IT 4153 Advanced Database

Module 5 More on Triggers

|  |
| --- |
| **Introduction and Module Summary** |
| In this module, you will continue learning how to write PL/SQL triggers. |
| **Objectives and Outcomes** |
| This module directly supports **highlighted** course outcome(s)  Students who complete this course successfully will be able to   1. Describe current and emerging database models and technologies; 2. **Develop functions and procedures for data manipulation and database access auditing;** 3. Describe database monitoring and performance tuning; 4. Describe database security and administration issues, including backup and recovery; 5. Explain the concepts of data warehousing and data mining   **Module outcomes and activities:**   |  |  | | --- | --- | | After completing this module, students will be able: | implement business rules as PL/SQL triggers | | Readings | Reinforced | | Practice exercises | Reinforced/mastered | | Test |  | |
| **Assigned Reading** |
| 1. Module 1-4 and lab reports. |
| **Optional Reading** |
| 1. Any topics from <http://ksuweb.kennesaw.edu/~speltsve/files/ALG_Document.htm> |
| **Assessments and Assignments** |
| 1. Test (100 points) |
| **Topics** |
| Open the navigation pane |
| **Module Checklist** |
| This is the suggested order of the completion of this module.  Save a copy of this file on your computer and make notes in this document while you are completing your assignments. Use the table below to keep track of your progress.   |  |  | | --- | --- | | **Activity** | **Completion** | | Review all code and labs from modules 1-4 (3 hour) | NO | | Complete all exercises from the module (2 hours) | NO | | Take the test (2 hours) | NO | | Take the interim survey to help improve the course. | NO | |
|  |

Complete the following exercises. Do not look at the answers until you solve the problem yourself. Post your questions on the discussion board under "Help with labs".

# Exercise 1

## Question

Create two tables:

Employee: empID (PK), empFname, empLname, deptID(FK)

Department: deptID(PK), deptName, chairID

chairID is empID from Employee table

Insert at least three rows in the Department table and at least six rows in the Employee table.

Assumption:

Employee ID once in the EMPLOYEE table can't be changed.

Business rules:

a. One employee cannot be a chair of more than one department.

b. Each department has exactly one chair.

c. Employee cannot be deleted if he/she is a chair (implement as a trigger)

Modify tables to enforce these rules.

Check what happens with department chair when you delete employee from the Employee table. Do you see a problem?

Write a trigger to fix this problem. You have to raise application error when someone tries to delete employee who is listed as a department chair in the department table. It can be achieved several different ways, but for this exercise, you will use a trigger.

Create at least four test cases for test the solution.

Steps:

1. Write SQL code to create both tables.

2. Write SQL code to insert values in tables.

3. Write Trigger code

4. Create Test cases and check trigger execution results.

## Solution

### Create and insert

create and insert

create table DEPARTMENT(

deptID int primary key,

deptName varchar(100),

chairID int not null unique

-- business rules a and b

);

create table EMPLOYEE(

empID int primary key,

empFname varchar(50),

empLname varchar(50),

deptID int,

FOREIGN KEY(deptID) REFERENCES DEPARTMENT(deptID)

);

Insert into DEPARTMENT (deptID,deptName,chairID) values

(125,'Payroll',12058);

Insert into DEPARTMENT (deptID,deptName,chairID) values

(142,'Research and Development',64510);

Insert into DEPARTMENT (deptID,deptName,chairID) values

(134,'Human Resources',24503);

Insert into DEPARTMENT (deptID,deptName,chairID) values

(121,'Public Relations',37941);

Insert into EMPLOYEE values (12058,'Richardson','Donald',125);

Insert into EMPLOYEE values(64510,'Davis','Paul',142);

Insert into EMPLOYEE values(24503,'Warren','Melissa',134);

Insert into EMPLOYEE values(37941,'Small','Ryan',121);

Insert into EMPLOYEE values(11743,'Camp','Laura',142);

Insert into EMPLOYEE values(94137,'Lawrence','Lisa',134);

