# Γλώσσα SQL
## ακαδ. έτος 2021-22


**Διδάσκων:**
**καθ. Γιάννης Θεοδωρίδης**

**+Διδάσκουσα:**
**Δρ. Μαυροπόδη Ρόζα**

**Εργαστηριακοί βοηθοί:**
**Γιάννης Κοντούλης**



Lectures on Databases: section I “Intro”, v. 2022.03
by
Data Science Lab. @ Univ. Piraeus (www.datastories.org)

# SQL - Functions and Procedures 1.1

- Είναι λειτουργικές μονάδες (program modules) προγραμμάτων βάσης δεδομένων —**procedures ή functions**— οι οποίες αποθηκεύονται και εκτελούνται από το DBMS στο εξυπηρετητή της βάσης δεδομένων.
- Είναι γνωστές, ιστορικά, ως **stored procedures**, αν και μπορεί να είναι procedures ή functions. (στην postgres είναι γνωστές και ως routines. π.χ. drop **routine** function ή procedure_name ; alter routine …. )
- Ο όρος που χρησιμοποιείται στο πρότυπο SQL είναι μόνιμες αποθηκευμένες λειτουργικές μονάδες (**persistent stored modules**), επειδή αυτά τα προγράμματα αποθηκεύονται μόνιμα (persistent) από το DBMS, παρόμοια με τη μόνιμη αποθήκευση δεδομένων.
- Επιτρέπουν την αποθήκευση του '**business logic**' της εφαρμογής να αποθηκεύεται μέσα στη βάση και όχι σε κάποιο κώδικα της εφαρμογής, π.χ. java.
- π.χ. σε πόσα μαθήματα μπορεί να εγγραφεί ένας φοιτητής, πόσα είναι τα ελάχιστα μαθήματα μπορεί να έχει ένας καθηγητής κλπ.

# SQL - Functions and Procedures 1.2

Είναι χρήσιμες στις ακόλουθες περιπτώσεις :

- Εάν ένα πρόγραμμα της βάσης δεδομένων απαιτείται από πολλές εφαρμογές, μπορεί να αποθηκευτεί στον εξυπηρετητή και να κληθεί από οποιαδήποτε από τις εφαρμογές (είτε αυτή είναι σε java, python, php). Αυτό μειώνει την επανάληψη της προσπάθειας και βελτιώνει τη μοντελοποίηση (modularity) του λογισμικού
- Η εκτέλεση ενός προγράμματος στον εξυπηρετητή μπορεί να μειώσει το κόστος μεταφοράς δεδομένων και επικοινωνίας μεταξύ του πελάτη και του εξυπηρετητή σε ορισμένες περιπτώσεις.
- Επεκτείνουν τη λειτουργία των όψεων (views) καθώς επιτρέπουν τη χρήση (και επιστροφή) πολυπλοκότερων τύπων δεδομένων. Επιπλέον, μπορούν να χρησιμοποιηθούν για τον έλεγχο περίπλοκων περιορισμών (constraints), triggers, assertions (πολύπλοκών constraints οι οποίοι αφορούν περισσότερους του ενός πίνακες).

# SQL- Functions and Procedures 1.3

Core SQL

```
create procedure procedure_name ([parameters])
[local declarations]
procedure_body ;

call procedure_name ([parameters]);

create function function_name ([parameters])
returns [return_type]
[local declarations]
function_body ;

select function_name ([parameters]);
select * from function_name ([parameters]);
select * from table1 where table1.col1 in function_name ([parameters]);
```

# SQL - Functions and Procedures 1.4

Οι βασικές διαφορές procedures/functions είναι:

- **create function / create procedure**
- Οι procedures δεν επιστρέφουν μια τιμή ως εκ τούτου, η CREATE PROCEDURE δεν διαθέτει το λεκτικό **RETURNS**. Ωστόσο, οι procedures μπορούν αντ' αυτού να επιστρέφουν δεδομένα μέσω των παραμέτρων εξόδου ( OUT/ INOUT).
- Ενώ μια συνάρτηση καλείται ως μέρος ενός ερωτήματος ή μιας εντολής DML, μια διαδικασία (procedure) καλείται μεμονωμένα χρησιμοποιώντας την εντολή **CALL**.
- **Οι συναρτήσεις καλούνται μέσα από τις εντολές SQL για να εκτελέσουν κάποια εργασία στις εγγραφές που ανακτώνται ή αποθηκεύονται, ενώ οι διαδικασίες στέκονται μόνες τους.**
- Μια διαδικασία (procedure) μπορεί να αποθηκεύσει (commit) ή να ανακαλέσει (roll-back) συναλλαγές κατά τη διάρκεια της εκτέλεσής της ( και στη συνέχεια να αρχίσει αυτόματα μια νέα συναλλαγή), εφόσον η εντολή CALL που την καλεί δεν αποτελεί μέρος του μπλοκ της συναλλαγής. Μια συνάρτηση δεν μπορεί να το κάνει αυτό. Δηλαδή σε μια συνάρτηση δεν επιτρέπονται οι εντολές **BEGIN, COMMIT, ROLLBACK, VACUUM or CREATE INDEX** κλπ. δεν ελέγχει τις συναλλαγές.

# SQL - Functions and Procedures 1.5

Η **PostgreSQL** παρέχει **τέσσερα είδη** συναρτήσεων/διαδικασιών:

- Συναρτήσεις/διαδικασίες γλώσσας ερωτήματος (**γραμμένες εξ` ολοκλήρου σε SQL**)
- Συναρτήσεις/διαδικασίες διαδικαστικής γλώσσας (**γραμμένες** σε διαδικαστική (**procedural programming language**) γλώσσα προγραμματισμού, για παράδειγμα, **PL/PGSQL** ή PL/TCL)
- Συναρτήσεις/διαδικασίες οι οποίες είναι εσωτερικές λειτουργίες (**internal εμφωλευμένος κώδικας σε C**)
- Συναρτήσεις/διαδικασίες **γλωσσών C**

# SQL- Functions and Procedures 1.6

**Εφαρμογή**

```
/*functions written in SQL*/
create function add_em(x integer, y integer) returns integer
as $$
select x + y;
$$ language sql;
```

```
select add_em(1, 2) as answer;
```

```
myunipi=# select add_em(1, 2) as answer;
+--------+
| answer |
+--------+
|      3 |
+--------+
(1 row)
```

```
select * from add_em(1, 2);
```

```
myunipi=# select * from add_em(1, 2);
+--------+
| add_em |
+--------+
|      3 |
+--------+
(1 row)
```

