Chapter 2

Using SQL in PL/SQL

Structured Query Language (SQL)

- SQL (Structured Query Language)
  - Pronounced as separate letters, "S"–"Q"–"L", not "sequel"
  - A programming language for selecting and manipulating sets of data in a relational database
  - A nonprocedural language
    - Focus is on input/output rather than on program steps
  - Standardized by the American National Standards Institute (ANSI)
    - Unfortunately, most vendors include some proprietary SQL features into their database environment

ANSI

- ANSI – American National Standards Institute
  - Structured Query Language (SQL) is the industry-standard language of relational database management systems (RDBMS)
  - Originally designed by IBM in the mid 1970s
  - Widespread use in the early 1980s
  - Became an industry standard in 1986 when it was adopted by ANSI
- Three ANSI standardizations of SQL
  - ANSI-86, ANSI-92, and ANSI-99

SQL Statements in PL/SQL

- Can use the following SQL statements in PL/SQL:
  - SELECT to retrieve data from the database
  - DML statements, such as INSERT, UPDATE, DELETE, and MERGE to make changes to the database
  - Transaction control statements, such as COMMIT, ROLLBACK, and SAVEPOINT to make changes to the database permanent or to discard them
    - Transaction control statements are covered later and are not available in the APEX environment

Limited Use of DDL and DCL Statements

- Cannot use DDL (Data Definition Language) and DCL (Data Control Language) directly in PL/SQL

Statement Type	Examples
DDL	CREATE TABLE, ALTER TABLE, DROP TABLE
DCL	GRANT, REVOKE

- DDL and DCL statements are constructed and executed at run time and are dynamic
  - Can use Dynamic SQL with the EXECUTE IMMEDIATE statement, which is discussed later

SELECT / INTO

- In PL/SQL, the INTO clause is mandatory
  - Occurs between the SELECT and FROM clauses

- INTO clause specifies the names of PL/SQL variables that hold the values that SQL returns from the SELECT clause

SELECT and INTO Example

- Must specify one PL/SQL variable for each column specified on the SELECT column-list
- The order of the variables must correspond with the order of the SELECT column-list

```
DECLARE
 v_emp_hiredate  employees.hire_date%TYPE;
 v_emp_salary    employees.salary%TYPE;
BEGIN
 SELECT   hire_date, salary
  INTO    v_emp_hiredate, v_emp_salary
  FROM    employees
  WHERE   employee_id = 100;
 DBMS_OUTPUT.PUT_LINE('Hiredate: ' || v_emp_hiredate);
 DBMS_OUTPUT.PUT_LINE('Salary: '|| v_emp_salary);
END;
```

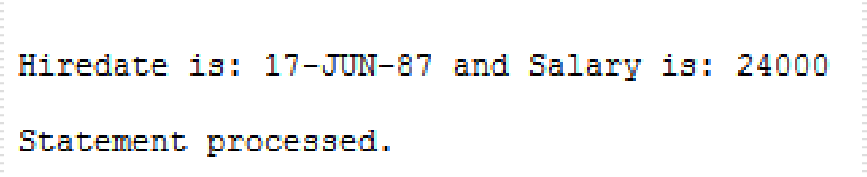
Retrieving Data in PL/SQL Embedded Rule

- ANSI classification of embedded SQL
  - Embedded queries must return exactly one row
  - A query that returns more than one row or no rows generates an error
- Usually uses the WHERE clause

Embedded Rule Example
1 Row Returned

- Retrieve hire_date and salary for the specified employee

```
DECLARE
  v_emp_hiredate   employees.hire_date%TYPE;
  v_emp_salary     employees.salary%TYPE;
BEGIN
  SELECT     hire_date, salary
    INTO     v_emp_hiredate, v_emp_salary
    FROM     employees
    WHERE    employee_id = 100;
  DBMS_OUTPUT.PUT_LINE('Hiredate is: ' || v_emp_hiredate
                       || ' and Salary is: '
                       || v_emp_salary);
END;
```



```
Hiredate is: 17-JUN-87 and Salary is: 24000

Statement processed.
```

Embedded Rule Example
No Rows Returned

- Retrieve hire_date and salary for employee 999

```
DECLARE
  v_emp_hiredate employees.hire_date%TYPE;
  v_emp_salary employees.salary%TYPE;
BEGIN
  SELECT hire_date, salary
    INTO v_emp_hiredate, v_emp_salary
    FROM employees
    WHERE employee_id = 999;
  DBMS_OUTPUT.PUT_LINE('Hiredate is: ' || v_emp_hiredate
                 || ' and Salary is: ' || v_emp_salary);
END;
```

```
ORA-01403: no data found
```

