Administrarea Bazelor de Date
Managementul în Tehnologia Informaţiei

Sisteme Informatice şi Standarde Deschise (SISD)

2009-2010

Curs 8

Standarde pentru programarea bazelor de date (2)

Types of Triggers

- A trigger:
  - Is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or database
  - Executes implicitly whenever a particular event takes place
  - Can be either of the following:
    - Application trigger: Fires whenever an event occurs with a particular application
    - Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database

Guidelines for Designing Triggers

- You can design triggers to:
  - Perform related actions
  - Centralize global operations
- You must not design triggers:
  - Where functionality is already built into the Oracle server
  - That duplicate other triggers
- You can create stored procedures and invoke them in a trigger, if the PL/SQL code is very lengthy.
- The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.

Creating DML Triggers

DML = data manipulation language

- Create DML statement or row type triggers by using:

```
CREATE [OR REPLACE] TRIGGER trigger_name
 timing
 event1 [OR event2 OR event3]
ON object_name
[[REFERENCING OLD AS old | NEW AS new]
 FOR EACH ROW
 [WHEN (condition)]]
trigger_body
```

  - A statement trigger fires once for a DML statement.
  - A row trigger fires once for each row affected.
- Note: Trigger names must be unique with respect to other triggers in the same schema.

Types of DML Triggers

- The trigger type determines if the body executes for each row or only once for the triggering statement.
  - A statement trigger:
    - Executes once for the triggering event
    - Is the default type of trigger
    - Fires once even if no rows are affected at all
  - A row trigger:
    - Executes once for each row affected by the triggering event
    - Is not executed if the triggering event does not affect any rows
    - Is indicated by specifying the `FOR EACH ROW` clause

Trigger Timing

- When should the trigger fire?
  - `BEFORE`: Execute the trigger body before the triggering DML event on a table.
  - `AFTER`: Execute the trigger body after the triggering DML event on a table.
  - `INSTEAD OF`: Execute the trigger body instead of the triggering statement. This is used for views that are not otherwise modifiable.
- Note: If multiple triggers are defined for the same object, then the order of firing triggers is arbitrary.

Trigger-Firing Sequence

- Use the following firing sequence for a trigger on a table when a single row is manipulated:

DML statement

```
INSERT INTO departments
   (department_id,department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```

Triggering action

→ BEFORE statement trigger

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
...		
400	CONSULTING	2400

→ BEFORE row trigger

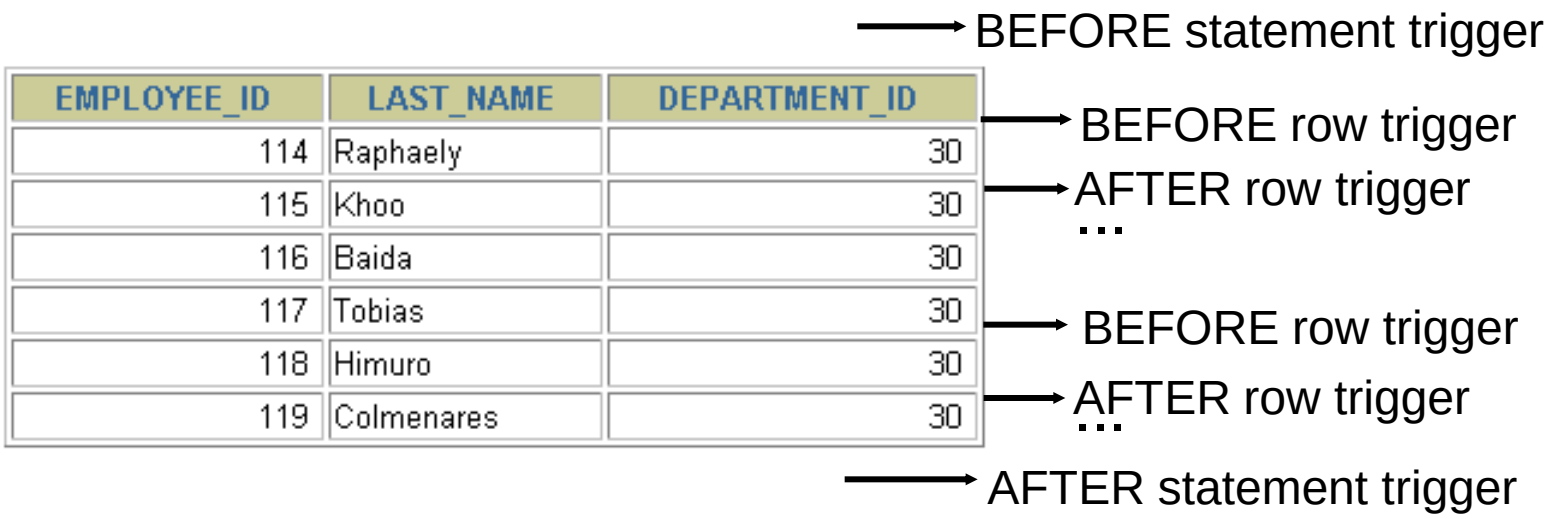
→ AFTER row trigger

→ AFTER statement trigger

Trigger-Firing Sequence

Use the following firing sequence for a trigger on a table when many rows are manipulated:

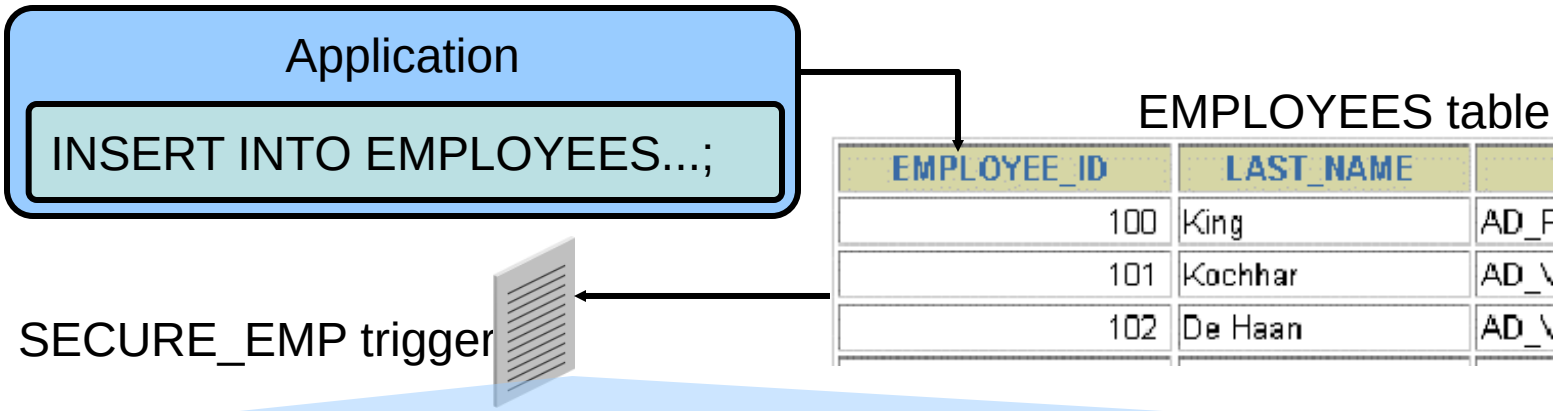
```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```



Trigger Event Types and Body

- A trigger event:
  - Determines which DML statement causes the trigger to execute
  - Types are:
    - `INSERT`
    - `UPDATE [OF column]`
    - `DELETE`
- A trigger body:
  - Determines what action is performed
  - Is a PL/SQL block or a `CALL` to a procedure

Creating a DML Statement Trigger




```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
     (TO_CHAR(SYSDATE,'HH24:MI')
      NOT BETWEEN '08:00' AND '18:00') THEN
  RAISE_APPLICATION_ERROR(-20500, 'You may insert'
    ||' into EMPLOYEES table only during '
    ||' business hours.');
  END IF;
END;
```

