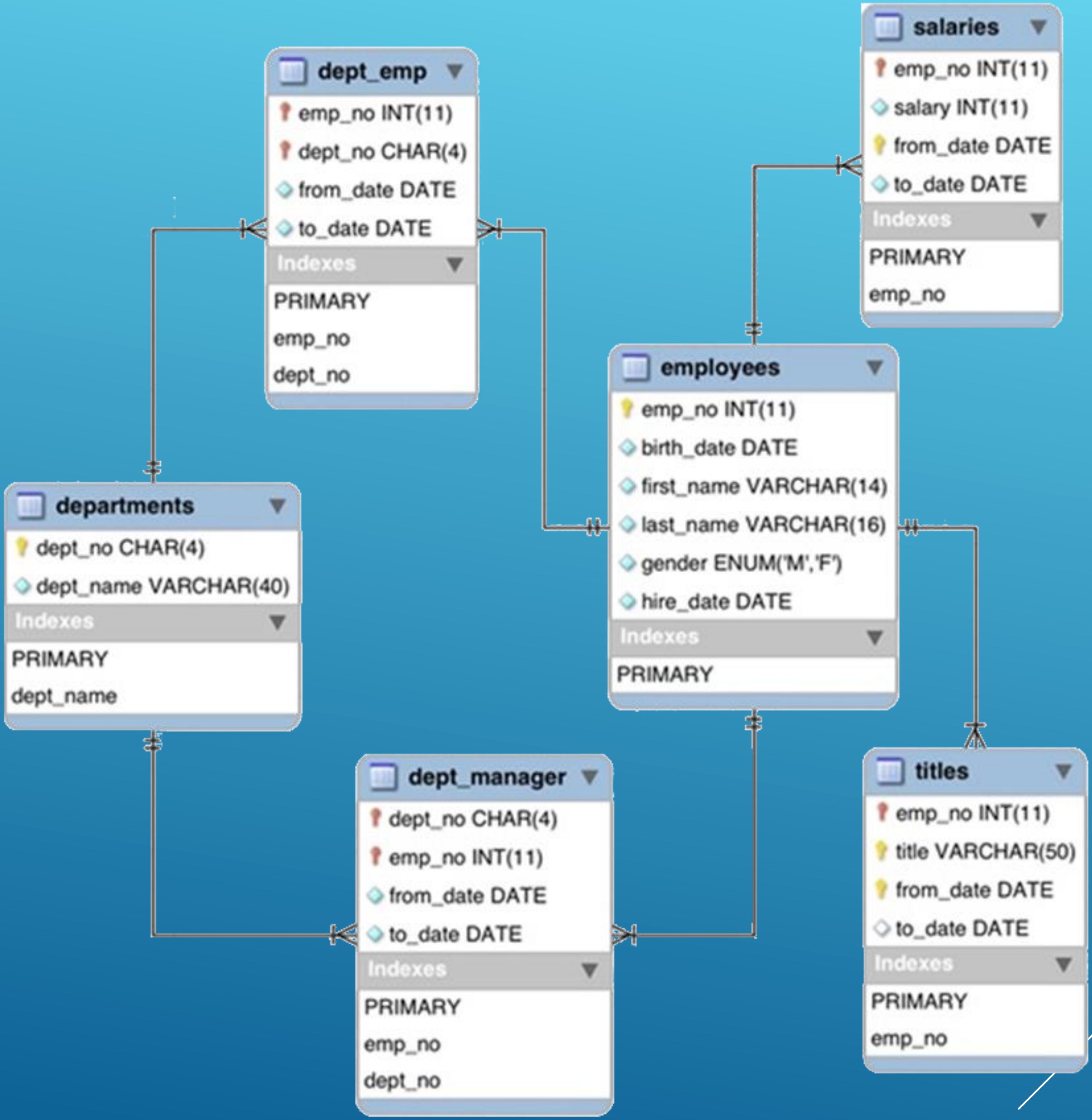
PL/SQL

420-983-VA Introduction to Databases using Oracle Adrian Onet

MODEL DATABASE

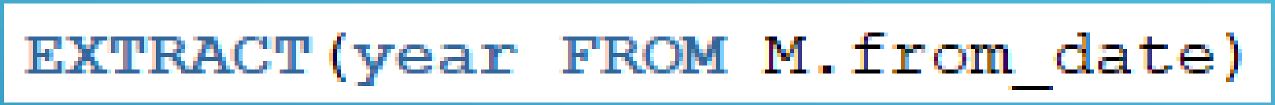


VIEWS

Return all department managers name from 1992.

VIEWS

EXTRACT FUNCTION.



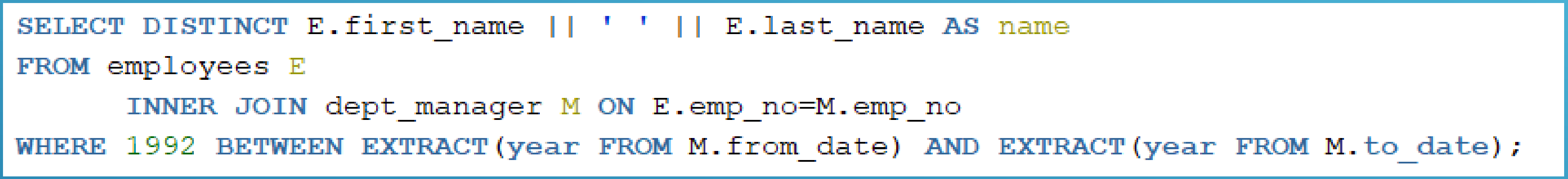
```
EXTRACT(year FROM M.from_date)
```

YEAR
MONTH
DAY
HOUR
MINUTE
SECOND
TIMEZONE_HOUR
TIMEZONE_MINUTE
TIMEZONE_REGION
TIMEZONE_ABBR

TIMESTAMP WITH TIME ZONE datatype

VIEWS

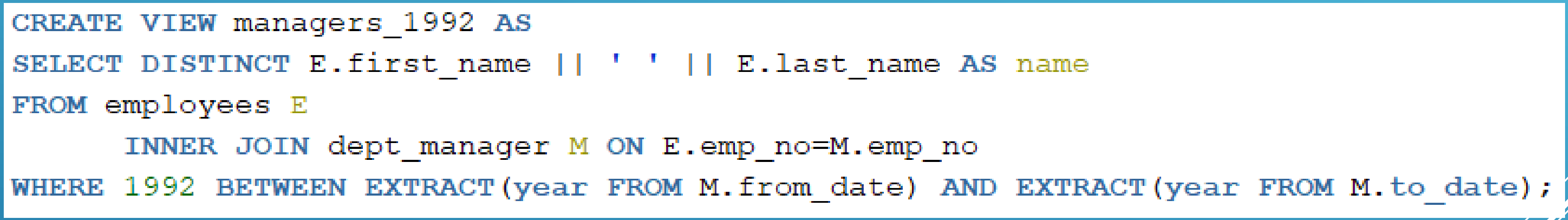
Return all department managers name from 1992.



```
SELECT DISTINCT E.first_name || ' ' || E.last_name AS name
FROM employees E
      INNER JOIN dept_manager M ON E.emp_no=M.emp_no
WHERE 1992 BETWEEN EXTRACT(year FROM M.from_date) AND EXTRACT(year FROM M.to_date);
```

VIEWS

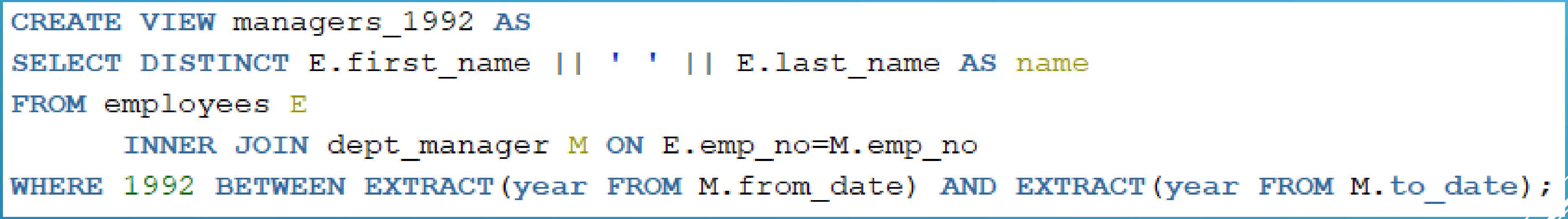
Creating a view with all department managers name from 1992.



```
CREATE VIEW managers_1992 AS
SELECT DISTINCT E.first_name || ' ' || E.last_name AS name
FROM employees E
      INNER JOIN dept_manager M ON E.emp_no=M.emp_no
WHERE 1992 BETWEEN EXTRACT(year FROM M.from_date) AND EXTRACT(year FROM M.to_date);
```

VIEWS

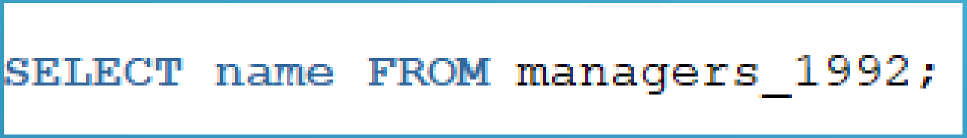
Creating a view with all department managers name from 1992.



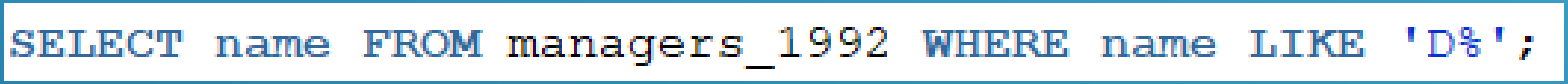
```
CREATE VIEW managers_1992 AS
SELECT DISTINCT E.first_name || ' ' || E.last_name AS name
FROM employees E
      INNER JOIN dept_manager M ON E.emp_no=M.emp_no
WHERE 1992 BETWEEN EXTRACT(year FROM M.from_date) AND EXTRACT(year FROM M.to_date);
```

VIEWS

Using the result.



```
SELECT name FROM managers_1992;
```



```
SELECT name FROM managers_1992 WHERE name LIKE 'D%';
```

VIEWS

Changing the view.

```
CREATE OR REPLACE VIEW managers_1992 AS
SELECT DISTINCT E.first_name || ' ' || E.last_name AS name
FROM employees E
      INNER JOIN dept_manager M ON E.emp_no=M.emp_no
WHERE 1992 BETWEEN EXTRACT(year FROM M.from_date) AND EXTRACT(year FROM M.to_date)
ORDER BY 1;
```

PL/SQL BLOCKS

PL/SQL BLOCKS

Similarly to Java's try-catch or Python try-except, PL/SQL has four keywords.

DECLARE : the declaration section, where you declare your cursors, variables, embedded functions and procedures. Not mandatory section.

BEGIN : the executable section. Each block has to have at least one statement, even if it is the no operation statement NULL. This part will contain the main DML statements that has to be executed.

EXCEPTION : the exception-handling section. This is where you'll catch any database or PL/SQL errors. Not mandatory section.

END : every PL/SQL block ends with the keyword END.

PL/SQL BLOCKS

PL/SQL BLOCKS – Anonymous blocks

Simply printing "Hello World".

```
SET SERVEROUTPUT ON SIZE 100000
```

Tells the system to printout the output generated by DBMS_OUTPUT.
Check doc: https://www.oreilly.com/library/view/oracle-sqlplus-the/0596007469/re85.html

```
BEGIN
 SYS.DBMS_OUTPUT.put_line('Hello World!');
END;
/
```

This is a simple Block. The Declare and Exception sections are missing.
/ tells the system to run the block.

```
Hello World!


PL/SQL procedure successfully completed.
```

Note that the block is Anonymous (no name is given). Anonymous blocks are not saved on the server.

PL/SQL BLOCKS

PL/SQL BLOCKS – Exceptions

Exceptions allow you to catch errors as your PL/SQL program executes, so you have control over what happens in response to those errors.

Following block tries to assign to variable my_date an invalid date (30 FEB 2020), this action will trigger the exception and print out an error message.

```
SET SERVEROUTPUT ON SIZE 100000
```

Tells the system to printout the output generated by DBMS_OUTPUT. No need to run if you run it previously.

Variable declaration. Multiple variables can be declared here.
Variable visible only inside this block.

```
DECLARE
  my_date DATE;
BEGIN
 my_date := TO_DATE('02/30/2020','MM/DD/YYYY');
EXCEPTION
 WHEN OTHERS THEN
   SYS.DBMS_OUTPUT.put_line('ERROR CONVERTING TO DATE');
END;
/
```

Variable assignment that raises exception

Catching exception

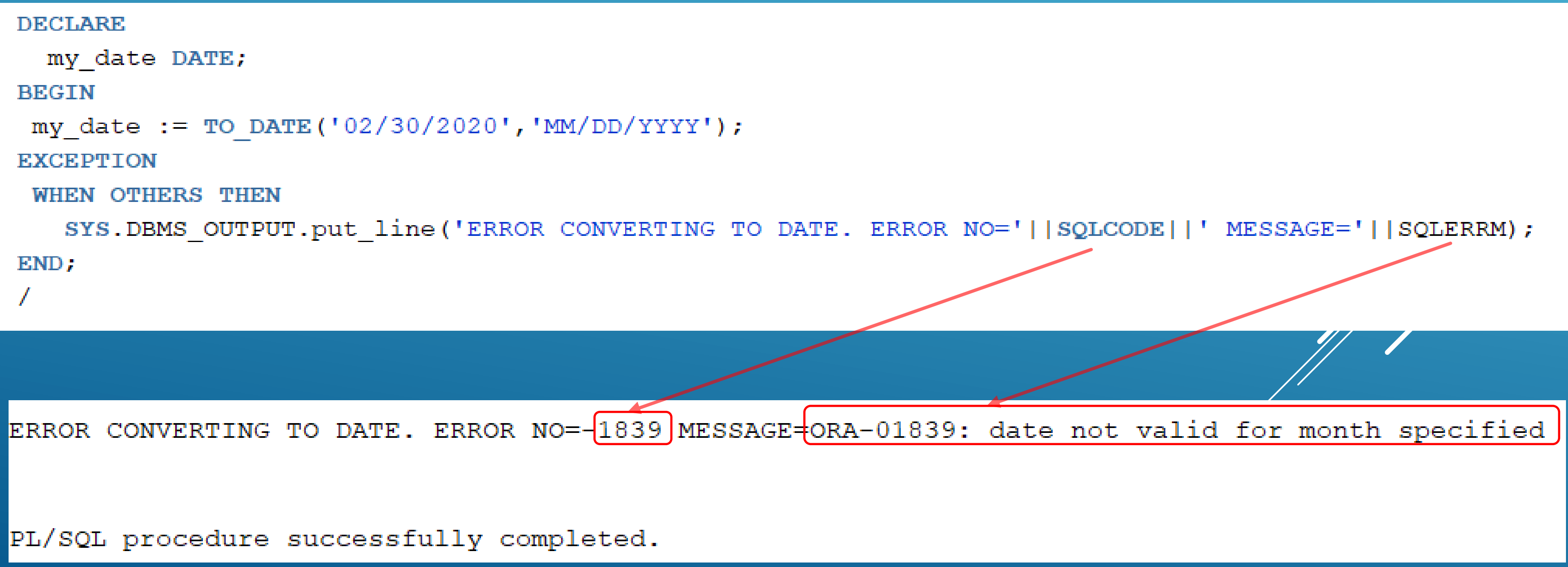
```
ERROR CONVERTING TO DATE


PL/SQL procedure successfully completed.
```

PL/SQL BLOCKS

PL/SQL BLOCKS – Exceptions

Exceptions allow you to catch errors as your PL/SQL program executes, so you have control over what happens in response to those errors.

Let's print out the system error number and message for this error. Recall | | is used for string concatenation.



```
DECLARE
  my_date DATE;
BEGIN
 my_date := TO_DATE('02/30/2020','MM/DD/YYYY');
EXCEPTION
 WHEN OTHERS THEN
   SYS.DBMS_OUTPUT.put_line('ERROR CONVERTING TO DATE. ERROR NO='||SQLCODE||' MESSAGE='||SQLERRM);
END;
/
```

```
ERROR CONVERTING TO DATE. ERROR NO=-1839 MESSAGE=ORA-01839: date not valid for month specified


PL/SQL procedure successfully completed.
```

PL/SQL BLOCKS

PL/SQL BLOCKS – Exceptions

Exceptions allow you to catch errors as your PL/SQL program executes, so you have control over what happens in response to those errors.

Let us create a block that insert a new employee in the employee table.

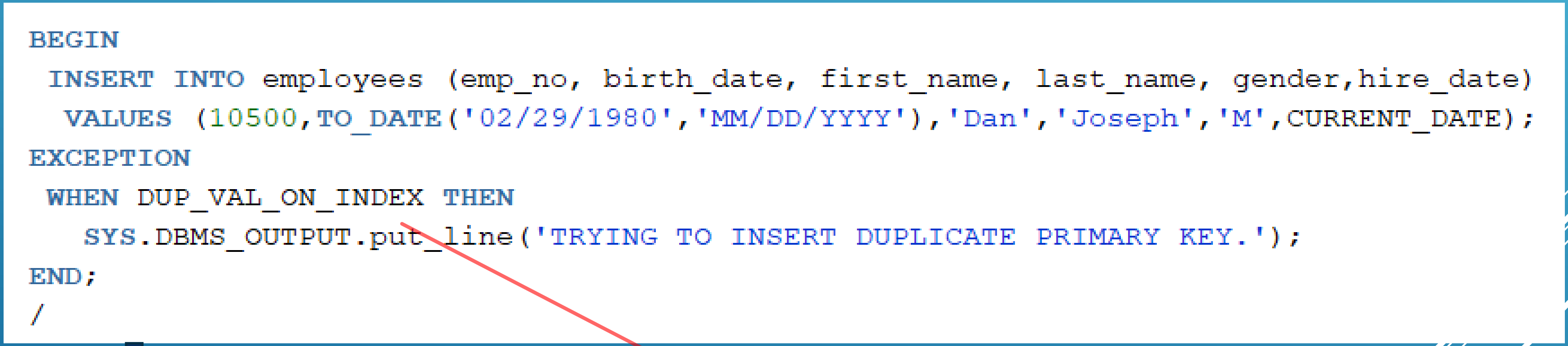
```
BEGIN
 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (10500,TO_DATE('02/29/1980','MM/DD/YYYY'),'Dan','Joseph','M',CURRENT_DATE);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
   SYS.DBMS_OUTPUT.put_line('TRYING TO INSERT DUPLICATE PRIMARY KEY.');
END;
/
```

PL/SQL BLOCKS

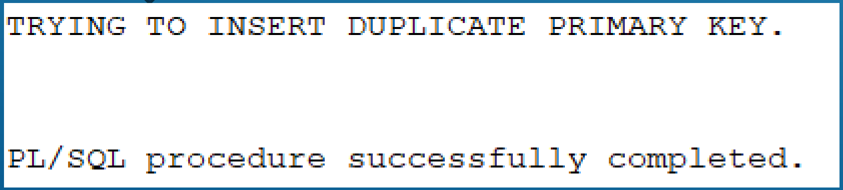
PL/SQL BLOCKS – Exceptions

Exceptions allow you to catch errors as your PL/SQL program executes, so you have control over what happens in response to those errors.

Let us create a block that insert a new employee in the employee table.



```
BEGIN
 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (10500,TO_DATE('02/29/1980','MM/DD/YYYY'),'Dan','Joseph','M',CURRENT_DATE);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
   SYS.DBMS_OUTPUT.put_line('TRYING TO INSERT DUPLICATE PRIMARY KEY.');
END;
/
```



```
TRYING TO INSERT DUPLICATE PRIMARY KEY.


PL/SQL procedure successfully completed.
```

We are trying to catch specific error not all errors.

In this case exception is triggered because there already exists an employee with emp_no=10500.

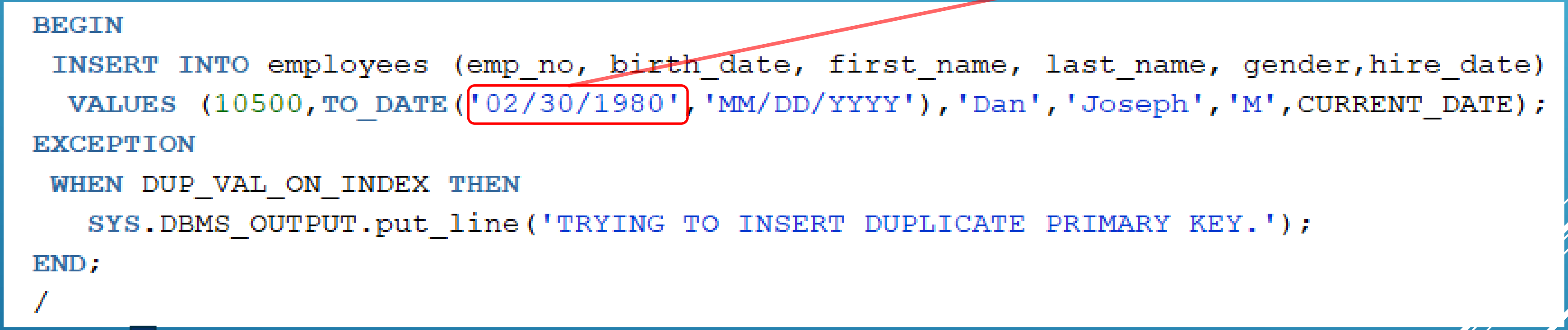
PL/SQL BLOCKS

PL/SQL BLOCKS – Exceptions

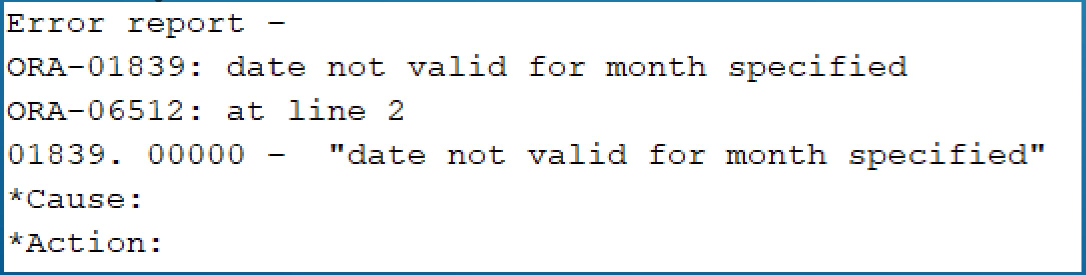
Exceptions allow you to catch errors as your PL/SQL program executes, so you have control over what happens in response to those errors.

For the previous example let us change the birth date with an invalid date.

Wrong date



```
BEGIN
 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (10500,TO_DATE('02/30/1980','MM/DD/YYYY'),'Dan','Joseph','M',CURRENT_DATE);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
   SYS.DBMS_OUTPUT.put_line('TRYING TO INSERT DUPLICATE PRIMARY KEY.');
END;
/
```



```
Error report -
ORA-01839: date not valid for month specified
ORA-06512: at line 2
01839. 00000 -  "date not valid for month specified"
*Cause:
*Action:
```

Because this exception is not threated it will raise the default exception

PL/SQL BLOCKS

PL/SQL BLOCKS – Exceptions

Exceptions allow you to catch errors as your PL/SQL program executes, so you have control over what happens in response to those errors.

Previous error can be fixed as below.

```
BEGIN
 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (10500,TO_DATE('02/30/1980','MM/DD/YYYY'),'Dan','Joseph','M',CURRENT_DATE);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
   SYS.DBMS_OUTPUT.put_line('TRYING TO INSERT DUPLICATE PRIMARY KEY.');
 WHEN OTHERS THEN
   SYS.DBMS_OUTPUT.put_line('OTHER ERROR.');
END;
/
```

```
OTHER ERROR.


PL/SQL procedure successfully completed.
```

This will be printed if we try to insert duplicate key.

This will be printed if any other error raises.

PL/SQL BLOCKS

PL/SQL BLOCKS – Exceptions

The following example ensures that either all 3 inserts are executed or none are. Can you explain why? (we will talk more about this in the transactions part)

```
SET AUTOCOMMIT OFF;
```

```
BEGIN
 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (11001,TO_DATE('02/25/1979','MM/DD/YYYY'),'Josh','Murray','M',CURRENT_DATE);

 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (11002,TO_DATE('03/17/1975','MM/DD/YYYY'),'Todd','Nick','M',CURRENT_DATE);

 INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender,hire_date)
  VALUES (10500,TO_DATE('02/30/1980','MM/DD/YYYY'),'Dan','Joseph','M',CURRENT_DATE);

 COMMIT;
EXCEPTION
 WHEN OTHERS THEN
   ROLLBACK;
   SYS.DBMS_OUTPUT.put_line('ERROR. ROLLING BACK CHANGES.');
END;
/
```

This the triggering error.

```
SET AUTOCOMMIT ON;
```

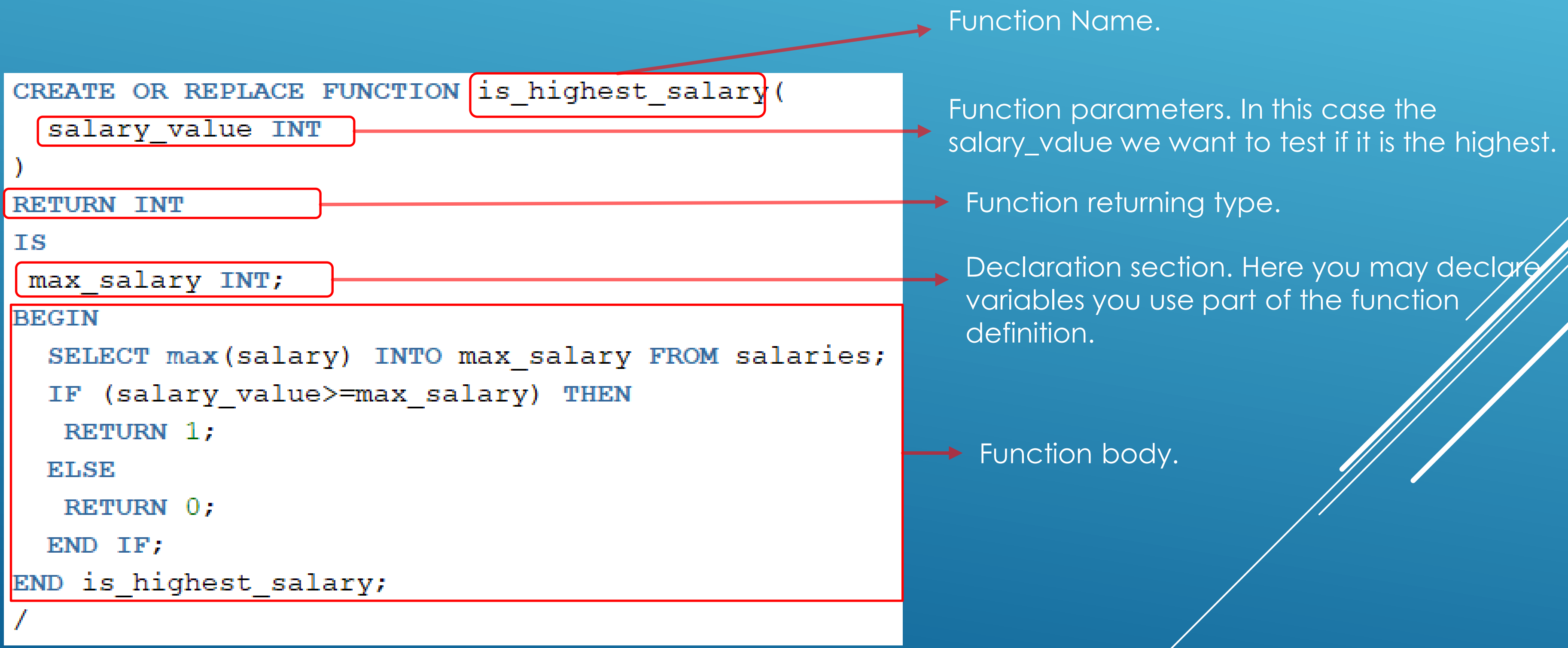
Check result.

```
SELECT * FROM employees WHERE emp_no>11000;
```

FUNCTION

A FUNCTION is a PL/SQL block or method that returns a value.

Let us create a function that has a salary as a parameter and returns 1 if the given salary is greater or equal with the highest salary from the salary table and 0 otherwise.



```
CREATE OR REPLACE FUNCTION is_highest_salary(
  salary_value INT
)
RETURN INT
IS
 max_salary INT;
BEGIN
  SELECT max(salary) INTO max_salary FROM salaries;
  IF (salary_value>=max_salary) THEN
   RETURN 1;
  ELSE
   RETURN 0;
  END IF;
END is_highest_salary;
/
```

Function Name.
Function parameters. In this case the salary_value we want to test if it is the highest.
Function returning type.
Declaration section. Here you may declare variables you use part of the function definition.
Function body.

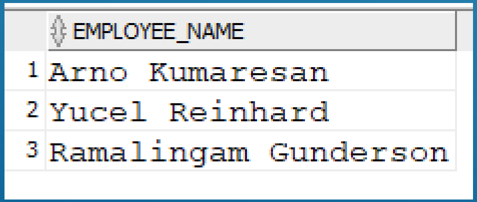
FUNCTION

A FUNCTION is a PL/SQL block or method that returns a value.

Functions are stored in the database. We can use them in conjunction with assignments (:=), selection, where conditions, updates and insert.

Return all employee names for which an increase of 10,000 of their annual salary will make them with a salary highest or equal with the current maximum salary.

```
SELECT DISTINCT E.first_name||' '||E.last_name as employee_name FROM salaries S
INNER JOIN employees E ON E.emp_no=S.emp_no
WHERE is_highest_salary(S.salary+10000)=1;
```



```
  EMPLOYEE_NAME
1 Arno Kumaresan
2 Yucel Reinhard
3 Ramalingam Gunderson
```

FUNCTION

A FUNCTION is a PL/SQL block or method that returns a value.

Functions are stored in the database. We can use them in conjunction with assignments (:=), selection, where conditions, updates and insert.

Print out if 12000 is the highest salary paid in the company.

```
SET SERVEROUTPUT ON SIZE 100000
```

```
BEGIN
  IF is_highest_salary(12000)=1 THEN
    SYS.DBMS_OUTPUT.put_line('VALUE 12000 IS HIGHER THEN THE MAX SALARY.');
  ELSE
    SYS.DBMS_OUTPUT.put_line('VALUE 12000 IS NOT HIGHER THEN THE MAX SALARY.');
  END IF;
END;
/
```

```
VALUE 12000 IS NOT HIGHER THEN THE MAX SALARY.


PL/SQL procedure successfully completed.
```

What will it return if you try with 144000?

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

procedure name

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

List of parameters of the form

```
<parameter_name_1> [IN] [OUT] <parameter_data_type_1>,
<parameter_name_2> [IN] [OUT] <parameter_data_type_2>,...
```

More on this when we'll go over an example

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

When AUTHID CURRENT_USER is specified, then the procedure is executed with the calling user entitlements. That is if the user does not have SELECT permission on a table from the procedure it will raise an exception.

When AUTHID DEFINER is specified, then the procedure is executed with the entitlements of the user that created the stored procedure (that is the owner of the schema). This is the default behavior if not specified otherwise.

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

Can use either IS or AS no difference. I prefer AS because is more close with standard SQL and other SQL based languages (T-SQL, PostgreSQL)

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

Place to declare variables used by the procedure.

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

Stored procedure block code. All statements pertinent to the stored procedure should be declared here.
The block may contain nested blocks, for example we may have s structure like:

```
BEGIN
  statements;
   BEGIN
        statements;
    END;
END name;
```

Anonymous block inside the procedure.

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

Exception handling inside the stored procedure, similar with what we did for anonymous blocks.

PL/SQL PROCEDURES

PL/SQL procedures don't return a value. They just perform their instructions and return.

Procedure Syntax

```
CREATE [OR REPLACE] PROCEDURE name
[ (parameter[,parameter]) ]
[AUTHID { CURRENT_USER | DEFINER } ]
{ IS | AS }
declaration_section
BEGIN
executable_section
[EXCEPTION
exception_section]
END [name];
```

The same as for functions, procedure blocks are closed by END followed by the name of the procedure.

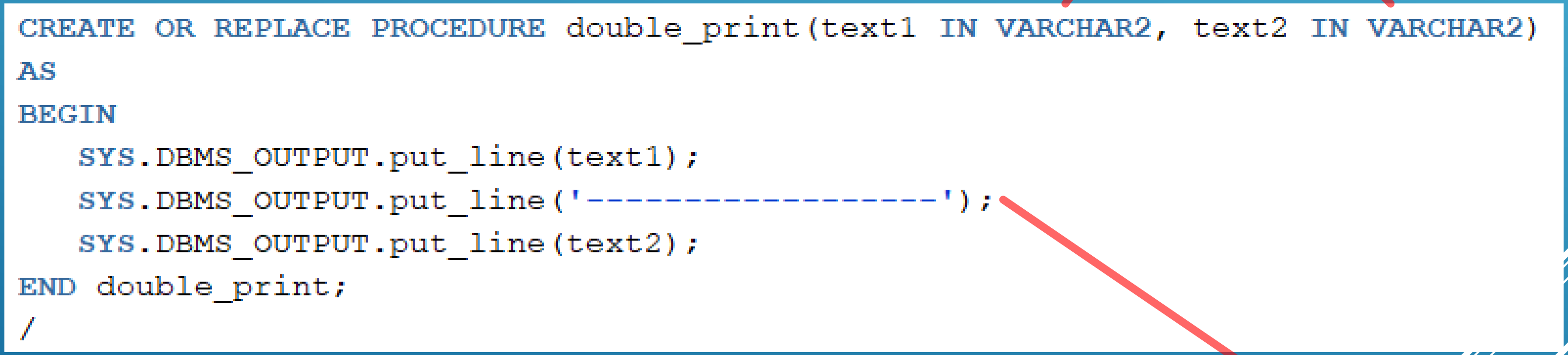
PL/SQL PROCEDURES

Let us create a simple stored procedure that print out the 2 string parameters separated by a text line.

PL/SQL PROCEDURES

Let us create a simple stored procedure that print out the 2 string parameters separated by a text line.

Size for VARCHAR2 can't be specified for parameter variables. Default size is 32K.



```
CREATE OR REPLACE PROCEDURE double_print(text1 IN VARCHAR2, text2 IN VARCHAR2)
AS
BEGIN
   SYS.DBMS_OUTPUT.put_line(text1);
   SYS.DBMS_OUTPUT.put_line('------------------');
   SYS.DBMS_OUTPUT.put_line(text2);
END double_print;
/
```

Procedure body.

PL/SQL PROCEDURES

Before running the stored procedure don't forget to run:

```
SET SERVEROUTPUT ON SIZE 100000;
```

Calling PL/SQL stored procedures.

Method 1: using standard SQL CALL statement:

```
CALL double_print('First Line', 'Second Line');
```

Method 2: using standard SQL*PLUS EXEC or EXECUTE statement:

```
EXEC double_print('First Line', 'Second Line');
```

```
EXECUTE double_print('First Line', 'Second Line');
```

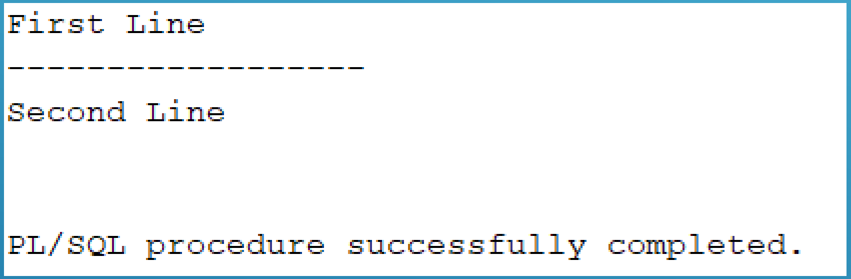
Method 3: using PL/SQL anonymous blocks:

```
BEGIN
   double_print('First Line', 'Second Line');
END;
```

PL/SQL PROCEDURES

Calling PL/SQL stored procedures.

Any of these calls will result in printing result



```
First Line
------------------
Second Line


PL/SQL procedure successfully completed.
```

If nothing is printed out be sure that you run before:

```
SET SERVEROUTPUT ON SIZE 100000;
```

PL/SQL EXECUTION CONTROL

Before creating more complex procedures let's review some basic PL/SQL program control. The following examples will use the previously defined procedure.

1. Conditional Control

IF-THEN

```
IF condition THEN
    executable statement(s)
END IF;
```

IF-THEN-ELSE

```
IF condition THEN
    executable statement(s)
ELSE
    executable statement(s)
END IF;
```

PL/SQL EXECUTION CONTROL

1. Conditional Control

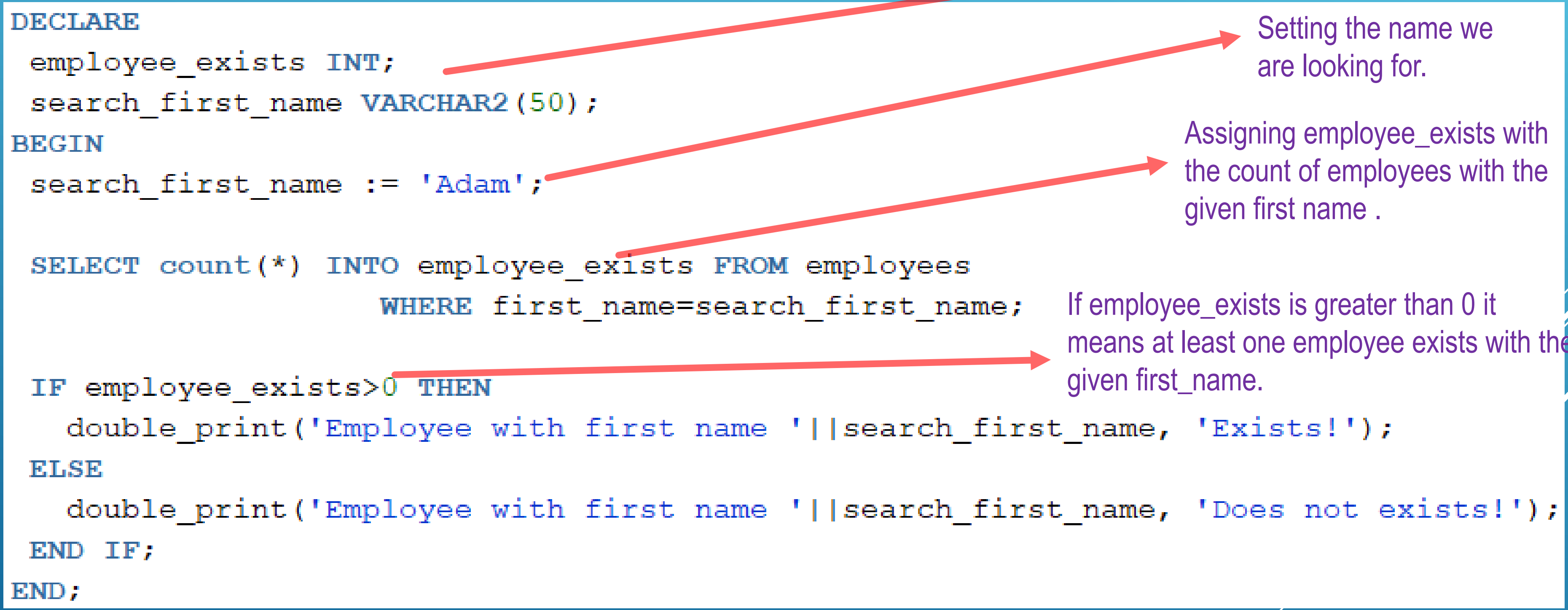
Example: check if an employee with a given first_name exists or not.

Declaring variables.

Setting the name we are looking for.

Assigning employee_exists with the count of employees with the given first name .

If employee_exists is greater than 0 it means at least one employee exists with the given first_name.



```
DECLARE
 employee_exists INT;
 search_first_name VARCHAR2(50);
BEGIN
 search_first_name := 'Adam';

 SELECT count(*) INTO employee_exists FROM employees
                   WHERE first_name=search_first_name;

 IF employee_exists>0 THEN
   double_print('Employee with first name '||search_first_name, 'Exists!');
 ELSE
   double_print('Employee with first name '||search_first_name, 'Does not exists!');
 END IF;
END;
```

Note that PL/SQL does not allow IF EXISTS (SELECT 1 FROM employees WHERE first_name='Adam') … that works fine in Sybase and SQL Server T-SQL and MySQL.

PL/SQL EXECUTION CONTROL

2. CASE expression

```
CASE WHEN condition_1 THEN value_1
     WHEN condition_2 THEN value_2
     …
     WHEN condition_n THEN value_n
     ELSE else_value
END;
```

Acts as a function that return value_1 if condition_1 is satisfied, if not value_2 if condition_2 is satisfied,…, if none of the conditions are satisfied, then it returns else_value.

PL/SQL EXECUTION CONTROL

2. CASE expression

Example: for a given employee first name display its gender as MALE, FEMALE or NULL if the employee does not exists.

```
DECLARE
 result_count int;
 search_first_name VARCHAR2(50);
 gender_value CHAR(1);
 gender_description VARCHAR2(10);
BEGIN
 search_first_name := 'Mihalis';

 SELECT count(*) INTO result_count FROM employees
               WHERE first_name=search_first_name;

 IF result_count>0 THEN
   SELECT gender INTO gender_value FROM employees
            WHERE first_name=search_first_name AND ROWNUM=1;
 ELSE
   gender_value := NULL;
 END IF;

 gender_description := CASE WHEN gender_value='M' THEN 'MALE'
                            WHEN gender_value='F' THEN 'FEMALE'
                            ELSE 'NULL'
                       END;

 double_print('The gender for '||search_first_name||' is', gender_description);
END;
```

Because there may be many employees with the same first name we are interested only in the first one.

Based on the gender found we set the variable to either MALE, FEMALE or NULL

Try this with employee Adam

PL/SQL EXECUTION CONTROL

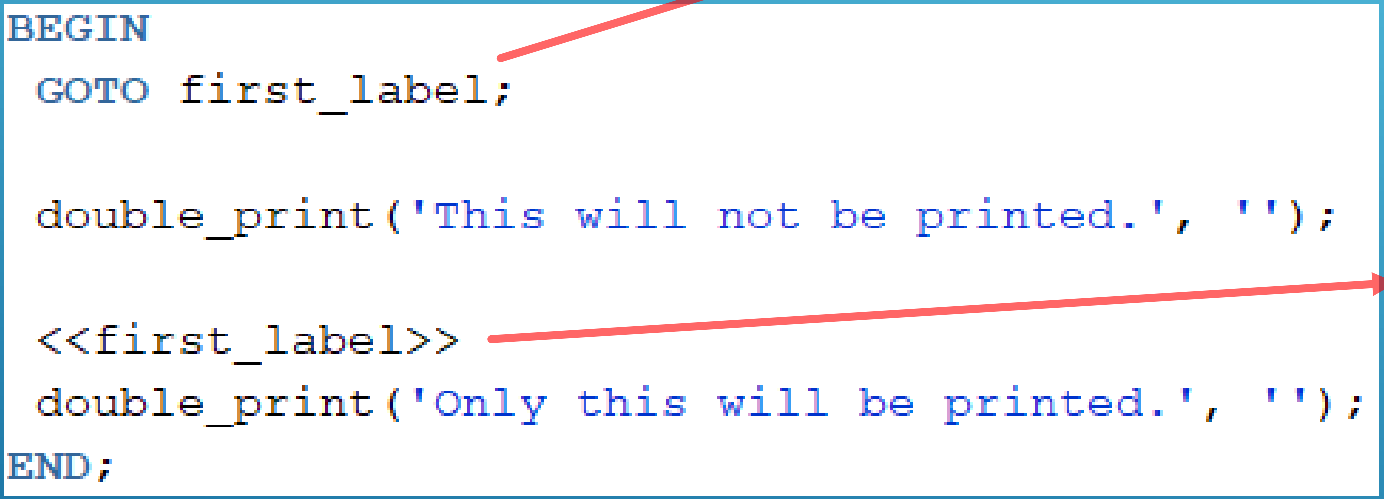
3. GOTO statement

```
GOTO label_name;
```

When reaching a GOTO statement the next statement executed will be the first one after the label definition.

PL/SQL EXECUTION CONTROL

3. GOTO statement



```
BEGIN
 GOTO first_label;

 double_print('This will not be printed.', '');

 <<first_label>>
 double_print('Only this will be printed.', '');
END;
```

GOTO statement.

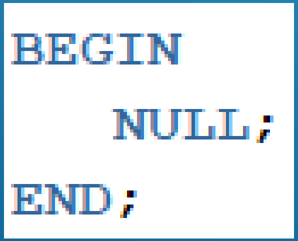
Label definition. Note the << and >> symbols used to define the label. Also no ; is needed.

PL/SQL EXECUTION CONTROL

4. NULL statement

```
NULL;
```

This is like a regular statement but does nothing. Used when there is mandatory to add a statement but we don’t want to execute anything (for example as the THEN part in an IF statement when only ELSE is needed; also to specify an empty block).



```
BEGIN
    NULL;
END;
```

PL/SQL EXECUTION CONTROL

5. Simple LOOP

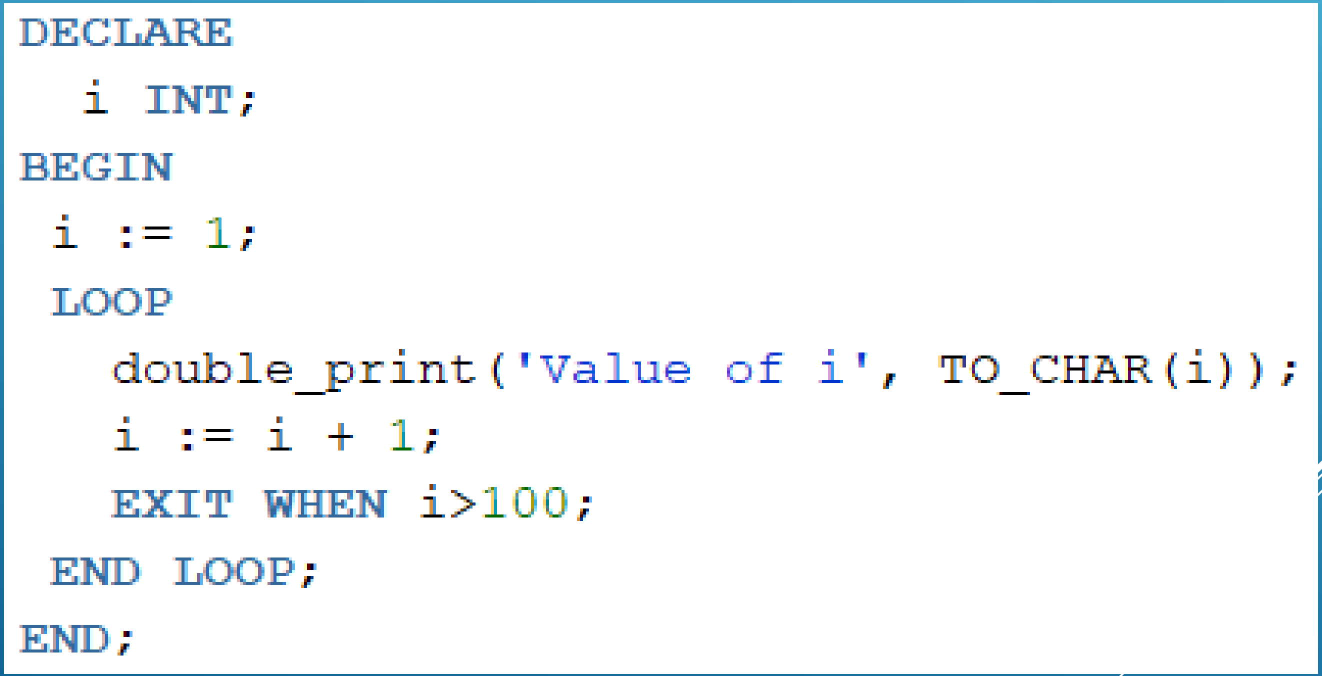
```
LOOP
    executable_statements;
   EXIT WHEN condition;
END LOOP;
```

Will execute the statements until the condition is meet. Without the exit loop condition this will run indefinitely

PL/SQL EXECUTION CONTROL

5. Simple LOOP

Following prints numbers from 1 to 100.



```
DECLARE
  i INT;
BEGIN
 i := 1;
 LOOP
   double_print('Value of i', TO_CHAR(i));
   i := i + 1;
   EXIT WHEN i>100;
 END LOOP;
END;
```

PL/SQL EXECUTION CONTROL

6. Numeric FOR LOOP

```
FOR loop_index IN [REVERSE]
    lowest_number..highest_number
LOOP
    executable_statements;
END LOOP;
```

If REVERSE is used the loop will start from the highest_value to the lowest value.

PL/SQL EXECUTION CONTROL

6. Numeric FOR LOOP

Prints numbers from 1 to 100.

```
DECLARE
  i INT;
BEGIN
 FOR i IN 1..100
 LOOP
   double_print('Value of i', TO_CHAR(i));
 END LOOP;
END;
```

Prints numbers from 100 to 1.

```
DECLARE
  i INT;
BEGIN
 FOR i IN REVERSE 1..100
 LOOP
   double_print('Value of i', TO_CHAR(i));
 END LOOP;
END;
```

PL/SQL EXECUTION CONTROL

7. WHILE LOOP

```
WHILE condition
LOOP
   executable_statements;
END LOOP;
```

Similar with WHILE statement from Java.

PL/SQL EXECUTION CONTROL

7. WHILE LOOP

Let’s print all capital letters from the English alphabet.

```
DECLARE
  str_value VARCHAR2(100);
  ascii_value INT;
BEGIN
 str_value := 'A';
 ascii_value := ASCII('A');

 WHILE LENGTH(str_value)<26
 LOOP
   ascii_value := ascii_value + 1;
   str_value := str_value || CHR(ascii_value);

 END LOOP;
 double_print('Value of my string', str_value);
END;
```

For a given character it returns its ascii code.

For a given string it returns its size.

For a integer it returns the character with the corresponding ascii value.

PL/SQL FUNCTIONS AND PROCEDURES

Let's now construct procedure add_update_employee with 3 parameters:

```
  b_date DATE
  name   VARCHAR(200)
  gender CHAR(1)
```

The procedure will check if there exists an employee with the given name. If exists it will update the b_date and gender with the given values. In case it does not exists, then it will insert a new employee with the emp_no being the highest emp_no+1, the hire_date will be set to the current date.
In case the gender is NULL or not M or F will display an error message.

For example:

CALL add_update_employee(TO_DATE('01/01/1982','MM/DD/YYYY'),' Susuma Larfeldt','F');
will update record with first_name='Susuma' and last_name='Larfeldt' setting the birth_date to 1 JAN 1982 and gender='F'

CALL add_update_employee(TO_DATE('05/02/1980','MM/DD/YYYY'),' John David','M');
Will create a new record for John David.

PL/SQL FUNCTIONS AND PROCEDURES

In order to construct the stored procedure, first we need to see how can we split the name parameter into first_name and last_name. For this let us build 2 functions that given name as parameter will return the first_name and last_name respectively.

In order to build these functions we will take advantage of the following PL/SQL functions:

LENGTH(str)	– returns the size of the string. Example: LENGTH('abc') is 3.
INSTR(str,chr)	– returns the position of chr in str. Example: INSTR('abcd ef',' ') is 5.
SUBSTR(str,start,end)	– returns the substring of str starting from start to end. Example: SUBSTR('abcdef',3,2) is 'cd'

PL/SQL FUNCTIONS AND PROCEDURES

```
CREATE OR REPLACE FUNCTION get_first_name(emp_name VARCHAR2)
RETURN VARCHAR2
IS
  space_position INT;
  name_value VARCHAR2(100);

BEGIN
NULL;
   IF emp_name IS NULL THEN
       RETURN ' ';
   END IF;

   name_value := TRIM(emp_name);

   IF LENGTH(name_value)>100 THEN
      name_value := SUBSTR(name_value,1,100);
   END IF;

   space_position := INSTR(name_value,' ');

   IF space_position=0 THEN
       RETURN name_value;
   ELSE
       RETURN SUBSTR(name_value,1,space_position);
   END IF;

END get_first_name;
```

Function that returns first name.

If the parameter is empty return empty string.

Eliminate possible spaces from the end and beginning of the string.

If the input string is larger than 100 characters consider only the first 100 characters.

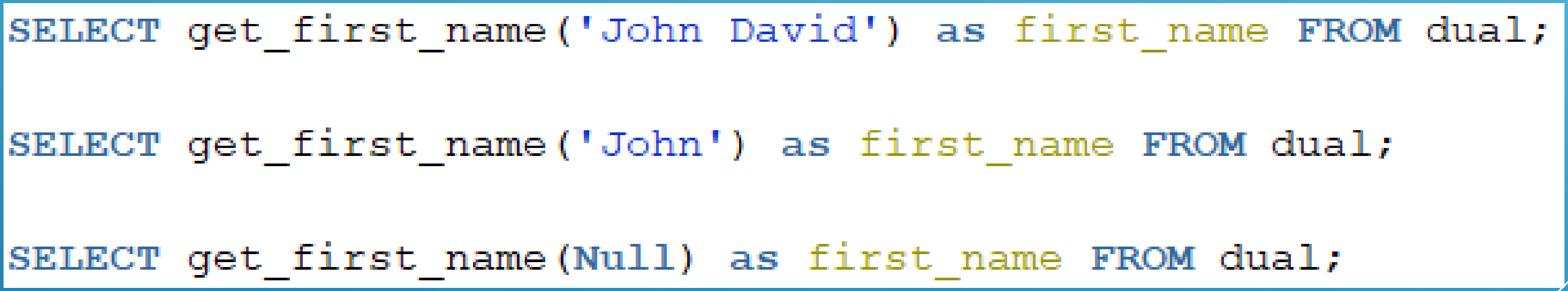
Assign to space_position the first space in the name.

If there is no space in the name, then it is the first name

Returns the first part of the name up to the space.

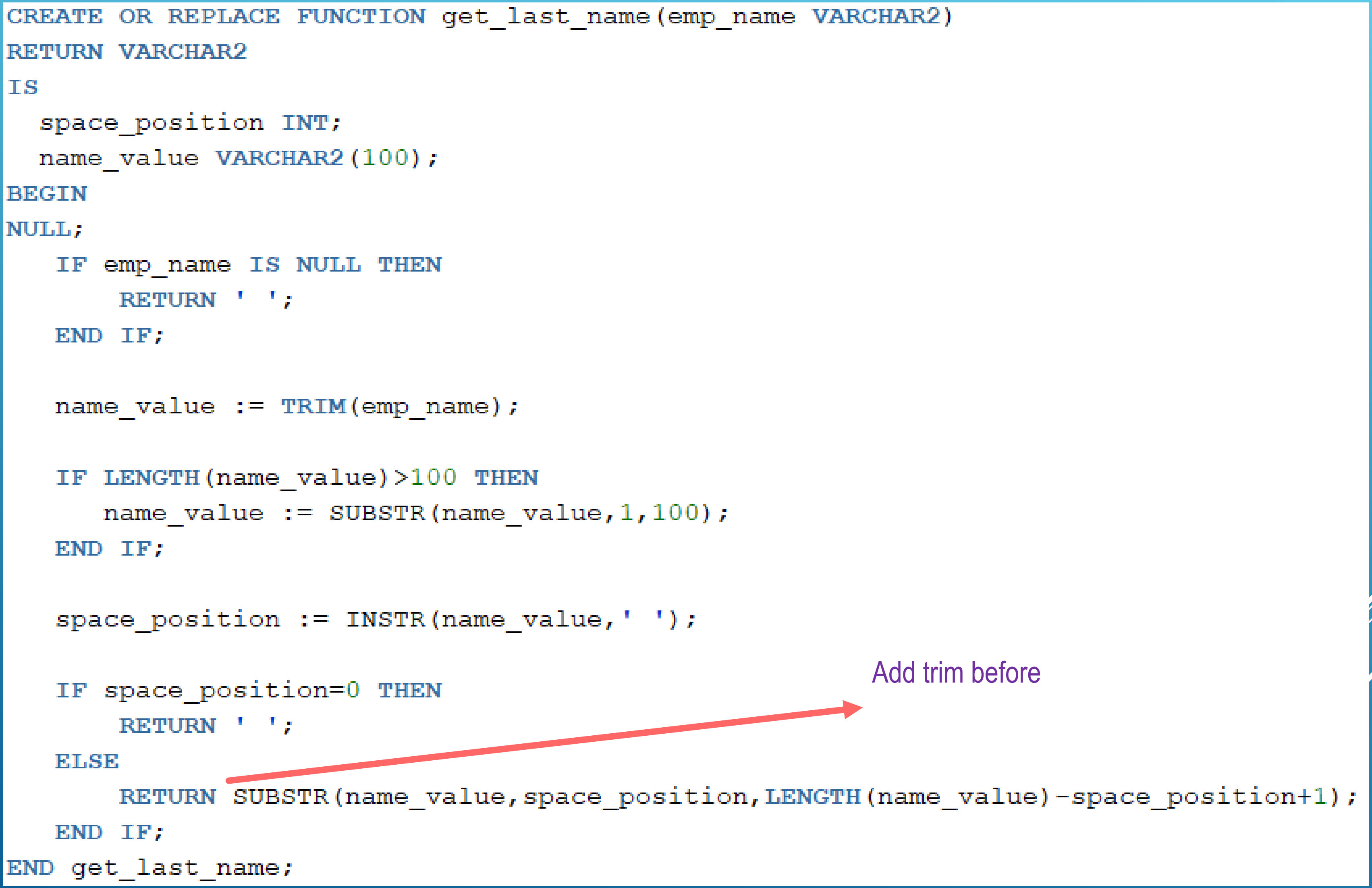
PL/SQL FUNCTIONS AND PROCEDURES

Check the function result using the following select statements.



```
SELECT get_first_name('John David') as first_name FROM dual;

SELECT get_first_name('John') as first_name FROM dual;

SELECT get_first_name(Null) as first_name FROM dual;
```

PL/SQL FUNCTIONS AND PROCEDURES



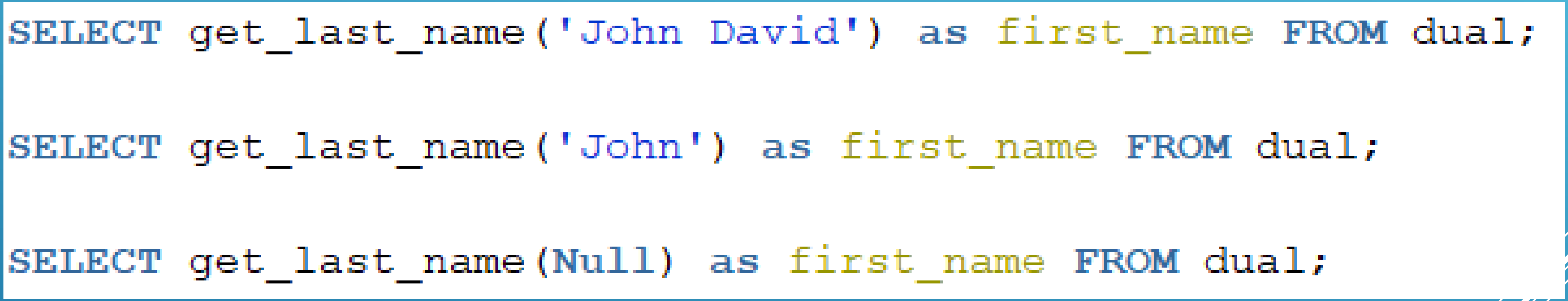
```
CREATE OR REPLACE FUNCTION get_last_name(emp_name VARCHAR2)
RETURN VARCHAR2
IS
  space_position INT;
  name_value VARCHAR2(100);
BEGIN
NULL;
   IF emp_name IS NULL THEN
       RETURN ' ';
   END IF;

   name_value := TRIM(emp_name);

   IF LENGTH(name_value)>100 THEN
      name_value := SUBSTR(name_value,1,100);
   END IF;

   space_position := INSTR(name_value,' ');

   IF space_position=0 THEN
       RETURN ' ';
   ELSE
       RETURN SUBSTR(name_value,space_position,LENGTH(name_value)-space_position+1);
   END IF;
END get_last_name;
```

Add trim before

Function that returns last name.

PL/SQL FUNCTIONS AND PROCEDURES

Check the function result using the following select statements.



```
SELECT get_last_name('John David') as first_name FROM dual;

SELECT get_last_name('John') as first_name FROM dual;

SELECT get_last_name(Null) as first_name FROM dual;
```

PL/SQL FUNCTIONS AND PROCEDURES

Main procedure code.

```
CREATE OR REPLACE PROCEDURE add_update_employee(b_date DATE,emp_name VARCHAR2,emp_gender CHAR)
AS
  max_emp_no int;
  exists_number int;
  emp_first_name VARCHAR2(100);
  emp_last_name VARCHAR2(100);
BEGIN
  IF emp_gender IS NULL OR emp_gender NOT IN ('M','F') THEN
      SYS.DBMS_OUTPUT.put_line('NOT A VALIED GENDER');
      RETURN;
  END IF;
  emp_first_name := trim(get_first_name(emp_name));
  emp_last_name  := trim(get_last_name(emp_name));

  SELECT count(*) INTO exists_number FROM employees
           WHERE first_name=emp_first_name AND last_name=emp_last_name;
  IF exists_number=0 THEN
     SELECT max(emp_no) INTO max_emp_no FROM employees;
     INSERT INTO employees(emp_no, birth_date, first_name, last_name, gender, hire_date)
         VALUES (max_emp_no+1, b_date, emp_first_name, emp_last_name, emp_gender, CURRENT_DATE);
  ELSE
     UPDATE employees SET birth_date=b_date, gender=emp_gender
        WHERE first_name=emp_first_name AND last_name=emp_last_name;
  END IF;
  commit;
END add_update_employee;
```

PL/SQL FUNCTIONS AND PROCEDURES

Main procedure code.

```
CREATE OR REPLACE PROCEDURE add_update_employee(b_date DATE,emp_name VARCHAR2,emp_gender CHAR)
AS
  max_emp_no int;
  exists_number int;
  emp_first_name VARCHAR2(100);
  emp_last_name VARCHAR2(100);
BEGIN
  IF emp_gender IS NULL OR emp_gender NOT IN ('M','F') THEN
      SYS.DBMS_OUTPUT.put_line('NOT A VALIED GENDER');
      RETURN;
  END IF;
  emp_first_name := trim(get_first_name(emp_name));
  emp_last_name  := trim(get_last_name(emp_name));

  SELECT count(*) INTO exists_number FROM employees
           WHERE first_name=emp_first_name AND last_name=emp_last_name;
  IF exists_number=0 THEN
     SELECT max(emp_no) INTO max_emp_no FROM employees;
     INSERT INTO employees(emp_no, birth_date, first_name, last_name, gender, hire_date)
         VALUES (max_emp_no+1, b_date, emp_first_name, emp_last_name, emp_gender, CURRENT_DATE);
  ELSE
     UPDATE employees SET birth_date=b_date, gender=emp_gender
        WHERE first_name=emp_first_name AND last_name=emp_last_name;
  END IF;
  commit;
END add_update_employee;
```

Check if Gender is ok.

Get first_name and last_name from parameter value.

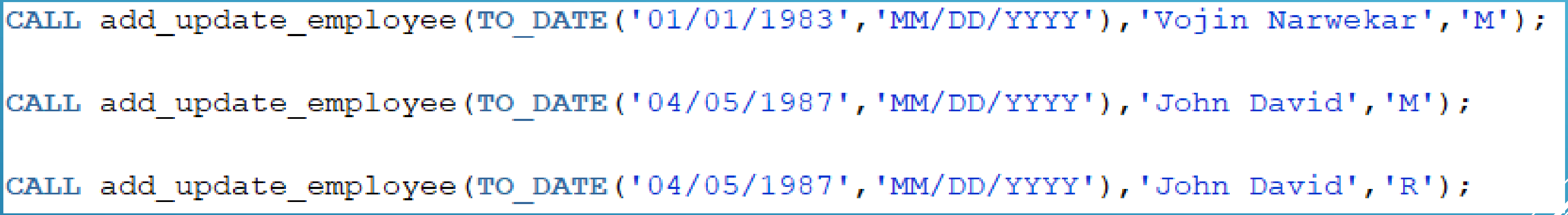
Check if employee with the given first_name and last_name exists.

Insert new employee.

Update existing employee.

PL/SQL FUNCTIONS AND PROCEDURES

Test procedure by running.



```
CALL add_update_employee(TO_DATE('01/01/1983','MM/DD/YYYY'),'Vojin Narwekar','M');

CALL add_update_employee(TO_DATE('04/05/1987','MM/DD/YYYY'),'John David','M');

CALL add_update_employee(TO_DATE('04/05/1987','MM/DD/YYYY'),'John David','R');
```

PL/SQL VARIABLES

Variables are named temporary storage locations that support a particular data type in your PL/SQL program. You must declare them in the declaration section of a PL/SQL block.

Variable names must be less than 31 characters length.

It is good practice to add as first letter in the variable name to identify variable type followed by _.

Prefix	**Data Type**
c_	CURSOR
d_	DATE
n_	NUMBER
i_	INT
r_	ROW
t_	TABLE
v_	VARCHAR2

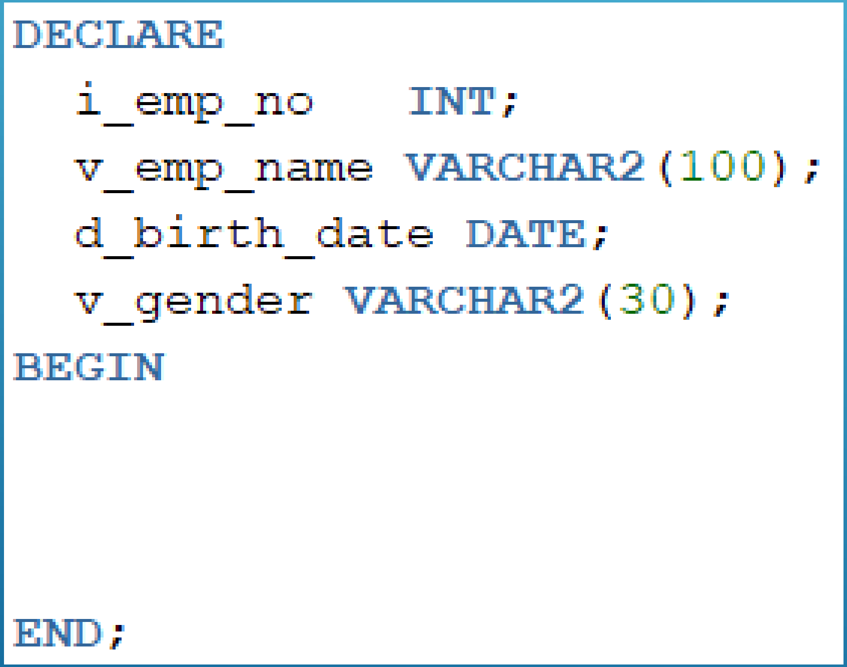
For example:

i_emp_no
d_birth_date
v_name

PL/SQL VARIABLES

Variable Declaration.

To declare a variable, type the variable name (identifier) followed by the data type definition terminated by a semicolon (;).



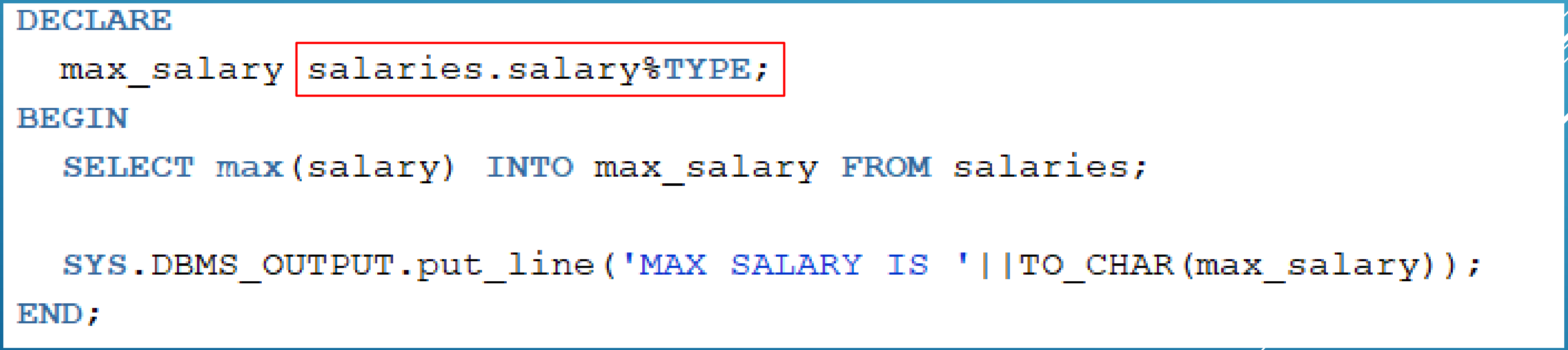
```
DECLARE
   i_emp_no    INT;
   v_emp_name VARCHAR2(100);
   d_birth_date DATE;
   v_gender VARCHAR2(30);
BEGIN




END;
```

PL/SQL VARIABLES

Variable Anchors.

An anchor refers to the use of the keyword %TYPE to “anchor” a PL/SQL data type definition in a PL/SQL variable.

In the next example we define the type of the max_salary variable to be the same type as the column salary from salaries table.

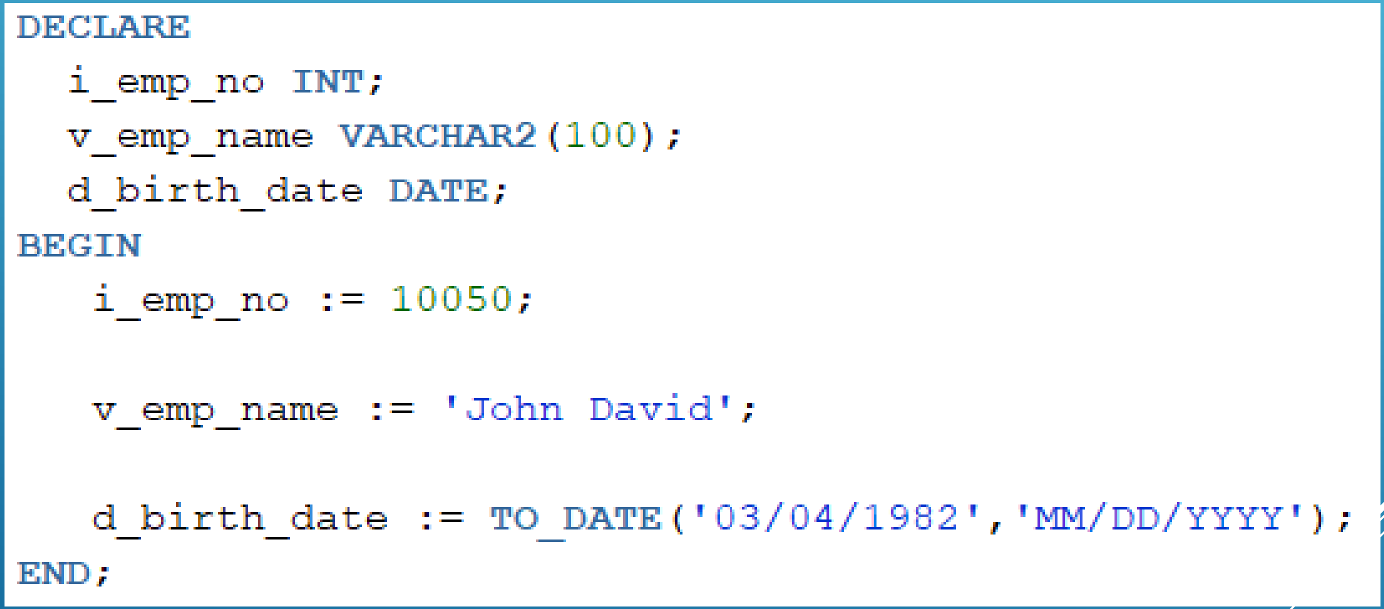


```
DECLARE
  max_salary salaries.salary%TYPE;
BEGIN
  SELECT max(salary) INTO max_salary FROM salaries;

  SYS.DBMS_OUTPUT.put_line('MAX SALARY IS '||TO_CHAR(max_salary));
END;
```

PL/SQL VARIABLES

Variable Assignments.

To assign a literal value to a variable in PL/SQL, you use the assignment operator, which is :=.



```
DECLARE
  i_emp_no INT;
  v_emp_name VARCHAR2(100);
  d_birth_date DATE;
BEGIN
   i_emp_no := 10050;

   v_emp_name := 'John David';

   d_birth_date := TO_DATE('03/04/1982','MM/DD/YYYY');
END;
```

PL/SQL VARIABLES

Variable Scope.

Any variable declared in the declaration section of a function/procedure/anonymous block is visible only within the same function/procedure/block.

PL/SQL TYPES

Table Type.

PL/SQL Table type is referring to a temporarily stored data in memory.

Define new type table_type as a TABLE of strings.

Table elements are identified by integers.

```
DECLARE
  TYPE table_type IS TABLE OF VARCHAR(100) INDEX BY BINARY_INTEGER;

  t_my_table table_type;

  i_index INT;
BEGIN
   t_my_table(1) := 'John';
   t_my_table(2) := 'Marry';
   t_my_table(3) := 'Adam';
   SYS.DBMS_OUTPUT.put_line('Our table has '||t_my_table.count()||' elements.');

   i_index := 1;

   WHILE (i_index<=3)
   LOOP
     SYS.DBMS_OUTPUT.put_line('Value of '||TO_CHAR(i_index)||' is '||t_my_table(i_index));
     i_index := i_index + 1;
   END LOOP;
END;
```

Define new variable of type table_type. TABLE type can be viewed as a list of elements type. In this case VARCHAR2(100);

Assign values to table elements;

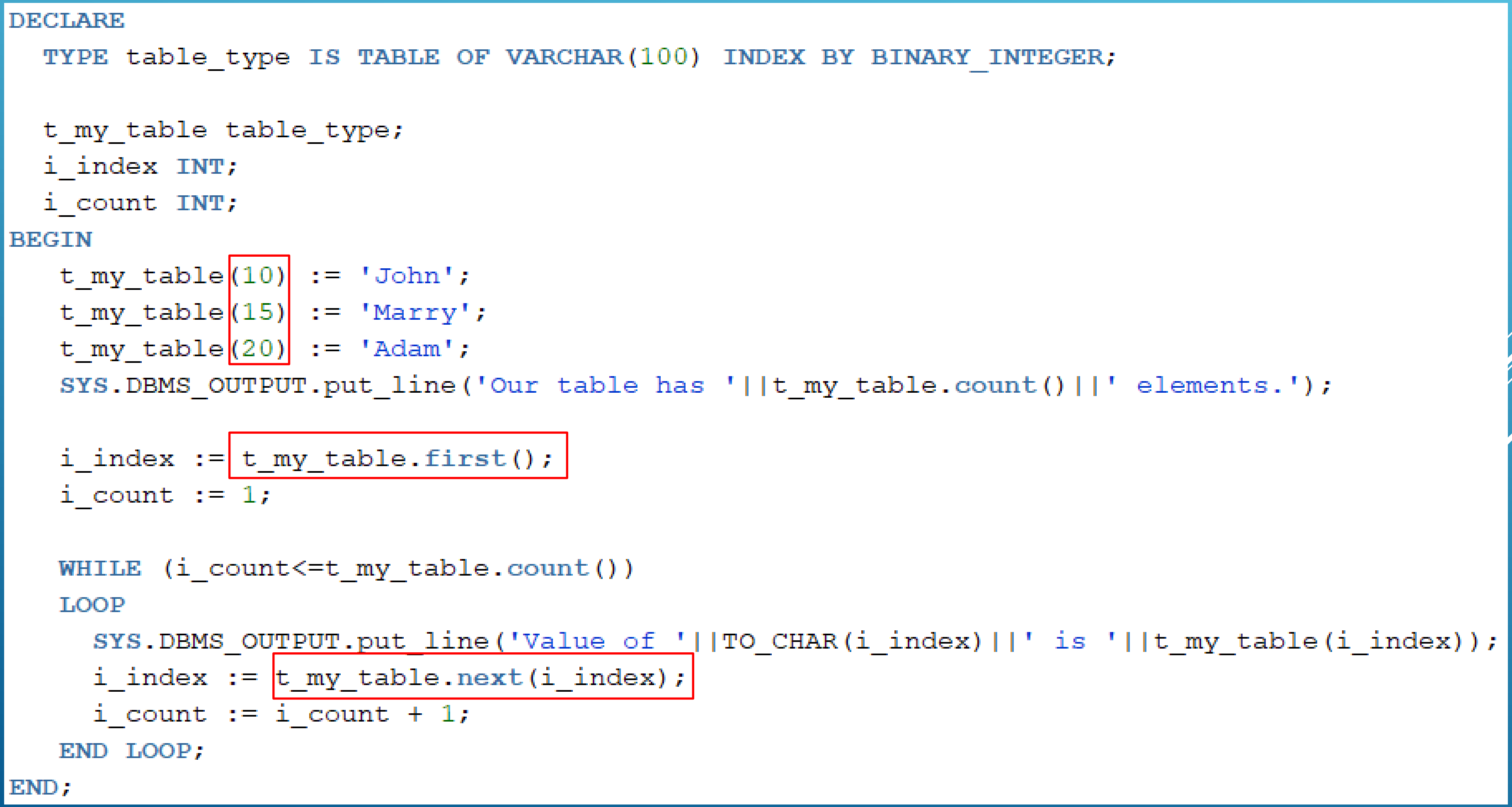
Count() function returns number of elements in the table;

Looping through the table elements;

PL/SQL TYPES

Table Type.

The index does not have to be sequential. Below example the indexes are random, the same problem as previously.



```
DECLARE
  TYPE table_type IS TABLE OF VARCHAR(100) INDEX BY BINARY_INTEGER;

  t_my_table table_type;
  i_index INT;
  i_count INT;
BEGIN
   t_my_table(10) := 'John';
   t_my_table(15) := 'Marry';
   t_my_table(20) := 'Adam';
   SYS.DBMS_OUTPUT.put_line('Our table has '||t_my_table.count()||' elements.');

   i_index := t_my_table.first();
   i_count := 1;

   WHILE (i_count<=t_my_table.count())
   LOOP
     SYS.DBMS_OUTPUT.put_line('Value of '||TO_CHAR(i_index)||' is '||t_my_table(i_index));
     i_index := t_my_table.next(i_index);
     i_count := i_count + 1;
   END LOOP;
END;
```

PL/SQL TYPES

Table Type.

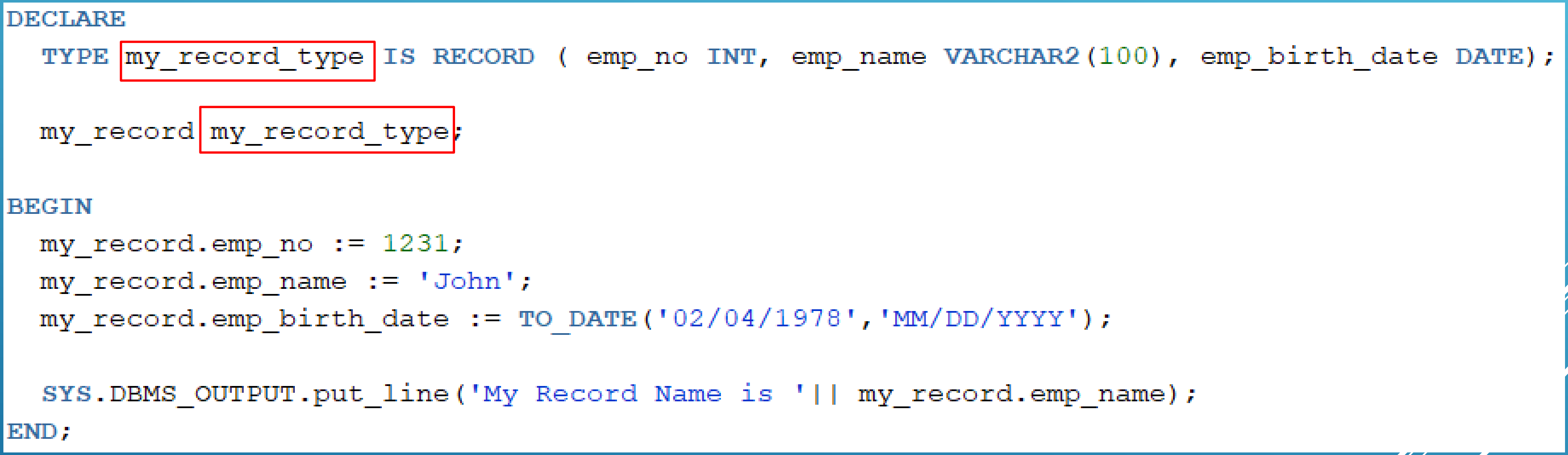
The index does not have to be sequential. Below example the indexes are random, the same problem as previously.

Method	Description
count()	Returns the number of elements
delete(index)	Deletes the specified element
delete()	Deletes all elements
exists(index)	Returns TRUE if the element exists; otherwise, FALSE
first()	Returns the index of the first element
last()	Returns the index of the last element
prior(index)	Returns the index of the first element before the specified element
next(index)	Returns the index of the first element after the specified element

PL/SQL TYPES

Record Type.

Similarly to table type you can define Record type. Below r_my_record is defined as of type my_record_type.



```
DECLARE
  TYPE my_record_type IS RECORD ( emp_no INT, emp_name VARCHAR2(100), emp_birth_date DATE);

  my_record my_record_type;

BEGIN
  my_record.emp_no := 1231;
  my_record.emp_name := 'John';
  my_record.emp_birth_date := TO_DATE('02/04/1978','MM/DD/YYYY');

  SYS.DBMS_OUTPUT.put_line('My Record Name is '|| my_record.emp_name);
END;
```

PL/SQL TYPES

Record Type.

You can now combine tables with records such that the new table type will be a list of records..

Define record type.

Define table type as list of record types.

New table variable.

```
DECLARE
  TYPE my_record_type IS RECORD ( emp_no INT, emp_name VARCHAR2(100), emp_birth_date DATE);
  TYPE my_table_type IS TABLE OF my_record_type INDEX BY BINARY_INTEGER;

  t_my_table my_table_type;
BEGIN
   t_my_table(1).emp_no := 1231;
   t_my_table(1).emp_name := 'John';
   t_my_table(1).emp_birth_date := TO_DATE('02/04/1978','MM/DD/YYYY');
   t_my_table(2).emp_no := 1578;
   t_my_table(2).emp_name := 'Marry';
   t_my_table(2).emp_birth_date := TO_DATE('05/20/1991','MM/DD/YYYY');

   SYS.DBMS_OUTPUT.put_line('First record name is '||t_my_table(1).emp_name);
   SYS.DBMS_OUTPUT.put_line('Second record name is '||t_my_table(2).emp_name);
END;
```

PL/SQL CURSORS

Cursor.

A cursor is a named Select statement that you can use in your PL/SQL program to access multiple rows from a table, yet retrieve them one row at a time.

Cursors are declared in the declaration section of your block just as you declare variables.

Cursors definition.

Parameters are optional. We will show an example on the usage.

```
CURSOR <cursor_name> [(
                <parameter_name_1> [IN] <parameter_data_type_1>,
                <parameter_name_2> [IN] <parameter_data_type_2>,...
                <parameter_name_N> [IN] <parameter_data_type_N> )] IS
<select_statement>;
```

PL/SQL CURSORS

Cursor.

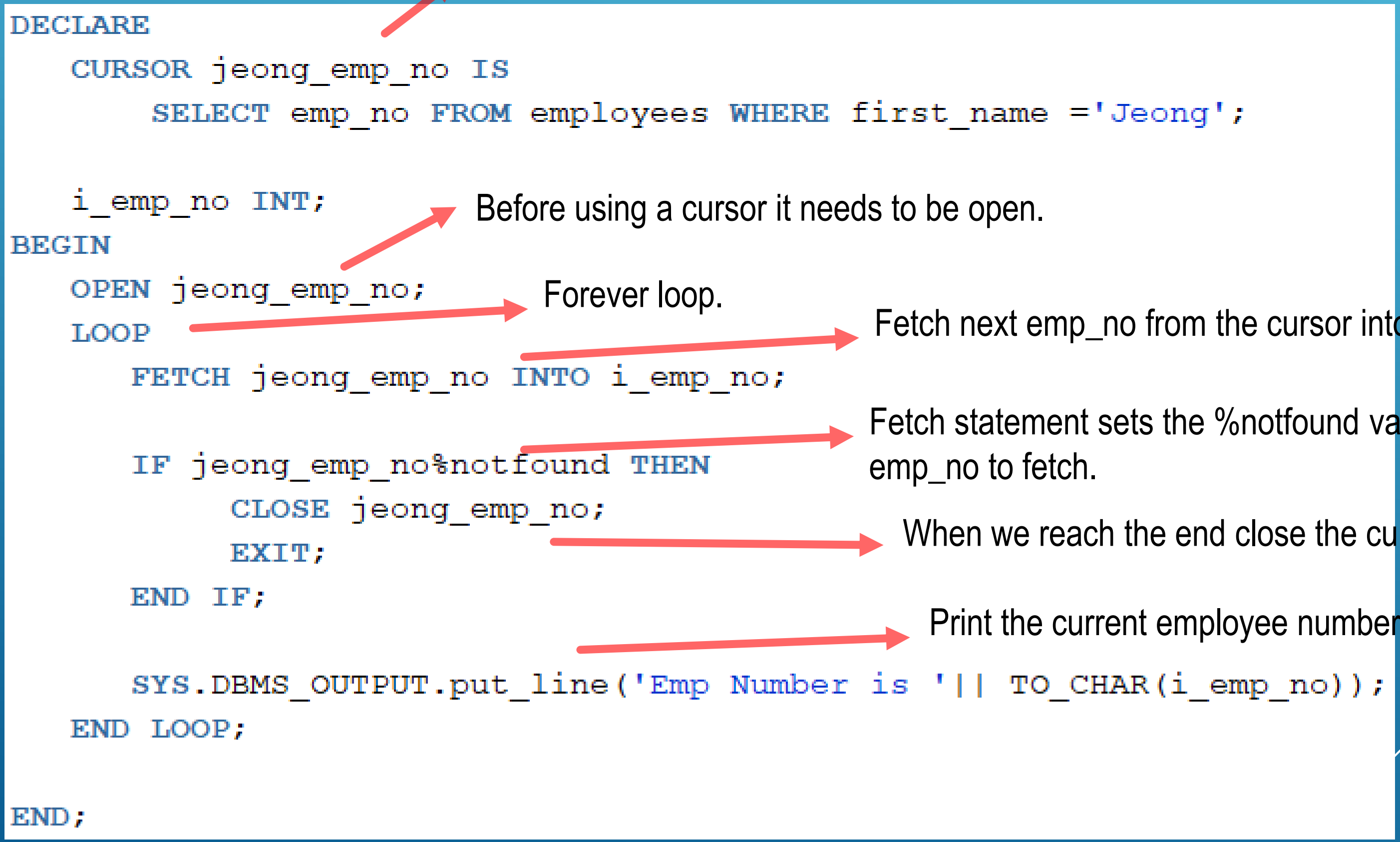
Fetching rows from a cursor.

```
OPEN <cursor_name> [(
                        <parameter_value_1,
                        <parameter_value_2>,...
                        <parameter_value_N> )];
LOOP

    FETCH <cursor_name> INTO <variable_name_1>, <variable_name_2>,... variable_name_N>;

    IF <cursor_name>%notfound THEN   -- if nothing to fetch exit
        CLOSE <cursor_name>;
        EXIT;
     END IF;

     -- process fetched variables
END LOOP;

CLOSE <cursor_name>;
```

PL/SQL CURSORS

Cursor.

Let us define a cursor for all employee numbers whose first_name is Jeong and then traverse the cursor and print these employee numbers.



```
DECLARE
   CURSOR jeong_emp_no IS
       SELECT emp_no FROM employees WHERE first_name ='Jeong';

   i_emp_no INT;
BEGIN
   OPEN jeong_emp_no;
   LOOP
      FETCH jeong_emp_no INTO i_emp_no;

      IF jeong_emp_no%notfound THEN
           CLOSE jeong_emp_no;
           EXIT;
      END IF;

      SYS.DBMS_OUTPUT.put_line('Emp Number is '|| TO_CHAR(i_emp_no));
   END LOOP;

END;
```

Define cursor jeong_emp_no as the given select statement.

Before using a cursor it needs to be open.

Forever loop.

Fetch next emp_no from the cursor into variable i_emp_no.

Fetch statement sets the %notfound value to True if there is not other emp_no to fetch.

When we reach the end close the cursor and exit the forever loop. .

Print the current employee number.

PL/SQL CURSORS

Parametrized Cursor.

Cursor parameter

```
DECLARE
   CURSOR cursor_emp_no (cursor_first_name VARCHAR2) IS
       SELECT emp_no FROM employees WHERE first_name =cursor_first_name;

   i_emp_no INT;
BEGIN
   OPEN cursor_emp_no('John');
   LOOP
      FETCH cursor_emp_no INTO i_emp_no;

      IF cursor_emp_no%notfound THEN
           CLOSE cursor_emp_no;
           EXIT;
      END IF;

      SYS.DBMS_OUTPUT.put_line('John Emp Number is '|| TO_CHAR(i_emp_no));
   END LOOP;

   OPEN cursor_emp_no('Jeong');
   LOOP
      FETCH cursor_emp_no INTO i_emp_no;

      IF cursor_emp_no%notfound THEN
           CLOSE cursor_emp_no;
           EXIT;
      END IF;

      SYS.DBMS_OUTPUT.put_line('Jeong Emp Number is '|| TO_CHAR(i_emp_no));
   END LOOP;
END;
```

Using cursor parameter

Same cursor used for both John and Joeng.

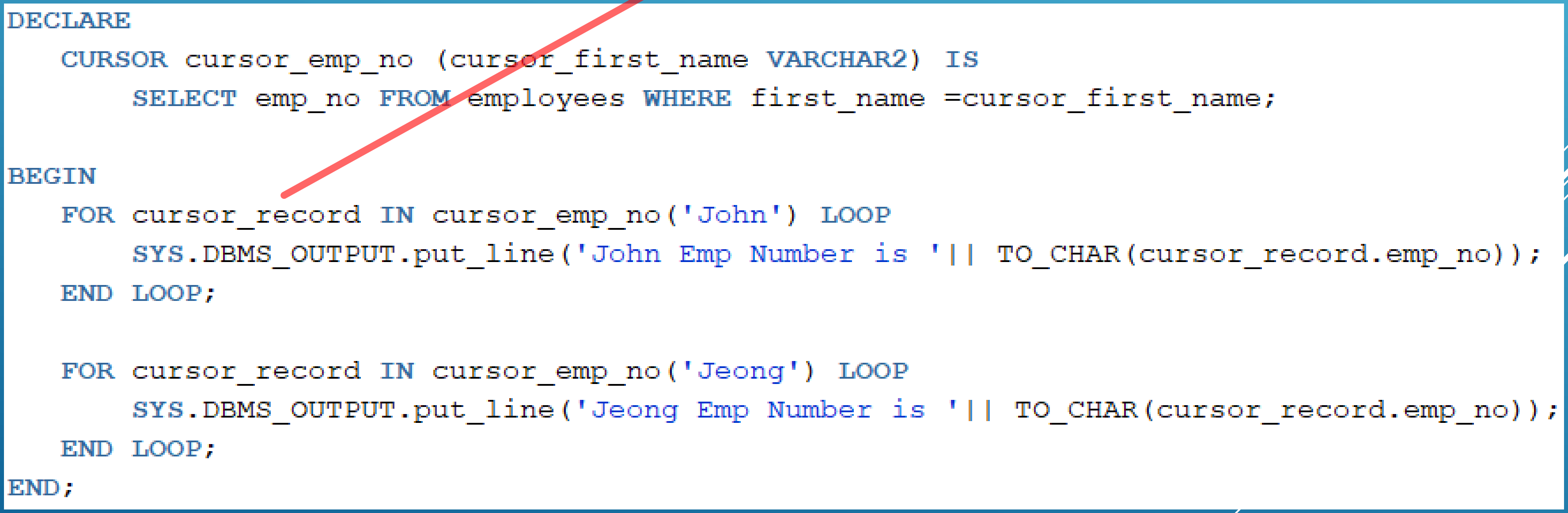
We can use parametrized cursors in case we want to use the same cursors but with different parameters. Below example prints employee numbers associated with John and Jeong

PL/SQL CURSORS

Fetching cursor automatically.

Instead of the manual cursor traversal we can use the CURSOR FOR LOOP. To make it more clear let's recreate previous example using automated cursor traversal

cursor_record will contain the record from cursor.



```
DECLARE
   CURSOR cursor_emp_no (cursor_first_name VARCHAR2) IS
       SELECT emp_no FROM employees WHERE first_name =cursor_first_name;

BEGIN
   FOR cursor_record IN cursor_emp_no('John') LOOP
       SYS.DBMS_OUTPUT.put_line('John Emp Number is '|| TO_CHAR(cursor_record.emp_no));
   END LOOP;

   FOR cursor_record IN cursor_emp_no('Jeong') LOOP
       SYS.DBMS_OUTPUT.put_line('Jeong Emp Number is '|| TO_CHAR(cursor_record.emp_no));
   END LOOP;
END;
```

PL/SQL CURSORS

Cursor for update.

We can use cursors to lock the fields we like to update. In the next example we update first_name for each employee with first_name='Jeong' adding an index number to differentiate them.

```
DECLARE
   CURSOR cursor_emp_no (cursor_first_name VARCHAR2) IS
       SELECT emp_no FROM employees WHERE first_name =cursor_first_name
       FOR UPDATE OF first_name;

   i_index INT;
BEGIN
   i_index :=1;
   FOR cursor_record IN cursor_emp_no('Jeong') LOOP
       UPDATE employees SET first_name = first_name||' '||TO_CHAR(i_index)
               WHERE emp_no = cursor_record.emp_no;
       i_index := i_index + 1;
   END LOOP;

   COMMIT;
END;
```

Locking the first_name columns to be updated.

You may check the result by running.

```
SELECT * FROM employees WHERE first_name LIKE 'Jeong%';
```

PL/SQL CURSORS

Cursor usage.

Reasons not to use cursors.

- One SQL statement to accomplish one goal means fewer cursors in use on your database, and that means better performance.
- One SQL statement to accomplish one task means consistent behavior across your application's presentation layers.
- One SQL statement to accomplish one requirement means it will be easier to maintain and modify your application's code.
- One SQL statement to attain the goals just mentioned means saving money, and saved money is profit.

PL/SQL CURSORS

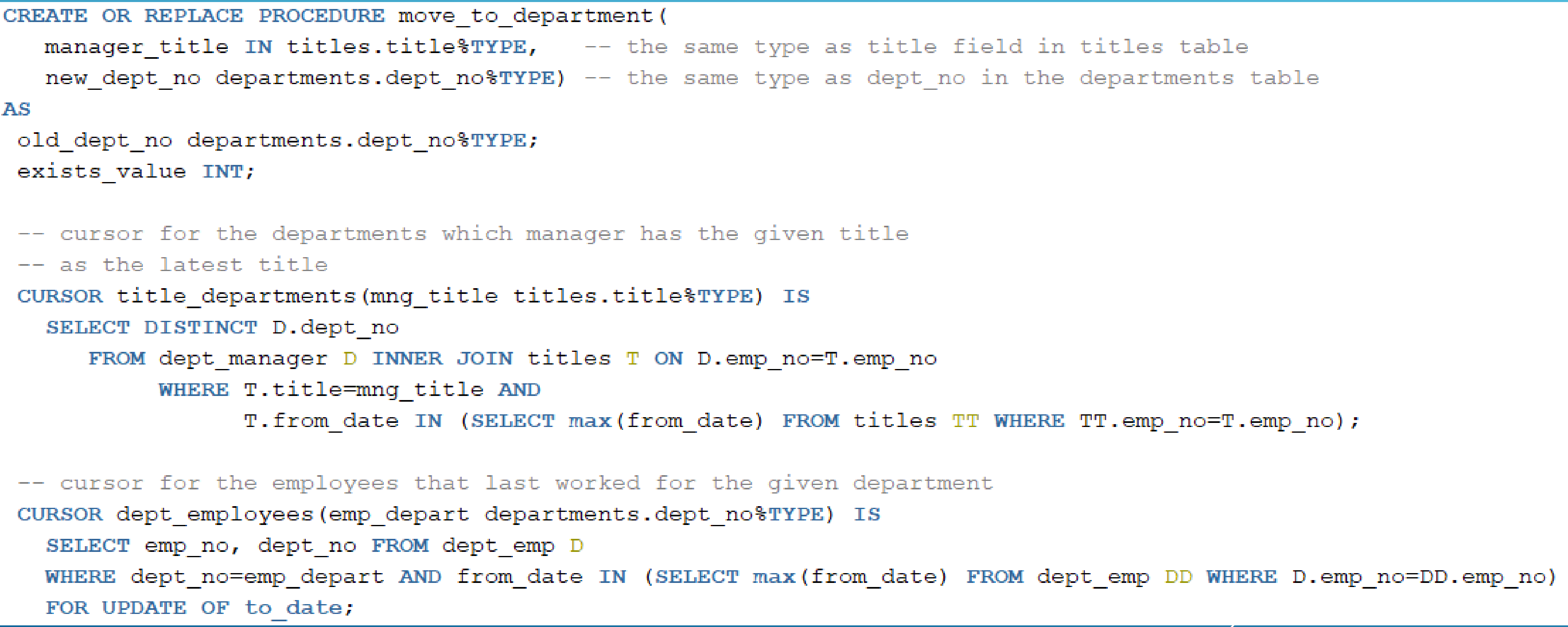
Cursor example.

Below is a more complicated example of cursor usage.
Problem: Create procedure move_to_department with 2 parameters manager_title and new_dept_no , that will move all employees for the department managed by an employee whose latest title is manager_title to the department new_dept_no.

PL/SQL CURSORS

Cursor example.

PART 1. Parameters, Variables and Cursors declaration.

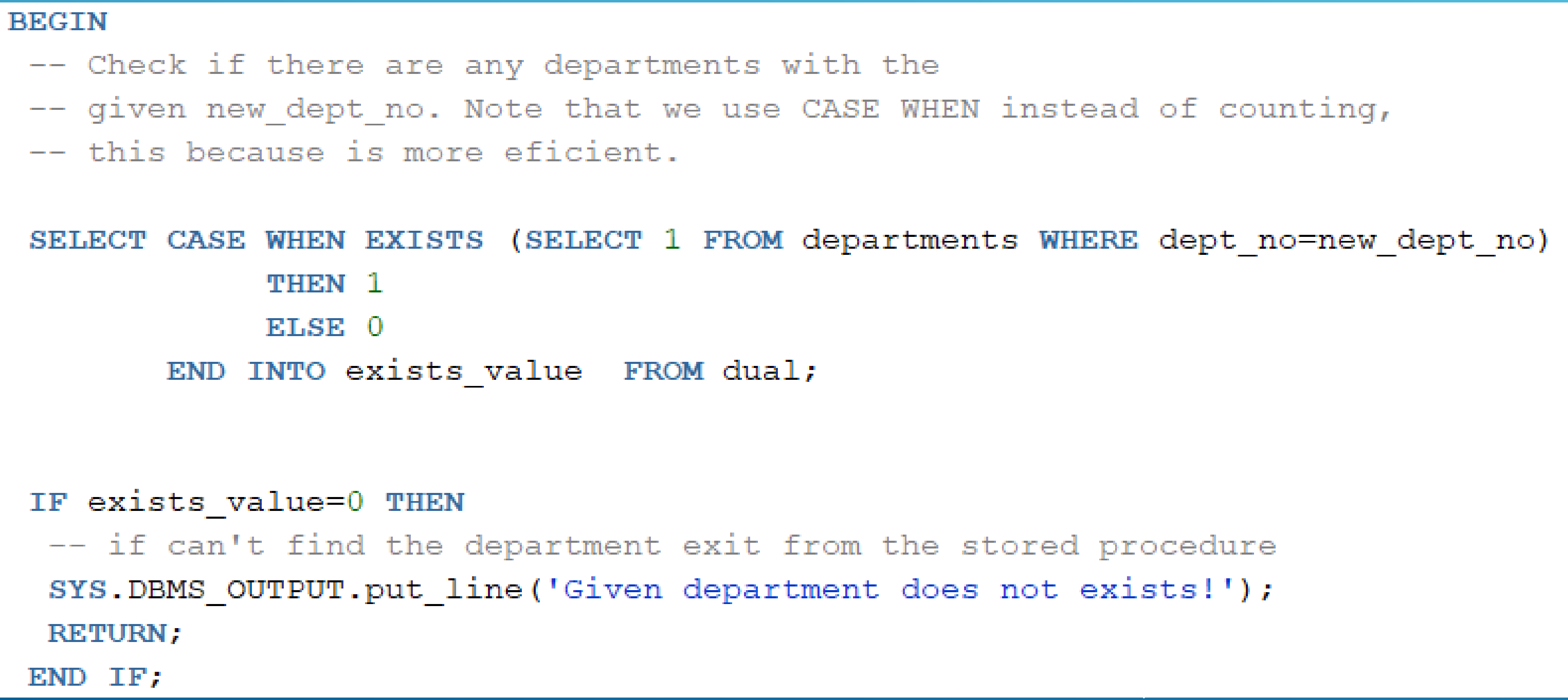


```
CREATE OR REPLACE PROCEDURE move_to_department(
   manager_title IN titles.title%TYPE,   -- the same type as title field in titles table
   new_dept_no departments.dept_no%TYPE) -- the same type as dept_no in the departments table
AS
 old_dept_no departments.dept_no%TYPE;
 exists_value INT;

 -- cursor for the departments which manager has the given title
 -- as the latest title
 CURSOR title_departments(mng_title titles.title%TYPE) IS
   SELECT DISTINCT D.dept_no
      FROM dept_manager D INNER JOIN titles T ON D.emp_no=T.emp_no
           WHERE T.title=mng_title AND
                 T.from_date IN (SELECT max(from_date) FROM titles TT WHERE TT.emp_no=T.emp_no);

 -- cursor for the employees that last worked for the given department
 CURSOR dept_employees(emp_depart departments.dept_no%TYPE) IS
   SELECT emp_no, dept_no FROM dept_emp D
   WHERE dept_no=emp_depart AND from_date IN (SELECT max(from_date) FROM dept_emp DD WHERE D.emp_no=DD.emp_no)
   FOR UPDATE OF to_date;
```

PL/SQL CURSORS

Cursor example.

PART 2. Check if given new department exists.

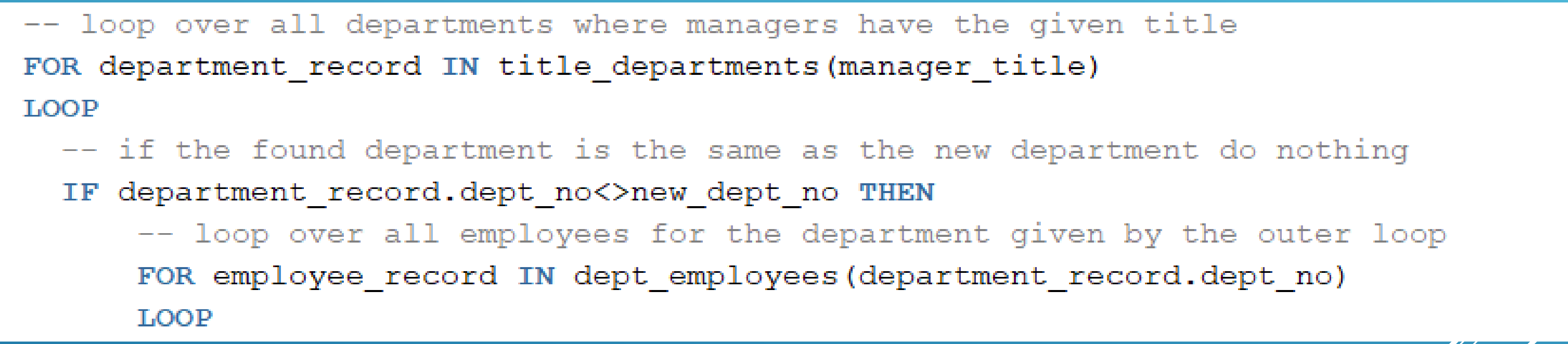


```
BEGIN
 -- Check if there are any departments with the
 -- given new_dept_no. Note that we use CASE WHEN instead of counting,
 -- this because is more eficient.

 SELECT CASE WHEN EXISTS (SELECT 1 FROM departments WHERE dept_no=new_dept_no)
             THEN 1
             ELSE 0
        END INTO exists_value  FROM dual;


 IF exists_value=0 THEN
  -- if can't find the department exit from the stored procedure
  SYS.DBMS_OUTPUT.put_line('Given department does not exists!');
  RETURN;
 END IF;
```

PL/SQL CURSORS

Cursor example.

PART 3. Nested loops

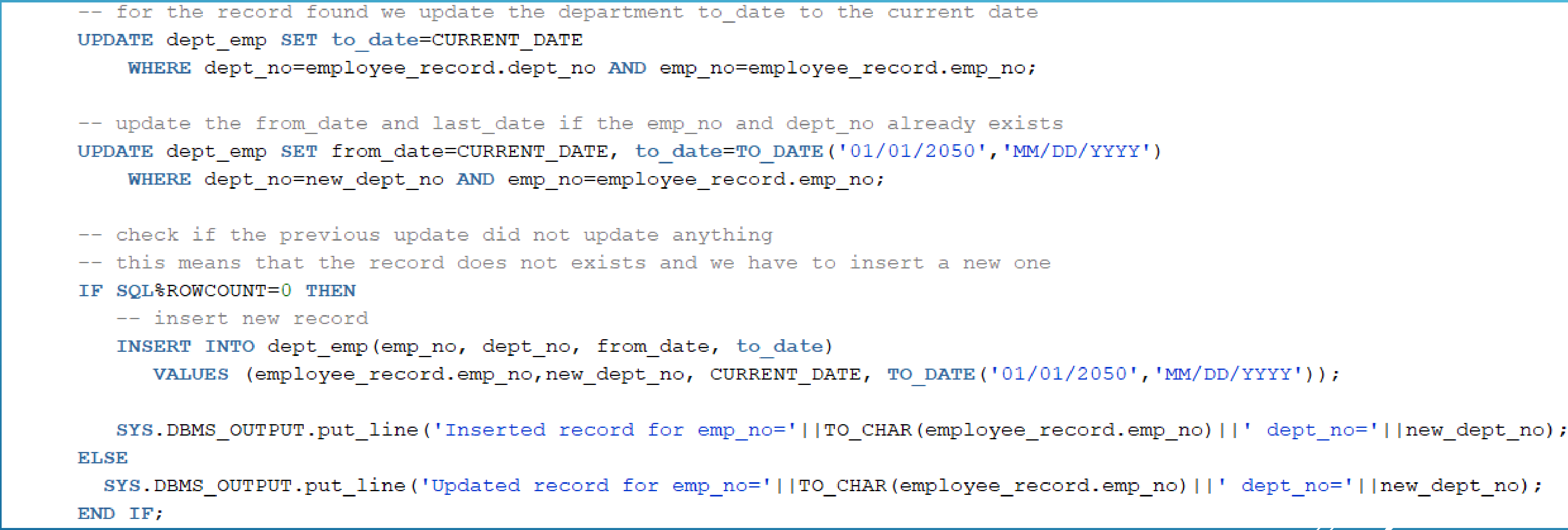


```
-- loop over all departments where managers have the given title
FOR department_record IN title_departments(manager_title)
LOOP
  -- if the found department is the same as the new department do nothing
  IF department_record.dept_no<>new_dept_no THEN
      -- loop over all employees for the department given by the outer loop
      FOR employee_record IN dept_employees(department_record.dept_no)
      LOOP
```

PL/SQL CURSORS

Cursor example.

PART 4. Loop code

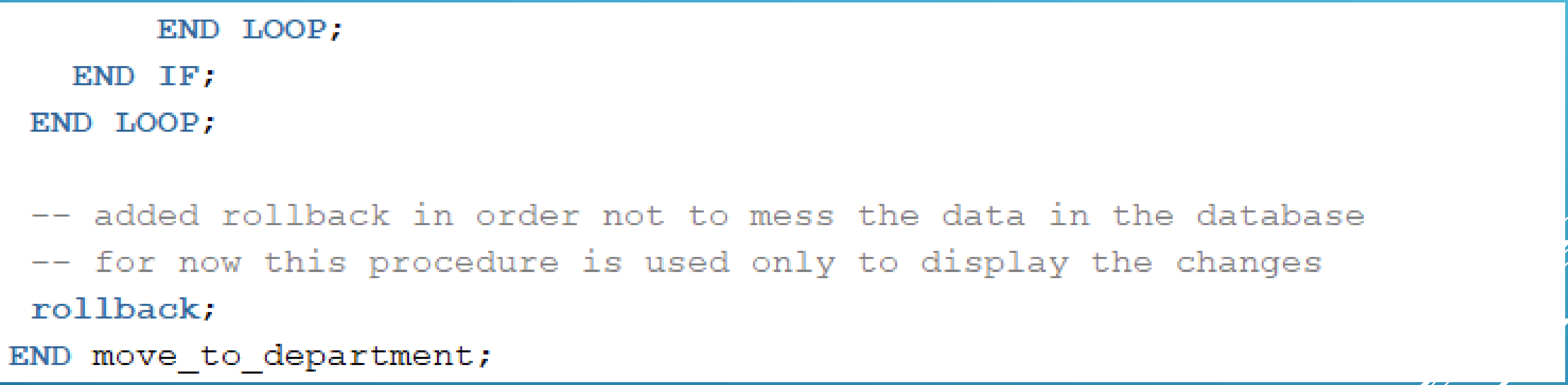


```
-- for the record found we update the department to_date to the current date
UPDATE dept_emp SET to_date=CURRENT_DATE
    WHERE dept_no=employee_record.dept_no AND emp_no=employee_record.emp_no;

-- update the from_date and last_date if the emp_no and dept_no already exists
UPDATE dept_emp SET from_date=CURRENT_DATE, to_date=TO_DATE('01/01/2050','MM/DD/YYYY')
    WHERE dept_no=new_dept_no AND emp_no=employee_record.emp_no;

-- check if the previous update did not update anything
-- this means that the record does not exists and we have to insert a new one
IF SQL%ROWCOUNT=0 THEN
   -- insert new record
   INSERT INTO dept_emp(emp_no, dept_no, from_date, to_date)
      VALUES (employee_record.emp_no,new_dept_no, CURRENT_DATE, TO_DATE('01/01/2050','MM/DD/YYYY'));

   SYS.DBMS_OUTPUT.put_line('Inserted record for emp_no='||TO_CHAR(employee_record.emp_no)||' dept_no='||new_dept_no);
ELSE
  SYS.DBMS_OUTPUT.put_line('Updated record for emp_no='||TO_CHAR(employee_record.emp_no)||' dept_no='||new_dept_no);
END IF;
```

PL/SQL CURSORS

Cursor example.

PART 5. Closing the loops and the procedure

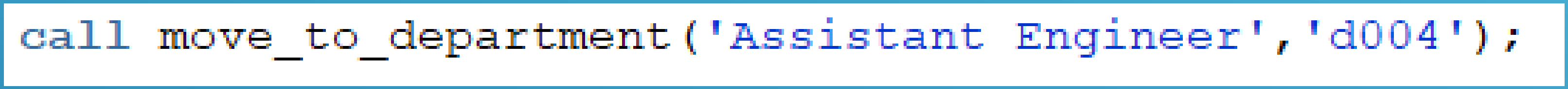


```
        END LOOP;
    END IF;
  END LOOP;

  -- added rollback in order not to mess the data in the database
  -- for now this procedure is used only to display the changes
  rollback;
END move_to_department;
```

PL/SQL CURSORS

Cursor example.

PART 6. Sample call



```
call move_to_department('Assistant Engineer','d004');
```

PL/SQL CURSORS

Cursor example – no cursor.

Let us now solve the same problem without using a cursor.
Problem: Create procedure move_to_department_sql with 2 parameters manager_title and new_dept_no , that will move all employees for the department managed by an employee whose latest title is manager_title to the department new_dept_no.

PL/SQL CURSORS

Cursor example – no cursor.

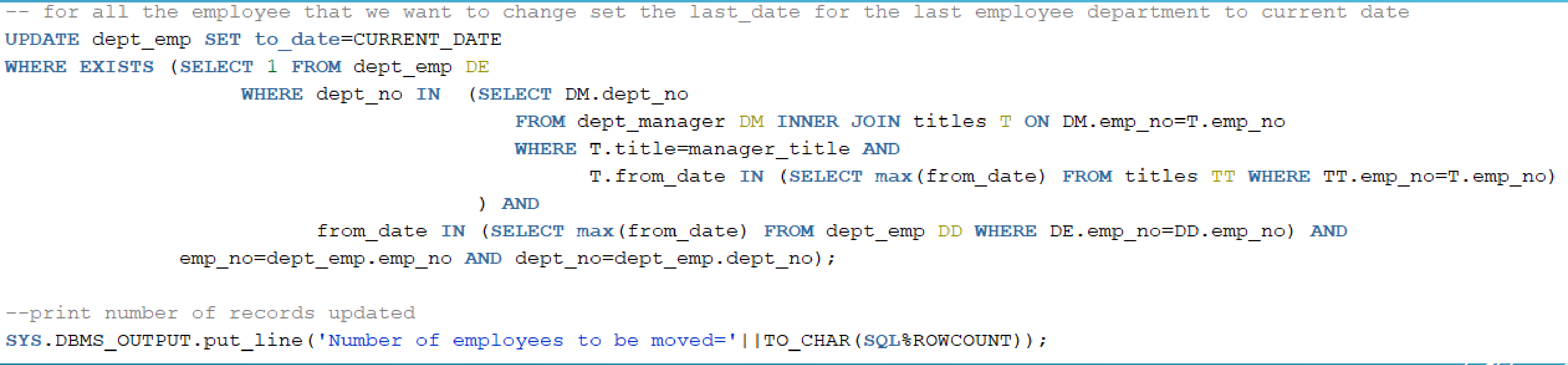
PART 1. Procedure header

```
CREATE OR REPLACE PROCEDURE move_to_department_sql(
   manager_title IN titles.title%TYPE,    -- the same type as title field in titles table
   new_dept_no departments.dept_no%TYPE) -- the same type as dept_no in the departments table
AS

BEGIN
```

PL/SQL CURSORS

Cursor example – no cursor.

PART 2. First update

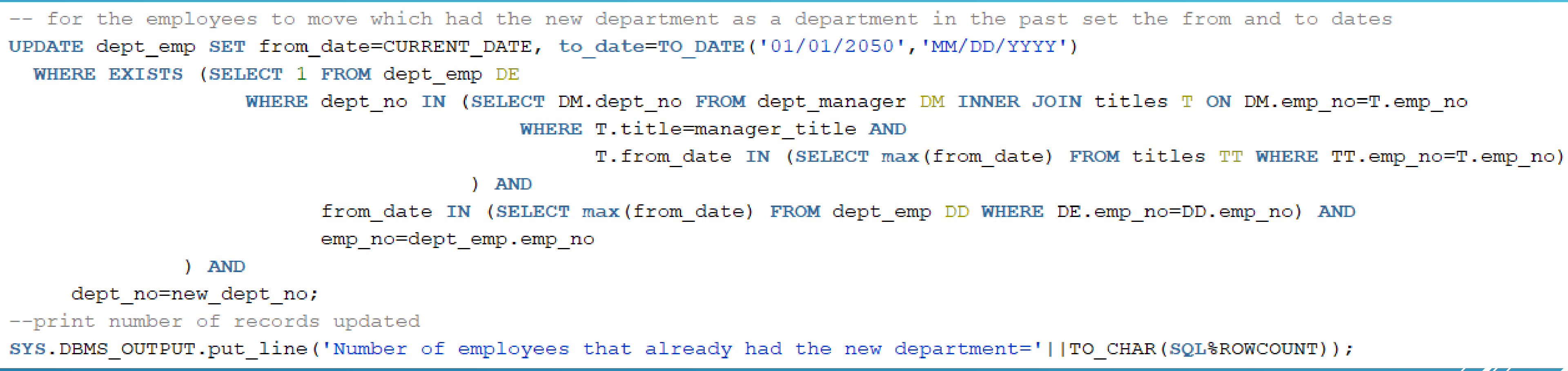


```
-- for all the employee that we want to change set the last_date for the last employee department to current date
UPDATE dept_emp SET to_date=CURRENT_DATE
WHERE EXISTS (SELECT 1 FROM dept_emp DE
                   WHERE dept_no IN  (SELECT DM.dept_no
                                         FROM dept_manager DM INNER JOIN titles T ON DM.emp_no=T.emp_no
                                         WHERE T.title=manager_title AND
                                               T.from_date IN (SELECT max(from_date) FROM titles TT WHERE TT.emp_no=T.emp_no)
                                      ) AND
                         from_date IN (SELECT max(from_date) FROM dept_emp DD WHERE DE.emp_no=DD.emp_no) AND
              emp_no=dept_emp.emp_no AND dept_no=dept_emp.dept_no);

--print number of records updated
SYS.DBMS_OUTPUT.put_line('Number of employees to be moved='||TO_CHAR(SQL%ROWCOUNT));
```

PL/SQL CURSORS

Cursor example – no cursor.

PART 3. Second update

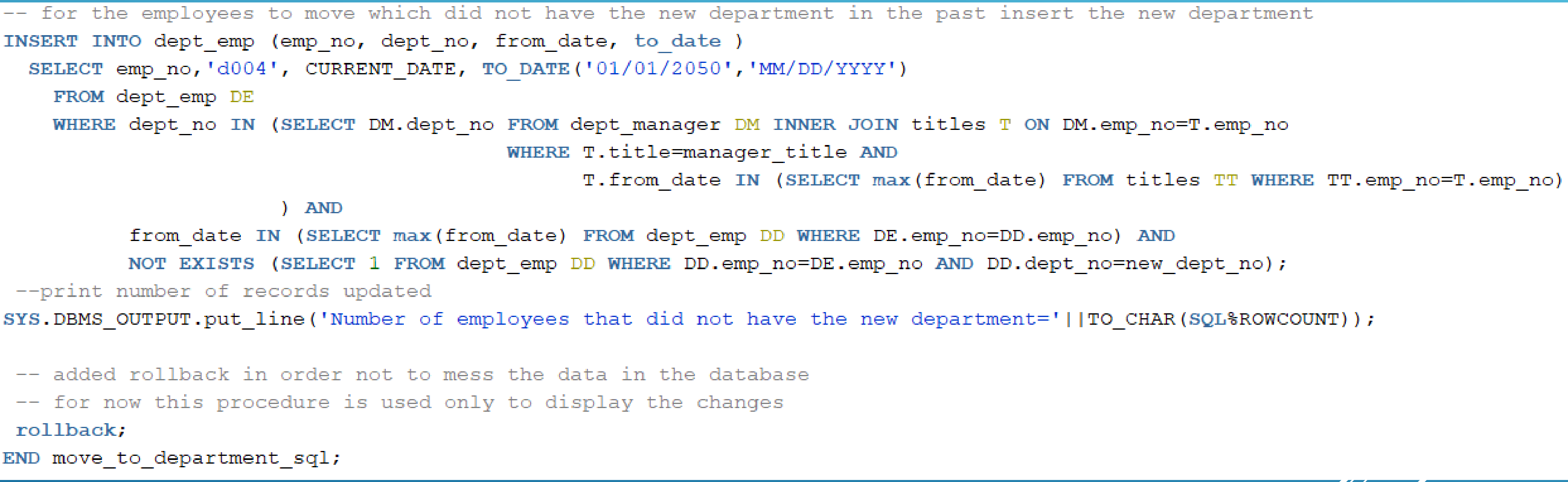


```
-- for the employees to move which had the new department as a department in the past set the from and to dates
UPDATE dept_emp SET from_date=CURRENT_DATE, to_date=TO_DATE('01/01/2050','MM/DD/YYYY')
  WHERE EXISTS (SELECT 1 FROM dept_emp DE
                 WHERE dept_no IN (SELECT DM.dept_no FROM dept_manager DM INNER JOIN titles T ON DM.emp_no=T.emp_no
                                    WHERE T.title=manager_title AND
                                          T.from_date IN (SELECT max(from_date) FROM titles TT WHERE TT.emp_no=T.emp_no)
                                   ) AND
                       from_date IN (SELECT max(from_date) FROM dept_emp DD WHERE DE.emp_no=DD.emp_no) AND
                       emp_no=dept_emp.emp_no
              ) AND
     dept_no=new_dept_no;
--print number of records updated
SYS.DBMS_OUTPUT.put_line('Number of employees that already had the new department='||TO_CHAR(SQL%ROWCOUNT));
```

PL/SQL CURSORS

Cursor example – no cursor.

PART 4. Inserts for the new department and procedure ending



```
-- for the employees to move which did not have the new department in the past insert the new department
INSERT INTO dept_emp (emp_no, dept_no, from_date, to_date )
  SELECT emp_no,'d004', CURRENT_DATE, TO_DATE('01/01/2050','MM/DD/YYYY')
    FROM dept_emp DE
    WHERE dept_no IN (SELECT DM.dept_no FROM dept_manager DM INNER JOIN titles T ON DM.emp_no=T.emp_no
                                        WHERE T.title=manager_title AND
                                              T.from_date IN (SELECT max(from_date) FROM titles TT WHERE TT.emp_no=T.emp_no)
                      ) AND
          from_date IN (SELECT max(from_date) FROM dept_emp DD WHERE DE.emp_no=DD.emp_no) AND
          NOT EXISTS (SELECT 1 FROM dept_emp DD WHERE DD.emp_no=DE.emp_no AND DD.dept_no=new_dept_no);
 --print number of records updated
SYS.DBMS_OUTPUT.put_line('Number of employees that did not have the new department='||TO_CHAR(SQL%ROWCOUNT));

 -- added rollback in order not to mess the data in the database
 -- for now this procedure is used only to display the changes
 rollback;
END move_to_department_sql;
```

MORE ON PL/SQL TRIGGERS

Full example of a PL/SQL trigger.

PROBLEM: Create a trigger for the student table that will record student class attendance.

To running commit we run

```
SET AUTOCOMMIT ON;
```

Create Student table

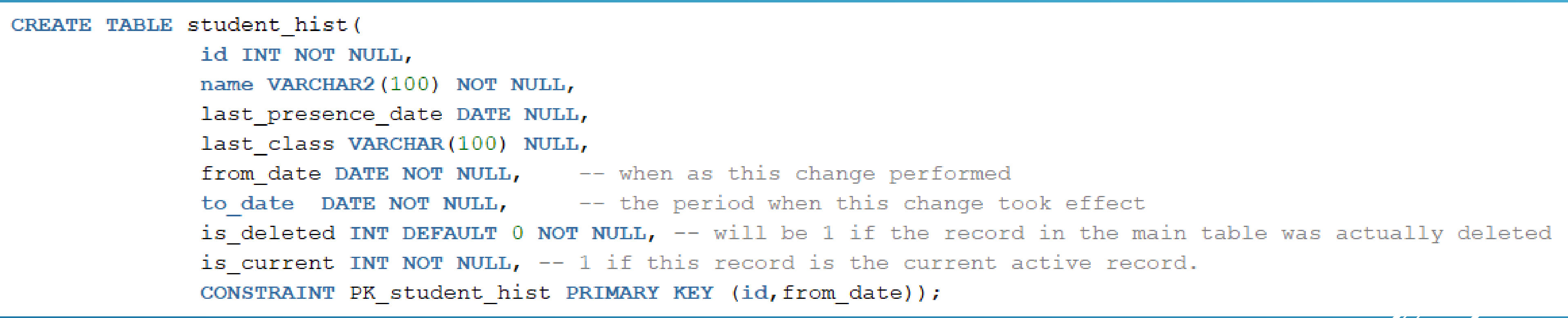
```
CREATE TABLE student(
              id INT NOT NULL,     -- student ID
              name VARCHAR2(100) NOT NULL,  -- student name
              last_presence_date DATE NULL,  -- when was last seen attending a class
              last_class VARCHAR(100) NULL,  -- last class attended
              CONSTRAINT PK_student PRIMARY KEY (id)); -- id is the primary key
```

MORE ON PL/SQL TRIGGERS

Full example of a PL/SQL trigger.

PROBLEM: Create a trigger for the student table that will record student class attendance.

Create Student Historical table



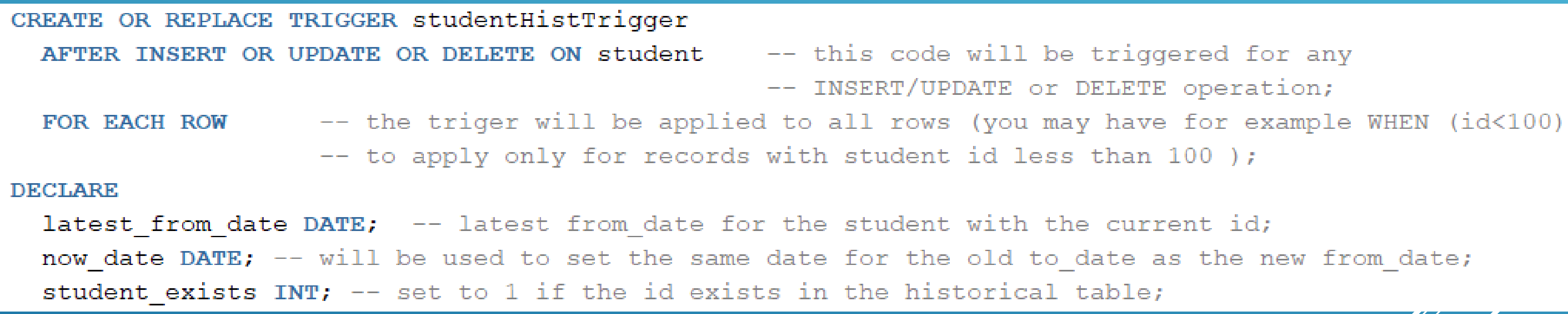
```
CREATE TABLE student_hist(
              id INT NOT NULL,
              name VARCHAR2(100) NOT NULL,
              last_presence_date DATE NULL,
              last_class VARCHAR(100) NULL,
              from_date DATE NOT NULL,    -- when as this change performed
              to_date  DATE NOT NULL,     -- the period when this change took effect
              is_deleted INT DEFAULT 0 NOT NULL, -- will be 1 if the record in the main table was actually deleted
              is_current INT NOT NULL, -- 1 if this record is the current active record.
              CONSTRAINT PK_student_hist PRIMARY KEY (id,from_date));
```

MORE ON PL/SQL TRIGGERS

Full example of a PL/SQL trigger.

PROBLEM: Create a trigger for the student table that will record student class attendance.

Trigger header



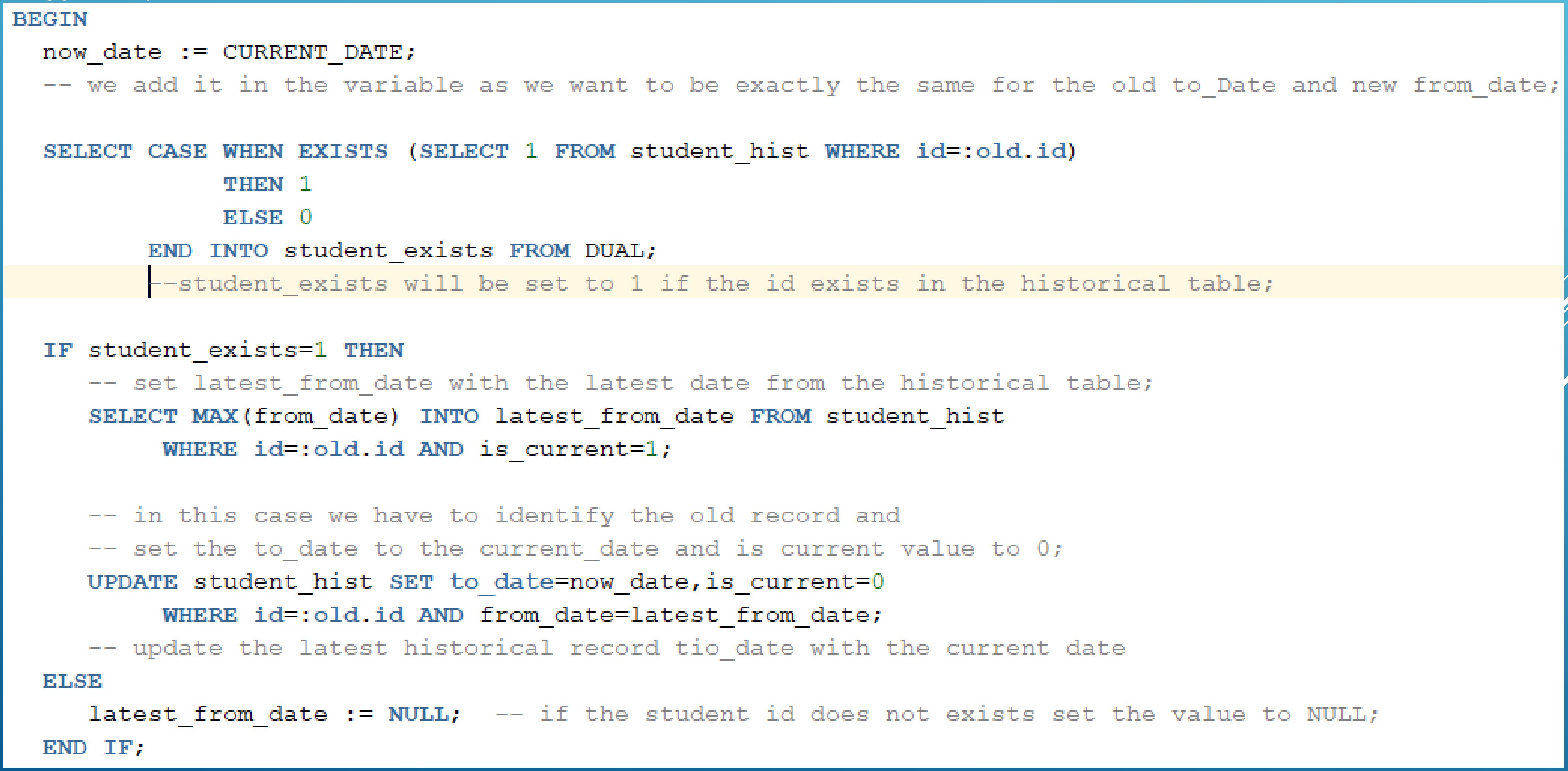
```
CREATE OR REPLACE TRIGGER studentHistTrigger
  AFTER INSERT OR UPDATE OR DELETE ON student    -- this code will be triggered for any
                                                 -- INSERT/UPDATE or DELETE operation;
  FOR EACH ROW        -- the triger will be applied to all rows (you may have for example WHEN (id<100)
                      -- to apply only for records with student id less than 100 );
DECLARE
  latest_from_date DATE;  -- latest from_date for the student with the current id;
  now_date DATE; -- will be used to set the same date for the old to_date as the new from_date;
  student_exists INT; -- set to 1 if the id exists in the historical table;
```

MORE ON PL/SQL TRIGGERS

Full example of a PL/SQL trigger.

PROBLEM: Create a trigger for the student table that will record student class attendance.

Trigger body (PART 1)



```
BEGIN
  now_date := CURRENT_DATE;
  -- we add it in the variable as we want to be exactly the same for the old to_Date and new from_date;

  SELECT CASE WHEN EXISTS (SELECT 1 FROM student_hist WHERE id=:old.id)
              THEN 1
              ELSE 0
         END INTO student_exists FROM DUAL;
         --student_exists will be set to 1 if the id exists in the historical table;

  IF student_exists=1 THEN
     -- set latest_from_date with the latest date from the historical table;
     SELECT MAX(from_date) INTO latest_from_date FROM student_hist
          WHERE id=:old.id AND is_current=1;

     -- in this case we have to identify the old record and
     -- set the to_date to the current_date and is current value to 0;
     UPDATE student_hist SET to_date=now_date,is_current=0
          WHERE id=:old.id AND from_date=latest_from_date;
     -- update the latest historical record tio_date with the current date
  ELSE
     latest_from_date := NULL;  -- if the student id does not exists set the value to NULL;
  END IF;
```

MORE ON PL/SQL TRIGGERS

Full example of a PL/SQL trigger.

PROBLEM: Create a trigger for the student table that will record student class attendance.

Trigger body (PART 2)

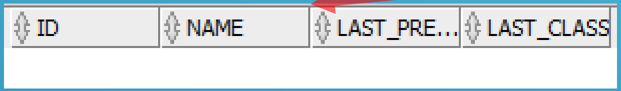
```
  IF DELETING THEN  -- if the record in the student table is deleted
   -- insert in historical table the deleted record;
   INSERT INTO student_hist(id, name, last_presence_date, last_class, from_date, to_date,
                              is_deleted, is_current)
     VALUES  (:old.id, :old.name, :old.last_presence_date, :old.last_class,
             now_date, TO_DATE('01/01/2100','MM/DD/YYYY'), 1, 1);
  END IF;

  IF INSERTING OR UPDATING THEN
      -- insert a record with the new changes into the historical table
      -- note that we add 2100 as the to date, this means that the record is current.
      INSERT INTO student_hist(id, name, last_presence_date, last_class, from_date, to_date,
                                  is_deleted, is_current)
          VALUES (:new.id, :new.name, :new.last_presence_date, :new.last_class,
                   now_date, TO_DATE('01/01/2100','MM/DD/YYYY'), 0, 1);
  END IF;

END studentHistTrigger;
```

MORE ON PL/SQL TRIGGERS

Testing the trigger.

Select from the student table.

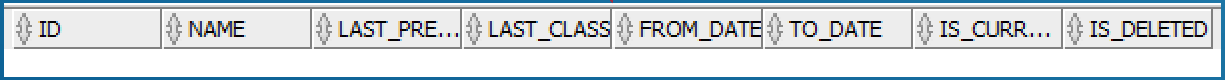
```
SELECT ID, NAME, LAST_PRESENCE_DATE, LAST_CLASS FROM student WHERE id=1;
```



ID	NAME	LAST_PRE...	LAST_CLASS

Select from the student_hist table.

```
SELECT
   id,
   name,
   last_presence_date,
   last_class,
   TO_CHAR(from_date,'MM/DD/YYYY HH:MI:SS') as from_date,
   TO_CHAR(to_date,'MM/DD/YYYY HH:MI:SS') as to_date,
   is_current,
   is_deleted
FROM student_hist
WHERE id=1;
```



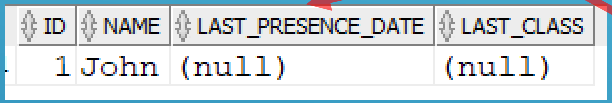
ID	NAME	LAST_PRE...	LAST_CLASS	FROM_DATE	TO_DATE	IS_CURR...	IS_DELETED

MORE ON PL/SQL TRIGGERS

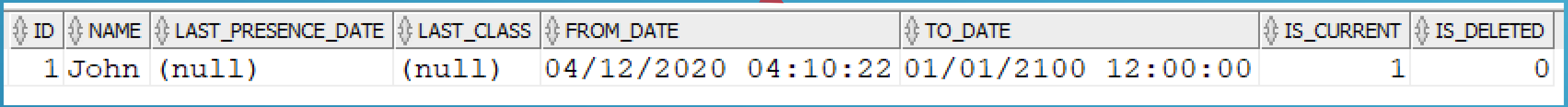
Full example of a PL/SQL trigger.

Adding a new student. For now the student didn't attend any class.

```
INSERT INTO student(id, name) VALUES (1, 'John');
```



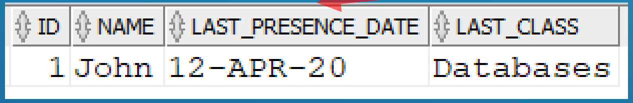
ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS
1	John	(null)	(null)



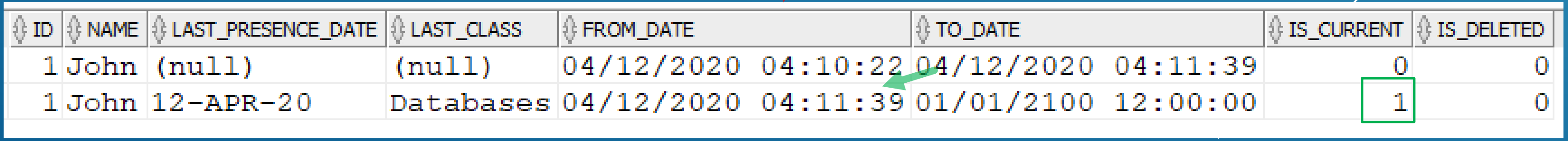
ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS	FROM_DATE	TO_DATE	IS_CURRENT	IS_DELETED
1	John	(null)	(null)	04/12/2020 04:10:22	01/01/2100 12:00:00	1	0

Update the student as he attends the database class.

```
UPDATE student SET LAST_PRESENCE_DATE=CURRENT_DATE, LAST_CLASS='Databases' WHERE id=1;
```



ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS
1	John	12-APR-20	Databases



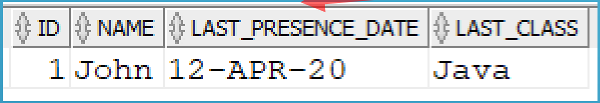
ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS	FROM_DATE	TO_DATE	IS_CURRENT	IS_DELETED
1	John	(null)	(null)	04/12/2020 04:10:22	04/12/2020 04:11:39	0	0
1	John	12-APR-20	Databases	04/12/2020 04:11:39	01/01/2100 12:00:00	1	0

MORE ON PL/SQL TRIGGERS

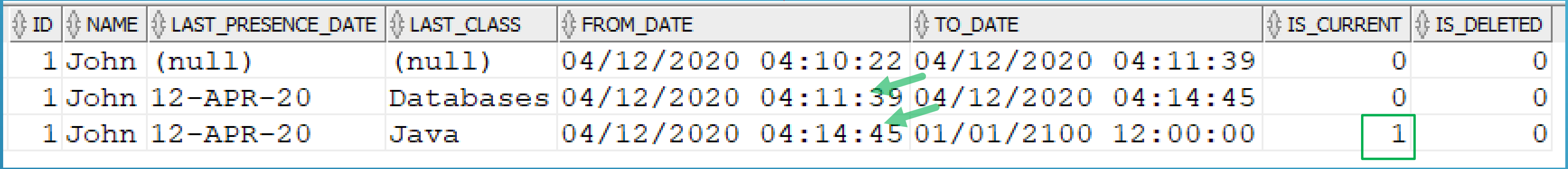
Full example of a PL/SQL trigger.

Update the student as he attends the Java class.

```
UPDATE student SET LAST_PRESENCE_DATE=CURRENT_DATE, LAST_CLASS='Java' WHERE id=1;
```



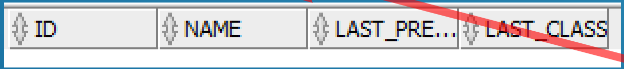
ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS
1	John	12-APR-20	Java



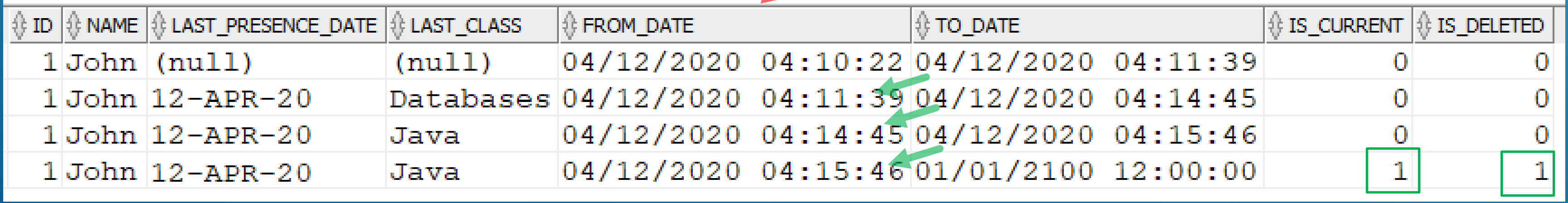
ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS	FROM_DATE	TO_DATE	IS_CURRENT	IS_DELETED
1	John	(null)	(null)	04/12/2020 04:10:22	04/12/2020 04:11:39	0	0
1	John	12-APR-20	Databases	04/12/2020 04:11:39	04/12/2020 04:14:45	0	0
1	John	12-APR-20	Java	04/12/2020 04:14:45	01/01/2100 12:00:00	1	0

Delete the student.

```
DELETE student WHERE id=1;
```



ID	NAME	LAST_PRE...	LAST_CLASS

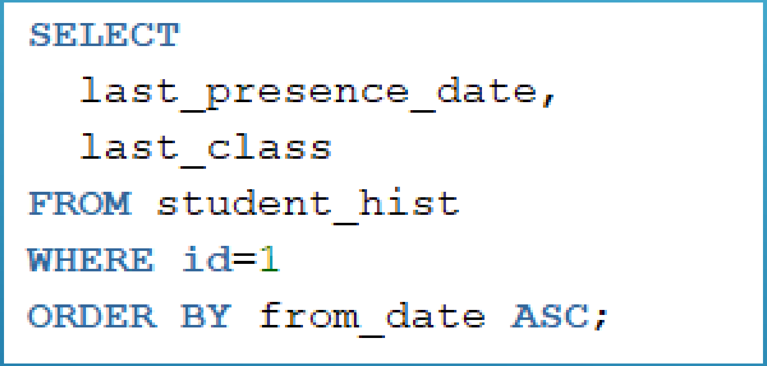


ID	NAME	LAST_PRESENCE_DATE	LAST_CLASS	FROM_DATE	TO_DATE	IS_CURRENT	IS_DELETED
1	John	(null)	(null)	04/12/2020 04:10:22	04/12/2020 04:11:39	0	0
1	John	12-APR-20	Databases	04/12/2020 04:11:39	04/12/2020 04:14:45	0	0
1	John	12-APR-20	Java	04/12/2020 04:14:45	04/12/2020 04:15:46	0	0
1	John	12-APR-20	Java	04/12/2020 04:15:46	01/01/2100 12:00:00	1	1

MORE ON PL/SQL TRIGGERS

Full example of a PL/SQL trigger.

With this we can easily find all the classes (together with the date) attended by student with id 1.



```
SELECT
  last_presence_date,
  last_class
FROM student_hist
WHERE id=1
ORDER BY from_date ASC;
```