Σύντομη Επανάληψη (όχι πλήρης) σε NESTED SUBQUERIES και πιο ΣΥΝΘΕΤΑ ΕΡΩΤΗΜΑΤΑ

Αντώνιος Δεληγιαννάκης

Υποερωτήματα - SUBQUERIES

Υποερωτήματα μπορούμε να έχουμε στο

- SELECT: Πρέπει να επιστρέφει 1 attribute και 1 tuple μόνο
- FROM: Ορισμός ενός πίνακα που θα χρησιμοποιήσουμε στο WHERE. Χρειάζεται να δηλώσουμε ψευδώνυμο
- WHERE: Συχνά με συνδυασμό τελεστών IN ή NOT IN ή EXISTS ή NOT EXISTS ή ALL ή ANY
- HAVING: Για έλεγχο αν ικανοποιείται μία σύνθετη συνθήκη σε συναθροιστικές τιμές κάποιων group από tuples

Χρειάζονται παρενθέσεις στα υποερωτήματα

Σημαντικό Σχόλιο

Τα IN και NOT IN επιτρέπουν έλεγχο **σε 1 γνώρισμα μόνο**

Αυτός είναι ο κανόνας, και **πρέπει τα ερωτήματά σας να ακολουθούν αυτόν τον κανόνα**

Κάποια ΣΔΒΔ παραβιάζουν τον κανόνα και επιτρέπουν να ελέγχεις αν πολλά γνωρίσματα παίρνουν ταυτόχρονα τιμή που υπάρχει ως tuple σε ένα ερώτημα που επιστρέφει πολλά γνωρίσματα

ΔΕΝ πρέπει να γράφετε έτσι ερωτήματα

- Το ερώτημά σας δε θα δουλεύει ούτε καν σε παλαιότερες εκδόσεις του ίδιου ΣΔΒΔ, αν παραβιάσετε τον κανόνα

Προσωπικά δε θα θεωρώ σωστά ερωτήματα που παραβιάζουν το πρότυπο της SQL και δε δουλεύουν σε όλα τα συστήματα

EXISTS και NOT EXISTS

Πώς κάνω σωστά έλεγχο σε πολλά γνωρίσματα ταυτόχρονα;

- EXISTS
- NOT EXISTS

Τελεστής EXISTS

Τυπικά θα αναφερθείτε μέσα στο υποερώτημα σε attribute του εξωτερικού ερωτήματος. Αντίστοιχα και στο NOT EXISTS

```
SELECT  A1, A2, ..., An
FROM  R1, R2, …, Rm
WHERE EXISTS   (SELECT B1, B2,  …, Br
                FROM V1, V2, …, Vc
                WHERE Vi.B1 = Rj.A3 and …. )
```

Τοπικό Attribute (Vi.B1)

και

Εξωτερικό Attribute (Rj.A3)

Τελεστής EXISTS

Θυμηθείτε ότι στο WHERE ελέγχουμε 1-1 τα tuples που προκύπτουν από το JOIN των πινάκων στο FROM

Κάθε ένα τέτοιο tuple στο WHERE θα έχει πολλά γνωρίσματα, και τιμή σε καθένα από αυτά

Μπορούμε να αναφερθούμε μέσα στο Nested Subquery EXISTS στις τιμές των γνωρισμάτων αυτού του tuple με Ri.ONOMA_ATTRIBUTE

```
SELECT  A1, A2, …, An
FROM  R1, R2, …, Rm
WHERE EXISTS   (SELECT B1, B2,  …, Br
                FROM V1, V2, …, Vc
                WHERE Vi.B1 = Rj.A3 and …. )
```

Τελεστής EXISTS

Βρες τα sID όλων των μαθητών που έχουν κάνει αίτηση σε κολλέγια στα οποία δεν είναι μοναδικοί υποψήφιοι (θα πρέπει και κάποιος άλλος να έχει κάνει αίτηση εκεί)

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS  (SELECT *  FROM APPLY A2
        WHERE A2.sID <> A1.sID and
              A2.cName = A1.cName)
```

Γιατί DISTINCT;

Τοπικό Attribute και Εξωτερικό Attribute

sID
1
3
4

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Student

sID	sName	GPA	hs
1	Jim	3.5	1200
2	Peter	3.7	800
3	Mary	3.5	1200
4	Jim	3.8	1400

Τελεστής EXISTS – Πώς εκτελείται;

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS  (SELECT *  FROM APPLY A2
                  WHERE A2.sID <> A1.sID and
                        A2.cName = A1.cName)
```

