CSCI235 Database Systems

Database Triggers

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Database Triggers

Outline

Database trigger ? What is it ?

Database trigger is a piece of code stored in a data dictionary and automatically processed whenever a pre-defined event happens and pre-defined condition is satisfied

For example, we would like to automatically increase job level for all employees whose salary is above 100000

Database trigger

```
ON UPDATE OF EMPLOYEE.salary
  IF :NEW.salary > 100000 THEN
    IncreaseJobLevel(:NEW.enumber, :NEW.salary);
  END IF;
```

For example, we would like to implement a data security rule saing that a salary cannot be updated over a weekend

Database trigger

```
ON UPDATE OF EMPLOYEE.salary
  IF TO_CHAR(SYSDATE,'Day') IN ('Saturday', 'Sunday') THEN
    RAISE_APPLICATION_ERROR(-20001, 'Salary cannot be updated over a weekend !');
  END IF;
```

Database trigger ? What is it ?

For example, we would like to enforce a consistency constraint saying that a department cannot have more than 100 employees

Database trigger
```
ON INSERT INTO EMPLOYEE
  SELECT COUNT(*)
  INTO total_employees
  FROM EMPLOYEE
  WHERE dname = :NEW.dname;
  IF total_employees = 100 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Too many employees in ' || :NEW.dname);
  END IF;
```

In the example above we assume that a trigger fires and it is processed before `INSERT` statement

Sometimes it is more convenient to fire a trigger that verifies a consistency constraint after modification of a relational table and before `COMMIT` statement

This is why we have two temporal options for triggers: `BEFORE` and `AFTER`

Database trigger ? What is it ?

What do we need database triggers for ?

- To verify the consistency constraints
- To enforce the sophisticated database access controls
- To implement transparent event logging
- To generate the values of derived attributes
- To maintain replicated data in a distributed database
- To update the relational views

Active Database Systems provide functionalities for implementation of database triggers

Database Triggers

Outline

Active database system

Active database system is a system which is able to detect the events that have happened in a certain period of time and in the response to these events it is able to execute the actions when the pre-defined conditions are met

A logic of active database system is implemented as a collection of Event-Condition-Action (ECA)rules

In SQL ECA rule can be created with `CREATE TRIGGER` statement and it can be deleted with `DROP TRIGGER` statement

Syntax of ECA rule:

- (`EVENT`, `CONDITION`, `ACTION`)

Semantics of ECA rule:

- Whenever an `EVENT` happens and a `CONDITION` is satisfied then a database system performs an `ACTION`

Active database system

A sample event

```
ON UPDATE OF EMPLOYEE.salary                                        Trigger
```

A sample condition

```
IF :NEW.salary > 100000                                             Trigger
```

A sample action

```
IncreaseJobLevel(:NEW.enumber, :NEW.salary);                        Trigger
```

`CREATE OR REPLACE TRIGGER` statement implements ECA rule

Database Triggers

Outline

CREATE OR REPLACE TRIGGER statement

A sample CREATE OR REPLACE TRIGGER statement

```
CREATE OR REPLACE TRIGGER CheckBudget                                   Trigger name
```

Temporal option

```
                                                       Temporal option specification
BEFORE
```

Event

```
                                                                 Event specification
UPDATE OF budget ON DEPARTMENT
```

Type of trigger, either statement or row trigger

```
                                                                         Row trigger
FOR EACH ROW                -- FOR EACH ROW means that it is a row trigger
```

Condition

```
                                                                   Trigger condition
WHEN NEW.name = 'Math'      -- NEW is a so called pseudorecord
```

CREATE OR REPLACE TRIGGER statement

Beginning of trigger's body

```
                                                        Start of trigger's body
BEGIN
```

Pseudorecords :OLD and :NEW that represents a row before modification or deletion and a row after modification or insertion

```
                              Application of correlation variables in a row trigger
IF NOT ( :NEW.budget BETWEEN 1 AND 7000 ) THEN
```

Abnormal termination of a trigger together with a transaction that fired a trigger

```
                                                   Abnormal termination of a trigger
RAISE_APPLICATION_ERROR(-200001, 'Budget of department ' || :NEW.name ||
                                 ' cannot be equal to ' || :NEW.budget );
```

