Consider the following relational schema that manages the telephone bills of a mobile phone company.

CUSTOMER ( SSN, Name, Surname, PhoneNum, Plan)

PRICINGPLAN ( Code, ConnectionFee, PricePerSecond )

PHONECALL ( SSN, Date, Time, CalledNum, Seconds)

BILL ( SSN, Month, Year, amount )

1) Write a trigger that after each phone call updates the customer's bill

We make the assumption that the bills to be updated are

always already present in the database. In order to do this, we

can create another trigger that creates a bill with an amount of

0 for each registered customer at the beginning of each month

(suppose we have the event END\_MONTH)

**create trigger InitialBill**

**after END\_MONTH**

**begin**

**insert into BILL**

**select SSN, sysdate().month, sysdate().year, 0**

**from CUSTOMER**

**end**

**create trigger CallCharges**

**after insert of PHONECALL**

**for each row**

**begin**

**update BILL B**

**set Amount = Amount + ( select PP.ConnectionFee +**

**PP.PricePerSecond \* new.Seconds**

**from PRICINGPLAN PP join CUSTOMER C**

**on C.Plan = PT.Code**

**where new.SSN = C.SSN )**

**where B.SSN = new.SSN**

**and B.Year = new.Date.year and B.Month = new.Date.month**

**end**

2) Write a trigger that at the end of each month discount the bills by 5 cents per call to direct users of the company (that is,to numbers of

registered users in the table CUSTOMER) if the total monthly amount of the bill exceeds 100 €.

**create trigger Offer**

**after END\_MONTH**

**begin**

**update BILL B**

**set Amount = Amount – 0,05 \* ( select count(\*)**

**from PHONECALL P**

**where P.SSN = B.SSN**

**and P.Date.month = ( sysdate() – 1 ).month**

**and P.Date.year = ( sysdate() – 1 ).year**

**and P.CalledNum in ( select PhoneNum from CUSTOMER) )**

**where B.amount > 100 and B.year = (sysdate() - 1).year**

**and B.month = (sysdate() - 1).month**

**end**

3) Write a PL/SQL program that creates a stored procedure in the database.

**declare**

**procedure p\_hello**

**is**

**my\_val varchar2(256):='Hello, World!';**

**begin**

**dbms\_output.put\_line(my\_val);**

**end;**

**begin**

**p\_hello;**

**end;**

Output: Hello, World!

4) Write a PL/SQL program that creates a procedure which have parameters with default value.

**declare**

**procedure p\_print**

**(i\_str1\_tx VARCHAR2 :='hello',**

**i\_str2\_tx VARCHAR2 :='world',**

**i\_end\_tx VARCHAR2 :='!' ) is**

**begin**

**DBMS\_OUTPUT.put\_line(i\_str1\_tx||','**

**||i\_str2\_tx||i\_end\_tx);**

**end;**

**begin**

**p\_print('Hi',’there’,'...'); -- both parameters**

**p\_print('Hi','people'); -- without the last**

**p\_print('Hi'); -- only the first**

**p\_print(); -- no parameters**

**p\_print; -- no parenthesis**

**end;**

Output:

Hi,there...

Hi,people!

Hi,world!

hello,world!

hello,world!

4) Write a PL/SQL program that creates a procedure which have parameters passed by reference.

**create or replace procedure p\_validate(io\_string\_tx IN OUT NOCOPY VARCHAR2)**

**is**

**v\_invalid\_tx VARCHAR2(8):='test';**

**begin**

**io\_string\_tx:=replace (io\_string\_tx,v\_invalid\_tx);**

**if length(io\_string\_tx)>4000**

**then**

**io\_string\_tx:=substr(io\_string\_tx,1,3997)||'...';**

**end if;**

**end;**

**/**

5) Write a PL/SQL function calculates the differences between total areas of circles with a number of radii.

