Inatel

Instituto Nacional de Telecomunicações

Oracle Database 11g & PL/SQL

“2 Day Developer's Guide”
Overview and Examples

Marcelo Vinícius Cysneiros Aragão
marcelovca90@inatel.br

Topics

1. Topics
2. Connecting to Oracle Database and Exploring It
3. About DML Statements and Transactions
4. Creating and Managing Schema Objects
5. Developing Stored Subprograms and Packages
   0. About Procedural Language/SQL (PL/SQL)
   1. Creating and Managing Standalone Subprograms
   2. Creating and Managing Packages
   3. Declaring and Assigning Values to Variables and Constants
   4. Controlling Program Flow
   5. Using Records and Cursors
   6. Using Associative Arrays
   7. Handling Exceptions (Runtime Errors)

Topics (continuation)

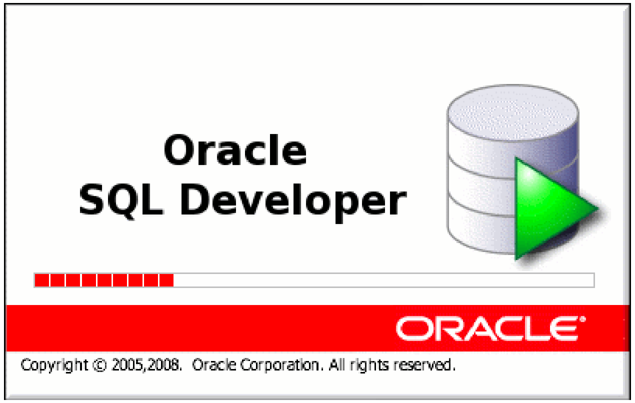
6. Using Triggers
7. Working in a Global Environment
8. Building Effective Applications
   1. Building Scalable Applications
   2. Recommended Programming Practices
   3. Recommended Security Practices
9. Study: Developing a Simple Oracle Database Application
10. Deploying an Oracle Database Application
11. Reference Material

2. Connecting to Oracle Database and Exploring It

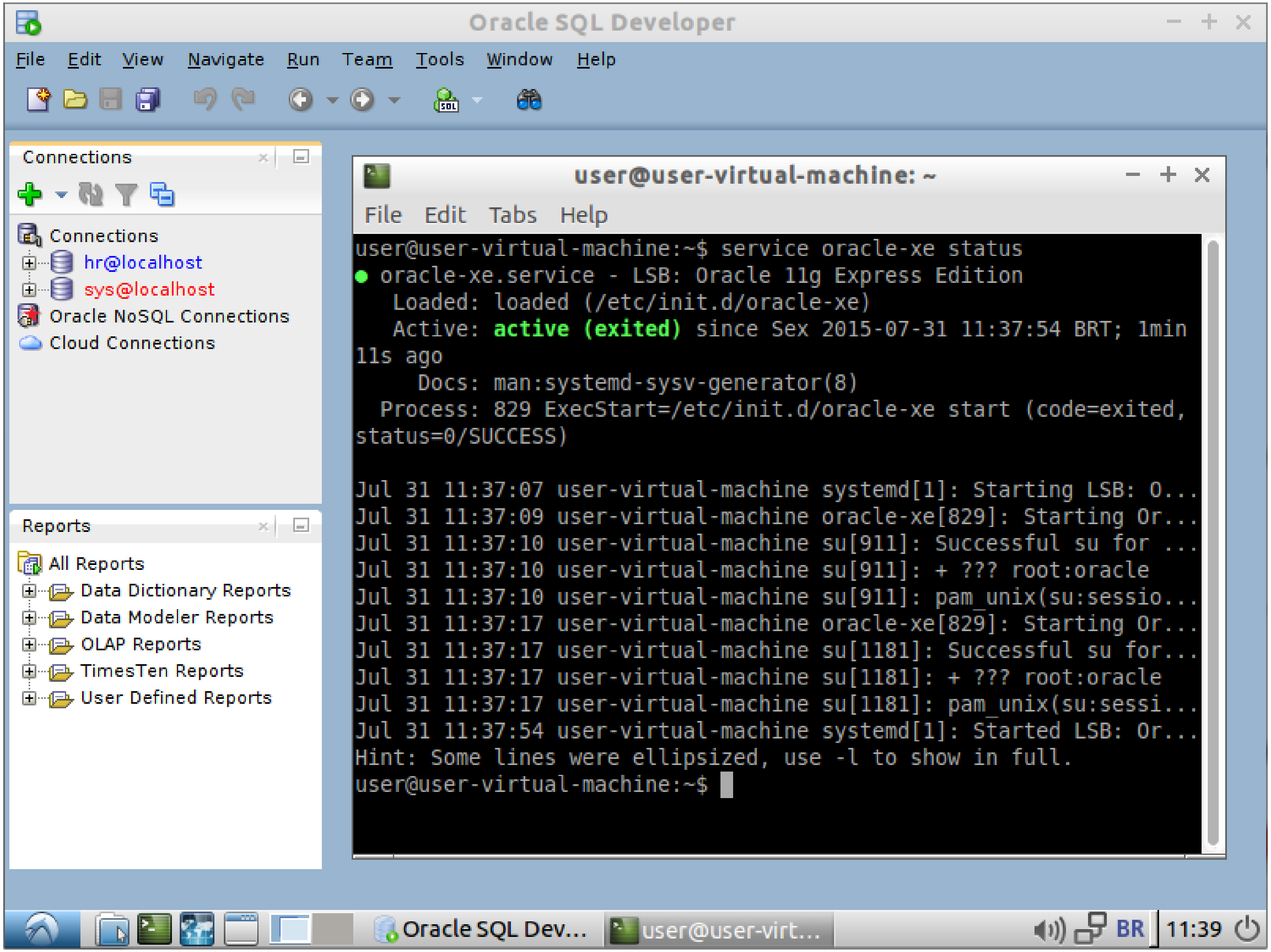
- Option 1:

```
> sqlplus
SQL*Plus: Release 12.1.0.1.0 Production on Thu Dec 27 07:43:41 2012
Copyright (c) 1982, 2012, Oracle.  All rights reserved.
Enter user-name: your_user_name
Enter password: your_password
Connected to: Oracle Database 12c Enterprise Edition Release - 12.1.0.1.0 64bit
SQL> select count(*) from employees;
```

- Option 2:

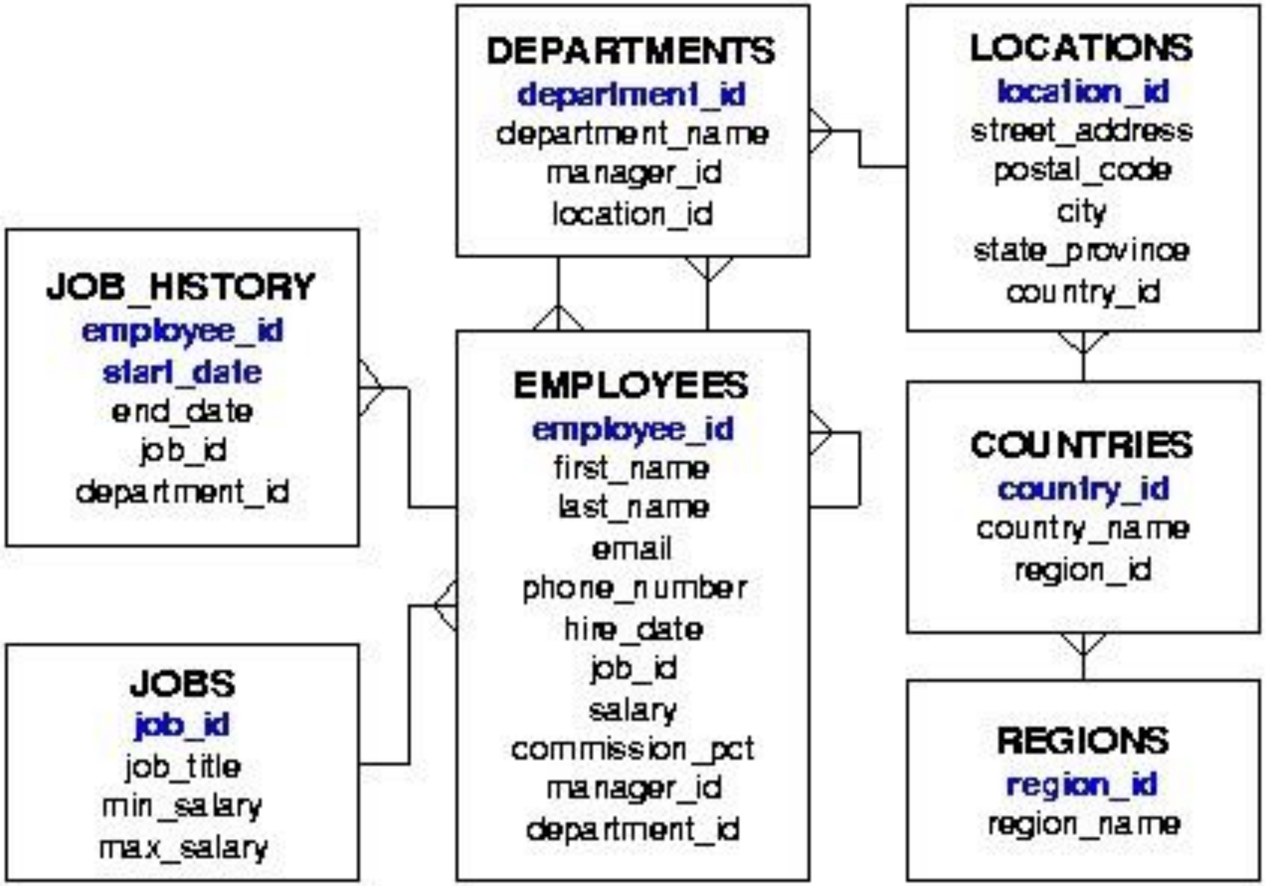


Copyright © 2005,2008. Oracle Corporation. All rights reserved.



```
user@user-virtual-machine:~$ service oracle-xe status
● oracle-xe.service - LSB: Oracle 11g Express Edition
   Loaded: loaded (/etc/init.d/oracle-xe)
   Active: active (exited) since Sex 2015-07-31 11:37:54 BRT; 1min
11s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 829 ExecStart=/etc/init.d/oracle-xe start (code=exited,
status=0/SUCCESS)

Jul 31 11:37:07 user-virtual-machine systemd[1]: Starting LSB: O...
Jul 31 11:37:09 user-virtual-machine oracle-xe[829]: Starting Or...
Jul 31 11:37:10 user-virtual-machine su[911]: Successful su for ...
Jul 31 11:37:10 user-virtual-machine su[911]: + ??? root:oracle
Jul 31 11:37:10 user-virtual-machine su[911]: pam_unix(su:sessio...
Jul 31 11:37:17 user-virtual-machine oracle-xe[829]: Starting Or...
Jul 31 11:37:17 user-virtual-machine su[1181]: Successful su for...
Jul 31 11:37:17 user-virtual-machine su[1181]: + ??? root:oracle
Jul 31 11:37:17 user-virtual-machine su[1181]: pam_unix(su:sessi...
Jul 31 11:37:54 user-virtual-machine systemd[1]: Started LSB: Or...
Hint: Some lines were ellipsized, use -l to show in full.
user@user-virtual-machine:~$
```

The HR (Human Resources) Schema



3. About DML Statements and Transactions

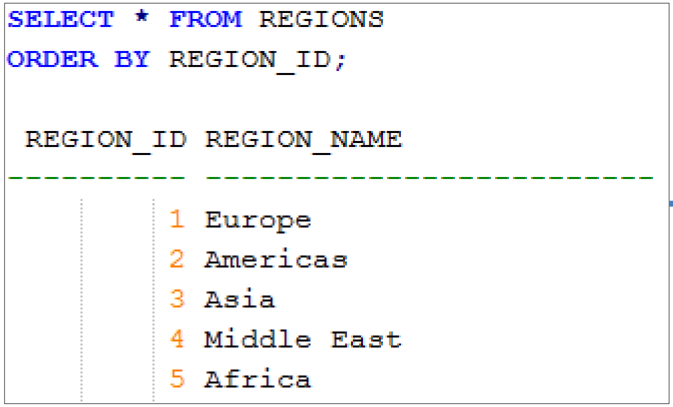
DML Statements

- INSERT, UPDATE and DELETE can be used in the same way as regular SQL.