Select \* from employee;

|  |  |  |  |
| --- | --- | --- | --- |
| EMPID, , , DEPTID | EMPFNAME | EMPLNAME |  |
| 12058 | Richardson | Donald | 125 |
| 64510 | Davis | Paul | 142 |
| 24503 | Warren | Melissa | 134 |
| 37941 | Small | Ryan | 121 |
| 11743 | Camp | Laura | 142 |
| 94137 | Lawrence | Lisa | 134 |

Select \* from department;

|  |  |  |
| --- | --- | --- |
| DEPTID | DEPTNAME | CHAIRID |
| 125 | Payroll | 12058 |
| 142 | Research and Development | 64510 |
| 134 | Human Resources | 24503 |
| 121 | Public Relations | 37941 |

### Testing rules a and b

Insert into DEPARTMENT (deptID,deptName) values (124,'IT');

Result: cannot insert NULL into (%s)

Update DEPARTMENT set chairID=24503 where deptID=125;

Result: unique constraint (%s.%s) violated

### Testing rule c

delete from employee where empid = 12058;

Result in the department table we still have reference to deleted employee

125 Payroll 12058

Insert the record back.

Insert into EMPLOYEE values (12058,'Richardson','Donald',125);

create or replace

TRIGGER deptChair

BEFORE DELETE ON employee

FOR EACH ROW

DECLARE

deptCount INTEGER; --how many departments

BEGIN

SELECT COUNT(\*) INTO deptCount FROM department WHERE chairID=:old.empID;

--employee is listed as a department chair

IF deptCount !=0 THEN

raise\_application\_error('-20999',

'Employee is a chair and can''t be deleted. Please elect a new chair first');

END IF;

END;

Test again

delete from employee where empid = 12058;

Result: Employee is a chair and cannot be deleted. Please elect a new chair first 10 points

delete from employee where empid = 11743;

Result : 1 rows deleted. 10 points

# Exercise 2

## Question

Create two tables:

1. Temp\_readings table to hold location ID and 6 columns for temperature readings at 1am, 5am, 9am, 1pm, 5pm 9pm  
   Temp\_readings (locationid (PK), T\_1am , T\_5am, T\_9am, T\_1pm, T\_5pm, T\_9pm).
2. Temp\_max\_min table to hold minimum and maximum temperature for the last 24 hours (not for a particular date, just for the past 24 hours)  
   Temp\_max\_min (locationid (PK), min, max)

Insert two rows in each table.

Write a trigger that will update ***Temp\_min\_max*** table every four hours when temperature values are updated in the ***Temp\_readings*** table.

 Assumptions:

1. All locations exist both in ***Temp\_readings*** and ***Temp\_min\_max*** tables.
2. New locations cannot be inserted in these tables.
3. Location ID cannot be changed.
4. All columns in both tables have NOT NULL to prevent null values.

Hint

1. To find maximum use greatest function  
   SELECT greatest(1, 2, 3, 4) from dual;  
   will return 4
2. To find minimum use least function  
   SELECT least(1, 2, 3, 4) from dual;  
   will return 1.
3. Do not run SQL queries on a table you running your trigger (e.g. if trigger was fired as a result of update on table MyTable, do not select, insert, update on MyTable inside the trigger.
4. You can easily find any temperature reading by using :new.column\_name (e.g. :new.11pm)
5. The body of you trigger should have only one line of code.

Example

Temp\_readings table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LocationID | T\_1am | T\_5am | T\_9am | T\_1pm | T\_5pm | T\_9pm |
| 1212 | 56 | 58 | 58 | 62 | 67 | 60 |
| 3434 | 76 | 77 | 79 | 80 | 80 | 77 |

Temp\_ min\_max table

|  |  |  |
| --- | --- | --- |
| LocationID | min | max |
| 1212 | 56 | 67 |
| 3434 | 76 | 80 |

After T\_9pm updated at 9 pm Temp\_readings table for location 1212

Temp\_readings table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LocationID | T\_1am | T\_5am | T\_9am | T\_1pm | T\_5pm | T\_9pm |
| 1212 | 56 | 58 | 58 | 62 | 67 | 54 |
| 3434 | 76 | 77 | 79 | 80 | 80 | 77 |

Temp\_min\_max table

|  |  |  |
| --- | --- | --- |
| LocationID | min | max |
| 1212 | 54 | 67 |
| 3434 | 76 | 80 |

