



EPL342 –Databases

Lecture 15: SQL DML II

SQL Structured Query Language

(Chapter 8.4-8.5, Elmasri-Navathe 5ED)

Διδάσκων: Παναγιώτης Ανδρέου

<http://www.cs.ucy.ac.cy/courses/EPL342>



Περιεχόμενο Διάλεξης

Κεφάλαιο 8.4-8.5.4: SQL DML I

- Σύγκριση Συμβολοσειρών με **LIKE**
- Διάταξη Αποτελεσμάτων με **ORDER-BY**
- Συγκρίσεις με **NULLS (IS NULL)**,
- Εμφωλευμένες Επερωτήσεις με Συσχέτιση / Χωρίς Συσχέτιση (**Correlated/Uncorrelated Nested Queries**)
- Σύγκριση Συνόλων / Πολύσυνόλων σε Επερωτήσεις SQL (**EXIST, IN, or-ALL, or-ANY**),
- Διαίρεση με χρήση **NOT EXISTS ... EXCEPT**

Σύγκριση Συμβολοσειρών με **LIKE**



- Για την σύγκριση συμβολοσειρών (substring matching) σε SQL γίνεται χρήση του **LIKE**.

WHERE Attribute [NOT] LIKE Pattern

- **Attribute**: Γνώρισμα ή οποιαδήποτε έγκυρη έκφραση.
- **Pattern**: Συμβολοσειρά (υπό μορφή «Κανονικής Έκφρασης») η οποία πρέπει να αναζητηθεί στο *attribute*.
 - Το Pattern μπορεί να περιέχει χαρακτήρες wildcard (next slide)
 - Το *pattern* μπορεί να είναι μέχρι 8,000 bytes στην TSQL
- Παράδειγμα
SELECT FirstName, LastName, Phone
FROM Person.Contact
WHERE phone **LIKE '415%'**

Σύγκριση Συμβολοσειρών με **LIKE**



Wildcard character	Περιγραφή	Παράδειγμα
%	Συμβολοσειρά 0 ή περισσότερων χαρακτήρων	WHERE title LIKE '%computer%' finds all book titles with the word 'computer' anywhere in the book title.
_ (underscore)	Οποιοσδήποτε Χαρακτήρας	WHERE name LIKE '_ean' finds all four-letter first names that end with ean (Dean, Sean, and so on).
[]	Οποιοσδήποτε χαρακτήρας σε εύρος ([a-f]) ή σύνολο ([abcdef]).	WHERE name LIKE '[C-P]arsen' finds author last names ending with arsen and starting with any single character between C and P, for example Carsen, Larsen, Karsen, and so on..
[^]	Οποιοσδήποτε χαρακτήρας που ΔΕΝ είναι σε ευρος ([^a-f]) ή σύνολο ([^abcdef]).	WHERE name LIKE 'de[^I]%' all author last names starting with de and where the following letter is not I.

Σύγκριση Συμβολοσειρών με **LIKE**



- **Query 12:** Βρες τους υπαλλήλους που γεννήθηκαν κατά την δεκαετία του 1950s.
 - Θεωρήστε ότι η ημερομηνία έχει την μορφοποίηση: MMDDYYYY
 - Συνεπώς, ψάχνουμε το BDATE με τιμή '____195_', όπου το underscore υποδηλώνει ένα αυθαίρετο χαρακτήρα.

```
Q12: SELECT      FNAME, LNAME
      FROM        EMPLOYEE
      WHERE BDATE LIKE '____195_'
```

Πράξεις σε Αποτελέσματα της Select



- Στα πλαίσια της Σχεσιακής Άλγεβρας είχαμε δει την **γενικευμένη προβολή** η οποία επέτρεπε εκτέλεση πράξεων σε αποτελέσματα της SQL. π.χ.,
 - Π **Ταυτότητα**, **Μισθός-Αποκοπές**, **2000*Χρόνια_Υπηρεσίας**, **0.25*Μισθός** (**EMPLOYEE**)
- Κάτι τέτοιο μπορεί να γίνει και στα πλαίσια της SQL.
- Τέτοιες πράξεις μπορεί να είναι
 - **Συναρτήσεις Συμβολοσειρών** (Substring, κτλ)
 - **Συναρτήσεις Ημερομηνιών** (Datediff, Getdate, κτλ)
 - **Αριθμητικές πράξεις** ('+', '-', '*', and '/')
 - **Μαθηματικές και άλλες Πράξεις** (sqrt, PI, κτλ.)
 - Δείτε το manual της TSQL για παραδείγματα ...

