





**Atzeni, Ceri, Fraternali,
Paraboschi, Torlone**
Basi di dati
Quinta edizione
McGraw-Hill Education, 2018

Capitolo 5:
SQL: caratteristiche evolute
Esercitazione su trigger in
Postgres

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e
©2018 McGraw-Hill Education (Italy) S.r.l.

1




Alcune slide sono riconducibili e riprese dalle slide originali realizzate dagli autori del libro di riferimento e sono reperibili su internet e da altro materiale reperibile online. Questa presentazione contiene anche alcuni estratti della documentazione di PostgreSQL, reperibile online sul sito di Postgres. Infine alcune slide contengono esercizi proposti dal prof A. D'Acierno come indicato in calce nelle slide.

Atzeni, Ceri, Fraternali, Paraboschi, Torlone
Basi di dati
Quinta edizione
McGraw-Hill Education, 2018


Capitolo 5:
SQL: caratteristiche evolute

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e
©2018 McGraw-Hill Education (Italy) S.r.l.

2



Triggers in PostgreSQL




On tables

- triggers can be defined to execute either before or after any INSERT, UPDATE, or DELETE operation, either once per modified row, or once per SQL statement.
- UPDATE triggers can moreover be set to fire only if certain columns are mentioned in the SET clause of the UPDATE statement.
- FOR EACH ROW/FOR EACH STATEMENT specifies whether the trigger function should be fired
- once for every row affected by the trigger event, or just once per SQL statement. If neither is specified, FOR EACH STATEMENT is the default.


P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

3



Creating the trigger function



A trigger function is similar to an ordinary function, except that it does not take any arguments and has return value type `trigger` as follows:

```
1 CREATE FUNCTION trigger_function() RETURN trigger AS
```

Notice that you can create trigger functions using any languages supported by PostgreSQL. In this tutorial, we will use PL/pgSQL for the demonstration.



The trigger function receives data about their calling environment through a special structure called *TriggerData*, which contains a set of local variables. For example, `OLD` and `NEW` represent the states of row in the table before or after the triggering event. PostgreSQL provides other local variables starting with `TG_` as the prefix such as `TG_WHEN`, `TG_TABLE_NAME`, etc.

Once the trigger function is defined, you can bind it to specific actions on a table.

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

4

Creating the trigger

To create a new trigger, you use the `CREATE TRIGGER` statement. The complete syntax of the `CREATE TRIGGER` is complex with many options. However, for the sake of demonstration, we will use the simple form of the `CREATE TRIGGER` syntax as follows:

```

1 CREATE TRIGGER trigger_name {BEFORE | AFTER | INSTEAD OF} {event [OR ...]}
2   ON table_name
3   [FOR [EACH] {ROW | STATEMENT}]
4   EXECUTE PROCEDURE trigger_function

```

The event could be `INSERT`, `UPDATE`, `DELETE` or `TRUNCATE`. You can define trigger that fires before (`BEFORE`) or after (`AFTER`) event. The `INSTEAD OF` is used only for `INSERT`, `UPDATE`, or `DELETE` on the views.

As mentioned in the [introduction to PostgreSQL trigger](#), PostgreSQL provides two kinds of triggers: row level trigger and statement level trigger, which can be specified by `FOR EACH ROW` (row level trigger) or `FOR EACH STATEMENT` (statement level trigger).

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

5

Trigger: Esempio 1

```

drop table if exists materiale cascade;
drop table if exists ordine cascade;

create table materiale (
    idPezzo integer primary key,
    qntDisponibile integer not null,
    qntLimite integer not null,
    qntRiordino integer not null);

create table ordine (
    idPezzo integer primary key references materiale(idPezzo),
    qntRiordino integer not null,
    dataOrdine date not null);

```

Regola aziendale: appena la quantità disponibile di un certo pezzo scende sotto la quantità limite bisogna inserire un riga corrispondente nella tabella ordine.