Transaction Control Statements

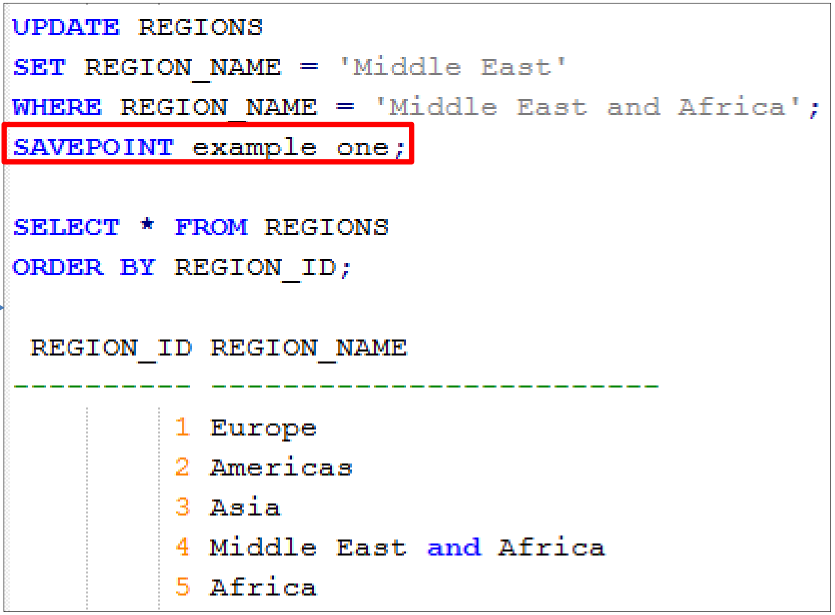
- A transaction is a sequence of one or more SQL statements that Oracle Database treats as a unit: either all of the statements are performed, or none of them are.
- SAVEPOINT, which marks a savepoint in a transaction - a point to which you can later roll back. Savepoints are optional, and a transaction can have multiple savepoints.
- COMMIT, which ends the current transaction, makes its changes permanent, erases its savepoints, and releases its locks.
- ROLLBACK, which rolls back (undoes) either the entire current transaction or only the changes made after the specified savepoint.

Transaction Control Statements



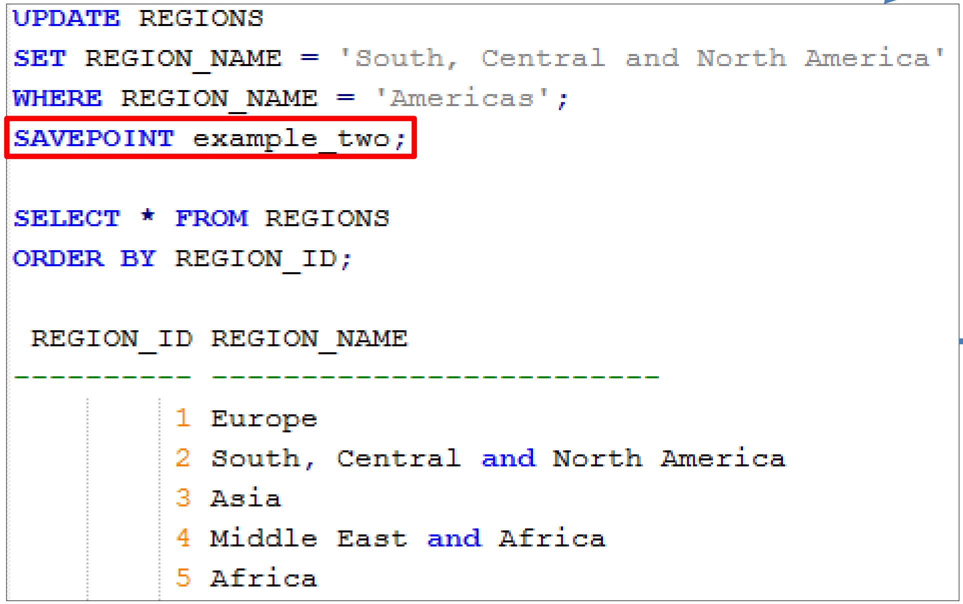
```
SELECT * FROM REGIONS
ORDER BY REGION_ID;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
         5 Africa
```

1



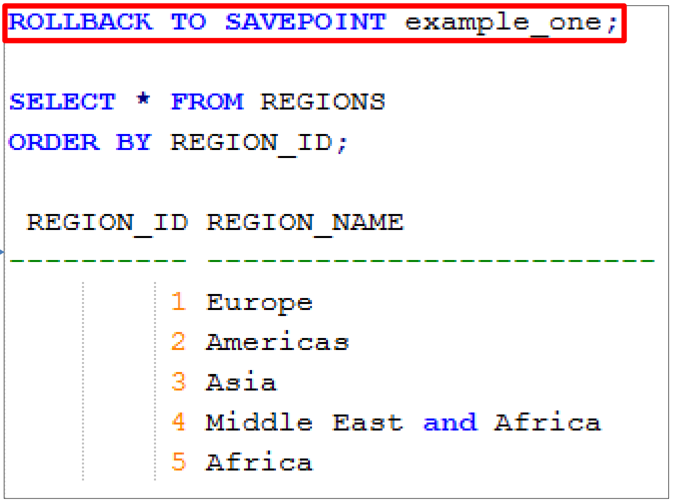
```
UPDATE REGIONS
SET REGION_NAME = 'Middle East'
WHERE REGION_NAME = 'Middle East and Africa';
SAVEPOINT example_one;

SELECT * FROM REGIONS
ORDER BY REGION_ID;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa
```

2



```
UPDATE REGIONS
SET REGION_NAME = 'South, Central and North America'
WHERE REGION_NAME = 'Americas';
SAVEPOINT example_two;

SELECT * FROM REGIONS
ORDER BY REGION_ID;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 South, Central and North America
         3 Asia
         4 Middle East and Africa
         5 Africa
```

3



```
ROLLBACK TO SAVEPOINT example_one;

SELECT * FROM REGIONS
ORDER BY REGION_ID;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa
```

4. Creating and Managing Schema Objects

- Tables - The basic units of data storage in Oracle Database. Tables hold all user-accessible data. You can create indexes on one or more columns of a table to speed SQL statement execution on that table, reducing disk I/O.

- Views - A view presents a query result as a table. Views are useful when you need frequent access to information that is stored in different tables.

- Sequences - Objects that can generate unique sequential values (useful when you need unique primary keys). The pseudocolumns CURRVAL and NEXTVAL return the current and next values of the sequence, respectively.

- Synonyms - An alias for another schema object. Some reasons to use synonyms are security and convenience. Examples:
  - Using a short synonym (e.g. SALES) for a long object name (e.g. ACME_CO.SALES_DATA);
  - Using a synonym for a renamed object, instead of changing that object name throughout the applications that use it.

5. Developing Stored Subprograms and Packages

5.0 About Procedural Language/SQL (PL/SQL)

Procedural Language/SQL (PL/SQL) (pronounced *P L sequel*) is a native Oracle Database extension to SQL. In PL/SQL, you can:

- Declare constants, variables, procedures, functions, types and triggers.
- Handle exceptions (runtime errors).
- Create units - procedures, functions, packages, types, and triggers - that are stored in the database for reuse by applications that use any of the Oracle Database programmatic interfaces.

The basic unit of a PL/SQL source program is the block, which groups related declarations and statements. A block has an optional declarative part, a required executable part, and an optional exception-handling part.

5. Developing Stored Subprograms and Packages

5.1 Creating and Managing Standalone Subprograms

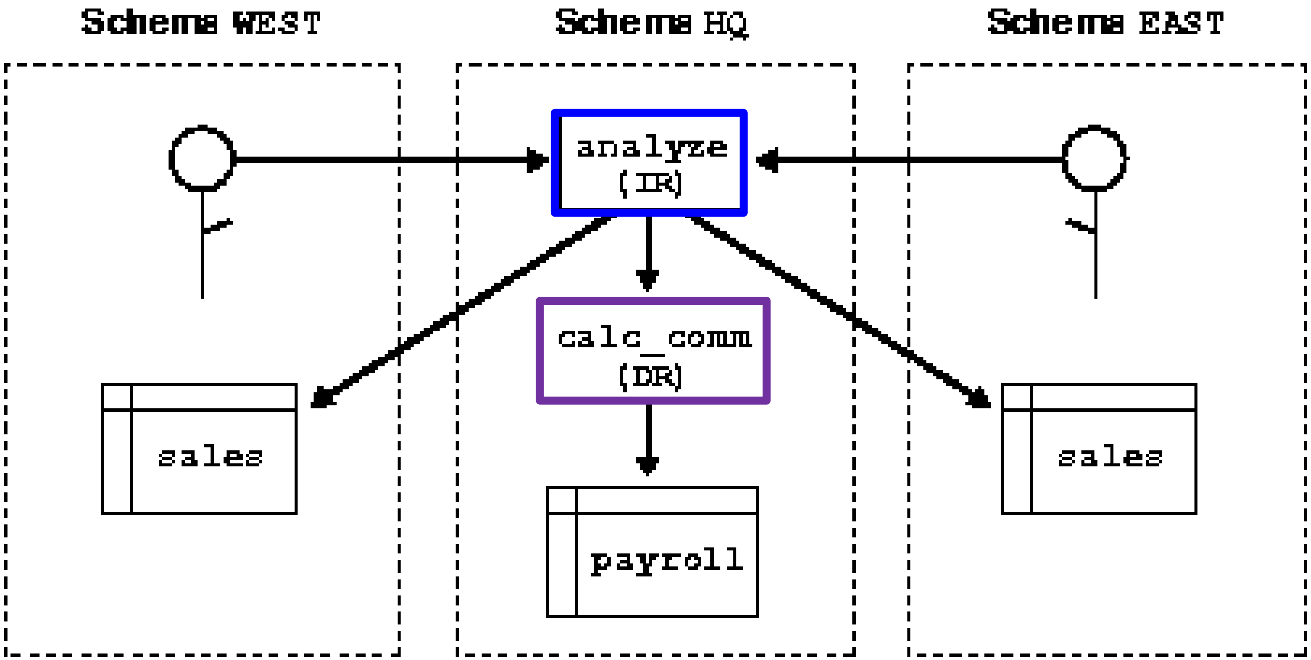
- A subprogram is a PL/SQL unit that consists of SQL and PL/SQL statements that solve a specific problem or perform a set of related tasks.
  - A subprogram can have parameters, whose values are supplied by the invoker.
  - A subprogram can be either a procedure (action) or a function (compute and return a value).

- A stored subprogram is a subprogram that is stored in the database. They can be used as building blocks for many different database applications.
  - Standalone subprogram, which is created at schema level
  - Package subprogram, which is created inside a package

5. Developing Stored Subprograms and Packages

5.1 Creating and Managing Standalone Subprograms

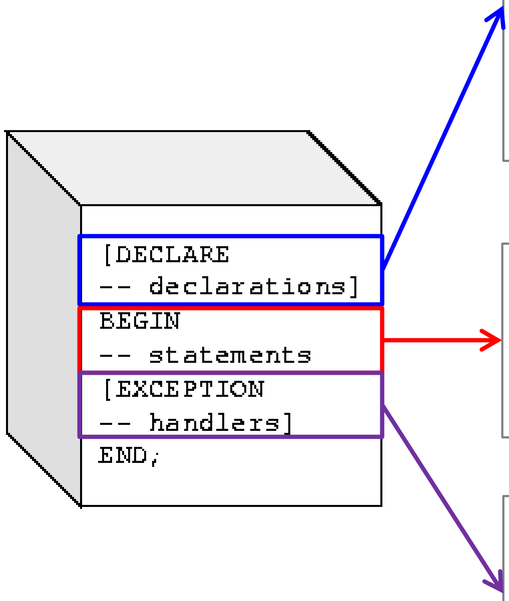
IR (*invoker-rights*): subprograms are **NOT** bound to a particular schema



DR (*definer-rights*): subprograms **ARE** bound to the schema in which they reside

5. Developing Stored Subprograms and Packages

5.1 Creating and Managing Standalone Subprograms



```
[DECLARE
-- declarations]
BEGIN
-- statements
[EXCEPTION
-- handlers]
END;
```

■ Declarative part (optional)

The declarative part contains declarations of types, constants, variables, exceptions, declared cursors, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.

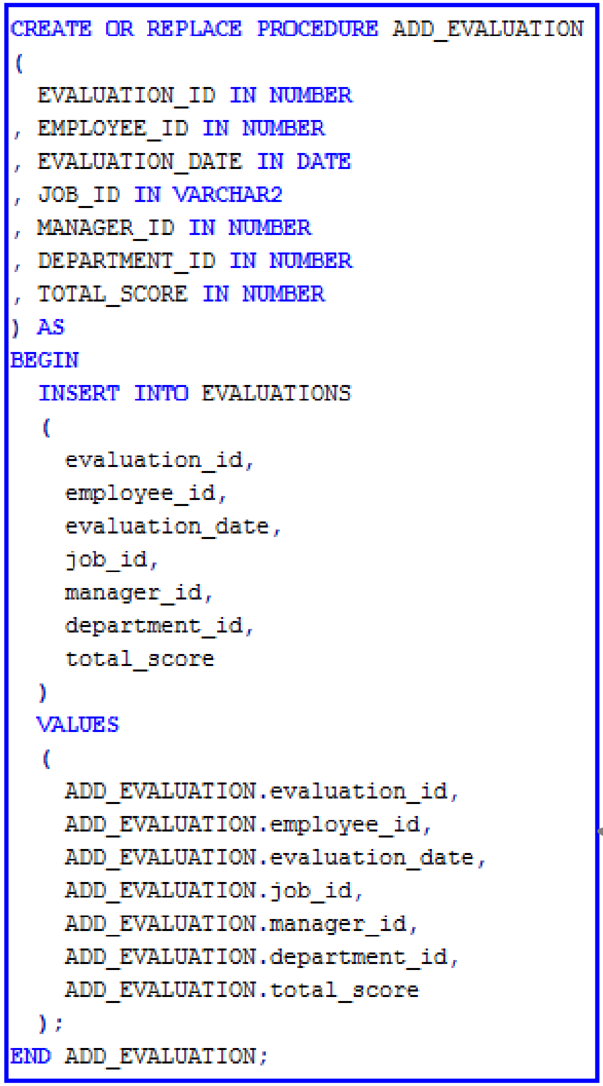
■ Executable part (required)

The executable part contains statements that assign values, control execution, and manipulate data.