Πράξεις σε Αποτελέσματα της Select



- **Παραδείγματα Πράξεων στην SELECT**

1. `SELECT LastName, SUBSTRING(FirstName, 1, 1) AS Initial`

2. `SELECT FNAME, LNAME, 1.1*SALARY`

3. `SELECT SOUNDEX(Name)`

- *Επιστρέφει ένα κωδικό 4 χαρακτήρων που μπορεί να αξιοποιηθεί για να αποτιμηθεί εάν 2 strings ακούγονται το ίδιο*

Π.χ., `SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe');`

Επιστρέφει `S530 S530`

4. `SELECT DIFFERENCE(Name, Surname)`

- *Βρίσκει την ομοιότητα δυο γνωρισμάτων βάσει του `SOUNDEX` code.*

- *Λαμβάνει υπόψη μόνο τα πρώτα 8000 bytes των `char`, `varchar` ή `text`*

- *Παίρνει τιμές από 0 (καμία ομοιότητα) .. 4 (ίδια)*

Διάταξη Αποτελεσμάτων με το ORDER BY



- Η όρος **ORDER BY** χρησιμοποιείται σε μια έκφραση SQL για να ταξινομούνται οι πλειάδες μιας επερώτησης βάσει κάποιου/ων γνωρισμάτων που δηλώνονται. → default

ORDER BY <attribute-list> [ASC | DESC]

- **Query 15:** Τύπωσε τα στοιχεία όλων των employees του department 5 ταξινομημένα ανά επίθετο και μετά ανά όνομα
Q15: **SELECT * FROM EMPLOYEE E WHERE E.Dno=5
ORDER BY E.LNAME, E.FNAME**

Επισημάνσεις

- Δεν χρησιμοποιείται για ntext, text, image, κτλ.
- Το **ORDER BY** μπορεί να αναφέρεται σε γνωρίσματα που **ΔΕΝ** εμφανίζονται στο **SELECT list**
 - Π.χ., **SELECT E.age FROM Emp E ORDER BY E.lname** **OK**
- Δεν ισχύει με **SELECT DISTINCT** ή **GROUP BY** σε TSQL.
 - Π.χ., **SELECT DISTINCT E.age FROM Emp E ORDER BY E.lname;** **WRONG**

NULLS σε Επερωτήσεις SQL

- Σε τυπικές γλώσσες προγραμματισμού, οι λογικές εκφράσεις αποτιμούνται σε **TRUE** ή **FALSE**.
- Στην SQL ωστόσο, η ύπαρξη NULL τιμών επιβάλλει την χρήση της **Λογικής Τριών Τιμών (Three-value logic 3VL)**
- Συγκεκριμένα, μια λογική έκφραση μπορεί να αποτιμηθεί σε **TRUE, FALSE** ή **UNKNOWN**
 - Π.χ., NULL AND TRUE αποτιμάται σε UNKNOWN
- Η αποτίμηση λογικών εκφράσεων γίνεται με βάσει των ακόλουθων πινάκων αληθείας:

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

NULLS σε Επερωτήσεις SQL

Παράδειγμα

```
CREATE TABLE test (  
  id INTEGER PRIMARY KEY,  
  age INTEGER CHECK (age < 0 AND age = 0 AND age > 0)  
);
```

- **Τι θα γίνει εάν προσπαθήσουμε να εισάγουμε το age=NULL;**
 - Θα απαγορέψει την εισαγωγή οποιασδήποτε τιμής ωστόσο μπορεί να εισαχθεί το NULL

