1.

CREATE OR REPLACE TRIGGER Print\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON Emp\_tab

FOR EACH ROW

WHEN (new.Empno > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :new.sal - :old.sal;

dbms\_output.put('Old salary: ' || :old.sal);

dbms\_output.put(' New salary: ' || :new.sal);

dbms\_output.put\_line(' Difference ' || sal\_diff);

END;

/

2. set up the following data structures for this example to work:

CREATE TABLE Project\_tab (

Prj\_level NUMBER,

Projno NUMBER,

Resp\_dept NUMBER);

CREATE TABLE Emp\_tab (

Empno NUMBER NOT NULL,

Ename VARCHAR2(10),

Job VARCHAR2(9),

Mgr NUMBER(4),

Hiredate DATE,

Sal NUMBER(7,2),

Comm NUMBER(7,2),

Deptno NUMBER(2) NOT NULL);

CREATE TABLE Dept\_tab (

Deptno NUMBER(2) NOT NULL,

Dname VARCHAR2(14),

Loc VARCHAR2(13),

Mgr\_no NUMBER,

Dept\_type NUMBER);

The following example shows an INSTEAD OF trigger for inserting rows into the MANAGER\_INFO view.

CREATE OR REPLACE VIEW manager\_info AS

SELECT e.ename, e.empno, d.dept\_type, d.deptno, p.prj\_level,

p.projno

FROM Emp\_tab e, Dept\_tab d, Project\_tab p

WHERE e.empno = d.mgr\_no

AND d.deptno = p.resp\_dept;

CREATE OR REPLACE TRIGGER manager\_info\_insert

INSTEAD OF INSERT ON manager\_info

REFERENCING NEW AS n -- new manager information

FOR EACH ROW

DECLARE

rowcnt number;

BEGIN

SELECT COUNT(\*) INTO rowcnt FROM Emp\_tab WHERE empno = :n.empno;

IF rowcnt = 0 THEN

INSERT INTO Emp\_tab (empno,ename) VALUES (:n.empno, :n.ename);

ELSE

UPDATE Emp\_tab SET Emp\_tab.ename = :n.ename

WHERE Emp\_tab.empno = :n.empno;

END IF;

SELECT COUNT(\*) INTO rowcnt FROM Dept\_tab WHERE deptno = :n.deptno;

IF rowcnt = 0 THEN

INSERT INTO Dept\_tab (deptno, dept\_type)

VALUES(:n.deptno, :n.dept\_type);

ELSE

UPDATE Dept\_tab SET Dept\_tab.dept\_type = :n.dept\_type

WHERE Dept\_tab.deptno = :n.deptno;

END IF;

SELECT COUNT(\*) INTO rowcnt FROM Project\_tab

WHERE Project\_tab.projno = :n.projno;

IF rowcnt = 0 THEN

INSERT INTO Project\_tab (projno, prj\_level)

VALUES(:n.projno, :n.prj\_level);

ELSE

UPDATE Project\_tab SET Project\_tab.prj\_level = :n.prj\_level

WHERE Project\_tab.projno = :n.projno;

END IF;

END;

The actions shown for rows being inserted into the MANAGER\_INFO view first test to see if appropriate rows already exist in the base tables from which MANAGER\_INFO is derived. The actions then insert new rows or update existing rows, as appropriate. Similar triggers can specify appropriate actions for UPDATE and DELETE.

3. For example, consider a department view that contains a nested table of employees.

CREATE OR REPLACE VIEW Dept\_view AS

SELECT d.Deptno, d.Dept\_type, d.Dept\_name,

CAST (MULTISET ( SELECT e.Empno, e.Empname, e.Salary)

FROM Emp\_tab e

WHERE e.Deptno = d.Deptno) AS Amp\_list\_ Emplist

FROM Dept\_tab d;

The following example shows how an insert trigger might be written:

CREATE OR REPLACE TRIGGER Dept\_emplist\_tr

INSTEAD OF INSERT ON NESTED TABLE Emplist OF Dept\_view

REFERENCING NEW AS Employee

PARENT AS Department

FOR EACH ROW

BEGIN

*-- The insert on the nested table is translated to an insert on the base table:*

INSERT INTO Emp\_tab VALUES (

:Employee.Empno, :Employee.Empname,:Employee.Salary, :Department.Deptno);