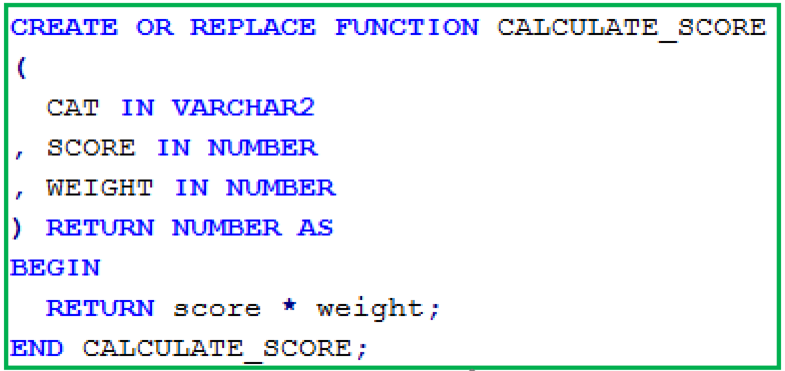
■ Exception-handling part (optional)

The exception-handling part contains code that handles exceptions (runtime errors).

5.1 Creating and Managing Standalone Subprograms



```
CREATE OR REPLACE PROCEDURE ADD_EVALUATION
(
  EVALUATION_ID IN NUMBER
, EMPLOYEE_ID IN NUMBER
, EVALUATION_DATE IN DATE
, JOB_ID IN VARCHAR2
, MANAGER_ID IN NUMBER
, DEPARTMENT_ID IN NUMBER
, TOTAL_SCORE IN NUMBER
) AS
BEGIN
  INSERT INTO EVALUATIONS
  (
    evaluation_id,
    employee_id,
    evaluation_date,
    job_id,
    manager_id,
    department_id,
    total_score
  )
  VALUES
  (
    ADD_EVALUATION.evaluation_id,
    ADD_EVALUATION.employee_id,
    ADD_EVALUATION.evaluation_date,
    ADD_EVALUATION.job_id,
    ADD_EVALUATION.manager_id,
    ADD_EVALUATION.department_id,
    ADD_EVALUATION.total_score
  );
END ADD_EVALUATION;
```



```
CREATE OR REPLACE FUNCTION CALCULATE_SCORE
(
  CAT IN VARCHAR2
, SCORE IN NUMBER
, WEIGHT IN NUMBER
) RETURN NUMBER AS
BEGIN
  RETURN score * weight;
END CALCULATE_SCORE;
```

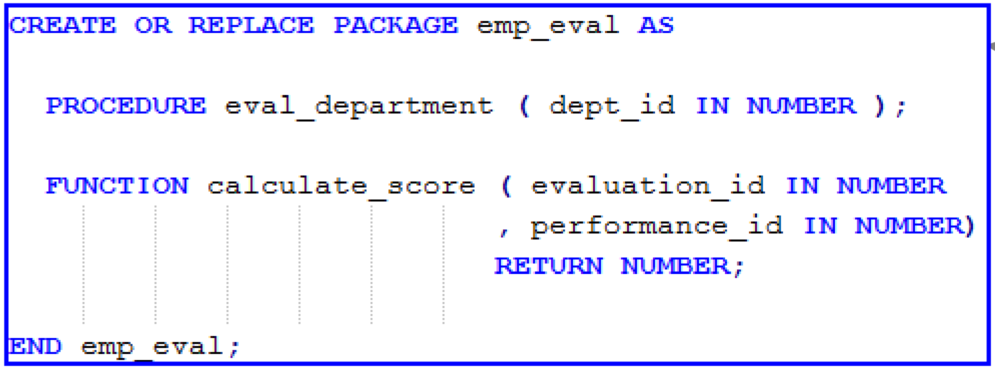
Standalone Function

Standalone Procedure

5.2 Creating and Managing Packages

- Package Specification
  - Defines the package, declaring the types, variables, constants, exceptions, declared cursors, and subprograms that can be referenced from outside the package.
  - Is an application program interface (API): It has all the information that client programs need to invoke its subprograms, but no information about their implementation.
- Package Body
  - Defines the queries for the declared cursors, and the code for the subprograms, that are declared in the package specification (therefore, a package with neither declared cursors nor subprograms does not need a body).
  - Can also define local subprograms, which are not declared in the specification and can be invoked only by other subprograms in the package.
  - Its contents are hidden from client programs.
  - Can be changed without invalidating the applications that call the package.

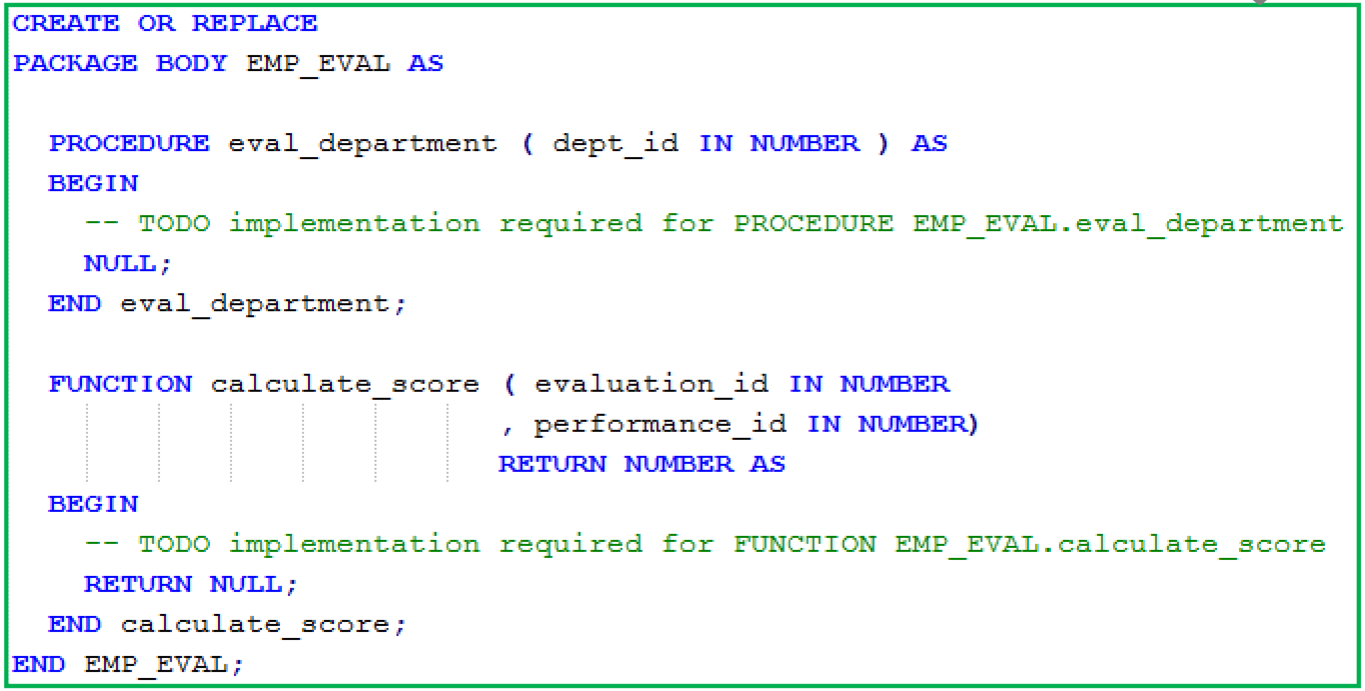
5.2 Creating and Managing Packages



```
CREATE OR REPLACE PACKAGE emp_eval AS

  PROCEDURE eval_department ( dept_id IN NUMBER );

  FUNCTION calculate_score ( evaluation_id IN NUMBER
                           , performance_id IN NUMBER)
                           RETURN NUMBER;

END emp_eval;
```

Package Specification

Package Body



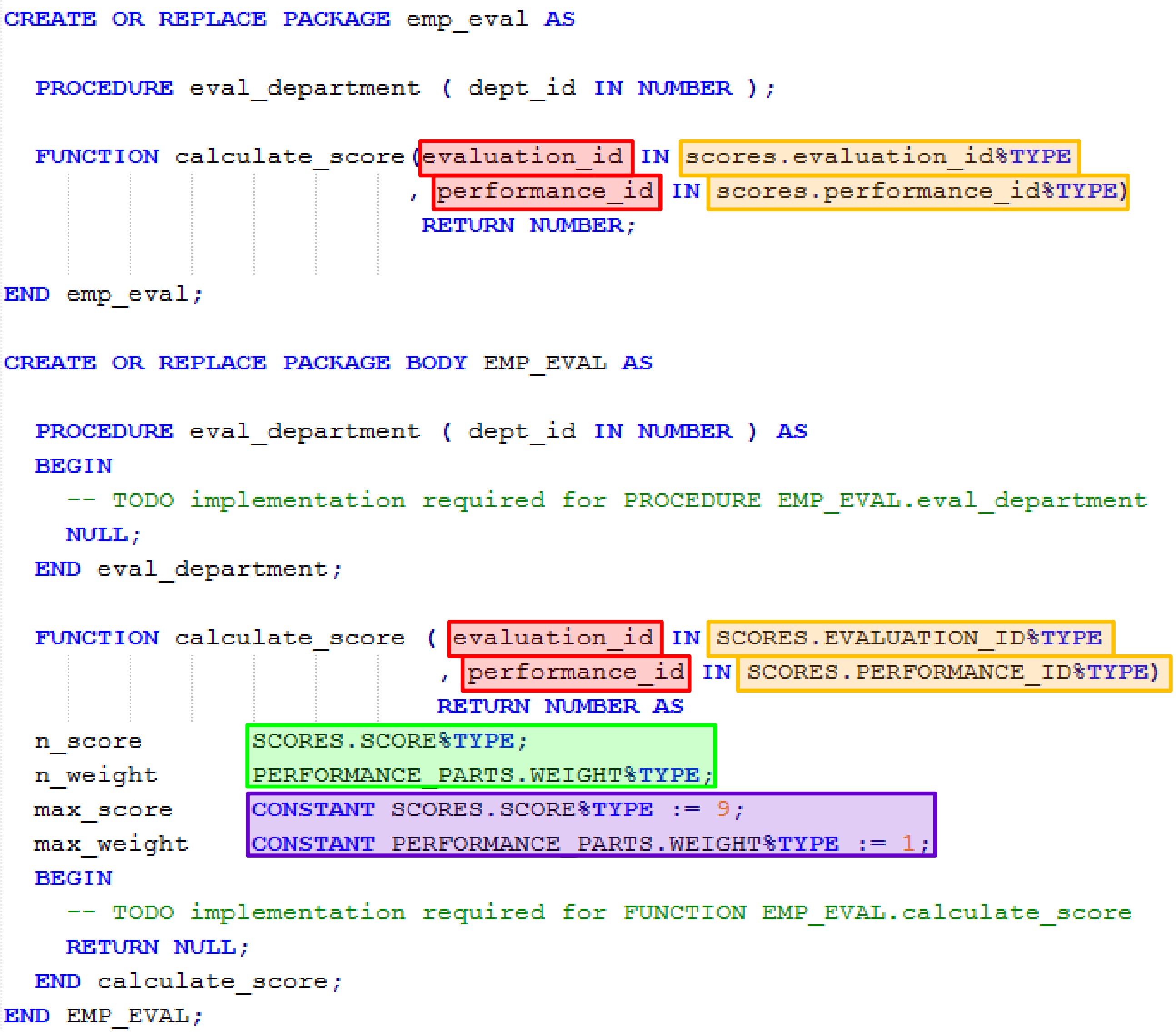
```
CREATE OR REPLACE
PACKAGE BODY EMP_EVAL AS

  PROCEDURE eval_department ( dept_id IN NUMBER ) AS
  BEGIN
    -- TODO implementation required for PROCEDURE EMP_EVAL.eval_department
    NULL;
  END eval_department;

  FUNCTION calculate_score ( evaluation_id IN NUMBER
                           , performance_id IN NUMBER)
                           RETURN NUMBER AS
  BEGIN
    -- TODO implementation required for FUNCTION EMP_EVAL.calculate_score
    RETURN NULL;
  END calculate_score;
END EMP_EVAL;
```

5.3 Declaring and Assigning Values to Variables and Constants

- One significant advantage that PL/SQL has over SQL is that PL/SQL lets you declare and use variables and constants.
  - A variable holds a value of a particular data type that can be changed at runtime. You can assign it an initial value; if you do not, its initial value is NULL.
  - A constant holds a value that cannot be changed. When declaring a constant, you must assign it an initial value.
  - To assign an initial value to a variable or constant, use the assignment operator (:=).
- Visibility
  - A variable or constant declared in a package specification is available to any program that has access to the package.
  - A variable or constant declared in a package body or subprogram is local to that package or subprogram.