- **Για να αποφύγω και τα NULL πλήρως:**

```
CREATE TABLE test (id INTEGER PRIMARY KEY,  
  age INTEGER NOT NULL CHECK (...));
```

NULLS σε Επερωτήσεις SQL



- Μια έκφραση του τύπου ~~WHERE Attribute=NULL~~, είναι ΛΑΝΘΑΣΜΕΝΗ στην SQL.
- Για σύγκριση ενός γνωρίσματος με NULL στην SQL χρησιμοποιείται η έκφραση **IS NULL** ή **IS NOT NULL**.
- **Επισημάνσεις**
 - **ANSI**: Δυο NULL τιμές είναι ανεξάρτητες (διαφορετικές)
 - **MSSQL**: ΔΥΟ NULL τιμές είναι οι ίδιες (εξ' ορισμού). Επομένως σε UNIQUE πεδίο δεν μπορούμε να έχουμε δυο εγγραφές με τιμή NULL.
 - Αυτό, επειδή υπάρχει η εξ' ορισμού ρύθμιση **ANSI_NULLS OFF** (στις ρυθμίσεις μιας βάσης)
 - Εάν ενεργοποιηθεί η εν λόγω μεταβλητή, με **SET ANSI_NULLS ON**, τότε δυο διακριτές NULL τιμές θα θεωρούνται διαφορετικές όπως άλλωστε προβλέπει και το ANSI πρότυπο.

NULLS σε Επερωτήσεις SQL



Relation

```
CREATE TABLE test ( id int PRIMARY KEY,  
    age INTEGER CHECK (age < 0 AND age = 0 AND age > 0 ) );
```

Query

```
SELECT CASE
```

```
    WHEN age IS NULL THEN 'Null Result' -- when age is NULL
```

```
    WHEN age < 0 THEN 'Less than 0'
```

```
    WHEN age > 0 THEN 'Greater than 0'
```

```
    WHEN age = 0 THEN 'Equal 0'
```

```
END
```

```
FROM test;
```

*Δεν εκτελούνται ποτέ
εάν η **CHECK**
συνθήκη εφαρμόστηκε
από την δημιουργία
της σχέσης **test**.*

test:

1, NULL

2, NULL

....

Query Returns:

1, Null Result

2, Null Result

....

NULLS σε Επερωτήσεις SQL



- **Query 14:** Βρες τα ονόματα όλων των υπαλλήλων που ΔΕΝ έχουν προϊστάμενους (supervisors).

```
Q14:      SELECT      FNAME, LNAME
           FROM        EMPLOYEE
           WHERE       SUPERSSN IS NULL
```

- Θυμηθείτε ότι εάν η **συνθήκη μιας συνένωσης παρουσιάσει NULL** τιμές τότε αυτές οι πλειάδες ΔΕΝ περιλαμβάνονται στο αποτέλεσμα
 - Π.χ., (ssn=12345, dno=NULL) *(dno=1,dname="A")
 - Στο αποτέλεσμα περιλαμβάνονται μόνο πλειάδες για τις οποίες η συνθήκη συνένωσης αποτιμάται σε **TRUE**
 - Το πιο πάνω ΔΕΝ ισχύει για Εξωτερικές Συνενώσεις (LEFT/RIGHT/FULL OUTER JOIN) τις οποίες θα δούμε στην ερχόμενη διάλεξη.

Απαριθμητά Σύνολα σε SQL (Enumerated Sets)



- Αντί για λογική συνθήκη στο WHERE της SQL, μπορούμε να κάνουμε την **σύγκριση-βάσει-προσδιορισμένου-συνόλου**, π.χ.,

- **Query 13:** Ανάκτησε το SSN όλων των employees που δουλεύουν στα projects 1, 2, ή 3.

```
Q13:      SELECT      DISTINCT ESSN
           FROM        WORKS_ON
           WHERE       PNO IN (1, 2, 3)
```

- Το (1,2,3) αποτελεί ένα απαριθμητό (enumerated) σύνολο βάσει του οποίου θα γίνει η σύγκριση.
- Το **IN** έχει αντίστοιχη λογική με τον τελεστή συνόλων **∈**
- Θα δούμε και άλλους όρους σύγκρισης με σύνολα (ANY, ALL, EXISTS, κτλ.).

