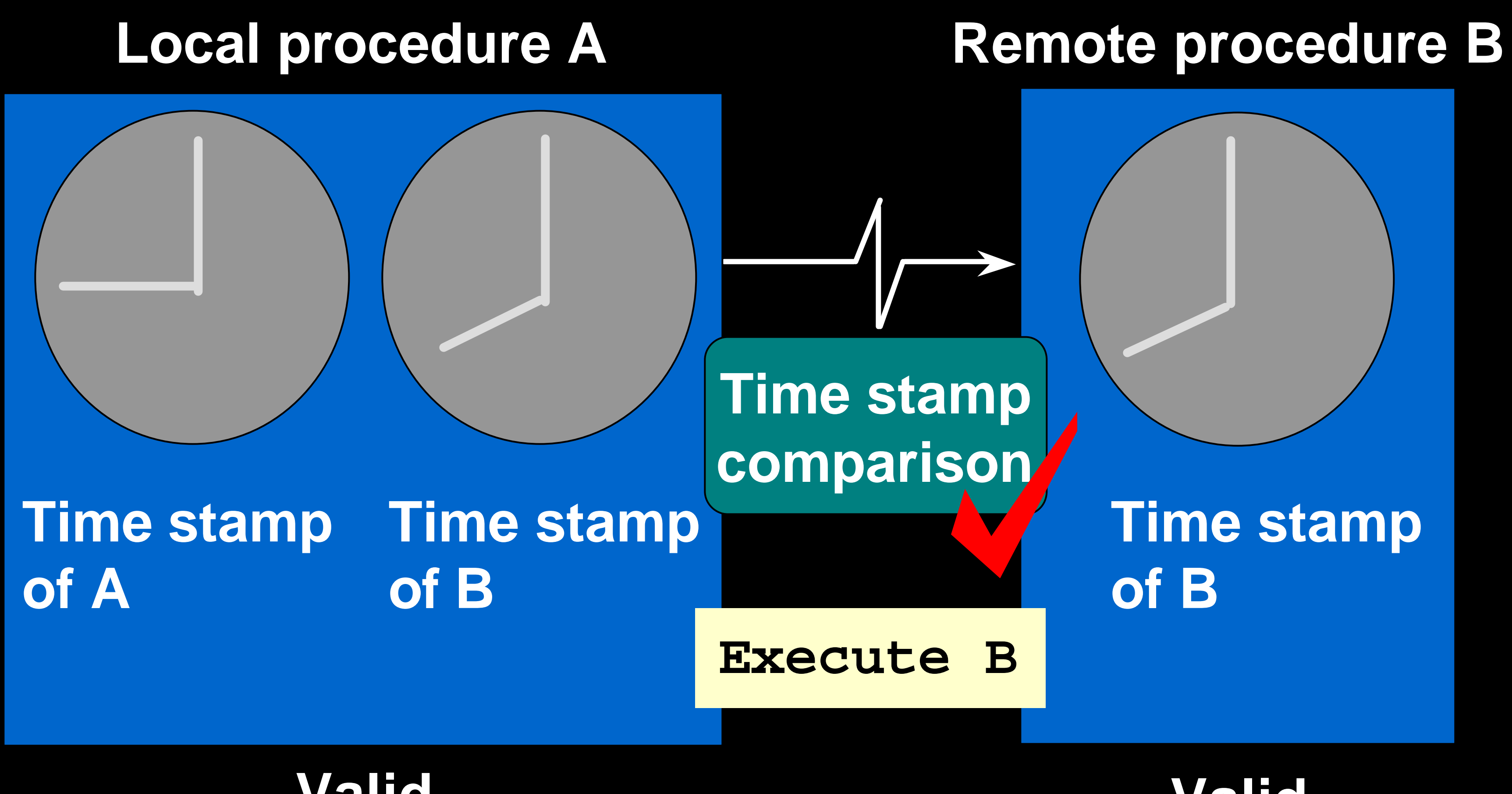
ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Local Procedure A Compiles at 9:00 a.m.



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Execute Procedure A



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

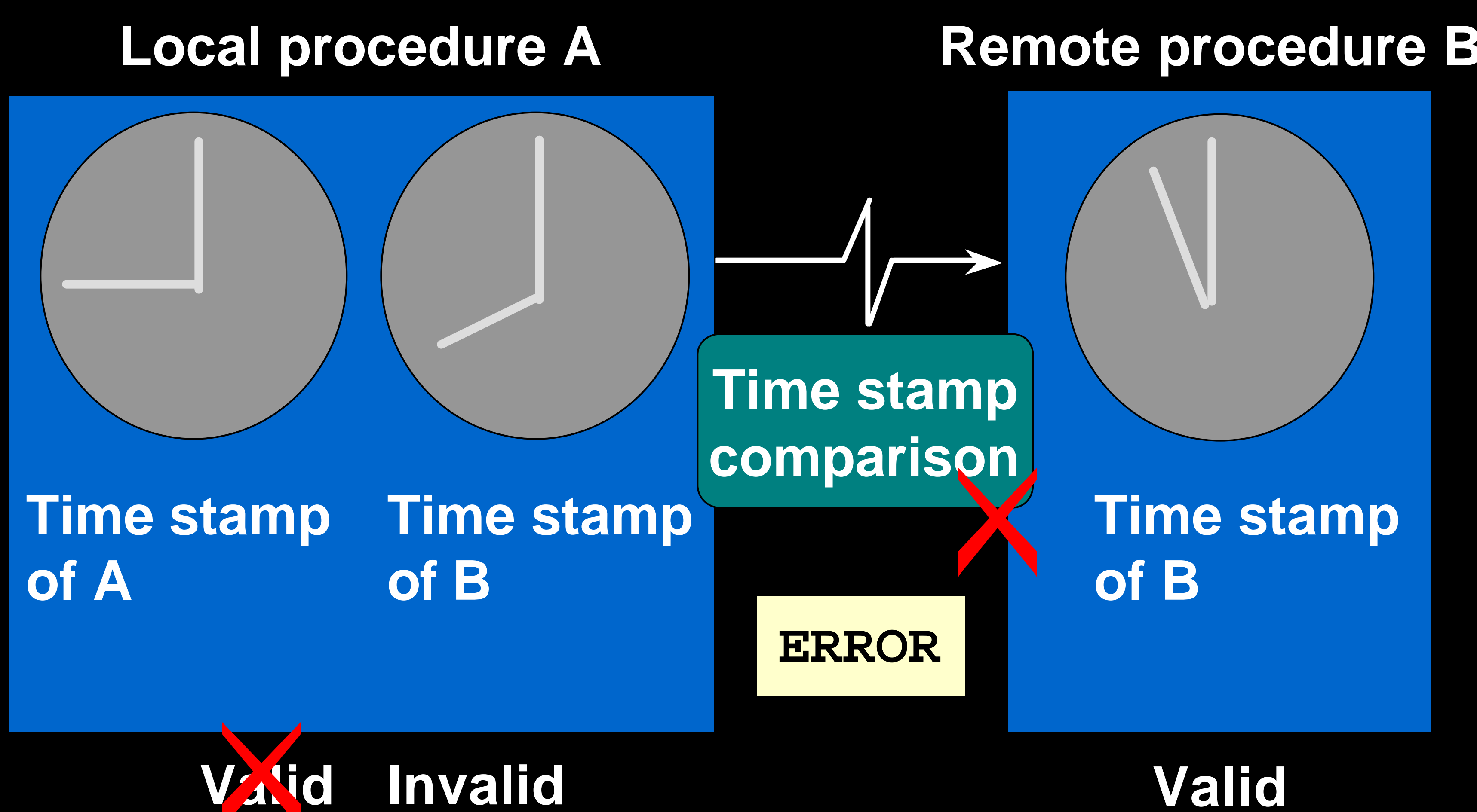
Remote Procedure B Recompiled at 11:00 a.m.

Remote procedure B

Valid

Copyright © Oracle Corporation, 2001. All rights reserved.

Execute Procedure A



Copyright © Oracle Corporation, 2001. All rights reserved.

Signature Mode

- The signature of a procedure is:
  - The name of the procedure
  - The datatypes of the parameters
  - The modes of the parameters
- The signature of the remote procedure is saved in the local procedure.
- When executing a dependent procedure, the signature of the referenced remote procedure is compared.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Recompiling a PL/SQL Program Unit

Recompilation:

- Is handled automatically through implicit run-time recompilation
- Is handled through explicit recompilation with the `ALTER` statement

```
ALTER PROCEDURE [SCHEMA.]procedure_name COMPILE;
```

```
ALTER FUNCTION  [SCHEMA.]function_name  COMPILE;
```

```
ALTER PACKAGE [SCHEMA.]package_name COMPILE [PACKAGE];
ALTER PACKAGE [SCHEMA.]package_name COMPILE BODY;
```

```
ALTER TRIGGER trigger_name [COMPILE[DEBUG]];
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Unsuccessful Recompilation

Recompiling dependent procedures and functions is unsuccessful when:

- The referenced object is dropped or renamed
- The data type of the referenced column is changed
- The referenced column is dropped
- A referenced view is replaced by a view with different columns
- The parameter list of a referenced procedure is modified

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Successful Recompilation

Recompiling dependent procedures and functions is successful if:

- The referenced table has new columns
- The data type of referenced columns has not changed
- A private table is dropped, but a public table, having the same name and structure, exists
- The PL/SQL body of a referenced procedure has been modified and recompiled successfully

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Recompilation of Procedures

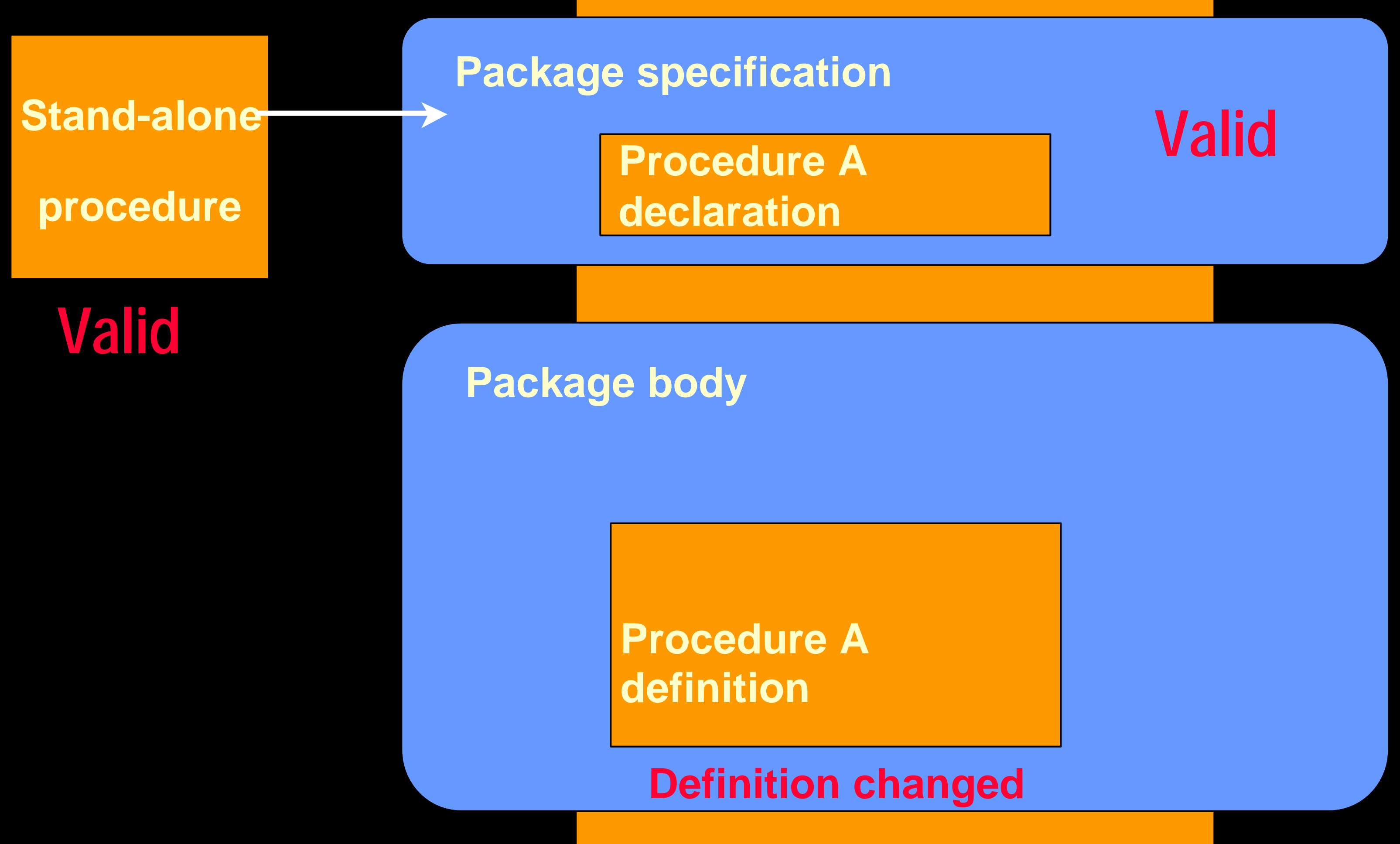
Minimize dependency failures by:

- Declaring records by using the `%ROWTYPE` attribute
- Declaring variables with the `%TYPE` attribute
- Querying with the `SELECT *` notation
- Including a column list with `INSERT` statements

ORACLE

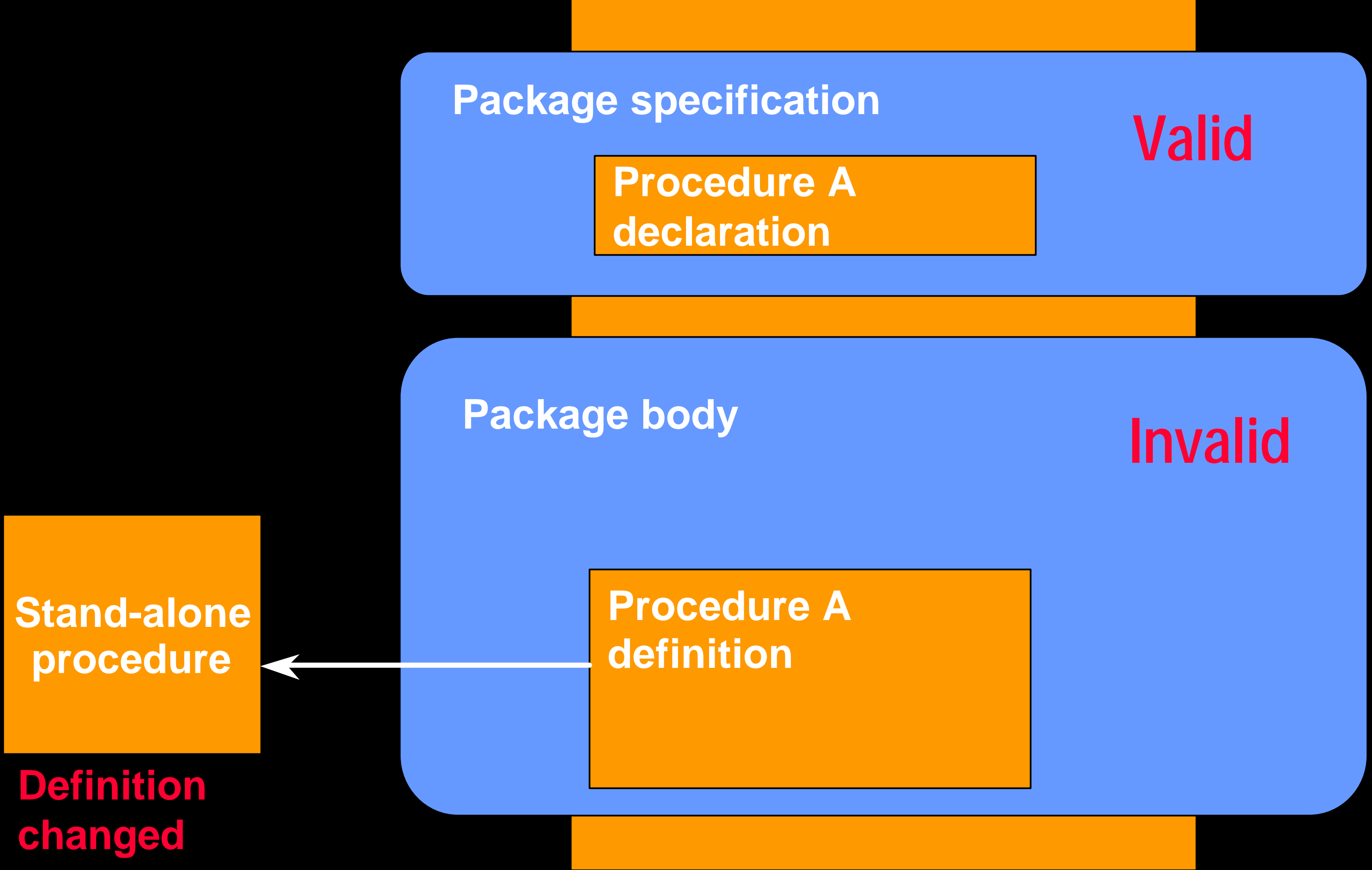
Copyright © Oracle Corporation, 2001. All rights reserved.

Packages and Dependencies



ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Packages and Dependencies



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Keep track of dependent procedures
- Recompile procedures manually as soon as possible after the definition of a database object changes

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 18 Overview

This practice covers the following topics:

- Using `DEPTREE_FILL` and `IDEPTREE` to view dependencies