END;

Any INSERT into the nested table fires the trigger, and the Emp\_tab table is filled with the correct values. For example:

INSERT INTO TABLE (SELECT d.Emplist FROM Dept\_view d WHERE Deptno = 10)

VALUES (1001, 'John Glenn', 10000);

##### 4. Example: Monitoring Logons with a Trigger

|  |
| --- |
| **Note:**  You may need to set up data structures similar to the following for certain examples to work:  CONNECT system/manager  GRANT ADMINISTER DATABASE TRIGGER TO scott;  CONNECT scott/tiger  CREATE TABLE audit\_table (  seq number,  user\_at VARCHAR2(10),  time\_now DATE,  term VARCHAR2(10),  job VARCHAR2(10),  proc VARCHAR2(10),  enum NUMBER); |

CREATE OR REPLACE PROCEDURE foo (c VARCHAR2) AS

BEGIN

INSERT INTO Audit\_table (user\_at) VALUES(c);

END;

CREATE OR REPLACE TRIGGER logontrig AFTER LOGON ON DATABASE

-- Just call an existing procedure. The ORA\_LOGIN\_USER is a function

-- that returns information about the event that fired the trigger.

CALL foo (ora\_login\_user)

/

##### 5. Example: Calling a Java Procedure from a Trigger

Although triggers are declared using PL/SQL, they can call procedures in other languages, such as Java:

CREATE OR REPLACE PROCEDURE Before\_delete (Id IN NUMBER, Ename VARCHAR2)

IS language Java

name 'thjvTriggers.beforeDelete (oracle.sql.NUMBER, oracle.sql.CHAR)';

CREATE OR REPLACE TRIGGER Pre\_del\_trigger BEFORE DELETE ON Tab

FOR EACH ROW

CALL Before\_delete (:old.Id, :old.Ename)

/

The corresponding Java file is thjvTriggers.java:

import java.sql.\*

import java.io.\*

import oracle.sql.\*

import oracle.oracore.\*

public class thjvTriggers

{

public state void

beforeDelete (NUMBER old\_id, CHAR old\_name)

Throws SQLException, CoreException

{

Connection conn = JDBCConnection.defaultConnection();

Statement stmt = conn.CreateStatement();

String sql = "insert into logtab values

("+ old\_id.intValue() +", '"+ old\_ename.toString() + ", BEFORE DELETE');

stmt.executeUpdate (sql);

stmt.close();

return;

}

}

#### 6. Example: Modifying LOB Columns with a Trigger

You can treat LOB columns the same as other columns, using regular SQL and PL/SQL functions with CLOB columns, and calls to the DBMS\_LOB package with BLOBcolumns:

drop table tab1;

create table tab1 (c1 clob);

insert into tab1 values ('<h1>HTML Document Fragment</h1><p>Some text.');

create or replace trigger trg1

before update on tab1

for each row

begin

dbms\_output.put\_line('Old value of CLOB column: '||:OLD.c1);

dbms\_output.put\_line('Proposed new value of CLOB column: '||:NEW.c1);

-- Previously, we couldn't change the new value for a LOB.

-- Now, we can replace it, or construct a new value using SUBSTR, INSTR...

-- operations for a CLOB, or DBMS\_LOB calls for a BLOB.

:NEW.c1 := :NEW.c1 || to\_clob('<hr><p>Standard footer paragraph.');

dbms\_output.put\_line('Final value of CLOB column: '||:NEW.c1);

end;

/

set serveroutput on;

update tab1 set c1 = '<h1>Different Document Fragment</h1><p>Different text.';

select \* from tab1;

#### 7. INSTEAD OF Triggers on Nested Table View Columns

|  |
| --- |
| **Note:**  You may need to set up the following data structures for certain examples to work:  CREATE TABLE new (  field1 NUMBER,  field2 VARCHAR2(20)); |

CREATE OR REPLACE TRIGGER Print\_salary\_changes

BEFORE UPDATE ON new

REFERENCING new AS Newest

FOR EACH ROW

BEGIN

:Newest.Field2 := TO\_CHAR (:newest.field1);

END;

### 8. Triggers and Handling Remote Exceptions

A trigger that accesses a remote site cannot do remote exception handling if the network link is unavailable. For example:

CREATE OR REPLACE TRIGGER Example

AFTER INSERT ON Emp\_tab

FOR EACH ROW

BEGIN

INSERT INTO Emp\_tab@Remote -- <- compilation fails here

VALUES ('x'); -- when dblink is inaccessible

EXCEPTION

WHEN OTHERS THEN

INSERT INTO Emp\_log

VALUES ('x');

END;

Because stored procedures are stored in a compiled form, the work-around for the previous example is as follows:

CREATE OR REPLACE TRIGGER Example

AFTER INSERT ON Emp\_tab

FOR EACH ROW

BEGIN

Insert\_row\_proc;

END;

CREATE OR REPLACE PROCEDURE Insert\_row\_proc AS

BEGIN

INSERT INTO Emp\_tab@Remote

VALUES ('x');

EXCEPTION

WHEN OTHERS THEN

INSERT INTO Emp\_log

VALUES ('x');

END;

9. CREATE OR REPLACE TRIGGER Emp\_count

AFTER DELETE ON Emp\_tab

FOR EACH ROW

DECLARE

n INTEGER;

BEGIN

SELECT COUNT(\*) INTO n FROM Emp\_tab;

DBMS\_OUTPUT.PUT\_LINE(' There are now ' || n ||

' employees.');

END;

## 9. Viewing Information About Triggers

CREATE OR REPLACE TRIGGER Reorder

AFTER UPDATE OF Parts\_on\_hand ON Inventory

FOR EACH ROW

WHEN(new.Parts\_on\_hand < new.Reorder\_point)

DECLARE

x NUMBER;

BEGIN

SELECT COUNT(\*) INTO x

FROM Pending\_orders

WHERE Part\_no = :new.Part\_no;

IF x = 0 THEN

INSERT INTO Pending\_orders

VALUES (:new.Part\_no, :new.Reorder\_quantity,

sysdate);

END IF;

END;

The following two queries return information about the REORDER trigger:

SELECT Trigger\_type, Triggering\_event, Table\_name

FROM USER\_TRIGGERS

WHERE Trigger\_name = 'REORDER';

TYPE TRIGGERING\_STATEMENT TABLE\_NAME

---------------- -------------------------- ------------

AFTER EACH ROW UPDATE INVENTORY

SELECT Trigger\_body

FROM USER\_TRIGGERS

WHERE Trigger\_name = 'REORDER';

TRIGGER\_BODY

--------------------------------------------

DECLARE

x NUMBER;

BEGIN

SELECT COUNT(\*) INTO x

FROM Pending\_orders

WHERE Part\_no = :new.Part\_no;

IF x = 0

THEN INSERT INTO Pending\_orders

VALUES (:new.Part\_no, :new.Reorder\_quantity,

sysdate);

END IF;

END;

#### 10. Auditing with Triggers: Example

The following example demonstrates a trigger that audits modifications to the Emp\_tab table for each row. It requires that a "reason code" be stored in a global package variable before the update. This shows how triggers can be used to provide value-based auditing and how to use public package variables.

|  |
| --- |
| **Note:**  You may need to set up the following data structures for the examples to work:  CREATE OR REPLACE PACKAGE Auditpackage AS  Reason VARCHAR2(10);  PROCEDURE Set\_reason(Reason VARCHAR2);  END;  CREATE TABLE Emp99 (  Empno NOT NULL NUMBER(4),  Ename VARCHAR2(10),  Job VARCHAR2(9),  Mgr NUMBER(4),  Hiredate DATE,  Sal NUMBER(7,2),  Comm NUMBER(7,2),  Deptno NUMBER(2),  Bonus NUMBER,  Ssn NUMBER,  Job\_classification NUMBER);    CREATE TABLE Audit\_employee (  Oldssn NUMBER,  Oldname VARCHAR2(10),  Oldjob VARCHAR2(2),  Oldsal NUMBER,  Newssn NUMBER,  Newname VARCHAR2(10),  Newjob VARCHAR2(2),  Newsal NUMBER,  Reason VARCHAR2(10),  User1 VARCHAR2(10),  Systemdate DATE); |