Εμφωλευμένες Επερωτήσεις (Nested Queries)



- Οι επερωτήσεις που είδαμε μέχρι τώρα ήταν σε ένα (1) SELECT-FROM-WHERE μπλοκ.
- Στην SQL είναι δυνατό να εμφωλεύονται τα μπλοκ κατά αντίστοιχο τρόπο που εμφωλεύουμε τις επαναλήψεις σε μια γλώσσα προγραμματισμού.
- **Query1:** Βρες το όνομα και την διεύθυνση όλων των υπαλλήλων που δουλεύουν για το 'Research' department.

```
Q1: SELECT E.FNAME, E.LNAME, E.ADDRESS | Outer  
      FROM EMPLOYEE E                  | Query  
      WHERE E.DNO IN (  
          SELECT D.DNUMBER  
          FROM DEPARTMENT D  
          WHERE D.DNAME='Research') | Inner  
                                     | Query
```

Εμφωλευμένες Επερωτήσεις (Nested Queries)



- **ΚΑΘΕ Εμφωλευμένη Επερώτηση** μπορεί να αναπαρασταθεί από Επερώτηση 1-μπλοκ (συνδυάζοντας τα ενδεχομένως με πράξεις συνόλων (UNION, EXCEPT, κτλ)
 - Θα δούμε παράδειγμα σε λίγο...
- Η Εμφώλευση μπορεί να γίνει για **όσα επίπεδα** κάτω επιθυμούμε (στην TSQL μέχρι 32).
- Ωστόσο η βάση δεδομένων δυσκολεύεται να **βελτιστοποιήσει τέτοιες επερωτήσεις**, για αυτό η εμφώλευση πρέπει να χρησιμοποιείται με προσοχή.
- Γενικά είναι καλή πρακτική να εκφράζουμε τις επερωτήσεις απλά χωρίς βαθιές εμφωλεύσεις

Εμφωλευμένες Επερωτήσεις και Εμβέλεια Γνωρισμάτων



- **SELECT** E.FNAME, E.LNAME, E.ADDRESS
FROM EMPLOYEE E
WHERE E.DNO **IN** (
 SELECT D.DNUMBER
 FROM DEPARTMENT D
 WHERE D.DNAME='Research')

→ Γνώρισμα Σύγκρισης

- **Εμβέλεια Γνωρισμάτων Χωρίς Πρόθεμα:**
 - Σημειώστε ότι εάν δεν κάναμε χρήση των προθεμάτων **D**, **E** τότε η αναφορά σε κάποιο γνώρισμα θα αναφερόταν στο Inner Loop.
 - Για την **εμβέλεια των μεταβλητών (scope)**, χρησιμοποιούνται ουσιαστικά οι ίδιοι κανόνες που χρησιμοποιούνται στις γλώσσες προγραμματισμού.
- Το **Γνώρισμα Σύγκρισης** μπορεί να είναι διατεταγμένη ν-άδα π.χ., **(E.DNO, E.ESSN)** εφόσον το **INNER Block** είναι συμβατό-ως-προς-τον-τύπο με το γνώρισμα σύγκρισης.

Εμφωλευμένες Επερωτήσεις (Χρήση **op-ANY**, **op-ALL**)



- Οι εντολές **op-ANY**, **op-ALL**, όπου **op** είναι ένας τελεστής σύγκρισης (>, <, >=, <=, <>), χρησιμοποιούνται για να επεκτείνουν την εκφραστική δύναμη της SQL.
 - Προϋπ: Συμβατότητα προς-τύπο μεταξύ Γνωρίσματος-Inner Block

