ΕΝΑΥΣΜΑΤΑ ΣΤΗΝ POSTGRES

ΓΕΝΙΚΑ

Μία διαδικασία την οποία δηλώνει ο Διαχειριστής της Βάσης Δεδομένων και η οποία ενεργοποιείται αυτόματα από το ΣΔΒΔ κάθε φορά που συμβαίνουν μεταβολές ορισμένου τύπου στα δεδομένα ονομάζεται *έναυσμα (trigger)*.

Μία Βάση Δεδομένων η δήλωση της οποίας συνδυάζεται με δηλώσεις εναυσμάτων ονομάζεται **ενεργή βάση δεδομένων**.

Στην περιγραφή ενός εναύσματος υπάρχουν τρία διακριτά μέρη:

- **Γεγονός**: μία μεταβολή στη βάση δεδομένων η οποία ενεργοποιεί το έναυσμα.
- **Συνθήκη**: ένα αίτημα ή ένας έλεγχος που εκτελείται με την ενεργοποίηση του εναύσματος.
- **Ενέργεια**: μία διαδικασία η οποία εκτελείται όταν ισχύει η συνθήκη και ενεργοποιείται το έναυσμα.

Το όνομα ενός εναύσματος στο σχήμα της βάσης θα πρέπει να είναι μοναδικό. Να μην υπάρχει άλλο έναυσμα με το ίδιο όνομα.

Τρία είναι τα είδη των εναυσμάτων:

- DML φράσεις (DELETE, INSERT, UPDATE)
- DDL φράσεις (CREATE, ALTER, DROP)
- Λειτουργίες Βάσης Δεδομένων (LOGON, LOGOFF)

Ένα έναυσμα μπορεί να εκτελείται πριν ή μετά την ολοκλήρωση της ενέργειας που το προκάλεσε. Ανάλογα με την περίπτωση θα πρέπει να δηλωθεί στο έναυσμα αν θα εκτελεστεί πριν (before) ή μετά (after):

- Before, εκτελείται πριν συμβούν μεταβολές στη βάση.
- After, εκτελείται αφού συμβούν μεταβολές στη βάση.

Όταν συντάσσεται ένα έναυσμα θα πρέπει να ακολουθηθούν οι παρακάτω κανόνες:

1. Δεν πρέπει να περιλαμβάνει δυναμικές παραμέτρους.
2. Δεν πρέπει να εκτελεί εντολές create, alter, ή drop για τον πίνακα που είναι ορισμένο το έναυσμα.

3. Δεν πρέπει να δημιουργεί ή να αφαιρεί ένα index για τον πίνακα που είναι ορισμένο το έναυσμα.
4. Δεν πρέπει να δημιουργεί ή να αφαιρεί ένα έναυσμα για τον πίνακα που είναι ορισμένο το έναυσμα.

ΤΡΟΠΟΣ ΥΛΟΠΟΙΗΣΗΣ

Η σύνταξη ενός εναύσματος στην postgresql αποτελείται από δύο τμήματα, (α) τη δήλωση της συνάρτησης τύπου trigger που θα υλοποιήσει την ενέργεια του trigger και (β) τον trigger.

Δήλωση Συνάρτησης

```
CREATE FUNCTION όνομα_συνάρτησης()
RETURNS trigger AS
$$
Begin
      Σώμα
end;
$$ LANGUAGE 'plpgsql'
```

Δήλωση Εναύσματος

```
CREATE TRIGGER όνομα_εναύσματος
Before|after insert|delete|update on όνομα_πίνακα
For each row execute procedure όνομα_συνάρτησης();
```

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΕΝΟΣ ΕΝΑΥΣΜΑΤΟΣ

Μία συνάρτηση εναύσματος δημιουργείται με την εντολή CREATE FUNCTION, δηλώνοντας την ως συνάρτηση χωρίς παραμέτρους που επιστρέφει τύπο έναυσμα. Πρέπει να επιστρέφει NULL ή μία εγγραφή/γραμμή της δομής του πίνακα για τον οποίο ενεργοποιήθηκε ο trigger. Οι παράμετροι σε μία συνάρτηση εναύσματος «περνάνε» με τις ειδικές μεταβλητές TG_ARGV. Συνοπτικά, οι σημαντικότερες ειδικές μεταβλητές είναι:

- **NEW** type RECORD, αυτή η μεταβλητή κρατάει τη νέα εγγραφή που προκύπτει από εντολή INSERT/UPDATE σε row-επιπέδου triggers. Η τιμή της μεταβλητής είναι NULL για την εντολή DELETE.
- **OLD** type RECORD, αυτή η μεταβλητή κρατάει τη παλιά γραμμή πριν αλλάξει από μία εντολή UPDATE/DELETE σε row-επιπέδου triggers. Η τιμή της μεταβλητής είναι NULL για την εντολή INSERT.
- **TG_NAME**, αυτή η μεταβλητή περιέχει το όνομα του trigger που μόλις ενεργοποιήθηκε.

- **TG_WHEN**, περιέχει την τιμή BEFORE ή AFTER όπως έχει δηλωθεί στον trigger.
- **TG_LEVEL**, περιέχει την τιμή ROW or STATEMENT όπως έχει δηλωθεί στον trigger.
- **TG_OP**, περιέχει την τιμή INSERT, UPDATE, DELETE, όπως έχει δηλωθεί στον trigger.
- **TG_RELID**, περιέχει το object ID του πίνακα για τον οποίο ενεργοποιήθηκε ο trigger.
- **TG_TABLE_NAME**, περιέχει το όνομα του πίνακα για τον οποίο ενεργοποιήθηκε ο trigger.
- **TG_TABLE_SCHEMA**, περιέχει το όνομα του σχήματος που ανήκει ο πίνακας για τον οποίο ενεργοποιήθηκε ο trigger.

<u>Παράδειγμα 1ο</u>

Να δημιουργηθεί έναυσμα το οποίο θα αυξάνει το rating ενός ναυτικού όταν πραγματοποιεί περισσότερες από 5 κρατήσεις την ίδια μέρα.

```
CREATE FUNCTION upgradeRating()
RETURNS TRIGGER AS
$$
DECLARE sid_per_day INTEGER;      --  δήλωση μεταβλητής
BEGIN
      SELECT COUNT(*) INTO sid_per_day    -- κατέβασμα το αποτέλεσμα
                                          --  στη μεταβλητή
      FROM Reserves
      WHERE sid = NEW.sid and -- εξασφαλίζει ότι ο έλεγχος γίνεται
            day1 = NEW.day1; -- με βάση τα δεδομένα της νέας εγγραφής
      IF sid_per_day > 5 THEN
            UPDATE Sailors
            SET rating=rating +1
            WHERE sid = NEW.sid;
      END IF;
      RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER upgradeRatingTrigger
AFTER INSERT ON Reserves
FOR EACH ROW
EXECUTE PROCEDURE upgradeRating();
```

Εκτελέστε τις κατάλληλες insert εντολές ώστε στον πίνακα reserves να καταχωρηθεί κωδικός ναυτικού να έχει κάνει την ίδια μέρα περισσότερες από 5 κρατήσεις προφανώς σε περισσότερες από 5 διαφορετικές βάρκες. Αν χρειαστεί καταχωρίστε και νέες βάρκες.