# SQL- Functions and Procedures 1.7

**Εφαρμογή postgres**

```
/*functions written in procedural language*/
create function somefunc(x integer, y text) returns integer
as $$
function body text
/*Εδώ οποιοσδήποτε κώδικας σε plpgsql */
$$ language plpgsql;

select somefunc(1, 'sometext') as answer;

select *  from somefunc(1, 'sometext');
```

# SQL- Functions and Procedures 1.8

**Εφαρμογή postgres**

```
/*functions written in internal functions*/
create function square_root(double precision)
returns double precision
as $$
dsqrt //κώδικας σε C
$$ language internal;

/*Internal functions αποτελούν εμφωλευμένο κώδικα σε C  */
```

# SQL- Functions and Procedures 1.9

**Εφαρμογή postgres**

```
/*functions written in C-language */
create function add_one_withC (integer) returns integer
as $$
directory/funcs
/*Η function σε C πρέπει να είναι precompiled*/
$$ language c;

select add_one_withC(1);


select * from add_one_withC(1);
```

# SQL- Functions and Procedures 1.10

**Core SQL**

/*διαγράφει το αντικείμενο και δημιουργεί ένα νέο*/
**drop procedure** procedure_name **(**[parameters]**) ;**
**drop function** function_name **(**[parameters]**);**
**drop routine** function_name **(**[parameters]**);**
**drop routine** procedure_name **(**[parameters]**);**

/*για την αλλαγή ιδιοτήτων της πχ. owner, schema*/
**alter function** function_name ….
**alter procedure** procedure_name **(**[parameters]**) …….**
**alter routine……**
/*για την αλλαγή του ορισμού/σώμα */
**create or replace function** function_name[ ] ……
**create or replace procedure** procedure_name[] ……

# SQL - Functions and Procedures 1.11

/*παράμετροι arguments*/

Οι παράμετροι μπορούν να είναι οποιοσδήποτε από τους τύπους τύπους δεδομένων της SQL, π.χ. varchar(10), integer, double precision, text.

Μπορεί να υπάρχουν συναρτήσεις με ίδια ονόματα αρκεί να έχουν διαφορετικό αριθμό και είδος παραμέτρων (IN) π.χ. οι:  add_one(integer), add_one(integer, text) είναι διαφορετικές.

Κάθε παράμετρος θα πρέπει επίσης να έχει έναν τρόπο λειτουργίας παραμέτρου, ο οποίος είναι ένας από τους **IN**,**OUT** ή **INOUT**.

Επιβάλλουν αυστηρή συμμόρφωση με τους τύπους: εάν μια παράμετρος είναι τύπου INTEGER, δεν μπορεί να κληθεί με όρισμα τύπου VARCHAR.

# SQL- Functions and Procedures 1.12

**postgres**
**π.χ.**

```
/*Τα ορίσματα μπορεί να είναι οποιαδήποτε τύπου δεδομένων της SQL */
/*Στις συναρτήσεις σε SQL η τελευταία εντολή πρέπει να είναι είτε select ή κάποιος τύπος delete, update με returning. */
/*τυπικός ορισμός παραμέτρων */
create function hello(name text)
returns text as $$
/*update … delete … select */
select 'hello, ' || name || '!';
$$ language sql;
```

```
myunipi=# select hello('rosa');
+--------------+
|    hello     |
+--------------+
| hello, rosa! |
+--------------+
(1 row)
```

```
/*Διαφορετικός ορισμός παραμέτρων. Δεν συνίσταται πάντα */
create function hello(text)
returns text as $$
select 'hello, ' || $1 || '!';
$$ language sql;
```

```
myunipi=# select hello('class 2022');
+--------------------+
|       hello        |
+--------------------+
| hello, class 2022! |
+--------------------+
(1 row)
```

# SQL- Functions and Procedures 1.13

**postgres**

**π.χ.**

```
/*ορισμός παραμέτρων τύπου in */
create function hello(in name text, in title text default 'mr.')
returns text as $$
select 'hello, ' || title || ' ' || name || '!';
$$ language sql;

select hello('rosa', 'Dr.');
select hello('rosa'); /*η default τιμή παραλήφθηκε */
select hello(title => 'Mrs.', name => 'Maria');
```

```
myunipi=# select hello('rosa', 'Dr.');
+-------------------+
|       hello       |
+-------------------+
| hello, Dr. rosa!  |
+-------------------+
(1 row)
```

```
myunipi=# select hello('rosa');
+-------------------+
|       hello       |
+-------------------+
| hello, mr. rosa!  |
+-------------------+
(1 row)
```

```
myunipi=# select hello(title => 'Mrs.', name => 'Maria');
+--------------------+
|       hello        |
+--------------------+
| hello, Mrs. Maria! |
+--------------------+
(1 row)
```

```
/*τί τιμές έχουν οι παράμετροι στο παρακάτω ; Γιατί;*/
select hello('alice', title => 'dr.');
```

# SQL- Functions and Procedures 1.14

**postgres**
**π.χ.**

```
/*ορισμός παραμέτρων τύπου in –null τιμές*/
create function hello(in name text, in title text default NULL)
returns text as $$
select 'hello, ' || title || ' ' || name || '!';
$$ language sql strict;

select hello('rosa', 'Dr.');

select hello('rosa', NULL);

select hello('rosa');
```

```
myunipi=# select hello('rosa', 'Dr.');
+------------------+
|      hello       |
+------------------+
| hello, Dr. rosa! |
+------------------+
(1 row)

myunipi=# select hello('rosa', NULL);
+-------+
| hello |
+-------+
|       |
+-------+
(1 row)

myunipi=# select hello('rosa');
+-------+
| hello |
+-------+
|       |
+-------+
(1 row)
```

# SQL- Functions and Procedures 1.15

**postgres**
**π.χ.**

```
/*ορισμός παραμέτρων τύπου out, return */
/*Υπάρχουν δύο τρόποι για να ορίσετε την τιμή επιστροφής:
- να χρησιμοποιήσετε RETURNS για να καθορίσετε τον τύπο δεδομένων επιστροφής
- να ορίσετε παραμέτρους εξόδου χρησιμοποιώντας τις INOUT ή OUT*/

create function hello(in name text, out text)
as $$
select 'hello, ' || ' ' || name || '!';
$$ language sql;
select hello('rosa');
create function hello(in name text)
returns text
as $$ select 'hello, ' || ' ' || name || '!';
$$ language sql;
```

```
myunipi=# select hello('rosa');
+---------------+
|     hello     |
+---------------+
| hello,  rosa! |
+---------------+
(1 row)
```