Inatel
Instituto Nacional de Telecomunicações

5.3 Declaring and Assigning Values to Variables and Constants



```
CREATE OR REPLACE PACKAGE emp_eval AS

  PROCEDURE eval_department ( dept_id IN NUMBER );

  FUNCTION calculate_score (evaluation_id IN scores.evaluation_id%TYPE
                          , performance_id IN scores.performance_id%TYPE)
                           RETURN NUMBER;

END emp_eval;

CREATE OR REPLACE PACKAGE BODY EMP_EVAL AS

  PROCEDURE eval_department ( dept_id IN NUMBER ) AS
  BEGIN
    -- TODO implementation required for PROCEDURE EMP_EVAL.eval_department
    NULL;
  END eval_department;

  FUNCTION calculate_score ( evaluation_id IN SCORES.EVALUATION_ID%TYPE
                            , performance_id IN SCORES.PERFORMANCE_ID%TYPE)
                            RETURN NUMBER AS
  n_score       SCORES.SCORE%TYPE;
  n_weight      PERFORMANCE_PARTS.WEIGHT%TYPE;
  max_score     CONSTANT SCORES.SCORE%TYPE := 9;
  max_weight    CONSTANT PERFORMANCE_PARTS.WEIGHT%TYPE := 1;
  BEGIN
    -- TODO implementation required for FUNCTION EMP_EVAL.calculate_score
    RETURN NULL;
  END calculate_score;
END EMP_EVAL;
```

PARAMETERS

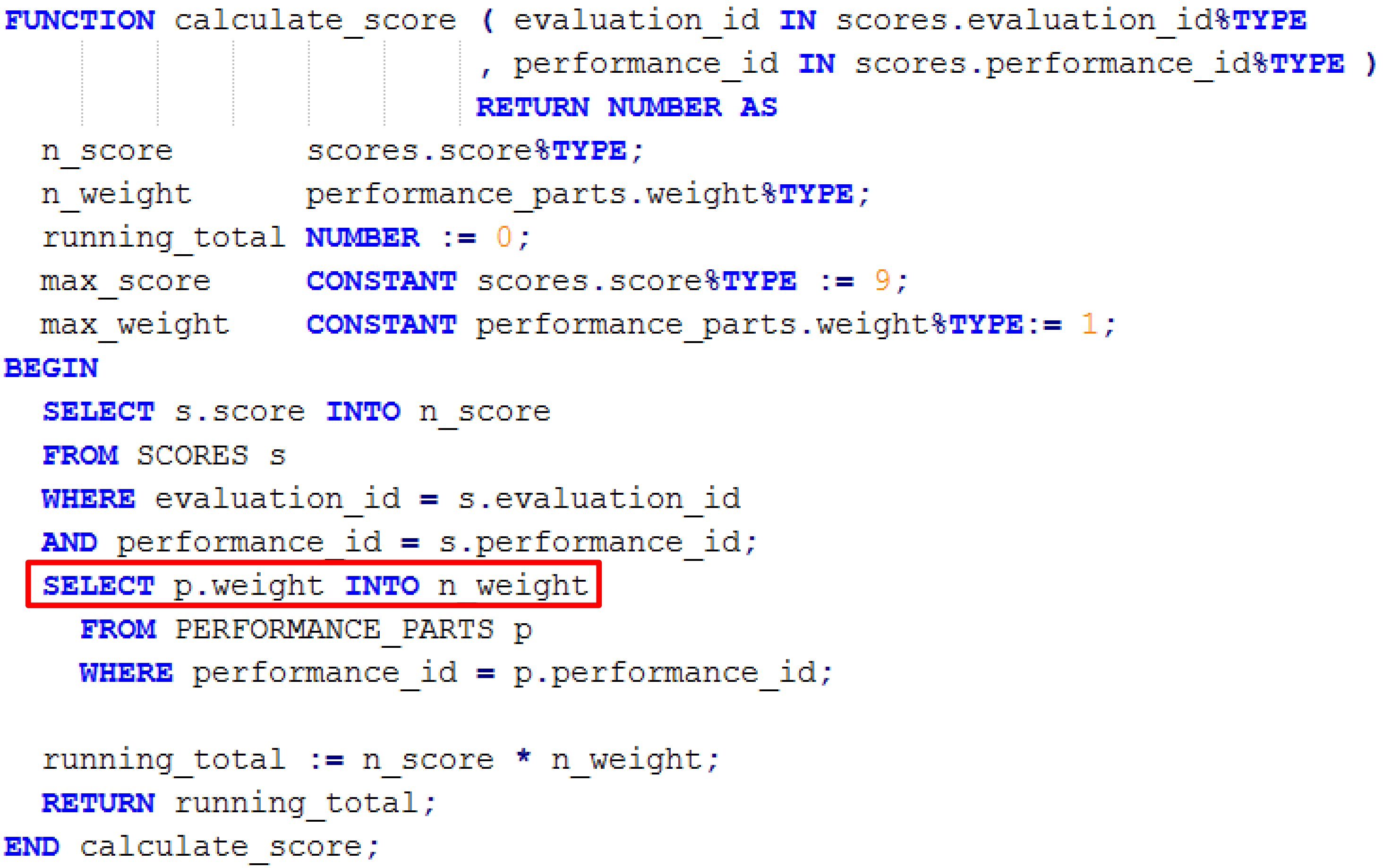
DATA TYPES

VARIABLES

CONSTANTS

5.3 Declaring and Assigning Values to Variables and Constants

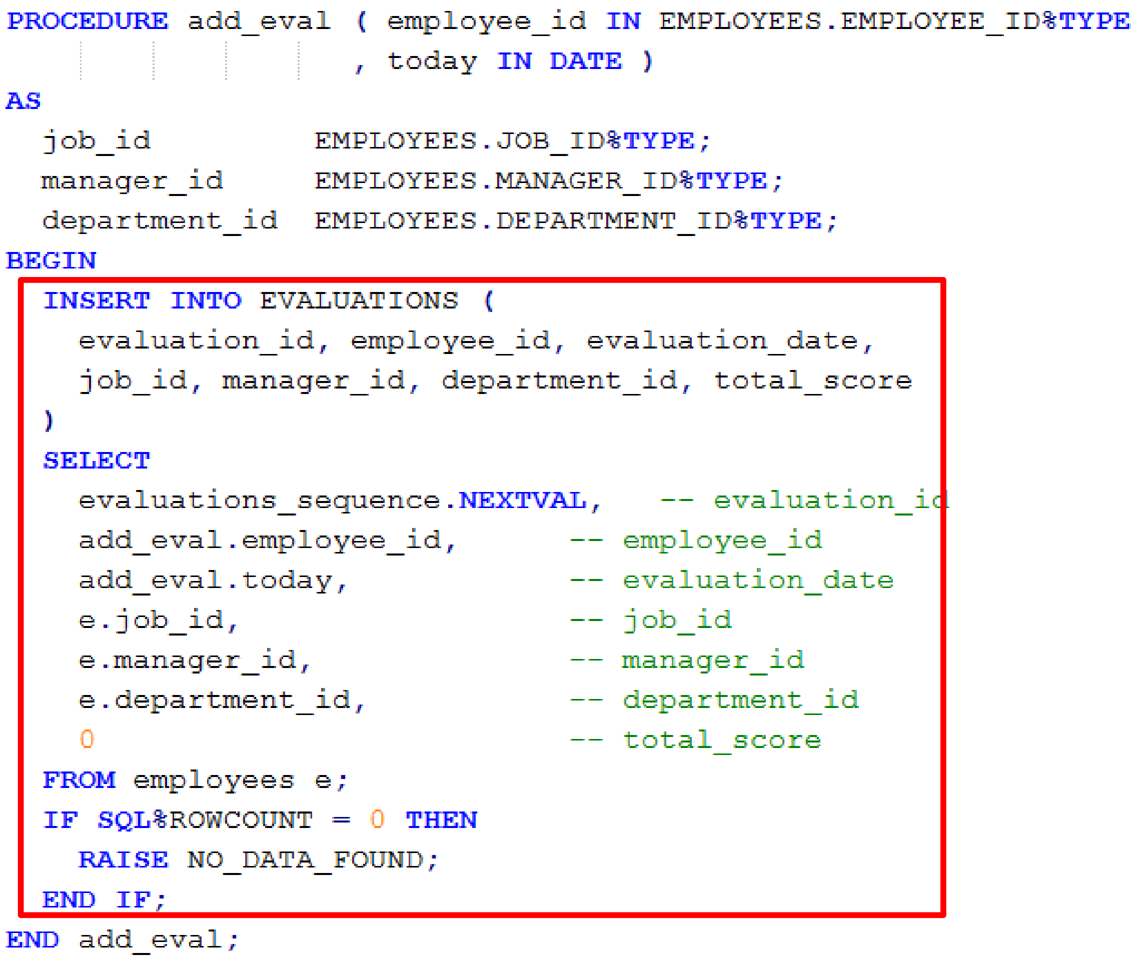
- Assigning Values to Variables with the SELECT INTO Statement



```
FUNCTION calculate_score ( evaluation_id IN scores.evaluation_id%TYPE
                         , performance_id IN scores.performance_id%TYPE )
                         RETURN NUMBER AS
  n_score       scores.score%TYPE;
  n_weight      performance_parts.weight%TYPE;
  running_total NUMBER := 0;
  max_score     CONSTANT scores.score%TYPE := 9;
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
BEGIN
  SELECT s.score INTO n_score
  FROM SCORES s
  WHERE evaluation_id = s.evaluation_id
  AND performance_id = s.performance_id;
  SELECT p.weight INTO n_weight
    FROM PERFORMANCE_PARTS p
    WHERE performance_id = p.performance_id;

  running_total := n_score * n_weight;
  RETURN running_total;
END calculate_score;
```

5.3 Declaring and Assigning Values to Variables and Constants

- Inserting a Table Row with Values from Another Table



```
PROCEDURE add_eval ( employee_id IN EMPLOYEES.EMPLOYEE_ID%TYPE
                   , today IN DATE )
AS
  job_id          EMPLOYEES.JOB_ID%TYPE;
  manager_id      EMPLOYEES.MANAGER_ID%TYPE;
  department_id   EMPLOYEES.DEPARTMENT_ID%TYPE;
BEGIN
  INSERT INTO EVALUATIONS (
    evaluation_id, employee_id, evaluation_date,
    job_id, manager_id, department_id, total_score
  )
  SELECT
    evaluations_sequence.NEXTVAL,    -- evaluation_id
    add_eval.employee_id,       -- employee_id
    add_eval.today,             -- evaluation_date
    e.job_id,                   -- job_id
    e.manager_id,               -- manager_id
    e.department_id,            -- department_id
    0                           -- total_score
  FROM employees e;
  IF SQL%ROWCOUNT = 0 THEN
    RAISE NO_DATA_FOUND;
  END IF;
END add_eval;
```

5.4 Controlling Program Flow

- **Conditional selection statements**
  - IF, ELSIF, ELSE and CASE.
- **Loop statements**
  - FOR LOOP, WHILE LOOP and basic LOOP;
  - EXIT: transfers control to the end of a loop;
  - CONTINUE: exits the current iteration of a loop and transfers control to the next iteration;
  - Both EXIT and CONTINUE have an optional WHEN clause, in which you can specify a condition.
- **Sequential control statements**
  - GOTO: go to a specified labeled statement;
  - NULL: do nothing.

5.4.1 IF/ELSE

```
IF boolean_expression THEN statement [, statement ]
[ ELSIF boolean_expression THEN statement [, statement ] ]...
[ ELSE  statement [, statement ] ]
END IF;
```

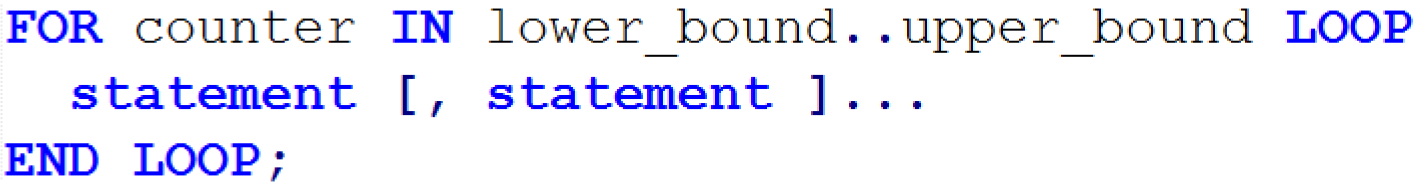
→ Code example in “queries.sql”, lines 240-261

5.4.2 CASE

```
CASE expression
WHEN value THEN statement
[ WHEN value THEN statement ]...
[ ELSE statement [, statement ]... ]
END CASE;
```

→ Code example in “queries.sql”, lines 263-302

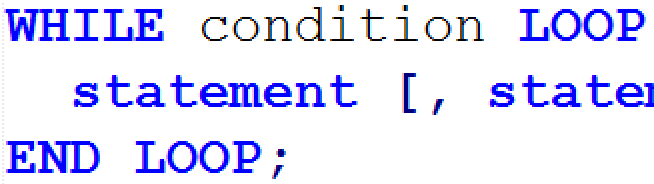
5.4.3 FOR



```
FOR counter IN lower_bound..upper_bound LOOP
  statement [, statement ]...
END LOOP;
```