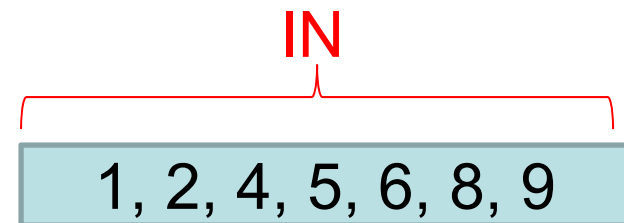
- **Παραδείγματα**

>= ALL ↔ **IN (9)** (εφόσον max=9)

> ANY ↔ **IN (2, 4, 5, 6, 8, 9)** (εφόσον min=1)

<= ALL ↔ **IN (1)** (εφόσον min=1)

< ANY ↔ **IN (8, 6, 5, 4, 2, 1)** (εφόσον max:9)



- **Παράδειγμα σε SQL:** Βρες τους υπαλλήλους που έχουν ψηλότερο μισθό από κάθε υπάλληλο στο τμήμα 5

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE E.Salary > ALL (
  SELECT E.Salary
  FROM EMPLOYEE E
  WHERE E.Dno=5)
```

Προσέξτε ότι μπορούμε να χρησιμοποιούμε δυο φορές την δήλωση E (overriding). Στον inner βρόχο χρησιμοποιείται το inner-E στο outer βρόχο το outer-E

Εμφωλευμένες Επερωτήσεις (Χρήση **op-ANY**, **op-ALL**)



- Παράδειγμα σε **SQL**: Βρες τους υπάλληλους που έχουν τον ψηλότερο μισθό από το τμήμα 5 με χρήση των τελεστών **op-ANY**, **op-ALL**

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE E.Dno=5 AND E.Salary >= ALL (
    SELECT E.Salary
    FROM EMPLOYEE E
    WHERE E.Dno=5)
```

- Αργότερα θα δούμε την ίδια Επερώτηση με συναρθροιστική συνάρτηση.

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE E.Dno=5 AND E.Salary = (
    SELECT MAX(E.Salary)
    FROM EMPLOYEE E
    WHERE E.Dno=5)
```

Εάν **INNER=(1000, 3000, 2000, 3000)** τότε το “>=ALL” query επιστρέφει 2 employees

Εάν το query ήταν “> ALL” με βάσει το πιο πάνω INNER θα επέστρεφε 0 employees

Εάν το query ήταν “= ALL” με βάσει το πιο πάνω INNER θα επέστρεφε 0 employees

Συσχετιζόμενες Εμφωλευμένες Επερωτήσεις (Correlated Nested Queries)



- Όταν μια συνθήκη στο WHERE του Outer-block αναφέρεται σε κάποιο γνώρισμα του Inner-block τότε οι δυο επερωτήσεις λέγεται ότι είναι **Συσχετιζόμενες (Correlated)**
 - **Σημείωση:** Η προηγούμενη επερώτηση δεν ήταν συσχετιζόμενη
- Query 12:** Βρες το **όνομα** κάθε employee που έχει ένα **dependent** με το **ίδιο όνομα** με τον **employee**.

Q12: SELECT
FROM
WHERE

E.FNAME, E.LNAME
EMPLOYEE **E**
E.SSN IN

Outer block

(SELECT **D.ESSN**
FROM **DEPENDENT D**
WHERE **D.ESSN=E.SSN AND**
E.FNAME=D.DEPENDENT_NAME)

Inner block

Εμφωλευμένες Επερωτήσεις vs. Επερωτήσεις 1 Μπλοκ



- **ΚΑΘΕ Εμφωλευμένη Επερώτηση** μπορεί να αναπαρασταθεί από Επερώτηση 1-μπλοκ (συνδυάζοντας τα ενδεχ. με πράξεις συνόλων (UNION, EXCEPT, κτλ)
 - Κάποτε η εμφώλευση είναι πιο βολική (δεν ισχύει πιο κάτω)
- Ας δούμε πως αναπαριστάται η προηγούμενη επερώτηση
- **Query 12:** Βρες το όνομα κάθε employee που έχει ένα dependent με το ίδιο όνομα με τον employee.

- **Χωρίς Εμφώλευση**