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Βήμα 1: Γίνεται το Join των πινάκων στο FROM

- Εδώ έχουμε μόνο 1 πίνακα

Βήμα 2: Σκεφτείτε ότι γίνεται ένα For Loop στα tuples που προκύπτουν από το JOIN και ελέγχεται το EXISTS σε 1-1 από αυτά

- Για κάθε tuple, χρησιμοποιούμε τις τιμές του στα εξωτερικά γνωρίσματα A1.sID, A1.cName

Τελεστής EXISTS – Πώς εκτελείται;

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS   (SELECT *  FROM APPLY A2
                  WHERE A2.sID <> A1.sID and
                        A2.cName = A1.cName)
```

sID
1

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Ελέγχω αν υπάρχει έστω ένα tuple στο

```
SELECT *  FROM APPLY A2
WHERE A2.sID <> 1 and
      A2.cName = 'TUC'
```

ΝΑΙ. Θα επιστραφεί η 4η εγγραφή του Apply

Τελεστής EXISTS – Πώς εκτελείται;

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS  (SELECT *  FROM APPLY A2
                  WHERE A2.sID <> A1.sID and
                        A2.cName = A1.cName)
```

sID
1

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Ελέγχω αν υπάρχει έστω ένα tuple στο

```
SELECT *  FROM APPLY A2
WHERE A2.sID <> 1 and
      A2.cName = 'MIT'
```

ΝΑΙ. Θα επιστραφεί η 3η εγγραφή του Apply

Το DISTINCT δε θα προσθέσει το sID = 1

Τελεστής EXISTS – Πώς εκτελείται;

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS   (SELECT *  FROM APPLY A2
                  WHERE A2.sID <> A1.sID and
                        A2.cName = A1.cName)
```

sID
1
3

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Ελέγχω αν υπάρχει έστω ένα tuple στο

```
SELECT *  FROM APPLY A2
WHERE A2.sID <> 3 and
      A2.cName = 'MIT'
```

ΝΑΙ. Θα επιστραφεί η 2η εγγραφή του Apply

Τελεστής EXISTS – Πώς εκτελείται;

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS  (SELECT *  FROM APPLY A2
                  WHERE A2.sID <> A1.sID and
                        A2.cName = A1.cName)
```

sID
1
3
4

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Ελέγχω αν υπάρχει έστω ένα tuple στο

```
SELECT *  FROM APPLY A2
WHERE A2.sID <> 4 and
      A2.cName = 'TUC'
```

ΝΑΙ. Θα επιστραφεί η 1η εγγραφή του Apply

Τελεστής EXISTS – Πώς εκτελείται;

```
SELECT  DISTINCT sID
FROM  Apply A1
WHERE EXISTS   (SELECT *  FROM APPLY A2
                  WHERE A2.sID <> A1.sID and
                        A2.cName = A1.cName)
```

sID
1
3
4

Apply

sID	cName	major	dec
1	TUC	CS	YES
1	MIT	MPD	NO
3	MIT	EE	NO
4	TUC	CS	YES
5	EMP	CS	NO

Ελέγχω αν υπάρχει έστω ένα tuple στο

```
SELECT *  FROM APPLY A2
WHERE A2.sID <> 5 and
      A2.cName = 'EMP'
```

ΟΧΙ. Δεν επιστρέφει καμία εγγραφή

Εξάσκηση …

Τυπώστε όλα τα στοιχεία των κολλεγίων που είναι τα μοναδικά στην πολιτεία τους.

Πολλές επιλογές:

- … δεν υπάρχει (NOT EXISTS) άλλο κολλέγιο (με διαφορετικό cName) που να έχει ίδια τιμή στο attribute state
- Από όλα τα κολλέγια, αφαιρούμε (EXCEPT) αυτά για τα οποία υπάρχει άλλο κολλέγιο (self join) με ίδια τιμή στο attribute *state*

Εξάσκηση (πιο σύνθετο) …

Τυπώστε όλα τα στοιχεία των κολλεγίων όπου έχουν κάνει αίτηση τουλάχιστον 2 μαθητές με το ίδιο όνομα (sName).