→ Code example in “queries.sql”, lines 304-348

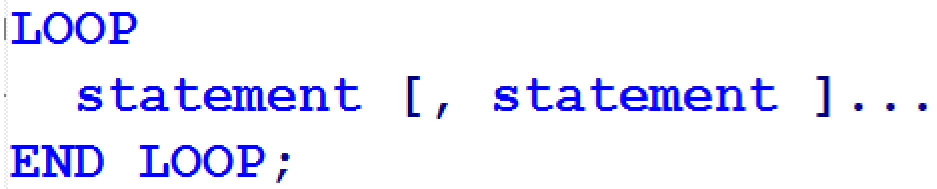
5.4.4 WHILE



```
WHILE condition LOOP
  statement [, statement ]...
END LOOP;
```

→ Code example in “queries.sql”, lines 350-397

5.4.5 Simple LOOP



```
LOOP
  statement [, statement ]...
END LOOP;
```

→ Code example in “queries.sql”, lines 399-447

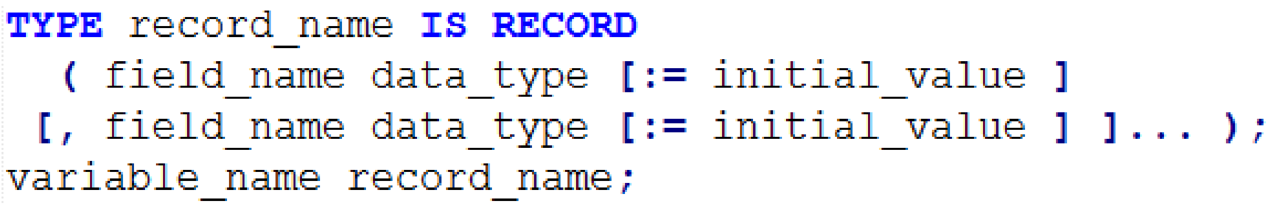
5.5 Using Records and Cursors

- A record is a PL/SQL composite variable that can store data values of different types, similar to a `struct` type in C, C++, or Java.
  - The internal components of a record are called fields.
  - To access a record field, you use dot notation: `record_name.field_name`.
- You can treat record fields like scalar variables. You can also pass entire records as subprogram parameters.
- Records are useful for holding data from table rows, or from certain columns of table rows. Each record field corresponds to a table column.

5.5 Using Records and Cursors

There are three ways to create a record:

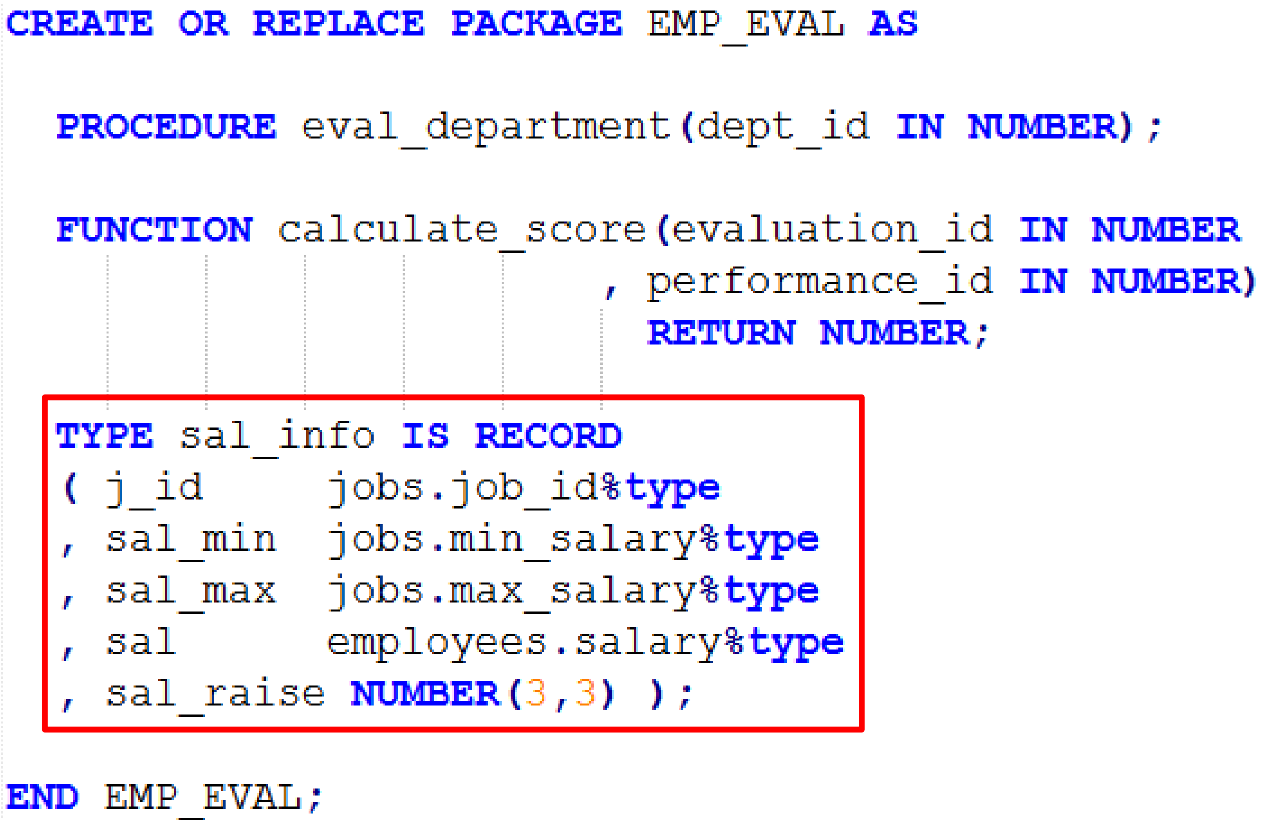
- Declare a RECORD type and then declare a variable of that type. The syntax is:



```
TYPE record_name IS RECORD
  ( field_name data_type [:= initial_value ]
 [, field_name data_type [:= initial_value ] ]... );
variable_name record_name;
```

- Declare a variable of the type `table_name%ROWTYPE`.
  - The fields of the record have the same names and data types as the columns of the table.
- Declare a variable of the type `cursor_name%ROWTYPE`.
  - The fields of the record have the same names and data types as the columns of the table in the FROM clause of the cursor SELECT statement.

5.5 Using Records and Cursors



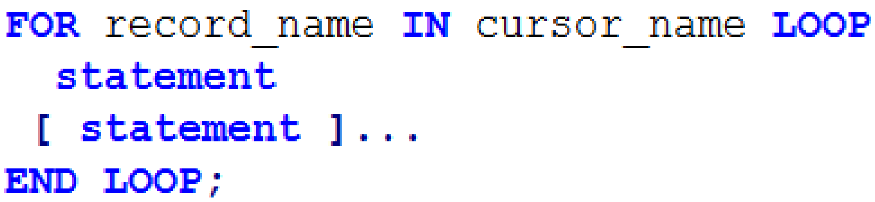
```
CREATE OR REPLACE PACKAGE EMP_EVAL AS

  PROCEDURE eval_department(dept_id IN NUMBER);

  FUNCTION calculate_score(evaluation_id IN NUMBER
                         , performance_id IN NUMBER)
                          RETURN NUMBER;

  TYPE sal_info IS RECORD
  ( j_id      jobs.job_id%type
  , sal_min   jobs.min_salary%type
  , sal_max   jobs.max_salary%type
  , sal       employees.salary%type
  , sal_raise NUMBER(3,3) );

END EMP_EVAL;
```

Creating and Invoking a Subprogram with a Record Parameter

→ Code example in “queries.sql”, lines 470-533

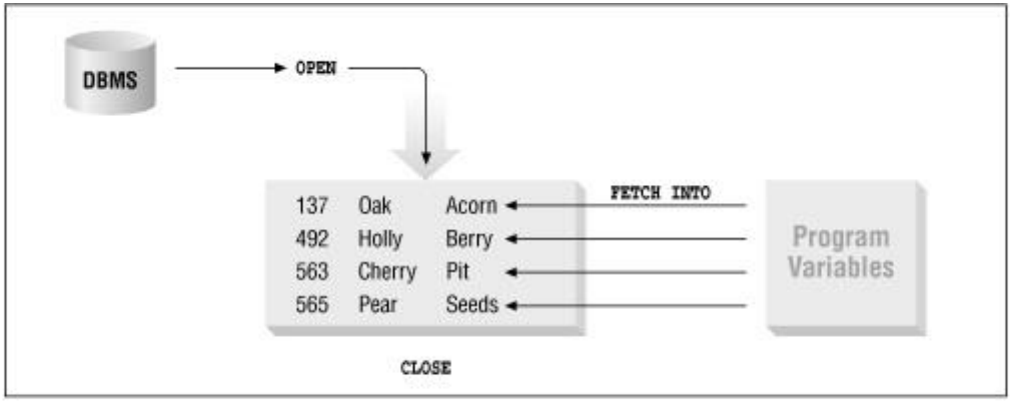
5.5 Using Records and Cursors

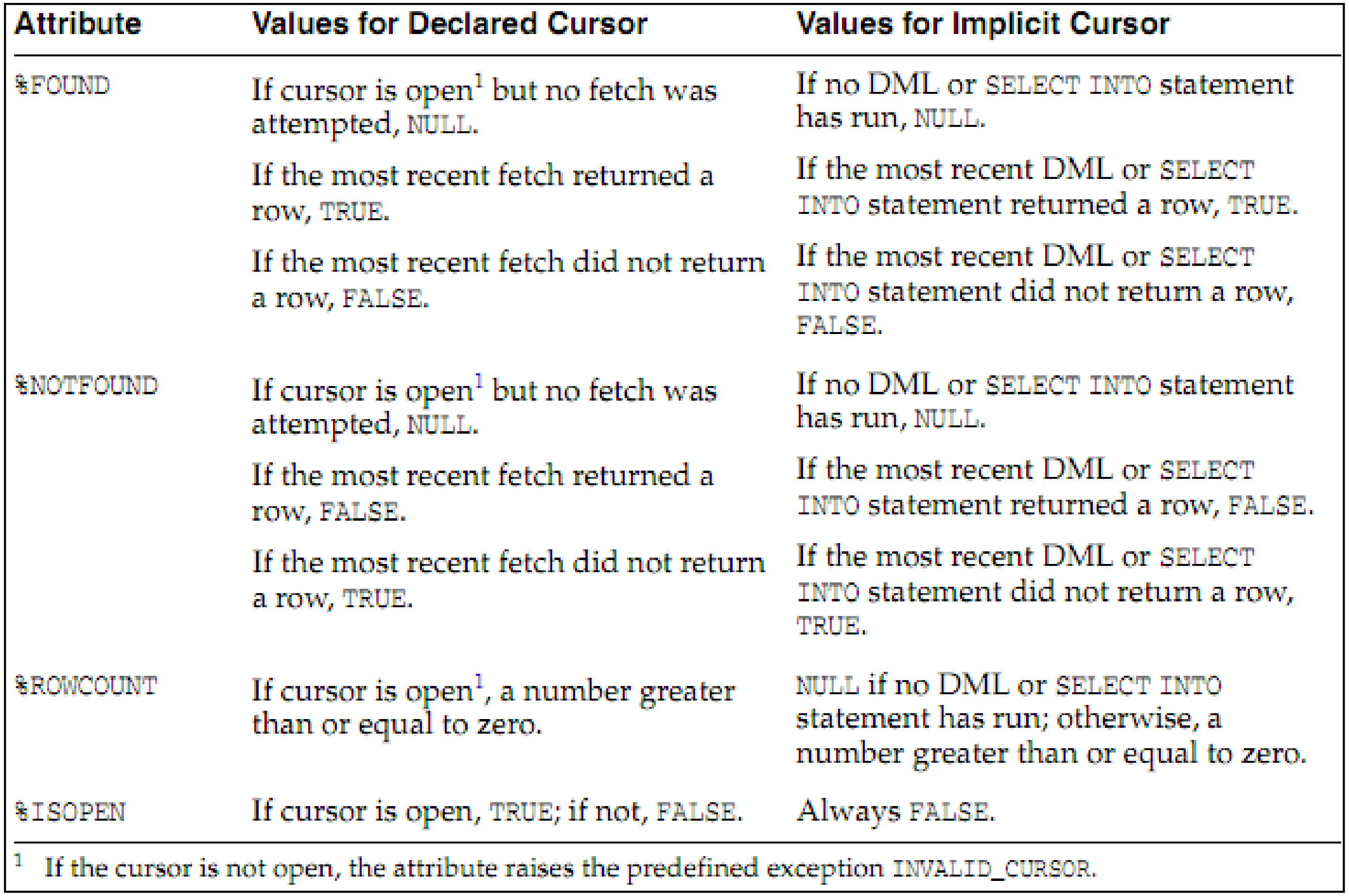
- When Oracle Database executes a SQL statement, it stores the result set and processing information in an unnamed private SQL area.
- A pointer to this unnamed area, called a cursor, lets you retrieve the result set one row at a time.
  - Cursor attributes return information about the state of the cursor.
  - Every time you run either a SQL DML statement or a PL/SQL SELECT INTO statement, PL/SQL opens an implicit cursor, but you cannot control it.
  - PL/SQL also lets you declare cursors. A declared cursor has a name and is associated with a query (SQL SELECT statement) - usually one that returns multiple rows.
  - After declaring a cursor, you must process it, either implicitly or explicitly.
- To process the cursor implicitly, use a cursor FOR LOOP. The syntax is:



```
FOR record_name IN cursor_name LOOP
  statement
 [ statement ]...
END LOOP;
```

- Using a Declared Cursor to Retrieve Result Set Rows One at a Time
  → Code example in “queries.sql”, lines 535-570



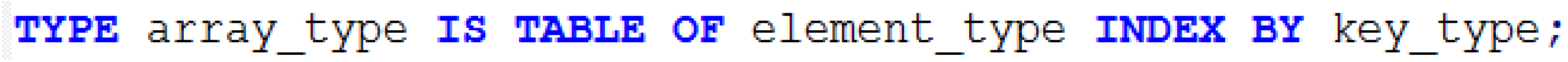


Attribute	Values for Declared Cursor	Values for Implicit Cursor
%FOUND	If cursor is open[1] but no fetch was attempted, NULL.	If no DML or SELECT INTO statement has run, NULL.
	If the most recent fetch returned a row, TRUE.	If the most recent DML or SELECT INTO statement returned a row, TRUE.
	If the most recent fetch did not return a row, FALSE.	If the most recent DML or SELECT INTO statement did not return a row, FALSE.
%NOTFOUND	If cursor is open[1] but no fetch was attempted, NULL.	If no DML or SELECT INTO statement has run, NULL.
	If the most recent fetch returned a row, FALSE.	If the most recent DML or SELECT INTO statement returned a row, FALSE.
	If the most recent fetch did not return a row, TRUE.	If the most recent DML or SELECT INTO statement did not return a row, TRUE.
%ROWCOUNT	If cursor is open[1], a number greater than or equal to zero.	NULL if no DML or SELECT INTO statement has run; otherwise, a number greater than or equal to zero.
%ISOPEN	If cursor is open, TRUE; if not, FALSE.	Always FALSE.