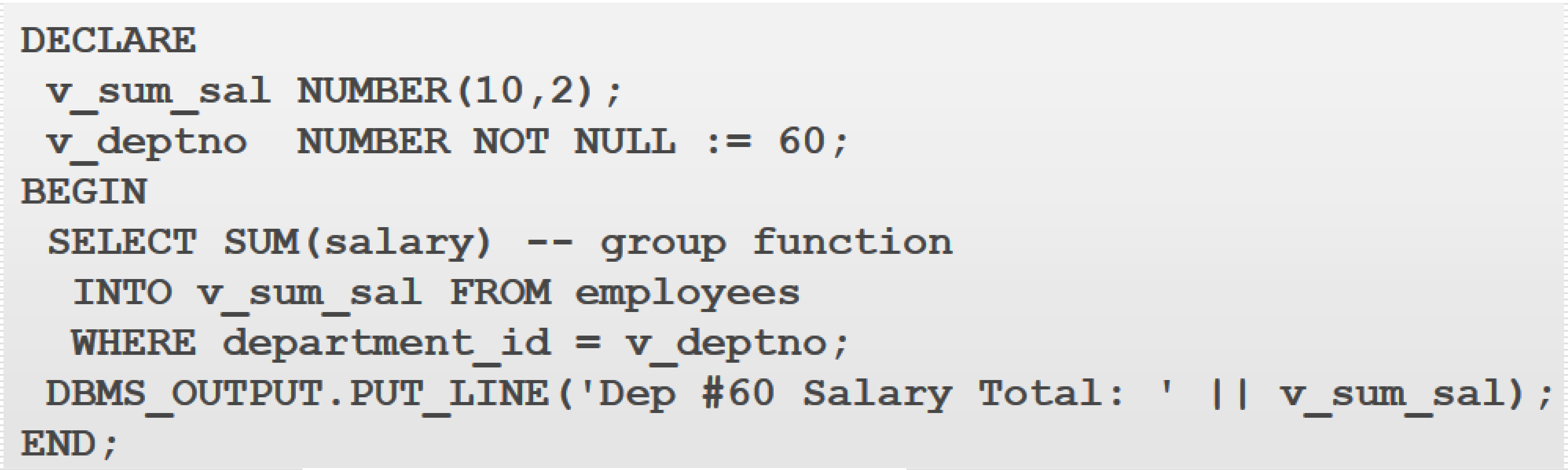
Embedded Rule Example
Multiple Rows Returned

```
DECLARE
  v_salary employees.salary%TYPE;
BEGIN
  SELECT salary INTO v_salary
    FROM employees;
DBMS_OUTPUT.PUT_LINE('Salary is : ' || v_salary);
END;
```

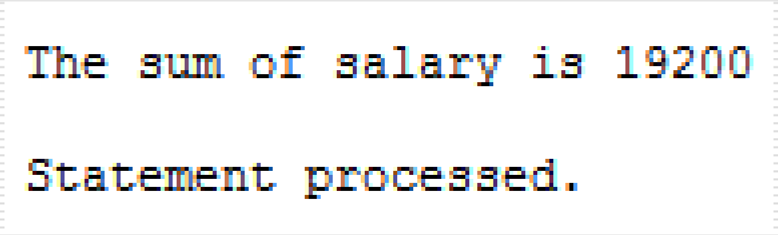
```
ORA-01422:  exact fetch returns more than requested number of rows
```

Returning Data in PL/SQL Group Function Example

- A Group function returns one row



```
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('Dep #60 Salary Total: ' || v_sum_sal);
END;
```



```
The sum of salary is 19200

Statement processed.
```

Guidelines for Retrieving Data in PL/SQL

- Terminate each SQL statement with a semicolon (;)
- Every value retrieved must be stored in a variable using the INTO clause
- Specify the same number of PL/SQL variables in the INTO clause as specified in the SELECT column-list clause
  - They must be in the same positional order
  - Their data types must are compatible

Guidelines for Retrieving Data in PL/SQL

- Fetch only one row and the usage of the WHERE clause is therefore needed in nearly all cases
- Declare the PL/SQL variables using %TYPE

