D

Applications of Database Triggers

Applications of Database Triggers

Develop database triggers within an application in order to enhance features that cannot otherwise be implemented by the Oracle Server.

Security	The Oracle Server allows table access to users or roles. Triggers allow table access according to data values.
Auditing	The Oracle Server tracks data operations on tables. Triggers track values for data operaions on tables.
Data Integrity	The Oracle Server enforces Integrity Constraints. Triggers implement complex integrity rules.
Referential Integrity	The Oracle Server enforces standard referential integrity rules. Triggers implement non–standard functionality.
Table Replication	The Oracle Server copies tables asynchronously into snapshots. Triggers copy tables synchronously into replicas.
Derived Data	The Oracle Server computes derived data values manually. Triggers compute derived data values automatically.
Event logging	The Oracle Server logs events explicitly. Triggers log events transparently.

Applications of Database Triggers

SECURITY

Example – Control security with a trigger

Restrict operations on the EMP table to certain times during the week.

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON emp
DECLARE
     v_dummy VARCHAR2(1);
BEGIN
     IF (TO_CHAR (sysdate,'DY') IN ('SAT','SUN'))
     OR (TO_NUMBER (sysdate,'HH24') NOT BETWEEN 8 AND
           17) THEN
       RAISE_APPLICATION_ERROR (-20506,
        'You may only change data during normal
          business hours.');
     END IF;
     SELECT COUNT (*)
       INTO v_dummy
       FROM holiday
       WHERE holiday_date <> TRUNC (sysdate);
     IF v_dummy > 0 THEN
       RAISE_APPLICATION_ERROR
       (-20507, 'You may not change data on a
                    holiday.');
     END IF;
END;
```

Example – Control security within the Server

Restrict operations on the EMP table to an individual authorized to access it.

```
SQL> GRANT CLERK TO JOE;
SQL> GRANT SELECT, INSERT, UPDATE, DELETE
  2  ON EMP
  3  TO CLERK;
```

Applications of Database Triggers

Develop triggers to handle more complex security requirements

Control Object Security with Triggers

- Base privileges upon any database values, such as the time of day, the day of the week, and so on.
- Determine access to tables only.
- Determine data manipulation privileges only.

Develop schemas and roles within ORACLE to control the security of data operations on tables according to the identity of the user.

Control Object Security within the Server

- Base privileges only upon the username supplied when the user connects to the database.
- Determine access to tables, views, synonyms, and sequences.
- Determine query, data manipulation, and data definition privileges.

Applications of Database Triggers

AUDITING

Example—Audit using a trigger

Track all values for successful data operations on the EMP table.

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
     IF audit_emp_package.g_reason IS NULL THEN
          RAISE_APPLICATION_ERROR (-20059, 'Specify a
          reason for the data operation with the
          procedure SET_REASON before proceeding.');
     ELSE
          INSERT INTO audit_emp_values (user_name,
          timestamp, empno,old_ename, new_ename,
          old_job, new_job, old_mgr, new_mgr, old_sal,
          new_sal, comments)
     VALUES (user, sysdate, :old.empno, :old.ename,
          :new.ename,:old.job, :new.job, :old.mgr,
          :new.mgr,:old.sal,:new.sal, audit_package.
          g_reason);
     END IF;
END;
```

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

Applications of Database Triggers

Example – Audit using the Server facility

Track all successful data operations on the EMP table.

```
SQL> AUDIT INSERT, UPDATE, DELETE
  2  ON emp
  3  BY ACCESS
  4  WHENEVER SUCCESSFUL;
```

Audit actual data values with triggers.

Audit Objects with Triggers

- Audit data manipulation statements only.
- Write the audit trail to a user-defined audit table.
- Generate audit records once for the statement or once for each row.
- Capture successful attempts only.
- Enable and disable dynamically.

Audit data operations within ORACLE.

Audit Objects within the Server

- Audit data retrieval, data manipulation, and data definition statements.
- Write the audit trail to the centralized audit table.
- Generate audit records once per session or once per access attempt.
- Capture successful attempts, unsuccessful attempts, or both.
- Enable and disable dynamically.

Applications of Database Triggers

DATA INTEGRITY

Example – Protect data integrity with a trigger

Ensure that the salary is never decreased nor increased more than 10% at a time.

```
CREATE OR REPLACE TRIGGER check_sal
BEFORE UPDATE OF sal ON emp
FOR EACH ROW
WHEN (new.sal < old.sal) OR (new.sal > old.sal * 1.1)
BEGIN
     RAISE_APPLICATION_ERROR (-20508,
          'Do not decrease salary nor increase by more
               than 10%.');
END;
```

Example – Enforce data integrity within the Server

Ensure that the salary is at least $500.

```
SQL> ALTER TABLE emp ADD
  2  CONSTRAINT ck_sal CHECK (sal >= 500);
```

Applications of Database Triggers

Develop triggers to handle more complex data integrity rules.