Testing SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
first_name, email, hire_date,
job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
'IT_PROG', 4500, 60);
```

```
INSERT INTO employees (employee_id, last_name, first_name, email,
       *
ERROR at line 1:
ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "PLSQL.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'
```

Using Conditional Predicates

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
   (TO_CHAR(SYSDATE,'HH24')
      NOT BETWEEN '08' AND '18') THEN
   IF DELETING THEN RAISE_APPLICATION_ERROR(
    -20502,'You may delete from EMPLOYEES table'||
           'only during business hours.');
   ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
    -20500,'You may insert into EMPLOYEES table'||
           'only during business hours.');
   ELSIF UPDATING('SALARY') THEN
    RAISE_APPLICATION_ERROR(-20503, 'You may '||
     'update SALARY only during business hours.');
   ELSE RAISE_APPLICATION_ERROR(-20504,'You may'||
     ' update EMPLOYEES table only during'||
     ' normal hours.');
   END IF;
 END IF;
END;
```

Creating a DML Row Trigger

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
     AND :NEW.salary > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;
/
```

Using OLD and NEW Qualifiers

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, id,
    old_last_name, new_last_name, old_title,
    new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
    :OLD.last_name, :NEW.last_name, :OLD.job_id,
    :NEW.job_id, :OLD.salary, :NEW.salary);
END;
/
```

Using OLD and NEW Qualifiers: Example Using audit_emp

```
INSERT INTO employees
 (employee_id, last_name, job_id, salary, ...)
VALUES (999, 'Temp emp', 'SA_REP', 6000,...);

UPDATE employees
 SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999;
```

```
SELECT user_name, timestamp, ...
FROM audit_emp;
```

USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA_REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