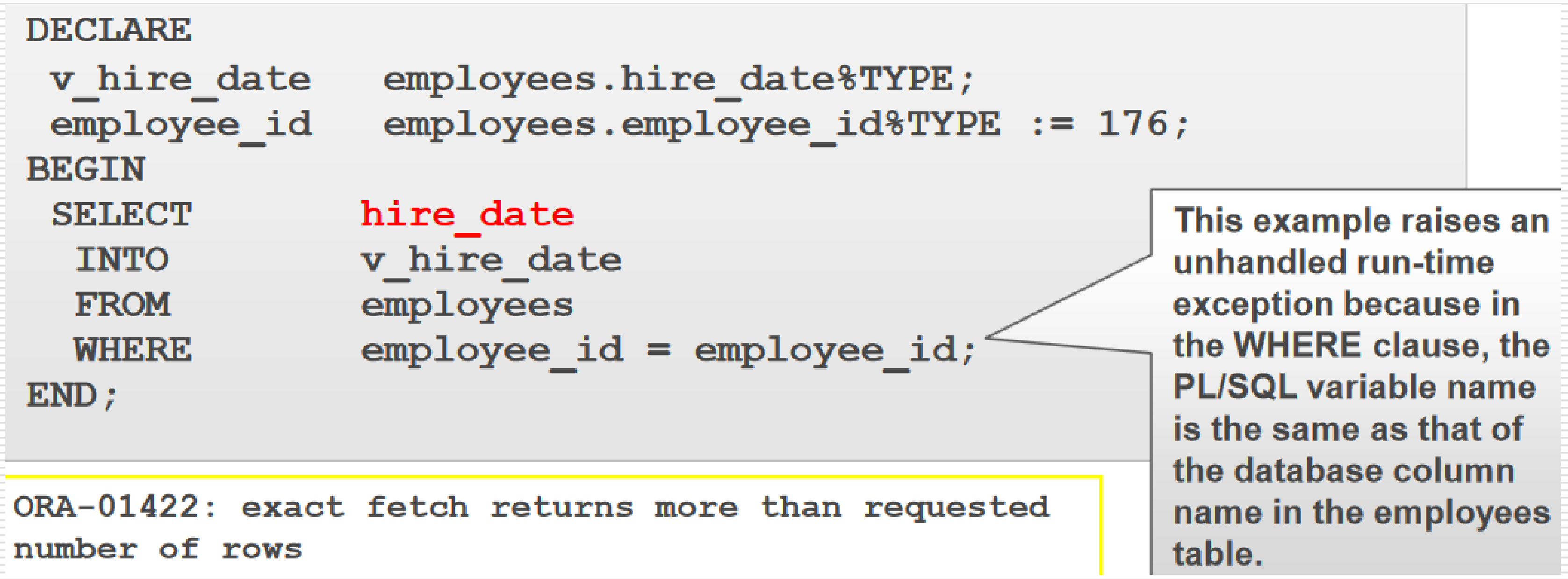
Guidelines for Naming Conventions-Lab

- What is returned from the SELECT statement?

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

Guidelines for Naming Conventions

- Avoid ambiguous PL/SQL variable names
  - The names of database columns take priority (rank higher) than names of local variables
  - Use v_ with variables, as in v_employee_id



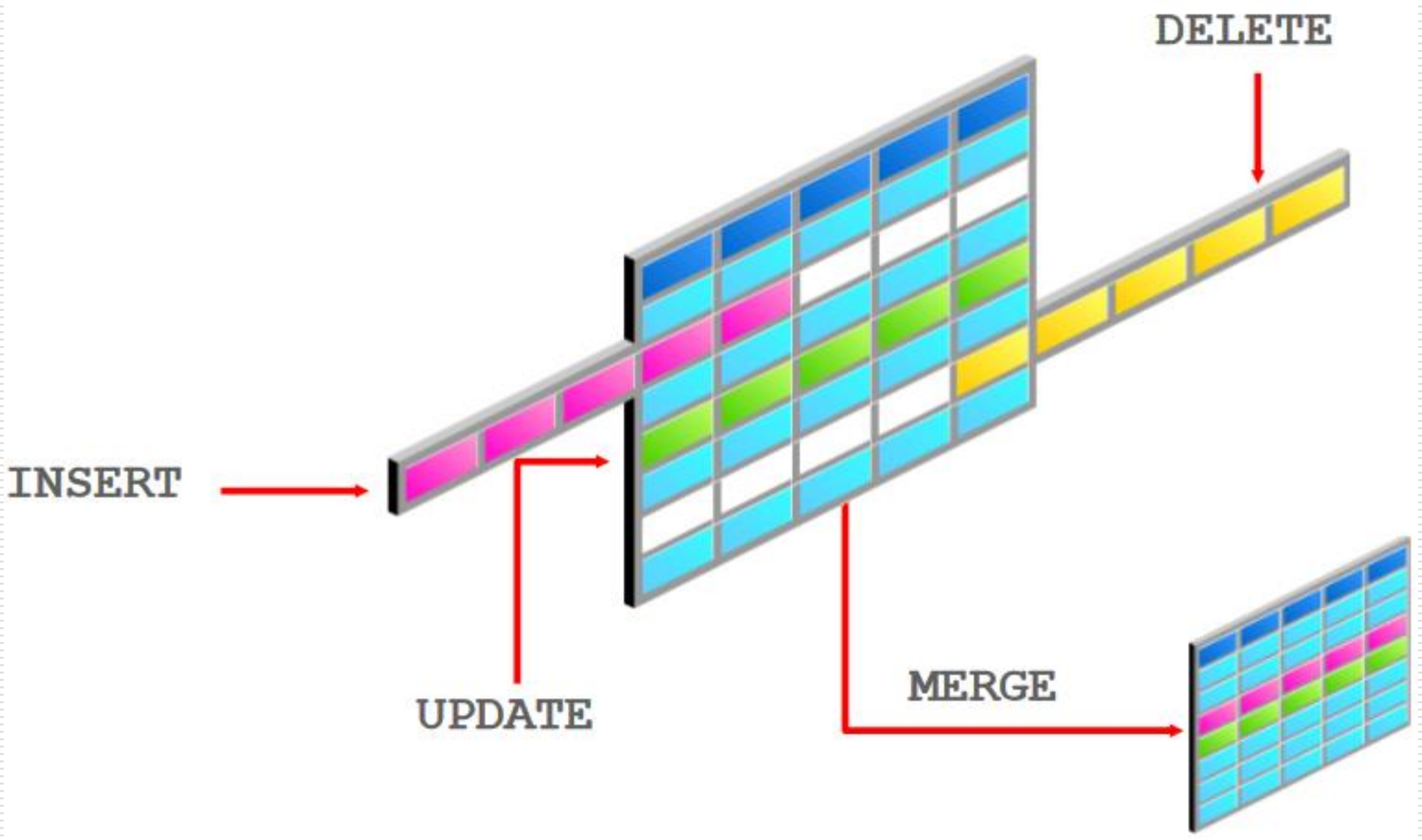
```
DECLARE
 v_hire_date    employees.hire_date%TYPE;
 employee_id    employees.employee_id%TYPE := 176;
BEGIN
 SELECT        hire_date
   INTO        v_hire_date
   FROM        employees
   WHERE       employee_id = employee_id;
END;
```

```
ORA-01422: exact fetch returns more than requested
number of rows
```

This example raises an unhandled run-time exception because in the WHERE clause, the PL/SQL variable name is the same as that of the database column name in the employees table.

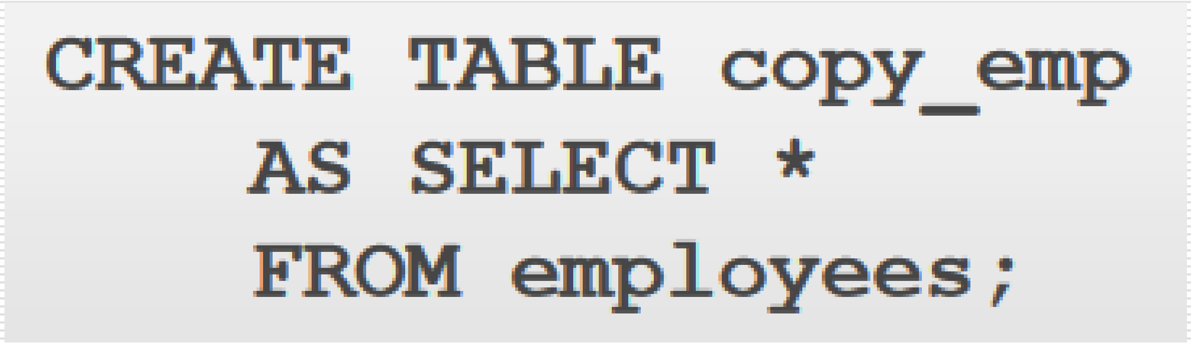
Manipulating Data in Pl/SQL

- Make changes to data by using DML commands within PL/SQL blocks:
  - INSERT
  - UPDATE
  - DELETE
  - MERGE



Create Copy of Original Table

- Do NOT modify existing tables (such as EMPLOYEES and DEPARTMENTS), because they will be needed later in the course
- Make copies of the original tables



```
CREATE TABLE copy_emp
    AS SELECT *
    FROM employees;
```

SELECT/INTO
Without User-Defined Records

- A lot of coding
- What if more columns were added to table

```
DECLARE
  v_employee_id   employees.employee_id%TYPE;
  v_first_name    employees.first_name%TYPE;
  ... -- seven more scalar variables here
  v_manager_id    employees.manager_id%TYPE;
  v_department_id employees.department_id%TYPE;
BEGIN
  SELECT employee_id, first_name, ..., department_id
    INTO v_employee_id, v_first_name, ..., v_department_id
    FROM employees
    WHERE employee_id = 100;
END;
```

Record Structure - %ROWTYPE

- %ROWTYPE:
  - Creates a record structure in memory to hold one row from a table or cursor
  - Field names are the same as the underlying column names
  - Refer to the whole record by its name
  - Reference individual fields by prefixing the field-name with the record-name

%ROWTYPE
User-Defined Record Structures

- Can create a record structure (data structure) as a single variable in memory
  - %ROWTYPE
    - Based on the structure of a table
  - TYPE
    - Based on your needs (different items)
    - Create record structure as a type (model or template) and then declare a variable of that type

%ROWTYPE

- Based on the structure of a table (or cursor)

```
DECLARE
  v_emp_record    employees%ROWTYPE;
BEGIN
  SELECT * INTO v_emp_record
    FROM employees
      WHERE employee_id = 100;
END;
```

```
... DBMS_OUTPUT.PUT_LINE(v_emp_record.salary);

... IF v_emp_record.department_id = 20 THEN ...
```

Inserting Data

- The INSERT statement adds new row(s) to a table

INSERT Explicit Syntax

- Must list each column in the table that can not be NULL
- Values for each column must be listed in the same order as the columns are listed

```
INSERT INTO employees (employee_id, first_name,
   last_name, email, hire_date, job_id)
  VALUES (305, 'Kareem', 'Naser',
   'naserk@oracle.com', SYSDATE, 'SR_SA_REP');
```

INSERT Implicit Syntax

- Column-names not listed
- The values must match the order in which the columns appear in the table and a value must be provided for each column

```
INSERT INTO employees
 VALUES (305, 'Kareem', 'Naser',
  'naserk@oracle.com', '111-222-3333', SYSDATE,
  'SR_SA_REP', 7000, NULL, NULL, NULL, NULL);
```

Updating Data

- The UPDATE statement modifies existing rows in a table

UPDATE – Modifying a Single Column

- A single column can be modified
- The WHERE clause identifies the row to be modified

```
UPDATE employees
  SET salary = 11000
  WHERE employee_id = 176;
```

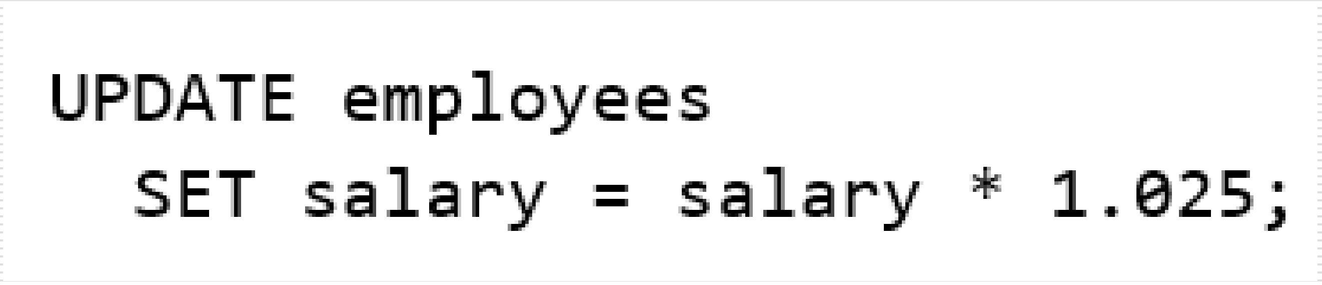
UPDATE – Modifying Multiple Columns

- Multiple columns can be modified
- The WHERE clause identifies the row to be modified

```
UPDATE employees
  SET salary = 11000, commission_pct = .3
  WHERE employee_id = 176;
```

UPDATE – Modify Multiple Rows

- Multiple rows can be updated with UPDATE
- The WHERE clause is optional depending on the subset to be updated



```
UPDATE employees
  SET salary = salary * 1.025;
```

Deleting Data

- The DELETE statement removes existing rows from a table
- If the WHERE clause is omitted, ALL rows are deleted
- In the following, how many rows are deleted?

```
DELETE FROM employees
 WHERE employee_id = 149;
```

```
DELETE FROM employees
 WHERE department_id = 80;
```

What is Deleted?

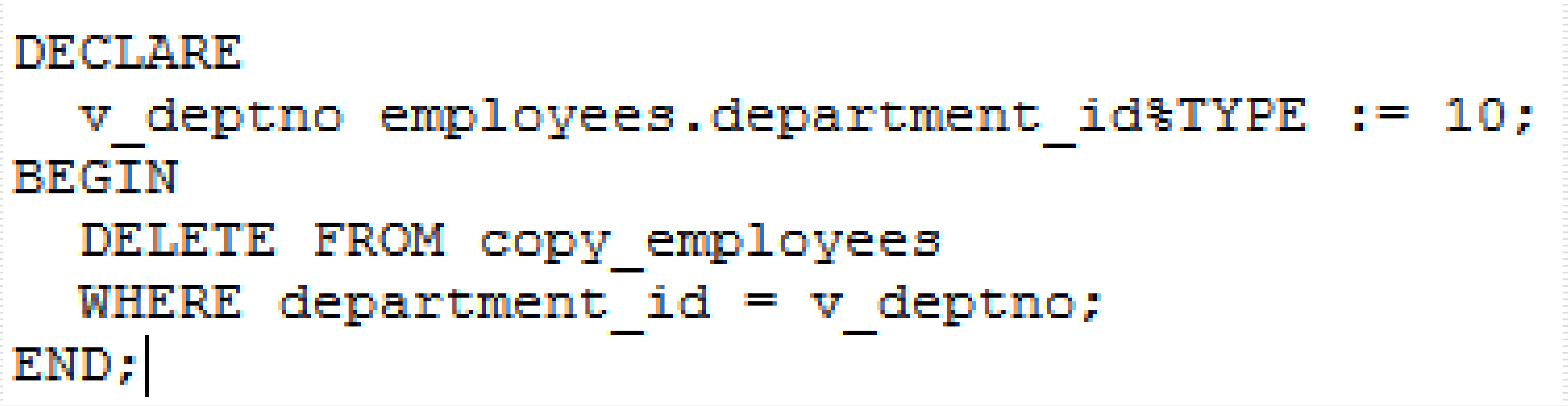
- The names of database columns take priority or rank higher than names of local variables
- Use v_ with variables

```
CREATE TABLE copy_employees AS
  SELECT * FROM employees;

DECLARE
  last_name  VARCHAR2(25) := 'King';
BEGIN
  DELETE FROM copy_employees
  WHERE last_name = last_name;
END;
```

```
SELECT * FROM copy_employees;

no data found
```

Getting Information From a Cursor

- It would be useful to know how many rows were deleted from the COPY_EMPLOYEES table
- Use cursors to obtain this information



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

Cursors

- Implicit cursors
  - Defined automatically by Oracle for all SQL DML statements and queries that return only one row

- Explicit cursors
  - Defined by the PL/SQL programmer for queries that return more than one row
  - More on these later

