SQL Triggers

Peter Scheuermann

Outline

- **Static Integrity Constraints**
  - Domain Constraints
  - Key / Referential Constraints
  - Semantic Integrity Constraints
- **Dynamic Integrity Constraints**

Triggers

- A **trigger** is a statement that is executed automatically if specified modifications occur to the DBMS.

- A trigger specification consists of three parts:

  ON *event* **IF** *condition* **THEN** *action*

  - *Event* (what activates the trigger?)
  - *Condition* (guard / test whether the trigger shall be executed)
  - *Action* (what happens if the trigger is run)

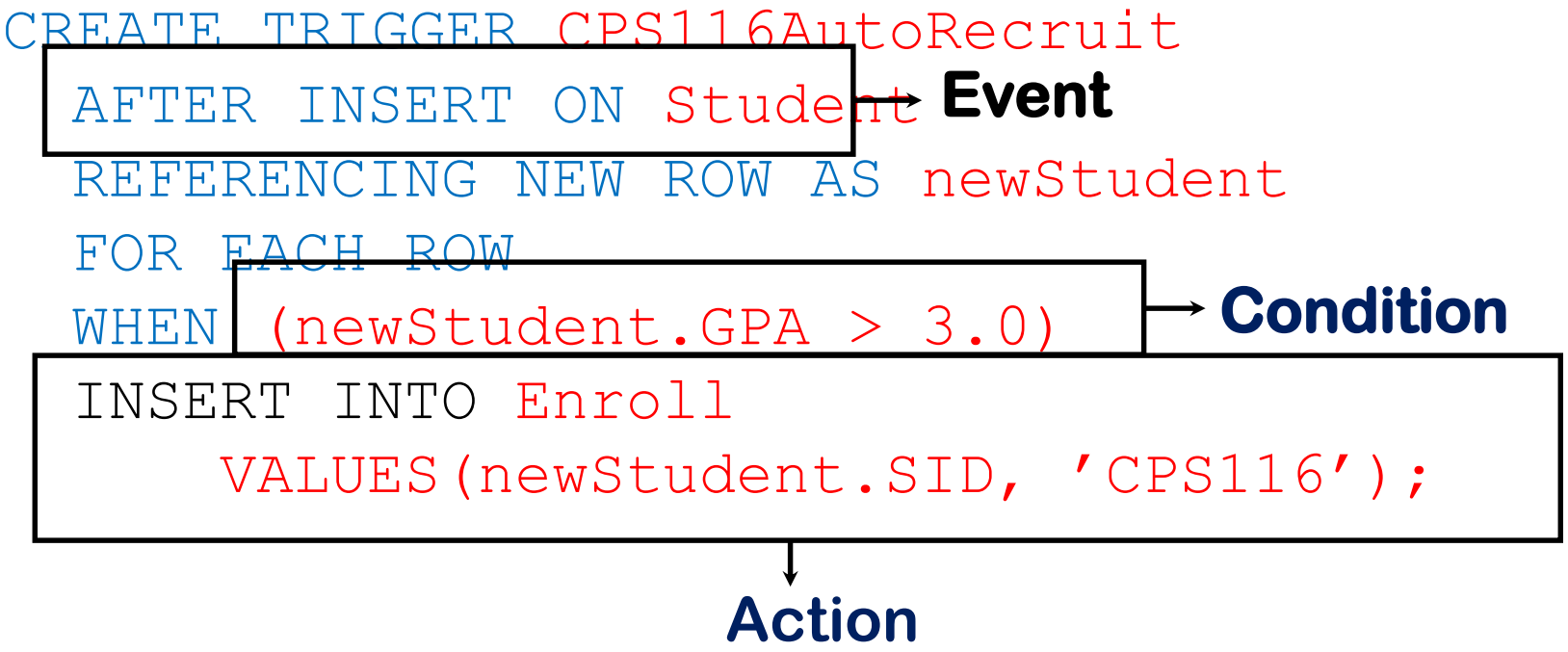
- Also called event-condition (ECA) rules

- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.

Why Triggers?

- Constraint maintenance
  - Triggers can be used to maintain foreign-key and semantic constraints; commonly used with ON DELETE and ON UPDATE
- Business rules
  - Some dynamic business rules can be encoded as triggers
- Monitoring
  - E.g. to react on the insertion of some kind of sensor reading into db
- Maintenance of auxiliary cached data
  - Careful! Many systems now support *materialized views* which should be preferred against such maintenance triggers
- Simplified application design
  - E.g. exceptions modelled as update operations on a database (if applicable)

Trigger example



Trigger options

- Possible events include:
  - *INSERT ON table*
  - *DELETE ON table*
  - *UPDATE* [`OF` *column*] `ON` *table*
- Granularity—trigger can be activated:
  - *FOR EACH ROW* modified
  - *FOR EACH STATEMENT* that performs modification
- Timing—action can be executed:
  - *AFTER or BEFORE* the triggering event