Restricting a Row Trigger: Example

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.commission_pct := 0;
 ELSIF :OLD.commission_pct IS NULL THEN
   :NEW.commission_pct := 0;
 ELSE
   :NEW.commission_pct := :OLD.commission_pct+0.05;
 END IF;
END;
/
```

Summary of Trigger Execution Model

1. Execute all `BEFORE STATEMENT` triggers.
2. Loop for each row affected:
   a. Execute all `BEFORE ROW` triggers.
   b. Execute the DML statement and perform integrity constraint checking.
   c. Execute all `AFTER ROW` triggers.
3. Execute all `AFTER STATEMENT` triggers.

- Note: Integrity checking can be deferred until the `COMMIT` operation is performed.

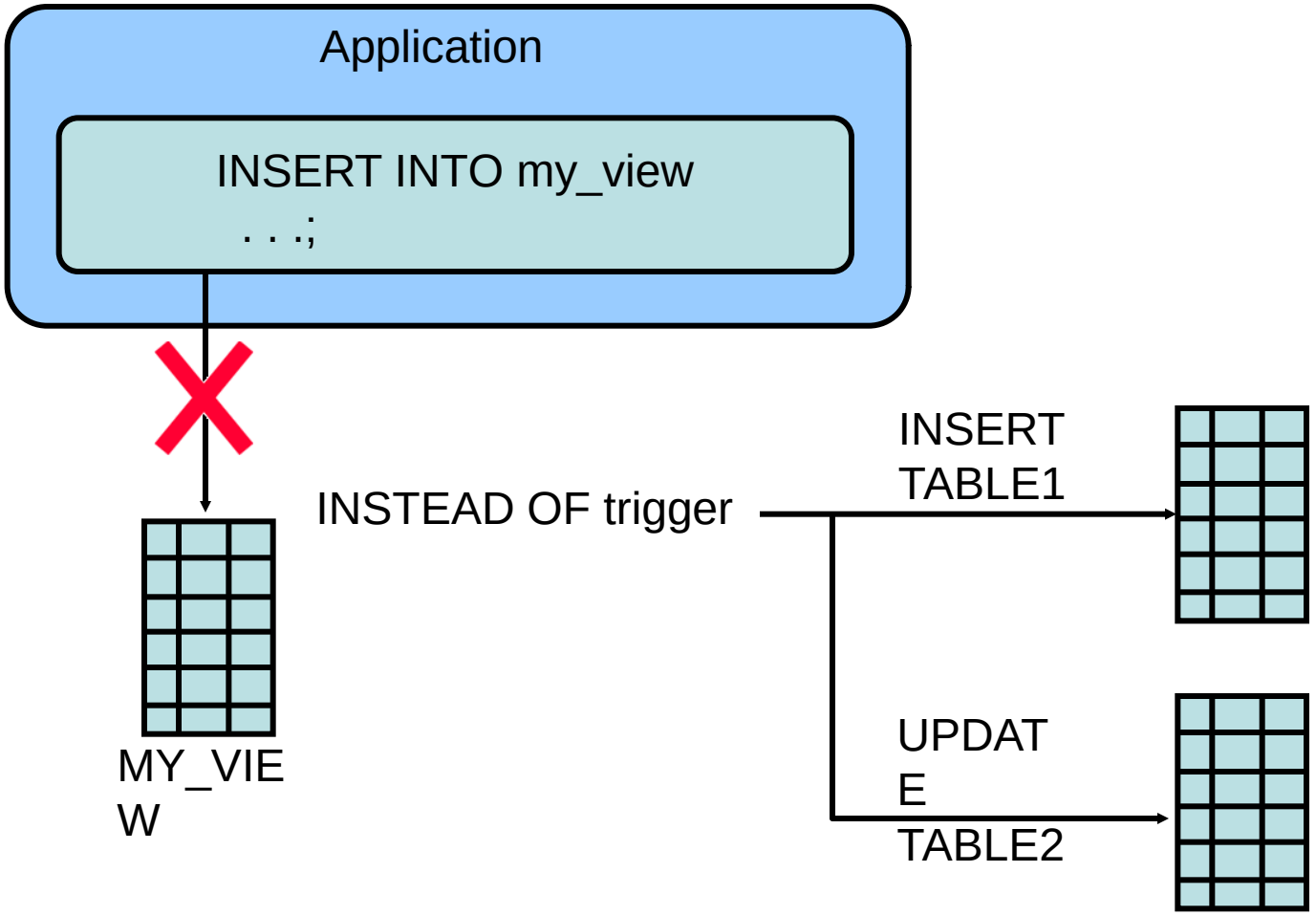
Implementing an Integrity Constraint with a Trigger

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Integrity constraint violation error
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id
ON employees FOR EACH ROW
BEGIN
 INSERT INTO departments VALUES(:new.department_id,
           'Dept '||:new.department_id, NULL, NULL);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
    NULL; -- mask exception if department exists
END;
/
```

```
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
-- Successful after trigger is fired
```

INSTEAD OF Triggers



Creating an INSTEAD OF Trigger

- Perform the INSERT into EMP_DETAILS that is based on EMPLOYEES and DEPARTMENTS tables:

```
INSERT INTO emp_details
VALUES (9001,'ABBOTT',3000, 10, 'Administration');
```

1 INSTEAD OF INSERT into EMP_DETAILS

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90

2 INSERT into NEW_EMPS

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
100	King	24000	90
101	Kochhar	17000	90
102	De Haan	17000	90
...			
9001	ABBOTT	3000	10

3 UPDATE NEW_DEPTS

DEPARTMENT_ID	DEPARTMENT_NAME	DEPT_SA
10	Administration	940
20	Marketing	1900
30	Purchasing	3012
40	Human Resources	650
...		