End of trigger's body

```
                                                          End of trigger's body
  END IF;
END;
```

CREATE OR REPLACE TRIGGER statement

A complete CREATE OR REPLACE TRIGGER statement

A sample row trigger

```
CREATE OR REPLACE TRIGGER CheckBudget
BEFORE UPDATE OF budget ON DEPARTMENT
  FOR EACH ROW
  WHEN NEW.name = 'Math'
  BEGIN
    IF NOT ( :NEW.budget BETWEEN 1 AND 7000 ) THEN
      RAISE_APPLICATION_ERROR(-200001, 'Budget of department ' || :NEW.name ||
                                       ' cannot be equal to ' || :NEW.budget );
    END IF;
  END;
```

CREATE OR REPLACE TRIGGER statement

The following temporal options are available

- BEFORE - a trigger fires before a triggering event
- AFTER - a trigger fires after a triggering event
- INSTEAD OF - a trigger fires instead of a triggering event, it is typically used to correctly implement view update operation i.e. a correct modification of base relational tables through an update peformed on a relational view

Sample applications of temporal options

Fire a trigger before UPDATE operation on a column budget in a relational table DEPARTMENT

A sample temporal option

```
BEFORE UPDATE OF budget ON DEPARTMENT
```

Fire a trigger after any DELETE or UPDATE operation performed on DEPARTMENT table

A sample temporal option

```
AFTER DELETE OR UPDATE ON DEPARTMENT
```

CREATE OR REPLACE TRIGGER statement

Fire a trigger instead of UPDATE operation on a relational view EMPVIEW

A sample temporal option

```
INSTEAD OF INSERT ON EMPVIEW
```

CREATE OR REPLACE TRIGGER statement

The following events can fire a trigger

- Data Manipulation event - any `INSERT` or `UPDATE` or `DELETE` statement
- Data Definition event - any `CREATE` or `ALTER` or `DROP` statement
- Database events - the events such as a database server error, startup/shutdown of a database server, logon/logoff of a user, etc

Sample applications of DML events

A sample DML event
```
BEFORE UPDATE OF attribute, attribute,... ON table
```

A sample DML event
```
AFTER INSERT ON table
```

A sample DML event
```
BEFORE DELETE ON table
```

A sample DML event
```
AFTER DELETE OR INSERT OR UPDATE ON table
```

CREATE OR REPLACE TRIGGER statement

Sample applications of DDL events

A sample DDL event
```
AFTER ALTER database object
```

A sample DDL event
```
BEFORE CREATE database object
```

A sample DDL event
```
AFTER DROP database object
```

A sample DDL event
```
AFTER GRANT database object
```

A sample DDL event
```
BEFORE ANALYZE database object
```

A sample DDL event
```
AFTER GRANT system privilege
```

CREATE OR REPLACE TRIGGER statement

Sample applications of Database events

A sample database event
```
AFTER SERVERERROR ON SCHEMA
```

A sample database event
```
BEFORE LOGON
```

A sample database event
```
BEFORE LOGOFF
```

A sample database event
```
AFTER STARTUP
```

A sample database event
```
BEFORE SHUTDOWN
```

CREATE OR REPLACE TRIGGER statement

Condition determines whether a trigger processes its body after it has been fired

Sample applications of condition

```
WHEN (condition)
```
A sample condition

```
WHEN (OLD.status = 'BUSY' AND NEW.status = 'AVAILABLE');
```
A sample condition

```
WHEN (NEW.amount > 1000 );
```
A sample condition

```
WHEN (OLD.credits IN (6, 12));
```
A sample condition

OLD and NEW are so called pseudorecords such that for

- INSERT triggering operation OLD contains no values and NEW contains the new values
- UPDATE triggering operation OLD contains the old values and NEW contains the new values
- DELETE triggering operation OLD contains the old values and NEW contains no values