# SQL- Functions and Procedures 1.16

**postgres**
**π.χ.**

```
/*ορισμός παραμέτρων τύπου inout*/
/*Δεν αλλάζει η τιμή των παραμέτρων τύπου inout. Απλά χρησιμοποιείται και ως input και ως output*/
/*Οι procedures έχουν inout αλλά όχι out. Είναι ο μόνος τρόπος ώστε να επιστρέψουν κάποια τιμή. */

create function hello(inout name text)
as $$
select 'hello, ' || ' ' || name || '!';
$$ language sql;
select hello('rosa');
```

```
myunipi=# select hello('rosa');
+---------------+
|     hello     |
+---------------+
| hello,  rosa! |
+---------------+
(1 row)
```

# SQL- Functions and Procedures 1.17

**postgres**
**π.χ.**

```
/*ορισμός παραμέτρων τύπου out, return */
/*Το return επιστρέφει μόνο μια τιμή ή…*/

create function hello(in name text, out greeting text, out clock time)
as $$
select 'hello, ' || ' ' || name || '!', current_time ;
$$ language sql;
select hello('rosa');
```

```
myunipi=# select hello('rosa');
+-----------------------------------+
|                hello              |
+-----------------------------------+
| ("hello,  rosa!",09:59:45.684282) |
+-----------------------------------+
(1 row)
```

# SQL- Functions and Procedures 1.18

**postgres**
**π.χ.**

```
/*ορισμός παραμέτρων τύπου out, return */
/*Το return επιστρέφει μόνο μια τιμή ή μια ομάδα γραμμών*/
/*Ο ορισμός του returns table και του επιστρεφόμενου select πρέπει να ταυτίζονται σε αριθμό και είδος*/
create function myinstructor (dept_name varchar(20))
returns table (
ID varchar (5),
name varchar (20),
dept_name varchar (20),
salary numeric (8,2)) as $$
select ID, name, dept_name, salary
from instructor
where
instructor.dept_name =
myinstructor.dept_name
$$ language SQL;
```

```
myunipi=# select myinstructor('Comp. Sci.');
+---------------------------------------+
|               myinstructor            |
+---------------------------------------+
| (10101,Srinivasan,"Comp. Sci.",65000.00) |
| (45565,Katz,"Comp. Sci.",75000.00)    |
| (83821,Brandt,"Comp. Sci.",92000.00)  |
+---------------------------------------+
(3 rows)
```

```
myunipi=# select * from myinstructor('Comp. Sci.');
+-------+------------+-----------+----------+
|  id   |    name    | dept_name |  salary  |
+-------+------------+-----------+----------+
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 45565 | Katz       | Comp. Sci. | 75000.00 |
| 83821 | Brandt     | Comp. Sci. | 92000.00 |
+-------+------------+-----------+----------+
(3 rows)
```

# SQL- Functions and Procedures 1.19

**postgres**
**π.χ.**

```
/*ορισμός παραμέτρων τύπου out, return */
/*Το return επιστρέφει μόνο μια τιμή ή μια ομάδα γραμμών*/
/*Ο ορισμός του returns table και του επιστρεφόμενου select πρέπει να ταυτίζονται σε αριθμό και είδος*/
create function myinstructor (dept_name varchar(20))
returns setof instructor as $$
select ID, name, dept_name, salary
from instructor
where
instructor.dept_name =
myinstructor.dept_name
$$ language SQL;
```

```
myunipi=# select myinstructor('Comp. Sci.');
+---------------------------------------+
|               myinstructor            |
+---------------------------------------+
| (10101,Srinivasan,"Comp. Sci.",65000.00) |
| (45565,Katz,"Comp. Sci.",75000.00)    |
| (83821,Brandt,"Comp. Sci.",92000.00)  |
+---------------------------------------+
(3 rows)


myunipi=# select * from myinstructor('Comp. Sci.');
+-------+------------+-----------+----------+
|  id   |    name    | dept_name |  salary  |
+-------+------------+-----------+----------+
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 45565 | Katz       | Comp. Sci. | 75000.00 |
| 83821 | Brandt     | Comp. Sci. | 92000.00 |
+-------+------------+-----------+----------+
(3 rows)
```

# SQL- Functions and Procedures 1.20

/*functions written in procedural language
create function με language plpgsql; */

Μια συνάρτηση/procedure η οποία είναι ορισμένη σε διαδικαστική γλώσσα (pl/pgsql για την postgres, PL/SQL για την Oracle, TransactSQL για Microsoft SQL Server)

- μπορεί να χρησιμοποιηθεί για τη δημιουργία συναρτήσεων, procedures και triggers,
- προσθέτει δομές ελέγχου στη γλώσσα SQL (loop, if κλπ),
- μπορεί να εκτελέσει πολύπλοκους υπολογισμούς,
- κληρονομεί όλους τους τύπους, τις συναρτήσεις, τις procedures και τους τελεστές που ορίζονται από τον χρήστη,
- μπορεί να οριστεί ότι είναι αξιόπιστος από τον εξυπηρετητή

# SQL- Functions and Procedures 1.21

/*functions written in procedural language*/

Εμφανίστηκε για πρώτη φορά στην έκδοση 6.4 το 1998 εγκαθίσταται ως προεπιλογή από την έκδοση 9.0 της postgres

Βασίζεται στη Oracle PL/SQL η οποία βασίζεται στην Ada, η οποία μοιάζει με την pascal σε αντίθεση με τις γλώσσες οι οποίες βασίζονται στη C.

language internal ή language C παρέχουν τη δυνατότητα για σύνδεση με συναρτήσεις γραμμένες σε C

# SQL- Functions and Procedures 1.22

**postgres**
**π.χ.**

```
/*functions written in procedural language*/
create function somefunc(integer, text) returns integer as
'function body text'
language plpgsql;
```

η περιοχή function body text είναι της μορφής

```
[ <<label>> ]                /*μη απαραίτητο.  Ετικέτα  */
[ DECLARE                    /*μη απαραίτητο. Περιοχή δηλώσεων
  declarations ]             μεταβλητών κλπ */
BEGIN                        /*Απαραίτητο. κυρίως σώμα.  BEGIN
statements                   περιοχή κώδικα */
END [ label ];               /*Ολοκλήρωση κώδικα END ; */
                             /*Η μη απαραίτητή ετικέτα πρέπει να είναι ίδια με ανωτέρω*/
```