CREATE OR REPLACE TRIGGER Audit\_employee

AFTER INSERT OR DELETE OR UPDATE ON Emp99

FOR EACH ROW

BEGIN

/\* AUDITPACKAGE is a package with a public package

variable REASON. REASON could be set by the

application by a command such as EXECUTE

AUDITPACKAGE.SET\_REASON(reason\_string). Note that a

package variable has state for the duration of a

session and that each session has a separate copy of

all package variables. \*/

IF Auditpackage.Reason IS NULL THEN

Raise\_application\_error(-20201, 'Must specify reason'

|| ' with AUDITPACKAGE.SET\_REASON(Reason\_string)');

END IF;

*/\* If the preceding conditional evaluates to TRUE, the*

*user-specified error number and message is raised,*

*the trigger stops execution, and the effects of the*

*triggering statement are rolled back. Otherwise, a*

*new row is inserted into the predefined auditing*

*table named AUDIT\_EMPLOYEE containing the existing*

*and new values of the Emp\_tab table and the reason code*

*defined by the REASON variable of AUDITPACKAGE. Note*

*that the "old" values are NULL if triggering*

*statement is an INSERT and the "new" values are NULL*

*if the triggering statement is a DELETE. \*/*

INSERT INTO Audit\_employee VALUES

(:old.Ssn, :old.Ename, :old.Job\_classification, :old.Sal,

:new.Ssn, :new.Ename, :new.Job\_classification, :new.Sal,

auditpackage.Reason, User, Sysdate );

END;

Optionally, you can also set the reason code back to NULL if you wanted to force the reason code to be set for every update. The following simple AFTER statement trigger sets the reason code back to NULL after the triggering statement is run:

CREATE OR REPLACE TRIGGER Audit\_employee\_reset

AFTER INSERT OR DELETE OR UPDATE ON Emp\_tab

BEGIN

auditpackage.set\_reason(NULL);

END;

11. This next trigger also uses triggers to do auditing. It tracks changes made to the Emp\_tab table and stores this information in AUDIT\_TABLE and AUDIT\_TABLE\_VALUES.

|  |
| --- |
| **Note:**  You may need to set up the following data structures for the example to work:  CREATE TABLE Audit\_table (  Seq NUMBER,  User\_at VARCHAR2(10),  Time\_now DATE,  Term VARCHAR2(10),  Job VARCHAR2(10),  Proc VARCHAR2(10),  enum NUMBER);  CREATE SEQUENCE Audit\_seq;  CREATE TABLE Audit\_table\_values (  Seq NUMBER,  Dept NUMBER,  Dept1 NUMBER,  Dept2 NUMBER); |

CREATE OR REPLACE TRIGGER Audit\_emp

AFTER INSERT OR UPDATE OR DELETE ON Emp\_tab

FOR EACH ROW

DECLARE

Time\_now DATE;

Terminal CHAR(10);

BEGIN

-- get current time, and the terminal of the user:

Time\_now := SYSDATE;

Terminal := USERENV('TERMINAL');

-- record new employee primary key

IF INSERTING THEN

INSERT INTO Audit\_table

VALUES (Audit\_seq.NEXTVAL, User, Time\_now,

Terminal, 'Emp\_tab', 'INSERT', :new.Empno);

*-- record primary key of the deleted row:*

ELSIF DELETING THEN

INSERT INTO Audit\_table

VALUES (Audit\_seq.NEXTVAL, User, Time\_now,

Terminal, 'Emp\_tab', 'DELETE', :old.Empno);

*-- for updates, record the primary key*

*-- of the row being updated:*

ELSE

INSERT INTO Audit\_table

VALUES (audit\_seq.NEXTVAL, User, Time\_now,

Terminal, 'Emp\_tab', 'UPDATE', :old.Empno);

*-- and for SAL and DEPTNO, record old and new values:*

IF UPDATING ('SAL') THEN

INSERT INTO Audit\_table\_values

VALUES (Audit\_seq.CURRVAL, 'SAL',

:old.Sal, :new.Sal);

ELSIF UPDATING ('DEPTNO') THEN

INSERT INTO Audit\_table\_values

VALUES (Audit\_seq.CURRVAL, 'DEPTNO',

:old.Deptno, :new.DEPTNO);

END IF;

END IF;

END;