Database Triggers

Outline

Statement database triggers

A statement trigger fires once either before or after a triggering event

A sample statement trigger

```
                                                                 Trigger name
CREATE OR REPLACE TRIGGER ModifyDepartment
                                           Temporal option and event specification
  AFTER DELETE OR UPDATE ON DEPARTMENT
                                                Start of statement trigger's body
  BEGIN      -- Statement triggers have no FOR EACH ROW clause!
                                                             Trigger condition
   IF DELETING THEN
      INSERT INTO DEPTAUDIT VALUES('DELETE', SYSDATE);          Trigger's body
   ELSIF UPDATING THEN                                          Trigger's body
      INSERT INTO DEPTAUDIT VALUES('UPDATE', SYSDATE);          Trigger's body
                                                          End of trigger's body
   END IF;
  END;                                                    End of trigger's body
```

Statement database trigger

Assume that the following `UPDATE` statement has been processed and not `COMMIT`ed yet

UPDATE statement
```
UPDATE DEPARTMENT
SET budget = budget + 1000
WHERE budget < 5000;
```

Feedback message
```
3 row updated
```

The following body of a trigger `ModifyDepartment` has been processed immediately after processing of `UPDATE` statement

A body of statement trigger
```
BEGIN
  IF DELETING THEN
    INSERT INTO DEPTAUDIT VALUES('DELETE', SYSDATE);
  ELSIF UPDATING THEN
    INSERT INTO DEPTAUDIT VALUES('UPDATE', SYSDATE);
  END IF;
END;
```

Database Triggers

Outline

Row database triggers

A row trigger fires either after or before a triggering event affects a row in a relational table

- When a temporal option `BEFORE` is used a trigger fires once before a triggering event affects a row in a relational table
- When a temporal option `AFTER` is used a trigger fires once after a triggering event affects a row in a relational table

For example, if a temporal option and event are

```
BEFORE INSERT ON DEPARTMENT                                    A temporal option and event
```

then a trigger fires before each insertion into a relational table (it is possible to have many insertions when a multirow `INSERT` statement is processed)

For example, if a temporal option and event are

```
AFTER UPDATE ON EMPLOYEE                                       A temporal option and event
```

then a trigger fires after a row is updated in a relational table, if a triggering event updates n rows then a trigger fires n times

Row database triggers

For example, if a temporal option and event are

```
A temporal option and event

AFTER DELETE ON PROJECT
```

Then a trigger fires after a row is deleted from a relational table, if a triggering event deletes n rows then a trigger fires n times

