5 Triggers

Learning Goals:
- Understand what the purpose, the advantages and the disadvantages of triggers are
- Be able to create suitable triggers in appropriate situations

Active database mechanisms:
Basic concept: event, condition, action (ECA)

Example:
Event: deposit on bank account
Condition: account balance > 10,000 Euro
Action: Transfer money to savings account.

SQL syntax for triggers:

```
CREATE TRIGGER triggerName
BEFORE | AFTER <triggerEvent> ON <tableName>
[REFERENCING <oldOrNewValuesAliasList>]
[FOR EACH {ROW | STATEMENT}]
[WHEN (triggerCondition)]
<triggerBody>
```

- triggerEvent: INSERT, DELETE, or UPDATE (of rows in a table)
- FOR EACH ROW: execute the trigger body for each row affected by the event
- FOR EACH STATEMENT: (default) execute the trigger body only once for the entire event
- oldOrNewValuesAliasList: definition of aliases for an old or new row (OLD/NEW or OLD ROW/NEW ROW) in the case of a FOR EACH ROW trigger; or an old or new table (OLD TABLE/NEW TABLE), in the case of a FOR EACH STATEMENT trigger.
  Examples:
  REFERENCING OLD ROW AS OldBlabla, NEW ROW AS NewBlabla
  REFERENCING OLD TABLE AS X, NEW TABLE AS Y
  Here OLD TABLE refers to the set of all deleted tuples and/or all updated tuples before the delete and/or update was executed; NEW TABLE refers to the set of all newly inserted tuples and/or all tuples that were updated.
  Instead of using an explicit alias definition for old and new rows one can simply use the keywords OLD and NEW to refer to the old or new tuple. The alias definition is defined only from SQL-99 upwards.
- triggerBody:
  an SQL procedure statement. It can consist of several commands. In this case, they must be embraced by "begin" and "end".
  It cannot contain
  - SQL transaction statements (COMMIT, ROLLBACK)

- SQL connection statements (CONNECT, DISCONNECT)
- SQL schema definition or manipulation statements (creation/deletion of tables, user-defined types, or other triggers)
- and others (e.g. session statements)

Example:

Consider the University Database. Assume there is an additional relation TACandidates (matNr, classNr, pName) that lists suitable candidates among the students for teaching assistant jobs for different classes.

```
CREATE TABLE TACandidates
(matNr int(11),
classNr varchar(9),
pName varchar(15),
PRIMARY KEY(matNr, classNr));
```

When grades are assigned, we want to automatically insert into this table all students with grades better than 3.

```
CREATE TRIGGER findTACandidates
   AFTER INSERT ON Takes
   FOR EACH ROW
      INSERT INTO TACandidates
         (SELECT NEW.matNr, NEW.classNr, Class.pName
          FROM Class
          WHERE Class.classNr = NEW.classNr
          AND NEW.grade <= 3);

INSERT INTO Takes VALUES
(2345, 'DTB-SS93', 1.0);
```

Note:
We use "FOR EACH ROW" here since an INSERT can insert several tuples.
For each of them, the check is done whether the grade information reveals a suitable TA.

More examples

The following examples are taken from Elmasri, Navathe: "Fundamentals of Database Systems".

Consider the following relational database schema:

Employee (ssn, eName, salary, dNo, supervisorSsn)
where
- ssn = social security number, a unique number for each employee,
- dNo = department number, foreign key to Department, may have a NULL value if

an employee is (temporarily) not assigned to a specific department
- supervisorSsn = the ssn of the supervisor of this employee, (recursive) foreign key to Employee

Department (dNo, dName, sumSalary, managerSsn)
where
- dNo = a unique number identifying each department
- sumSalary = the sum of the salaries of all employees assigned to this department
- managerSsn = the ssn of the department's manager, foreign key to Employee

In this example we see the case of a derived attribute: sumSalary. Its values will change, when new employees are inserted, when some salary is updated, when an employee is deleted, or when an employee is reassigned from one department to another.
To keep the values of this attribute consistent with the rest of the data, the following set of triggers may be defined.

Triggers based on the specified database:

1. Insertion of new employees

Event: One or more new employees are inserted into the Employee table
Condition: The department numbers of the newly inserted employees are not NULL
Action: Update the sum of salaries in the respective departments

In SQL:

```
CREATE TRIGGER SumSalaryAfterNewEmps
AFTER INSERT ON Employee
FOR EACH ROW
WHEN (NEW.dNo IS NOT NULL)
    UPDATE Department
    SET sumSalary = sumSalary + NEW.salary
    WHERE dNo = NEW.dNo;
```

Note:
- FOR EACH ROW means: the action will be executed once for each tuple that is concerned by the event, i.e. here for each newly inserted employee.
- NEW refers to the tuples concerned by the event, in the form they have after the event, i.e. here they now exist

2. Update of the salary of one or more existing employees

Event: The salary of one ore more existing employees is changed.
Condition: The concerned employees are currently assigned to a specific department => dNo is not NULL
Action: Add the new salary to the sum of salaries and subtract the old one, i.e. add the difference of the salaries

In SQL:

```
CREATE TRIGGER SumSalaryAfterSalaryUpdate
AFTER UPDATE OF salary ON Employee
FOR EACH ROW
WHEN (NEW.dNo IS NOT NULL)
    UPDATE Department
    SET sumSalary = sumSalary + NEW.salary - OLD.salary
    WHERE dNo = NEW.dNo;
```

Note:

- FOR EACH ROW means here: the action will be executed once for each tuple where the salary was changed.
- NEW refers to the tuples concerned by the event, in the form they have after the event, i.e. here with their new value of salary
- OLD refers to the tuples concerned by the event, in the form they have before the event, i.e. here with the old value of salary.