- Recompiling procedures, functions, and packages

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Program Units by Using Procedure Builder

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

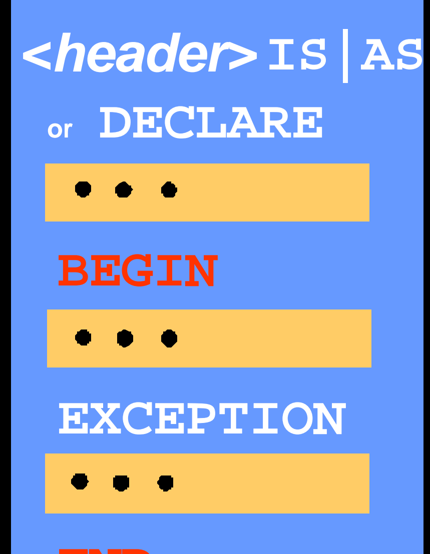
Objectives

After completing this appendix, you should be able to do the following:

- Describe the features of Oracle Procedure Builder
- Manage program units using the Object Navigator
- Create and compile program units using the Program Unit Editor
- Invoke program units using the PL/SQL Interpreter
- Debug subprograms using the debugger
- Control execution of an interrupted PL/SQL program unit
- Test possible solutions at run time

Copyright © Oracle Corporation, 2001. All rights reserved.

PL/SQL Program Constructs



Tools Constructs
Anonymous blocks
Application procedures or functions
Application packages
Application triggers
Object types

Database Server Constructs
Anonymous blocks
Stored procedures or functions
Stored packages
Database triggers
Object types

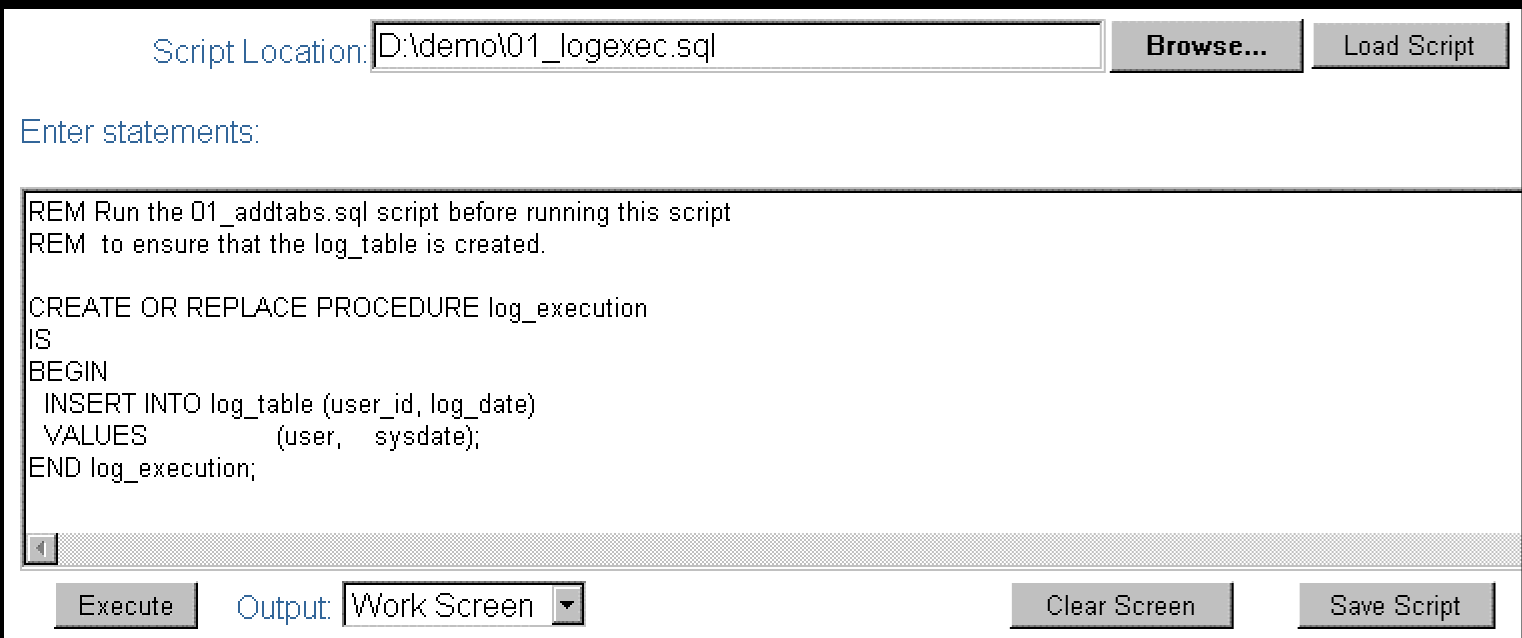
Copyright © Oracle Corporation, 2001. All rights reserved.

Development Environments

- *i*SQL*Plus uses the PL/SQL engine in the Oracle Server
- Oracle Procedure Builder uses the PL/SQL engine in the client tool or in the Oracle Server. It includes:
  - A GUI development environment for PL/SQL code
  - Built-in editors
  - The ability to compile, test, and debug code
  - Application partitioning that allows drag-and-drop of program units between client and server

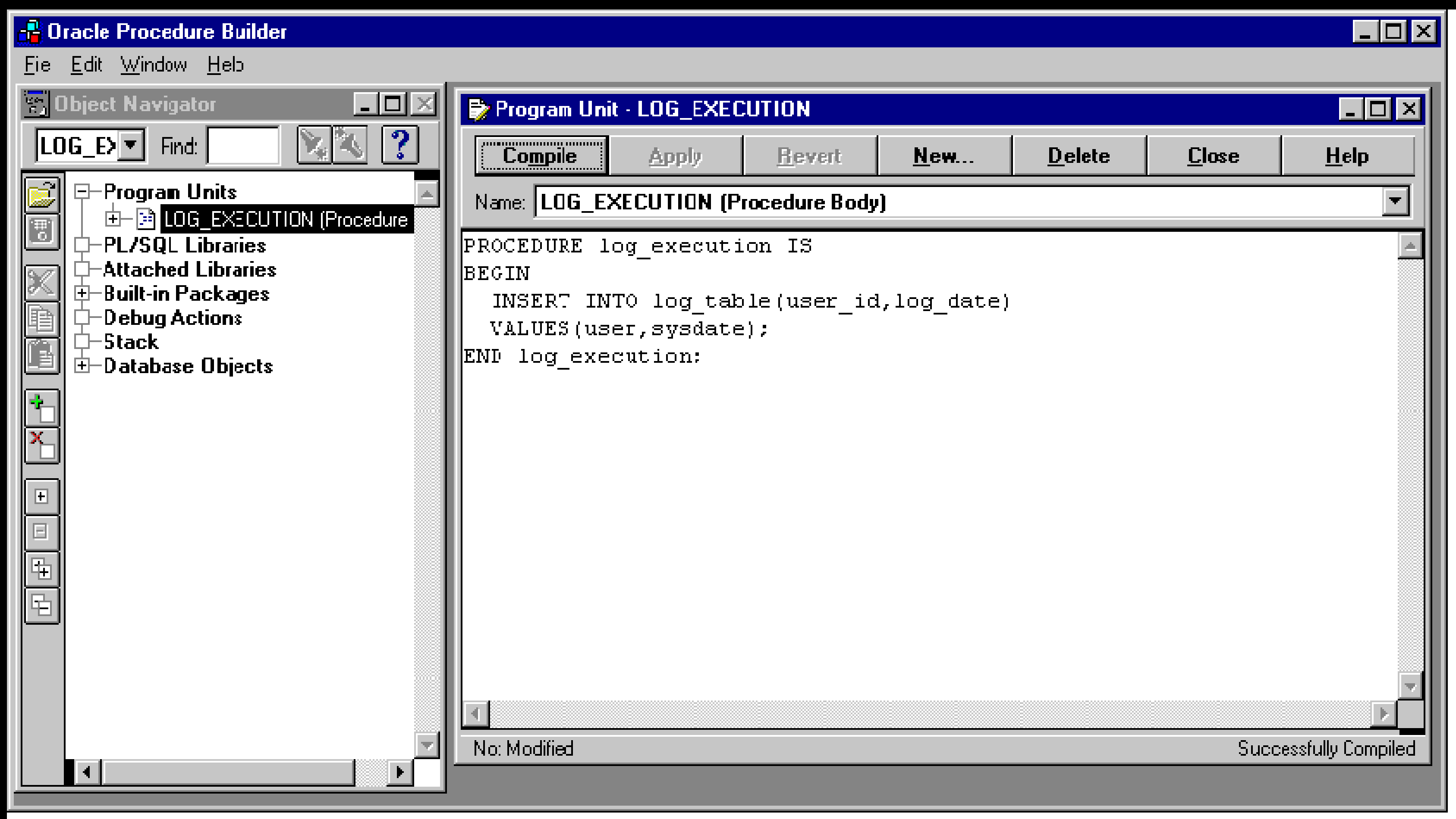
Copyright © Oracle Corporation, 2001. All rights reserved.

Developing Procedures and Functions Using *i*SQL*Plus



Copyright © Oracle Corporation, 2001. All rights reserved.

Developing Procedures and Functions Using Oracle Procedure Builder



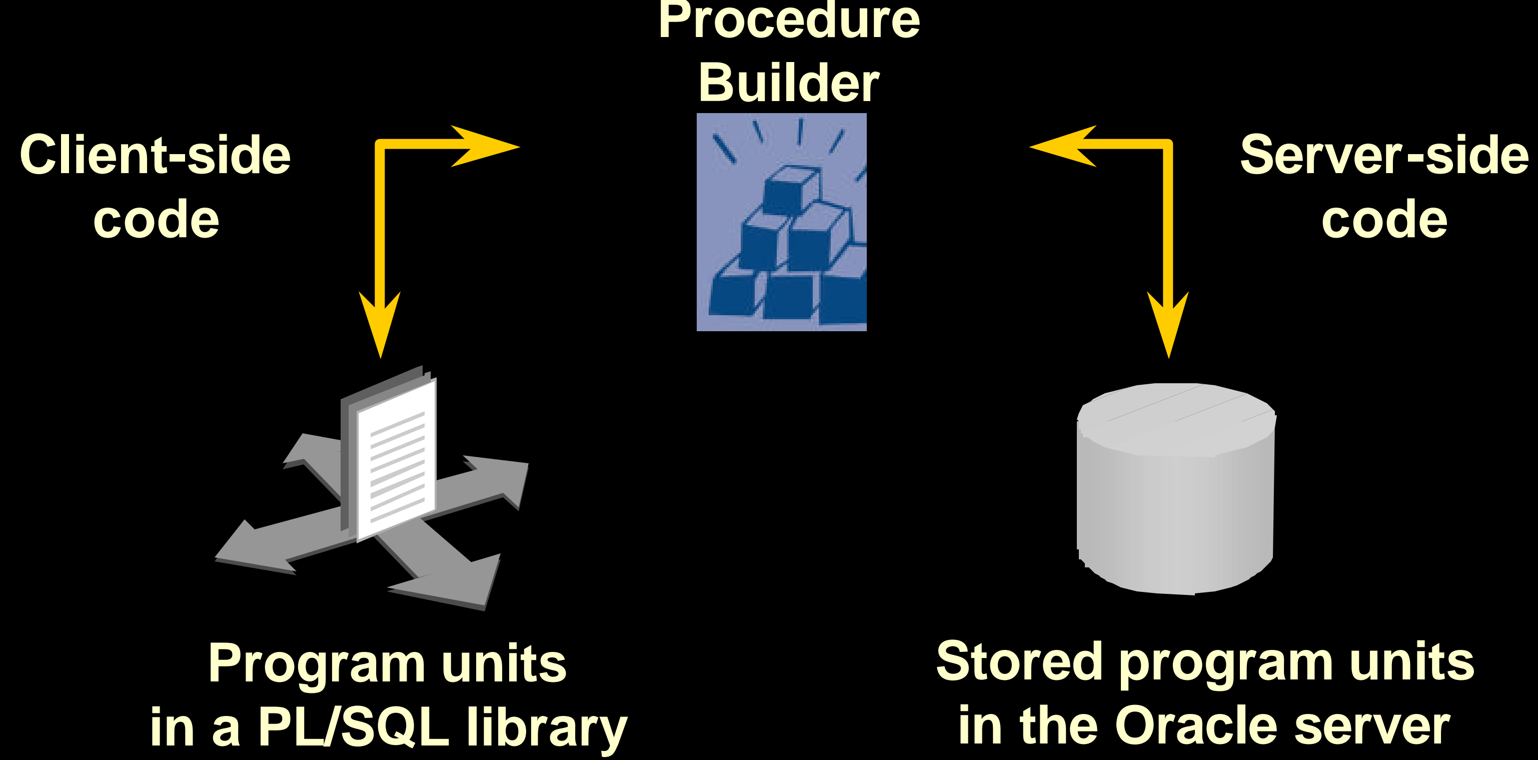
Copyright © Oracle Corporation, 2001. All rights reserved.

Components of Procedure Builder

Component	Function
Object Navigator	Manages PL/SQL constructs; performs debug actions
PL/SQL Interpreter	Debugs PL/SQL code; evaluates PL/SQL code in real time