##### 12. Foreign Key Trigger for Child Table

CREATE OR REPLACE TRIGGER Emp\_dept\_check

BEFORE INSERT OR UPDATE OF Deptno ON Emp\_tab

FOR EACH ROW WHEN (new.Deptno IS NOT NULL)

*-- Before a row is inserted, or DEPTNO is updated in the Emp\_tab*

*-- table, fire this trigger to verify that the new foreign*

*-- key value (DEPTNO) is present in the Dept\_tab table.*

DECLARE

Dummy INTEGER; *-- to be used for cursor fetch*

Invalid\_department EXCEPTION;

Valid\_department EXCEPTION;

Mutating\_table EXCEPTION;

PRAGMA EXCEPTION\_INIT (Mutating\_table, -4091);

*-- Cursor used to verify parent key value exists. If*

*-- present, lock parent key's row so it can't be*

*-- deleted by another transaction until this*

*-- transaction is committed or rolled back.*

CURSOR Dummy\_cursor (Dn NUMBER) IS

SELECT Deptno FROM Dept\_tab

WHERE Deptno = Dn

FOR UPDATE OF Deptno;

BEGIN

OPEN Dummy\_cursor (:new.Deptno);

FETCH Dummy\_cursor INTO Dummy;

*-- Verify parent key. If not found, raise user-specified*

*-- error number and message. If found, close cursor*

*-- before allowing triggering statement to complete:*

IF Dummy\_cursor%NOTFOUND THEN

RAISE Invalid\_department;

ELSE

RAISE valid\_department;

END IF;

CLOSE Dummy\_cursor;

EXCEPTION

WHEN Invalid\_department THEN

CLOSE Dummy\_cursor;

Raise\_application\_error(-20000, 'Invalid Department'

|| ' Number' || TO\_CHAR(:new.deptno));

WHEN Valid\_department THEN

CLOSE Dummy\_cursor;

WHEN Mutating\_table THEN

NULL;

END;

13. The following trigger is defined on the DEPT\_TAB table to enforce the UPDATE and DELETE RESTRICT referential action on the primary key of the DEPT\_TAB table:

CREATE OR REPLACE TRIGGER Dept\_restrict

BEFORE DELETE OR UPDATE OF Deptno ON Dept\_tab

FOR EACH ROW

*-- Before a row is deleted from Dept\_tab or the primary key*

*-- (DEPTNO) of Dept\_tab is updated, check for dependent*

*-- foreign key values in Emp\_tab; rollback if any are found.*

DECLARE

Dummy INTEGER; *-- to be used for cursor fetch*

Employees\_present EXCEPTION;

employees\_not\_present EXCEPTION;

*-- Cursor used to check for dependent foreign key values.*

CURSOR Dummy\_cursor (Dn NUMBER) IS

SELECT Deptno FROM Emp\_tab WHERE Deptno = Dn;

BEGIN

OPEN Dummy\_cursor (:old.Deptno);

FETCH Dummy\_cursor INTO Dummy;

*-- If dependent foreign key is found, raise user-specified*

*-- error number and message. If not found, close cursor*

*-- before allowing triggering statement to complete.*

IF Dummy\_cursor%FOUND THEN

RAISE Employees\_present;  *-- dependent rows exist*

ELSE

RAISE Employees\_not\_present; *-- no dependent rows*

END IF;

CLOSE Dummy\_cursor;

EXCEPTION

WHEN Employees\_present THEN

CLOSE Dummy\_cursor;

Raise\_application\_error(-20001, 'Employees Present in'

|| ' Department ' || TO\_CHAR(:old.DEPTNO));

WHEN Employees\_not\_present THEN

CLOSE Dummy\_cursor;

END;

**Caution:**

This trigger does not work with self-referential tables (tables with both the primary/unique key and the foreign key). Also, this trigger does not allow triggers to cycle (such as, A fires B fires A).