Creating an INSTEAD OF Trigger

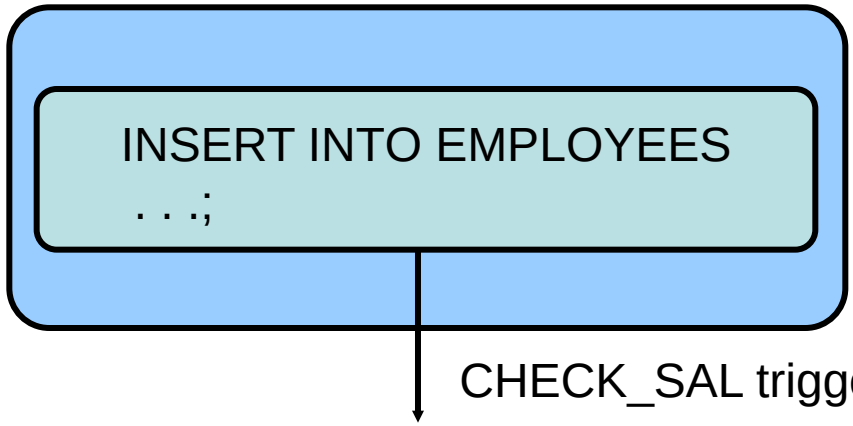
- Use INSTEAD OF to perform DML on complex views:

```
CREATE TABLE new_emps AS
 SELECT employee_id,last_name,salary,department_id
 FROM employees;

CREATE TABLE new_depts AS
 SELECT d.department_id,d.department_name,
        sum(e.salary) dept_sal
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
 SELECT e.employee_id, e.last_name, e.salary,
        e.department_id, d.department_name
 FROM employees e, departments d
 WHERE e.department_id = d.department_id
GROUP BY d.department_id,d.department_name;
```

Comparison of Database Triggers and Stored Procedures

Triggers	Procedures
Defined with CREATE TRIGGER Data dictionary contains source code in USER_TRIGGERS. Implicitly invoked by DML COMMIT, SAVEPOINT, and ROLLBACK are not allowed.	Defined with CREATE PROCEDURE Data dictionary contains source code in USER_SOURCE. Explicitly invoked COMMIT, SAVEPOINT, and ROLLBACK are allowed.

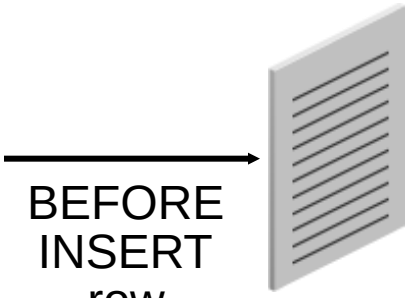
Comparison of Database Triggers and Oracle Forms Triggers



EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000

. . .



Managing Triggers

- Disable or reenable a database trigger:

```
ALTER TRIGGER trigger_name DISABLE | ENABLE
```

- Disable or reenable all triggers for a table:

```
ALTER TABLE table_name DISABLE | ENABLE
  ALL TRIGGERS
```

- Recompile a trigger for a table:

```
ALTER TRIGGER trigger_name COMPILE
```

Removing Triggers

- To remove a trigger from the database, use the `DROP TRIGGER` statement:

```
DROP TRIGGER trigger_name;
```

- Example:

```
DROP TRIGGER secure_emp;
```

- Note: All triggers on a table are removed when the table is removed.

Creating Triggers on DDL Statements

- Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

Creating Triggers on System Events

- Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

LOGON and LOGOFF Triggers: Example

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON  ON  SCHEMA
BEGIN
 INSERT INTO log_trig_table(user_id,log_date,action)
 VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF  ON  SCHEMA
BEGIN
 INSERT INTO log_trig_table(user_id,log_date,action)
 VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

CALL Statements

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON table_name
[REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
[WHEN condition]
CALL procedure_name
/
```

```
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
CALL log_execution
/
```

- statement.

Reading Data from a Mutating Table