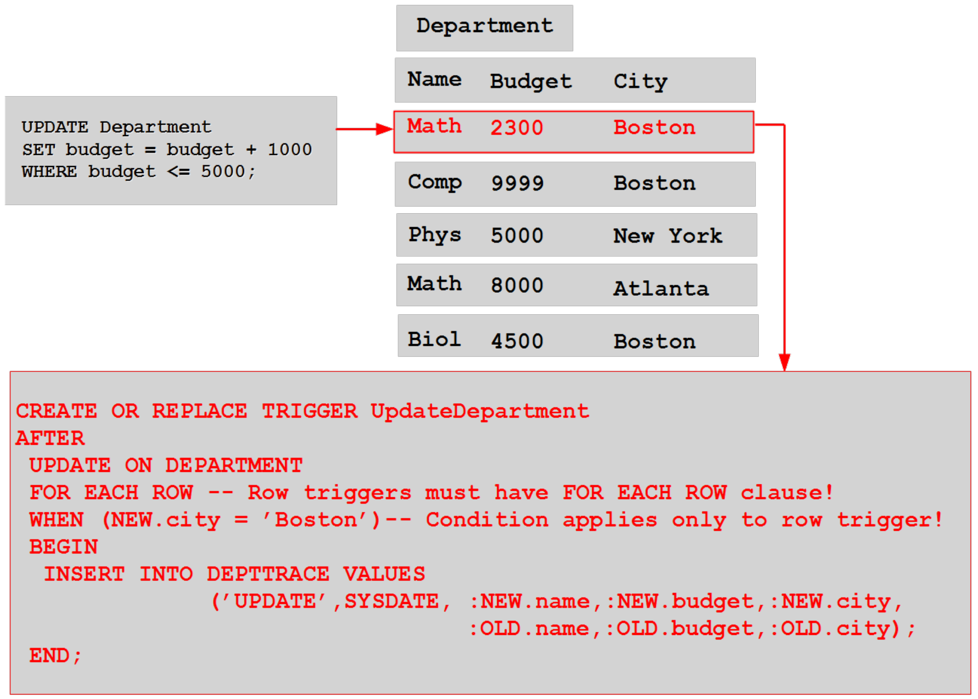
Row database triggers

A sample row trigger

```
CREATE OR REPLACE TRIGGER UpdateDepartment                    Trigger name
  AFTER UPDATE ON DEPARTMENT                                  Temporal option and event
  FOR EACH ROW             -- Row trigger must have FOR EACH ROW clause !     Row trigger
  WHEN (NEW.city = 'Boston')-- Only for row triggers!         Trigger condition
  BEGIN                                                       Start of trigger's body
    INSERT INTO DEPTTRACE VALUES                              Trigger's body
           ('UPDATE', SYSDATE, :NEW.name, :NEW.budget, :NEW.city,   Trigger's body
                               :OLD.name, :OLD.budget, :OLD.city );  Trigger's body
  END;                                                        End of trigger's body
```

Row database triggers

A sample processing of a row database trigger



```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	5000	New York
Math	8000	Atlanta
Biol	4500	Boston

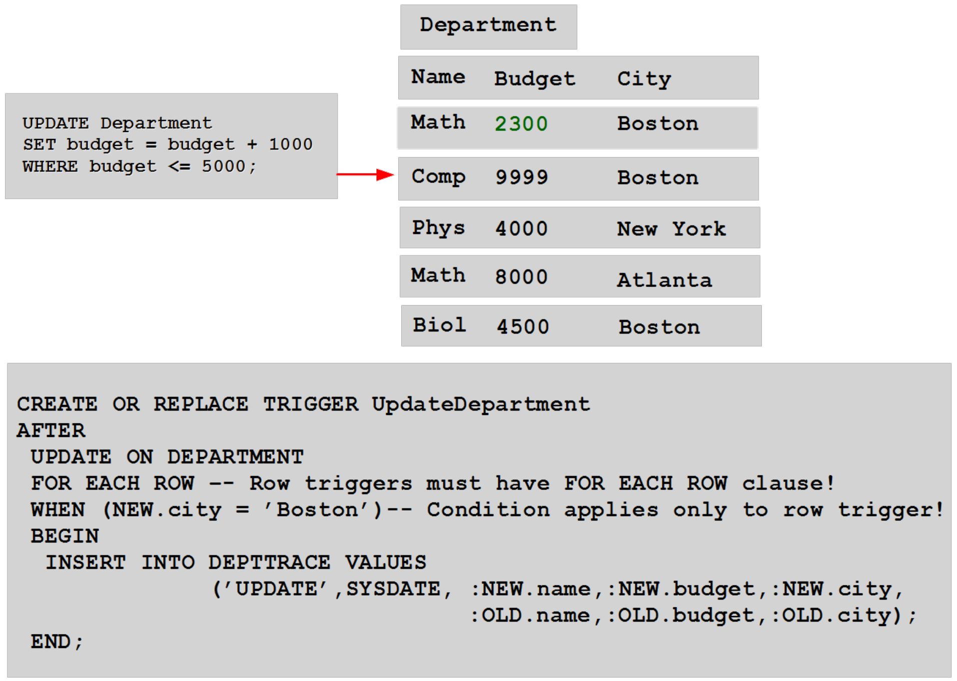
```
CREATE OR REPLACE TRIGGER UpdateDepartment
AFTER
 UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
              ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                 :OLD.name,:OLD.budget,:OLD.city);
 END;
```

A trigger fires after `UPDATE` of a row `[Math 2300 Boston]`

`WHEN` condition is statisfied and a trigger processes its body

Row database triggers

A sample processing of a row database trigger



```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	4000	New York
Math	8000	Atlanta
Biol	4500	Boston

