Triggers

SWEN 304
Trimester 2, 2017

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Slides by: Pavle Morgan & Hui Ma

Outline

- Triggers and active databases
- The structure of a trigger
- PostgreSQL trigger syntax
- PL/pgSQL for Triggers

- Readings:
  - *Textbook, Section 26.1*
  - *PostgreSQL 9.4 Documentation , CREATE TRIGGER statement, Chapter V.34 and Chapter V.37*

Triggers and Active Databases

- Active databases are founded on the use of active rules
- Active rules are most often implemented by triggers
- A trigger is a program that is invoked as an answer to a situation
- A situation may be a certain database state or an event
- A DBMS monitors situations and invokes triggers
- Users may be unaware of trigger existence

Motivation Examples

- Events, like updates made to database relations, may provoke some actions to be undertaken afterwards or even before
- Examples:
  - After recording the pass of an exam, the number of points in student's record should be augmented appropriately
  - When the number of points scored becomes equal to or greater than 360, student's data should be recorded in the *Graduation* table
  - Before a new tuple is inserted, the referential integrity constraint should be checked
- Instead of embedding commands to monitor events in each program that updates a table, an appropriate trigger is associated with the table and executed automatically whenever event occurs

Triggers and DBMS Vendors

- Many major DBMS vendors introduced triggers during ninetieth, e.g. Oracle, DB2, and SYBASE
- The SQL Standard:1999 requires support of triggers
- **Triggers** are executed when a specified condition occurs during insert/delete/update
  - Triggers are action that fire automatically based on these conditions

Triggers

- Common uses of triggers
  - To enforce complex business rules
  - To enforce integrity constraints
  - To generate value for derived attributes
  - To centralise global operations, e.g., maintaining replicable tables

Event-Condition-Action (ECA) Model

- Triggers follow an Event-condition-action (**ECA**) model
  - **Event**:
    - Database modification, e.g., INSERT, DELETE, UPDATE
  - **Condition**:
    - Any true/false expression
      - Optional: if no condition is specified then condition is always true
  - **Action**:
    - Sequence of SQL statements that will be automatically executed,
      - e.g. user defined function

Condition

- Any true/false **condition** to control whether a trigger is activated or not
  - Absence of condition means that the trigger will always execute for the event
  - Otherwise, condition is evaluated
    - before the event for BEFORE trigger
    - after the event for AFTER trigger