[1] If the cursor is not open, the attribute raises the predefined exception INVALID_CURSOR.

5.6 Using Associative Arrays

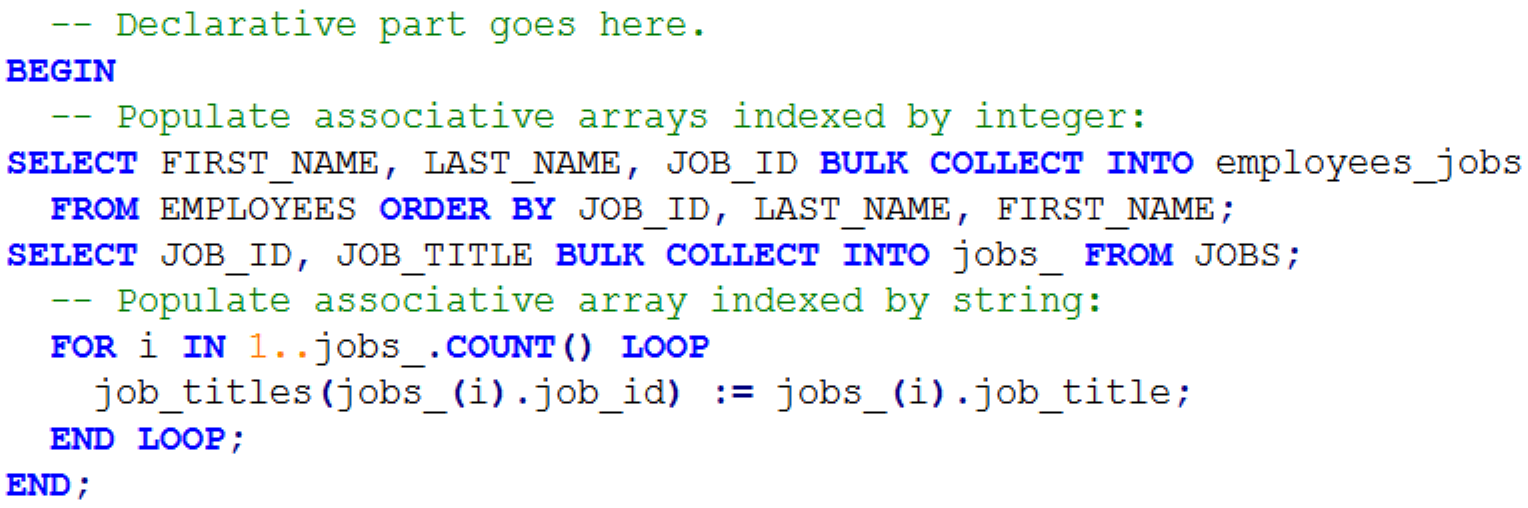
- An associative array is an unbounded set of key-value pairs. Each key is unique, and serves as the subscript of the element that holds the corresponding value. Therefore, you can access elements without knowing their positions in the array, and without traversing the array.
- The data type of the key can be either PLS_INTEGER or VARCHAR2 (length).
  - If the data type of the key is PLS_INTEGER and the associative array is indexed by integer and is dense (that is, has no gaps between elements), then every element between the first and last element is defined and has a value (which can be NULL).
  - If the key type is VARCHAR2 (length), the associative array is indexed by string (of length characters) and is sparse; that is, it might have gaps between elements.
- When traversing a dense associative array, you need not beware of gaps between elements; when traversing a sparse associative array, you do.



```
TYPE array_type IS TABLE OF element_type INDEX BY key_type;
```

5.6.1 Populating Associative Arrays

- The most efficient way to populate a dense associative array is usually with a SELECT statement with a BULK COLLECT INTO clause.



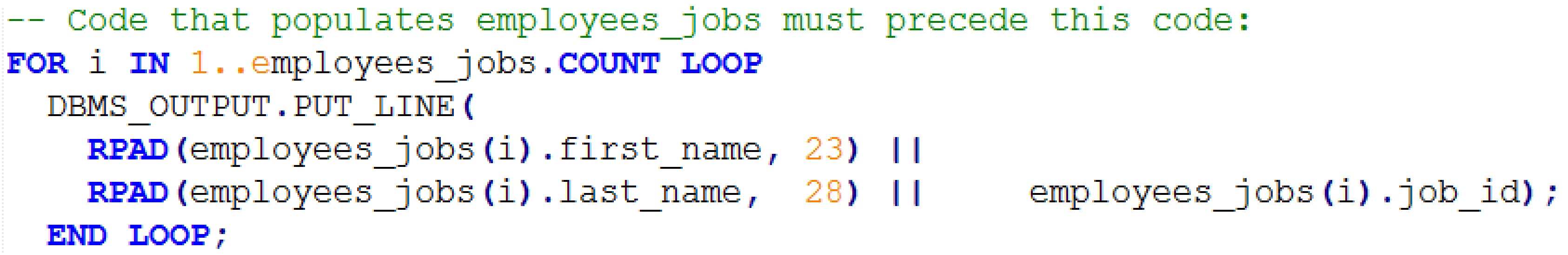
```
  -- Declarative part goes here.
BEGIN
  -- Populate associative arrays indexed by integer:
SELECT FIRST_NAME, LAST_NAME, JOB_ID BULK COLLECT INTO employees_jobs
  FROM EMPLOYEES ORDER BY JOB_ID, LAST_NAME, FIRST_NAME;
SELECT JOB_ID, JOB_TITLE BULK COLLECT INTO jobs_ FROM JOBS;
  -- Populate associative array indexed by string:
  FOR i IN 1..jobs_.COUNT() LOOP
    job_titles(jobs_(i).job_id) := jobs_(i).job_title;
  END LOOP;
END;
```

→ Code example in “queries.sql”, lines 572-607

- *Note: If a dense associative array is so large that a SELECT statement would a return a result set too large to fit in memory, then do not use a SELECT statement. Instead, populate the array with a cursor and the FETCH statement with the clauses BULK COLLECT INTO and LIMIT.*
- *For information about using the FETCH statement with BULK COLLECT clause, see Oracle Database PL/SQL Language Reference.*

5.6.2 Traversing Dense Associative Arrays

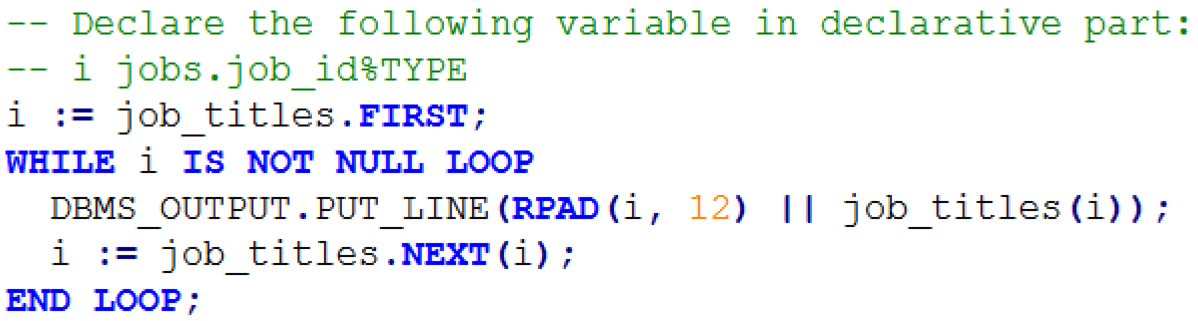
- A dense associative array (indexed by integer) has no gaps between elements—every element between the first and last element is defined and has a value (which can be NULL).



```
-- Code that populates employees_jobs must precede this code:
FOR i IN 1..employees_jobs.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE(
    RPAD(employees_jobs(i).first_name, 23) ||
    RPAD(employees_jobs(i).last_name,  28) ||     employees_jobs(i).job_id);
  END LOOP;
```

5.6.3 Traversing Sparse Associative Arrays

- A sparse associative array (indexed by string) might have gaps between elements.



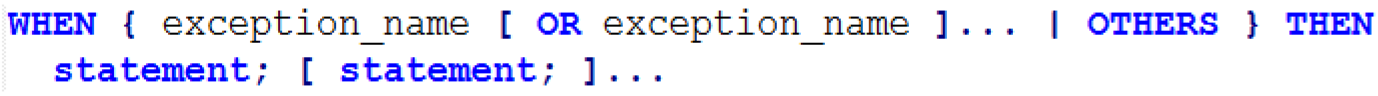
```
-- Declare the following variable in declarative part:
-- i jobs.job_id%TYPE
i := job_titles.FIRST;
WHILE i IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE(RPAD(i, 12) || job_titles(i));
  i := job_titles.NEXT(i);
END LOOP;
```

5.7 Handling Exceptions (Runtime Errors)

- When a runtime error occurs in PL/SQL code, an exception is raised. If the subprogram (or block) in which the exception is raised has an exception-handling part then control transfers to it; otherwise, execution stops.
- Runtime errors can arise from design faults, coding mistakes, hardware failures, and many other sources.
- Oracle Database has many predefined exceptions, which it raises automatically when a program violates database rules or exceeds system-dependent limits.
- Unlike a predefined exception, a user-defined exception must be raised explicitly, using either the RAISE statement or the DBMS_STANDARD.RAISE_APPLICATION_ERROR procedure.

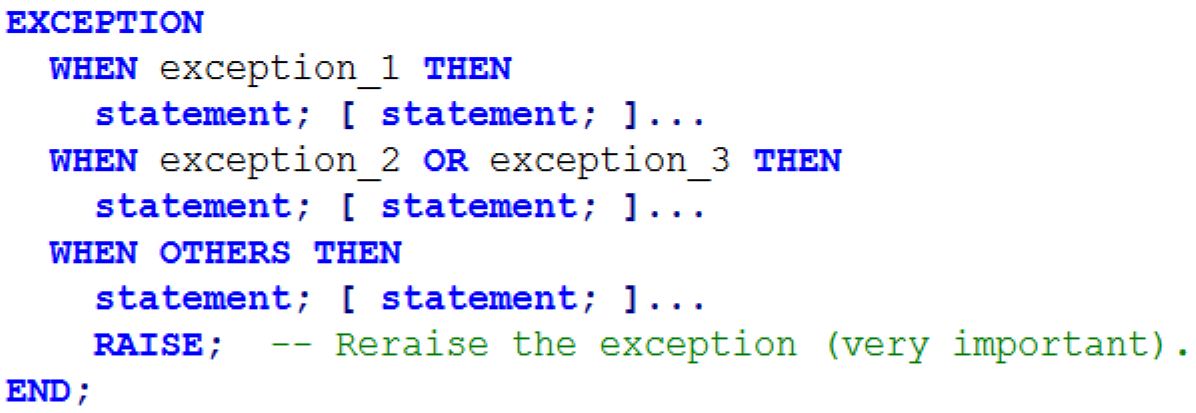
5.7 Handling Exceptions (Runtime Errors)

- The exception-handling part of a subprogram contains one or more exception handlers. An exception handler has this syntax:



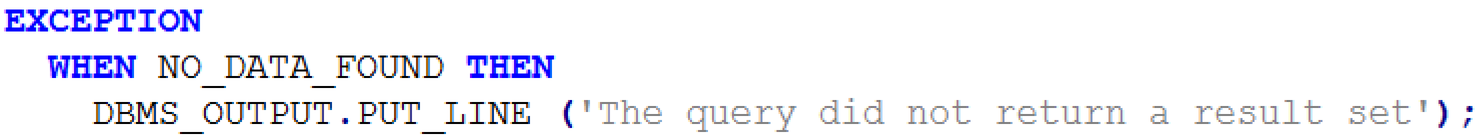
```
WHEN { exception_name [ OR exception_name ]... | OTHERS } THEN
  statement; [ statement; ]...
```