Steps:

1. Write SQL code to create both tables.

2. Write SQL code to insert values in tables.

3. Write Trigger code

4. Create Test cases and check trigger execution results.

## Solution

### Create and insert

CREATE TABLE Temp\_Readings (locationID NUMBER primary key,  
T\_1am NUMBER(4,1) NOT NULL,   
T\_5am NUMBER(4,1) NOT NULL,   
T\_9am NUMBER(4,1) NOT NULL,

T\_1pm NUMBER(4,1) NOT NULL,   
T\_5pm NUMBER(4,1) NOT NULL,   
T\_9pm NUMBER(4,1) NOT NULL);

CREATE TABLE Temp\_min\_max (locationID NUMBER primary key, lowTemp NUMBER(4,1) NOT NULL, highTemp NUMBER(4,1) NOT NULL);

INSERT INTO Temp\_readings VALUES (1212, 56, 58, 58, 62, 67, 60);

INSERT INTO Temp\_readings VALUES (3434, 76, 77, 79, 80, 80, 77);

INSERT INTO Temp\_min\_max VALUES ('1212', 0,0);

INSERT INTO Temp\_min\_max VALUES ('3434,0,0');

UPDATE Temp\_min\_max SET lowTemp= (SELECT least(T\_1am,T\_5am,T\_9am,T\_1pm,T\_5pm,T\_9pm ) from Temp\_Readings WHERE locationID = 1212) WHERE locationID = 1212;

UPDATE Temp\_min\_max SET highTemp= (SELECT greatest (T\_1am,T\_5am,T\_9am,T\_1pm,T\_5pm,T\_9pm) from Temp\_Readings WHERE locationID = 1212) *WHERE locationID = 1212;*

*UPDATE Temp\_min\_max SET lowTemp= (SELECT least(T\_1am,T\_5am,T\_9am,T\_1pm,T\_5pm,T\_9pm) from Temp\_Readings WHERE locationID = 3434) WHERE locationID = 3434;*

*UPDATE Temp\_min\_max SET highTemp= (SELECT greatest (T\_1am,T\_5am,T\_9am,T\_1pm,T\_5pm,T\_9pm) from Temp\_Readings WHERE locationID = 3434) WHERE locationID = 3434;*

***Create Trigger to calculate min/max temps and update Temp\_min\_max table.***

CREATE OR REPLACE

TRIGGER calculate\_min\_max

AFTER UPDATE ON Temp\_Readings

FOR EACH ROW

DECLARE

current\_location NUMBER; --what location we are updating?

BEGIN

current\_location := :new.locationID;

-- note that we do not use SELECT least(T\_1am,T\_5am,T\_9am,T\_1pm,T\_5pm,T\_9pm) from Temp\_Readings. We cannot do this because table is mutating, :new.T\_1am is the way to read new values being inserted in the table.

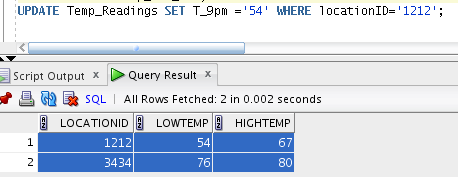
UPDATE Temp\_min\_max SET lowTemp= least(:new.T\_1am, :new.T\_5am, :new.T\_9am, :new.T\_1pm, :new.T\_5pm, :new.T\_9pm ) WHERE locationID = current\_location;

UPDATE Temp\_min\_max SET highTemp= greatest (:new.T\_1am, :new.T\_5am, :new.T\_9am, :new.T\_1pm, :new.T\_5pm, :new.T\_9pm) WHERE locationID = current\_location;

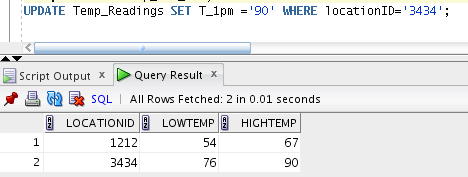
END;