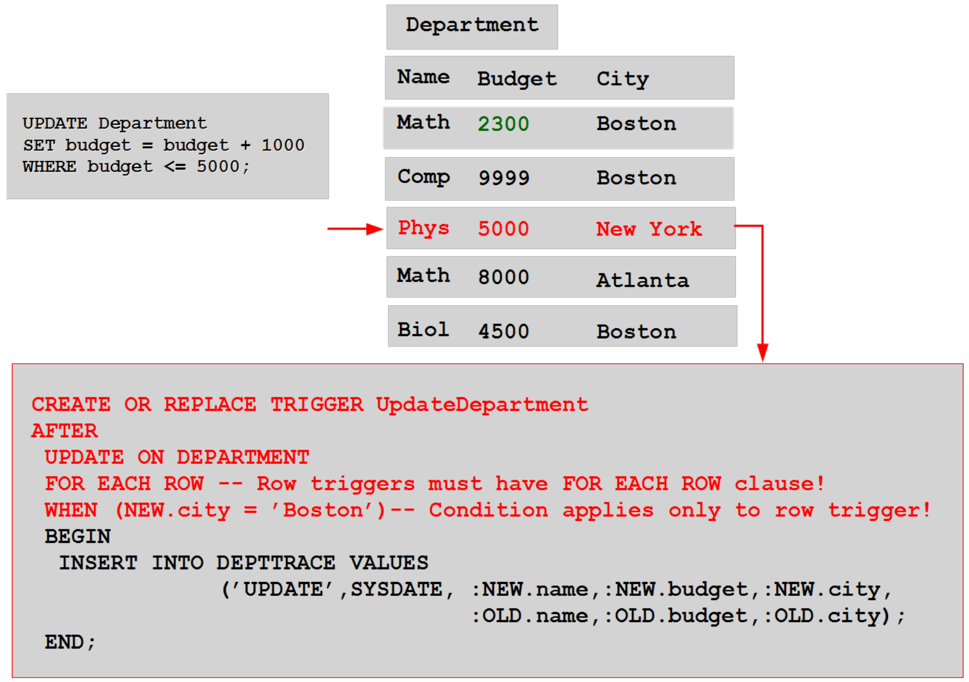
```
CREATE OR REPLACE TRIGGER UpdateDepartment
AFTER
 UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
              ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                 :OLD.name,:OLD.budget,:OLD.city);
 END;
```

A row `[Comp 9999 Boston]` does not satisfy a condition in `WHERE` clause and it is not `UPDATE`ed

A trigger does not fire

Row database triggers

A sample processing of a row database trigger



```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	5000	New York
Math	8000	Atlanta
Biol	4500	Boston

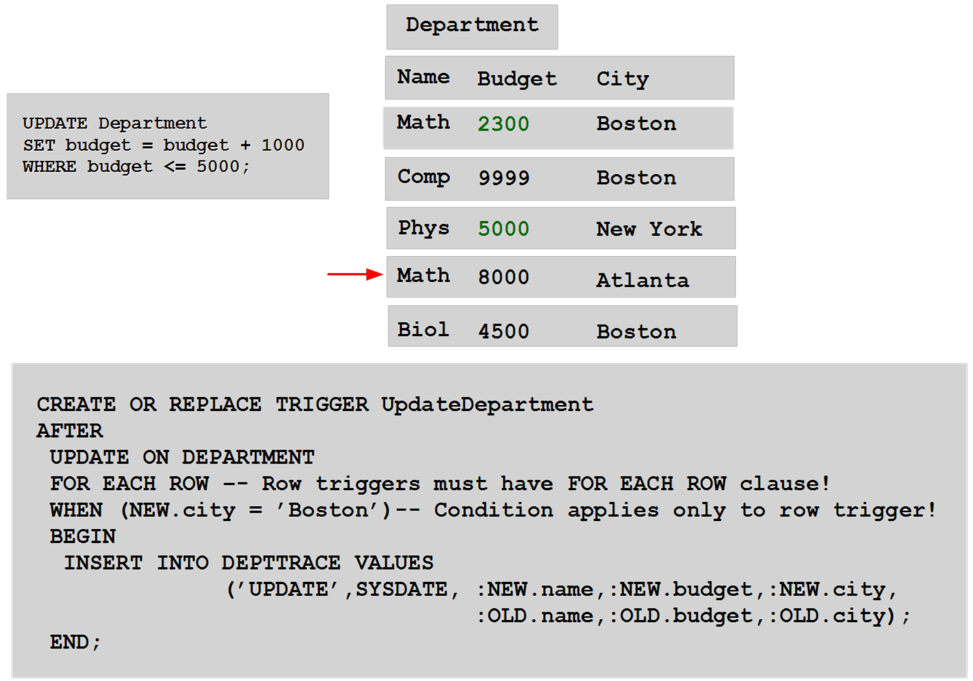
```
CREATE OR REPLACE TRIGGER UpdateDepartment
AFTER
 UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
              ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                 :OLD.name,:OLD.budget,:OLD.city);
 END;
```

A trigger fires after `UPDATE` of a row `[Phys 5000 New York]`

`WHEN` condition is not statisfied and a trigger does not process its body

Row database triggers

A sample processing of a row database trigger



