CSE2DBF – CSE4DBF

Triggers

Triggers

- A trigger is a self-contained routine associated with a table that ***automatically*** performs an action when a row in the table is inserted, updated, or deleted.

- A trigger is not called directly. Instead, when an application or user attempts to INSERT, UPDATE, or DELETE a row in a table, any triggers associated with that table and operation are automatically executed (“fired”).

- The main difference between triggers and stored procedures: triggers executed automatically by the system, stored procedures has to be called or executed explicitly from user program.

Triggers

- Automatic enforcement of data restrictions, to make sure users enter only valid values into columns.

- Reduce application maintenance, since changes to a trigger are automatically reflected in all applications that use the associated table without the need to recompile and relink.

Triggers: Syntax

```
CREATE [OR REPLACE] TRIGGER <trigger_name>

{BEFORE | AFTER | INSTEAD OF } {UPDATE | INSERT | DELETE}
  [OF <attribute_name>] ON <table name>

[FOR EACH ROW ]

[DECLARE  <local_variable> <variable_type>;]

BEGIN
      << trigger body goes here >>

END <trigger_name>;
/
```

Process of designing a Trigger

- What table? (linked to the trigger)
- Which operation? (fires the trigger)
- When? (the trigger is fired)
- How many times? (the trigger is fired)
- What should the trigger do? (what operations/tasks performed by the trigger)

Triggers: Context Variable

- Note that triggers can use two context variables: **OLD and NEW**.
  - OLD context variable refers to the current or previous values in a row being updated or deleted.
  - NEW context variable refers to a new set of INSERT or UPDATE values for a row. NEW is not used for DELETE.
- Context variables are often used to compare the values of a column before and after it is modified.
- Syntax: `:old.column :new.column`

Example

TableX

A	B
1	2
3	4

```
Insert into TableX values(5,6);
:NEW.A =5
:NEW.B =6

delete from TableX where A = 1;
:OLD.A =1
:OLD.B =2

update TableX set A = 100 where A = 3;
:OLD.A =3    :NEW.A =100
:OLD.B =4    :NEW.B =4
```

Triggers: Example 1

```
CREATE VIEW room_summary AS
  SELECT building, sum(number_seats) total_seats
    FROM rooms
      GROUP BY building;
```

Rooms

Building	Room	Number_Seats
BG	BG-114	35
BG	BG-115	40
PS1	PS1-101	20
PS1	PS1-102	25

```
CREATE TRIGGER room_summary_delete
  INSTEAD OF DELETE ON room_summary
  FOR EACH ROW

BEGIN
  -- Delete all of the rows in rooms which match this
  -- single row in room_summary

  DELETE FROM rooms
    WHERE building = :old.building;

END room_summary_delete;
/
```

Room_Summary

Building	Total_Seats
BG	75
PS1	45

Triggers: Different Type of Triggers

- STATEMENT TRIGGER:
  - executed once regardless of the number of rows affected.
- ROW TRIGGER:
  - executed once for every affected row.
- BEFORE TRIGGER (Before Insert | Update | Delete)
- AFTER TRIGGER (After Insert | Update | Delete)
- INSTEAD OF TRIGGER (Normally Instead Of Delete)

Triggers: Sequence

```
CREATE SEQUENCE trigger_seq
   START WITH 1
   INCREMENT BY 1;
```

The above trigger is used to create a sequenced number generator, start with 1, and increment by 1.

Example:

```
CREATE SEQUENCE client_seq START WITH 1 INCREMENT BY 1;

INSERT INTO Client (ClientID, ClientName)
VALUES ('C'||client_seq.nextval,'John Doe');
```

Triggers: Example 2

```
** A sequence trigger
CREATE SEQUENCE trigger_seq
  START WITH 1
  INCREMENT BY 1;
```

Classes

Department	Course	Enrolment
CS	BCS	50
CS	BIT	150
Maths	B.Maths	40
Physics	B.Physics	20

Temp_Table

Num_Col	Char_Col

```
UPDATE Classes
SET Department = 'CSIT'
WHERE Department = 'CS';
```

1

```
CREATE OR REPLACE TRIGGER classes_BStatement
  BEFORE UPDATE ON classes
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'Before Statement trigger');
END classes_BStatement;
/
```

2

```
CREATE OR REPLACE TRIGGER classes_AStatement
  AFTER UPDATE ON classes
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'After Statement trigger');
END classes_AStatement;
/
```

3

```
CREATE OR REPLACE TRIGGER classes_BRow
  BEFORE UPDATE ON classes
  FOR EACH ROW
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'Before Row trigger');
END classes_BRow;
/
```

4

```
CREATE OR REPLACE TRIGGER classes_ARow
  AFTER UPDATE ON classes
  FOR EACH ROW
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'After Row trigger');
END classes_ARow;
/
```

Triggers: Example 3

STUDENTS

ID	Name	Course

```
CREATE SEQUENCE student_sequence
  START WITH 1000
  INCREMENT BY 100;
```

```
CREATE OR REPLACE TRIGGER GenerateStudentID
  BEFORE INSERT ON students
  FOR EACH ROW
BEGIN
  /* Fill in the ID field of students with the next value
     from student_sequence. Since ID is a column in
     students, :new.ID is a valid reference. */
  SELECT student_sequence.nextval
    INTO :new.ID
    FROM dual;
END GenerateStudentID;
/
```

```
INSERT INTO STUDENTS (Name, Course)
VALUES('Rick Grimes', 'BCS');
```