Program Unit Editor	Creates and edits PL/SQL source code
Stored Program Unit Editor	Creates and edits server-side PL/SQL source code
Database Trigger Editor	Creates and edits database triggers

Copyright © Oracle Corporation, 2001. All rights reserved.

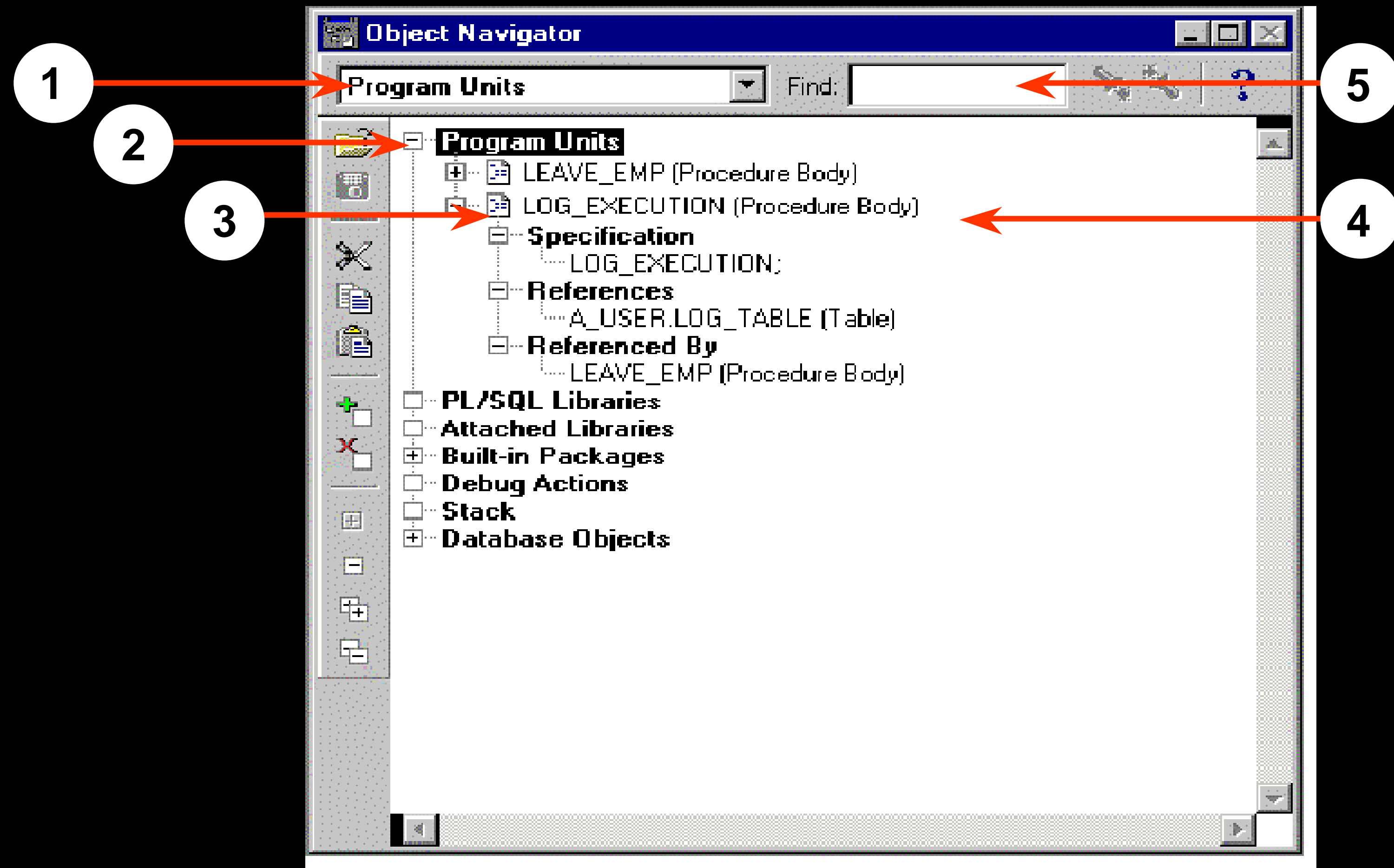
Developing Program Units and Stored Programs Units



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

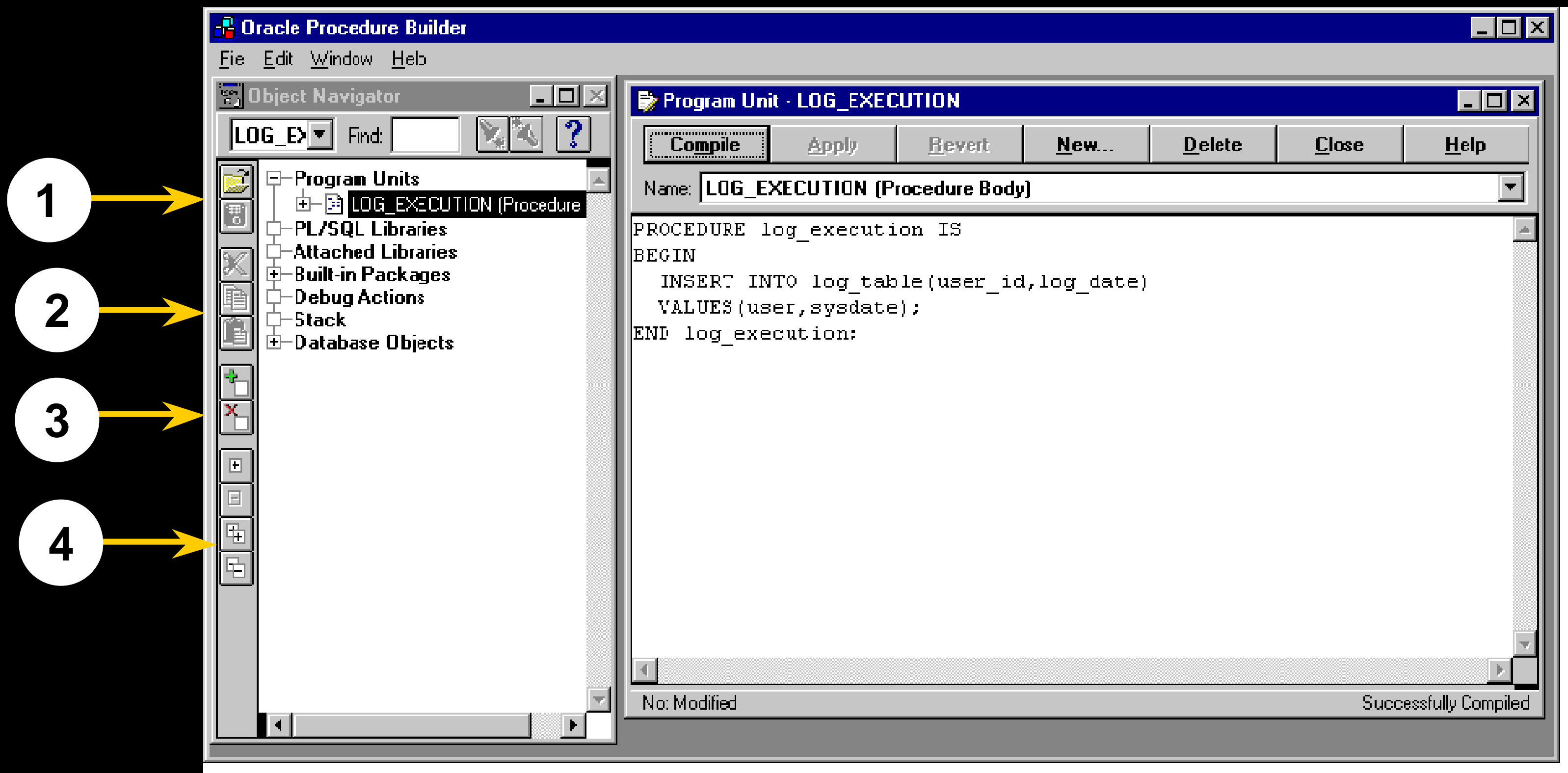
Procedure Builder Components: The Object Navigator



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Procedure Builder Components: The Object Navigator



Copyright © Oracle Corporation, 2001. All rights reserved.

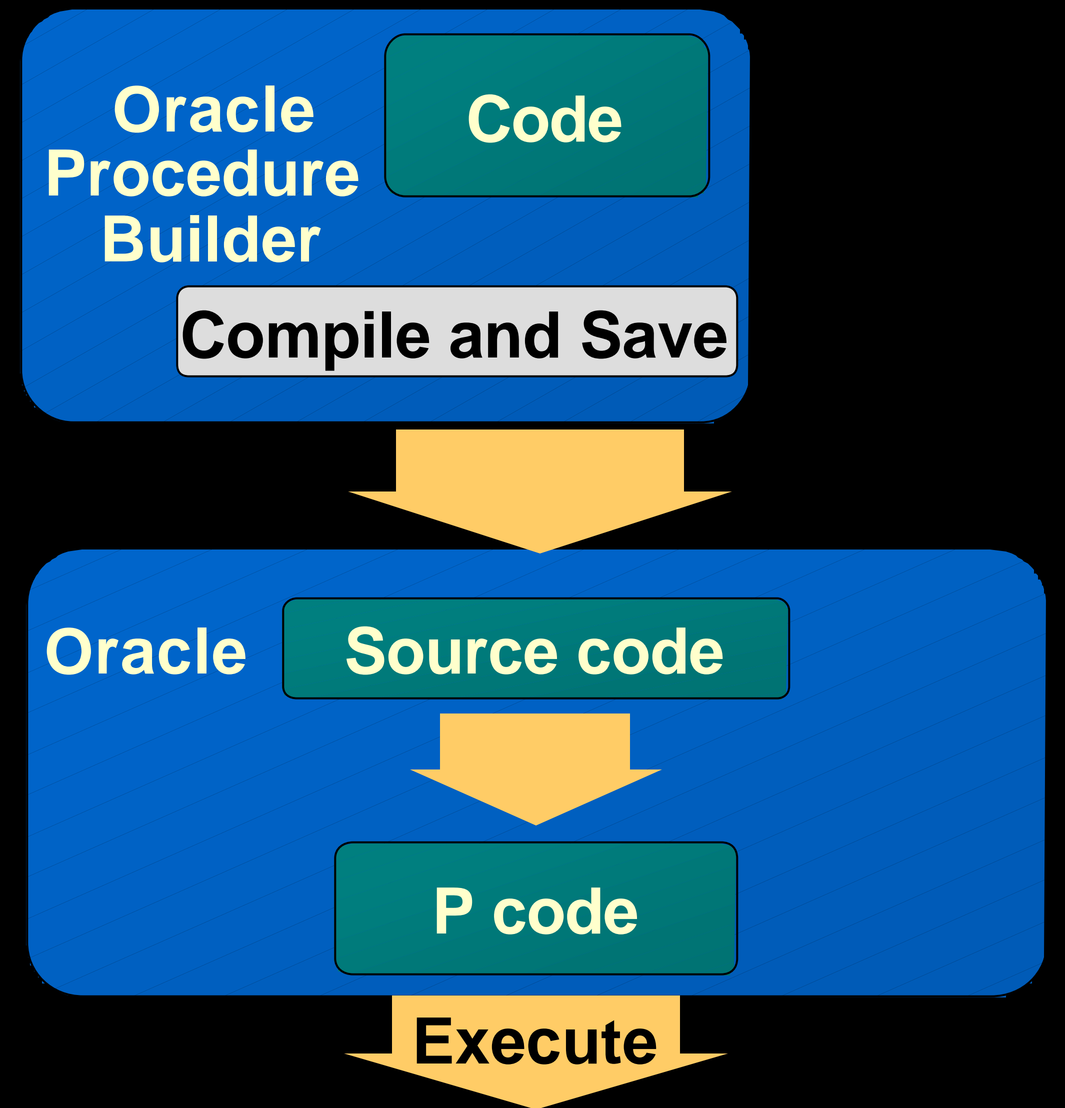
ORACLE

Procedure Builder Components: Objects of the Navigator

- Program Units
  - Specification
  - References
  - Referenced By
- Libraries
- Attached Libraries
- Built-in Packages
- Debug Actions
- Stack
- Database Objects

Copyright © Oracle Corporation, 2001. All rights reserved.

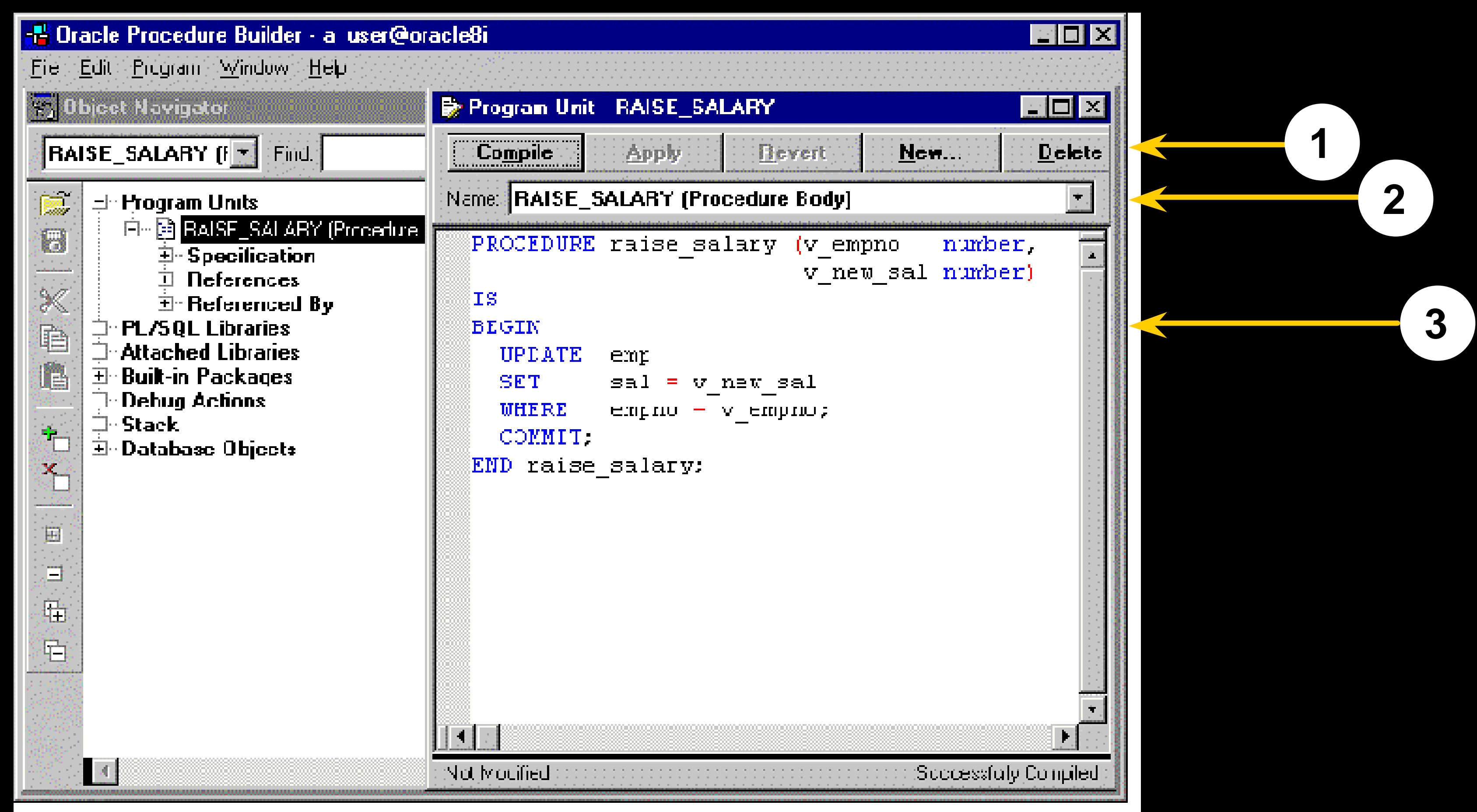
Developing Stored Procedures



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

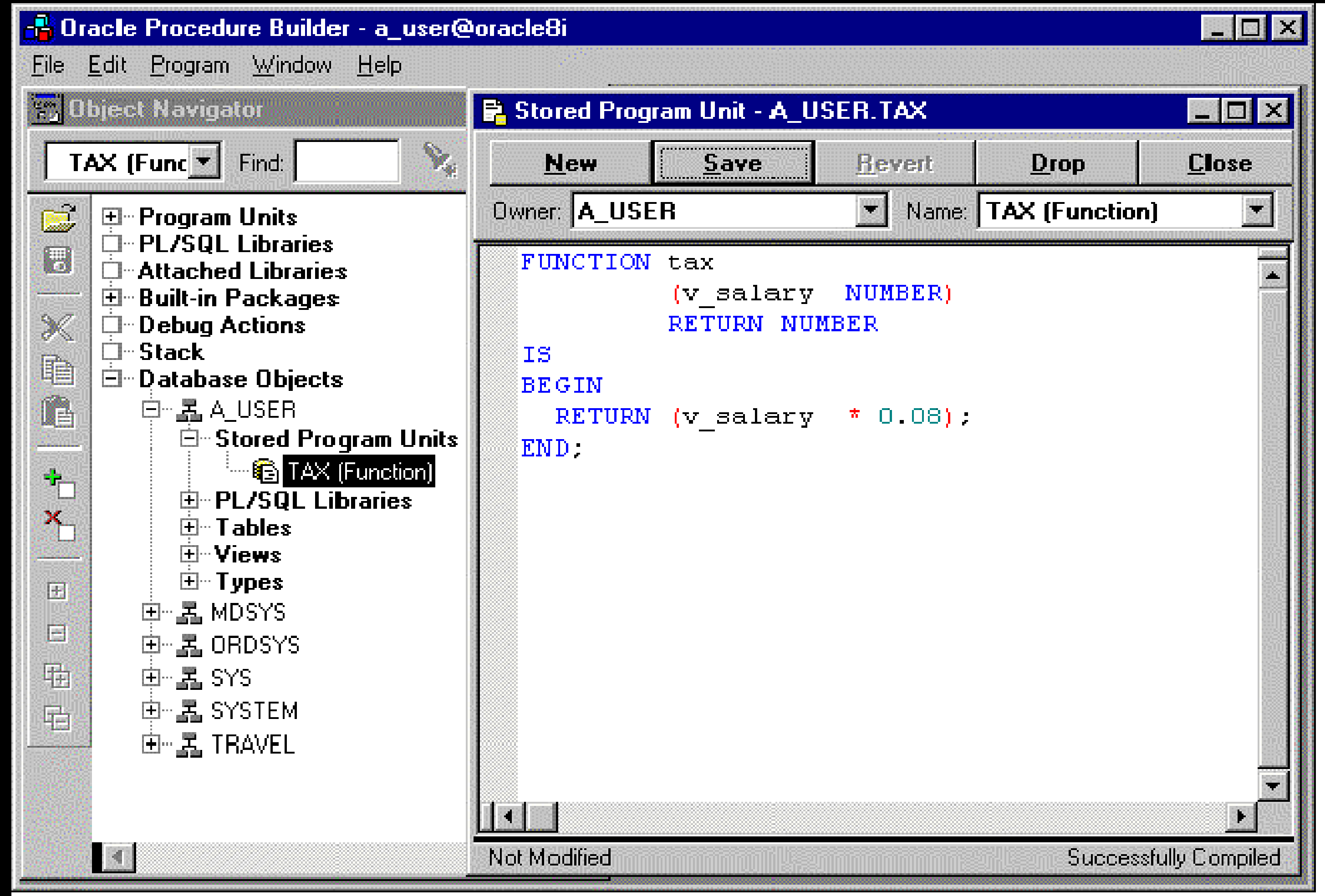
Procedure Builder Components: The Program Unit Editor



ORACLE

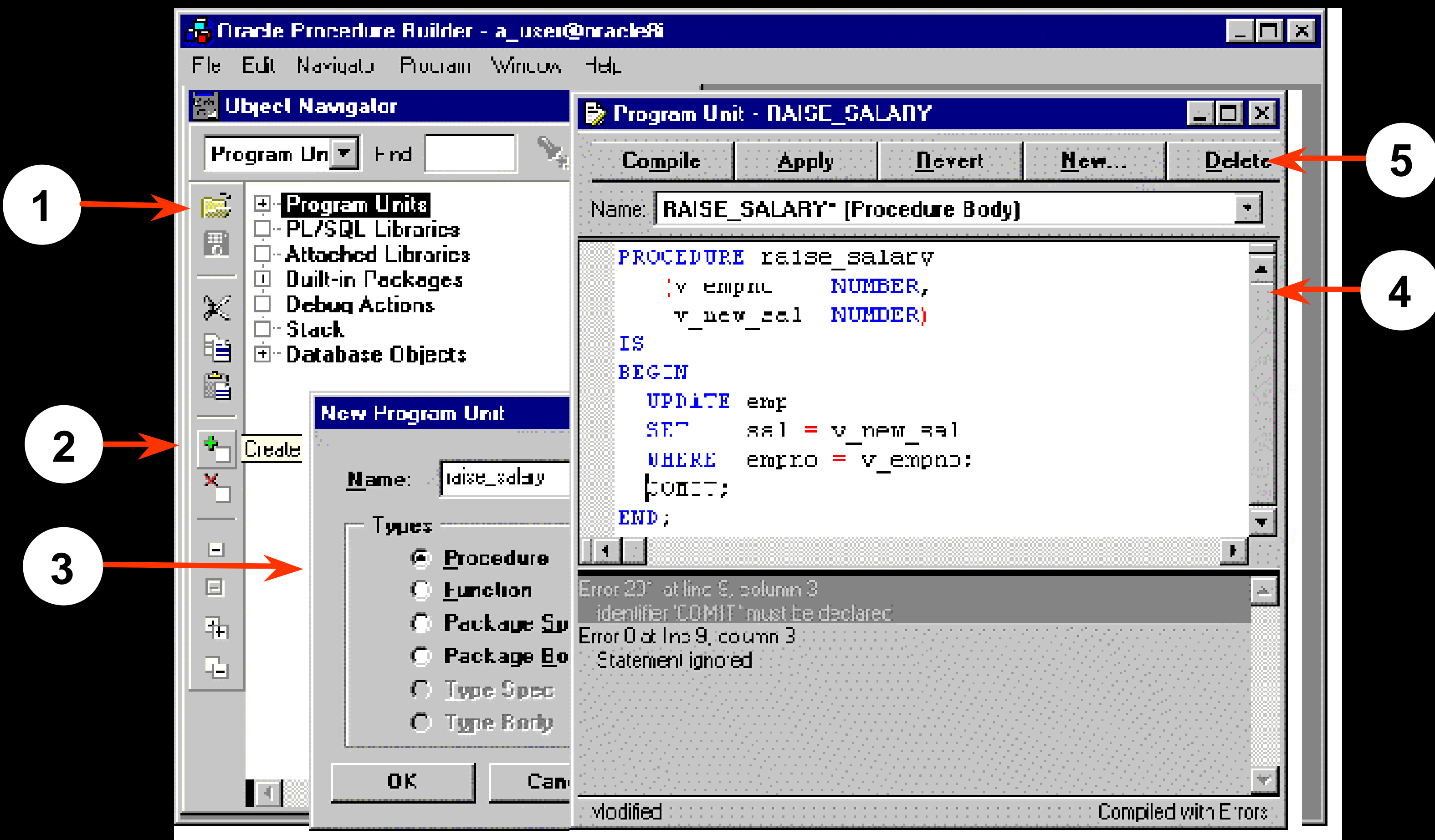
Copyright © Oracle Corporation, 2001. All rights reserved.

Procedure Builder Components: The Stored Program Unit Editor



Copyright © Oracle Corporation, 2001. All rights reserved.

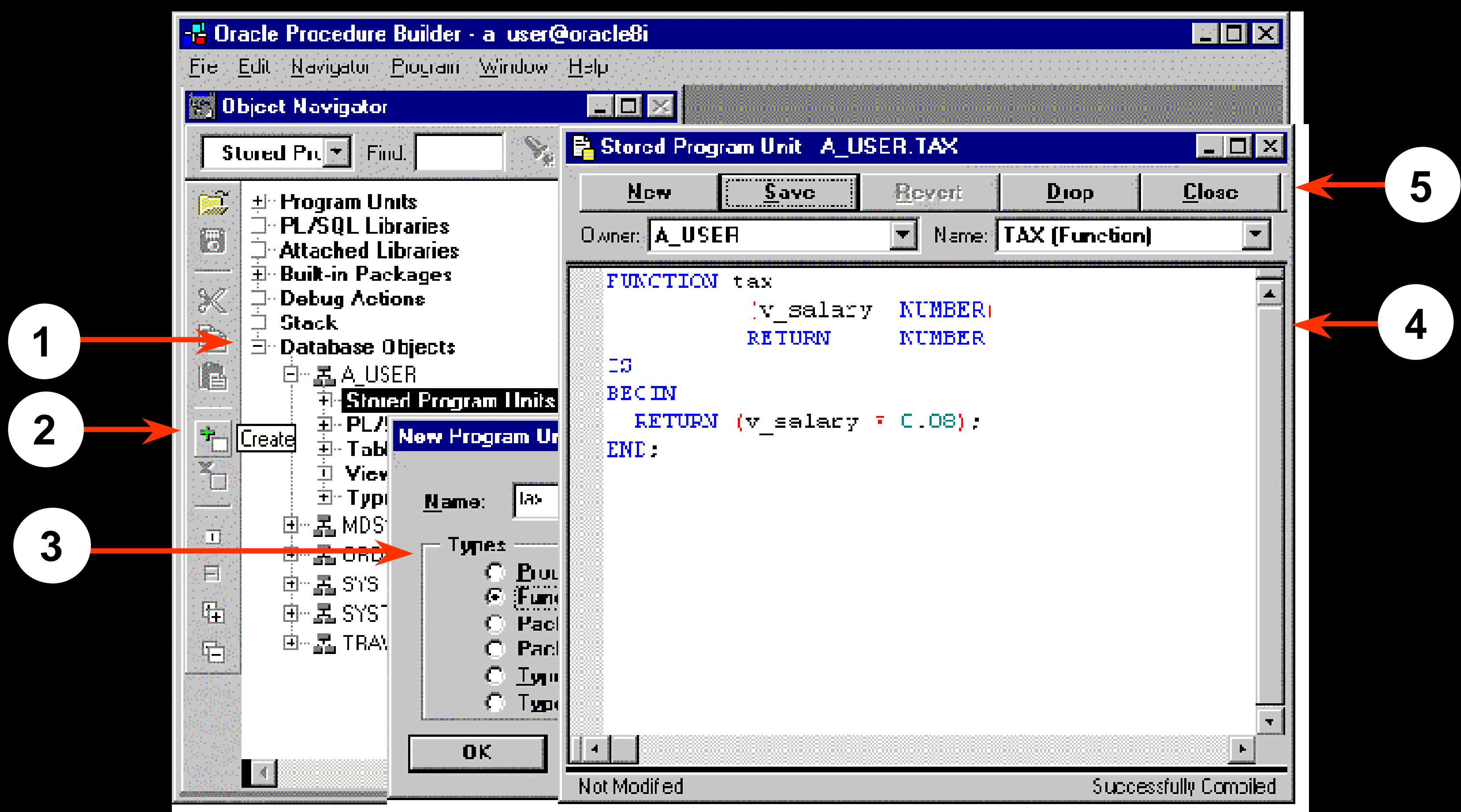
Creating a Client-Side Program Unit



Copyright © Oracle Corporation, 2001. All rights reserved.

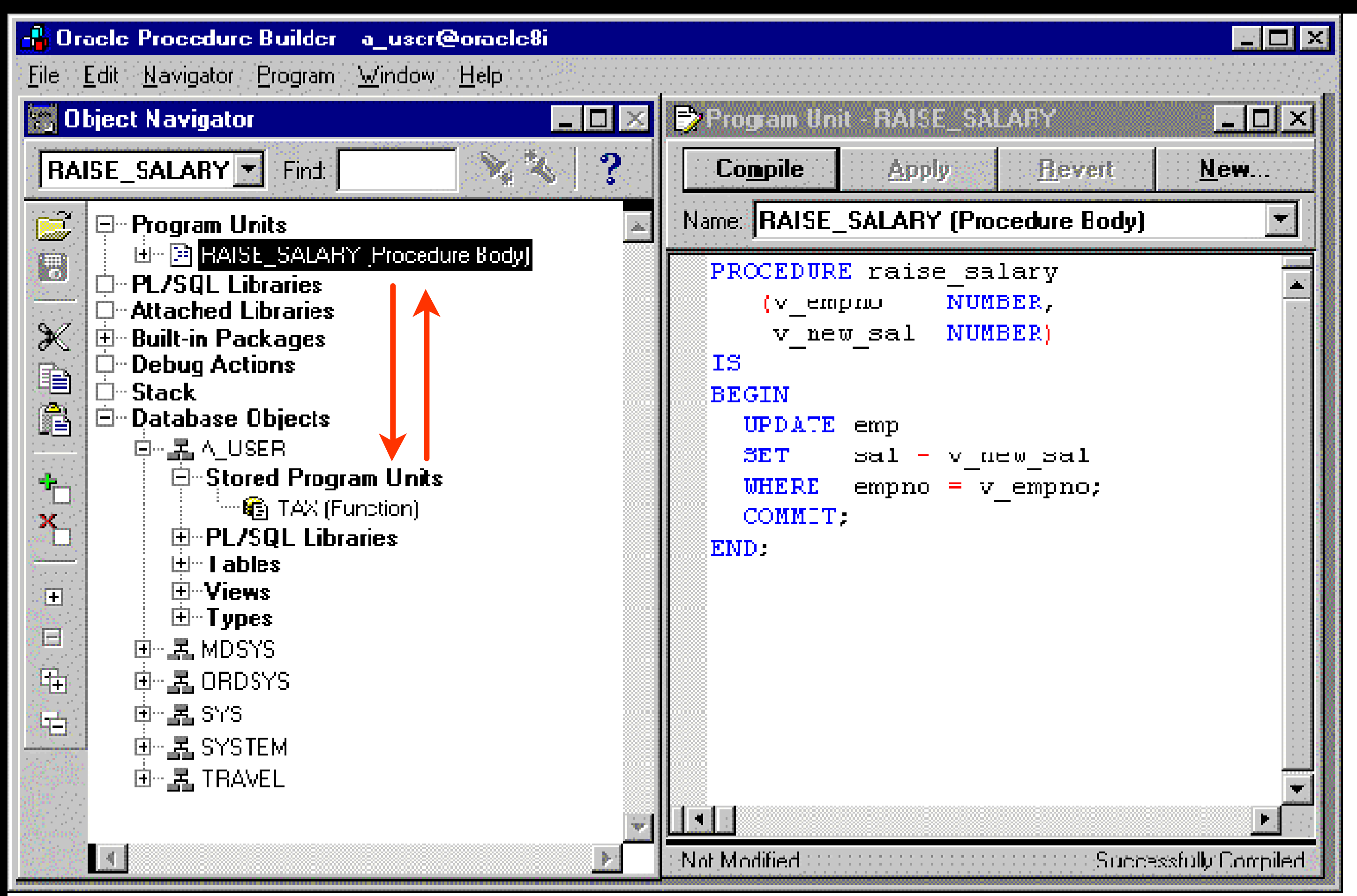
ORACLE

Creating a Server-Side Program Unit



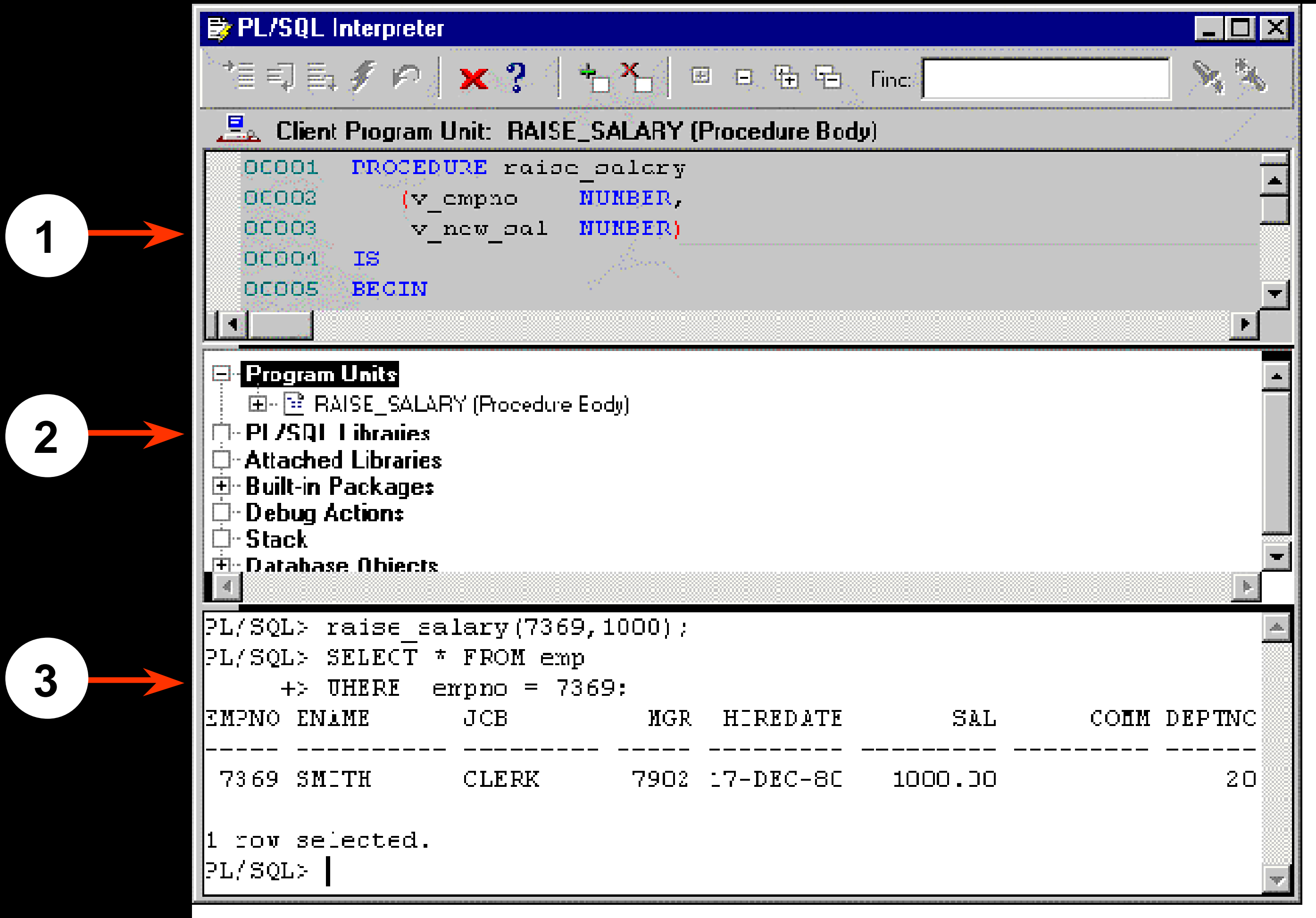
Copyright © Oracle Corporation, 2001. All rights reserved.
ORACLE

Transferring Program Units Between Client and Server



Copyright © Oracle Corporation, 2001. All rights reserved.

Procedure Builder Components: The PL/SQL Interpreter



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Client-Side Program Units

Program Unit - MY_MESSAGE

Compile | Apply | Revert | New...

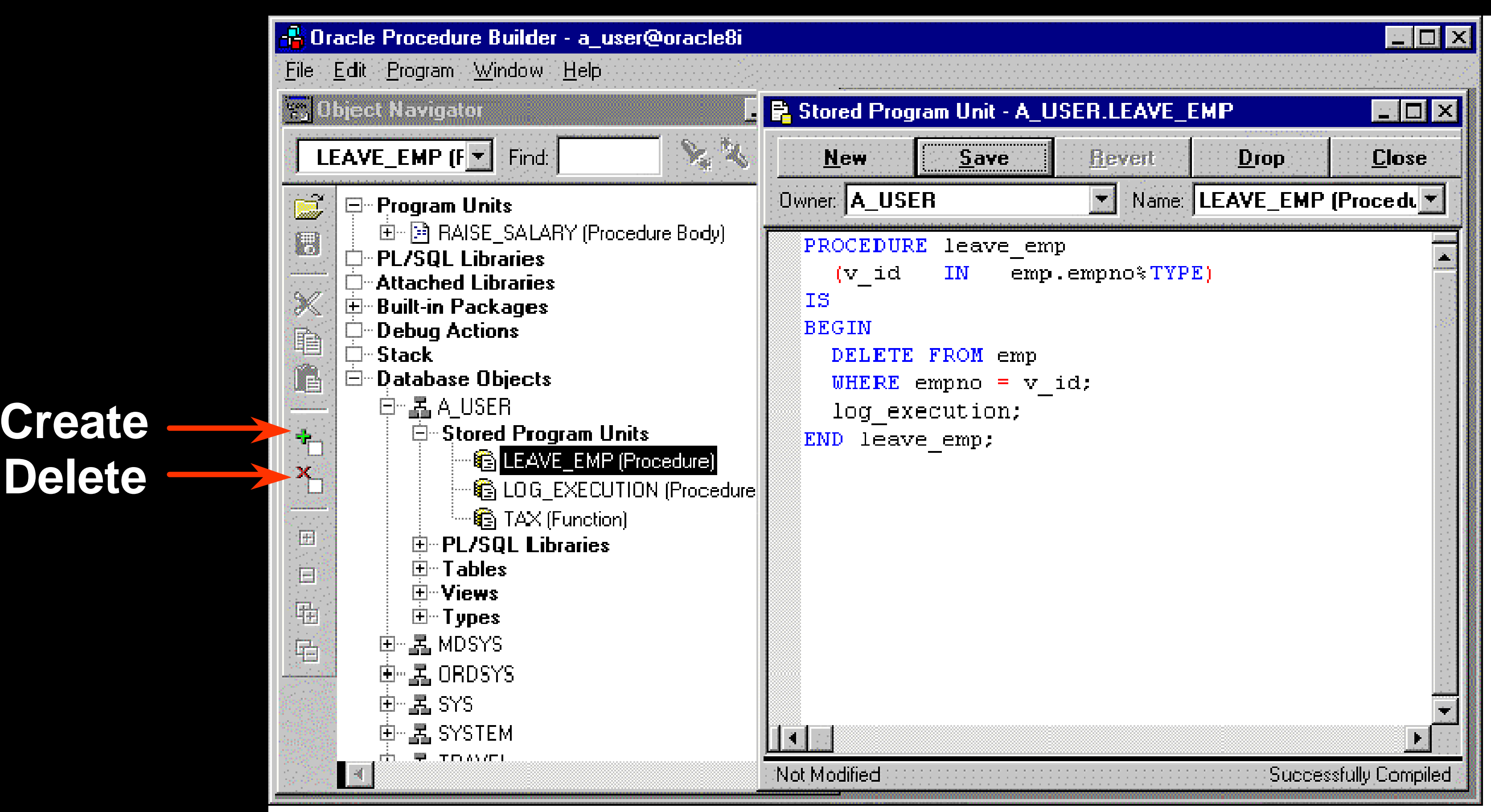
Name: MY_MESSAGE (Procedure Body)