```
Q12A:  SELECT      E.FNAME, E.LNAME
        FROM        EMPLOYEE E, DEPENDENT D
        WHERE       E.SSN=D.ESSN AND
                   E.FNAME=D.DEPENDENT_NAME
```

- **Με Εμφώλευση**

```
Q12: SELECT  E.FNAME, E.LNAME
           FROM EMPLOYEE E
           WHERE E.SSN IN
                (SELECT D.ESSN FROM DEPENDENT D
                 WHERE D.ESSN=E.SSN AND
                       E.FNAME = D.DEPENDENT_NAME)
```

Εμφωλευμένες Επερωτήσεις και η Εντολή **EXISTS**



- Η εντολή **EXISTS** επιστρέφει **TRUE** εάν το αποτέλεσμα μιας εμφωλευμένης επερώτησης **υπάρχει** (**ΔΕΝ** είναι κενό)

WHERE [NOT] EXISTS subquery

- Μια επερώτηση με **EXISTS** μπορεί να διατυπώνεται και με άλλους τρόπους (π.χ., με **IN**, **Ενός-block**, κτλ).
- **Query 12b:** Βρες το όνομα κάθε employee που έχει ένα dependent με το ίδιο όνομα με τον employee.

```
Q12B: SELECT      E.FNAME, E.LNAME
          FROM      EMPLOYEE E
          WHERE     EXISTS (SELECT *
                              FROM    DEPENDENT D
                              WHERE   E.SSN=D.ESSN   AND
                                      E.NAME = D.DEPENDENT_NAME)
```

- Το **EXISTS** έχει αντίστοιχη λογική με τον υπαρξιακό ποσοδείκτη $\exists x: P(x)$ που χρησιμοποιείται στον ορισμό συνόλων $\{ x \mid \dots \}$

Εμφωλευμένες Επερωτήσεις και η Εντολή **NOT EXISTS**



- **Query 6 (Παράδειγμα NOT EXISTS):** Βρες τα ονόματα των employees που ΔΕΝ έχουν dependents.

```
Q6:      SELECT      E.FNAME, E.LNAME
          FROM        EMPLOYEE E
          WHERE       NOT EXISTS
                    (SELECT      *
                     FROM        DEPENDENT D
                     WHERE       E.SSN=D.ESSN)
```

Επισημάνσεις:

- Προσέξτε ότι η EXISTS αναφέρεται συνήθως σε **συσχετιζόμενες** εμφωλευμένες επερωτήσεις.
- Εναλλακτικά η συνθήκη αποτίμησης θα ήταν κάτι σταθερό.¹⁵⁻²³

Εμφωλευμένες Επερωτήσεις και η Εντολή EXISTS



- **Query 6:** Βρες το SSN των employees χωρίς dependents.

Q6a: Με 1-block Queries και Except

```
(SELECT E.SSN FROM EMPLOYEE E) // ALL  
EXCEPT
```

```
(SELECT DISTINCT D.ESSN FROM Dependent D) // WITH DEPENDENT
```

Το DISTINCT (πάνω σε non-key) μειώνει τον αριθμό των ενδιάμεσων αποτελεσμάτων

Q6b: Με Χρήση του IN

```
SELECT E.SSN FROM EMPLOYEE E  
WHERE E.SSN NOT IN (  
    SELECT DISTINCT D.ESSN  
    FROM DEPENDENT D)
```

EMPLOYEE									
Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPENDENT				
<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship

Q6c: Με Χρήση EXISTS