# SQL- Functions and Procedures 1.23

**postgres**
**π.χ.**

```
/*functions written in procedural language*/
/*Ανώνυμο μπλόκ κώδικα. Χωρίς παραμέτρους΄, χωρίς όνομα, χωρίς αποθήκευση*/
do $$
declare

foo text;

bar text := 'world'; /*αντί του := μπορεί να χρησιμοποιηθεί το = ή default*/
begin
foo := 'hello';
raise notice '%, %!', foo, bar; /*προβολή μηνύματος*/

end;
$$;
```

```
myunipi=# do $$
myunipi$# declare
myunipi$# foo text;
myunipi$# bar text := 'world';
myunipi$# begin
myunipi$# foo := 'hello';
myunipi$# raise notice '%, %!', foo, bar;
myunipi$# end;
myunipi$# $$;
NOTICE:  hello, world!
DO
```

# SQL- Functions and Procedures 1.24

**postgres**
**π.χ.**

```
/*functions written in procedural language*/
create function sqr_out(in a numeric, out retval numeric)
as $$
begin
retval := a * a;
end;
$$ language plpgsql;
```

```
myunipi=# select * from sqr_out(10);
+--------+
| retval |
+--------+
|    100 |
+--------+
(1 row)
```

```
/*ομοίως */
create function sqr_out(inout a numeric)
as $$
begin
a := a * a;
end;
$$ language plpgsql;
```

```
myunipi=# select * from sqr_out(100);
+-------+
|   a   |
+-------+
| 10000 |
+-------+
(1 row)
```

# SQL- Functions and Procedures 1.25

**postgres**
**π.χ.**

/*functions written in procedural language
εντολές υπό συνθήκη.
Μια συνθήκη μπορεί να είναι true, false και το NULL δεν χαρακτηρίζεται */

**IF** condition **THEN**
-- operators
**ELSIF** condition **THEN**
-- operators
**ELSIF** condition **THEN**
-- operators
**ELSE**
-- operators
**END IF;**

**CASE**
**WHEN** condition **THEN**
-- operators
**WHEN** condition **THEN**
-- operators
**ELSE**
-- operators
**END CASE;**

# SQL- Functions and Procedures 1.26

**postgres**
**π.χ.**

```
/*functions written in procedural language - εντολές υπό συνθήκη. */

create function decode_isbn( in isbn text, out country text,
out publisher_and_book text, out check_digit integer ) as $$
declare
country_len integer;
begin
if left(isbn,1)::integer in (0,1,2,3,4,5,7) then country_len := 1;
elsif left(isbn,2)::integer between 80 and 94 then country_len := 2;
elsif left(isbn,3)::integer between 600 and 649 then country_len := 3;
elsif left(isbn,3)::integer between 950 and 993 then country_len := 3;
elsif left(isbn,4)::integer between 9940 and 9989 then country_len := 4;
else country_len := 5;
end if;
country := left(isbn, country_len);
publisher_and_book := substr(isbn, country_len+1, 12);
check_digit := right(isbn, 1);
end;
$$ language plpgsql;
```

```
myunipi=# SELECT * FROM decode_isbn('1484268849');
+---------+--------------------+-------------+
| country | publisher_and_book | check_digit |
+---------+--------------------+-------------+
| 1       | 484268849          |           9 |
+---------+--------------------+-------------+
```

# SQL- Functions and Procedures 1.27

**postgres**
**π.χ.**

```
/*functions written in procedural language - εντολές υπό συνθήκη. */
do $$
declare
country text := (decode_isbn('1484268849')).country;
begin
case
when country in ('0','1') then raise notice '% — english-speaking area', country;
when country = '7' then raise notice '% — russia', country;
when country = '88' then raise notice '% — italy', country;
else raise notice '% — other', country;
end case;
end;
$$;
```

```
NOTICE:  1 - english-speaking area
DO
```

# SQL- Functions and Procedures 1.28

**postgres**
**π.χ.**

```
/*functions written in procedural language – Βρόχοι επανάληψης.
*/

/*ατέρμονος βρόχος */
loop
-- operators
end loop;
/*μη ατέρμονος βρόχος */
loop
-- operators
exit when condition;
end loop;
```

# SQL- Functions and Procedures 1.29

**postgres**
**π.χ.**

```
/*functions written in procedural language – Βρόχοι επανάληψης. */


    /*βρόχος for
    η μεταβλητή name υπάρχει και έχει ζωή μόνον μέσα στο βρόχο*/
    for name in bottom .. top by increment
    loop
      -- operators
    end loop;
    /*αντίστροφη επανάληψη*/
    for name in reverse bottom .. top by increment
    loop
      -- operators
    end loop;
```

# SQL- Functions and Procedures 1.30

**postgres**
**π.χ.**

/*functions written in procedural language – Βρόχοι επανάληψης. for */

```
create function reverse_for (line text) returns text
as $$
declare
  line_length constant int := length(line);
  retval text := '';
begin
 for i in 1 .. line_length
  loop
     retval := substr(line, i, 1) || retval; /*to || κάνει string concatenation*/
  end loop;
 return retval;
end;
$$ language plpgsql strict;
```

```
for my_value   in
select dept_name from department ;
```

```
myunipi=# select reverse_for('abcdefgh');
+-------------+
| reverse_for |
+-------------+
| hgfedcba    |
+-------------+
(1 row)
```

# SQL- Functions and Procedures 1.31

postgres
π.χ.

/*functions written in procedural language – Βρόχοι επανάληψης. While */

```
create function reverse_while (line text) returns
text
as $$
declare
      line_length constant int := length(line);
      i int := 1;
      retval text := '';
begin
      while i <= line_length
      loop
            retval := substr(line, i, 1) || retval;
            i := i + 1;
      end loop;
       return retval;
end;
$$ language plpgsql  strict;
```

```
while condition
loop
-- operators
end loop;
```

```
myunipi=# select reverse_while('abcdefgh');
+---------------+
| reverse_while |
+---------------+
| hgfedcba      |
+---------------+
(1 row)
```

# SQL- Functions and Procedures 1.32

postgres
π.χ.