15

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

6

```

create OR replace function RIORDINO() returns trigger as $$
begin
    if (not exists (select * from ordine where idPezzo = NEW.idPezzo)) then
        insert into ordine (idpezzo, qntRiordino, dataordine)
        values (NEW.idPezzo, NEW.qntRiordino, current_date);
    end if;
return NEW;
end $$ language plpgsql;

create trigger riordino
after update of qntDisponibile on materiale
for each row
when (NEW.qntDisponibile < NEW.qntLimite)
execute procedure Riordino();

```

16

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

7

```

insert into materiale(idpezzo,qntDisponibile,qntLimite,qntRiordino)
values(1,200,150,100);
insert into materiale(idpezzo,qntDisponibile,qntLimite,qntRiordino)
values(2,780,500,200);
insert into materiale(idpezzo,qntDisponibile,qntLimite,qntRiordino)
values(3,450,400,120);
insert into materiale(idpezzo,qntDisponibile,qntLimite,qntRiordino)
values(4,350,400,120);

```

```

update materiale set
qntDisponibile = qntDisponibile - 70
where idPezzo=2;

```

```

update materiale set
qntDisponibile = qntDisponibile - 70
where idPezzo=1;

```

```

update materiale set
qntDisponibile = qntDisponibile - 70
where idPezzo <= 3;

```

idpezzo	qntriordino	dataordine
integer	integer	date

idpezzo	qntriordino	dataordine
integer	integer	date
1	1	100
		2019-04-03

idpezzo	qntriordino	dataordine
integer	integer	date
	1	100
		2019-04-03
	3	120
		2019-04-03

17

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

8

Valori di ritorno per funzioni usate nei triggers

- Una funzione ha un valore di ritorno specifico dall'istruzione RETURN
- Se il trigger e' definito BEFORE e FOR EACH ROW, tale valore puo' essere NULL (i.e. l'operazione sulla riga corrente e' annullata) oppure una variabile di tipo RECORD
- Se il trigger e' definito AFTER oppure FOR EACH STATEMENT, il valore di ritorno e' ignorato

Variabili NEW e OLD

All'interno del blocco di istruzioni di una funzione PL/pgSQL per un trigger sono disponibili alcune variabili speciali tra cui:

- **NEW**: In operazioni INSERT o UPDATE, rappresenta la **nuova riga della tabella che si vuole aggiornare**.
- **OLD**: In operazioni DELETE o UPDATE, rappresenta la **riga della tabella che si vuole cancellare o modificare**
- Le variabili NEW e OLD sono di tipo RECORD
- I singoli attributi sono denotati NEW.<nome_colonna> e OLD.<nome_colonna>

9

Note ...

- Trigger functions invoked by per-statement triggers should always return NULL.
- Trigger functions invoked by per-row triggers can return a table row to the calling executor, if they choose.
- A row-level trigger fired BEFORE an operation has the following choices:
 - It can return NULL to skip the operation for the current row. This instructs the executor to not perform the row-level operation that invoked the trigger (the insertion, modification, or deletion of a particular table row).
 - For row-level INSERT and UPDATE triggers only, the returned row becomes the row that will be inserted or will replace the row being updated. This allows the trigger function to modify the row being inserted or updated.
 - A row-level BEFORE trigger that does not intend to cause either of these behaviors must be careful to return as its result the same row that was passed in (that is, the NEW row for INSERT and UPDATE triggers, the OLD row for DELETE triggers).

10

Trigger: Esempio 2

- Dato un DB costituito dalla sola relazione

Impiegato(Matricola, Cognome, Nome)

impostare le strutture necessarie a tenere traccia di tutte le operazioni di modifica dati su tale tabella.

- Quali strutture servono?
 - Una tabella in cui memorizzare le modifiche
 - Tre trigger: su insert, su update, su delete

19

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

12

```
drop table if exists impiegato cascade;  
drop table if exists storiaimpiegato cascade;
```

Creare le tabelle

```
create table IMPIEGATO (  
  matricola char(4) primary key,  
  cognome varchar(20),  
  nome varchar(20));
```

```
create table STORIAIMPIEGATO (  
  utente varchar(20),  
  tempo timestamp,  
  operazione varchar(20),  
  oldmatricola char(4),  
  oldcognome varchar(20),  
  oldnome varchar(20),  
  newmatricola char(4),  
  newcognome varchar(20),  
  newnome varchar(20),  
  primary key (utente,tempo));
```

20

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

13

Trigger su insert

```
CREATE OR REPLACE FUNCTION archivia_insert() RETURNS trigger AS $
$
BEGIN
INSERT INTO storiainpleado (utente, tempo, operazione, newmatricola,
newcognome, newnome)
VALUES(user, clock_timestamp(), 'INSERT', NEW.matricola, NEW.cognome,
NEW.nome);
RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER archivia_insert
AFTER INSERT ON empleado
FOR EACH ROW EXECUTE PROCEDURE archivia_insert();
```