```
UPDATE employees
  SET salary = 3400
  WHERE last_name = 'Stiles';
```

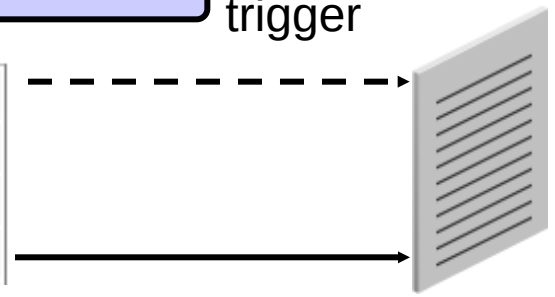
EMPLOYEES table

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
125	Nayer	ST_CLERK	3200
126	Mikkilineni	ST_CLERK	2700
127	Landry	ST_CLERK	2400
...			
138	Stiles	ST_CLERK	3400
...			

Triggered table/
mutating table

Failure

CHECK_SALARY
trigger



BEFORE UPDATE row

Trigger event

Mutating Table: Example

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  minsalary employees.salary%TYPE;
  maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
   INTO  minsalary, maxsalary
   FROM  employees
   WHERE job_id = :NEW.job_id;
  IF :NEW.salary < minsalary OR
     :NEW.salary > maxsalary THEN
     RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
/
```

Mutating Table: Example

```
UPDATE employees
 SET salary = 3400
 WHERE last_name = 'Stiles';
```

```
UPDATE employees
       *
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "PLSQL.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'PLSQL.CHECK_SALARY'
```

Benefits of Database Triggers

- Improved data security:
  - Provide enhanced and complex security checks
  - Provide enhanced and complex auditing
- Improved data integrity:
  - Enforce dynamic data integrity constraints
  - Enforce complex referential integrity constraints
  - Ensure that related operations are performed together implicitly

Managing Triggers

- The following system privileges are required to manage triggers:
  - `CREATE/ALTER/DROP (ANY) TRIGGER` privilege: enables you to create a trigger in any schema
  - `ADMINISTER DATABASE TRIGGER` privilege: enables you to create a trigger on `DATABASE`
  - `EXECUTE` privilege (if your trigger refers to any objects that are not in your schema)
- Note: Statements in the trigger body use the privileges of the trigger owner, not the privileges of the user executing the operation that fires the trigger.

Business Application Scenarios for Implementing Triggers

- You can use triggers for:
  - Security
  - Auditing
  - Data integrity
  - Referential integrity
  - Table replication
  - Computing derived data automatically
  - Event logging
- Note: Appendix C covers each of these examples in more detail.

Viewing Trigger Information

- You can view the following trigger information:
  - `USER_OBJECTS` data dictionary view: object information
  - `USER_TRIGGERS` data dictionary view: text of the trigger
  - `USER_ERRORS` data dictionary view: PL/SQL syntax errors (compilation errors) of the trigger

Using USER_TRIGGERS

Column	Column Description
TRIGGER_NAME	Name of the trigger
TRIGGER_TYPE	The type is BEFORE, AFTER, INSTEAD OF
TRIGGERING_EVENT	The DML operation firing the trigger
TABLE_NAME	Name of the database table
REFERENCING_NAMES	Name used for :OLD and :NEW
WHEN_CLAUSE	The when_clause used
STATUS	The status of the trigger
TRIGGER_BODY	The action to take

* Abridged column list

Listing the Code of Triggers

```
SELECT trigger_name, trigger_type, triggering_event,
         table_name, referencing_names,
         status, trigger_body
FROM   user_triggers
WHERE  trigger_name = 'RESTRICT_SALARY';
```

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_NAME	REFERENCING_NAMES	WHEN_CLAUS	STATUS	TRIGGER_BODY
RESTRICT_SALARY	BEFORE EACH ROW	INSERT OR UPDATE	EMPLOYEES	REFERENCING NEW AS NEW OLD AS OLD		ENABLED	BEGIN IF NOT (:NEW.JOB_ ID IN ('AD_PRES ', 'AD_VP')) AND :NE W.SAL