/*functions written in procedural language – Βρόχοι επανάληψης. for */

```
do $$
declare
s integer := 0;
begin
    for i in 1 .. 100
    loop
        s := s + i;
        continue when mod(i, 10) != 0;
        raise notice 'i = %, s = %', i, s;
    end loop;
end;
$$;
```

/*παράληψη ενός κύκλου εκτέλεσης κώδικα όταν ισχύει η συνθήκη… */

# SQL - Triggers 1.1

/*τι είναι */

- ✓ Ένα trigger είναι μια διαδικασία/συνάρτηση η οποία καλείται αυτόματα από το ΣΔΒΔ ως αντίδραση/απόκριση σε καθορισμένες αλλαγές στη βάση δεδομένων
- ✓ Συνήθως τους triggers τους ορίζει ο διαχειριστής της βάσης ή όποιος χρήστης έχει τα κατάλληλα δικαιώματα και στην εκτέλεση της διαδικασίας και στον πίνακα που εφαρμόζεται.
- ✓ active database

- ✓ Μπορούν να χρησιμοποιηθούν για την υλοποίηση ορισμένων περιορισμών ακεραιότητας οι οποίοι δεν μπορούν να προσδιοριστούν με τη χρήση μηχανισμών της SQL (π.χ. foreign key on delete/update SET NULL/CASCADE/SET DEFAULT deferred)
- ✓ Είναι χρήσιμοι μηχανισμοί για την ειδοποίηση ανθρώπων ή για την αυτόματη εκκίνηση ορισμένων εργασιών (και ειδοποιήσεων) όταν πληρούνται ορισμένες συνθήκες.

# SQL - Triggers 1.2

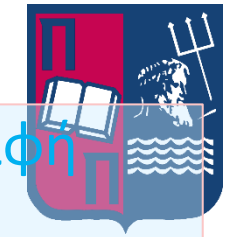
/*παράδειγμα χρήσης */

- ✓ Ως παράδειγμα, θα μπορούσαμε να σχεδιάσουμε ένα trigger ο οποίος, κάθε φορά που εισάγεται μια πλειάδα στη σχέση takes, ενημερώνει την πλειάδα στη σχέση student για τον φοιτητή που παρακολουθεί το μάθημα προσθέτοντας τον αριθμό των πιστωτικών μονάδων για το μάθημα στο σύνολο των πιστωτικών μονάδων του φοιτητή.

- ✓ Ως άλλο παράδειγμα, ας υποθέσουμε ότι μια αποθήκη επιθυμεί να διατηρεί ένα ελάχιστο απόθεμα για κάθε είδος- όταν το επίπεδο αποθέματος ενός είδους πέσει κάτω από το ελάχιστο επίπεδο, μπορεί να δοθεί αυτόματα μια παραγγελία.
  Κάθε φορά που εξάγεται ή εισάγεται ένα τεμάχιο, ο trigger συγκρίνει το τρέχον επίπεδο αποθέματος με το ελάχιστο επίπεδο, και αν το επίπεδο είναι στο ελάχιστο ή κάτω από το ελάχιστο, δημιουργείται μια νέα παραγγελία.

  Οι triggers δεν μπορούν, συνήθως, να εκτελούν ενημερώσεις εκτός της βάσης δεδομένων, και ως εκ τούτου, στο παράδειγμα αναπλήρωσης αποθεμάτων, δεν μπορούμε να χρησιμοποιήσουμε ένα trigger για να τοποθετήσουμε μια παραγγελία σε εξωτερικό σύστημα, εκτός του ΣΔΒΔ. Αντ' αυτού, προσθέτουμε μια παραγγελία σε μια σχέση/πίνακα ο οποίος περιέχει παραγγελίες προς εκτέλεση. Οι ειδοποιήσεις χρειάζονται υλοποίηση σε κώδικα στη host language.

# SQL- Triggers 1.3



postgres
π.χ.

/*Έστω ότι κάθε φορά που θέλουμε εισάγουμε/ενημερώνουμε μια εγγραφή στον πίνακα instructor θέλουμε να ελέγξουμε ότι η τιμή στο salary είναι μικρότερη από όλες τις υπάρχουσες για το ίδιο dept_name. */

**event-condition-action (ECA)**

```
create trigger salary_violation
before insert or update of salary, dept_name
on instructor
for each row
when (new.dept_name <> 'statistics')
execute function salary_violation();
```

**όνομα**: ενώ εκτελείται αυτόματα χρειάζεται ένα όνομα για drop και deactivate

**event**: τί θα πρέπει να συμβεί ώστε να εκτελεστεί

που θα λειτουργήσει. σε ποια σχέση πάντα σε κάποιο **table** ή **view**

ανά γραμμή/ ανά ερώτημα
for each statement
**row level / statement level**

Ποιες οι **συνθήκες** θα πρέπει να ελεγχθούν πριν ξεκινήσει η εκτέλεση

**action**
τελικά τί θα εκτελεστεί
**trigger function**
execute procedure (deprecated)

# SQL - Triggers 1.4

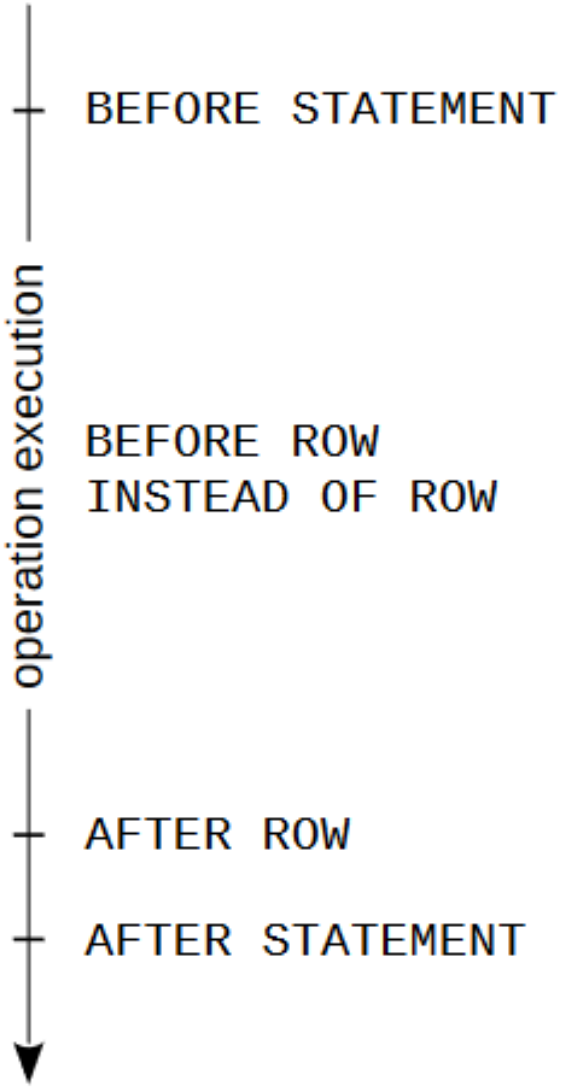
- **DML statement triggers.** Συνηθέστεροι από όλους
- Είναι δυνατόν να οριστούν triggers DDL statements (**event triggers**) π.χ. CREATE, ALTER, DROP, SECURITY LABEL, COMMENT, GRANT or REVOKE οπότε και εφαρμόζονται σε όλη τη βάση και όχι μόνο σε πίνακες/views
- Οι triggers θα μπορούσαν επίσης να είναι χρονικά γεγονότα ή άλλα είδη εξωτερικών γεγονότων. Ένα παράδειγμα θα μπορούσε να είναι ένα χρονικό γεγονός που ορίζεται ως ένας περιοδικός χρόνος, όπως: ενεργοποίηση αυτού του κανόνα κάθε μέρα στις 5:30 π.μ. οπότε λειτουργούν ως daemons.
- ή π.χ όταν ένας χρήστης (εφαρμογή) συνδέεται στη βάση δεδομένων (δηλαδή ανοίγει μια σύνδεση) (**logon triggers**), το σύστημα κλείνει (shutdown), ή γίνονται αλλαγές στις ρυθμίσεις του συστήματος.