Πολλές επιλογές:

- Σύνθετο self-join (College με Apply με Student μπορεί να γίνει self join με ένα άλλο αντίγραφο του ίδιου τριπλού join)
- Για κάθε μαθητή που έχει κάνει αίτηση στο Κολλέγιο, έλεγξε αν υπάρχει (EXISTS) άλλος μαθητής (με διαφορετικό sID) αλλά με ίδιο όνομα (sName) που να έχει κάνει αίτηση στο ίδιο κολλέγιο
- Για κάθε μαθητή που έχει κάνει αίτηση στο Κολλέγιο, έλεγξε αν το όνομα του Κολλεγίου ανήκει (IN) στα ονόματα των κολλεγίων στα οποία έχουν κάνει αίτηση (Apply) μαθητές με άλλο sID, αλλά με ίδιο όνομα (sName)
- Ποια κολλέγια θέλουμε να εμφανίσουμε; Αυτά όπου ο συνδυασμός cName με το sName μαθητών που έχουν κάνει αίτηση στο ίδιο κολλέγιο δεν είναι μοναδικός (το count του είναι μεγαλύτερο ίσο του 2)

ALL & ANY

Θυμηθείτε ότι συγκρίσεις κάνουμε στο WHERE και στο HAVING

- Στο WHERE μπορούμε να συγκρίνουμε την τιμή ενός attribute
- Στο HAVING μπορούμε να συγκρίνουμε την τιμή κάποιας συναθροιστικής τιμής που υπολογίζεται για κάθε ομάδα
- Πώς θα υπολογίζαμε τα ονόματα των κολλεγίων στα οποία έχουν γίνει οι περισσότερες αιτήσεις;
  - Όλη η πληροφορία υπάρχει στον πίνακα Apply
  - Θέλουμε αριθμό αιτήσεων, άρα κάποιο count
  - Θέλουμε count αιτήσεων ανά κολλέγιο, άρα group by με το όνομα του κολλεγίου
  - Συνθήκη στο count (θέλουμε το μεγαλύτερο), άρα συνθήκη στο HAVING

ALL & ANY

Πώς θα υπολογίζαμε τα ονόματα των κολλεγίων στα οποία έχουν γίνει οι περισσότερες αιτήσεις;

```
SELECT cName
FROM    Apply
GROUP BY cName
HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                        FROM Apply
                        GROUP BY cName)
```

Σημείωση: Πολύ συχνά (**<u>όχι πάντα</u>**), το υποερώτημα μέσα στο ALL θα μοιάζει με το εξωτερικό ερώτημα.

1. Στο SELECT συχνά θα περιέχει τη συναθροιστική τιμή που ελέγχουμε στο HAVING
2. Το υπόλοιπο ερώτημα συχνά θα είναι ολόιδιο με το εξωτερικό ερώτημα

CTE – Πολλαπλοί ορισμοί

```
WITH MaxHS (maxhs) AS (
    SELECT MAX(hs)
    FROM Student
),
StudentsWithMaxHS AS (
    SELECT sID, sName
    FROM Student, MaxHS
    WHERE hs = MaxHS.maxhs
)
SELECT *
FROM StudentsWithMaxHS
```

Αν δε δηλώσουμε ρητά τα ονόματα των attributes, παίρνει τα ονόματα των attributes του ερωτήματος που ορίζει το CTE

Εξάσκηση …

Τυπώστε όλα τα στοιχεία των κολλεγίων που έχουν τον μεγαλύτερο πληθυσμό φοιτητών.

Συνθήκη σε attribute (enr), συνεπώς χρειαζόμαστε συνθήκη στο WHERE

Εξάσκηση …

Τυπώστε τα ονόματα των κολλεγίων που έχουν το μεγαλύτερο αριθμό αρνητικών απαντήσεων σε αιτήσεις.