3. An employee changes from one department to another one

Event: An employee changes to a new department (from a specified or unspecified one, i.e. it does not matter whether the old value was NULL or not)

Condition: In this case no condition needs to be specified since an action must always be taken:
- If the old value was NULL the new value will be not NULL; => a change in sum of salaries results.
- If the old value was not NULL, the new value may be NULL or not, but in both cases, the sum of salaries must be updated.
- We assume that a change of the dNo from NULL to NULL does not happen. If it should happen anyway, the trigger has no effect, since the where clauses are not fulfilled.

Action: The sum of salaries in the old and in the new department must be changed.

In SQL:

```
CREATE TRIGGER SumSalariesAfterDeptChange
AFTER UPDATE OF dNo ON Employee
FOR EACH ROW
    BEGIN
        UPDATE Department
        SET sumSalaries = sumSalaries + NEW.salary
        WHERE dNo = NEW.dNo;            (* if NEW.dNo = NULL, no update is done! *)
        UPDATE Department
        SET sumSalary = sumSalary - OLD.salary
        WHERE dNo = OLD.dNo;            (* if OLD.dno = NULL, no update is done! *)
    END;
```

Note:

- BEGIN and END are used here because there is more than one SQL statement in the action.
- The employee's salary is added to the new department's sum of salaries and subtracted from the old department's sum of salaries.

4. An employee is deleted

Event: An employee is deleted from the Employee table.
Condition: The employee was assigned to a department, i.e.the dNo was not NULL.
Action: The salary of the deleted employee is subtracted from the sum of salaries of the department.

In SQL:

```
CREATE TRIGGER SumSalariesAfterDeletion
AFTER DELETE ON Employee
FOR EACH ROW
WHEN (OLD.dNo IS NOT NULL)
      UPDATE Department
      SET sumSalary = sumSalary - OLD.salary
      WHERE dNo = OLD.dNo;
```

Note:

- OLD refers to the tuples concerned by the event, in the form they have before the event, i.e. here the employees before they are deleted.
- Examples 1.-4. are row level triggers, i.e. the triggers are fired once for each row that is concerned by the event, as is correct in the above examples. If FOR EACH ROW is not specified, the default FOR EACH STATEMENT holds, which would lead to incorrect results in the above examples since in one SQL statement, more than one employee can be inserted, deleted, or get a salary update. Example: UPDATE Employee SET salary = 1.1 * salary WHERE dNo = 5. This raises the salaries of all employees of department number 5. Hence several adjustments of the sum of salaries are needed. The keyword OLD and NEW can only be used with row level triggers.

5. Test whether the salary of an employee is higher than his/her supervisor's

Event: A new employee is inserted, an employee's salary is changed, or an employee is assigned a new supervisor.
Condition: The salary of the employee in question is higher than the supervisor's one.
Action: Inform the supervisor that the employee's salary is higher than his/her own.

In SQL:

```
CREATE TRIGGER SupervisorAlert
BEFORE INSERT OR UPDATE OF salary, supervisorSsn ON Employee
FOR EACH ROW
WHEN (NEW.salary > (SELECT salary FROM Employee)
                              WHERE ssn = NEW.supervisorSsn))
        InformSupervisor(NEW.supervisorSsn, NEW.ssn);
```

Note:

- There are two events specified in the BEFORE clause: "insert on Employee" and "update of the attributes salary or supervisorSsn on Employee". They are connected by logical "or".
- Explanation of the condition: When the salary of the newly inserted employee tuple or of the updated (in salary or supervisorSsn) employee tuple is greater than the salary of the supervisor who is specified in the new version of the employee tuple, then the action must be taken.
- It is assumed here that a routine by the name InformSupervisor exists and can be called.

Some notes about MySQL:

1. Displaying existing trigger:

With the MySQL command
`show triggers;`
you can list which triggers are already defined in the system.

Likewise, it is possible to directly query the table „triggers” in the database information_schema.

2. Creating triggers with several statements:

If the trigger body consists of several statements, you need to end each statement inside with a semicolon. This, however, would normally be understood by MySQL to end the entire trigger definition which would lead to a syntax error.

Therefore you should redefine the character that officially ends a SQL statement, for instance by
`DELIMITER //`

After this, statements can be ended by //, for instance
`select * from Student//`

It may be useful to redefine the delimiter after you have completed your trigger definition by entering
`DELIMITER ;`