| When | Event | Row-level | Statement-level |
|---|---|---|---|
| BEFORE | INSERT/UPDATE/DELETE | Tables | Tables and views |
| | TRUNCATE | — | Tables |
| AFTER | INSERT/UPDATE/DELETE | Tables | Tables and views |
| | TRUNCATE | — | Tables |
| INSTEAD OF | INSERT/UPDATE/DELETE | Views | — |
| | TRUNCATE | — | — |

# SQL - Triggers 1.5

- Με ποια σειρά γίνεται η εκτέλεση

Εάν έχουν οριστεί περισσότερα από ένα trigger για το **ίδιο συμβάν** στην ίδια **σχέση**, τα triggers θα **ενεργοποιηθούν** με **αλφαβητική σειρά** ανάλογα με το όνομα του trigger.
Στην περίπτωση των triggers BEFORE και INSTEAD OF, η πιθανώς τροποποιημένη γραμμή που επιστρέφεται από κάθε trigger γίνεται η είσοδος στο επόμενο trigger.
Εάν οποιοδήποτε trigger BEFORE ή INSTEAD OF επιστρέψει NULL, η λειτουργία εγκαταλείπεται **για αυτή τη γραμμή** και τα επόμενα triggers δεν πυροδοτούνται (για αυτή τη γραμμή).

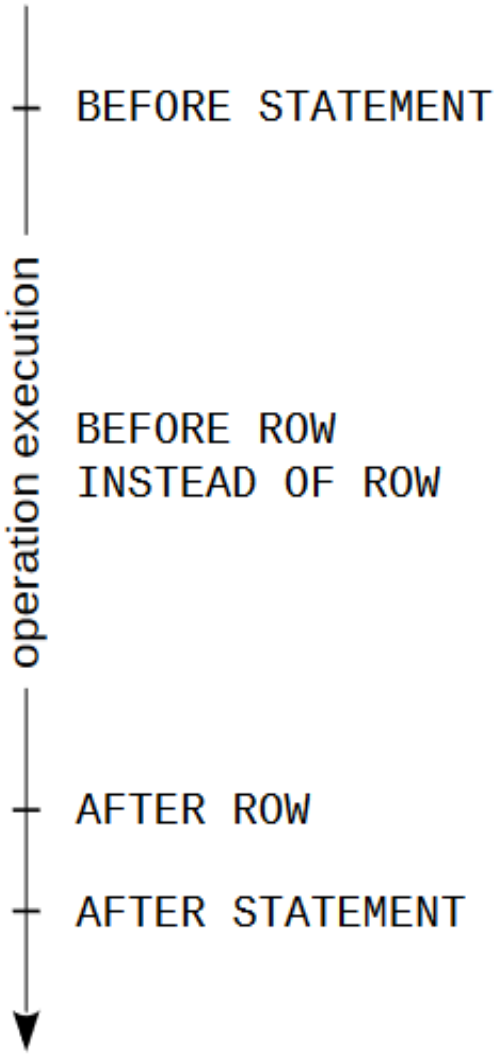


# SQL - Triggers 1.6

- Με ποια σειρά γίνεται η εκτέλεση

- **row-level BEFORE** triggers: χρησιμοποιούνται για τον έλεγχο ή την τροποποίηση των δεδομένων που θα εισαχθούν ή θα ενημερωθούν. Για παράδειγμα, ένα BEFORE trigger μπορεί να χρησιμοποιηθεί για την εισαγωγή της τρέχουσας ώρας σε μια στήλη timestamp ή για τον έλεγχο ότι δύο στοιχεία της γραμμής είναι συνεπή.
- **row-level AFTER** triggers: χρησιμοποιούνται για τη διάδοση των ενημερώσεων σε άλλους πίνακες ή για τη διενέργεια ελέγχων συνέπειας έναντι άλλων πινάκων.

Βλέπει την τελική τιμή της γραμμής, ενώ ένα BEFORE trigger δεν μπορεί- ενδέχεται να υπάρχουν άλλα BEFORE triggers που πυροδοτούνται μετά από αυτό.

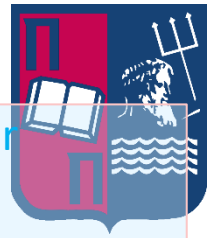


# SQL - Triggers 1.7

➢ trigger functions

- Είναι μια κανονική συνάρτηση που ακολουθεί ορισμένες συμβάσεις:
  - - Μπορεί να γραφτεί σε οποιαδήποτε γλώσσα εκτός από την καθαρή SQL.
  - - Δεν πρέπει να έχει παραμέτρους π.χ. salary_violation (a integer, b text)
  - - Η τιμή επιστροφής της είναι του τύπου trigger (στην πραγματικότητα είναι ψευδότυπος).
- Η συνάρτηση trigger εκτελείται στην ίδια συναλλαγή με την κύρια λειτουργία. Συνεπώς, εάν μια συνάρτηση ενεργοποίησης καταλήξει σε σφάλμα, διακόπτεται ολόκληρη η συναλλαγή.
- Μπορεί να χρησιμοποιηθεί σε διάφορά  triggers
- Πρέπει να είναι ορισμένη πριν τη δημιουργία του trigger που τη χρησιμοποιεί