```
SELECT E.SSN FROM EMPLOYEE E  
WHERE NOT EXISTS (  
    SELECT *  
    FROM DEPENDENT D  
    WHERE D.ESSN=E.SSN)
```

Μπορεί να διατυπωθεί το Query σε 1 block χωρίς πράξη συνόλων;

Διαίρεση σε SQL



- Παρόλο που οι αρχικές εκδόσεις της SQL όριζαν **εξειδικευμένη εντολή διαίρεσης**, την **CONTAINS**, μια τέτοια δυνατότητα **ΔΕΝ** υπάρχει στα **νεότερα πρότυπα** και **υλοποιήσεις**:
 - Ενδεχόμενοι **Λόγοι Εγκατάλειψης** του CONTAINS:
 - Δυσκολία **Αποδοτικής Υλοποίησης**
 - **Μειωμένη Χρήση** της εν λόγω εντολής
 - Στα πλαίσια αυτού του μαθήματος θεωρήστε ότι **ΔΕΝ** υπάρχει η εντολή διαίρεσης CONTAINS
- Για να **διαιρέσουμε δυο σχέσεις** θα χρησιμοποιήσουμε την λογική του **NOT EXISTS EXCEPT** που ακολουθεί.
- Σημειώστε ότι σε **TSQL**, η εντολή **CONTAINS** χρησιμοποιείται για κάτι διαφορετικό ... την **αναζήτηση σε πεδία κειμένου (όπως η LIKE)**

Παράδειγμα Διαίρεσης σε SQL



- Query 3: Βρες το όνομα κάθε employee που δουλεύει σε ΚΑΘΕ project που ελέγχεται από το department number 5.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