- A `WHEN OTHERS` exception handler handles unexpected runtime errors. If used, it must be last. For example:



```
EXCEPTION
  WHEN exception_1 THEN
    statement; [ statement; ]...
  WHEN exception_2 OR exception_3 THEN
    statement; [ statement; ]...
  WHEN OTHERS THEN
    statement; [ statement; ]...
    RAISE;  -- Reraise the exception (very important).
END;
```

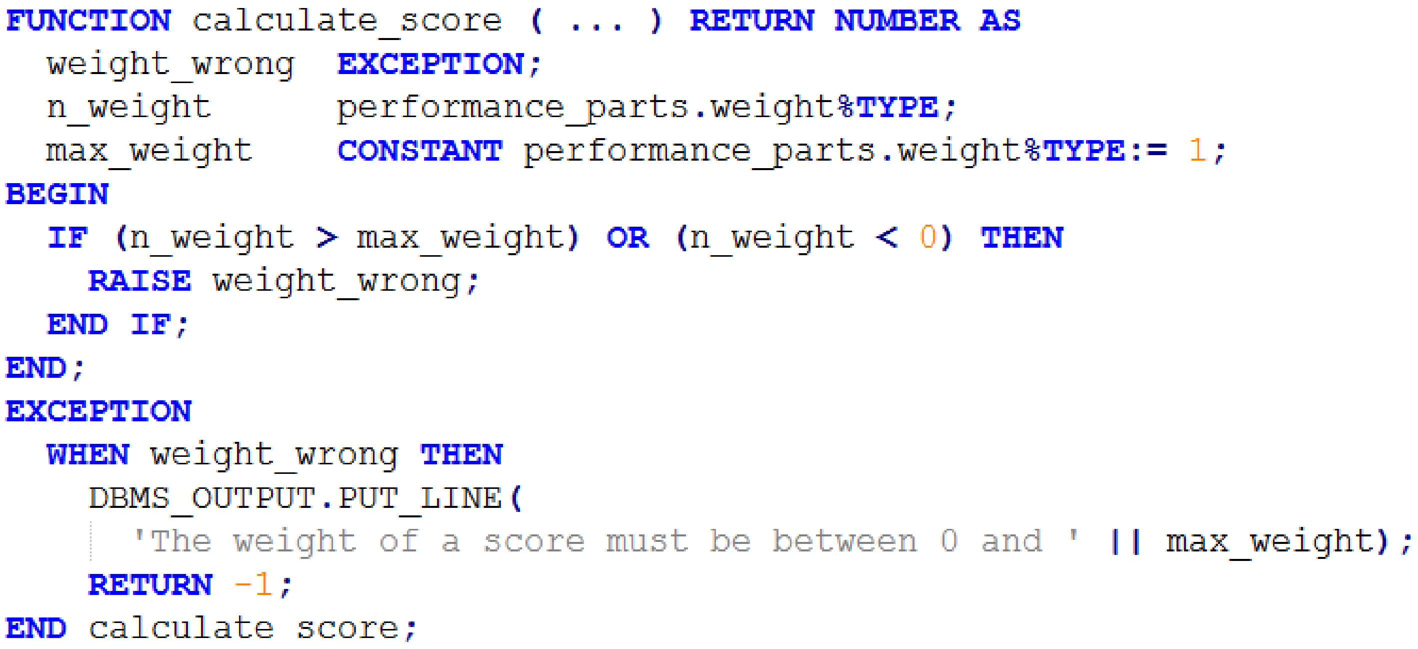
5.7.1 Handling Predefined Exceptions



```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('The query did not return a result set');
```

→ Code example in "queries.sql", lines 632-658

5.7.2 Declaring and Handling User-Defined Exceptions

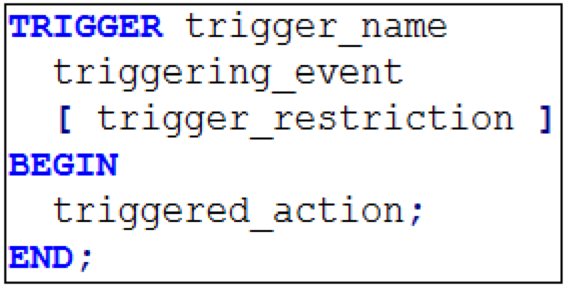


```
FUNCTION calculate_score ( ... ) RETURN NUMBER AS
  weight_wrong  EXCEPTION;
  n_weight      performance_parts.weight%TYPE;
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
BEGIN
  IF (n_weight > max_weight) OR (n_weight < 0) THEN
    RAISE weight_wrong;
  END IF;
END;
EXCEPTION
  WHEN weight_wrong THEN
    DBMS_OUTPUT.PUT_LINE(
      'The weight of a score must be between 0 and ' || max_weight);
    RETURN -1;
END calculate score;
```

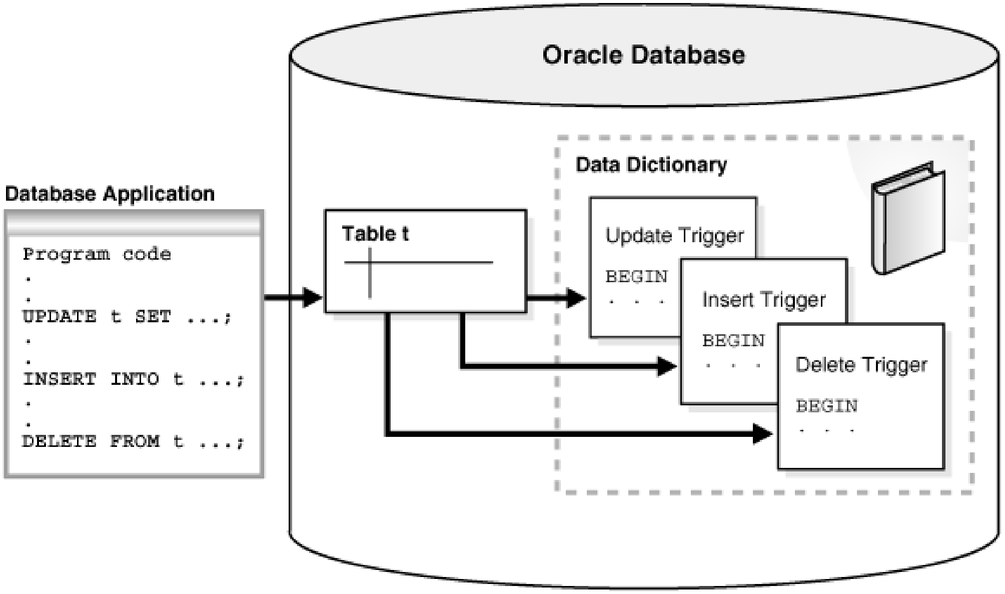
→ Code example in "queries.sql", lines 660-703

6. Using Triggers

- A trigger is a PL/SQL unit that is stored in the database and (if it is in the enabled state) automatically executes ("fires") in response to a specified event.



```
TRIGGER trigger_name
  triggering_event
   [ trigger_restriction ]
BEGIN
  triggered_action;
END;
```



```
Program code
.
.
UPDATE t SET ...;
.
INSERT INTO t ...;
.
.
DELETE FROM t ...;
```

6. Using Triggers

- A simple trigger can fire at exactly one of these timing points:
  - Before the triggering event executes (statement-level BEFORE trigger)
  - After the triggering event executes (statement-level AFTER trigger)
  - Before each row that the event affects (row-level BEFORE trigger)
  - After each row that the event affects (row-level AFTER trigger)
- A compound trigger can fire at multiple timing points.
- An INSTEAD OF trigger is defined on a view, and its triggering event is a DML statement. Instead of executing the DML statement, Oracle Database executes the INSTEAD OF trigger.
- A system trigger is defined on a schema or the database. A trigger defined on a schema fires for each event associated with the owner of the schema (the current user).
  - A trigger defined on a database fires for each event associated with all users.

6. Using Triggers (examples)

Example	Lines in "queries.sql"
Creating a Trigger that Logs Table Changes	705-729
Creating a Trigger that Generates a Primary Key for a Row Before It Is Inserted	731-742
Creating an INSTEAD OF Trigger	744-755
Creating Triggers that Log LOGON and LOGOFF Events	757-781
Disabling or Enabling a Single Trigger	783-788
Disabling or Enabling All Triggers on a Single Table	790-795
Dropping Triggers	797-801

7. Working in a Global Environment

- Globalization support features enable you to develop multilingual applications that can be run simultaneously from anywhere in the world. An application can render the content of the user interface, and process data, using the native language and locale preferences of the user.

- In the past, Oracle called globalization support National Language Support (NLS), but NLS is actually a subset of globalization support. NLS is the ability to choose a national language and store data using a specific character set. NLS is implemented with NLS parameters.

- → Oracle Database Globalization Support Guide

7. Working in a Global Environment

- Language Support
  - Oracle Database enables you to store, process, and retrieve data in native languages.
  - NLS_LANGUAGE (7-10) parameter

- Territory Support
  - Oracle Database supports cultural conventions that are specific to geographical locations.
  - NLS_TERRITORY (7-12) parameter

- Date and Time Formats
  - Some countries have different conventions for displaying the hour, day, month and year.
  - NLS_DATE_FORMAT (7-13), NLS_DATE_LANGUAGE (7-15), NLS_TIMESTAMP_FORMAT (7-17) and NLS_TIMESTAMP_TZ_FORMAT (7-17) parameters

- Calendar Formats
  - Oracle Database stores calendar information for each territory.
  - NLS_TERRITORY (7-12) and NLS_CALENDAR (7-17) parameters

7. Working in a Global Environment

- Numeric and Monetary Formats
  - Different countries have different numeric and monetary conventions.
  - NLS_NUMERIC_CHARACTERS (7-18), NLS_CURRENCY (7-20), NLS_ISO_CURRENCY (7-21) and NLS_DUAL_CURRENCY (7-22) parameters
- Linguistic Sorting and String Searching
  - Different languages have different sort orders (collating sequences).
  - Also, different countries or cultures that use the same alphabets sort words differently
  - NLS_SORT (7-22) and NLS_COMP (7-24) parameters
- Length Semantics
  - In single-byte character sets, the number of bytes and the number of characters in a string are the same. In multibyte charsets, a character consists of one or more bytes.
  - Column length in bytes = “byte semantics”; column length in chars = “char semantics”.
  - NLS_LENGTH_SEMANTICS (7-25) parameter
- Unicode and SQL National Character Data Types
  - The SQL national character data types are NCHAR, NVARCHAR2, and NCLOB. They are also called Unicode data types, because they are used only for storing Unicode data.

8. Building Effective Applications

8.1 Creating and Managing Standalone Subprograms

- A scalable application can process a larger workload with a proportional increase in system resource usage. For example, if you double its workload, a scalable application uses twice as many system resources.
- An unscalable application exhausts a system resource; therefore, if you increase the application workload, no more throughput is possible. Unscalable applications result in fixed throughputs and poor response times.

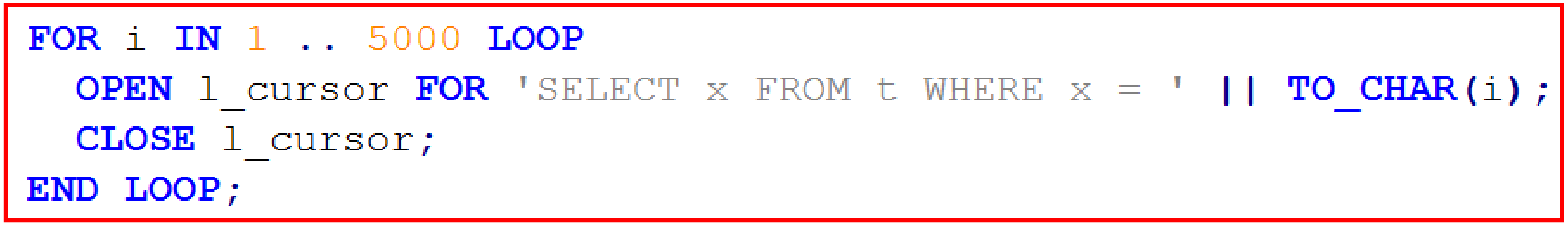
8.1 Creating and Managing Standalone Subprograms

8.1.1 Using Bind Arguments to Improve Scalability

- Bind arguments, used correctly, let you develop efficient, scalable applications.
- Just as a subprogram can have parameters, whose values are supplied by the invoker, a SQL statement can have bind argument placeholders, whose values (called bind arguments) are supplied at runtime.
- A hard parse, which includes optimization and row source generation, is a very CPU-intensive operation.
- A soft parse, which skips optimization and row source generation and proceeds straight to execution, is usually much faster than a hard parse of the same statement.