# SQL- Triggers 1.8



**postgres**
**π.χ.**

```
/*Έστω ότι κάθε φορά που θέλουμε εισάγουμε/ενημερώνουμε μια εγγραφή στον πίνακα instructor
θέλουμε να ελέγξουμε ότι η τιμή στο salary είναι μικρότερη από όλες τις υπάρχουσες για το ίδιο
dept_name. */
create or replace function salary_violation ()
returns trigger
as $$
declare max_salary integer := (select max(salary) from instructor where dept_name =
new.dept_name);
begin
if new.salary>max_salary then
        begin
                raise notice '% is bigger then everyone else in %!', new.salary, new.dept_name
                        using hint = 'Give him/her less money';
                return null;
        end ;
else raise notice '% is NOT bigger then everyone else in %!', new.salary, new.dept_name;
end if;
return new;
end ;
$$ language plpgsql        ;
```

```
myunipi=# insert into instructor values ('rosa1', 'Rosa Mavropodi', 'Comp. Sci.', '93000');
NOTICE:  93000.00 is bigger then everyone else in Comp. Sci.!
HINT:  Give him/her less money
INSERT 0 0
```

# SQL- Triggers 1.9

**postgres π.χ.**

- trigger διαφορετικοί τύποι, τι επιστρέφουν ανάλογα τον trigger

- Before E* statement-level: Η τιμή επιστροφής της συνάρτησης αγνοείται, μπορεί απλώς να επιστρέψει NULL. Εάν υπάρχει σφάλμα, η λειτουργία ακυρώνεται, προβάλλοντας το αντίστοιχο error.
- Before E* row-level και instead of row:
  - return null: ακυρώνει την επεξεργασία της τρέχουσας γραμμής.
  - return new για insert/update και return old για delete: επιτυχής εκτέλεση του κώδικα.
- after E* row-level: Η επιστρεφόμενη τιμή αγνοείται (επειδή η λειτουργία έχει ήδη ολοκληρωθεί)
- after E* statement-level: Η επιστρεφόμενη τιμή αγνοείται (επειδή η λειτουργία έχει ήδη ολοκληρωθεί)

- * το E(vent) μπορεί να είναι insert/update/delete

# SQL- Triggers 1.10

**postgres**
**π.χ.**

- γνωρίζει πολλά για τον εαυτό του

```
CREATE OR REPLACE FUNCTION trig_demo()
RETURNS trigger AS
$$
BEGIN
  RAISE NOTICE 'TG_NAME: %', TG_NAME;
  RAISE NOTICE 'TG_RELNAME: %', TG_RELNAME;
  RAISE NOTICE 'TG_TABLE_SCHEMA: %', TG_TABLE_SCHEMA;
  RAISE NOTICE 'TG_TABLE_NAME: %', TG_TABLE_NAME;
  RAISE NOTICE 'TG_WHEN: %', TG_WHEN;
  RAISE NOTICE 'TG_LEVEL: %', TG_LEVEL;
  RAISE NOTICE 'TG_OP: %', TG_OP;
  RAISE NOTICE 'TG_NARGS: %', TG_NARGS;
-- RAISE NOTICE 'TG_ARGV: %', TG_NAME;
 RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER sensor_trig
BEFORE INSERT ON instructor
FOR EACH ROW
EXECUTE PROCEDURE trig_demo();
```

```
myunipi=# insert into instructor values ('rosa1', 'Rosa Mavropodi', 'Comp. Sci.', '90000');
NOTICE:  90000.00 is NOT bigger then everyone else in Comp. Sci.!
NOTICE:  TG_NAME: sensor_trig
NOTICE:  TG_RELNAME: instructor
NOTICE:  TG_TABLE_SCHEMA: public
NOTICE:  TG_TABLE_NAME: instructor
NOTICE:  TG_WHEN: BEFORE
NOTICE:  TG_LEVEL: ROW
NOTICE:  TG_OP: INSERT
NOTICE:  TG_NARGS: 0
INSERT 0 1
```

# SQL- Triggers 1.11

**postgres π.χ.**

- transition table και for each statement

```
CREATE OR REPLACE FUNCTION transition_trigger()
RETURNS TRIGGER AS $$
DECLARE
 v_record record;
BEGIN
 IF (TG_OP = 'INSERT') THEN
        RAISE NOTICE 'new data: ';
        FOR v_record IN SELECT * FROM new_table
        LOOP
           RAISE NOTICE '%', v_record;
        END LOOP;
 ELSE
        RAISE NOTICE 'old data: ';
        FOR v_record IN SELECT * FROM old_table
         LOOP
           RAISE NOTICE '%', v_record;
         END LOOP;
 END IF; RETURN NULL; -- result is ignored since this is an AFTER trigger
END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER transition_test_trigger_ins
AFTER INSERT ON instructor
REFERENCING NEW TABLE AS new_table
FOR EACH STATEMENT EXECUTE function
transition_trigger();
```

```
CREATE TRIGGER transition_test_trigger_del
AFTER DELETE ON instructor
REFERENCING OLD TABLE AS old_table
FOR EACH STATEMENT EXECUTE PROCEDURE
transition_trigger();
```

# SQL- Triggers 1.12

- transition table δεν χρησιμοποιούνται με before triggers, μόνον με after

```
CREATE TRIGGER transition_test_trigger_ins
AFTER INSERT ON instructor
REFERENCING NEW TABLE AS new_table
FOR EACH STATEMENT EXECUTE function
transition_trigger();
```