Trigger Example (SQL:1999)

```
CREATE TRIGGER gradeUpgrade
  AFTER INSERT INTO Assessment
  REFERENCING NEW TABLE Assess
  FOR EACH STATEMENT
    UPDATE Enrolled
       SET grade='P'
     WHERE Enrolled.sid=Assess.sid   AND
       Enrolled.ucode=Assess.ucode AND
       Assess.mark >= 50
```

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes

```
CREATE TRIGGER overdraft-trigger AFTER UPDATE OF balance
                                              ON account
```

- Values of attributes before and after an update can be referenced
  - **REFERENCING OLD ROW AS** *name* : for deletes and updates
  - **REFERENCING NEW ROW AS** *name* : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints.
  - E.g. convert blanks to null:

```
CREATE TRIGGER Setnull-trigger BEFORE UPDATE ON S
 REFERENCING NEW ROW AS nrow
 FOR EACH ROW
      WHEN nrow.country = ' '
       SET nrow.country = null
```

Transition variables

- **OLD ROW**: the modified row before the triggering event
- **NEW ROW**: the modified row after the triggering event
- **OLD TABLE**: a hypothetical read-only table containing all modified rows before the triggering event
- **NEW TABLE**: a hypothetical table containing all modified rows after the triggering event

☞ **Not all of them make sense all the time**, e.g.

- *AFTER INSERT* statement-level triggers
  - Can use only `NEW TABLE`
- *BEFORE DELETE* row-level triggers
  - Can use only `OLD ROW`
- etc.

Trigger Granularity

- **Granularity**
  - ***Row-level granularity***: change of a single row is an event (a single UPDATE statement might result in multiple events)
  - ***Statement-level granularity***: events are statements (a single UPDATE statement that changes multiple rows is a single event).

- Can be more efficient when dealing with SQL statements that update a large number of rows…

After Trigger Example
(statement granularity)

Keep track of salary averages in the log

```
CREATE TRIGGER RecordNewAverage
    AFTER UPDATE OF Salary ON Employee
    FOR EACH STATEMENT
    BEGIN
        INSERT INTO Log
        VALUES (CURRENT_DATE, SELECT AVG(Salary)
                              FROM Employee );
    END;
```

Trigger Granularity - Syntax

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use **`FOR EACH STATEMENT`** instead of **`for each row`**
  - Use **`REFERENCING OLD TABLE`**
    or **`REFERENCING NEW TABLE`**
    to refer to temporary tables (called *transition tables*) containing the affected rows

Triggers in SQL:1999

- **Events**: INSERT, DELETE, or UPDATE statements or changes to individual rows caused by these statements
  - Since SQL:2008: also INSTEAD OF triggers
- **Condition**: Anything that is allowed in a WHERE clause
- **Action**: An individual SQL statement or a program written in the language of Procedural Stored Modules (PSM) (which can contain embedded SQL statements)

Before Trigger Example (row granularity)

Check that enrollment ≤ limit

```
CREATE TRIGGER  Max_EnrollCheck
    BEFORE INSERT ON Transcript
    REFERENCING NEW AS N          --row to be added
    FOR EACH ROW
    WHEN ((SELECT COUNT (T.studId)
              FROM Transcript T
             WHERE T.CrsCode = N.CrsCode AND
                   T.semester = N.semester)
            >=
           (SELECT CRSLIMIT.maxEnroll
              FROM CRSLIMIT U
             WHERE U.CrsCode = N.CrsCode And
                   U.semester =N.semester)
))
    BEGIN
        ROLLBACK;
    END;
```

Some Tips on Triggers

- Use BEFORE triggers
  - For checking integrity constraints
- Use AFTER triggers
  - For integrity maintenance and update propagation
- In Oracle, triggers cannot access “mutating” tables
  - e.g. AFTER trigger on the same table which just updates

Triggers in SQL:1999

- **Consideration**: *Immediate*
  - Condition can refer to both the state of the affected row or table before *and* after the event occurs
- **Execution**: *Immediate* – can be before or after the execution of the triggering event
  - Action of before trigger cannot modify the database
- **Granularity**: Both *row-level* and *statement-level*

Design Space of Triggers

- **Activation** - Occurrence of the *event*
- **Consideration** - The point, after activation, when *condition* is evaluated
  - Immediate or deferred (when the transaction requests to commit)
  - *Condition* might refer to both the state before and the state after *event* occurs
- **Execution** – point at which *action* occurs
  - With deferred consideration, execution is also deferred
  - With immediate consideration, execution can occur immediately after consideration or it can be deferred
    - If execution is immediate, execution can occur before, after, or instead of triggering event.
    - Before triggers adapt naturally to maintaining integrity constraints: violation results in rejection of event.

You should now be able to:

- **Capture Integrity Constraints in an SQL Schema**
  - Including key constraints, referential integrity, domain constraints and semantic constraints
  - And simple triggers for dynamic consttraints
- Formulate complex semantic constraints using Assertions
- Know when to use Assertions, when triggers, and when CHECK constraints
- Know the semantic of deferring integrity constraints

- **Be able to formulate simple triggers**
- Know the difference between row-level & statement-level triggers