```
PROCEDURE my_message
   (v_message   VARCHAR2)
IS
BEGIN
  TEXT_IO.PUT_LINE (v_message);
END;
```

Not Modified   Successfully Compiled

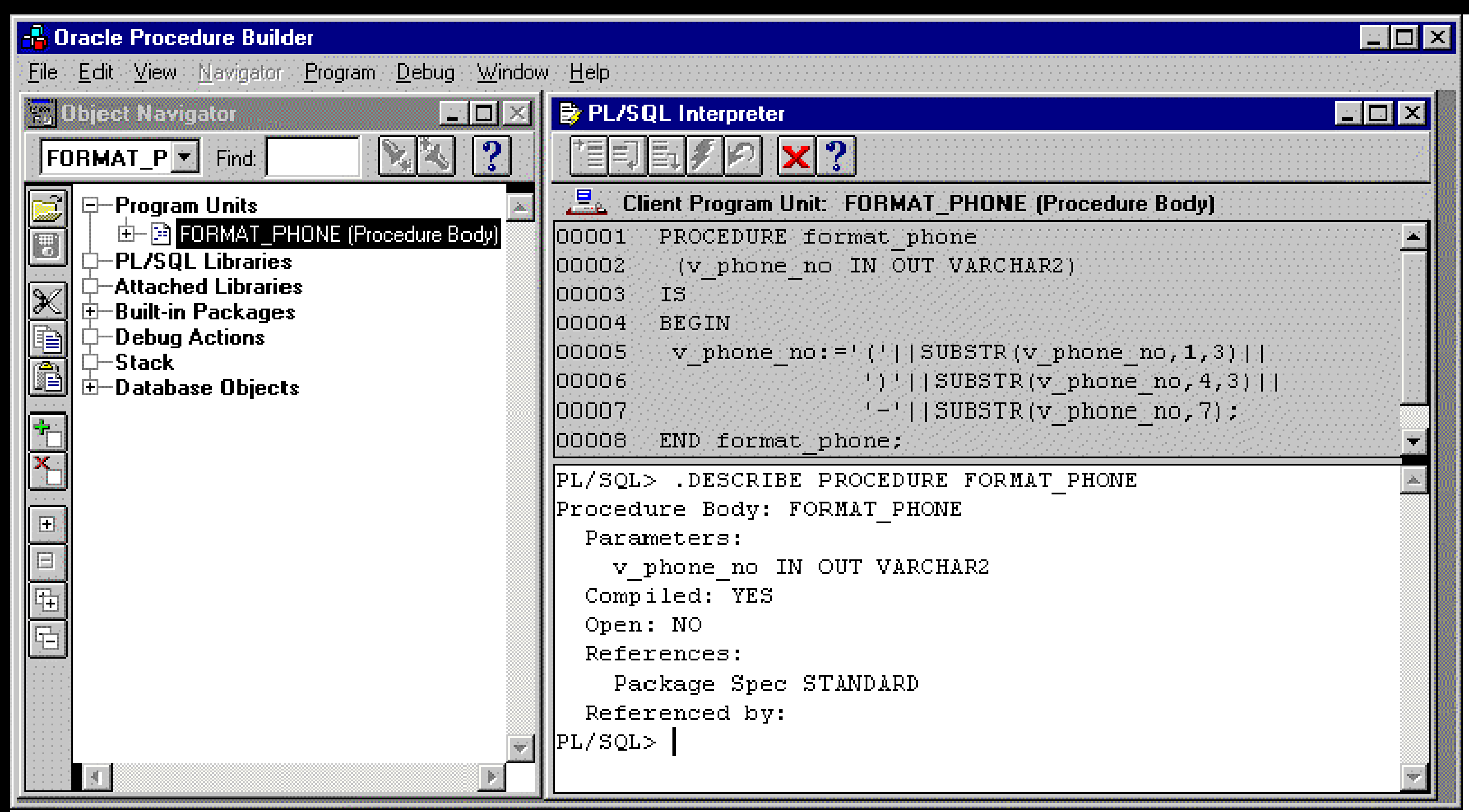
ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Server-Side Program Units



Copyright © Oracle Corporation, 2001. All rights reserved.

The DESCRIBE Command in Procedure Builder



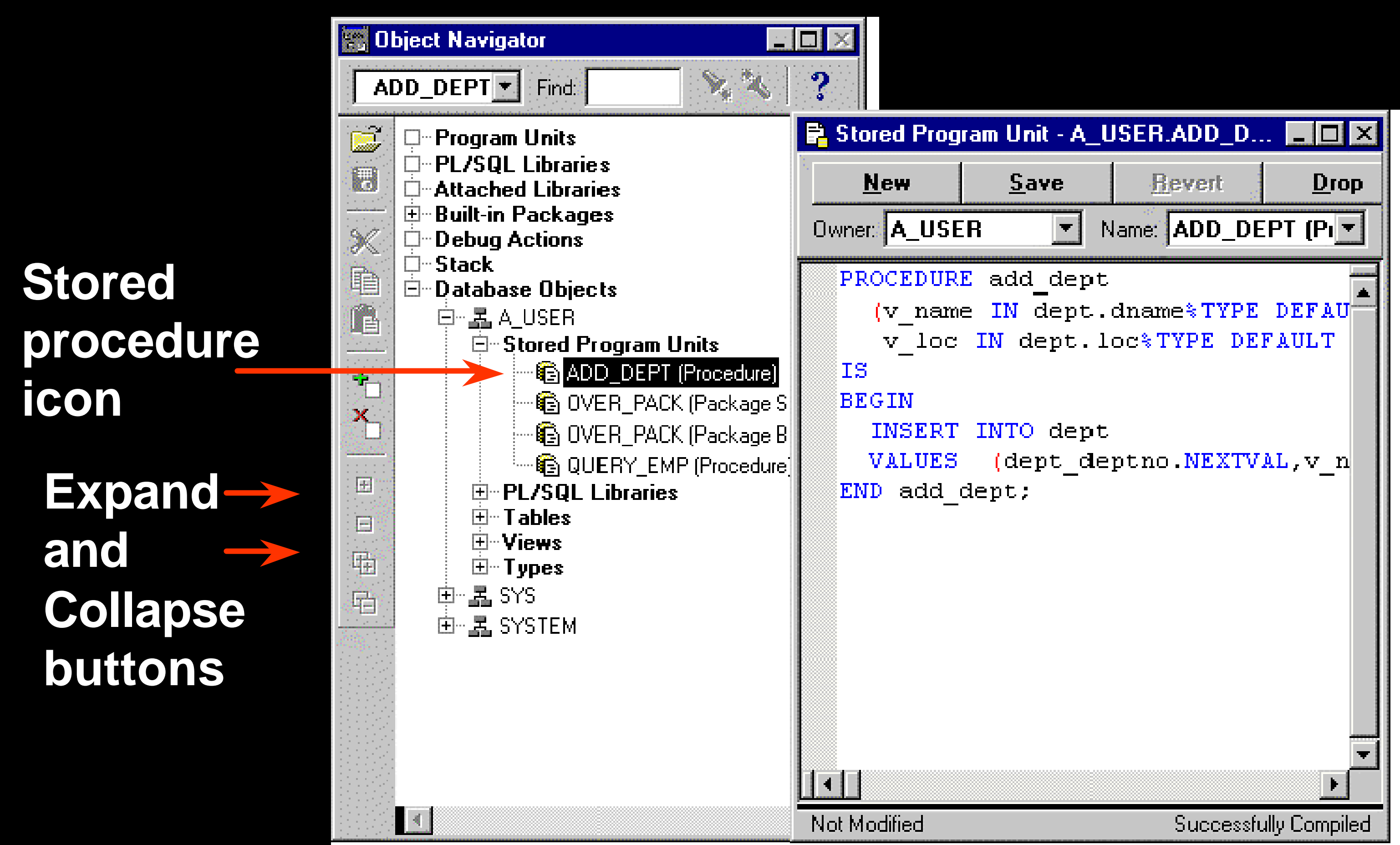
ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Listing Code of Stored Program Units

Stored procedure icon

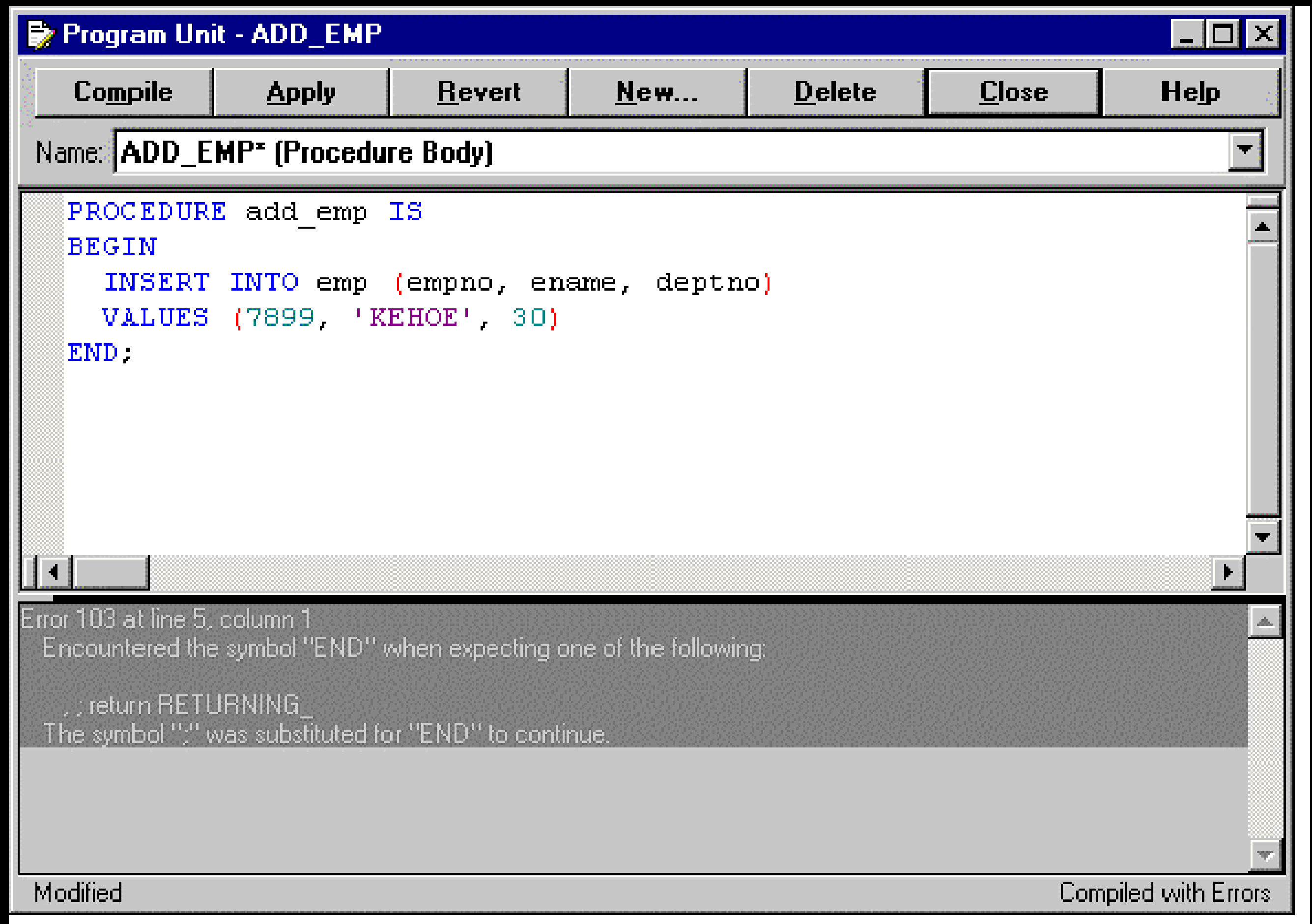
Expand and Collapse buttons



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Navigating Compilation Errors in Procedure Builder



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

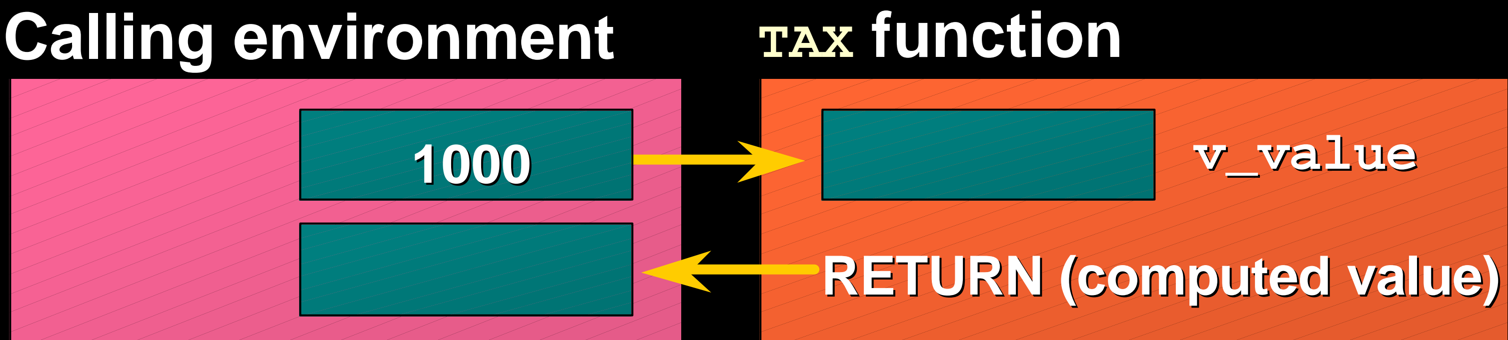
Procedure Builder Built-in Package: TEXT_IO

- The TEXT_IO package:
  - Contains a procedure PUT_LINE, which writes information to the PL/SQL Interpreter window
  - Is used for client-side program units
- The TEXT_IO.PUT_LINE accepts one parameter