```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	5000	New York
Math	8000	Atlanta
Biol	4500	Boston

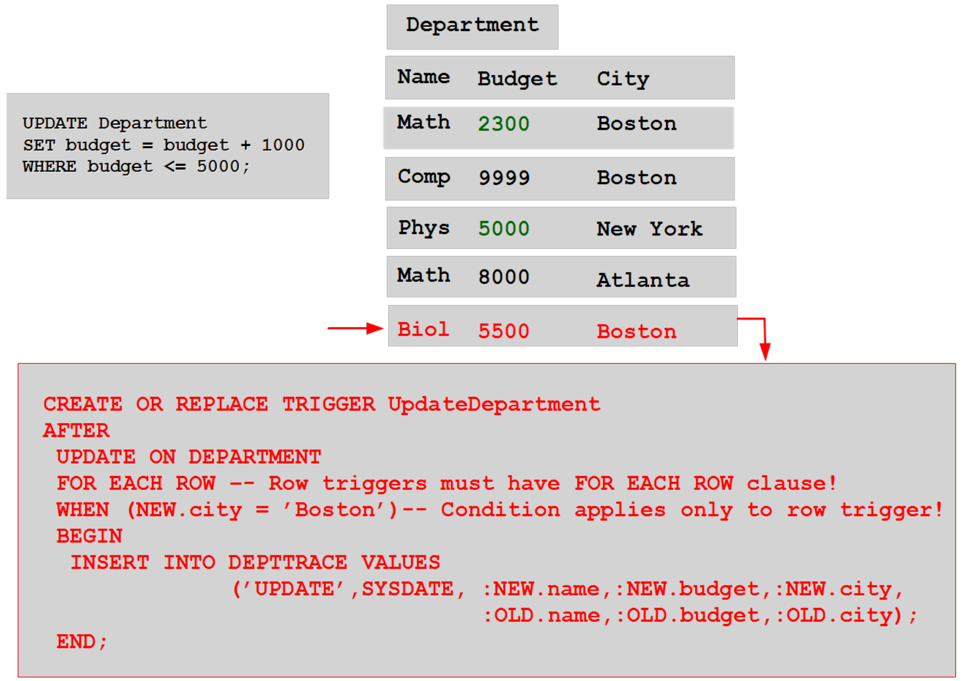
```
CREATE OR REPLACE TRIGGER UpdateDepartment
AFTER
 UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
              ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                 :OLD.name,:OLD.budget,:OLD.city);
 END;
```

A row `[Math 8000 Atlanta]` does not satisfy a condition in `WHERE` clause and it is not `UPDATE`ed

A trigger does not fire

Row database triggers

A sample processing of a row database trigger



```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	5000	New York
Math	8000	Atlanta
Biol	5500	Boston

```
CREATE OR REPLACE TRIGGER UpdateDepartment
AFTER
 UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
              ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                 :OLD.name,:OLD.budget,:OLD.city);
 END;
```

A trigger fires after `UPDATE` of a row `[Biol 5500 Boston]`

`WHEN` condition is statisfied and a trigger processes its body

Row database triggers

A sample processing of a row database trigger is completed

```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	5000	New York
Math	8000	Atlanta
Biol	5500	Boston

```
CREATE OR REPLACE TRIGGER UpdateDepartment
AFTER
 UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
             ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                :OLD.name,:OLD.budget,:OLD.city);
 END;
```

Row database triggers

Assume that while processing a rows trigger it attempts to access a relational table affected by a triggering event