21

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

14

Trigger su delete

```
CREATE OR REPLACE FUNCTION archivia_delete() RETURNS trigger AS
$$
BEGIN
INSERT INTO storiainpleado (utente, tempo, operazione, oldmatricola,
oldcognome, oldnome)
VALUES(user, clock_timestamp(), 'DELETE', OLD.matricola, OLD.cognome,
OLD.nome);
RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER archivia_delete
AFTER DELETE ON empleado
FOR EACH ROW EXECUTE PROCEDURE archivia_delete();
```

22

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

15

Trigger su update

```
CREATE OR REPLACE FUNCTION archivia_update() RETURNS trigger AS
$$
BEGIN
INSERT INTO storiainpiegato (utente, tempo, operazione,
                           newmatricola, newcognome, newnome,
                           oldmatricola, oldcognome, oldnome)
VALUES(user, clock_timestamp(), 'UPDATE',
       NEW.matricola, NEW.cognome, NEW.nome,
       OLD.matricola, OLD.cognome, OLD.nome);
RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER archivia_update
AFTER UPDATE ON impiegato
FOR EACH ROW EXECUTE PROCEDURE archivia_update();
```

23

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

16

The screenshot shows the pgAdmin 4 interface. The SQL query editor contains the following SQL code:

```
TestVari on postgres@localhost PostgreSQL 10
1 insert into impiegato(matricola, cognome,nome) values('AAA1','Rossi','Mario');
2 insert into impiegato(matricola, cognome,nome) values('AAA2','Rossi','Carlo');
3 insert into impiegato(matricola, cognome,nome) values('AAA3','Bianchi','Antonio');
4 insert into impiegato(matricola, cognome,nome) values('AAA4','Bianchi','Francesco');
5 insert into impiegato(matricola, cognome,nome) values('AAA5','Verdi','Ciro');
6 insert into impiegato(matricola, cognome,nome) values('AAA6','Verdi','Gennaro');
7 insert into impiegato(matricola, cognome,nome) values('BBB1','Paolino','Paperino');
8 insert into impiegato(matricola, cognome,nome) values('BBB2','Paperon','De paperoni');
9 update impiegato set Cognome = 'Esposito' where matricola = 'AAA6';
10 delete from impiegato where matricola = 'AAA1';
11 delete from impiegato where matricola like 'BBB.';
12
13 select * from storiainpiegato;
```

The Data Output tab shows the results of the query:

	utente character varying (20)	tempo timestamp without time zone	operazione character varying (20)	oldmatricola character (4)	oldcognome character varying (20)	oldnome character varying (20)	newmatricola character (4)	n c
1	postgres	2018-05-02 12:03:46.381281	INSERT	[null]	[null]	[null]	AAA1	R
2	postgres	2018-05-02 12:03:46.381767	INSERT	[null]	[null]	[null]	AAA2	R
3	postgres	2018-05-02 12:03:46.381964	INSERT	[null]	[null]	[null]	AAA3	B
4	postgres	2018-05-02 12:03:46.382175	INSERT	[null]	[null]	[null]	AAA4	B
5	postgres	2018-05-02 12:03:46.382316	INSERT	[null]	[null]	[null]	AAA5	V
6	postgres	2018-05-02 12:03:46.382686	INSERT	[null]	[null]	[null]	AAA6	V
7	postgres	2018-05-02 12:03:46.382779	INSERT	[null]	[null]	[null]	BBB1	P
8	postgres	2018-05-02 12:03:46.382847	INSERT	[null]	[null]	[null]	BBB2	P
9	postgres	2018-05-02 12:03:46.383248	UPDATE	AAA6	Verdi	Gennaro	AAA6	E
10	postgres	2018-05-02 12:03:46.383758	DELETE	AAA1	Rossi	Mario	[null]	[r]
11	postgres	2018-05-02 12:03:46.384052	DELETE	BBB1	Paolino	Paperino	[null]	[r]
12	postgres	2018-05-02 12:03:46.384189	DELETE	BBB2	Paperon	De paperoni	[null]	[r]

17

Trigger: Esempio 2

- Dato un DB costituito dalla sola relazione

Impiegato(Matricola, Cognome, Nome)

impostare le strutture necessarie a tenere traccia di tutte le operazioni di modifica dati su tale tabella.