##### 14. UPDATE and DELETE SET NULL Triggers for Parent Table: Example

The following trigger is defined on the DEPT\_TAB table to enforce the UPDATE and DELETE SET NULL referential action on the primary key of the DEPT\_TAB table:

CREATE OR REPLACE TRIGGER Dept\_set\_null

AFTER DELETE OR UPDATE OF Deptno ON Dept\_tab

FOR EACH ROW

-- Before a row is deleted from Dept\_tab or the primary key

-- (DEPTNO) of Dept\_tab is updated, set all corresponding

-- dependent foreign key values in Emp\_tab to NULL:

BEGIN

IF UPDATING AND :OLD.Deptno != :NEW.Deptno OR DELETING THEN

UPDATE Emp\_tab SET Emp\_tab.Deptno = NULL

WHERE Emp\_tab.Deptno = :old.Deptno;

END IF;

END;

##### 15. DELETE Cascade Trigger for Parent Table: Example

The following trigger on the DEPT\_TAB table enforces the DELETE CASCADE referential action on the primary key of the DEPT\_TAB table:

CREATE OR REPLACE TRIGGER Dept\_del\_cascade

AFTER DELETE ON Dept\_tab

FOR EACH ROW

-- Before a row is deleted from Dept\_tab, delete all

-- rows from the Emp\_tab table whose DEPTNO is the same as

-- the DEPTNO being deleted from the Dept\_tab table:

BEGIN

DELETE FROM Emp\_tab

WHERE Emp\_tab.Deptno = :old.Deptno;

END;

|  |
| --- |
| **Note:**  Typically, the code for DELETE CASCADE is combined with the code for UPDATE SET NULL or UPDATE SET DEFAULT to account for both updates and deletes. |

##### 16. UPDATE Cascade Trigger for Parent Table: Example

The following trigger ensures that if a department number is updated in the Dept\_tab table, then this change is propagated to dependent foreign keys in the Emp\_tabtable:

-- Generate a sequence number to be used as a flag for

-- determining if an update has occurred on a column:

CREATE SEQUENCE Update\_sequence

INCREMENT BY 1 MAXVALUE 5000

CYCLE;

CREATE OR REPLACE PACKAGE Integritypackage AS

Updateseq NUMBER;

END Integritypackage;

CREATE OR REPLACE PACKAGE BODY Integritypackage AS

END Integritypackage;

-- create flag col:

ALTER TABLE Emp\_tab ADD Update\_id NUMBER;

CREATE OR REPLACE TRIGGER Dept\_cascade1 BEFORE UPDATE OF Deptno ON Dept\_tab

DECLARE

Dummy NUMBER;

-- Before updating the Dept\_tab table (this is a statement

-- trigger), generate a new sequence number and assign

-- it to the public variable UPDATESEQ of a user-defined

-- package named INTEGRITYPACKAGE:

BEGIN

SELECT Update\_sequence.NEXTVAL

INTO Dummy

FROM dual;

Integritypackage.Updateseq := Dummy;

END;

CREATE OR REPLACE TRIGGER Dept\_cascade2 AFTER DELETE OR UPDATE

OF Deptno ON Dept\_tab FOR EACH ROW

-- For each department number in Dept\_tab that is updated,

-- cascade the update to dependent foreign keys in the

-- Emp\_tab table. Only cascade the update if the child row

-- has not already been updated by this trigger:

BEGIN

IF UPDATING THEN

UPDATE Emp\_tab

SET Deptno = :new.Deptno,

Update\_id = Integritypackage.Updateseq --from 1st

WHERE Emp\_tab.Deptno = :old.Deptno

AND Update\_id IS NULL;

/\* only NULL if not updated by the 3rd trigger

fired by this same triggering statement \*/

END IF;

IF DELETING THEN

-- Before a row is deleted from Dept\_tab, delete all

-- rows from the Emp\_tab table whose DEPTNO is the same as

-- the DEPTNO being deleted from the Dept\_tab table:

DELETE FROM Emp\_tab

WHERE Emp\_tab.Deptno = :old.Deptno;

END IF;

END;

CREATE OR REPLACE TRIGGER Dept\_cascade3 AFTER UPDATE OF Deptno ON Dept\_tab

BEGIN UPDATE Emp\_tab

SET Update\_id = NULL

WHERE Update\_id = Integritypackage.Updateseq;

END;

|  |
| --- |
| **Note:**  Because this trigger updates the Emp\_tab table, the Emp\_dept\_check trigger, if enabled, is also fired. The resulting mutating table error is trapped by the Emp\_dept\_check trigger. You should carefully test any triggers that require error trapping to succeed to ensure that they always work properly in your environment. |