Condition Evaluation

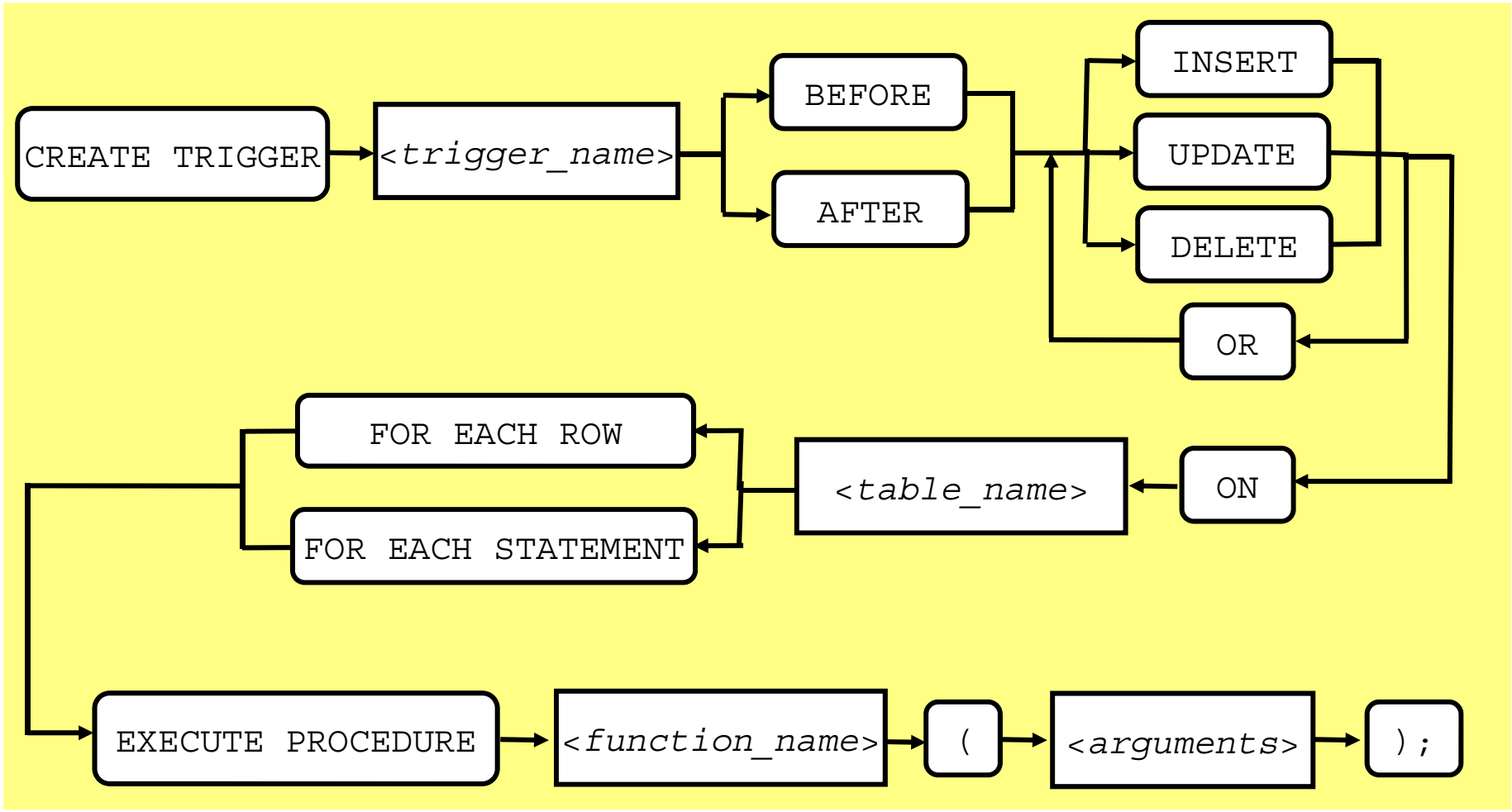
- Three ways of condition evaluation/rule consideration
  - **Immediate** consideration
    - Part of the same transaction and can be one of the following depending on the situation
      - Before
      - After
  - **Deferred** consideration
    - Condition is evaluated at the end of the transaction
  - **Detached** consideration
    - Condition is evaluated in a separate transaction

Triggers and Active Databases

Potential Applications for Active Databases

- Notification
  - Automatic notification when certain condition occurs
  - Enforcing integrity constraints
    - Triggers are smarter and more powerful than constraints
  - Maintenance of derived data
    - Automatically update derived data and avoid anomalies due to redundancy
      - E.g., trigger to update the Total points

Trigger Layout



Timing of the Action

- An important issue is when the action part of the trigger is going to be executed:
  - AFTER: executes after the event
  - BEFORE: executes before the event
- For example:
  - A trigger that maintains referential integrity constraint makes sense to execute before a tuple is inserted, or deleted
  - A trigger that monitors the number of students enrolled in a school should be executed after successful insertion of a new enrollment tuple

Before and After Triggers

- Before triggers don't see the final value of the row affected by
- After triggers see the final value of the row affected by
- Typically, `row before` triggers are used for checking or modifying data that will be inserted or updated
- `row after` triggers are mostly sensibly used to propagate updates to other tables, or make consistency checks against tables

Row-Level versus Statement-level

- Triggers can be
  - **Row-level**
    - FOR EACH ROW specifies a row-level trigger
  - **Statement-level**
    - Default (when FOR EACH ROW is not specified)
- Row level triggers
  - Executed separately for each affected row
- Statement-level triggers
  - Execute once for the SQL statement,