18

Trigger: Esempio 2

- Dato un DB costituito dalla sola relazione

Impiegato(Matricola, Cognome, Nome)

impostare le strutture necessarie a tenere traccia di tutte le operazioni di modifica dati su tale tabella.

- Quali strutture servono?
 - Una tabella in cui memorizzare le modifiche
 - Tre trigger: su insert, su update, su delete

19

drop table if exists impiegato cascade;
drop table if exists storiaimpiegato cascade;

create table IMPIEGATO (
matricola char(4) primary key,
cognome varchar(20),
nome varchar(20));

Creare le tabelle

create table STORIAIMPIEGATO (
utente varchar(20),
tempo timestamp,
operazione varchar(20),
oldmatricola char(4),
oldcognome varchar(20),
oldnome varchar(20),
newmatricola char(4),
newcognome varchar(20),
newnome varchar(20),
primary key (utente,tempo));

20

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

20

Trigger su insert

```

CREATE OR REPLACE FUNCTION archivia_insert() RETURNS trigger AS $
$
BEGIN
INSERT INTO storiaimpiegato (utente, tempo, operazione, newmatricola,
newcognome, newnome)
VALUES(user, clock_timestamp(), 'INSERT', NEW.matricola, NEW.cognome,
NEW.nome);
RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER archivia_insert
AFTER INSERT ON impiegato
FOR EACH ROW EXECUTE PROCEDURE archivia_insert();

```

21

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

21

Trigger su delete

```
CREATE OR REPLACE FUNCTION archivia_delete() RETURNS trigger AS
$$
BEGIN
INSERT INTO storiainpleado (utente, tempo, operazione, oldmatricola,
oldcognome, oldnome)
VALUES(user, clock_timestamp(), 'DELETE', OLD.matricola, OLD.cognome,
OLD.nome);
RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER archivia_delete
AFTER DELETE ON empleado
FOR EACH ROW EXECUTE PROCEDURE archivia_delete();
```

22

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

22

Trigger su update

```
CREATE OR REPLACE FUNCTION archivia_update() RETURNS trigger AS
$$
BEGIN
INSERT INTO storiainpleado (utente, tempo, operazione,
newmatricola, newcognome, newnome,
oldmatricola, oldcognome, oldnome)
VALUES(user, clock_timestamp(), 'UPDATE',
NEW.matricola, NEW.cognome, NEW.nome,
OLD.matricola, OLD.cognome, OLD.nome);
RETURN NULL;
END
$$ LANGUAGE plpgsql;

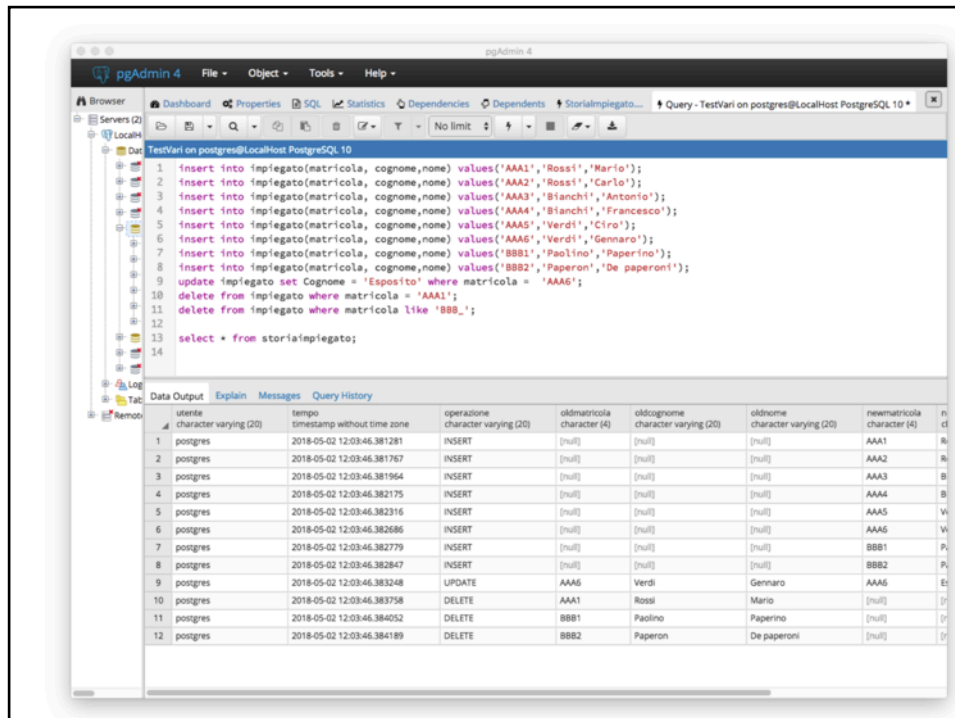
CREATE TRIGGER archivia_update
AFTER UPDATE ON empleado
FOR EACH ROW EXECUTE PROCEDURE archivia_update();
```

