PL/SQL Queries

* **Procedure which uses SQL%ROWCOUNT to determine the number of rows affected**

1) CREATE OR REPLACE PROCEDURE update\_product\_price(

p\_product\_id IN NUMBER,

p\_new\_price IN NUMBER)

IS

BEGIN

UPDATE PRODUCT

SET PRICE = p\_new\_price

WHERE PRODUCT\_ID = p\_product\_id;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' rows updated.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('No rows updated.');

END IF;

END;

BEGIN

update\_product\_price(p\_product\_id => 35, p\_new\_price => 25000);

END;

2) CREATE OR REPLACE PROCEDURE delete\_deliver\_by\_name(d\_name IN VARCHAR2)

IS

BEGIN

DELETE FROM DELIVERS

WHERE DELIVER\_NAME = d\_name;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' rows deleted.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('No rows deleted.');

END IF;

END;

BEGIN

delete\_deliver\_by\_name('Temp');

END;

* **Create a trigger before insert on any entity which will show the current number of rows in the table**

CREATE OR REPLACE TRIGGER PRODUCT\_PRICE\_TRIG

BEFORE INSERT ON PRODUCT

FOR EACH ROW

DECLARE

n NUMBER;

BEGIN

SELECT COUNT(\*) INTO n FROM PRODUCT;

DBMS\_OUTPUT.PUT\_LINE('Before insert product table has ' || n || ' values.');

END;

* **To check whether trigger is working**

INSERT INTO PRODUCT (PRODUCT\_ID, NAME, CATEGORY\_ID,PRICE, RATING, QUANTITY, DESCRIPTION, CITY)

VALUES (11, 'Banana', 71, 3700, 5, 100,'yellow banana', 'Aktau');

* **Procedure which does group by information**

1. CREATE OR REPLACE PROCEDURE count\_products\_with\_rating\_1 IS

CURSOR c\_product\_ratings IS

SELECT product\_id, COUNT(\*) AS num\_products

FROM product

WHERE rating = 1

GROUP BY product\_id;

BEGIN

FOR r\_product\_ratings IN c\_product\_ratings LOOP

DBMS\_OUTPUT.PUT\_LINE(r\_product\_ratings.product\_id || ': ' ||r\_product\_ratings.num\_products);

END LOOP;

END;

* To check whether procedure is working

BEGIN

count\_products\_with\_rating\_1;

END;

**1)A function that determines the number of orders**

**create or replace FUNCTION count\_orders**

**RETURN NUMBER**

**IS**

**count\_order NUMBER;**

**BEGIN**

**SELECT COUNT(order\_id) INTO count\_order FROM orders;**

**RETURN count\_order;**

**END;**

/

And we can check this by writing:

**Select count\_orders() from dual;**

**2) The function in which the average value of the rating is determined**

**create or replace FUNCTION get\_avg\_rating(rating IN VARCHAR2)**

**RETURN NUMBER**

**IS**

**l\_avg\_rating NUMBER;**

**BEGIN**

**SELECT AVG(TO\_NUMBER(rating))**

**INTO l\_avg\_rating**

**FROM product;**

**RETURN l\_avg\_rating;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**RETURN NULL;**

**END;**

**/**

And we can check this by writing:

**Select get\_avg\_rating('rating') AS avg\_rating**

**FROM dual;**

**-EXCEPTION**

**DECLARE**

**u\_name users.user\_name%TYPE := 'Craydon';**

**u\_id users.user\_id%TYPE;**

**user\_not\_found EXCEPTION;**

**BEGIN**

**SELECT id INTO u\_id FROM users WHERE user\_name = u\_name;**

**IF u\_id IS NULL THEN**

**RAISE user\_not\_found;**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('successfually find user');**

**END IF;**

**EXCEPTION**

**WHEN user\_not\_found THEN**

**DBMS\_OUTPUT.PUT\_LINE('Student with name ' u\_name ' not found.');**

**END;**

**DECLARE**

**u\_name users.user\_name%TYPE;**

**u\_phone users.user\_phone%TYPE;**

**u\_email users.user\_email%TYPE;**

**user\_not\_found EXCEPTION;**

**BEGIN**

**FOR user\_rec IN (SELECT \* FROM users) LOOP**

**u\_name := user\_rec.user\_name;**

**u\_email := user\_rec.user\_email;**

**u\_phone := user\_rec.user\_phone;**

**IF LENGTH(u\_phone) <> 12 AND INSTR(u\_email, '@') <= 0 THEN**

**RAISE user\_not\_found;**

**END IF;**

**END LOOP;**

**DBMS\_OUTPUT.PUT\_LINE('All users information is right');**

**EXCEPTION**

**WHEN user\_not\_found THEN**

**DBMS\_OUTPUT.PUT\_LINE('User with name ' u\_name ' has invalid information');**

**END;**