```
Q3: SELECT E.FNAME, E.LNAME
      FROM EMPLOYEE E
      WHERE NOT EXISTS ((
        (SELECT P.Pnumber
         FROM PROJECT P
         WHERE P.Dnum=5)
        EXCEPT
        (SELECT W.PNO
         FROM WORKS_ON W
         WHERE E.SSN=W.ESSN)
```

ALL Projects by
Department 5

ALL Projects of
Current Employee
(outer block)

Παράδειγμα Διαίρεσης



Suppliers(sid, sname, address)

sid	sname	Address
s1	Supplier 1	Address 1
s2	Supplier 2	Address 2
s3	Supplier 3	Address 3
s4	Supplier 4	Address 4

Parts(pid, pname, color)

pid	pname	color
p1	Part 1	red
p2	Part 2	red
p3	Part 3	green
p4	Part 4	blue

Catalog(sid, pid, cost)

sid	pid	cost
s1	p1	100
s1	p2	150
s1	p3	100
s1	p4	200
s2	p1	100
s2	p2	150
s3	p2	100
s4	p2	200
s4	p4	250

Query 4: Βρες τους suppliers που πουλάνε τα parts p2 και p4

Query 4': Βρες τους suppliers που δεν υπάρχει part (μέσα στη λίστα με τα parts p2 και p4) που να μην παρέχουν

Διπλή Άρνηση → Διπλό **NOT EXISTS**

Παράδειγμα Διαίρεσης



- Q4a: SELECT **s**.sid FROM suppliers **s**
WHERE **NOT EXISTS** (
 SELECT **p**.pid FROM parts **p** WHERE pid IN ('p2', 'p4')
 AND **NOT EXISTS** (
 SELECT **c**.pid from catalog **c** WHERE **s**.sid = **c**.sid
 AND **p**.pid=**c**.pid))
- Q4b: SELECT **s**.sid FROM suppliers **s**
WHERE **NOT EXISTS** (
 SELECT **p**.pid FROM parts **p** WHERE **p**.pid IN ('p2', 'p4')
 EXCEPT
 SELECT **c**.pid FROM catalog **c** WHERE **s**.sid=**c**.sid)
- Q4c: SELECT **s**.sid FROM suppliers **s**
WHERE **NOT EXISTS** (
 SELECT **p**.pid FROM parts **p** WHERE pid IN ('p2', 'p4')
 AND pid **NOT IN** (
 SELECT pid FROM catalog **c** WHERE **s**.sid=**c**.sid))

Διαδικασιακοί Τελεστές

Διαίρεση (Division)



A

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

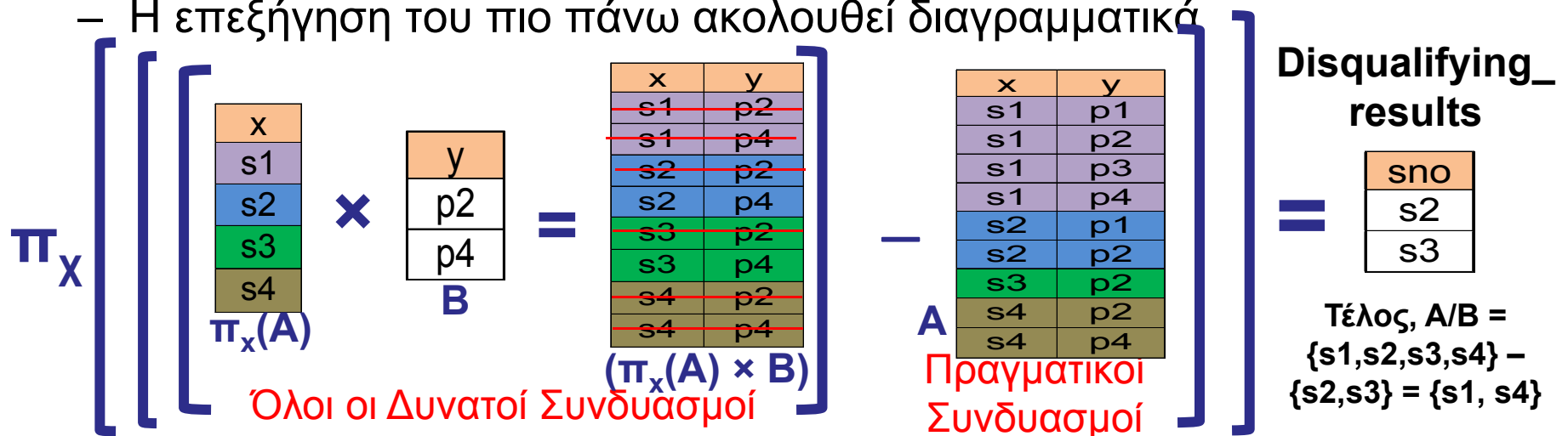
B

y
p2
p4

$$A / B = \pi_x(A) - \pi_x(\pi_x(A) \times B) - A$$

$$= \text{all_results} - \text{disqualifying_results}$$

- **all_results**: Όλες οι δυνατές πλειάδες του αποτελέσματος
 - π.χ., όλοι οι suppliers $\pi_x(A) = \{s_1, s_2, s_3, s_4\}$
- Από το **all_results** πρέπει να αφαιρέσουμε τους ακατάλληλους suppliers (**disqualifying_results**)
 - αυτούς που ΔΕΝ προσφέρουν **ΟΛΑ** τα parts $y \in B$. (δηλ., s_2, s_3)
 - **disqualifying_results** συμβολίζεται με $[\pi_x(\pi_x(A) \times B) - A]$
 - Η επεξήγηση του πιο πάνω ακολουθεί διαγραμματικά



Παράδειγμα Διαίρεσης



Suppliers(sid, sname, address) Parts(pid, pname, color) Catalog(sid, pid, cost)

Q4d: SELECT sid FROM catalog $\pi_x(A)$

EXCEPT —

SELECT sid FROM ($\pi_x((\pi_x(A) \times B) - A)$

SELECT DISTINCT sid, Q1.pid

FROM catalog, $\pi_x(A) \times$

(SELECT pid FROM parts **B**

WHERE pid IN ('p2', 'p4')) Q1]

$(\pi_x(A) \times B)$

$(\pi_x(A) \times B) - A$

EXCEPT —

SELECT sid, pid FROM catalog **A**

) Q2

$$A \div B = \pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

Παρατηρήσεις



- Η διαίρεση μπορεί να επιλυθεί και με JOINS σε συνδυασμό με συναθροιστικές συναρτήσεις (δες επόμενη διάλεξη)
- Απλό παράδειγμα με JOIN και COUNT

```
Q4e: SELECT sid FROM catalog c
      JOIN ( SELECT pid FROM parts
            WHERE pid IN ('p2', 'p4') ) q
      ON c.pid=q.pid
      GROUP BY sid
      HAVING COUNT(c.pid)=(
            SELECT COUNT(*) FROM parts WHERE pid IN ('p2','p4')
      )
```