#### 17. Trigger for Complex Check Constraints: Example

Triggers can enforce integrity rules other than referential integrity. For example, this trigger performs a complex check before allowing the triggering statement to run.

|  |
| --- |
| **Note:**  You may need to set up the following data structures for the example to work:  CREATE TABLE Salgrade (  Grade NUMBER,  Losal NUMBER,  Hisal NUMBER,  Job\_classification NUMBER) |

CREATE OR REPLACE TRIGGER Salary\_check

BEFORE INSERT OR UPDATE OF Sal, Job ON Emp99

FOR EACH ROW

DECLARE

Minsal NUMBER;

Maxsal NUMBER;

Salary\_out\_of\_range EXCEPTION;

BEGIN

/\* Retrieve the minimum and maximum salary for the

employee's new job classification from the SALGRADE

table into MINSAL and MAXSAL: \*/

SELECT Minsal, Maxsal INTO Minsal, Maxsal FROM Salgrade

WHERE Job\_classification = :new.Job;

/\* If the employee's new salary is less than or greater

than the job classification's limits, the exception is

raised. The exception message is returned and the

pending INSERT or UPDATE statement that fired the

trigger is rolled back:\*/

IF (:new.Sal < Minsal OR :new.Sal > Maxsal) THEN

RAISE Salary\_out\_of\_range;

END IF;

EXCEPTION

WHEN Salary\_out\_of\_range THEN

Raise\_application\_error (-20300,

'Salary '||TO\_CHAR(:new.Sal)||' out of range for '

||'job classification '||:new.Job

||' for employee '||:new.Ename);

WHEN NO\_DATA\_FOUND THEN

Raise\_application\_error(-20322,

'Invalid Job Classification '

||:new.Job\_classification);

END;

#### 18. Complex Security Authorizations and Triggers: Example

|  |
| --- |
| **Note:**  You may need to set up the following data structures for the example to work:  CREATE TABLE Company\_holidays (Day DATE); |

CREATE OR REPLACE TRIGGER Emp\_permit\_changes

BEFORE INSERT OR DELETE OR UPDATE ON Emp99

DECLARE

Dummy INTEGER;

Not\_on\_weekends EXCEPTION;

Not\_on\_holidays EXCEPTION;

Non\_working\_hours EXCEPTION;

BEGIN

*/\* check for weekends: \*/*

IF (TO\_CHAR(Sysdate, 'DY') = 'SAT' OR

TO\_CHAR(Sysdate, 'DY') = 'SUN') THEN

RAISE Not\_on\_weekends;

END IF;

*/\* check for company holidays:\*/*

SELECT COUNT(\*) INTO Dummy FROM Company\_holidays

WHERE TRUNC(Day) = TRUNC(Sysdate);

*/\* TRUNC gets rid of time parts of dates: \*/*

IF dummy > 0 THEN

RAISE Not\_on\_holidays;

END IF;

*/\* Check for work hours (8am to 6pm): \*/*

IF (TO\_CHAR(Sysdate, 'HH24') < 8 OR

TO\_CHAR(Sysdate, 'HH24') > 18) THEN

RAISE Non\_working\_hours;

END IF;

EXCEPTION

WHEN Not\_on\_weekends THEN

Raise\_application\_error(-20324,'May not change '

||'employee table during the weekend');

WHEN Not\_on\_holidays THEN

Raise\_application\_error(-20325,'May not change '

||'employee table during a holiday');

WHEN Non\_working\_hours THEN

Raise\_application\_error(-20326,'May not change '

||'Emp\_tab table during non-working hours');

END;

#### 19. Derived Column Values and Triggers: Example

The following example illustrates how a trigger can be used to derive new column values for a table whenever a row is inserted or updated.

|  |
| --- |
| **Note:**  You may need to set up the following data structures for the example to work:  ALTER TABLE Emp99 ADD(  Uppername VARCHAR2(20),  Soundexname VARCHAR2(20)); |

CREATE OR REPLACE TRIGGER Derived

BEFORE INSERT OR UPDATE OF Ename ON Emp99

*/\* Before updating the ENAME field, derive the values for*

*the UPPERNAME and SOUNDEXNAME fields. Users should be*

*restricted from updating these fields directly: \*/*

FOR EACH ROW

BEGIN

:new.Uppername := UPPER(:new.Ename);

:new.Soundexname := SOUNDEX(:new.Ename);

END;

#### 20. Building Complex Updatable Views Using Triggers: Example

Consider a library system where books are arranged under their respective titles. The library consists of a collection of book type objects. The following example explains the schema.

