

Triggers Activity – Monday November 14

In this activity you'll explore some of the subtleties of triggers, using SQLite and PostgreSQL.

Sample trigger script in SQLite

```
create table T(A int);
create trigger R
after insert on T
for each row
when (new.A > 3)
begin
    update T set A = 5 where A = 4;
    update T set A = A+1 where A = 5;
end;
insert into T values (2); insert into T values (3); insert into T values (4);
insert into T values (5); insert into T values (6); insert into T values (7);
select * from T;
drop trigger R; drop table T;
```

Sample trigger script in PostgreSQL

```
create table T(A int);
create function Rf() returns trigger as
'
declare
begin
    update T set A = 5 where A = 4;
    update T set A = A+1 where A = 5;
    return null;
end;
'
language plpgsql;
create trigger R
after insert on T
for each row
when (new.A > 3)
execute procedure Rf();
insert into T values (2); insert into T values (3); insert into T values (4);
insert into T values (5); insert into T values (6); insert into T values (7);
select * from T;
drop trigger R on T; drop function Rf(); drop table T;
```

Problem 1: Row-level versus statement-level triggers

Only PostgreSQL supports both row-level and statement-level triggers. If a PostgreSQL trigger does not have the “for each row” clause, then it is a statement-level trigger. As a reminder, a row-level trigger executes once for each modified row, while a statement-level trigger executes once at the end of the modification statement. Demonstrate the difference by writing two PostgreSQL triggers, identical except one includes “for each row” and the other doesn’t, that behave differently. To show their different behavior, start with exactly the same database, perform exactly the same modification(s), and show how using one of the triggers ends with a different database state than using the other trigger.

Problem 2: Different semantics for row-level “after” triggers

The correct semantics for a row-level “after” trigger, according to the SQL standard, is to activate the trigger after the entire triggering data modification statement completes, executing the trigger once for each modified row. PostgreSQL implements this semantics. SQLite instead implements a semantics where the trigger is activated immediately after each row-level change, interleaving trigger execution with execution of the modification statement. Demonstrate the difference by writing one or more row-level “after” triggers for PostgreSQL and SQLite that are identical (modulo syntactic differences across the systems), but behave differently on PostgreSQL and SQLite. To show the different behavior on PostgreSQL and SQLite, start with exactly the same database, perform exactly the same modification(s), and show how the two systems end with different database states.

Problem 3: When do row-level “before” triggers run?

The SQL standard is somewhat unclear about the semantics for a row-level “before” trigger. Is it activated before the entire modification statement executes, firing once for each to-be-modified row? Or is it activated and fired immediately before each row-level change, interleaved with execution of the modification statement? Try to determine which behavior is implemented in PostgreSQL and in SQLite. Create one or more triggers, an initial database state, and triggering modifications that demonstrate clearly one possible behavior over the other.