



EPL342 –Databases

Lecture 17: SQL DML IV

SQL Structured Query Language

(Chapter 6.5.5-6.6, Elmasri-Navathe 7ED)

Demetris Zeinalipour

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



Περιεχόμενο Διάλεξης

Ολοκλήρωση Διάλεξης 16.

Κεφάλαιο 6.5.5-6.6: SQL DML III

- *Εντολή Μετονομασίας **AS** σε SQL*
- *Προχωρημένες Συνενώσεις σε SQL (**JOINS**)*
- *Συναθροιστικές Συναρτήσεις σε SQL (**COUNT, MAX, MIN, AVG, SUM**),*
- *Εντολή Ομαδοποίησης (**GROUP-BY**) και Εντολή Επιλογής-μετά-από-Ομαδοποίηση (**HAVING**) σε SQL*
- *Εντολές Εισαγωγής/Διαγραφής/Ενημέρωσης (**INSERT / DELETE / UPDATE**) σε SQL,*

Ομαδοποίηση σε SQL (GROUP BY)



- Σε πολλές περιπτώσεις θέλουμε να εφαρμόσουμε τα αποτελέσματα **συναθροιστικών συναρτήσεων** σε **υποομάδες πλειάδων** σε μια σχέση.
 - **Μέλη υποομάδας:** Πλειάδες που έχουν την ίδια τιμή στο/α γνωρίσματα ομαδοποίησης, π.χ.,
 - Παραδείγματα:
 - Εύρεση Μέσου Όρου Μισθών ανά Τμήμα (Dno).
 - Εύρεση Τμήματος/ων με Μέσο Μισθό πάνω από 50000.

ssn	Fname	Minit	Lname	Dno	salary
1	John	B	Smith	2	NULL
2	Franklin	T	Wong	5	10000
3	Alicia	J	Wong	2	40000
4	Jennifer	NULL	Zelaya	4	40000
5	Ramesh	NULL	Wallace	2	30000
6	Joyce	A	Narayan	2	20000
7	Ahmad	T	Wong	2	56000
8	James	E	Jabbar	2	15000
1	John	B	Smith	2	NULL

Ομαδοποίηση σε SQL (GROUP BY)



- Για την υλοποίηση της ομαδοποίησης σε SQL εκτελείται πρώτα η **συνάρτηση συνάθροισης σε κάθε υποομάδα ανεξάρτητα**.
 - Γίνεται χρήση της εντολής **GROUP BY**
 - Π.χ., Εύρεση Μέσου Μισθού ανά τμήμα
- Στην συνέχεια μπορεί να χρησιμοποιηθεί και ένα **φιλτράρισμα των αποτελεσμάτων βάσει της υποομάδας ομάδας**.
 - Γίνεται χρήση της εντολής **HAVING**
 - Π.χ., Φιλτράρισμα Αποτελεσμάτων Μέσων Μισθών > 50,000

ssn	Fname	Minit	Lname	Dno	salary
1	John	B	Smith	2	NULL
2	Franklin	T	Wong	5	10000
3	Alicia	J	Wong	2	40000
4	Jennifer	NULL	Zelaya	4	40000
5	Ramesh	NULL	Wallace	2	30000
6	Joyce	A	Narayan	2	20000
7	Ahmad	T	Wong	2	56000
8	James	E	Jabbar	2	15000
1	John	B	Smith	2	NULL

Εννοιολογική Εκτέλεση Ενός SQL Μπλοκ



- Ας δούμε λίγο καλύτερα με ποια **σειρά εκτελείται** ένα SQL μπλοκ σε ένα **αφαιρετικό (εννοιολογικό)** επίπεδο.
- Η **περιγραφή** αυτή είναι σε **εννοιολογικό επίπεδο**, επομένως το **πλάνο εκτέλεσης (query plan)** της επερώτησης στα παραδείγματα ΔΕΝ θα είναι βέλτιστο.
- Η **πραγματικό πλάνο εκτέλεσης** εναπόκειται αποκλειστικά στην βάση δεδομένων.
 - Αυτή η δουλειά διεκπεραιώνεται από τον **βελτιστοποιητή επερωτήσεων (query optimizer)** ο οποίος έχει μεγάλη πολυπλοκότητα.
 - *Μια υλοποίηση του σε εμπορική βάση λέγεται ότι πήρε 50 ανθρωποχρόνια εργασίας.*
- Θεωρούμε ότι η βάση μας είναι **ένας ή περισσότεροι πίνακες αποθηκευμένοι στον δίσκο χωρίς άλλες δομές δεδομένων** (ευρετήρια αναζήτησης, κτλ).
- Η **περιγραφή** αυτή θα μας επιτρέψει να **καταλάβουμε καλύτερα τι παράγεται από μια επερώτηση**.

Εννοιολογική Εκτέλεση Ενός SQL Μπλοκ



- Ένα μπλοκ επερώτησης SQL αποτελείται από έξι όρους (clauses) οι οποίοι εκτελούνται (σε λογικό επίπεδο) όπως φαίνεται πιο κάτω:
 - 6) **SELECT** <Attribute(s) AS Alias(s)> 3) Agg1 AS Alias, ...AggN as Alias
 - 1) **FROM** <table(s)>
 - 2) [**WHERE** <condition>]
 - 3) [**GROUP BY** <grouping attribute(s)>]
 - 4) [**HAVING** <group condition>]
 - 5) [**ORDER BY** <attribute list>]
- Ένα Query εκτελείται εννοιολογικά με την ακόλουθη σειρά:
 1. **FROM:** Συνένωσε (ή Καρ. Γιν) τους πίνακες του table-list
π.χ., FROM Employee E, Department D
 2. **WHERE:** