EXPERIMENT - XI
PROCEDURES, FUNCTIONS AND PACKAGES

August 23, 2019

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To study procedures, functions and packages

PROCEDURES

A procedure is a subprogram that performs a specific action.

Syntax

PROCEDURE name [(parameter, parameter, ...])]
IS [local declarations]
BEGIN <executable statements>
EXCEPTION <exception handlers]
END [name];

A procedure has two parts: the specification (spec for short) and the body. The procedure spec begins with the keyword PROCEDURE and ends with the procedure name or a parameter list.
Parameter declarations are optional. Procedures that take no parameters are written without parentheses.
The procedure body begins with the keyword IS and ends with the keyword END followed by an optional procedure name. The procedure body has three parts: a declarative part, an executable part, and an optional exception- handling part.
The declarative part contains local declarations, which are placed between the keywords IS and BEGIN. The keyword DECLARE, which introduces declarations in an anonymous PL/SQL block, is not used. The executable part contains statements, which are placed between the keywords BEGIN and EXCEPTION (or END). At least one statement must appear in the executable part of a procedure. The NULL statement meets this requirement. The exception-handling part contains exception handlers, which are placed between the keywords EXCEPTION and END.

FUNCTIONS

A function is a subprogram that computes a value. Functions and procedures are structured alike, except that functions have a RETURN clause.

Syntax

FUNCTION name [(parameter[, parameter, ...])]
RETURN datatype IS [local declarations]
BEGIN <executable statements>
EXCEPTION <exception handlers>
END [name];

PACKAGES

A package is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The specification (spec for short) is the interface to your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the spec.
To create packages, use the CREATE PACKAGE statement, which you can execute interactively from SQL*Plus.

Syntax

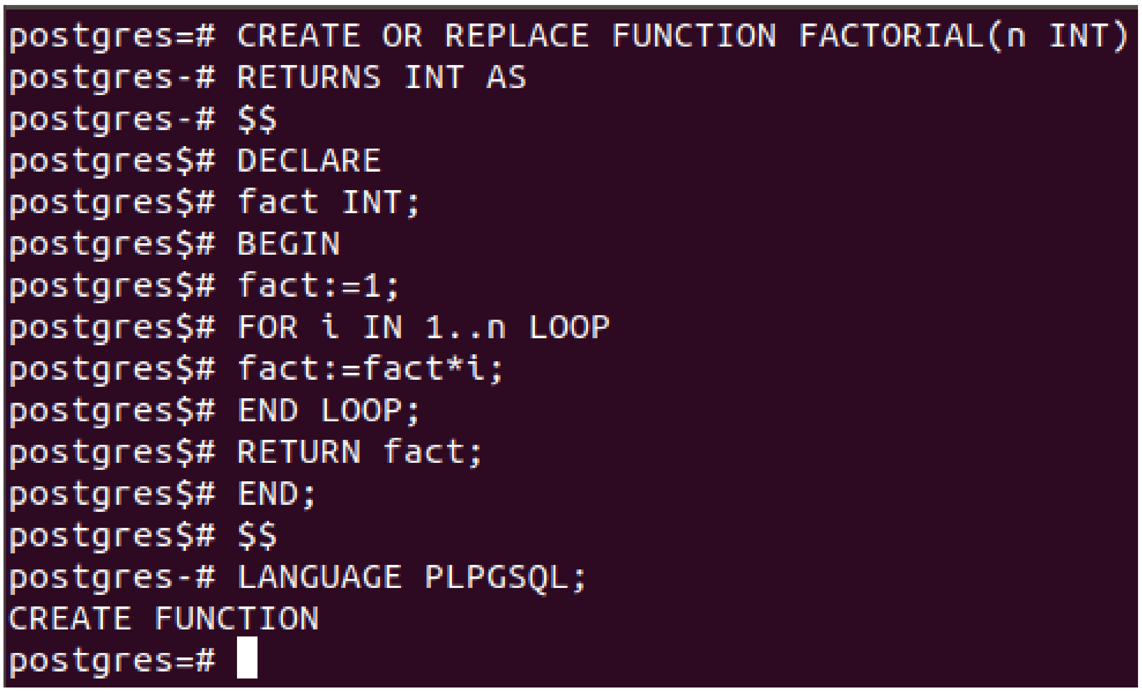
```
CREATE [OR REPLACE] PACKAGE
packagename (AUTHID CURRENTUSER | DEFINER] IS | AS
(typedefinition (typedefinition) ...)
(cursorspec (cursorspec) ...)
(itemdeclaration (itemdeclaration) ...)
(subprogramspec | callspec (subprogramspec | callspec)...)
END [packagename];
(CREATE [OR REPLACE] PACKAGE BODY packagename
IS | AS (typedefinition (typedefinition) ...)
(cursorbody (cursorbody) ...)
(itemdeclaration (itemdeclaration) ...)
(subprogramspec | callspec (subprogramspec | callspec)...)
(BEGIN
sequence_of_statements
END [packagename];)
```

The spec holds public declarations, which are visible to your application. The body holds implementation details and private declarations, which are hidden from your application. Following the declarative part of the package body is the optional initialization part, which typically holds statements that initialize package variables.

QUESTIONS

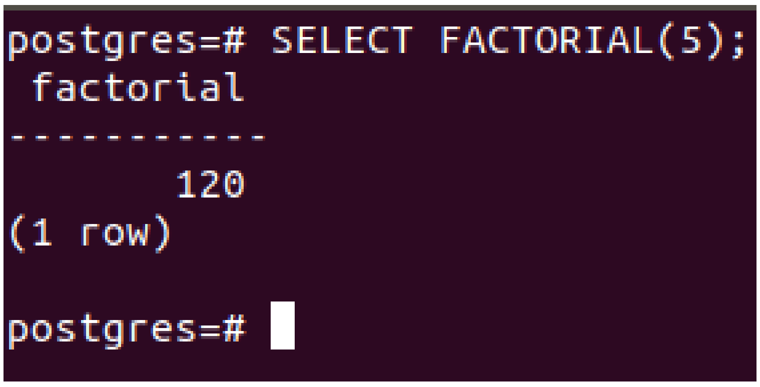
1. Create a function factorial to find the factorial of a number. Use this function in a PL/SQL Program to display the factorial of a number read from the user.

SOURCE CODE



```
postgres=# CREATE OR REPLACE FUNCTION FACTORIAL(n INT)
postgres-# RETURNS INT AS
postgres-# $$
postgres$# DECLARE
postgres$# fact INT;
postgres$# BEGIN
postgres$# fact:=1;
postgres$# FOR i IN 1..n LOOP
postgres$# fact:=fact*i;
postgres$# END LOOP;
postgres$# RETURN fact;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

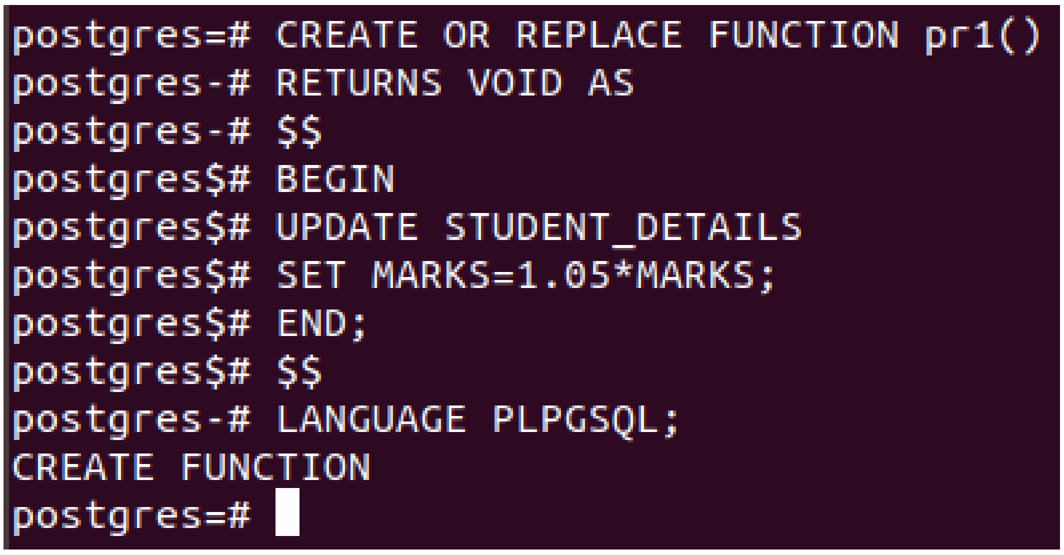
OUTPUT



```
postgres=# SELECT FACTORIAL(5);
 factorial
-----------
       120
(1 row)

postgres=#
```

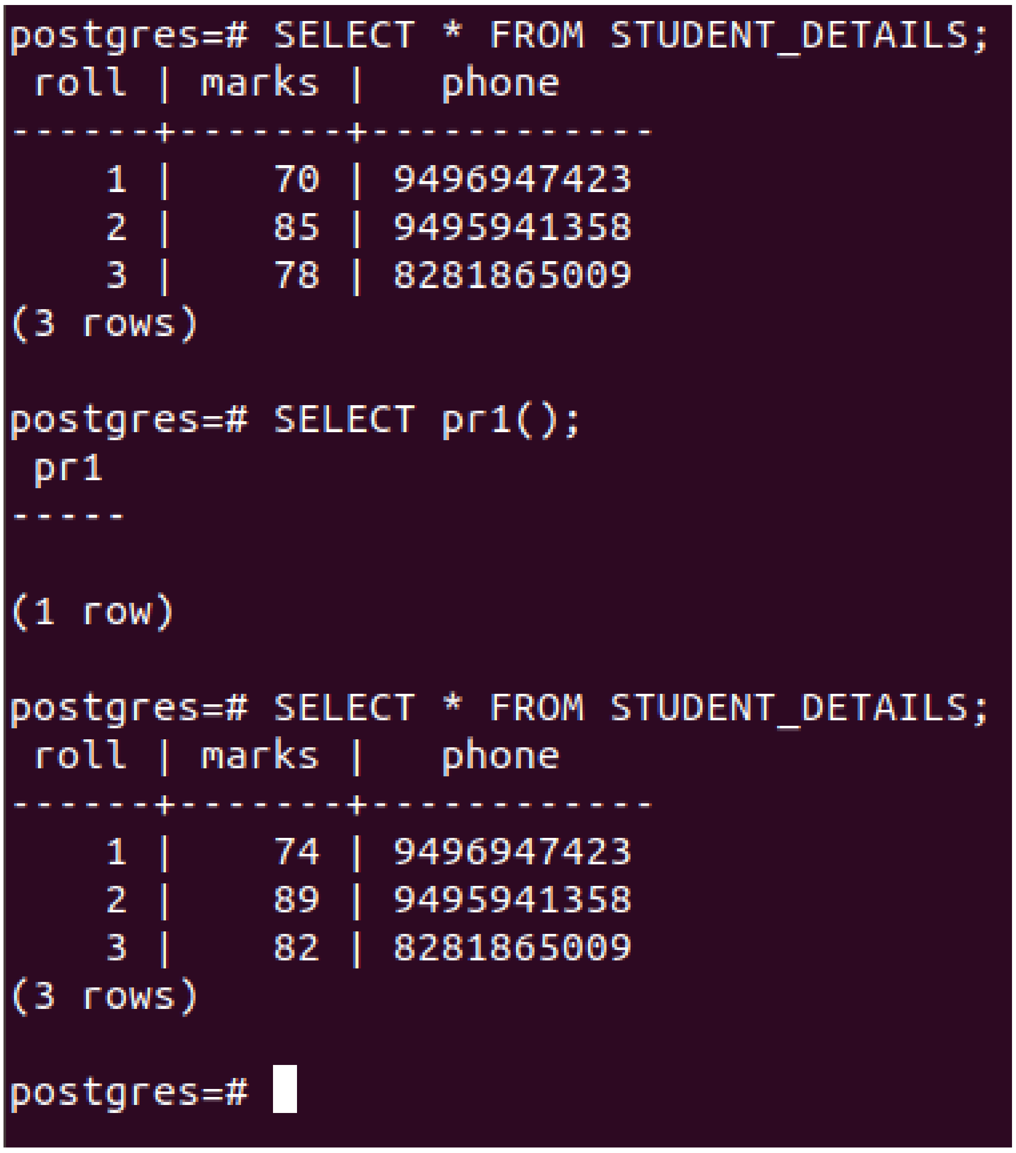
2. Create a table student_details(roll int,marksint, phone int). Create a procedure pr1 toupdate all rows in the database. Boost the marks of all students by 5%.

SOURCE CODE



```
postgres=# CREATE OR REPLACE FUNCTION pr1()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$# UPDATE STUDENT_DETAILS
postgres$# SET MARKS=1.05*MARKS;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# 
```

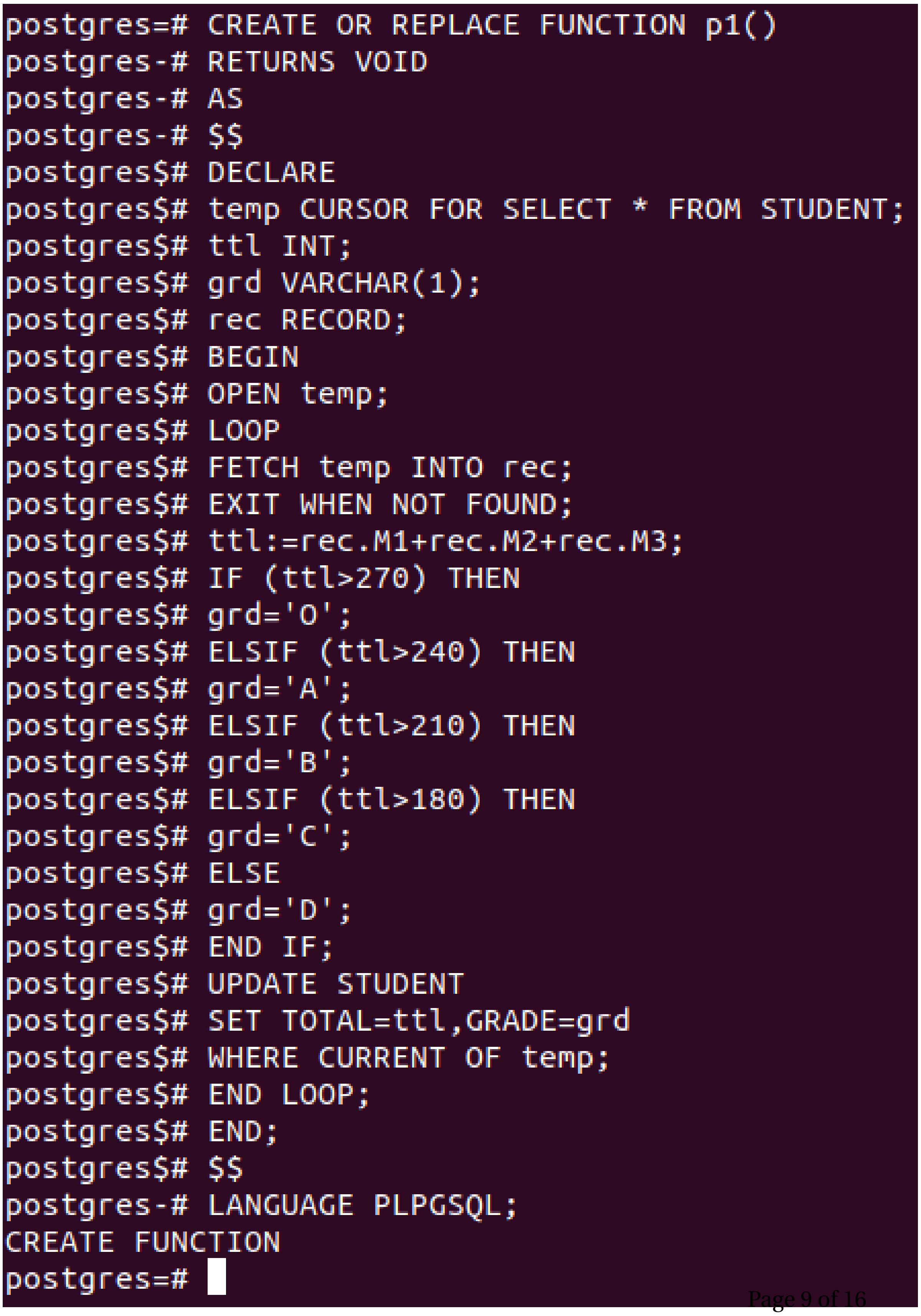
OUTPUT



```
postgres=# SELECT * FROM STUDENT_DETAILS;
 roll | marks |   phone
------+-------+------------
    1 |    70 | 9496947423
    2 |    85 | 9495941358
    3 |    78 | 8281865009
(3 rows)

postgres=# SELECT pr1();
 pr1
-----

(1 row)

postgres=# SELECT * FROM STUDENT_DETAILS;
 roll | marks |   phone
------+-------+------------
    1 |    74 | 9496947423
    2 |    89 | 9495941358
    3 |    82 | 8281865009
(3 rows)

postgres=# 
```

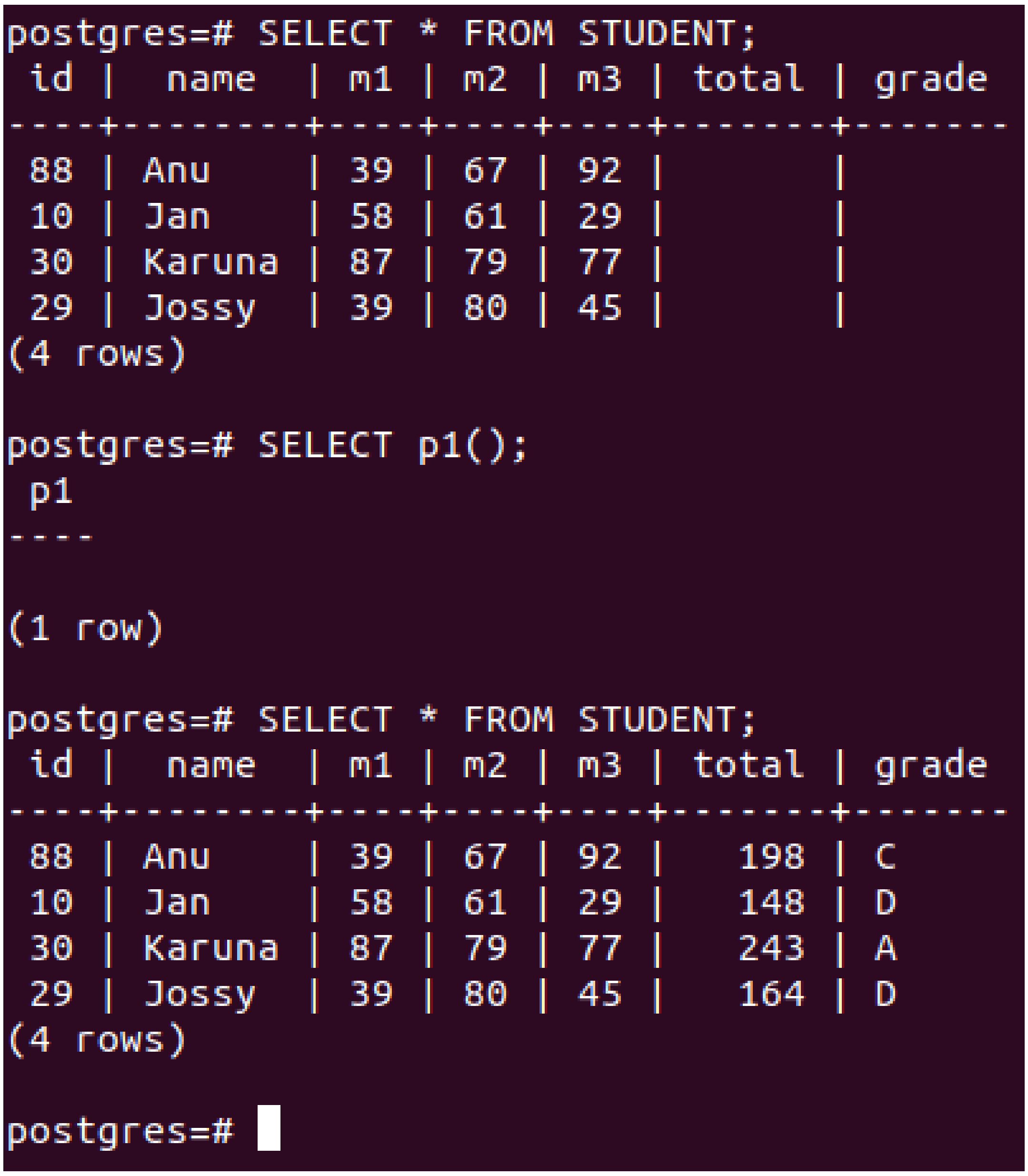
3. Create table student (id, name, m1, m2, m3, total, grade).Create a function f1 to calculate grade. Create a procedure p1 to update the total and grade.
   1. Read id,name,m1,m2,m3 from the user
   2. Insert the tuple into the database
   3. Using function f1 calculate the grade
   Using procedure p1, update the grade value for the tuple.

SOURCE CODE



```
postgres=# CREATE OR REPLACE FUNCTION p1()
postgres-# RETURNS VOID
postgres-# AS
postgres-# $$
postgres$# DECLARE
postgres$# temp CURSOR FOR SELECT * FROM STUDENT;
postgres$# ttl INT;
postgres$# grd VARCHAR(1);
postgres$# rec RECORD;
postgres$# BEGIN
postgres$# OPEN temp;
postgres$# LOOP
postgres$# FETCH temp INTO rec;
postgres$# EXIT WHEN NOT FOUND;
postgres$# ttl:=rec.M1+rec.M2+rec.M3;
postgres$# IF (ttl>270) THEN
postgres$# grd='O';
postgres$# ELSIF (ttl>240) THEN
postgres$# grd='A';
postgres$# ELSIF (ttl>210) THEN
postgres$# grd='B';
postgres$# ELSIF (ttl>180) THEN
postgres$# grd='C';
postgres$# ELSE
postgres$# grd='D';
postgres$# END IF;
postgres$# UPDATE STUDENT
postgres$# SET TOTAL=ttl,GRADE=grd
postgres$# WHERE CURRENT OF temp;
postgres$# END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# 
```

OUTPUT

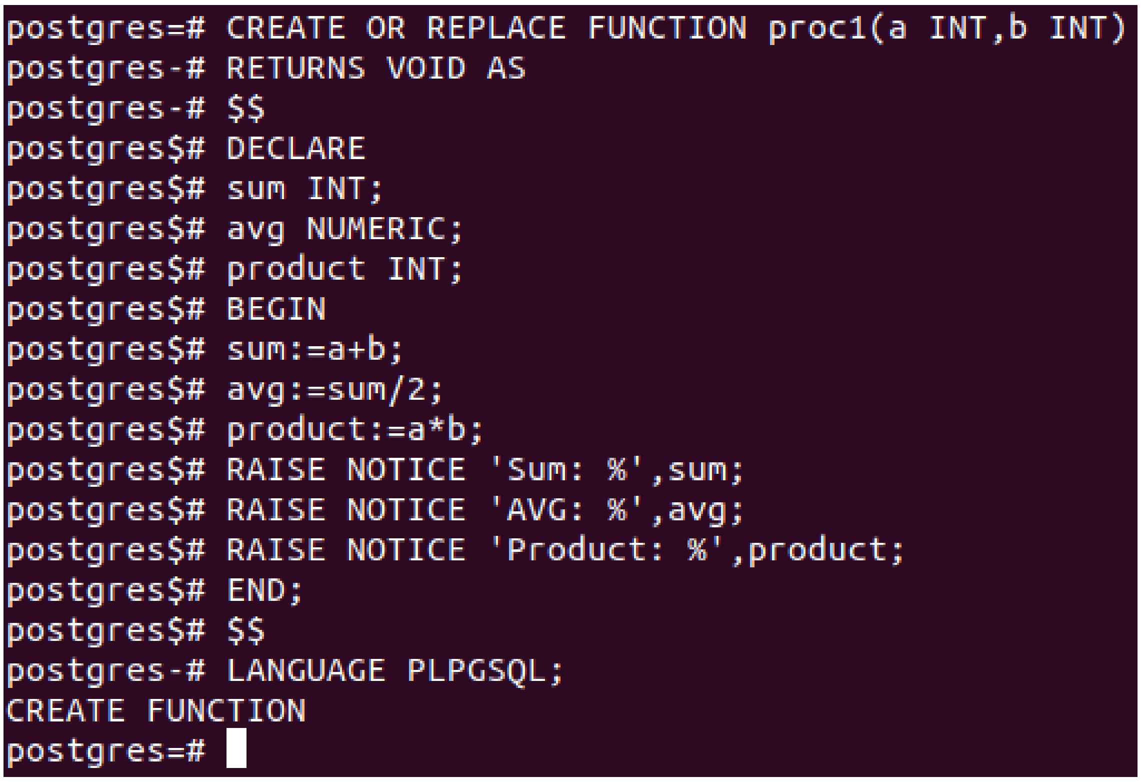


```
postgres=# SELECT * FROM STUDENT;
 id |  name   | m1 | m2 | m3 | total | grade
----+---------+----+----+----+-------+-------
 88 | Anu     | 39 | 67 | 92 |       |
 10 | Jan     | 58 | 61 | 29 |       |
 30 | Karuna  | 87 | 79 | 77 |       |
 29 | Jossy   | 39 | 80 | 45 |       |
(4 rows)

postgres=# SELECT p1();
 p1
----

(1 row)

postgres=# SELECT * FROM STUDENT;
 id |  name   | m1 | m2 | m3 | total | grade
----+---------+----+----+----+-------+-------
 88 | Anu     | 39 | 67 | 92 |   198 | C
 10 | Jan     | 58 | 61 | 29 |   148 | D
 30 | Karuna  | 87 | 79 | 77 |   243 | A
 29 | Jossy   | 39 | 80 | 45 |   164 | D
(4 rows)

postgres=# 
```

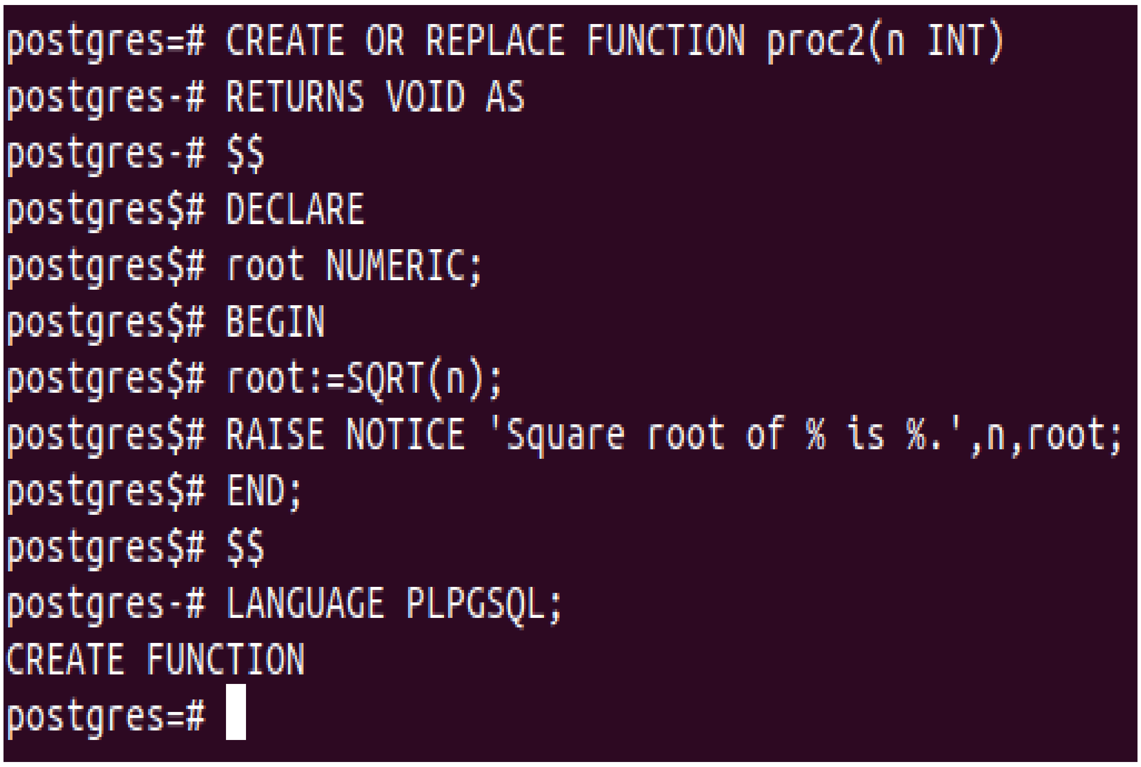
4. Create a package pk1 consisting of the following functions and procedures
   1. Procedure proc1 to find the sum, average and product of two numbers
   2. Procedure proc2 to find the square root of a number
   3. Function named fn11 to check whether a number is even or not
   4. A function named fn22 to find the sum of 3 numbers

   Use this package in a PL/SQL program. Call the functions f11, f22 and procedures pro1, pro2 within the program and display their results.

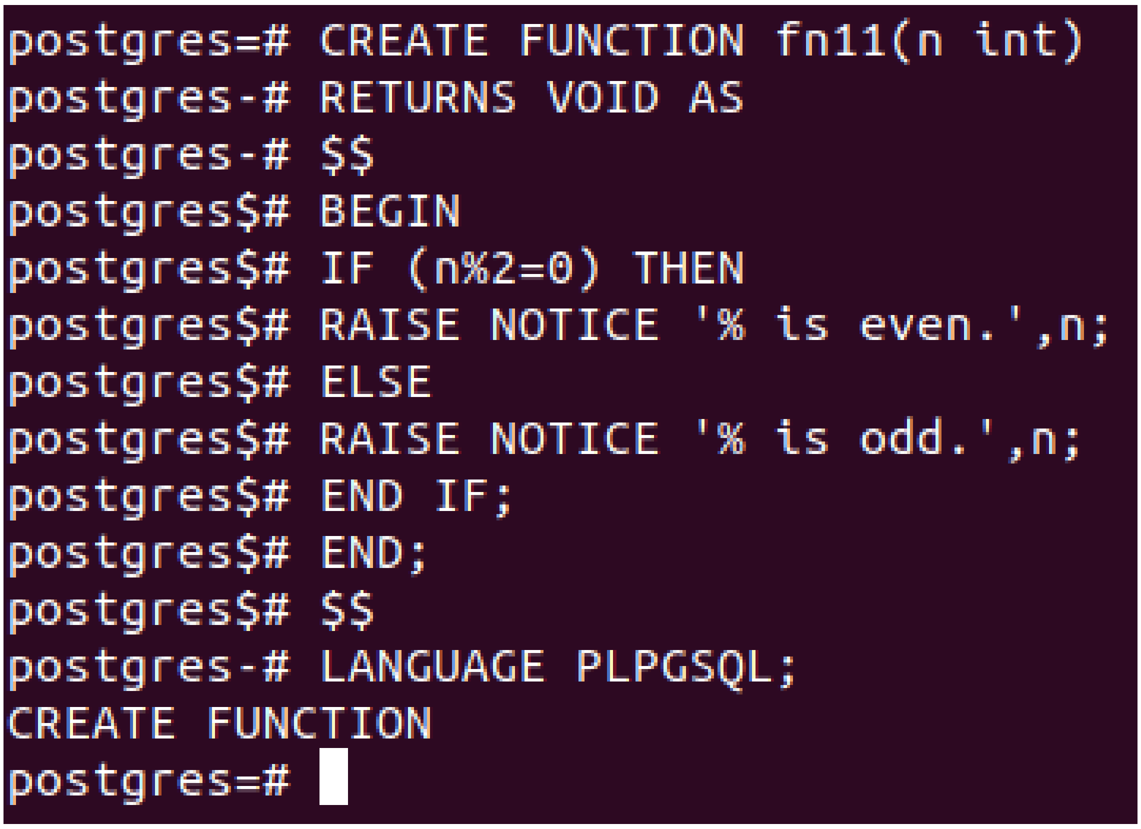
SOURCE CODE



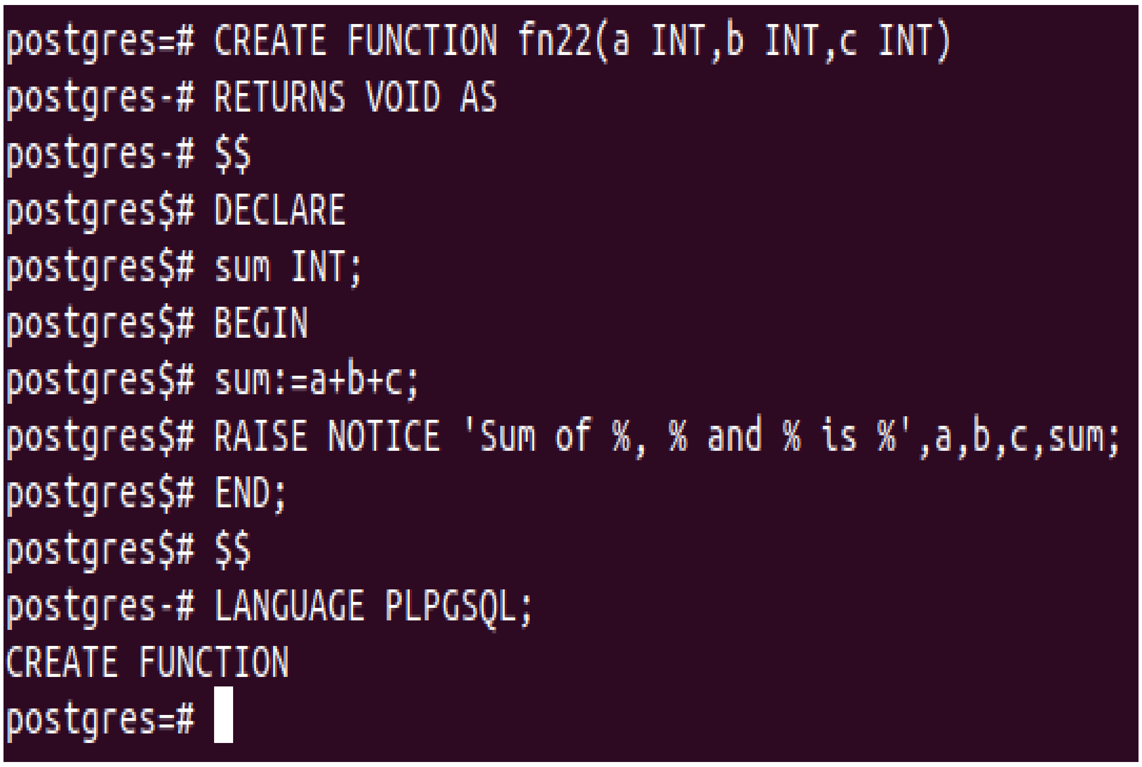
```
postgres=# CREATE OR REPLACE FUNCTION proc1(a INT,b INT)
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# sum INT;
postgres$# avg NUMERIC;
postgres$# product INT;
postgres$# BEGIN
postgres$# sum:=a+b;
postgres$# avg:=sum/2;
postgres$# product:=a*b;
postgres$# RAISE NOTICE 'Sum: %',sum;
postgres$# RAISE NOTICE 'AVG: %',avg;
postgres$# RAISE NOTICE 'Product: %',product;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# 
```



```
postgres=# CREATE OR REPLACE FUNCTION proc2(n INT)
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# root NUMERIC;
postgres$# BEGIN
postgres$# root:=SQRT(n);
postgres$# RAISE NOTICE 'Square root of % is %.',n,root;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# 
```

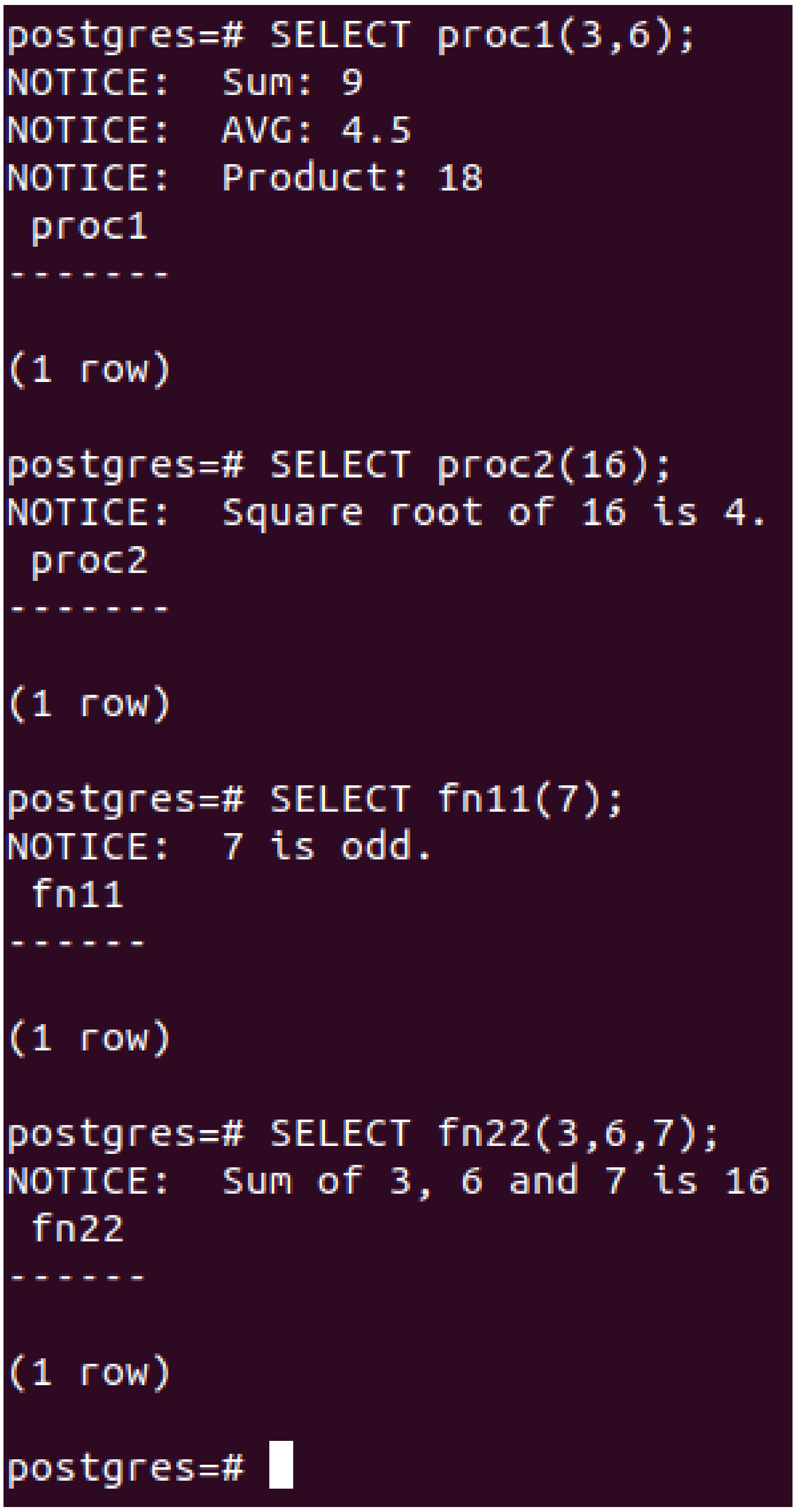


```
postgres=# CREATE FUNCTION fn11(n int)
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$# IF (n%2=0) THEN
postgres$# RAISE NOTICE '% is even.',n;
postgres$# ELSE
postgres$# RAISE NOTICE '% is odd.',n;
postgres$# END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```



```
postgres=# CREATE FUNCTION fn22(a INT,b INT,c INT)
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# sum INT;
postgres$# BEGIN
postgres$# sum:=a+b+c;
postgres$# RAISE NOTICE 'Sum of %, % and % is %',a,b,c,sum;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT



```
postgres=# SELECT proc1(3,6);
NOTICE:  Sum: 9
NOTICE:  AVG: 4.5
NOTICE:  Product: 18
 proc1
-------

(1 row)

postgres=# SELECT proc2(16);
NOTICE:  Square root of 16 is 4.
 proc2
-------

(1 row)

postgres=# SELECT fn11(7);
NOTICE:  7 is odd.
 fn11
------

(1 row)

postgres=# SELECT fn22(3,6,7);
NOTICE:  Sum of 3, 6 and 7 is 16
 fn22
------

(1 row)

postgres=# 
```

RESULT

The PL/SQL program was executed and the output was obtained.