```
PL/SQL> TEXT_IO.PUT_LINE(1);
1
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Executing Functions in Procedure Builder: Example



Display the tax based on a specified value.

```
PL/SQL> .CREATE NUMBER x PRECISION 4
PL/SQL> :x := tax(1000);
PL/SQL> TEXT_IO.PUT_LINE (TO_CHAR(:x));
80
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Statement Triggers

Database Trigger

Table Owner: A_USER

Table: EMP

Name: SECURE_EMP

Triggering
- (•) Before
- ( ) After
- ( ) Instead Of

Statement
- [ ] UPDATE
- [x] INSERT
- [ ] DELETE

Of Columns

For Each
- (•) Statement
- ( ) Row

Referencing OLD As: NEW As:

When:

Trigger Body:

```
BEGIN
    IF      TO_CHAR(SYSDATE, 'DY') IN ('SAT','SUN')
     OR  TO_CHAR(SYSDATE, 'HH24') NOT BETWEEN '08' AND '18'
  THEN
    RAISE_APPLICATION_ERROR (-20500,
      'You may only insert into the EMP table during business hours.');
  END IF;
END;
```

New | Save | Revert | Drop | Close | Help

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Row Triggers

Database Trigger

Table Owner: A_USER

Table: EMP

Name: DERIVE_COMMISSION_PCT

Triggering
- (•) Before
- ( ) After
- ( ) Instead Of

Statement
- [x] UPDATE
- [x] INSERT
- [ ] DELETE

Of Columns: EMPNO, ENAME, JOB, MGR, HIREDATE, SAL

For Each
- ( ) Statement
- (•) Row

Referencing OLD As: OLD NEW As: NEW

When:

Trigger Body:

```
BEGIN
  IF  NOT (:NEW.JOB IN ('MANAGER' , 'PRESIDENT'))
      AND :NEW.SAL > 5000
  THEN
     RAISE_APPLICATION_ERROR
       (-20202, 'EMPLOYEE CANNOT EARN THIS AMOUNT');
  END IF;
END;
```

New | Save | Revert | Drop | Close | Help

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Server-Side Program Units

Using Procedure Builder:

1. Connect to the database.
2. Expand the Database Objects node.
3. Expand the schema of the owner of the program unit.
4. Expand the Stored Program Units node.
5. Click the program unit that you want to drop.
6. Click Delete in the Object Navigator.
7. Click Yes to confirm.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Client-Side Program Units

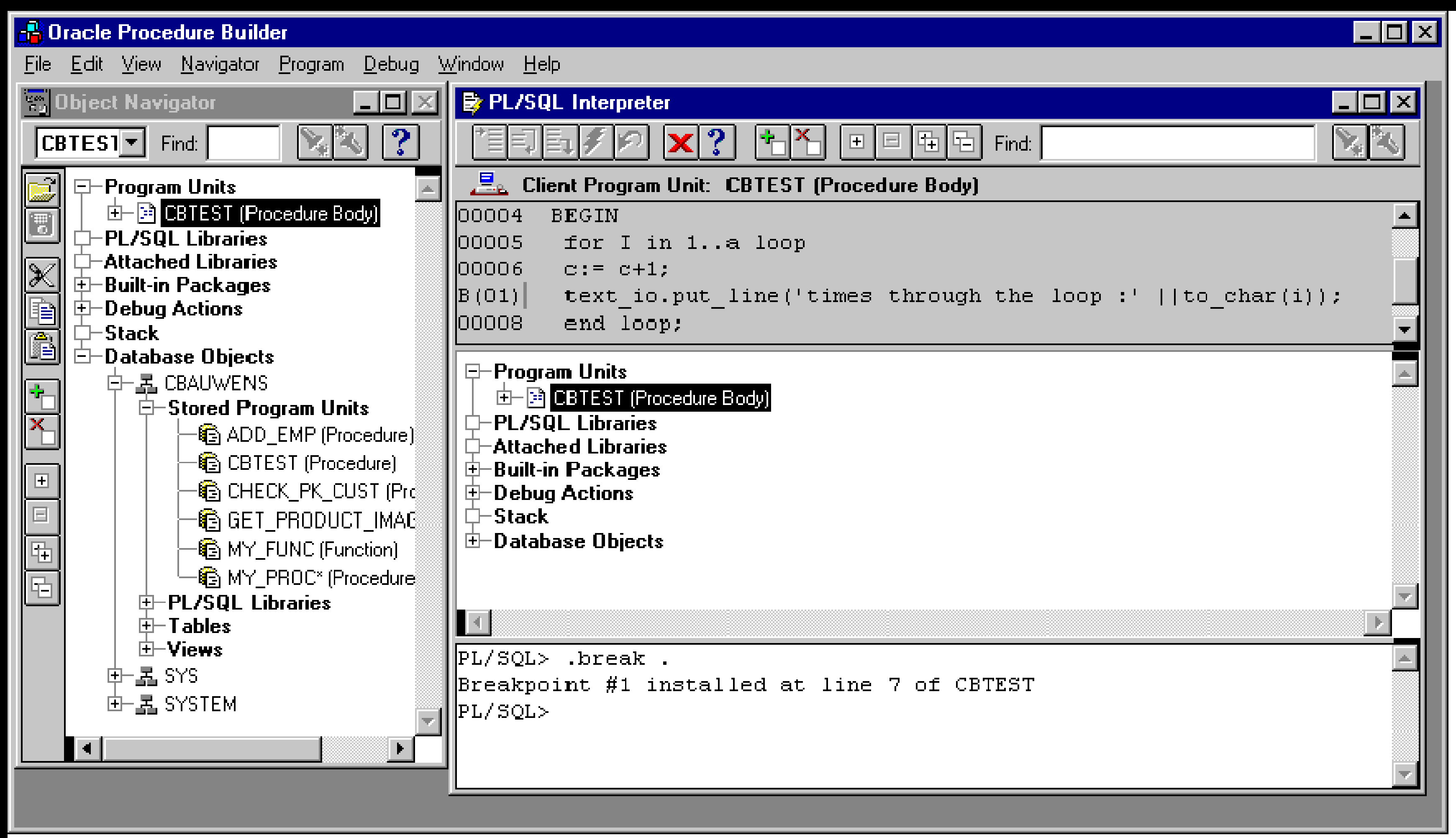
Using Procedure Builder:

1. Expand the Program Units node.
2. Click the program unit that you want to remove.
3. Click Delete in the Object Navigator.
4. Click Yes to confirm.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

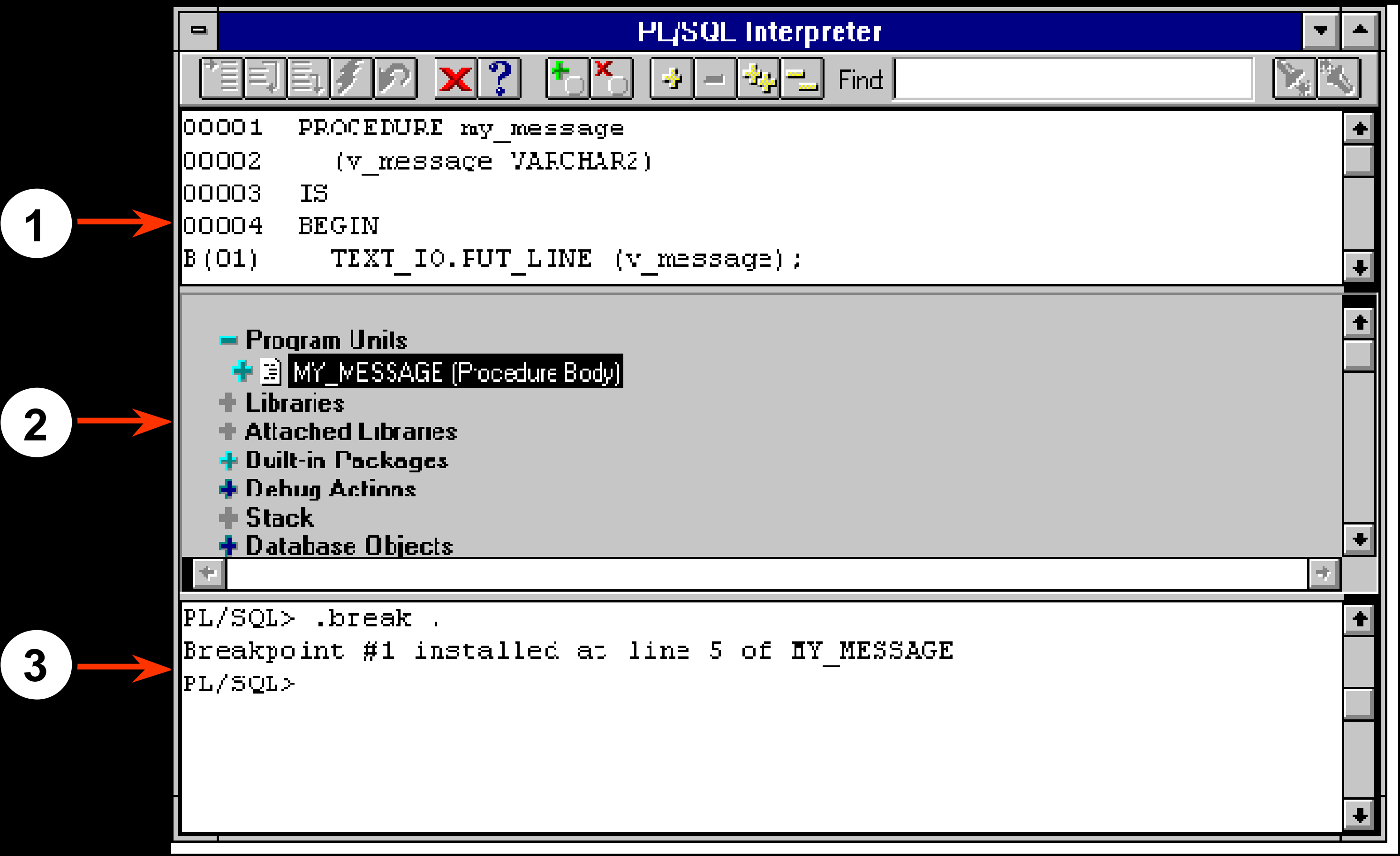
Debugging Subprograms by Using Procedure Builder



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

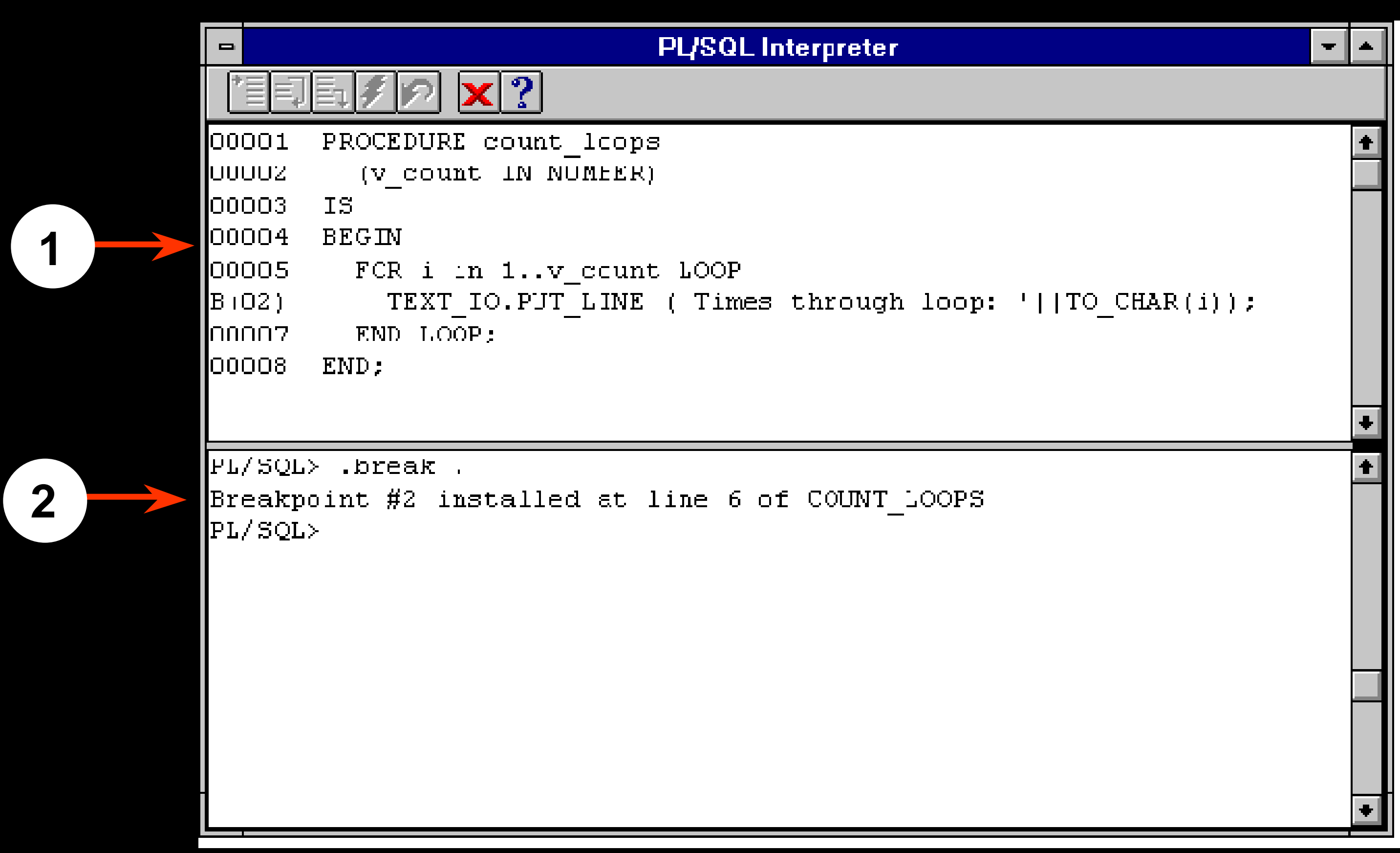
Listing Code in the Source Pane



Copyright © Oracle Corporation, 2001. All rights reserved.

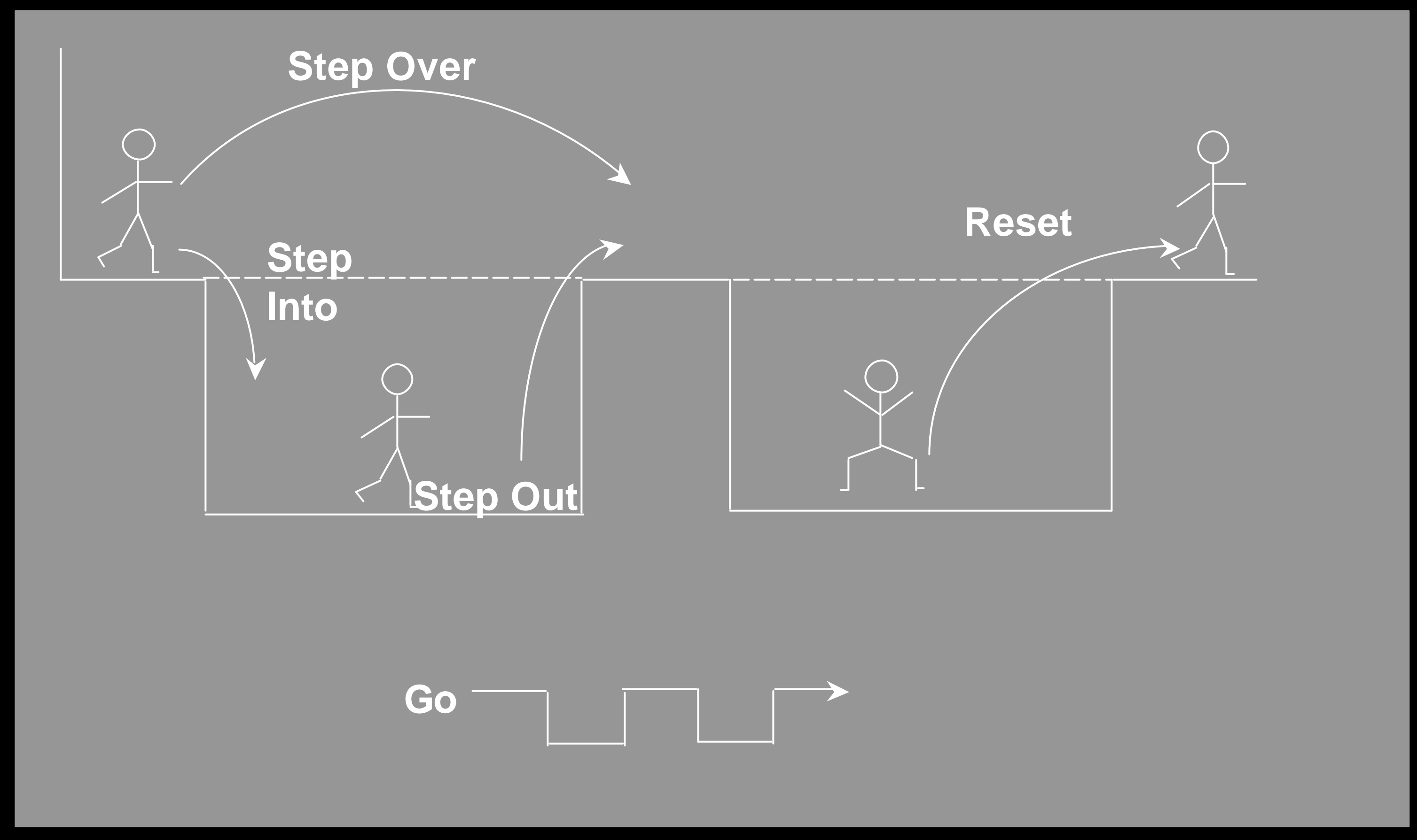
ORACLE

Setting a Breakpoint



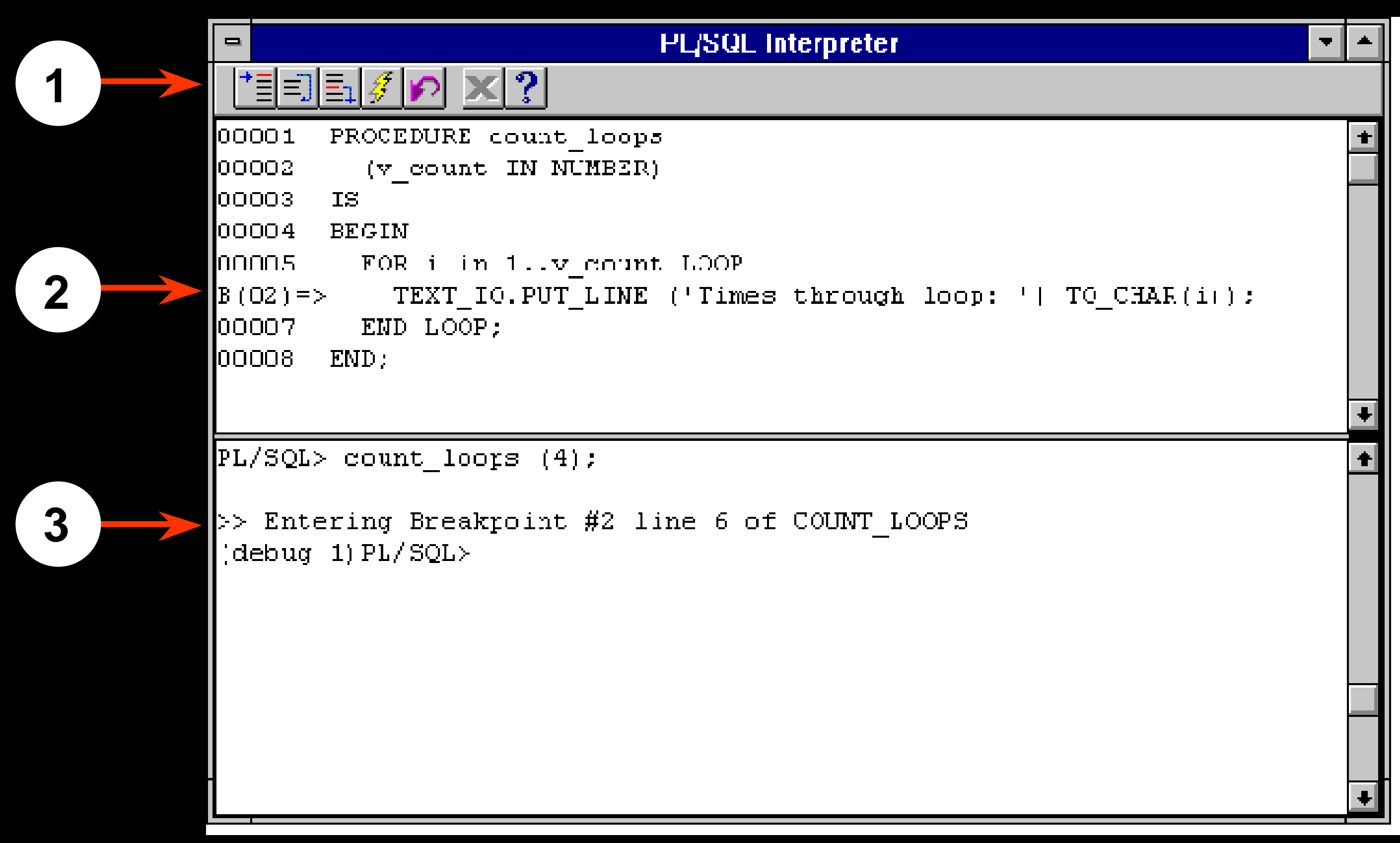
ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Debug Commands



Copyright © Oracle Corporation, 2001. All rights reserved.

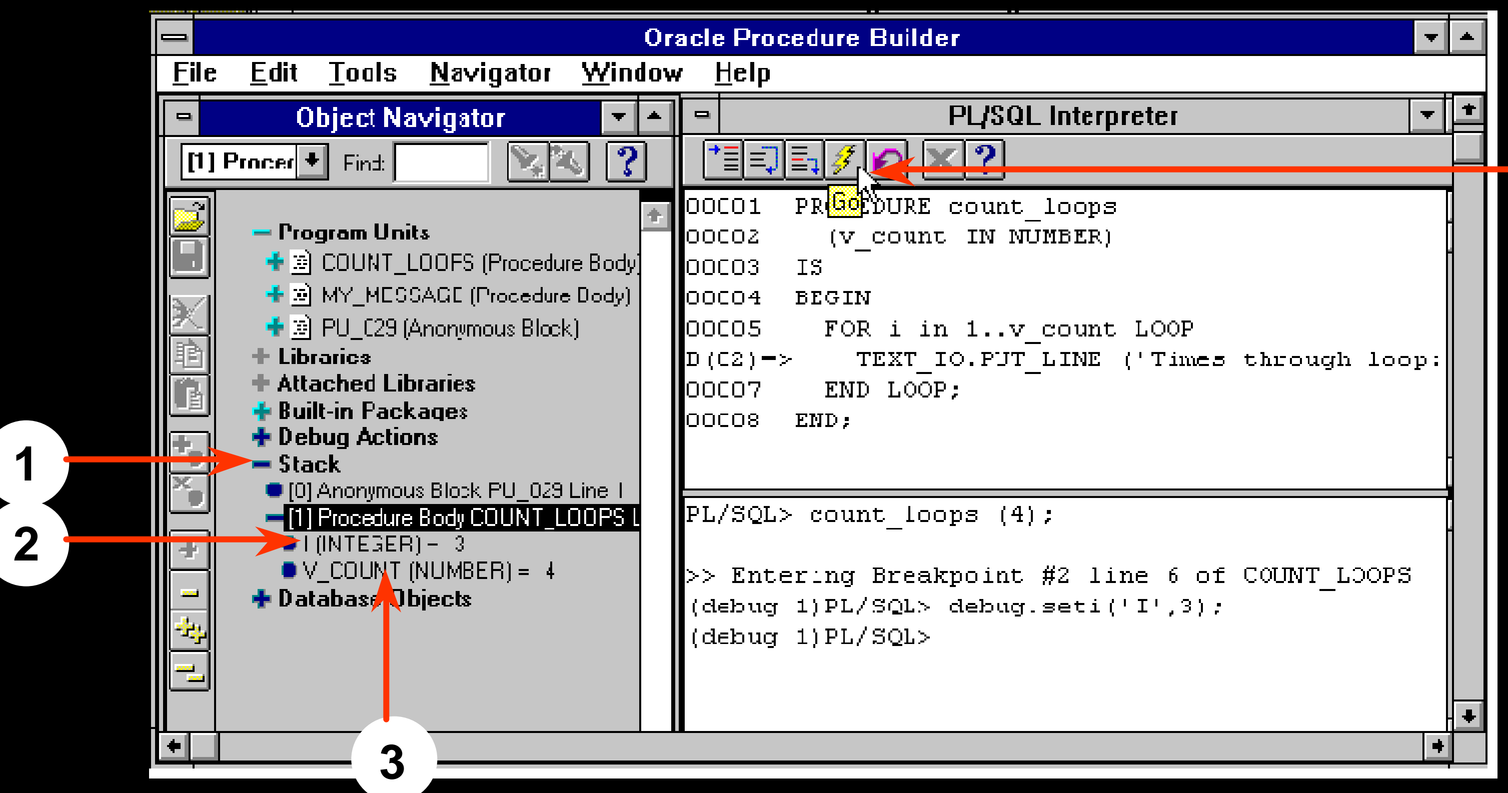
Stepping through Code



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Changing a Value



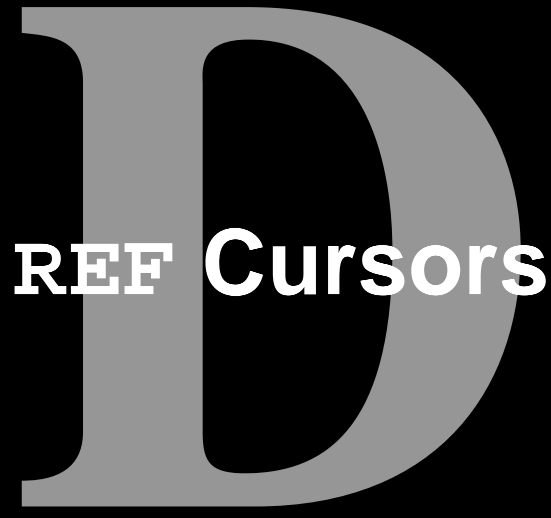
Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this appendix, you should have learned how to:

- Use Procedure Builder:
  - Application partitioning
  - Built-in editors
  - GUI execution environment
- Describe the components of Procedure Builder
  - Object Navigator
  - Program Unit Editor
  - PL/SQL Interpreter
  - Debugger

Copyright © Oracle Corporation, 2001. All rights reserved.



REF Cursors

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Cursor Variables

- Cursor variables are like C or Pascal pointers, which hold the memory location (address) of an item instead of the item itself
- In PL/SQL, a pointer is declared as `REF X`, where `REF` is short for `REFERENCE` and `X` stands for a class of objects
- A cursor variable has the data type `REF CURSOR`
- A cursor is static, but a cursor variable is dynamic
- Cursor variables give you more flexibility

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Why Use Cursor Variables?

- You can use cursor variables to pass query result sets between PL/SQL stored subprograms and various clients.
- PL/SQL can share a pointer to the query work area in which the result set is stored.
- You can pass the value of a cursor variable freely from one scope to another.
- You can reduce network traffic by having a PL/SQL block open (or close) several host cursor variables in a single round trip.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Defining REF CURSOR Types

- Define a REF CURSOR type.

