Studies for Implementing Triggers

Objectives

After completing this lesson, you should be able to do the following:

- Enhance database security with triggers
- Audit data changes using data manipulation language (DML) triggers
- Enforce data integrity with DML triggers
- Maintain referential integrity using triggers
- Use triggers to replicate data between tables
- Use triggers to automate computation of derived data
- Provide event-logging capabilities using triggers

Controlling Security Within the Server

Using database security with the `GRANT` statement

```
GRANT SELECT, INSERT, UPDATE, DELETE
 ON    employees
 TO    clerk;                -- database role
GRANT clerk TO scott;
```

Controlling Security with a Database Trigger

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
dummy PLS_INTEGER;
BEGIN
 IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT','SUN')) THEN
   RAISE_APPLICATION_ERROR(-20506,'You may only
     change data during normal business hours.');
 END IF;
 SELECT COUNT(*) INTO dummy FROM holiday
 WHERE holiday_date = TRUNC (SYSDATE);
 IF dummy > 0 THEN
   RAISE_APPLICATION_ERROR(-20507,
     'You may not change data on a holiday.');
 END IF;
END;
/
```

Using the Server Facility to Audit Data Operations

The Oracle server stores the audit information in a data dictionary table or an operating system file.

```
AUDIT INSERT, UPDATE, DELETE
  ON  departments
  BY ACCESS
WHENEVER SUCCESSFUL;
```

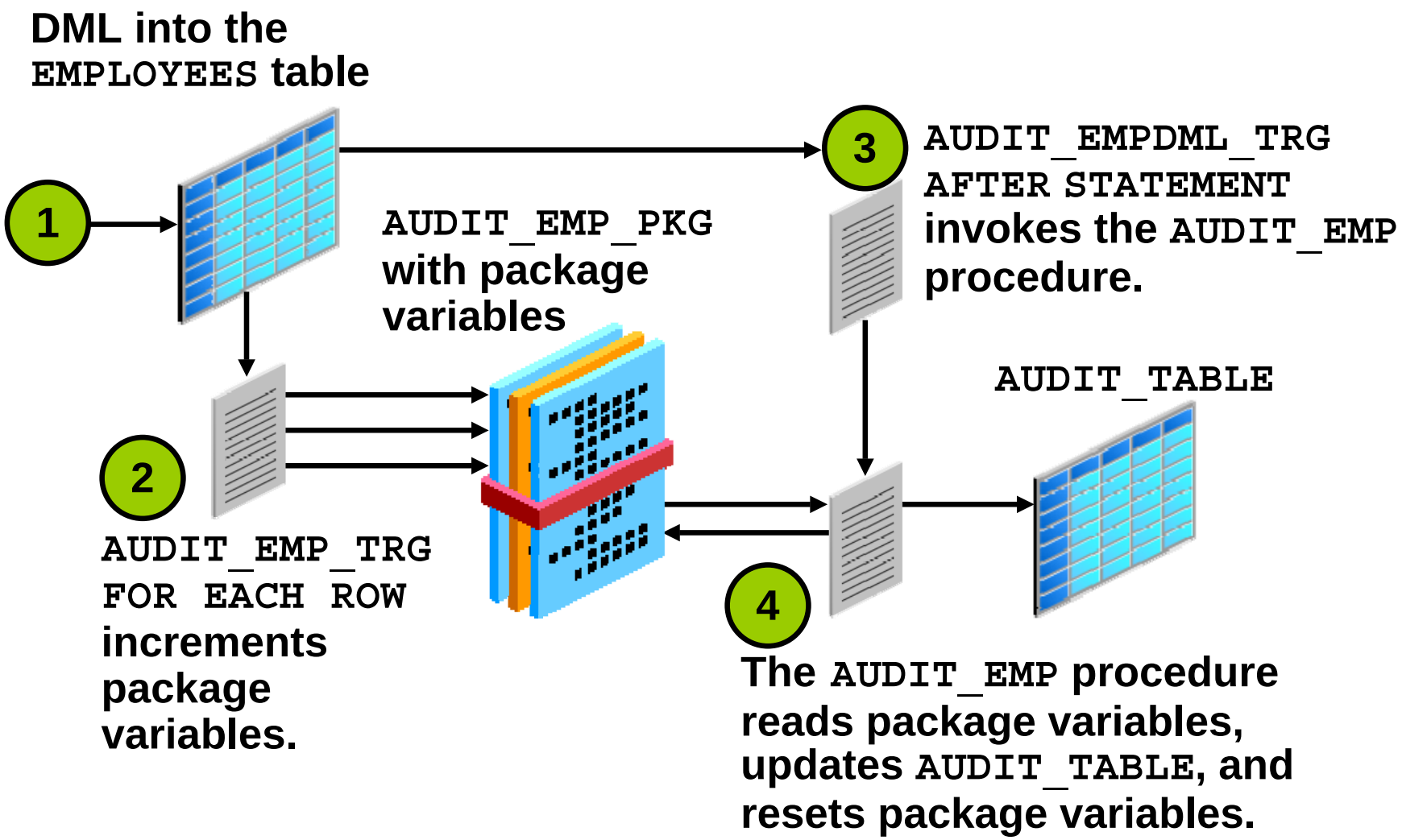
```
AUDIT INSERT, succeeded.
```

Auditing by Using a Trigger

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE
ON employees FOR EACH ROW
BEGIN
 IF (audit_emp_pkg. reason IS NULL) THEN
   RAISE_APPLICATION_ERROR (-20059, 'Specify a
    reason for operation through the procedure
    AUDIT_EMP_PKG.SET_REASON to proceed.');
 ELSE
   INSERT INTO audit_emp_table (user_name,
     timestamp, id, old_last_name, new_last_name,
     old_salary, new_salary, comments)
   VALUES (USER, SYSDATE, :OLD.employee_id,
     :OLD.last_name, :NEW.last_name,:OLD.salary,
     :NEW.salary, audit_emp_pkg.reason);
 END IF;
END;
```

```
CREATE OR REPLACE TRIGGER cleanup_audit_emp
 AFTER INSERT OR UPDATE OR DELETE ON employees
BEGIN audit_emp_package.g_reason := NULL;
END;
```