23

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

23



24

Trigger: Esempio 3

```

create table PersonaleMedico (
    matricola char(4) primary key,
    cognome varchar(20));

```

```

create table PersonaleParaMedico (
    matricola char(4) primary key,
    cognome varchar(20));

```

- Non possono esistere un medico ed un paramedico con la stessa matricola.

25

Trigger: Esempio 3

```
create function check_matr_paramedico() returns trigger as $$
BEGIN
if (exists (select * from PersonaleParaMedico where matricola = new.matricola)) then
    raise exception 'Matricola presente PersonaleParamedico';
end if;
RETURN new;
END
$$ LANGUAGE plpgsql;
```

26

Trigger: Esempio 3

```
create function check_matr_paramedico() returns trigger as $$
BEGIN
if (exists (select * from PersonaleParaMedico where matricola = new.matricola)) then
    raise exception 'Matricola presente PersonaleParamedico';
end if;
RETURN new;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_insert
BEFORE INSERT ON PersonaleMedico
FOR EACH ROW EXECUTE PROCEDURE check_matr_paramedico();

CREATE TRIGGER check_update
BEFORE UPDATE OF matricola ON PersonaleMedico
FOR EACH ROW EXECUTE PROCEDURE check_matr_paramedico();
```

27

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

27

Trigger: Esempio 3

- Cosa succede se i trigger li faccio After?
- In questo caso, non cambia nulla!
- Un trigger ed il comando che lo scatena sono visti come un comando unico!

28

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

28

Trigger: Esempio 4

```
create table IMPIEGATO (  
  matricola char(4) primary key,  
  cognome varchar(20) not null,  
  nome varchar(20) not null);
```

Regola aziendale: per ogni operazione di insert voglio memorizzare il numero di impiegati inseriti ed il numero di righe presente nella tabella dopo l'inserimento.

```
create table STORIAIMPIEGATO (  
  utente varchar(20),  
  tempo timestamp,  
  operazione varchar(20),  
  newRowsNumber integer,  
  insertedrows integer,  
  primary key (utente,tempo));
```

29

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acierno

La Programmazione

29

Trigger: Esempio 4

```
CREATE OR REPLACE FUNCTION archivia_insert() RETURNS trigger AS
$$
BEGIN
INSERT INTO storiapiiegato(utente, tempo, operazione,
newRowsNumber, insertedRows)
SELECT user, clock_timestamp(), 'INSERT',
(select count(*) From impiegato),
(select count(*) From newtab);
RETURN NULL;
END
$$ LANGUAGE plpgsql;
```

30

Trigger: Esempio 4

```
CREATE OR REPLACE FUNCTION archivia_insert() RETURNS trigger AS
$$
BEGIN
INSERT INTO storiapiiegato(utente, tempo, operazione,
newRowsNumber, insertedRows)
SELECT user, clock_timestamp(), 'INSERT',
(select count(*) From impiegato),
(select count(*) From newtab);
RETURN NULL;
END
$$ LANGUAGE plpgsql;
CREATE TRIGGER archivia_insert
AFTER INSERT ON impiegato
REFERENCING NEW TABLE AS newtab
FOR EACH STATEMENT
EXECUTE PROCEDURE archivia_insert();
```

30

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

31

Trigger: Esempio 4

```
create table IMPIEGATOTEMP (  
  matricola char(4) primary key,  
  cognome varchar(20) not null,  
  nome varchar(20) not null);  
  
insert into IMPIEGATOTEMP(matricola, cognome,nome)  
values('BBB1','Rossi','Mario');  
insert into IMPIEGATOTEMP(matricola, cognome,nome)  
values('BBB2','Rossi','Carlo');  
insert into IMPIEGATOTEMP(matricola, cognome,nome)  
values('BBB3','Bianchi','Antonio');
```