### Testing

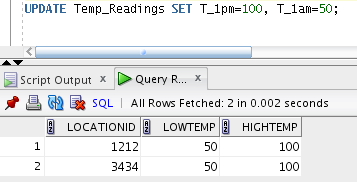
UPDATE Temp\_Readings SET T\_9pm ='54' WHERE locationID='1212';



UPDATE Temp\_Readings SET T\_1pm ='90' WHERE locationID='3434';



UPDATE Temp\_Readings SET T\_1pm=100, T\_1am=50;



# Exercise 3

Create three tables:

Products

|  |  |  |  |
| --- | --- | --- | --- |
| PRODUCTID | PRODUCTNAME | LISTPRICE | PRODUCTDESC |
| 1111 | Side Table | 99.99 | This end table is a and the perfect gift for any interior-design enthusiast. |
| 2222 | Leather Chair | 1023.99 | Sharp design and neutral and neutral Materials: Bonded leather. |
| 3333 | Storage Bench | 200 | Warehouse of Tiffany presents this versatile Malm storage bench with a stylish quilted cover. |

ProductsOnHand

|  |  |
| --- | --- |
| PRODUCTID | ONHANDQTTY |
| 1111 | 15 |
| 2222 | 15 |
| 3333 | 15 |

BackOrder

|  |  |
| --- | --- |
| PRODUCTID | BACKORDERQTTY |
| 1111 | 0 |
| 2222 | 0 |
| 3333 | 0 |

CREATE TABLE Products (ProductID NUMBER(4,0) PRIMARY KEY, ListPrice NUMBER(10,2) NOT NULL, ProductName VARCHAR2(20) NOT NULL, ProductDesc VARCHAR2(100) NOT NULL);

INSERT INTO Products VALUES (1111, 99.99, 'Side Table', 'This end table is a and the perfect gift for any interior-design enthusiast.');

INSERT INTO Products VALUES (2222, 1023.99, 'Leather Chair', 'Sharp design and neutral and neutral \nMaterials: Bonded leather.');

INSERT INTO Products VALUES (3333, 200, 'Storage Bench', 'Warehouse of Tiffany presents this versatile Malm storage bench with a stylish quilted cover.');

CREATE TABLE ProductsOnHand (ProductID NUMBER(4,0), OnHandQtty NUMBER(4,0));

INSERT INTO ProductsOnHand VALUES (1111, 15);

INSERT INTO ProductsOnHand VALUES (2222, 15);

INSERT INTO ProductsOnHand VALUES (3333, 15);

CREATE TABLE BackOrder (ProductID NUMBER(4,0), BackOrderQtty NUMBER(4,0));

INSERT INTO BackOrder VALUES (1111, 0);

INSERT INTO BackOrder VALUES (2222, 0);

INSERT INTO BackOrder VALUES (3333, 0);

## Question 1

Write a query that will produce the following result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PRODUCTID | PRODUCTNAME | LISTPRICE | ONHANDQTTY | BACKORDERQTTY |
| 1111 | Side Table | 99.99 | 15 | 0 |
| 2222 | Leather Chair | 1023.99 | 15 | 0 |
| 3333 | Storage Bench | 200 | 15 | 0 |

## Answer 1

SELECT

P.PRODUCTID, P.PRODUCTNAME, P.LISTPRICE, PO.ONHANDQTTY, BO.BACKORDERQTTY

FROM

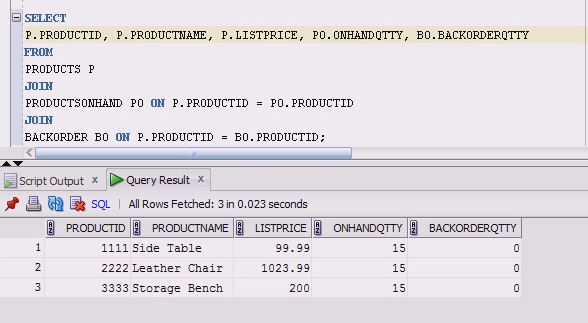
PRODUCTS P

JOIN

PRODUCTSONHAND PO ON P.PRODUCTID = PO.PRODUCTID

JOIN

BACKORDER BO ON P.PRODUCTID = BO.PRODUCTID;