Auditing Triggers by Using Package Constructs



Auditing Triggers by Using Package Constructs

AFTER statement trigger:

```
CREATE OR REPLACE TRIGGER audit_empdml_trg
AFTER UPDATE OR INSERT OR DELETE on employees
BEGIN
  audit_emp;       -- write the audit data
END audit_emp_tab;
/
```

AFTER row trigger:

```
CREATE OR REPLACE TRIGGER audit_emp_trg
AFTER UPDATE OR INSERT OR DELETE ON EMPLOYEES
FOR EACH ROW
-- Call Audit package to maintain counts
CALL audit_emp_pkg.set(INSERTING,UPDATING,DELETING);
/
```

AUDIT_PKG Package

```
CREATE OR REPLACE PACKAGE audit_emp_pkg IS
  delcnt PLS_INTEGER := 0;
  inscnt PLS_INTEGER := 0;
  updcnt  PLS_INTEGER := 0;
  PROCEDURE init;
  PROCEDURE set(i BOOLEAN,u BOOLEAN,d BOOLEAN);
END audit_emp_pkg;
/
CREATE OR REPLACE PACKAGE BODY audit_emp_pkg IS
 PROCEDURE init IS
  BEGIN
    inscnt := 0; updcnt := 0; delcnt := 0;
  END;
  PROCEDURE set(i BOOLEAN,u BOOLEAN,d BOOLEAN) IS
  BEGIN
    IF i THEN inscnt := inscnt + 1;
    ELSIF d THEN delcnt := delcnt + 1;
    ELSE upd := updcnt + 1;
    END IF;
  END;
END audit_emp_pkg;
/
```

AUDIT_TABLE Table and AUDIT_EMP Procedure

```
CREATE TABLE audit_table (
 USER_NAME   VARCHAR2(30),
 TABLE_NAME  VARCHAR2(30),
 INS         NUMBER,
 UPD         NUMBER,
 DEL         NUMBER)
/
CREATE OR REPLACE PROCEDURE audit_emp IS
BEGIN
  IF  delcnt + inscnt + updcnt <> 0 THEN
    UPDATE audit_table
     SET del = del + audit_emp_pkg.delcnt,
         ins = ins + audit_emp_pkg.inscnt,
         upd = upd + audit_emp_pkg.updcnt
    WHERE user_name = USER
    AND   table_name = 'EMPLOYEES';
    audit_emp_pkg.init;
  END IF;
END audit_emp;
/
```