Enforce Data Integrity with Triggers

- Enforce non-standard data integrity checks.
- Provide variable default values.
- Enforce dynamic constraints.
- Enable and disable dynamically.

Incorporate declarative constraints within the definition of a table to protect data integrity.

Enforce Data Integrity within the Server

Enforce standard data integrity rules: not null, not unique, primary key, and foreign key.

- Provide constant default values.
- Enforce static constraints.
- Enable and disable dynamically.

REFERENTIAL INTEGRITY

Example – Enforce referential integrity with a trigger

When the value of DEPTNO changes in the DEPT parent table, cascade the update to the corresponding rows in the EMP child table. This example works only if there is no referential integrity between EMP and DEPT within the table definitions.

```
CREATE OR REPLACE TRIGGER cascade_updates
AFTER UPDATE OF deptno ON dept
FOR EACH ROW
BEGIN
     UPDATE emp
          SET emp.deptno = :new.deptno
          WHERE emp.deptno = :old.deptno;
END;
```

Example – Enforce referential integrity within the Server

When a department is removed from the DEPT parent table, cascade the delete to the corresponding rows in the EMP child table.

```
SQL> ALTER TABLE emp ADD
  2  CONSTRAINT fk_emp_deptno
  3  FOREIGN KEY (deptno) REFERENCES dept(deptno)
  4  ON DELETE CASCADE;
```

Applications of Database Triggers

You can develop triggers to implement non-standard referential integrity.

Enforce Referential Integrity with Triggers

- Cascade updates.
- Set to NULL on updates and deletes.
- Set to a default value on updates and deletes.
- Enforce referential integrity in a distributed system.
- Enable and disable dynamically.

Incorporate referential integrity constraints within the definition of a table to prevent data inconsistency.

Enforce Referential Integrity within the Server

- Restrict updates and deletes.
- Cascade deletes.
- Enable and disable dynamically.

Note: If there is referential integrity within the table definitions, this example violates Rule #1 for constraining tables.

TABLE REPLICATION

Example – Replicate a table with a trigger

In New York replicate the local EMP table to San Francisco.

```
CREATE OR REPLACE TRIGGER emp_replica
BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN  /*  Only proceed if the user initiated the data
operation, NOT the cascaded trigger.  */
     IF INSERTING THEN
          IF :new.flag IS NULL THEN
           INSERT INTO emp@sf
            VALUES (:new.empno, :new.ename,..., 'B');
           :new.flag = 'A';
          END IF;
     ELSE  /*  Updating.  */
          IF :new.flag = :old.flag THEN
           UPDATE emp@sf
            SET ename = :new.ename, ...
             flag = :new.flag
            WHERE empno = :new.empno;
          END IF;
          IF :old.flag = 'A' THEN
           :new.flag = 'B';
          ELSE
           :new.flag = 'A';
          END IF;
     END IF;
END;
```

Applications of Database Triggers

Example – Copy a table with a snapshot

In San Francisco, create a snapshot of the remote EMP table in New York.

```
CREATE SNAPSHOT emp_copy AS
SELECT * FROM emp@ny;
```

Note: Do not attempt to create AFTER row triggers on tables having snapshot logs against them, and vice versa, because the Oracle Server implements snapshots internally with AFTER row triggers of its own.

Alternatively, replicate tables using triggers.

Replicate Tables with Triggers

- Copy tables synchronously, in real time.
- Usually base replicas upon a single master table.
- Read from replicas, as well as write to them.
- Impair the performance of data manipulation on the master table, particularly if the network fails.

Maintain copies of tables automatically with snapshots, particularly on remote nodes.

Copy Tables with Server Snapshots

- Copy tables asynchronously, at user-defined intervals.
- Base snapshots upon multiple master tables.
- Read from snapshots only.
- Improve the performance of data manipulation on the master table, particularly if the network fails.

DERIVED DATA

Example – Compute derived values with a trigger

Keep a running total of the salary for each department within the special TOTAL_SAL column of the DEPT table.

```
CREATE OR REPLACE PROCEDURE increment_sal
     (v_deptno IN        dept.deptno%TYPE,
     v_sal     IN        dept.total_sal%TYPE)
IS
BEGIN
     UPDATE dept
          SET total_sal = NVL(total_sal,0) + v_sal
          WHERE deptno = v_deptno;
END increment_sal;
```

```
CREATE OR REPLACE TRIGGER compute_sal
AFTER INSERT OR UPDATE OF sal OR DELETE ON emp
FOR EACH ROW
BEGIN
     IF DELETING THEN
          increment_sal (:old.deptno, -1 * :old.sal);
     ELSIF UPDATING THEN
          increment_sal (:new.deptno, :new.sal -
               :old.sal);
     ELSE  /*  inserting  */
          increment_sal (:new.deptno, :new.sal);
     END IF;
END;
```

Example – Compute derived values in a batch job

Keep the salary total for each department within the special TOTAL_SAL column of the DEPT table.

```
SQL> UPDATE dept
  2  SET total_sal =
  3  (SELECT SUM(sal)
  4  FROM emp
  5  WHERE emp.deptno = dept.deptno);
```

With triggers, keep running computations of derived data.