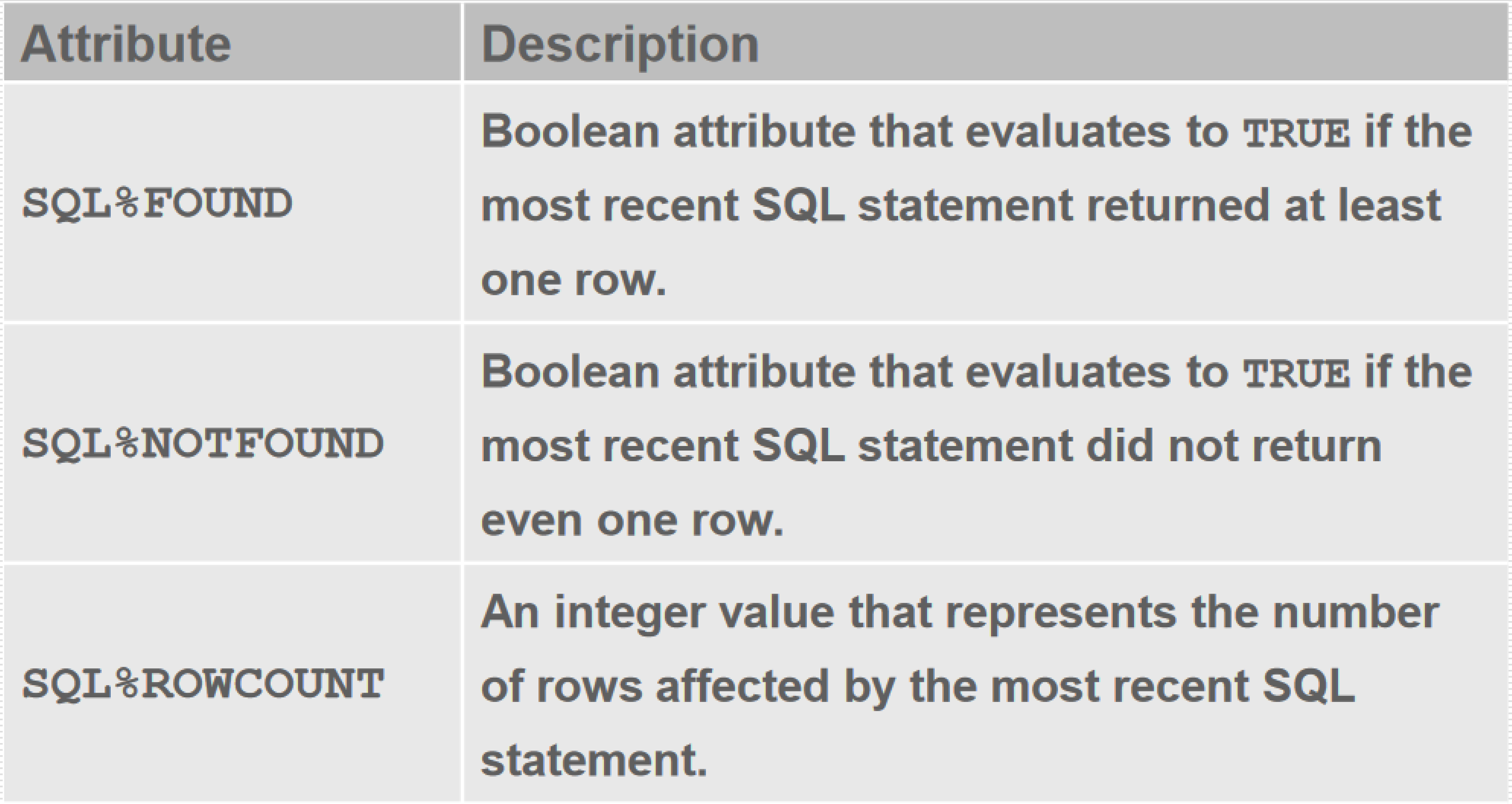
Implicit Cursor

- Every time an SQL statement is executed:
  - The Oracle server allocates a private memory area to store the SQL statement and the data that it uses
  - Memory area is called an implicit cursor
  - Developers have no direct control over it

Cursor Attributes for Implicit Cursors

- Cursor attributes:
  - Predefined PL/SQL variables
  - Automatically declared variables
  - Retrieves information about the last SQL statement (implicit cursor) executed
- Prefixed with "SQL "
- Used in PL/SQL statements, but not SQL statements

Cursor Attributes



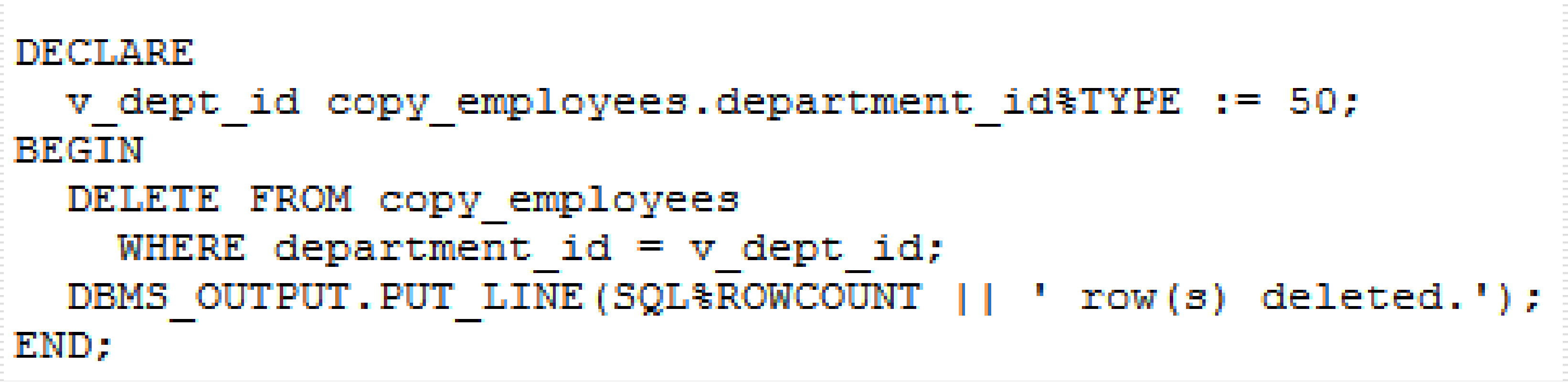
Attribute	Description
`SQL%FOUND`	Boolean attribute that evaluates to `TRUE` if the most recent SQL statement returned at least one row.
`SQL%NOTFOUND`	Boolean attribute that evaluates to `TRUE` if the most recent SQL statement did not return even one row.
`SQL%ROWCOUNT`	An integer value that represents the number of rows affected by the most recent SQL statement.