**declare**

**v\_pi\_nr NUMBER:=3.14;**

**function f\_getDiff\_Nr(i\_rad1\_nr NUMBER,i\_rad2\_nr NUMBER)**

**return NUMBER is**

**v\_area1\_nr NUMBER;**

**v\_area2\_nr NUMBER;**

**v\_out\_nr NUMBER;**

**function f\_getArea\_Nr (i\_rad\_nr NUMBER)**

**return NUMBER**

**is**

**begin**

**return v\_pi\_nr\*(i\_rad\_nr\*\*2);**

**end;**

**begin**

**v\_area1\_nr := f\_getArea\_Nr (i\_rad1\_nr);**

**v\_area2\_nr := f\_getArea\_Nr (i\_rad2\_nr);**

**v\_out\_nr :=v\_area1\_nr-v\_area2\_nr;**

**return v\_out\_nr;**

**end;**

**begin**

**DBMS\_OUTPUT.put\_line('Diff between 3 and 4: '||f\_getDiff\_Nr(4,3));**

**DBMS\_OUTPUT.put\_line('Diff between 4 and 5: '||f\_getDiff\_Nr(5,4));**

**DBMS\_OUTPUT.put\_line('Diff between 5 and 6: '||f\_getDiff\_Nr(6,5));**

**end;**

**/**

Output:

Diff between 3 and 4: 21.98

Diff between 4 and 5: 28.26

Diff between 5 and 6: 34.54

6) Create a PL/SQL package to place procedure and use it in package body.

**create or replace package pkg\_test1**

**as**

**function f\_getArea\_Nr (i\_rad\_nr NUMBER) return NUMBER;**

**procedure p\_print (i\_str1\_tx VARCHAR2 :='hello',**

**i\_str2\_tx VARCHAR2 :='world',**

**i\_end\_tx VARCHAR2 :='!' );**

**end;**

**/**

**create or replace package body pkg\_test1**

**as**

**function f\_getArea\_Nr (i\_rad\_nr NUMBER)**

**return NUMBER**

**is**

**v\_pi\_nr NUMBER:=3.14;**

**begin**

**return v\_pi\_nr \* (i\_rad\_nr \*\* 2);**

**end;**

**procedure p\_print**

**(i\_str1\_tx VARCHAR2 :='hello',**

**i\_str2\_tx VARCHAR2 :='world',**

**i\_end\_tx VARCHAR2 :='!' ) is**

**begin**

**DBMS\_OUTPUT.put\_line(i\_str1\_tx||','**

**||i\_str2\_tx||i\_end\_tx);**

**end;**

**end;**

**/**

7) Suppose that the Middlesex Transport Authority (MTA) has a rule stating that a bus driver’s salary

cannot be changed by more than 20% of the original salary. Create a trigger ‘salary\_change\_monitoring’ to

enforce this constraint. The trigger fires whenever there is an update to the Busdriver table and outputs a

suitable error message when the rule is violated.

**create or replace trigger salary\_change\_monitoring**

**before update on busdriver**

**for each row**

**begin**

**if ((:new.bdsalary/:old.bdsalary) >= 1.2) or**

**((:old.bdsalary/:new.bdsalary) >= 1.2)**

**then**

**RAISE\_APPLICATION\_ERROR(-20002, 'Warning: Large percentage change in salary**

**prohibited.');**

**end if;**

**end;**

**/**

8) Write a trigger to ensure that no employee of age less than 25 can be inserted in the database.

**CREATE TRIGGER Check\_age BEFORE INSERT ON employee**

**FOR EACH ROW**

**BEGIN**

**IF NEW.age < 25 THEN**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'ERROR:**

**AGE MUST BE ATLEAST 25 YEARS!';**

**END IF;**

**END;**

**/**

9) Write a trigger to count number of new tuples inserted using each insert statement.

**Declare count int**

**Set count=0;**

**delimiter $$**

**CREATE TRIGGER Count\_tupples**

**AFTER INSERT ON employee**

**FOR EACH ROW**

**BEGIN**

**SET count = count + 1;**

**END;**

**/**