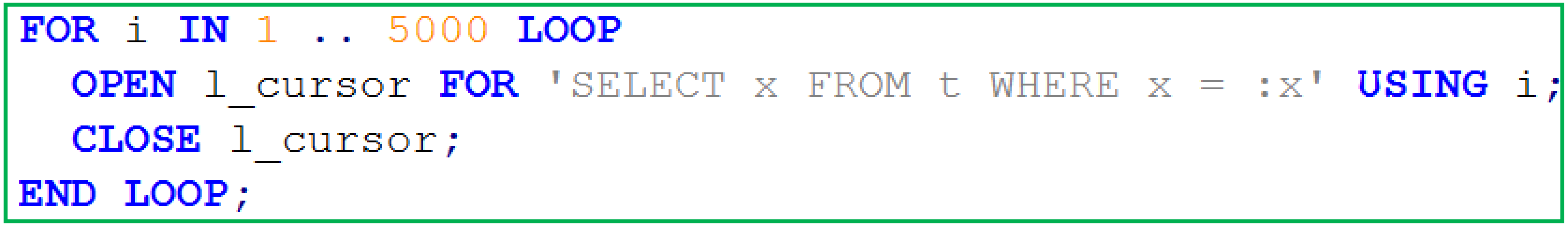
8.1 Creating and Managing Standalone Subprograms

8.1.1 Using Bind Arguments to Improve Scalability



```
FOR i IN 1 .. 5000 LOOP
  OPEN l_cursor FOR 'SELECT x FROM t WHERE x = ' || TO_CHAR(i);
  CLOSE l_cursor;
END LOOP;
```

Hard parse → took 740 hsec



```
FOR i IN 1 .. 5000 LOOP
  OPEN l_cursor FOR 'SELECT x FROM t WHERE x = :x' USING i;
  CLOSE l_cursor;
END LOOP;
```

Soft parse → took 30 hsec (almost 25x faster)

8.1 Creating and Managing Standalone Subprograms

8.1.2 Using PL/SQL to Improve Scalability

- How PL/SQL Minimizes Parsing
  - PL/SQL, which is optimized for database access, silently caches statements.
  - When you close a cursor, the cursor closes from your perspective, but PL/SQL actually keeps the cursor open and caches its statement.
  - If you use the cached statement again, PL/SQL uses the same cursor, thereby avoiding a parse.

- About the EXECUTE IMMEDIATE Statement
  - `EXECUTE IMMEDIATE sql_statement`
  - If sql_statement has the same value every time the EXECUTE IMMEDIATE statement runs, then PL/SQL can cache the EXECUTE IMMEDIATE statement.
  - If sql_statement can be different every time the EXECUTE IMMEDIATE statement runs, then PL/SQL cannot cache the EXECUTE IMMEDIATE statement.

8.1 Creating and Managing Standalone Subprograms

8.1.2 Using PL/SQL to Improve Scalability

- About OPEN FOR Statements
  - `OPEN cursor_variable FOR query`
  - Your application can open cursor_variable for several different queries before closing it. Because PL/SQL cannot determine the number of different queries until runtime, PL/SQL cannot cache the OPEN FOR statement.
  - If you do not need to use a cursor variable, then use a declared cursor, for both better performance and ease of programming.

- About the DBMS_SQL Package
  - The DBMS_SQL package is an API for building, running, and describing dynamic SQL statements. Using the DBMS_SQL package takes more effort than using the EXECUTE IMMEDIATE statement, but you must use the DBMS_SQL package if the PL/SQL compiler cannot determine at compile time the number or types of output host variables (select list items) or input bind variables.

8.1 Creating and Managing Standalone Subprograms

8.1.2 Using PL/SQL to Improve Scalability

- About Bulk SQL
  - Bulk SQL reduces the number of "round trips" between PL/SQL and SQL, thereby using fewer resources.
  - With bulk SQL, you retrieve a set of rows from the database, process the set of rows, and then return the whole set to the database.
  - Code example in “queries.sql”, lines 803-833
  - Without bulk SQL, you retrieve one row at a time from the database (SQL), process it (PL/SQL), and return it to the database (SQL).
  - Code example in “queries.sql”, lines 835-848
  - Using bulk SQL showed a reduction of almost 50% CPU time

8.1 Creating and Managing Standalone Subprograms

8.1.3 About Concurrency and Scalability

- The better your application handles concurrency, the more scalable it is.
- Concurrency is the simultaneous execution of multiple transactions. Statements within concurrent transactions can update the same data. Concurrent transactions must produce meaningful and consistent results. Therefore, a multiuser database must provide the following:
  - Data concurrency , which ensures that users can access data at the same time.
  - Data consistency, which ensures that each user sees a consistent view of the data, including visible changes from his or her own transactions and committed transactions of other users.

8.1 Creating and Managing Standalone Subprograms

8.1.3 About Concurrency and Scalability

- About Sequences and Concurrency
  - A sequence is a schema object from which multiple users can generate unique integers, which is very useful when you need unique primary keys.
  - A user gets a new primary key value by selecting the most recently produced value and incrementing it. This technique requires a lock during the transaction and causes multiple users to wait for the next primary key value—that is, the transactions serialize. Sequences eliminate serialization, thereby improving the concurrency and scalability of your application.

- About Latches and Concurrency
  - A latch is a simple, low-level serialization mechanism that coordinates multiuser access to shared data structures. Latches protect shared memory resources from corruption when accessed by multiple processes.
  - An increase in latches means more concurrency-based waits, and therefore a decrease in scalability. If you can use either an approach that runs slightly faster during development or one that uses fewer latches, use the latter.

8.2 Recommended Programming Practices

- Use Instrumentation Packages
  - Oracle Database supplies instrumentation packages whose subprograms let your application generate trace information whenever necessary. Using this trace information, you can debug your application without a debugger and identify code that performs badly. Instrumentation provides your application with considerable functionality; therefore, it is not overhead.
    - DBMS_APPLICATION_INFO, which enables a system administrator to track the performance of your application by module.
    - DBMS_SESSION, which enables your application to access session information and set preferences and security levels
    - UTL_FILE, which enables your application to read and write operating system text files

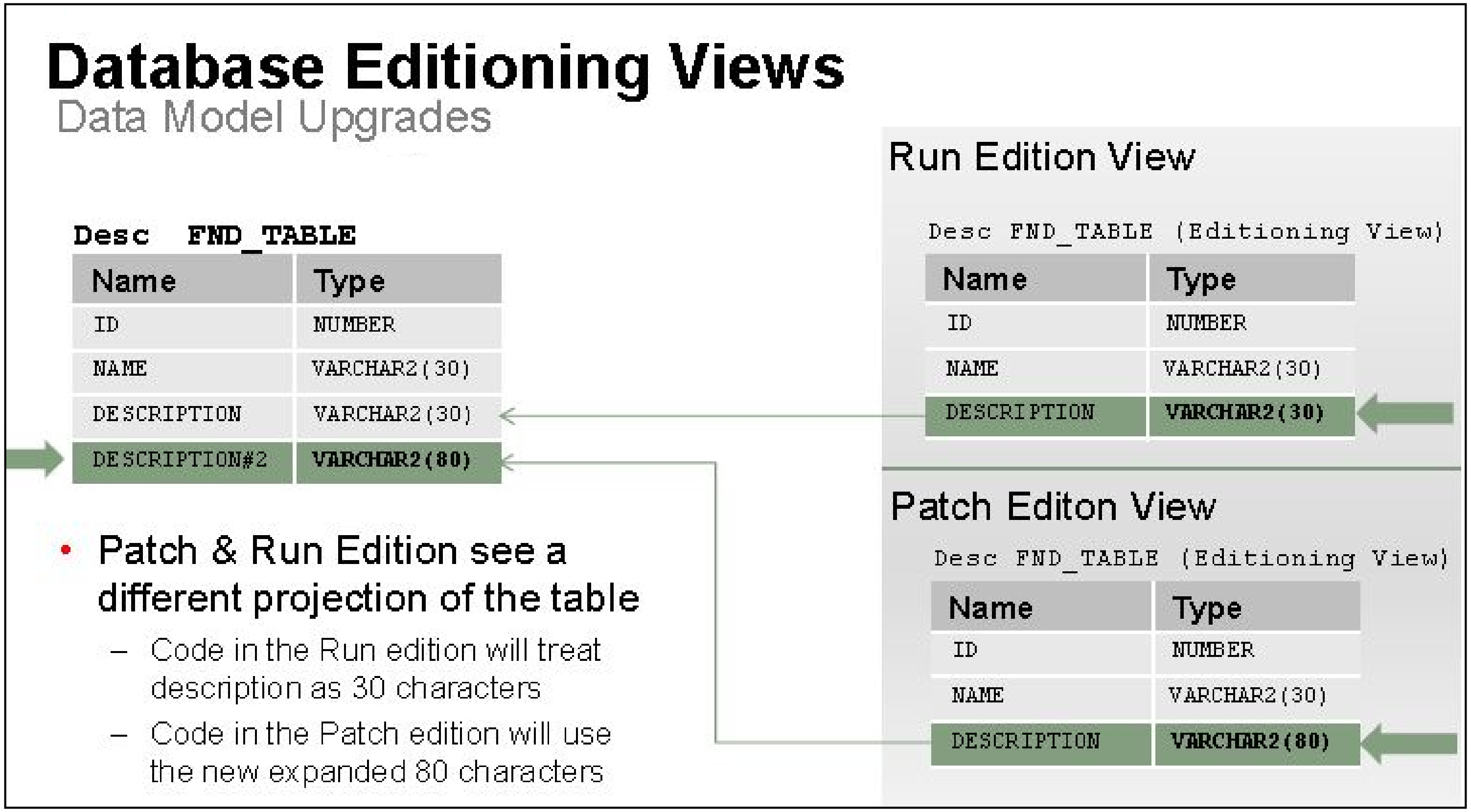
- Statistics Gathering and Application Tracing
  - Database statistics provide information about the type of load on the database and the internal and external resources used by the database.
  - To accurately diagnose performance problems with the database using ADDM (Automatic Database Diagnostic Monitor), statistics must be available.
  - If Oracle Enterprise Manager is unavailable, then you can gather statistics using DBMS_MONITOR subprograms.

8.2 Recommended Programming Practices

- Use Existing Functionality
  - When developing your application, use the existing functionality of your programming language, your operating system, Oracle Database, and the PL/SQL packages and types that Oracle Database supplies as much as possible.
  - An application that uses existing functionality is easier to develop and maintain than one that does not, and it also runs faster.
  - Examples of existing functionality that many developers reinvent are:
    - Constraints
    - SQL functions (functions that are "built into" SQL)
    - Sequences (which can generate unique sequential values)
    - Auditing (the monitoring and recording of selected user database actions)
    - Replication (the process of copying and maintaining database objects, such as tables, in multiple databases that comprise a distributed database system)
    - Message queuing (communication between web-based business applications)
    - Maintaining a history of record changes

8.2 Recommended Programming Practices

- Cover Database Tables with Editioning Views



Database Editioning Views

Data Model Upgrades

```
Desc  FND_TABLE
```

Name	Type
ID	NUMBER
NAME	VARCHAR2(30)
DESCRIPTION	VARCHAR2(30)
DESCRIPTION#2	VARCHAR2(80)

- Patch & Run Edition see a different projection of the table
  - Code in the Run edition will treat description as 30 characters
  - Code in the Patch edition will use the new expanded 80 characters

Run Edition View

```
Desc FND_TABLE (Editioning View)
```

Name	Type
ID	NUMBER
NAME	VARCHAR2(30)
DESCRIPTION	VARCHAR2(30)

Patch Editon View

```
Desc FND_TABLE (Editioning View)
```

Name	Type
ID	NUMBER
NAME	VARCHAR2(30)
DESCRIPTION	VARCHAR2(80)

- → Oracle Database Advanced Application Developer's Guide

8.3 Recommended Security Practices

- When granting privileges on the schema objects that comprise your application, use the **principle of least privilege**.

“Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.”

Jerome Saltzer, *Communications of the ACM*

- → Oracle Database 2 Day + Security Guide

9. Developing a Simple Oracle Database Application

- This chapter shows how to develop a simple Oracle Database application.

- About the Application
- Creating the Schemas for the Application
- Granting Privileges to the Schemas
- Creating the Schema Objects and Loading the Data
- Creating the employees_pkg Package
- Creating the admin_pkg Package

- → Oracle Database 2 Day Developer's Guide (197-224)

10. Deploying an Oracle Database Application

- Creating Installation Scripts with the Cart
  - The Cart is a convenient tool for deploying Oracle Database objects from one or more database connections to a destination connection. You drag and drop objects from the navigator frame into the Cart window, specify the desired options, and click the Export Cart icon to display the Export Objects dialog box. After you complete the information in that dialog box, SQL Developer creates a .zip file containing scripts (including a master script) to create the objects in the schema of a desired destination connection.

- Creating an Installation Script with the Database Export Wizard
  - To create an installation script in SQL Developer with the Database Export wizard, you specify the name of the installation script, the objects and data to export, and the desired options, and the wizard generates an installation script.

11. Reference Material

Oracle Database 2 Day Developer's Guide

https://docs.oracle.com/cd/E11882_01/appdev.112/e10766.pdf

Oracle Database PL/SQL Language Reference

https://docs.oracle.com/cd/E11882_01/appdev.112/e25519.pdf

Oracle Database Concepts, 11g Release 2

https://docs.oracle.com/cd/E11882_01/server.112/e40540.pdf

Oracle Database Globalization Support Guide

http://docs.oracle.com/database/121/NLSPG/E41669-06.pdf