For example, a triggers attempts to count the total number of rows in `UPDATE`ed realtional table

```
UPDATE Department
SET budget = budget + 1000
WHERE budget <= 5000;
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	4000	New York
Math	8000	Atlanta
Biol	4500	Boston

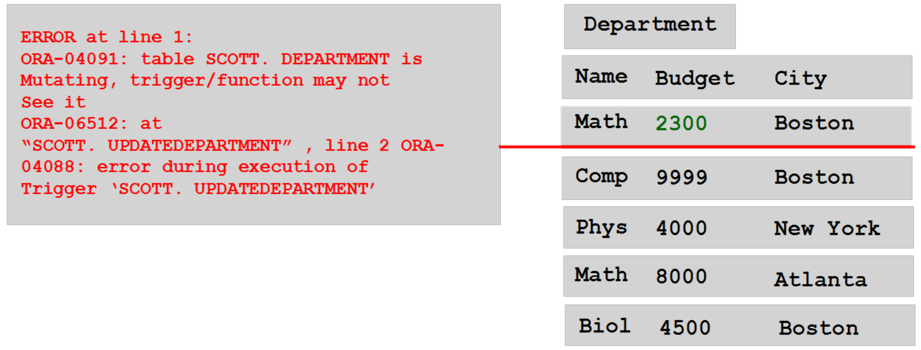
```
CREATE OR REPLACE TRIGGER UpdateDepartment
 AFTER UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
             ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                :OLD.name,:OLD.budget,:OLD.city);
  SELECT SUM(budget) FROM DEPARTMENT;
 END;
```

What is a correct of summation over a column `budget` ?

Row database triggers

It is impossible to provide a correct result of summation over a column `budget` while an `UPDATE` statement changes the values in the column

An outcome is a mutating table error when processing a row trigger



```
ERROR at line 1:
ORA-04091: table SCOTT. DEPARTMENT is
Mutating, trigger/function may not
See it
ORA-06512: at
“SCOTT. UPDATEDEPARTMENT” , line 2 ORA-
04088: error during execution of
Trigger ‘SCOTT. UPDATEDEPARTMENT’
```

Department

Name	Budget	City
Math	2300	Boston
Comp	9999	Boston
Phys	4000	New York
Math	8000	Atlanta
Biol	4500	Boston

```
CREATE OR REPLACE TRIGGER UpdateDepartment
 AFTER UPDATE ON DEPARTMENT
 FOR EACH ROW -- Row triggers must have FOR EACH ROW clause!
 WHEN (NEW.city = 'Boston')-- Condition applies only to row trigger!
 BEGIN
  INSERT INTO DEPTTRACE VALUES
              ('UPDATE',SYSDATE, :NEW.name,:NEW.budget,:NEW.city,
                                 :OLD.name,:OLD.budget,:OLD.city);
  SELECT SUM(budget) FROM DEPARTMENT;
 END;
```

Row database triggers

The solution to a mutating table error problem

- If a trigger fires on `INSERT` then use `BEFORE INSERT` temporal option
- Rewrite a trigger as a statement trigger
- Run a trigger as an autonomous transaction
- Record the modifications in a temporary table and fire a row trigger that reapplies the modifications as a statement trigger

Database Triggers

Outline

Other problems with triggers

Infinite chains of trigger invocations

- What to do when a trigger `A` while processing its body fires a trigger `B` and a trigger `B` while processing its body fires a trigger `A` ?

Indeterministic trigger invocations

- It may happen that due to a database transaction serialization mechanisms the same chain of trigger invocations will be processed (serialized) in many different way by a transaction scheduler, e.g. if two triggers `A` and `B` fire in more or less the same moment in time then sometimes `A` will be processed before `B` and sometimes `B` will be processed before `A`

Lack of external control

- Long chains of trigger invocations contribute to very serious data security risks, e.g. it is possible to "hide" malicious code at the end of long chains of trigger invocations

Lack of design methodology

- The ad hoc uncontrolled and not well planned additions of new triggers lead to a situation where after addition or modification of a trigger there is no certainty that the chains of trigger invocations do not corrupt a databse

References

T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 8.3 Triggers, Pearson Education Ltd, 2015

Database SQL Language Reference, `CREATE TRIGGER`

Database PL/SQL Language Reference, 9 PL/SQL Triggers