The Action of a Trigger

- Action can be
  - One SQL statement
  - A sequence of SQL statements enclosed between a BEGIN and an END
  - A database transaction program, or an external program
- Action specifies the relevant modifications
- In PostgreSQL environment a trigger action is a special UDF that can be written in PL/pgSQL, or some other procedural languages like C

The Trigger Syntax in PL/pgSQL

```
<trigger>::= CREATE TRIGER <trigger_name>
 {AFTER | BEFORE} <triggering_events>
 ON <table_name>
 [FOR [EACH] { ROW | STATEMENT }]
 EXECUTE PROCEDURE <function_name>(arguments);
```

```
<triggering_events>::=
   <triggering_event> [OR <triggering_event>…..]
```

```
<triggering_event>::= {INSERT | DELETE | UPDATE}
```

Trigger Procedures in PL/pgSQL

- A trigger PROCEDURE is created with

  `CREATE FUNCTION`

  command, which:
  - Does not have any parameters, and
  - Has a `trigger` return type
- When a PL/pgSQL function is called as a trigger, several special variables are created automatically in the top level block

Automatically Created Variables

- The most important automatically created variables:
  - `NEW` of data type `RECORD` holding the new database row for `INSERT/UPDATE` operations in row-level triggers
  - `OLD` of data type `RECORD` holding the old database row for `UPDATE/DELETE` operations in row-level triggers

`RETURN` Type

- A trigger has to return:
  - Either `NULL`, or
  - A record/row value having exactly the structure of the table the trigger was fired for
- The return value of a:
  - `BEFORE` or `AFTER` per-statement trigger, or
  - An `AFTER` row-level trigger is always ignored
  - Both should be `NULL`
  - But they can still abort an entire operation by raising an error

Returning a Value Being Not `NULL`

- A row level trigger fired `BEFORE` may return `NULL` to signal the trigger manager to skip the rest of operations for this row (`INSERT/DELETE/UPDATE` will not be executed for this row)
- If a `BEFORE` row level trigger returns a non null row value, the operation proceeds with that row value

Triggers - Examples

- Consider the following part of a relational database schema:

  Student (StudId, Name, NoOfPts, Degree)

  Exam (StudId, CourseId, Term, Grade)

  Exam [StudId] $\subseteq$ Student [StudId]

- Suppose *Grade* is NOT NULL
- Suppose all courses are worth *15* points
- Whenever a student passes a course, *15* points is added to her/his *NoOfPts*

Trigger Example: Add_Points

```
CREATE OR REPLACE FUNCTION add_points()
RETURNS trigger AS $$
BEGIN
  IF (NEW.Grade < 'D') THEN
    UPDATE Student SET NoOfPts = NoOfPts + 15
    WHERE StudId = NEW.StudId;
  END IF;
  RETURN NULL;
END;
$$ LANGUAGE 'PLpgSQL';
```

```
CREATE TRIGGER add_points
AFTER INSERT ON Exam
FOR EACH ROW EXECUTE PROCEDURE add_points();
```

Triggers – Potentials and Pitfalls

- Triggers have great potential to simplify database system development, because they relay on object-oriented features like encapsulation and message exchange
- But, it is quite difficult to verify that a set of rules is consistent (i.e. that there are no two rules in the set that contradict each other)
- It is also hard to guarantee termination of a set of rules

A Question for You

- Is there anything wrong with the following two triggers:

```
CREATE TRIGGER t1
AFTER INSERT ON table1
FOR EACH ROW EXECUTE PROCEDURE
update_table_2();
```

```
CREATE TRIGGER t2
AFTER UPDATE ON table2
FOR EACH ROW EXECUTE PROCEDURE
insert_into_table_1();
```

Summary

- Active databases react automatically on certain events by activating triggers
- Structure of a trigger encompass:
  - Timing of the trigger (`BEFORE` or `AFTER`)
  - Event that fires the trigger (`INSERT`, or `DELETE`, or `UPDATE`),
  - Scope (`FOR EACH ROW`, or `STATEMENT`)
  - Monitored table, and
  - Action that actually enforces some business or integrity rules
- Action programs are server side functions