PL/SQL CONCEPTS

Unit– 10

Prepared By-
Rutika Patel

Basics of PL / SQL(procedural language extension to Structured Query Language)

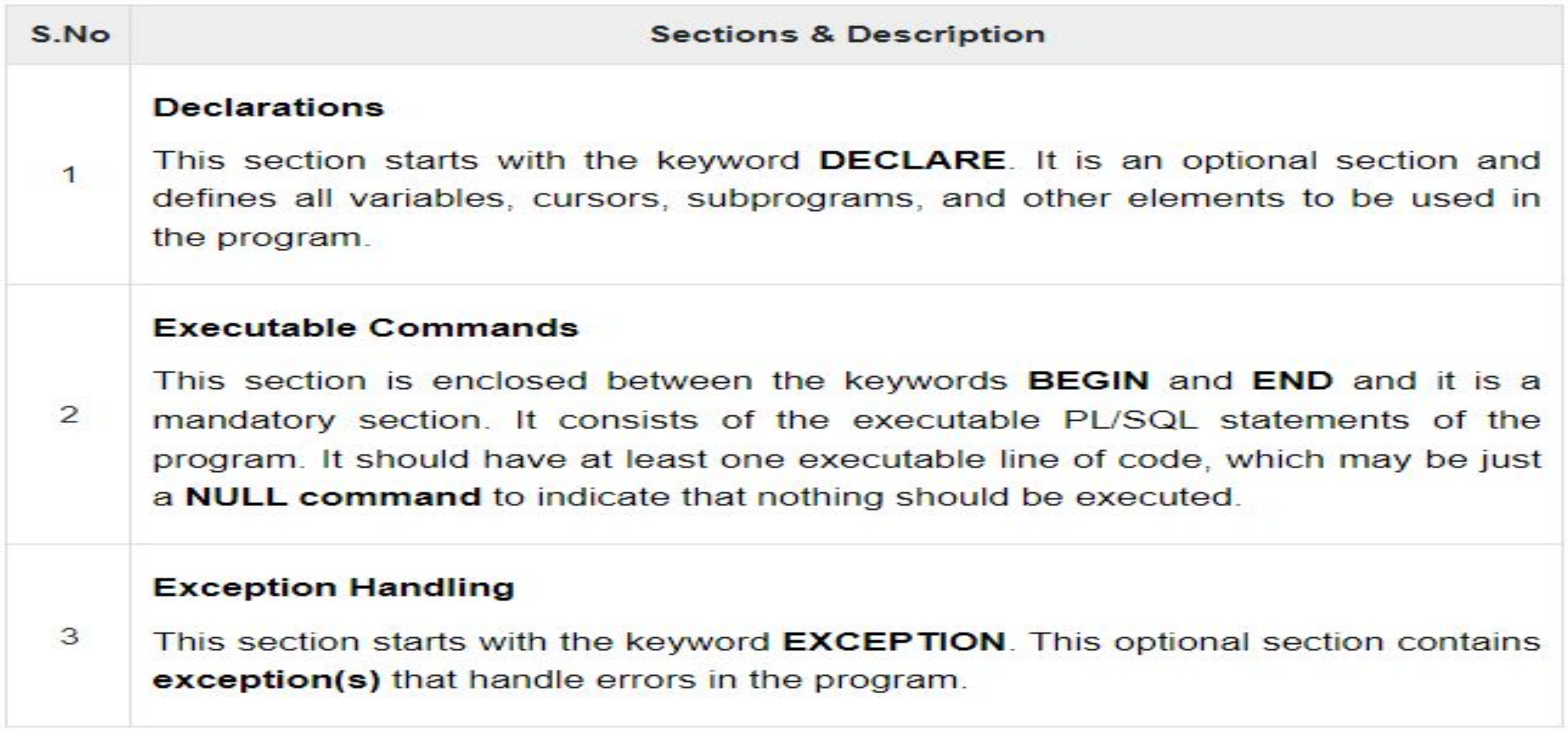
- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are certain notable facts about PL/SQL −
- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line **SQL*Plus interface**.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that Pascal programming language.
- Apart from Oracle, PL/SQL is available in **TimesTen in-memory database** and **IBM DB2**.

Features of PL/SQL

- PL/SQL has the following features −
- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

PL/SQL - Basic Syntax

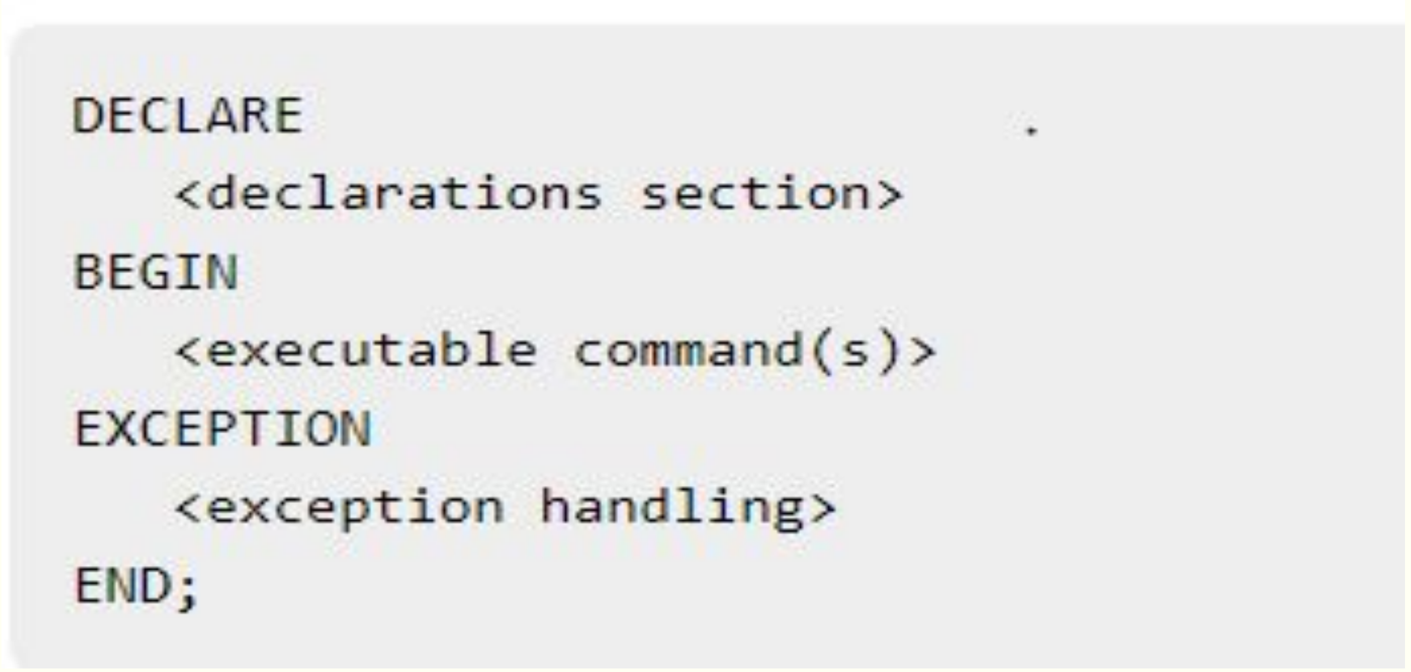
- Basic Syntax of PL/SQL which is a **block-structured** language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts −



S.No	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL command** to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

Cont…

- Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Following is the basic structure of a PL/SQL block −



```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```

PL/SQL Procedure

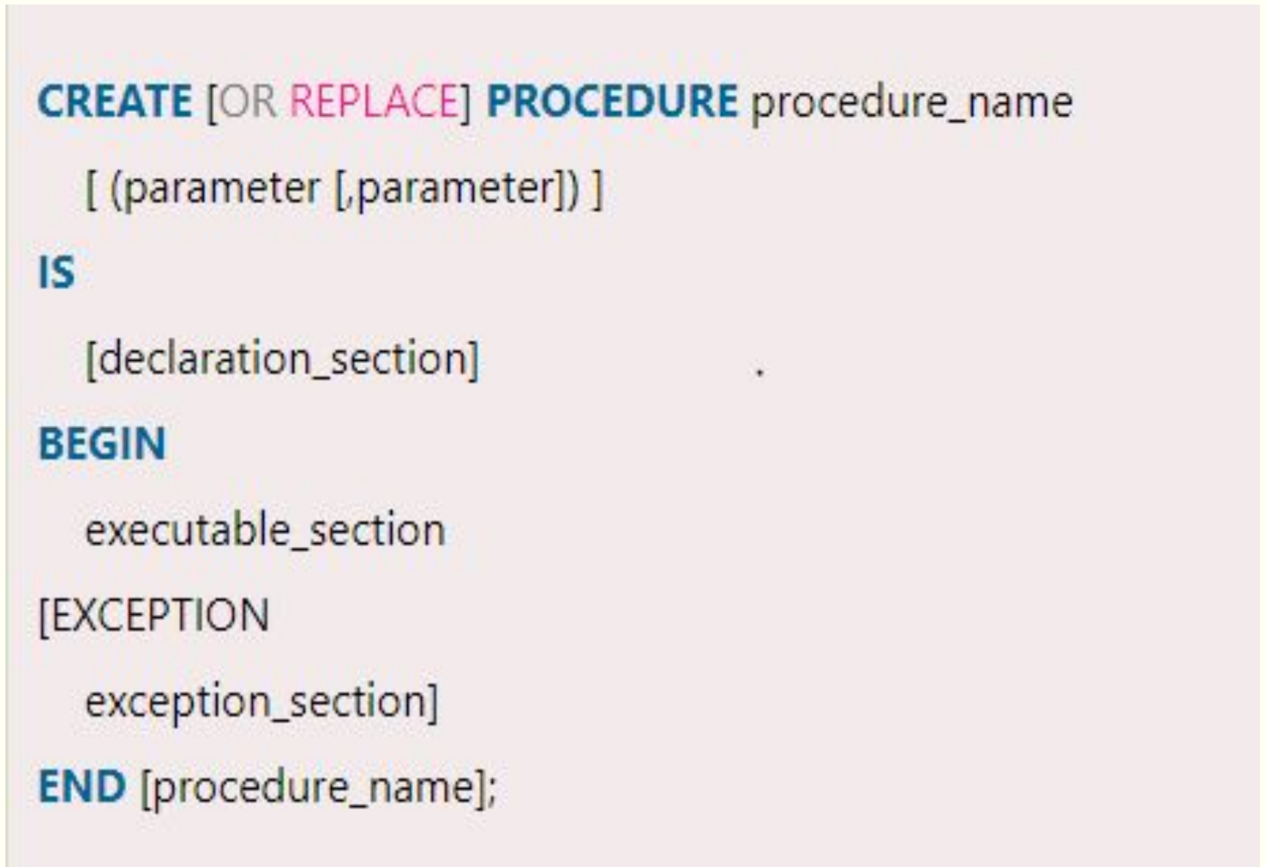
- The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.
- The procedure contains a header and a body.
- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

Cont…

- How to pass parameters in procedure:
- When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:
- **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
- **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
- **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

Cont…

- PL/SQL Create Procedure
- **Syntax for creating procedure:**

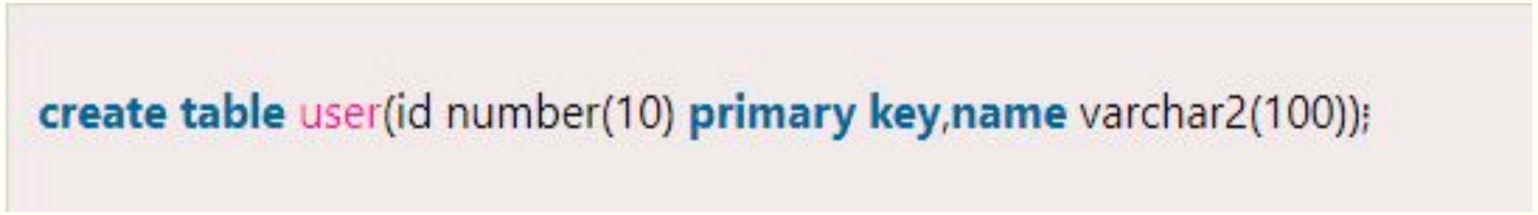


```
CREATE [OR REPLACE] PROCEDURE procedure_name
    [ (parameter [,parameter]) ]
IS
    [declaration_section]
BEGIN
    executable_section
[EXCEPTION
    exception_section]
END [procedure_name];
```

Cont…

Create procedure example

- In this example, we are going to insert record in user table. So you need to create user table first.
- **Table creation:**

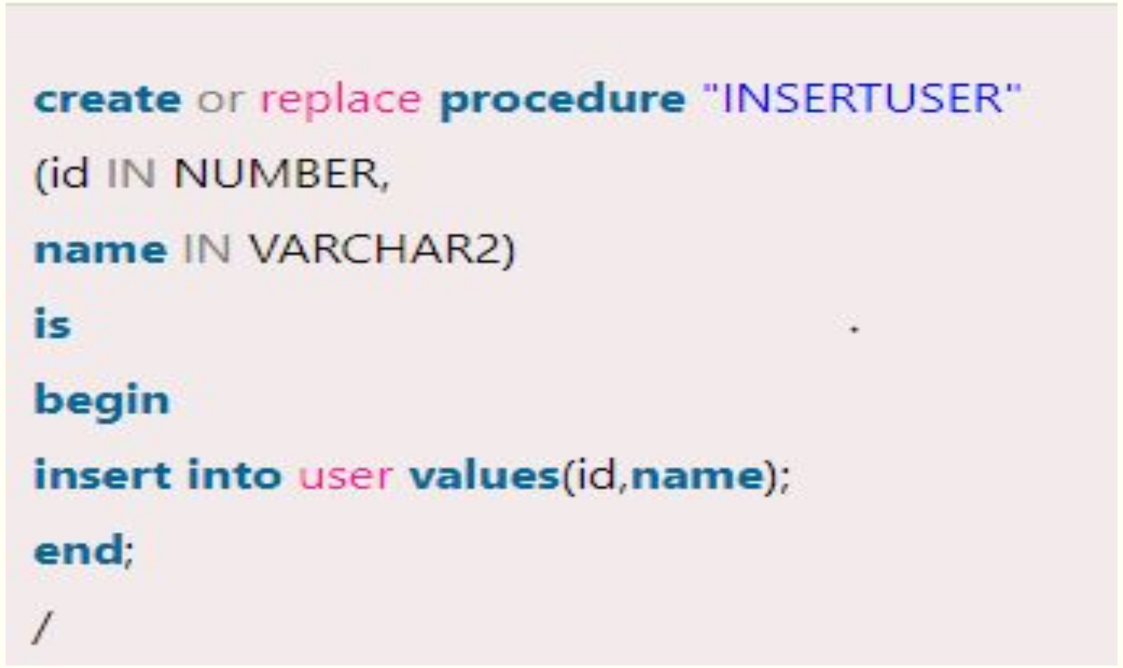


```
create table user(id number(10) primary key,name varchar2(100));
```

Cont…

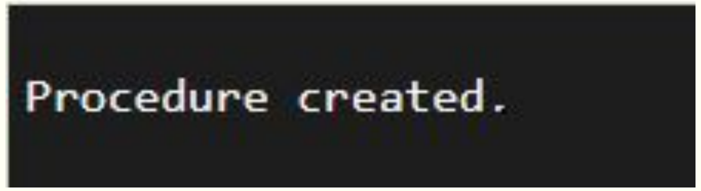
Now write the procedure code to insert record in user table.

Procedure Code:



```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:

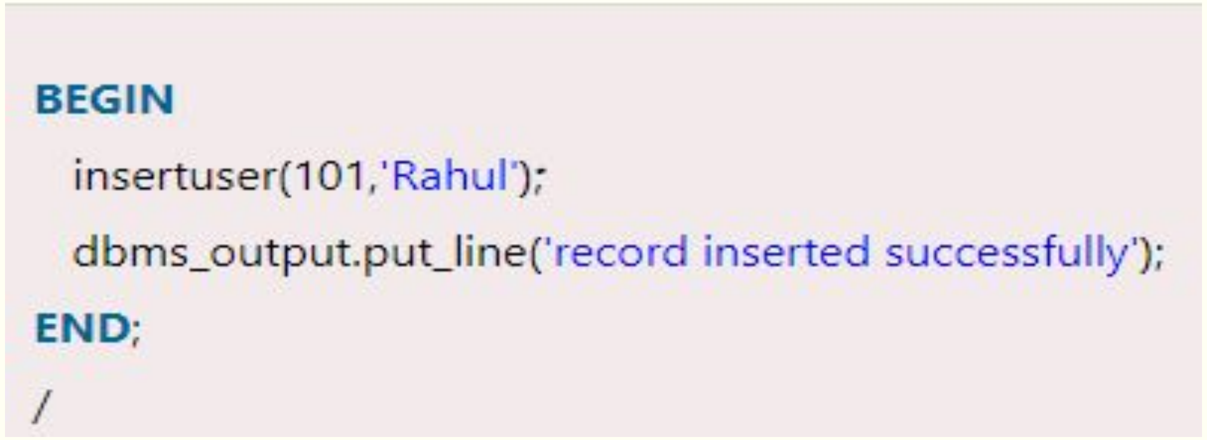


```
Procedure created.
```

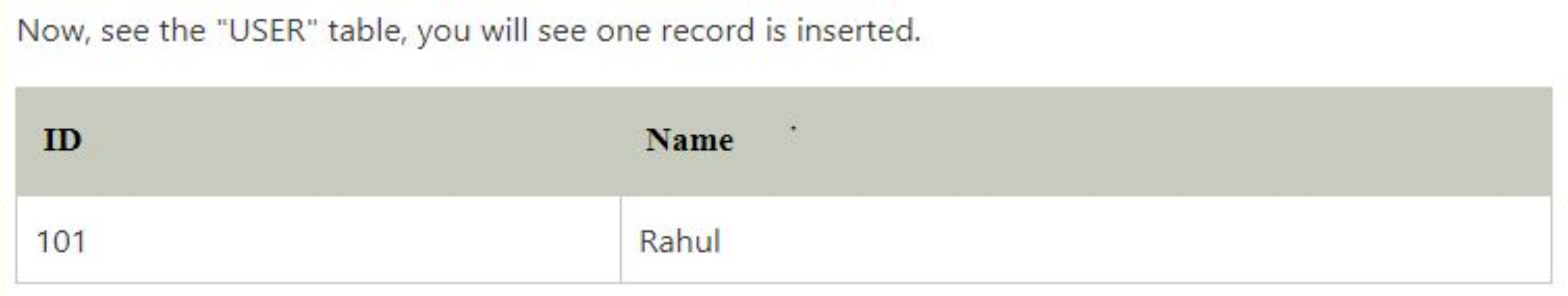
Cont…

PL/SQL program to call procedure

- Let's see the code to call above created procedure.



```
BEGIN
   insertuser(101,'Rahul');
   dbms_output.put_line('record inserted successfully');
END;
/
```

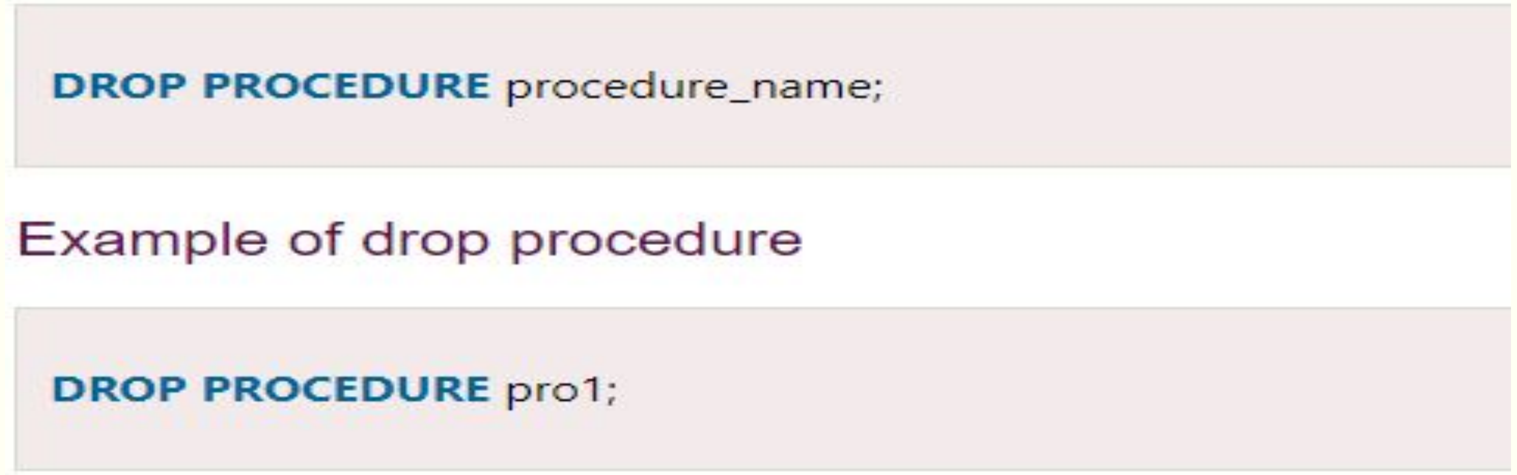


Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

Cont…

- PL/SQL Drop Procedure
- **Syntax for drop procedure**



```
DROP PROCEDURE procedure_name;
```

Example of drop procedure

```
DROP PROCEDURE pro1;
```

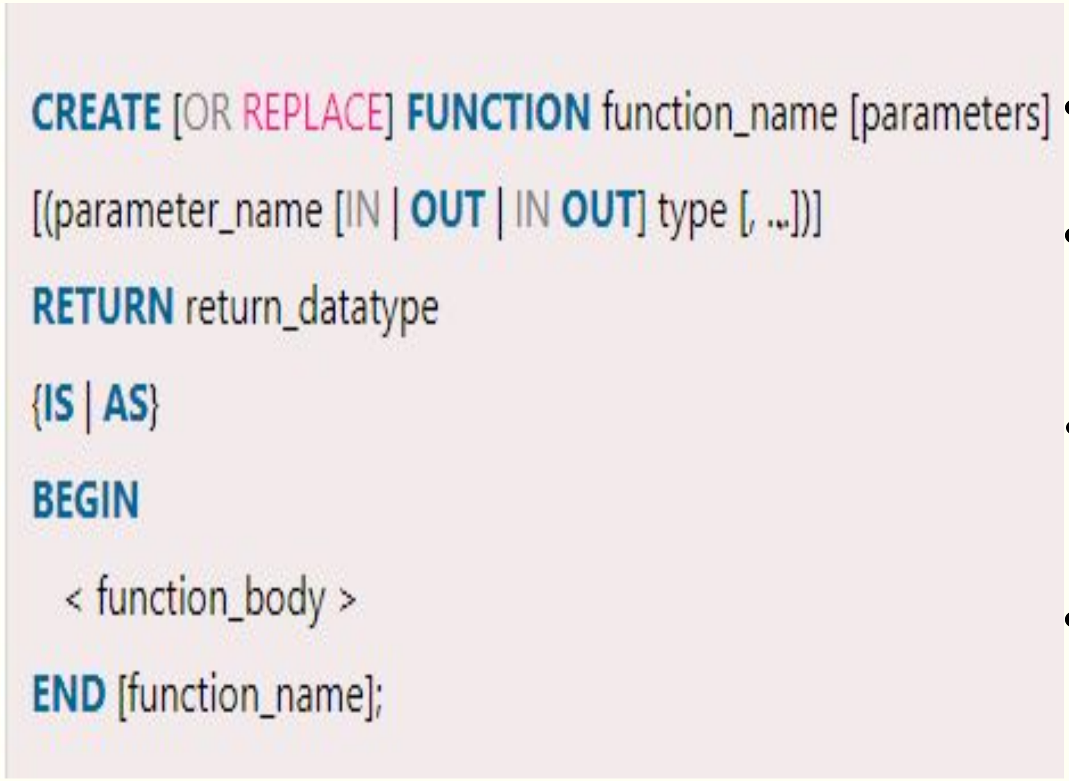
Advantages of stored procedure

- **Security**:- We can **improve security** by giving rights to selected persons only.
- **Faster Execution**:- It is **precompiled** so **compilation** of procedure is **not required every time** you call it.
- **Sharing of code**:- Once procedure is created and stored, it can be **used by more than one user**.
- **Productivity**:- Code written in procedure is **shared by all programmers**. This eliminates redundant coding by multiple programmers so overall improvement in productivity.

PL/SQL Function

- The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

- **Syntax to create a function:**



```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Here:

•**Function_name:** specifies the name of the function.

•**[OR REPLACE]** option allows modifying an existing function.

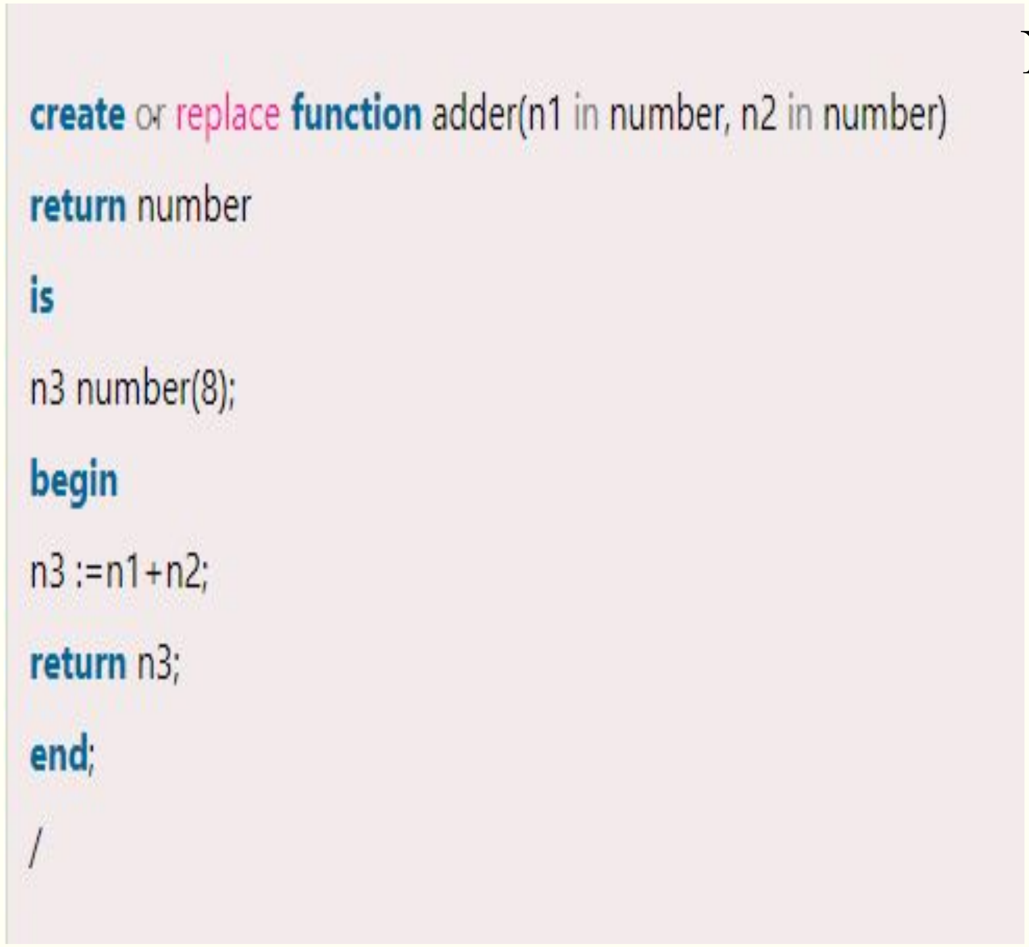
•The **optional parameter list** contains name, mode and types of the parameters.

•**IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

Cont…

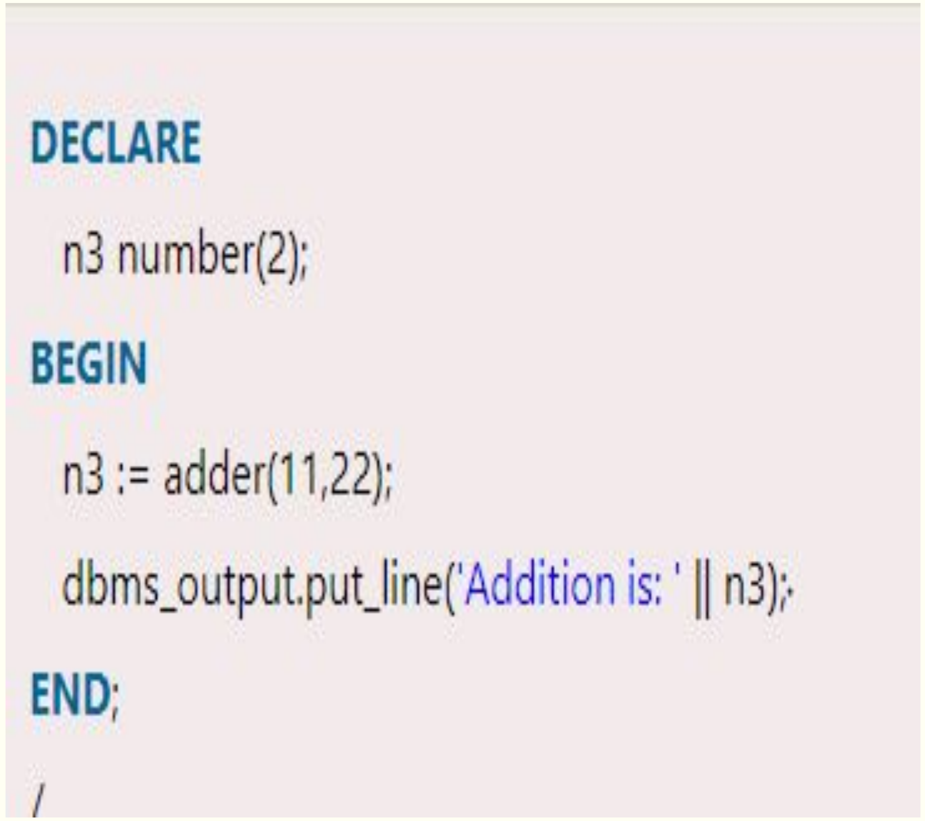
- The function must contain a return statement.
- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

PL/SQL Function Example



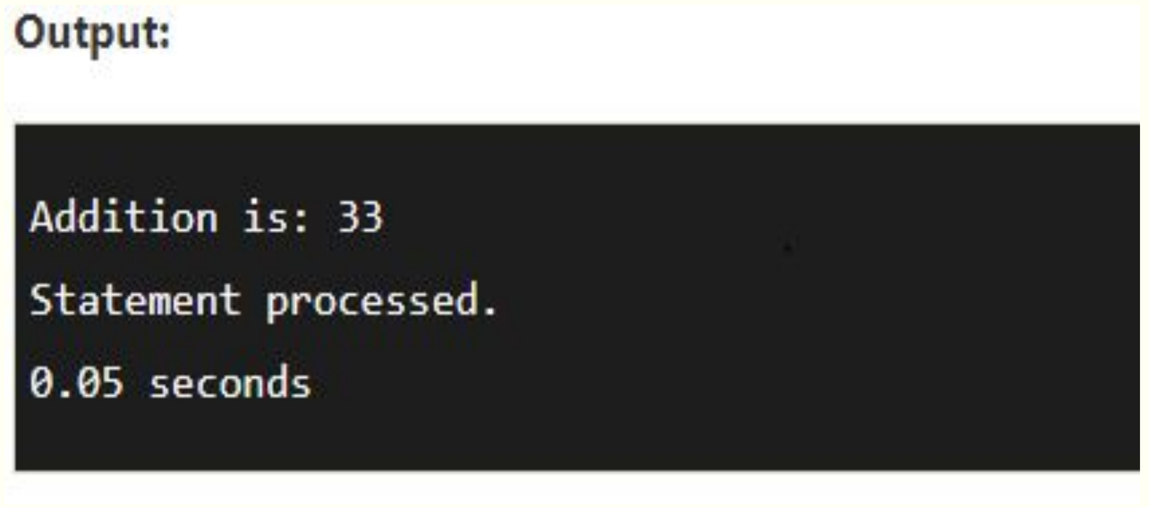
```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

Now write another program to **call the function**.



```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
```

Cont…


Output:

```
Addition is: 33

Statement processed.

0.05 seconds
```

PL/SQL Trigger

- Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.
- Triggers are stored programs, which are automatically executed or fired when some event occurs.
- Triggers are written to be executed in response to any of the following events.
- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- Triggers could be defined on the table, view, schema, or database with which the event is associated.

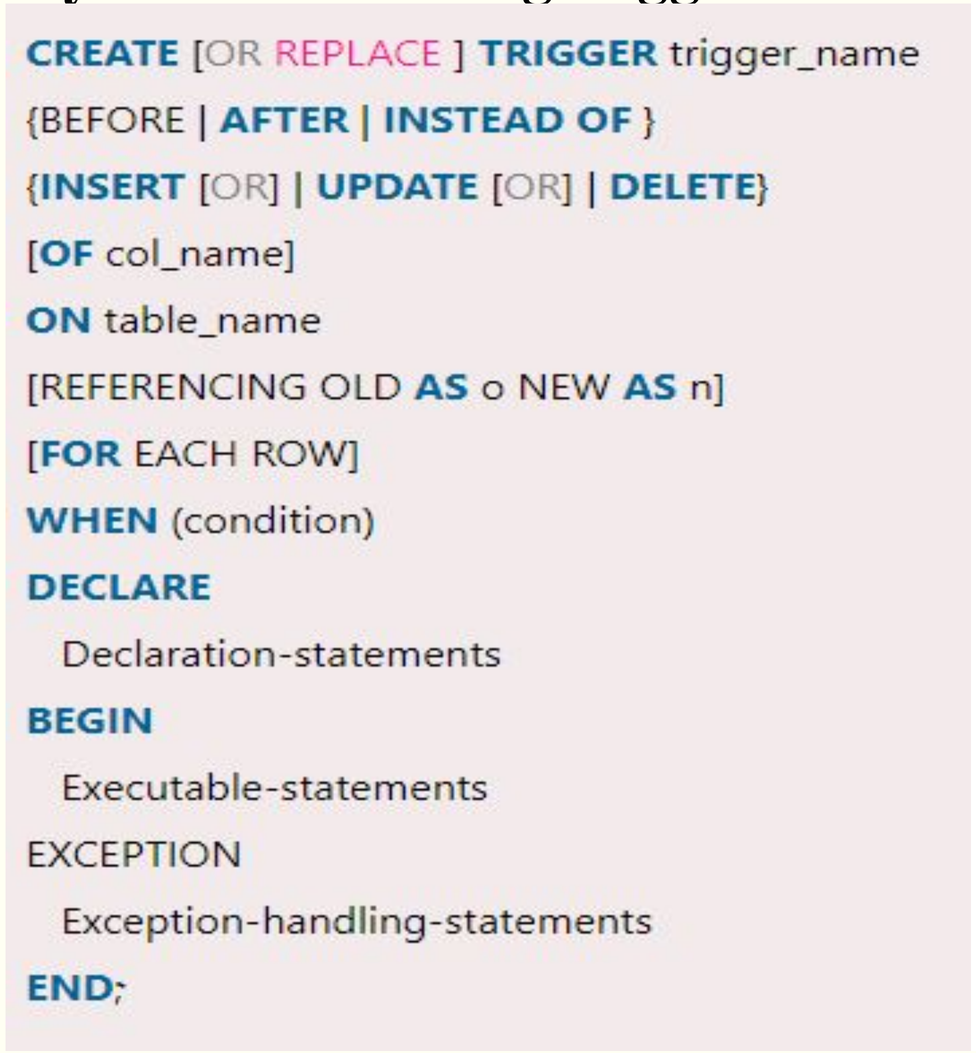
Advantages of Triggers

These are the following advantages of Triggers:

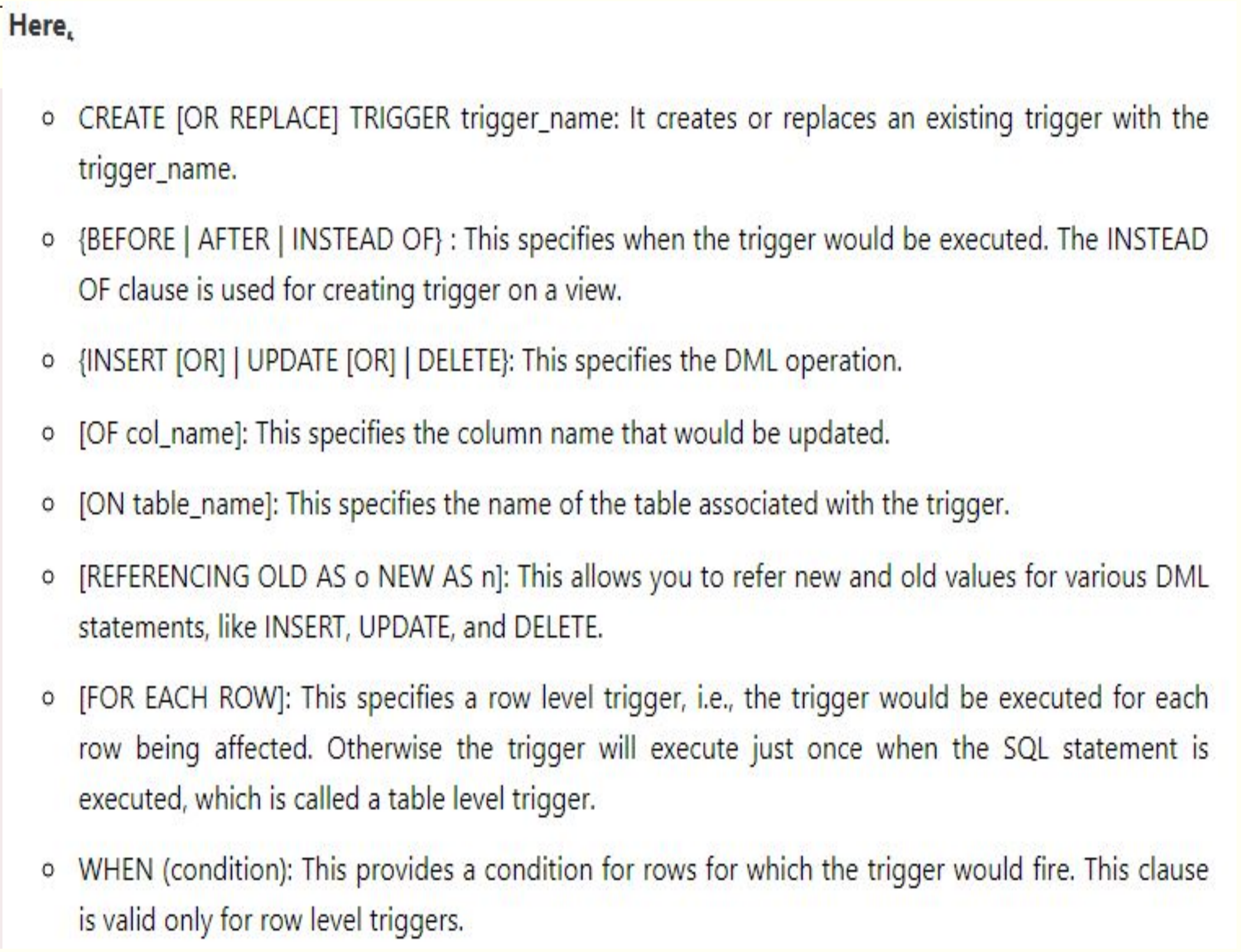
- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger

- **Syntax for creating trigger:**



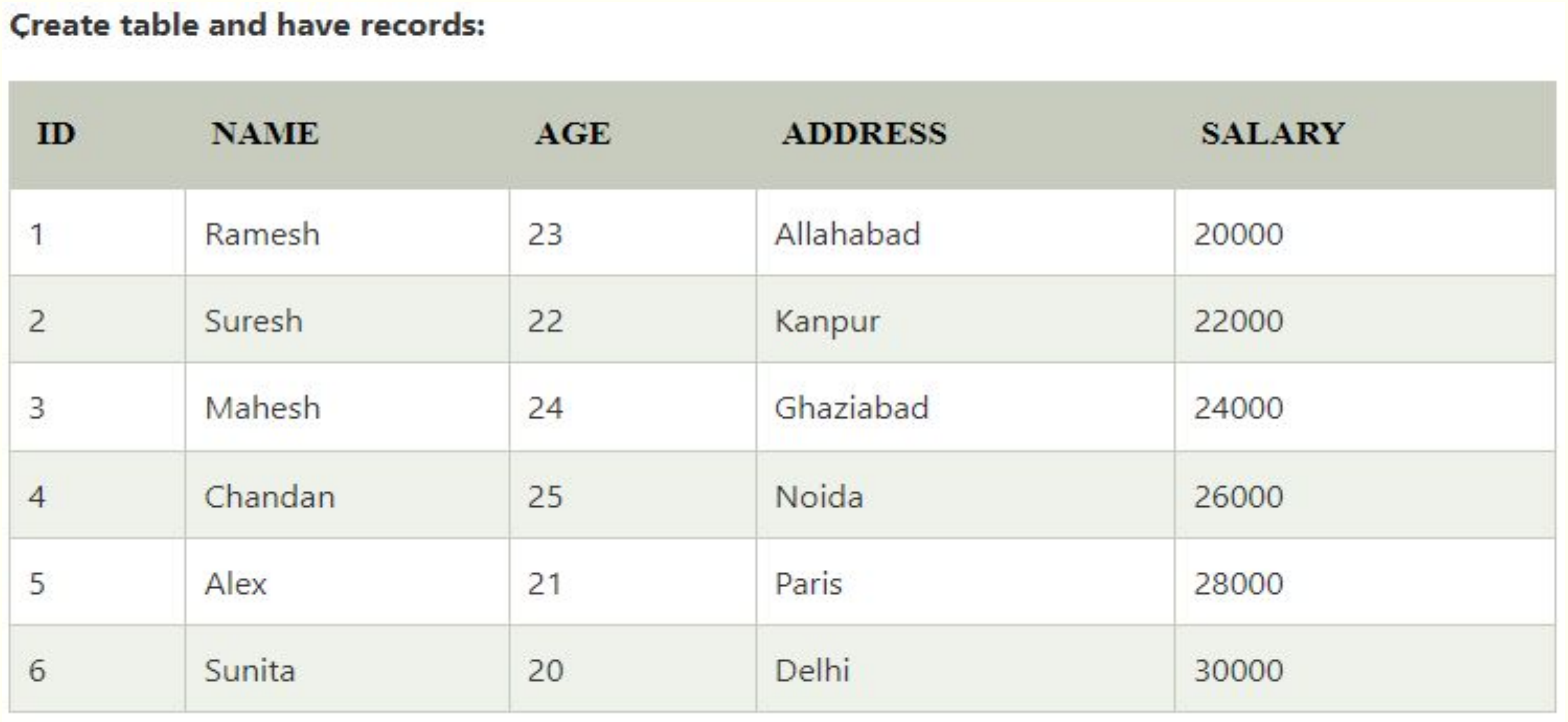
```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```



Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example



Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Cont…

- **Create trigger:**
- Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

Cont…

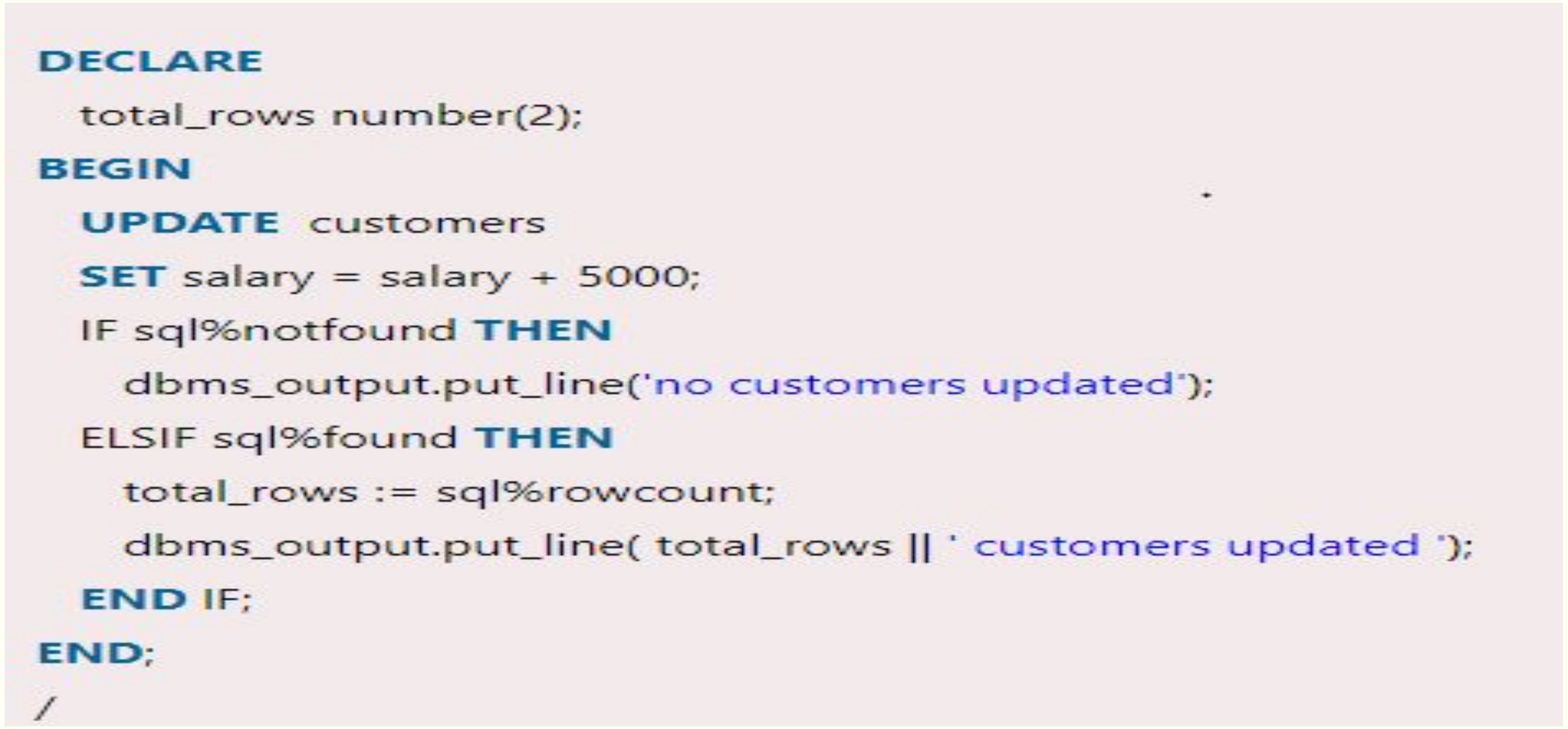
```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

Cont…

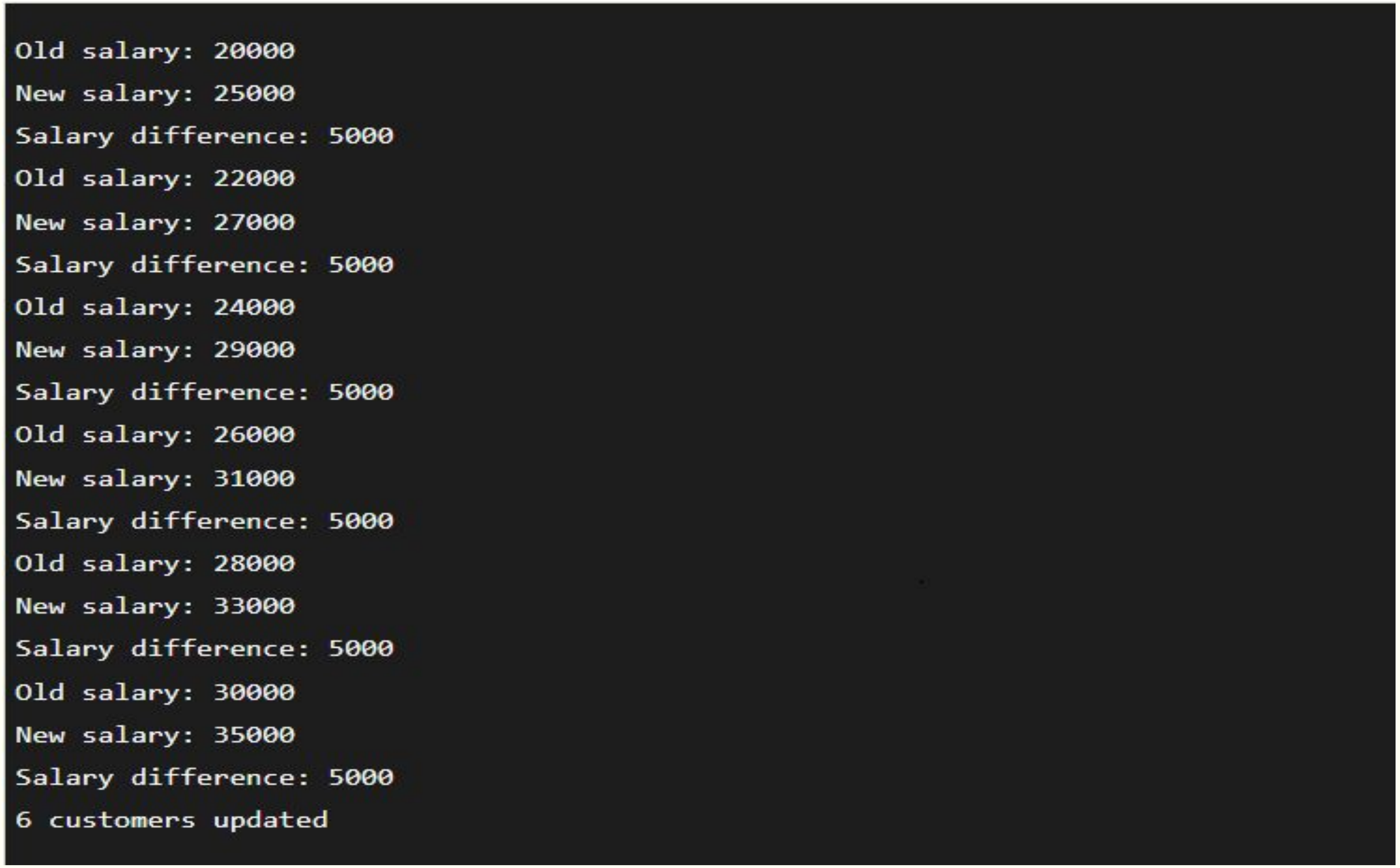
- **Check the salary difference by procedure:**
- Use the following code to get the old salary, new salary and salary difference after the trigger created.



```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Cont…

- **Output:**



```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Cont…

- **Important Points:**

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not avialable for table level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

cursor

- A cursor in SQL is a temporary work area created in system memory when a SQL statement is executed. A SQL cursor is a set of rows together with a pointer that identifies a current row. It is a database object to retrieve data from a result set one row at a time. It is useful when we want to manipulate the record of a table in a singleton method, in other words, one row at a time. In other words, a cursor can hold more than one row but can process only one row at a time. The set of rows the cursor holds is called the active set.
- Types of Cursors in SQL
  - There are the following two types of cursors in SQL:
    - Implicit Cursor
    - Explicit Cursor

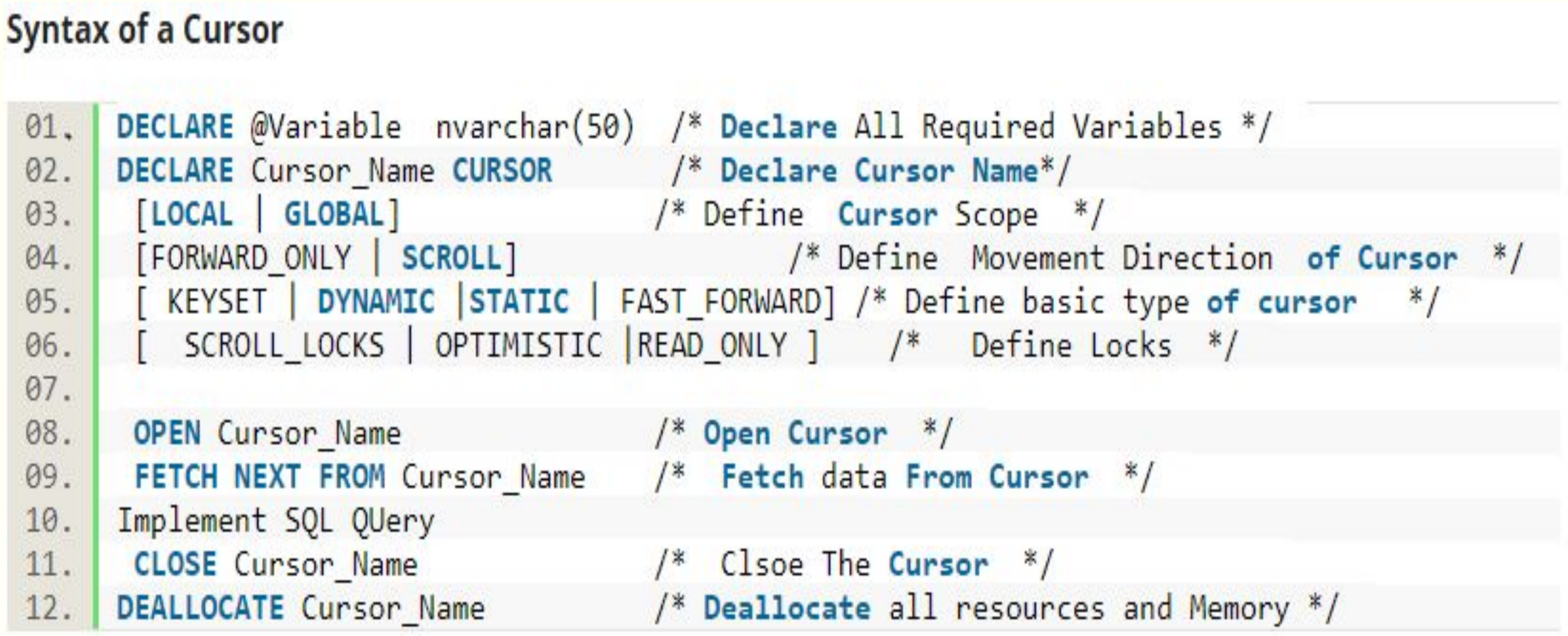
Cont…

 Main components of Cursors

Each cursor contains the followings 5 parts,

- **Declare Cursor:** In this part, we declare variables and return a set of values.
- **Open:** This is the entering part of the cursor.
- **Fetch:** Used to retrieve the data row by row from a cursor.
- **Close:** This is an exit part of the cursor and used to close a cursor.
- **Deallocate:** In this part, we delete the cursor definition and release all the system resources associated with the cursor.

Cont…



Syntax of a Cursor

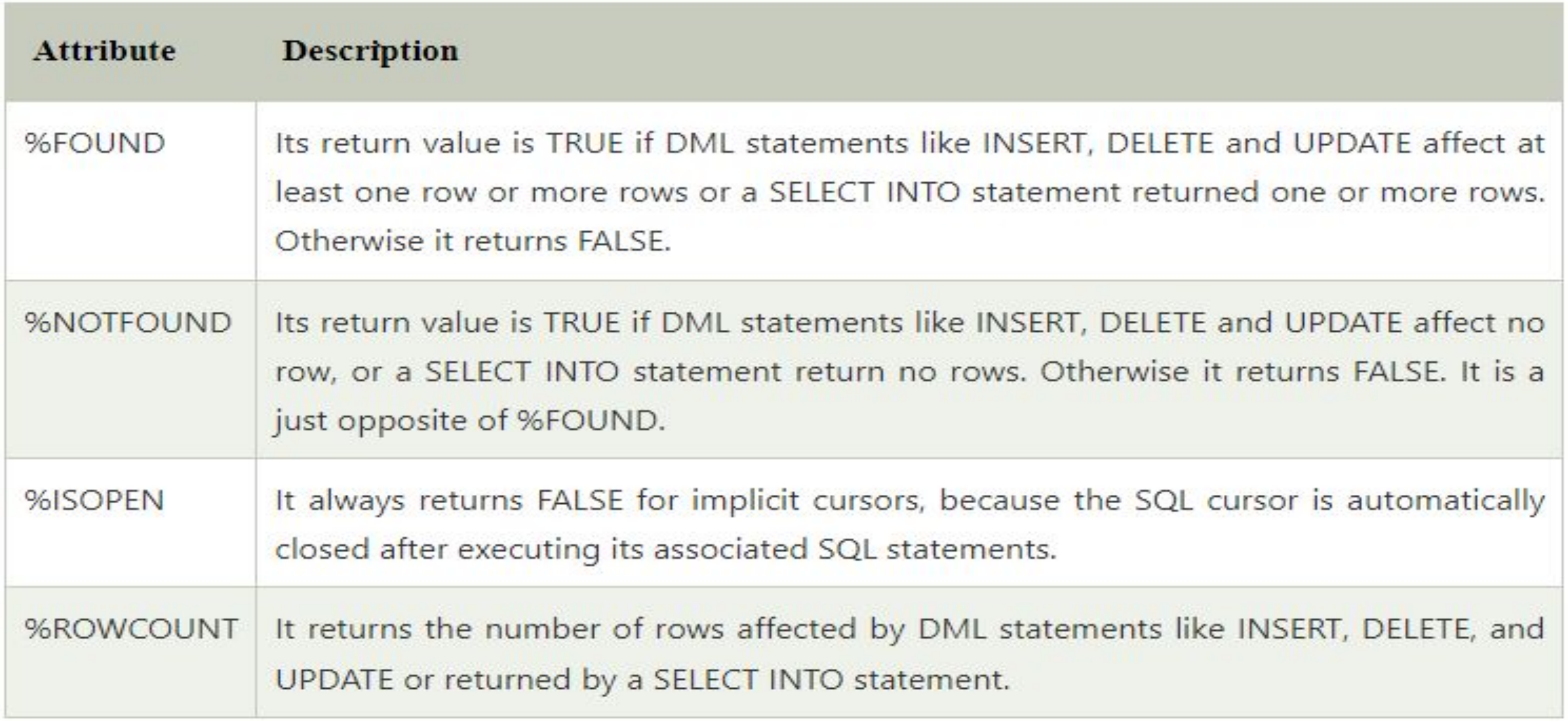
```
DECLARE @Variable  nvarchar(50)  /* Declare All Required Variables */
DECLARE Cursor_Name CURSOR        /* Declare Cursor Name*/
 [LOCAL | GLOBAL]                /* Define  Cursor Scope  */
 [FORWARD_ONLY | SCROLL]                 /* Define  Movement Direction  of Cursor  */
 [ KEYSET | DYNAMIC |STATIC | FAST_FORWARD] /* Define basic type of cursor   */
 [  SCROLL_LOCKS | OPTIMISTIC |READ_ONLY ]    /*   Define Locks  */

 OPEN Cursor_Name                /* Open Cursor  */
 FETCH NEXT FROM Cursor_Name     /*  Fetch data From Cursor  */
Implement SQL QUery
 CLOSE Cursor_Name               /*  Clsoe The Cursor  */
DEALLOCATE Cursor_Name           /* Deallocate all resources and Memory */
```

PL/SQL Implicit Cursors

- The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.
- These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.
- Orcale provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.
- **For example:** When you execute the SQL statements like INSERT, UPDATE, DELETE then the cursor attributes tell whether any rows are affected and how many have been affected. If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

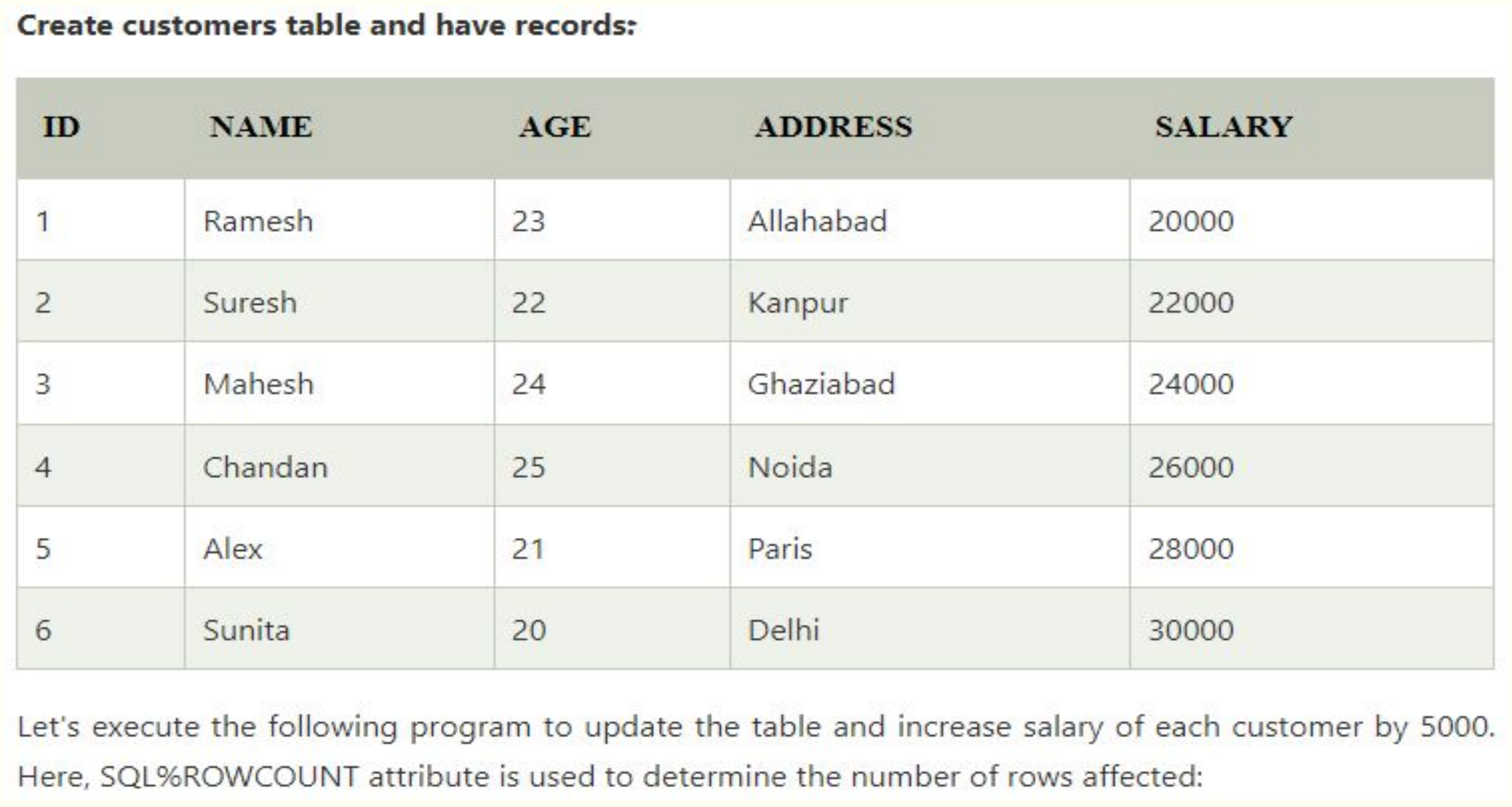
Cont…

- The following table soecifies the status of the cursor with each of its attribute.



Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

PL/SQL Implicit Cursor Example

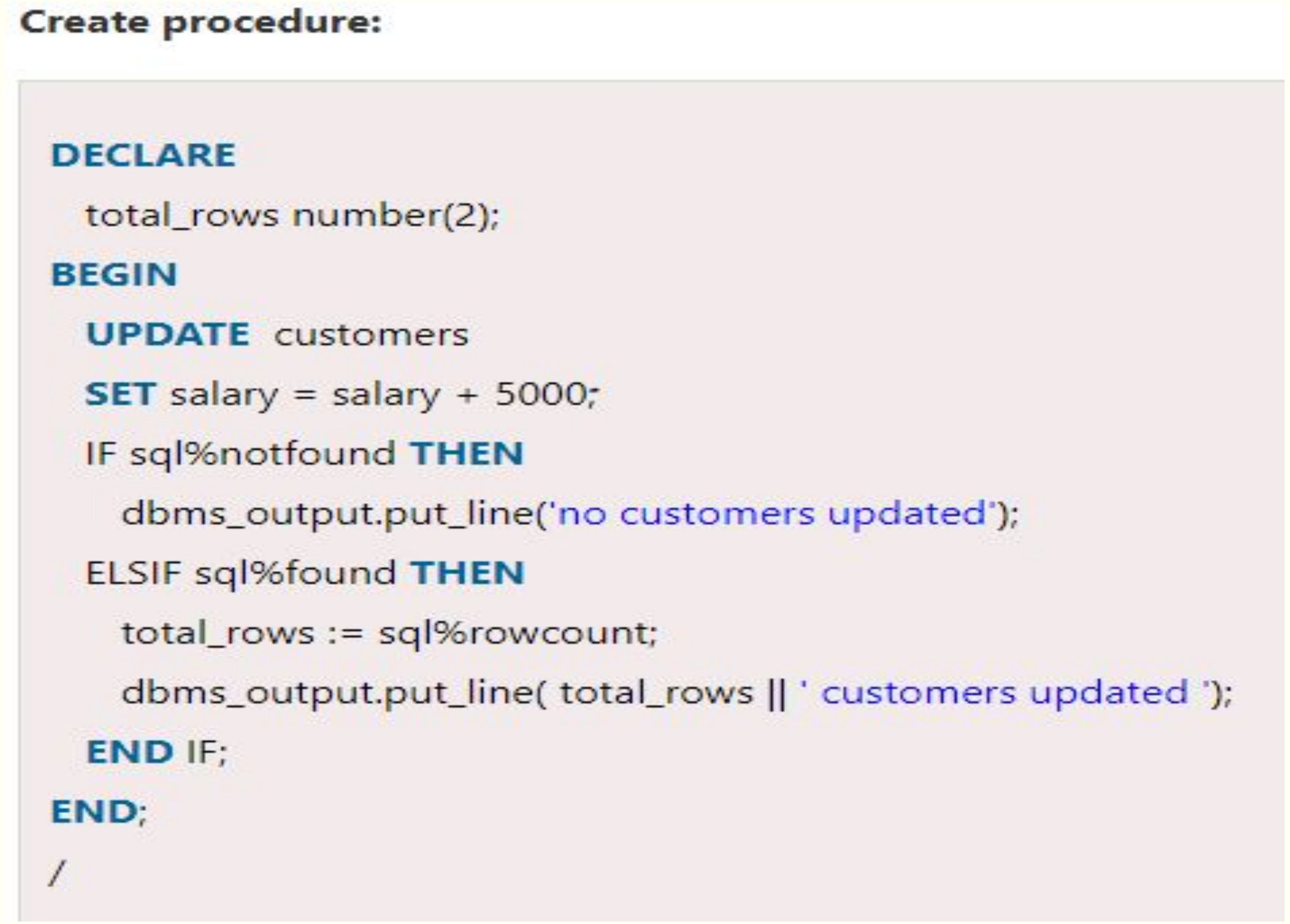


Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

Cont…



Create procedure:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Cont…

- **Output:**

```
6 customers updated
PL/SQL procedure successfully completed.
```

```
select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
2	Suresh	22	Kanpur	27000
3	Mahesh	24	Ghaziabad	29000
4	Chandan	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

2) PL/SQL Explicit Cursors

- The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.
- Following is the syntax to create an explicit cursor:

  - **CURSOR** cursor_name **IS** select_statement;;

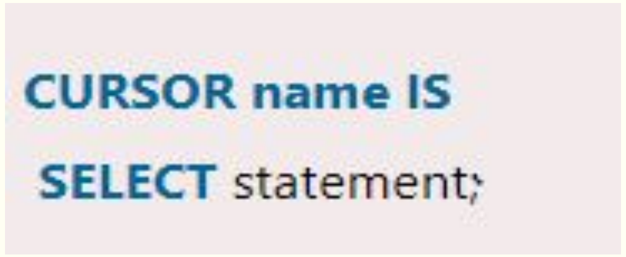
__**Steps:**__

- You must follow these steps while working with an explicit cursor.
- Declare the cursor to initialize in the memory.
- Open the cursor to allocate memory.
- Fetch the cursor to retrieve data.
- Close the cursor to release allocated memory.

Cont…

1) Declare the cursor:

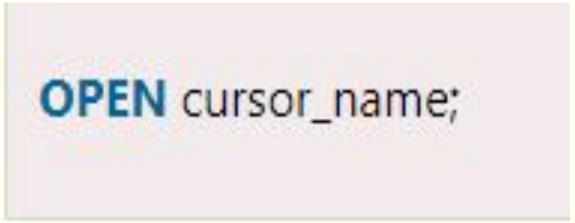
- It defines the cursor with a name and the associated SELECT statement.
- **Syntax for explicit cursor decleration**



```
CURSOR name IS
 SELECT statement;
```

2) Open the cursor:

- It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.
- **Syntax for cursor open:**

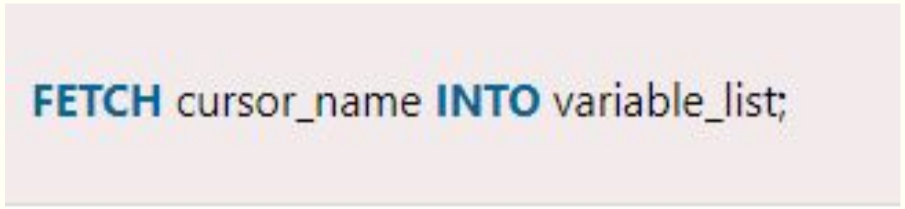


```
OPEN cursor_name;
```

Cont…

3) Fetch the cursor:

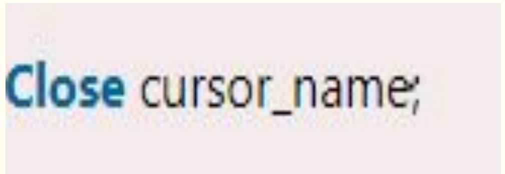
- It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:
- **Syntax for cursor fetch:**



```
FETCH cursor_name INTO variable_list;
```

4) Close the cursor:

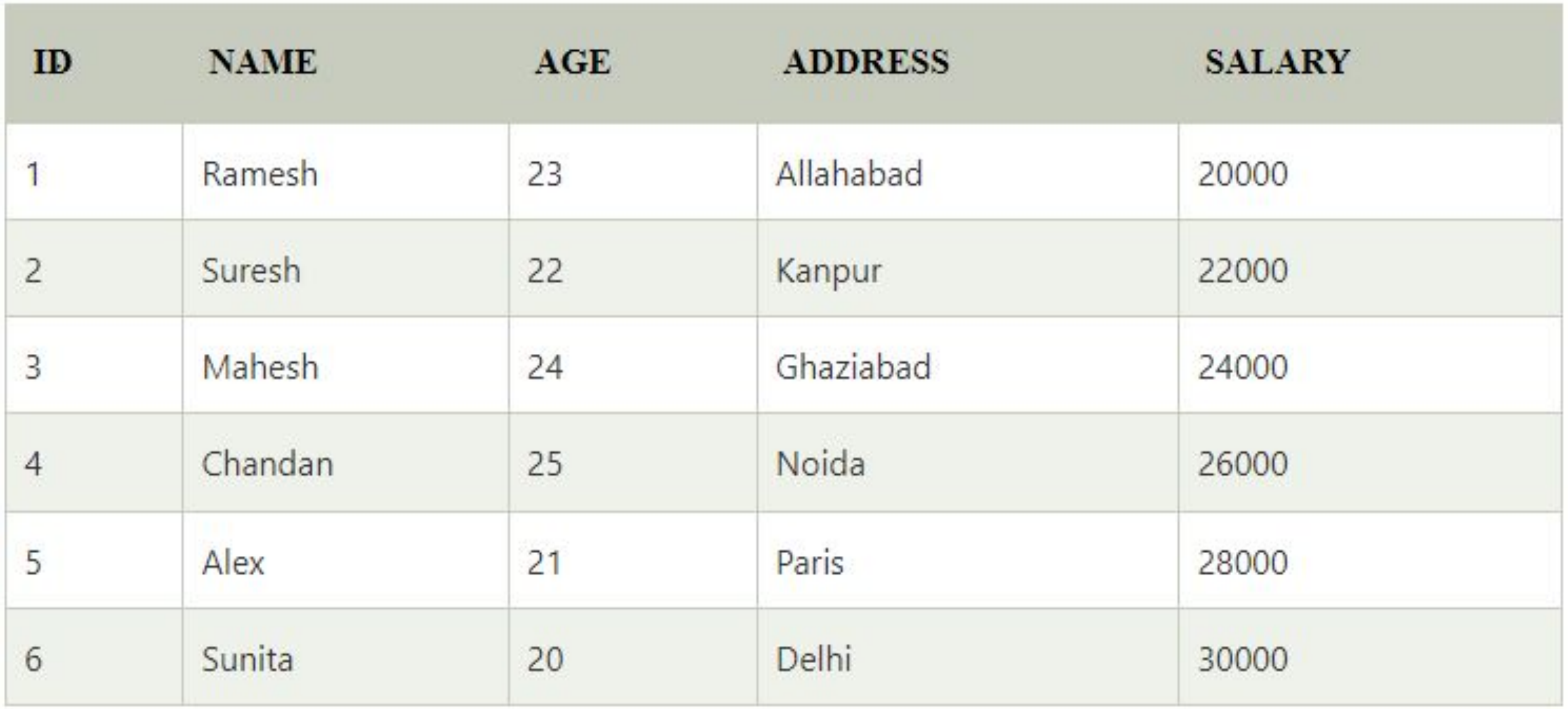
- It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.
- **Syntax for cursor close:**



```
Close cursor_name;
```

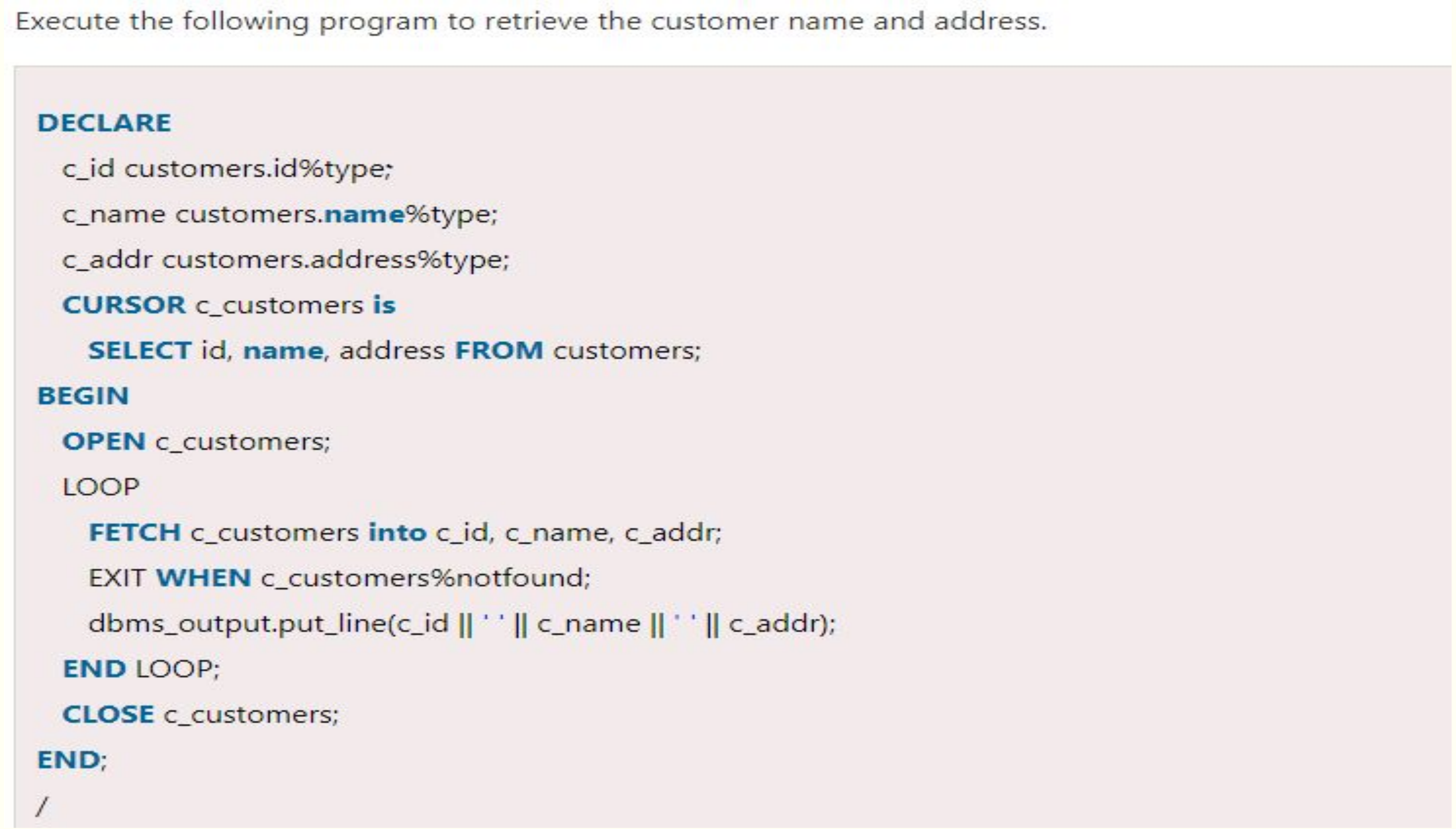
PL/SQL Explicit Cursor Example

Create customers table and have records:



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Cont…

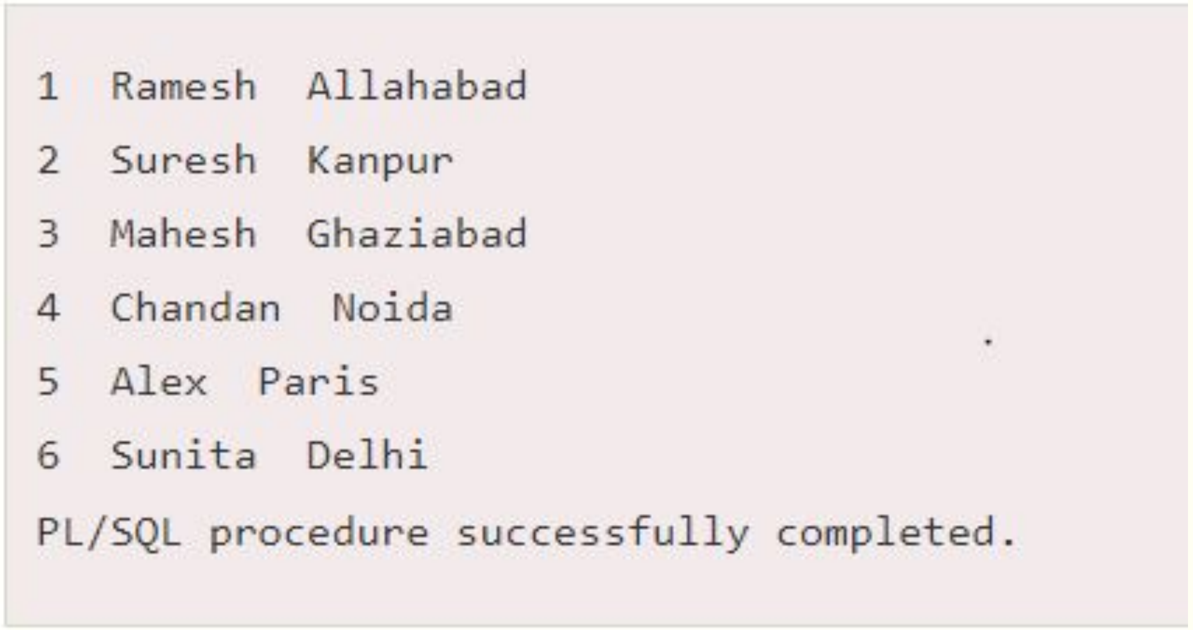


Execute the following program to retrieve the customer name and address.

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Cont…

Output:



```
1  Ramesh  Allahabad
2  Suresh  Kanpur
3  Mahesh  Ghaziabad
4  Chandan  Noida
5  Alex  Paris
6  Sunita  Delhi
PL/SQL procedure successfully completed.
```

Questions asked in GTU

1. Write a PL/SQL block to print the sum of Numbers from 1 to 100.
2. Write a PL/SQL block to print the given number is prime or not.
3. Write a PL/SQL program for inserting even numbers in EVEN table and odd number in ODD table from number 1 to 50.
4. Explain Cursor in PL/SQL.
5. Explain stored procedure with proper example.
6. What are triggers? Explain the advantages and the needs of triggers.

Questions asked in GTU

7. Write a PL/SQL block using explicit cursor that will display the customer name, the fixed deposit number and the fixed deposit amount of the first 5 customers holding the highest amount in fixed deposits. Use following database:cust_mstr(custno, name, occupation) fd_dtls(fd_ser_no, fd_no, type, period, opndt, duedt, amt, dueamt) acct_fd_cust_dtls(acct_fd_no, custno)

8. A stored function is created to perform the acct_no check operation. f_ChkAcctNo() is the name of function which accepts a variable acct_no from the user and returns value 0 if acct_no does not exist or 1 if acct_no exists.Write a PL/SQL block that performs transaction(i.e., deposit/withdrawal) on account. If account exists, change balance depending on the transaction amount to be deposited or withdrawal. Assume account table with fields – account number, name, type and balance.

Thank You…!!!