Implicit Cursors

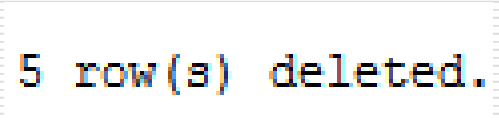
- The following use implicit cursors:
  - DML statements
    - INSERT
    - UPDATE
    - DELETE
    - MERGE
  - SELECT statements that return a single row

Using Implicit Cursor Attributes: Example 1

- Output the number of rows deleted



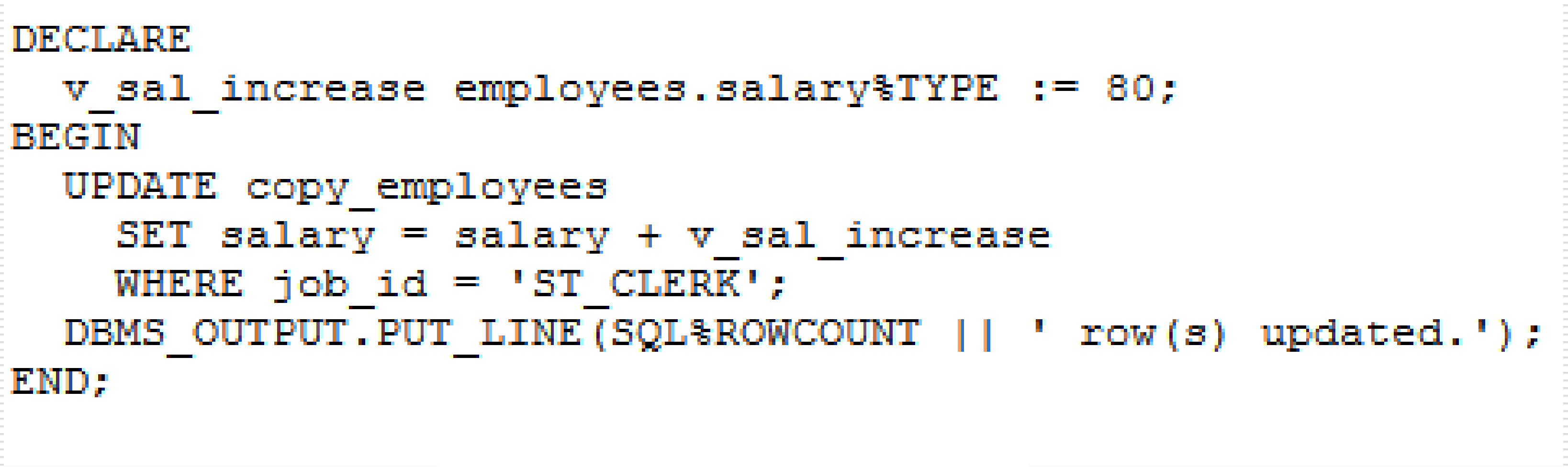
```
DECLARE
  v_dept_id copy_employees.department_id%TYPE := 50;
BEGIN
  DELETE FROM copy_employees
    WHERE department_id = v_dept_id;
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' row(s) deleted.');
END;
```



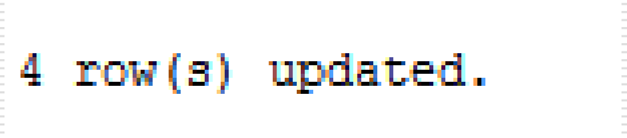
```
5 row(s) deleted.
```

Using Implicit Cursor Attributes: Example 2

- Use cursor attributes in PL/SQL statements, but not in SQL statements
- Output the number of rows updated



```
DECLARE
  v_sal_increase employees.salary%TYPE := 80;
BEGIN
  UPDATE copy_employees
    SET salary = salary + v_sal_increase
    WHERE job_id = 'ST_CLERK';
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' row(s) updated.');
END;
```



```
4 row(s) updated.
```

Using Implicit Cursor Attributes: Example 3

- Use cursor attributes in PL/SQL statements, but not in SQL statements
- The following is invalid:

```
CREATE TABLE results (num_rows NUMBER(4));

BEGIN
  UPDATE copy_employees
    SET salary = salary + 100
    WHERE job_id = 'SA_REP';
  INSERT INTO results (num_rows)
    VALUES (SQL%ROWCOUNT);
END;
```

Q & A
Questions & Answers