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;
```

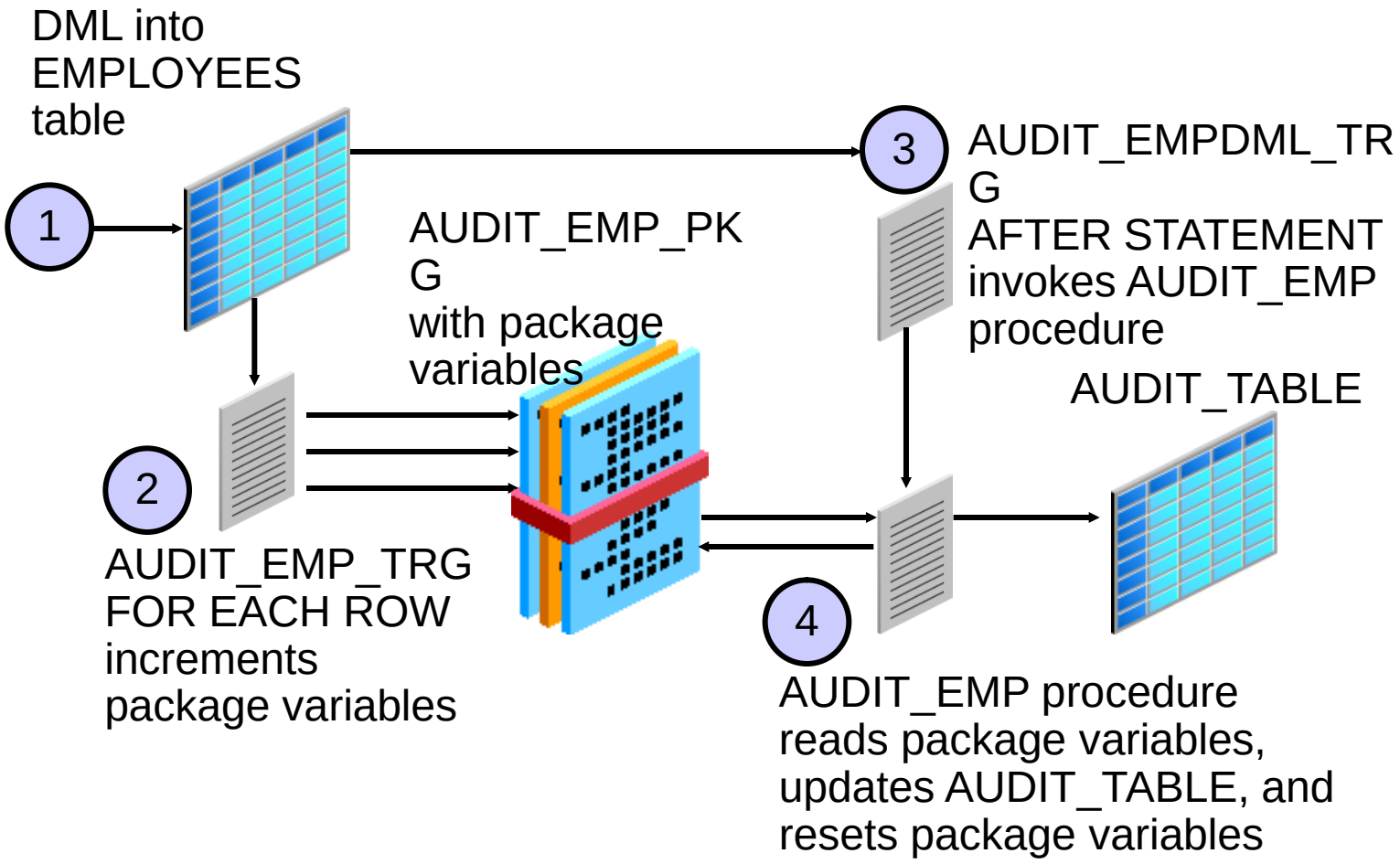
Audit succeeded.

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

- The `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;
/
```

- The `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 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;
/
```

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;
```

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;
```

Exam's quizzes

- **1.** Care sunt principalele scopuri ale utilizării trigger-elor?
- **2.** Descrieţi pe scurt modul de execuţie al trigger-elor.
- **3.** Propuneţi o soluţie pentru implementarea trigger-elor.