```
Define a REF CURSOR type
TYPE ref_type_name IS REF CURSOR [RETURN return_type];
```

- Declare a cursor variable of that type.

```
ref_cv ref_type_name;
```

- Example:

```
DECLARE
TYPE DeptCurTyp IS REF CURSOR RETURN
departments%ROWTYPE;
dept_cv DeptCurTyp;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the `OPEN-FOR`, `FETCH`, and `CLOSE` Statements

- The `OPEN-FOR` statement associates a cursor variable with a multirow query, executes the query, identifies the result set, and positions the cursor to point to the first row of the result set.
- The `FETCH` statement returns a row from the result set of a multirow query, assigns the values of select-list items to corresponding variables or fields in the `INTO` clause, increments the count kept by `%ROWCOUNT`, and advances the cursor to the next row.
- The `CLOSE` statement disables a cursor variable.

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

An Example of Fetching

```
DECLARE
   TYPE EmpCurTyp IS REF CURSOR;
   emp_cv   EmpCurTyp;
   emp_rec  employees%ROWTYPE;
   sql_stmt VARCHAR2(200);
   my_job   VARCHAR2(10) := 'ST_CLERK';
BEGIN
   sql_stmt := 'SELECT * FROM employees
                WHERE job_id = :j';
   OPEN emp_cv FOR sql_stmt USING my_job;
   LOOP
      FETCH emp_cv INTO emp_rec;
      EXIT WHEN emp_cv%NOTFOUND;
      -- process record
   END LOOP;
   CLOSE emp_cv;
END;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.