Derive Data with Triggers

- Compute derived columns synchronously, in real time.
- Store derived values within database tables or within package global variables.
- Modify data and calculate derived data in a single pass to the database.

Within the Server, compute derived data manually, for example, in a batch job.

Derive Data within the Server

- Compute derived columns asynchronously, at user defined intervals.
- Store derived values within database tables only.
- Modify data in one pass to the database and calculate derived data in a second pass.

Example – Log events transparently

```
CREATE OR REPLACE TRIGGER notify_reorder_rep
AFTER UPDATE OF parts_on_hand, reorder_point ON inventory
FOR EACH ROW
WHEN new.parts_on_hand <= new.reorder_point
DECLARE
            v_reorder_rep_name      emp.ename%TYPE;
            v_clerk_name            emp.ename%TYPE;
            v_manager_name          emp.ename%TYPE;
            v_msg_subject           VARCHAR2;
            v_part_name             part.part_name%TYPE;
            v_msg_text              VARCHAR2;
BEGIN
            SELECT ename
                  INTO v_reorder_rep_name FROM emp, part
                  WHERE part.part_no = :new.part_no
                  AND emp.empno = part.reorder_rep_no;
                  v_clerk_name := user;
            SELECT manager.ename INTO v_manager_name
                  FROM emp worker, emp manager
                  WHERE worker.mgr = manager.empno
                  AND worker.ename = v_clerk_name;
```

```
v_msg_subject := 'LOW INVENTORY FOR PART #'||
      TO_CHAR(:new.part_no);
SELECT part_name INTO v_part_name FROM part
      WHERE part_no = :new.part_no;
v_msg_text :=
      CHR(10) || 'Dear ' || v_reorder_rep_name || ':' ||
      CHR(10) || CHR(10) ||
      'It has come to my personal attention that, due to recent '
      CHR(10) || 'transactions, our inventory for part #' ||
      TO_CHAR(:new.part_no) || '--' || v_part_name ||
      '--has fallen' || CHR(10) || CHR(10) || 'Yours,' || CHR(10)
                  || CHR(10) || v_clerk_name || CHR(10) ||
CHR(10) ||
      'PS  I am sending a copy of this notice to my manager, ' ||
      v_manager_name || '.' ;
dbms_mail.send (v_clerk_name, v_order_rep_name, v_manager_name,
      NULL, v_msg_subject, NULL, v_msg_text);
END;
```

Note:

- In the trigger code:
  - *CHR(10)* is a carriage return
  - *Parts_on_hand* is not null
  - *Reorder_point* is not null

Soon, Turner receives an e–mail.

```
Received:   1-14-93 14:11                          Sent:  1-14-93
14:09
From:       JOE
To:         TURNER
Subject:    LOW INVENTORY FOR PART #129
Cc:         BLAKE

Dear TURNER:

It has come to my personal attention that, due to recent
transactions, our inventory for part #129--WIDGET--has fallen
to critical levels.

Yours,

JOE

PS  I am sending a copy of this notice to my manager, BLAKE.
```

EVENT LOGGING

Example – Log events in a batch job.

Display all parts that need to be reordered because the parts on hand have fallen below the reorder point.

```
SQL> SELECT part.part_no, part.part_name,
     inventory.parts_on_hand,
  2  inventory.reorder_point, emp.ename
     reorder_rep_name
  3  FROM part, inventory, emp
  4  WHERE part.part_no = inventory.part_no
  5  AND inventory.parts_on_hand <=
     inventory.reorder_point
  6  AND part.reorder_rep_no = emp.empno;
```

PART_NO	PART_NAME	PARTS_ON_HAND	REORDER_POINT	REORDER_REP_NAME
129	WIDGET	72	100	TURNER

Applications of Database Triggers

Within the Server, log events by querying data and performing operations manually, in order to send electronic mail, for instance.

Log Events with Triggers

- Implicitly perform the operation transparent to the user, such as firing off an automatic electronic memo.
- Modify data and perform its dependent operation in a single step.

With triggers, log events automatically as data is changing.

Log Events within the Server

- Explicitly query data to determine if an operation is necessary.
- In a second step, perform the operation, such as sending electronic mail.