Χρειάζεται να μετρήσουμε (count) τις αρνητικές απαντήσεις (dec = 'NO) ανά κολλέγιο, άρα θέλουμε συνθήκη στο HAVING

Triggers
(Σκανδάλη)

OLD και NEW

Σε εισαγωγές, OLD = NULL – αναφερόμαστε μόνο σε NEW

Σε διαγραφές, NEW = NULL – αναφερόμαστε μόνο σε OLD

Σε ενημερώσεις έχουν νόημα και το NEW και το OLD

Τα NEW, OLD είναι προσβάσιμα ΚΑΙ στη συνάρτηση του trigger

INSERT	DELETE	UPDATE
NEW: Το νέο tuple που κάνουμε insert	NEW: NULL Δεν θα υπάρχει το tuple στον πίνακα μετά	NEW: Το tuple που κάνουμε update ΜΕΤΑ την ενημέρωση
OLD: NULL Δεν υπήρχε το tuple στον πίνακα πιο πριν	OLD: Το tuple που κάνουμε delete	OLD: Το tuple που κάνουμε update ΠΡΙΝ την ενημέρωση

Τιμές Επιστροφής σε Row Level Triggers

BEFORE TRIGGERS:

1. Αν θέλουμε να ΜΗ γίνει η εισαγωγή, διαγραφή, ενημέρωση της εγγραφής, επιστρέφουμε NULL
2. Αν θέλουμε να προχωρήσει η διαγραφή, εισαγωγή, ενημέρωση τότε:
   1. Για DELETE, επιστρέφουμε OLD ή κάτι που δεν είναι NULL
   2. Για INSERT/UPDATE επιστρέφουμε NEW.

      Σημείωση: Αν θέλουμε, στα πεδία του NEW μπορούμε να βάλουμε δικές μας τιμές μέσα στη συνάρτηση του trigger

AFTER TRIGGERS: **Η τιμή επιστροφής αγνοείται**

Σημαντικά για το Χρονισμό των Triggers

Αν θέλουμε σε κάποιες περιπτώσεις να μην επιτρέψουμε μία μεταβολή, θέλουμε BEFORE trigger

Πολύ συχνά τα triggers τα δημιουργούμε και πραγματοποιούν ενέργειες μεταξύ πινάκων που σχετίζονται (έχουν foreign key ο ένας στον άλλον)

Έχει μεγάλη σημασία η σειρά των ενεργειών όταν έχουμε foreign key. Π.χ.:

1. Πρώτα πρέπει να γίνει η εισαγωγή tuple στον πίνακα με το PRIMARY KEY και μετά σε αυτόν με το FOREIGN KEY
2. Πρώτα πρέπει να γίνει η διαγραφή tuple στον πίνακα με το FOREIGN KEY και μετά σε αυτόν με το PRIMARY KEY

Παραδείγματα – Insert στη Student

Σε κάθε εισαγωγή μαθητή με GPA > 3.75 θα εισάγουμε απευθείας αίτησή του στο CS του MIT και του TUC με μη καθορισμένη (NULL) απόφαση

```
CREATE TRIGGER check_Student_insert
AFTER INSERT ON "Student"
FOR EACH ROW
WHEN (NEW."GPA" > 3.75)
EXECUTE PROCEDURE insert_apply_dec();

CREATE OR REPLACE FUNCTION insert_apply_dec()
RETURNS TRIGGER AS $$
BEGIN
        INSERT INTO "Apply" VALUES(NEW."sID", 'MIT', 'CS', null);
        INSERT INTO "Apply" VALUES(NEW."sID", 'TUC', 'CS', null);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Αν έχουμε ξένο κλειδί, ΠΡΕΠΕΙ πρώτα να γίνει η εισαγωγή στη Student και μετά στην Apply

AFTER TRIGGER
Δεν έχει σημασία η τιμή επιστροφής. Για να μη μπερδεύεστε, μπορείτε να επιστρέφετε ότι και σε BEFORE trigger

Τι Συμβαίνει σε Insert;

Αρχίζουμε το παράδειγμα με τους πίνακες στο κάτω μέρος

```
insert into "Student" values(6, 'Ge', 3.8, 900);
insert into "Student" values(7, 'Hi', 3.7, 1900);
select * from "Apply";
```

Περιμένουμε να έχουν προστεθεί αιτήσεις για το μαθητή με sID=6

Έστω ότι υπάρχει το MIT

College

cName	state	enr
MIT	MA	1500
TUC	CH	800
EMP	AT	1100

Student

sID	sName	GPA	hs
1	Jim	3.6	1100
2	Peter	3.8	800
3	Mary	3.7	1200
4	Jim	3.9	1400
5	Mac	3.55	900

Apply

sID	cName	major	dec
1	TUC	CS	YES
4	TUC	CS	NO
5	TUC	CS	YES

Τι Συμβαίνει σε Insert;

Αρχίζουμε το παράδειγμα με τους πίνακες στο κάτω μέρος