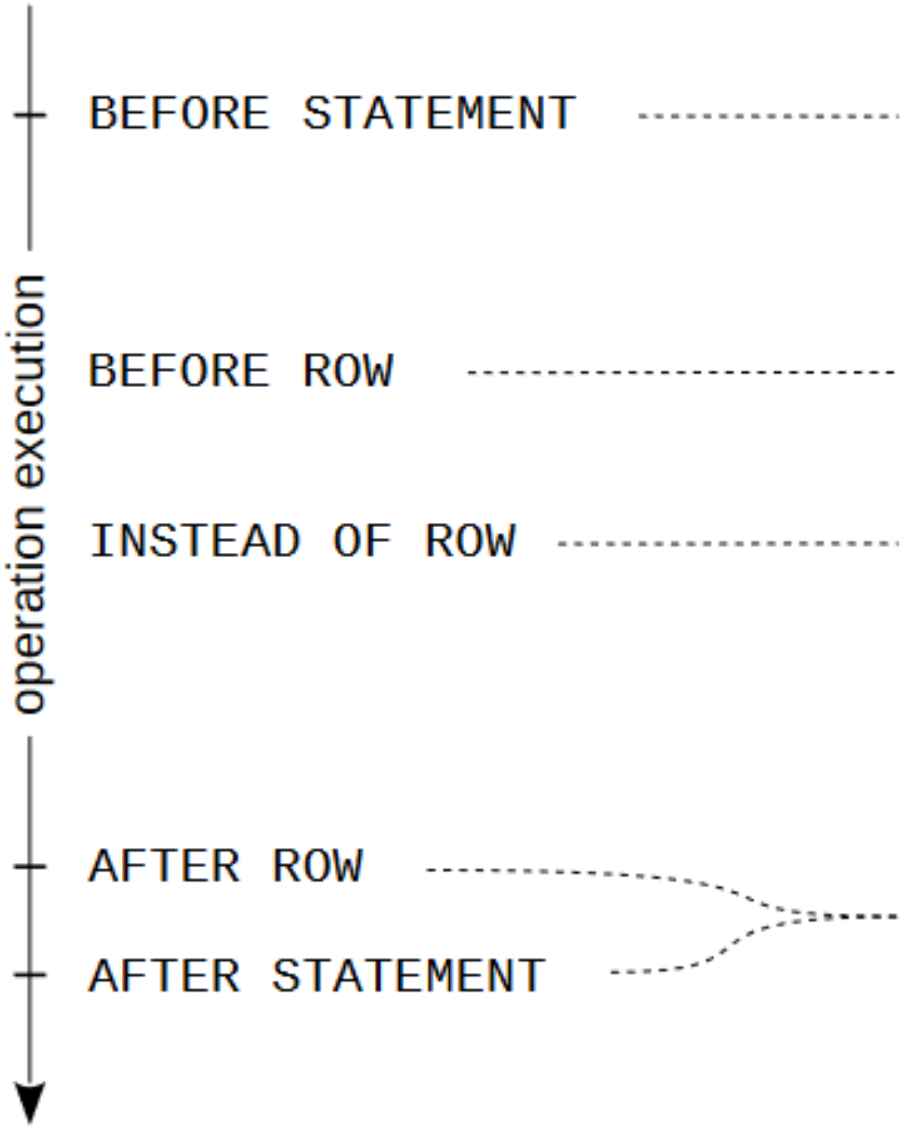
**postgres**
**π.χ.**

```
myunipi=# insert into instructor values ('rosa1', 'Rosa Mavropodi', 'Comp. Sci.', '93000');
NOTICE:  new data:
NOTICE:  (rosa1,"Rosa Mavropodi","Comp. Sci.",93000.00)
INSERT 0 1
```

```
CREATE TRIGGER transition_test_trigger_del
AFTER DELETE ON instructor
REFERENCING OLD TABLE AS old_table
FOR EACH STATEMENT EXECUTE PROCEDURE
transition_trigger();
```

```
myunipi=# delete from instructor where id='rosa1';
NOTICE:  old data:
NOTICE:  (rosa1,"Rosa Mavropodi","Comp. Sci.",93000.00)
DELETE 1
```

# SQL- Triggers 1.13




Έλεγχοι εφαρμογής της λειτουργίας insert/update/delete

Έλεγχοι ορθότητας, τροποποιήσεις γραμμών

μετάδοση της αλλαγή στους πίνακες για τις όψεις/views

Έλεγχοι συνέπειας,
συμπεριλαμβανομένων των ελέγχων σε επίπεδο πίνακα,
καταγραφή λειτουργιών για σκοπούς ελέγχου (logging, statistics),
cascade ενημερώσεις πινάκων

# SQL- Triggers 1.14

**postgres**
**π.χ.**

- Drop alter

**DROP TRIGGER** [ **IF EXISTS** ] name **ON** table_name [ **CASCADE** | **RESTRICT** ]

**drop trigger** transition_test_trigger_ins **on** instructor;

**ALTER TABLE** table_name
**DISABLE|ENABLE TRIGGER** trigger_name | ALL

**alter table** instructor
**disable trigger** transition_test_trigger_ins ;

**alter table** instructor
**enable trigger** all;

# SQL- Triggers 1.15

- triggers συνηθέστερες εφαρμογές τους

- ✓ Καταγραφή logging και εξαγωγή στατιστικών στοιχείων
- ✓ Ειδοποιήσεις σε συνεργασία με τη host γλώσσα προγραμματισμού
- ✓ Ενσωμάτωση της λογικής (business) της εφαρμογής στη βάση
- ✓ Επιβολή σύνθετων εξουσιοδοτήσεων ασφαλείας
- ✓ Αυτόματη δημιουργία και καταγραφή δεδομένων. Συνδυασμός τιμών από διάφορες στήλες και αόρατη αποθήκευσή τους.
- ✓
- ✓ Εφαρμογή πολύπλοκων περιορισμών ξένων κλειδιών οι οποίοι δεν μπορούν να ικανοποιηθούν από τους έτοιμους μηχανισμούς της SQL
- ✓ Replication της βάσης, αν και αυτό τείνει να εγκαταλειφθεί

# SQL- Triggers 1.16

- triggers λόγοι αποφυγής

✓ Έχουν την τάση να κρύβουν τη λογική της εφαρμογής καθιστώντας τη συντήρησή της εξαιρετικά περίπλοκη. Είναι προτιμότερο να εκτελέσουμε ένα on delete cascade παρά να χρησιμοποιηθεί trigger για αυτό το σκοπό. Είναι δόκιμη η διατήρηση καλής έγγραφης τεκμηρίωσης (manual)

✓ Πιθανότητα ατέρμονης επανάληψης.

✓ Η αλυσιδωτή/σειριακή εκτέλεση πολλών triggers για ένα γεγονός

✓ Κίνδυνος παραβίασης περιορισμών ακεραιότητας. (Ένας περιορισμός εμποδίζει τα δεδομένα να γίνουν ασυνέπεια από οποιοδήποτε είδος δήλωσης, ενώ μέσω triggers μόνο insert/delete/update)

*Οι triggers πρέπει να είναι λίγοι σε αριθμό και να περιέχουν λίγες γραμμές απλού κώδικα.*

# SQL- Triggers 1.17

oracle

π.χ.

```
CREATE TRIGGER reorder
AFTER UPDATE OF parts_on_hand ON inventory
FOR EACH ROW
WHEN(new.parts_on_hand < new.reorder_point)
DECLARE
  x NUMBER;                    /*Trigger και trigger function σε μια δήλωση*/
BEGIN
  SELECT COUNT(*) INTO x
    FROM pending_orders
    WHERE part_no = :new.part_no;
  IF x = 0  THEN
    INSERT INTO pending_orders
      VALUES (:new.part_no, :new.reorder_quantity,
              sysdate);
  END IF;
END;
```