Enforcing Data Integrity Within the Server

```
ALTER TABLE employees ADD
  CONSTRAINT ck_salary CHECK (salary >= 500);
```

```
Table altered.
```

Protecting Data Integrity with a Trigger

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20508,
      'Do not decrease salary.');
END;
/
```

Enforcing Referential Integrity Within the Server

```
ALTER TABLE employees
  ADD CONSTRAINT emp_deptno_fk
  FOREIGN KEY (department_id)
     REFERENCES departments(department_id)
ON DELETE CASCADE;
```

Protecting Referential Integrity with a Trigger

```
CREATE OR REPLACE TRIGGER cascade_updates
 AFTER UPDATE OF department_id ON departments
 FOR EACH ROW
BEGIN
 UPDATE employees
  SET employees.department_id=:NEW.department_id
  WHERE employees.department_id=:OLD.department_id;
 UPDATE job_history
  SET department_id=:NEW.department_id
  WHERE department_id=:OLD.department_id;
END;
/
```

Replicating a Table Within the Server

```
CREATE MATERIALIZED VIEW emp_copy
  NEXT sysdate + 7
  AS SELECT * FROM employees@ny;
```

Replicating a Table with a Trigger

```
CREATE OR REPLACE TRIGGER emp_replica
 BEFORE INSERT OR UPDATE ON employees FOR EACH ROW
BEGIN /* Proceed if user initiates data operation,
         NOT through the cascading trigger.*/
  IF INSERTING THEN
   IF :NEW.flag IS NULL THEN
     INSERT INTO employees@sf
     VALUES(:new.employee_id,...,'B');
     :NEW.flag := 'A';
   END IF;
  ELSE   /* Updating. */
   IF :NEW.flag = :OLD.flag THEN
     UPDATE employees@sf
      SET ename=:NEW.last_name,...,flag=:NEW.flag
      WHERE employee_id = :NEW.employee_id;
   END IF;
   IF :OLD.flag = 'A' THEN :NEW.flag := 'B';
                      ELSE :NEW.flag := 'A';
   END IF;
  END IF;
END;
```

Computing Derived Data Within the Server

```
UPDATE departments
 SET total_sal=(SELECT SUM(salary)
                FROM employees
                WHERE employees.department_id =
                     departments.department_id);
```

Computing Derived Values with a Trigger

```
CREATE PROCEDURE increment_salary
  (id NUMBER, new_sal NUMBER) IS
BEGIN
   UPDATE departments
   SET    total_sal = NVL (total_sal, 0)+ new_sal
   WHERE  department_id = id;
END increment_salary;
```

```
CREATE OR REPLACE TRIGGER compute_salary
AFTER INSERT OR UPDATE OF salary OR DELETE
ON employees FOR EACH ROW
BEGIN
 IF DELETING THEN     increment_salary(
     :OLD.department_id,(-1*:OLD.salary));
 ELSIF UPDATING THEN  increment_salary(
     :NEW.department_id,(:NEW.salary-:OLD.salary));
 ELSE   increment_salary(
     :NEW.department_id,:NEW.salary); --INSERT
 END IF;
END;
```

Logging Events with a Trigger

```
CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF quantity_on_hand, reorder_point
 ON inventories FOR EACH ROW
DECLARE
 dsc product_descriptions.product_description%TYPE;
 msg_text VARCHAR2(2000);
BEGIN
  IF :NEW.quantity_on_hand <=
     :NEW.reorder_point THEN
    SELECT product_description INTO dsc
    FROM product_descriptions
    WHERE product_id = :NEW.product_id;
  _ msg_text := 'ALERT: INVENTORY LOW ORDER:'||
       'Yours,' ||CHR(10) ||user || '.'|| CHR(10);
  ELSIF :OLD.quantity_on_hand >=
        :NEW.quantity_on_hand THEN
    msg_text := 'Product #'||... CHR(10);
  END IF;
  UTL_MAIL.SEND('inv@oracle.com','ord@oracle.com',
   message=>msg_text, subject=>'Inventory Notice');
END;
```

Summary

In this lesson, you should have learned how to:

- Use database triggers and database server functionality to:
  - Enhance database security
  - Audit data changes
  - Enforce data integrity
  - Maintain referential integrity
  - Replicate data between tables
  - Automate computation of derived data
  - Provide event-logging capabilities
- Recognize when to use triggers to database functionality