```
insert into "Student" values(6, 'Ge', 3.8, 900);
insert into "Student" values(7, 'Hi', 3.7, 1900);
select * from "Apply";
```

Περιμένουμε να έχουν προστεθεί αιτήσεις για το μαθητή με sID=6

Νέο Apply

sID	cName	major	dec
1	TUC	CS	YES
4	TUC	CS	NO
5	TUC	CS	YES
6	MIT	CS	
6	TUC	CS	

Νέο Student

sID	sName	GPA	hs
1	Jim	3.6	1100
2	Peter	3.8	800
3	Mary	3.7	1200
4	Jim	3.9	1400
5	Mac	3.55	900
6	Ge	3.8	900
7	Hi	3.7	1900

College

cName	state	enr
MIT	MA	1500
TUC	CH	800
EMP	AT	1100

Παλιό Apply

sID	cName	major	dec
1	TUC	CS	YES
4	TUC	CS	NO
5	TUC	CS	YES

Παραδείγματα – Delete στην College

Σε περίπτωση διαγραφής ενός κολλεγίου από τη σχέση College, θα διαγράφουμε όλες τις αιτήσεις προς αυτό το κολλέγιο

```
CREATE TRIGGER check_College_delete
BEFORE DELETE ON "College"
FOR EACH ROW
EXECUTE PROCEDURE delete_apply();

CREATE OR REPLACE FUNCTION delete_apply()
RETURNS TRIGGER AS $$
BEGIN
        DELETE FROM "Apply" WHERE "Apply"."cName" = OLD."cName";
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

Αν έχουμε ξένο κλειδί, ΠΡΕΠΕΙ πρώτα να διαγράψουμε από την Apply και μετά από την College

BEFORE TRIGGER
Για DELETE επιστρέφουμε OLD, ή κάτι που δεν είναι NULL

Παραδείγματα – Απόρριψη Delete

Σε αυτή την περίπτωση πρέπει η συνάρτηση μας να επιστρέψει NULL αν εκτελεστεί πριν το DELETE. Αν εκτελεστεί μετά το DELETE, πρέπει μέσα στη συνάρτηση να διαγράψουμε τη νέα εγγραφή. Η πρώτη επιλογή είναι πιο ασφαλής

```
CREATE TRIGGER check_Student_delete
BEFORE DELETE ON "Student"
FOR EACH ROW EXECUTE PROCEDURE check_delete_student();

CREATE OR REPLACE FUNCTION check_delete_student()
RETURNS TRIGGER AS $$
BEGIN
        IF OLD."sID" > 5 THEN
                RAISE NOTICE 'Will not delete sID greater than 5';
                RETURN NULL;
        END IF;
        RETURN OLD;
 END;
$$ LANGUAGE plpgsql;
```

Αφού ίσως απορριφθεί η διαγραφή, θέλουμε BEFORE

Επιστροφή NULL σε BEFORE DELETE σημαίνει απόρριψη της διαγραφής

Τι Συμβαίνει σε Delete;

Αρχίζουμε το παράδειγμα με τους πίνακες στο κάτω μέρος

delete from "Student" where "sID" = 5;

delete from "Student" where "sID" = 6;

select * from "Student";

Περιμένουμε να μην έχει διαγραφεί ο μαθητής με sID=6

College

cName	state	enr
MIT	MA	1500
TUC	CH	800
EMP	AT	1100

Student

sID	sName	GPA	hs
1	Jim	3.6	1100
2	Peter	3.8	800
3	Mary	3.7	1200
4	Jim	3.9	1400
~~5~~	~~Mac~~	~~3.55~~	~~900~~
6	Ge	3.8	900
7	Hi	3.7	1900

Τι Συμβαίνει σε Delete;

Αρχίζουμε το παράδειγμα με τους πίνακες στο κάτω μέρος

```
delete from "Student" where "sID" = 5;
delete from "Student" where "sID" = 6;
select * from "Student";
```

Περιμένουμε να μην έχει διαγραφεί ο μαθητής με sID=6

College

cName	state	enr
MIT	MA	1500
TUC	CH	800
EMP	AT	1100

Νέο Student

sID	sName	GPA	hs
1	Jim	3.6	1100
2	Peter	3.8	800
3	Mary	3.7	1200
4	Jim	3.9	1400
6	Ge	3.8	900
7	Hi	3.7	1900