31

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

32

Trigger: Esempio 4

```
insert into impiegato(matricola, cognome,nome)  
values('AAA1','Rossi','Mario');  
insert into impiegato(matricola, cognome,nome)  
values('AAA2','Rossi','Carlo');  
insert into impiegato(matricola, cognome,nome)  
values('AAA3','Bianchi','Antonio');  
insert into impiegato(matricola, cognome,nome)  
values('AAA4','Bianchi','Francesco');  
  
insert into impiegato(matricola, cognome,nome)  
select matricola, cognome,nome from impiegatotemp;  
  
insert into impiegato(matricola, cognome,nome)  
select matricola, cognome,nome from impiegatotemp  
where matricola like 'C%';
```

32

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

33

Trigger: Esempio 4

```
select *  
from storiainpiegato  
order by tempo desc;
```

utente	tempo	operazione	newrowsnumber	insertedrows
character varying (20)	timestamp without time zone	character varying (20)	integer	integer
postgres	2019-04-03 11:57:10.441389	INSERT	7	0
postgres	2019-04-03 11:57:10.441246	INSERT	7	3
postgres	2019-04-03 11:57:10.441102	INSERT	4	1
postgres	2019-04-03 11:57:10.441018	INSERT	3	1
postgres	2019-04-03 11:57:10.440926	INSERT	2	1
postgres	2019-04-03 11:57:10.440634	INSERT	1	1

34

Triggers su viste in PostgreSQL

- On views
 - triggers can be **also** defined to execute
 - instead of
INSERT, UPDATE, or DELETE operations.
 - In this case, it is the responsibility of the trigger's function to perform the necessary modifications to the underlying base tables and, where appropriate, return the modified row as it will appear in the view.

35

Trigger: Esempio 5

```
create table impiegato (  
  matricola char(4) primary key,  
  cognome varchar(20) not null,  
  nome varchar(20) not null,  
  stipendio integer not null,  
  dipartimento varchar(20) not null);
```

```
insert into impiegato(matricola, cognome,nome,dipartimento,stipendio)  
  values('aaa1', 'Rossi', 'Mauro', 'Amministrazione',52);  
insert into impiegato(matricola, cognome,nome,dipartimento,stipendio)  
  values('aaa2', 'Rossi', 'Giulio', 'Ragioneria',51);  
insert into impiegato(matricola, cognome,nome,dipartimento,stipendio)  
  values('aaa3', 'Esposito', 'Gennaro', 'Produzione',49);  
insert into impiegato(matricola, cognome,nome,dipartimento,stipendio)  
  values('aaa4', 'Paolino', 'Paperino', 'Amministrazione',40);
```

35

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

36

Trigger: Esempio 5

```
create or replace view ImpiegatiAmministrazione as  
SELECT matricola, Cognome, Nome, Stipendio  
FROM impiegato  
where Dipartimento = 'Amministrazione';
```

```
insert into ImpiegatiAmministrazione(matricola, cognome,nome,stipendio)  
values('aaa6', 'Russo', 'Paolo',48);
```

```
ERROR: null value in column "dipartimento" violates not-null constraint  
DETAIL: Failing row contains (aaa6, Russo, Paolo, 48, null).  
SQL state: 23502
```

36

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

37

Trigger: Esempio 5

```
CREATE OR REPLACE FUNCTION insert_ImpAmm() RETURNS trigger AS $$
BEGIN
    INSERT INTO impiegato (matricola,cognome,nome,stipendio, dipartimento)
    VALUES (NEW.matricola, NEW.cognome, NEW.nome, NEW.stipendio,
            'Amministrazione');

    RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER InsImpAmm
INSTEAD OF INSERT ON ImpiegatiAmministrazione
FOR EACH ROW
EXECUTE PROCEDURE insert_ImpAmm();
```

37

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

38

Trigger: Esempio 5

```
insert into ImpiegatiAmministrazione(matricola, cognome,nome,stipendio)
values('aaa6', 'Russo', 'Paolo',48);

select * from Impiegato;
```

matricola character (4)	cognome character varying (20)	nome character varying (20)	stipendio integer	dipartimento character varying (20)
aaa1	Rossi	Mauro	52	Amministrazione
aaa2	Rossi	Giulio	51	Ragioneria
aaa3	Esposito	Gennaro	49	Produzione
aaa4	Paolino	Paperino	40	Amministrazione
aaa6	Russo	Paolo	48	Amministrazione

38

AA 2011-2012, Basi di Dati, Prof. Antonio d'Acerno

La Programmazione

39