CREATE OR REPLACE TYPE Book\_t AS OBJECT

(

Booknum NUMBER,

Title VARCHAR2(20),

Author VARCHAR2(20),

Available CHAR(1)

);

CREATE OR REPLACE TYPE Book\_list\_t AS TABLE OF Book\_t;

Assume that the following tables exist in the relational schema:

Table Book\_table (Booknum, Section, Title, Author, Available)

| **Booknum** | **Section** | **Title** | **Author** | **Available** |
| --- | --- | --- | --- | --- |
| 121001 | Classic | Iliad | Homer | Y |
| 121002 | Novel | Gone With the Wind | Mitchell M | N |

Library consists of library\_table(section).

| **Section** |
| --- |
| Geography |
| Classic |

You can define a complex view over these tables to create a logical view of the library with sections and a collection of books in each section.

CREATE OR REPLACE VIEW Library\_view AS

SELECT i.Section, CAST (MULTISET (

SELECT b.Booknum, b.Title, b.Author, b.Available

FROM Book\_table b

WHERE b.Section = i.Section) AS Book\_list\_t) BOOKLIST

FROM Library\_table i;

Make this view updatable by defining an INSTEAD OF trigger over the view.

CREATE OR REPLACE TRIGGER Library\_trigger INSTEAD OF INSERT ON Library\_view FOR

EACH ROW

Bookvar BOOK\_T;

i INTEGER;

BEGIN

INSERT INTO Library\_table VALUES (:NEW.Section);

FOR i IN 1..:NEW.Booklist.COUNT LOOP

Bookvar := Booklist(i);

INSERT INTO book\_table

VALUES ( Bookvar.booknum, :NEW.Section, Bookvar.Title, Bookvar.Author,

bookvar.Available);

END LOOP;

END;

/

The library\_view is an updatable view, and any INSERTs on the view are handled by the trigger that gets fired automatically. For example:

INSERT INTO Library\_view VALUES ('History', book\_list\_t(book\_t(121330,

'Alexander', 'Mirth', 'Y');

##### 21. Fine-Grained Access Control Using Triggers: Example

In the example that follows, procedure set\_ctx sets the application context based on the user profile. The trigger setexpensectx ensures that the context is set for every user.

CONNECT secdemo/secdemo

CREATE OR REPLACE CONTEXT Expenses\_reporting USING Secdemo.Exprep\_ctx;

*REM =================================================================*

*REM Creation of the package which implements the context:*

*REM =================================================================*

CREATE OR REPLACE PACKAGE Exprep\_ctx AS

PROCEDURE Set\_ctx;

END;

SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY Exprep\_ctx IS

PROCEDURE Set\_ctx IS

Empnum NUMBER;

Countrec NUMBER;

Cc NUMBER;

Role VARCHAR2(20);

BEGIN

*-- SET emp\_number:*

SELECT Employee\_id INTO Empnum FROM Employee

WHERE Last\_name = SYS\_CONTEXT('userenv', 'session\_user');

DBMS\_SESSION.SET\_CONTEXT('expenses\_reporting','emp\_number', Empnum);

*-- SET ROLE:*

SELECT COUNT (\*) INTO Countrec FROM Cost\_center WHERE Manager\_id=Empnum;

IF (countrec > 0) THEN

DBMS\_SESSION.SET\_CONTEXT('expenses\_reporting','exp\_role','MANAGER');

ELSE

DBMS\_SESSION.SET\_CONTEXT('expenses\_reporting','exp\_role','EMPLOYEE');

END IF;

*-- SET cc\_number:*

SELECT Cost\_center\_id INTO Cc FROM Employee

WHERE Last\_name = SYS\_CONTEXT('userenv','session\_user');

DBMS\_SESSION.SET\_CONTEXT(expenses\_reporting','cc\_number',Cc);

END;

END;

**CALL Syntax**

CREATE OR REPLACE TRIGGER Secdemo.Setexpseetx

AFTER LOGON ON DATABASE

CALL Secdemo.Exprep\_etx.Set\_otx