Παράδειγμα 2ο

Να δημιουργηθεί έναυσμα το οποίο εξασφαλίζει την υποχρεωτική εισαγωγή των πεδίων empid, empname και salary του πίνακα emp (η δομή του πίνακα φαίνεται παρακάτω), την εισαγωγή θετικής τιμή στο πεδίο salary και την εισαγωγή της στιγμής καταχώρησης μιας νέας εγγραφής στο πεδίο last_date και το όνομα του χρήστη που έκανε την καταχώρηση στο πεδίο last_user.

```
CREATE TABLE emp (
     empid integer NOT NULL primary key,
     empname varchar(20),
     salary integer,
     last_date timestamp,
     last_user varchar(20)
 );

CREATE FUNCTION emp_stamp () RETURNS TRIGGER AS $$
  BEGIN
    -- Έλεγξε αν εισάγονται τιμές για τα πεδία  empname και salary
    IF NEW.empname ISNULL THEN
    -- με τη RAISE EXCEPTION εμφανίζεται το σφάλμα.
       RAISE EXCEPTION 'empname cannot be NULL value';
    END IF;
    IF NEW.salary ISNULL THEN
      RAISE EXCEPTION '% cannot have NULL salary', NEW.empname;
    END IF;

    --  Έλεγξε αν δίνεται αρνητικός μισθός
    IF NEW.salary < 0 THEN
      RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
    END IF;

    -- Καταχώρησε ποιος έκανε την  αλλαγή στον μισθό
    NEW.last_date := ''now'';
    NEW.last_user := current_user;
    RETURN NEW;
 END;
 $$ LANGUAGE 'plpgsql';

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
     FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

Δοκιμάζουμε το έναυσμα με την εκτέλεση των παρακάτω εντολών.

```
insert into emp(empid, salary) values (1, 100);
ERROR: empname cannot be NULL value
SQL state: P0001

insert into emp(empid, empname) values (2, 'George');
ERROR: George cannot have NULL salary
SQL state: P0001


insert into emp(empid, empname,salary) values (3, 'George', -100);
ERROR: George cannot have a negative salary
SQL state: P0001
```

```
insert into emp(empid, empname,salary) values (4, 'George', 100);
Query returned successfully: 1 rows affected, 0 ms execution time.
```

Παράδειγμα 3ο

Να δημιουργηθεί έναυσμα το οποίο θα καταχωρεί σε ένα πίνακα Log File όλες τις ενέργειες INSERT/UPDATE/DELETE δεδομένων που γίνονται σε μία βάση δεδομένων..

```
-- Βοηθητικός πίνακας emp για παρακολούθηση των ενεργειών που
-- γίνονται σε αυτόν
CREATE TABLE emp2 (
    empname varchar(20) NOT NULL primary key,
    salary       integer
);

-- Ο πίνακας Log File
CREATE TABLE emp_audit (
    operation   char(1)   NOT NULL,
    stamp        timestamp NOT NULL primary key,
    userid        varchar(20)      NOT NULL,
    empname  varchar(20)      NOT NULL,
    salary        integer
);

-- Η συνάρτηση εναύσματος
CREATE OR REPLACE FUNCTION process_emp_audit()
RETURNS TRIGGER AS
$$
BEGIN
  IF (TG_OP = 'DELETE') THEN
  -- Καταχωρείται D για Delete, η στιγμή της διαγραφή, ο χρήστης
  -- και οι τιμές των δύο στηλών της εγγραφής του  emp πριν διαγραφούν
    INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
    RETURN OLD;
  ELSIF (TG_OP = 'UPDATE') THEN
-- Καταχωρείται U για Update, η στιγμή της μεταβολής, ο χρήστης
-- και οι νέες τιμές των δύο στηλών της εγγραφής του  emp μετά
-- την μεταβολή.
    INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
    RETURN NEW;
  ELSIF (TG_OP = 'INSERT') THEN
-- Καταχωρείται I για Insert, η στιγμή της εισαγωγής, ο χρήστης
-- και οι τιμές των δύο στηλών της νέας εγγραφής του  emp.
    INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
    RETURN NEW;
  END IF;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;

-- Το έναυσμα
CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE ON emp2
    FOR EACH ROW EXECUTE PROCEDURE process_emp_audit();
```

Μπορείτε να δοκιμάσετε το έναυσμα με τις παρακάτω εντολές.

```
1. insert into emp2(empname, salary) values ('John', 10000);
2. insert into emp2(empname, salary) values ('Maria', 10000);
3. delete from emp2 where empname = ‘Maria';
4. update emp2 set salary = salary * 1.1 where empname = 'John';
```

Περισσότερα μπορείτε να βρείτε στο

http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html