STUDENTS

ID	Name	Course
1000	Rick Grimes	BCS

** student_sequence is a **sequence trigger**

Triggers: Example 4

```
CREATE OR REPLACE TRIGGER total_salary
  AFTER DELETE OR INSERT OR UPDATE OF deptno, sal ON EMP
  FOR EACH ROW

BEGIN
  /* assume that DEPTNO and SAL are non-null fields */
   IF DELETING OR (UPDATING AND :old.deptno != :new.deptno)
   THEN UPDATE dept
   SET total_sal = total_sal - :old.sal
   WHERE deptno = :old.deptno;
   END IF;

   IF INSERTING OR (UPDATING AND :old.deptno != :new.deptno)
   THEN UPDATE dept
   SET total_sal = total_sal + :new.sal
   WHERE deptno= :new.deptno;
   END IF;

   IF (UPDATING AND :old.deptno = :new.deptno AND :old.sal
   != :new.sal)
   THEN UPDATE dept
   SET total_sal = total_sal - :old.sal + :new.sal
   WHERE deptno = :new.deptno;
   END IF;
END;
/
```

EMP

EmpNo	Sal	DeptNo
E1	200	D1
E2	100	D1
E3	500	D2

DEPT

DeptNo	Total_Sal
D1	300
D2	500

What happened with DEPT when you do these operations?

```
DELETE FROM EMP
WHERE EmpNo = 'E1';

INSERT INTO EMP
VALUES ('E4', 100, 'D2');

UPDATE EMP
SET Sal = 700
WHERE EmpNo = 'E3';

UPDATE EMP
SET DeptNo = 'D2'
WHERE EmpNo = 'E1';
```

Triggers: Error Message

- In a trigger, we can also display an error message whenever a trigger is executed and as a result a certain operation is cancelled.

- To display this error message we can use:

```
RAISE_APPLICATION_ERROR
        (-20000, 'Cannot perform the operation');
```

Triggers: Example 5

EMPLOYEE

FNAME	*LNAME*	**SSN**	*ADDRESS*	*SEX*	*SALARY*	*DEPT NO*
John	Smith	123456789	731 Plenty, Clayton	M	30000	5
Franklin	Wong	333445555	638 Voss, Preston	M	40000	5
Alicia	Zelaya	999887777	3321 Castle, Balwyn	F	25000	4
Jennifer	Wallace	987654321	291 Berry, Preston	F	43000	4
Ramesh	Narayan	666884444	975 Fire, Carlton	M	38000	5
Joyce	English	453453453	5631 Rice, Hawthorn	F	25000	5
Ahmad	Jabbar	987987987	980 Henry, Clayton	M	25000	4
James	Borg	888665555	450 Stone, Caufield	M	55000	1

DEPARTMENT

DNAME	**DEPTNO**	*MGRSSN*	*MGRSTARTDATE*
Research	5	333445555	22/5/78
Administration	4	987654321	1/1/85
Headquarters	1	888665555	19/6/71

Based on the above EMPLOYEE table:

a) Write a trigger which automatically rejects an operation that deletes an employee who is currently a manager of a department.

[15 marks]

Triggers: Example 5

```
CREATE OR REPLACE TRIGGER delete_emp
BEFORE DELETE on Employee
FOR EACH ROW
DECLARE
       empCount NUMBER;
BEGIN
 SELECT COUNT(*)
 INTO empCount
 FROM Department D
 WHERE D.mgrssn = :old.ssn;

 IF empCount > 0 THEN
  RAISE_APPLICATION_ERROR
       (-20000,'Cannot delete employee...');
 END IF;

END delete_emp;
/
```

Triggers

To list all triggers in the user schema

```
SELECT *
FROM user_triggers
```

Example:

```
SELECT trigger_type, Table_name, triggering_event
FROM user_triggers
WHERE trigger_name = 'total_salary';
```

- More examples for Triggers can be found from:
  - Lab exercises on Triggers
  - Trigger exercises from past exams distributed during lectures

```
CREATE OR REPLACE TRIGGER classes_BStatement
  BEFORE UPDATE ON classes
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'Before Statement trigger');
END classes_BStatement;
/
```

1

```
CREATE OR REPLACE TRIGGER classes_AStatement
  AFTER UPDATE ON classes
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'After Statement trigger');
END classes_AStatement;
/
```

2

```
CREATE OR REPLACE TRIGGER classes_BRow
  BEFORE UPDATE ON classes
  FOR EACH ROW
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'Before Row trigger');
END classes_BRow;
/
```

3

```
CREATE OR REPLACE TRIGGER classes_ARow
  AFTER UPDATE ON classes
  FOR EACH ROW
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (trigger_seq.NEXTVAL, 'After Row trigger');
END classes_ARow;
/
```

4

**** A sequence trigger**

```
CREATE SEQUENCE
trigger_seq
  START WITH 1
  INCREMENT BY 1;
```