## Question 2

Implement a feature that will prevent the insertion of the duplicate records in ProductsOnHand and BackOrder tables. Provide several test cases to test your implementation. (no pl/sql)

## Answer 2

Add UNIQUE constraints

ALTER TABLE ProductsOnHand

ADD CONSTRAINT unique\_constraint UNIQUE (ProductID , OnHandQtty );

ALTER TABLE BackOrder

ADD CONSTRAINT uniqueRec\_constraint UNIQUE (ProductID , BackOrderQtty);

Test 1: Attempt to insert existing record into ProductsOnHand

INSERT INTO ProductsOnHand VALUES (1111, 15);

Result: SQL Error: unique constraint (SYSTEM.unique\_constraint) violated

Test2: Try to insert existing record into BackOrder

INSERT INTO BackOrder VALUES (1111, 0);

Result: SQL Error: unique constraint (SYSTEM.uniquerec\_constraint) violated

Test 3: Try to insert new record into ProductsOnHand

INSERT INTO ProductsOnHand VALUES (4444, 15);

Result: 1 row inserted

Test 4: Try to insert new record into BackOrder

INSERT INTO BackOrder VALUES (4444, 1);

Result: 1 row inserted

## Question 3

Implement a feature that will prevent the insertion of the products that do not exist in the Products table into ProductsOnHand and BackOrder tables. Provide several test cases to test your implementation. (no pl/sql)

## Answer 3

Use referential integrity

ALTER TABLE ProductsOnHand

ADD CONSTRAINT fk\_productid

FOREIGN KEY (productid)

REFERENCES products(productid);

ALTER TABLE BackOrder

ADD CONSTRAINT fk\_productid\_constraint

FOREIGN KEY (productid)

REFERENCES products(productid);

Test 1: Attempt to insert new record into ProductsOnHand

INSERT INTO ProductsOnHand VALUES (5555, 15);

Result: SQL Error: integrity constraint (SYSTEM.FK\_productid) violated. Parent key not found

Test2: Try to insert existing record into BackOrder

INSERT INTO BackOrder VALUES (5555, 0);

Result: SQL Error: integrity constraint (SYSTEM.FK\_productid\_constraint) violated. Parent key not found

Test 3: Insert new record in the products table, and then attempt to insert the same product id into ProductsOnHand and BackOrder:

INSERT INTO PRODUCTS VALUES(6666, 850, 'TV Stand', 'Entertainment Center');

Result: 1 row inserted

INSERT INTO ProductsOnHand VALUES (6666, 10);

Result: 1 row inserted

INSERT INTO BackOrder VALUES (6666, 0);

Result 1row inserted

## Question 4

Write a PL/SQL stored procedure that takes product ID and displays the name of the product, its price, ProductsOnHand quantity, and BackOrder quantity. If product ID is not in the table, your procedure should return “product not found”. 20 points for the correct code of the procedure and 40 points for testing your procedure (ten points per test case).

## Answer 4

CREATE OR REPLACE PROCEDURE p\_product (prodId NUMBER) AS

ProdName VARCHAR(50);

ListPr NUMBER;

OnHand NUMBER;

BackOrder NUMBER;

BEGIN

SELECT

P.PRODUCTNAME, P.LISTPRICE, PO.ONHANDQTTY, BO.BACKORDERQTTY

INTO ProdName, ListPr, OnHand, BackOrder

FROM

PRODUCTS P

JOIN PRODUCTSONHAND PO ON P.PRODUCTID = PO.PRODUCTID

JOIN BACKORDER BO ON P.PRODUCTID = BO.PRODUCTID

WHERE P.ProductId = prodId;

dbms\_output.put\_line(ProdName || ListPr || OnHand || BackOrder);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE ('No Data found for SELECT on product Id ' || prodId);

END;

/

SET SERVEROUTPUT ON;

Test Cases:

EXEC P\_PRODUCT(1111)

anonymous block completed

Side Table99.99 15 0

EXEC P\_PRODUCT(6666)

anonymous block completed

TV Stand850 10 0

EXEC P\_PRODUCT(4444)

anonymous block completed

Couch600 15 1

EXEC P\_PRODUCT(7777)

anonymous block completed

No Data found for SELECT on product Id 7777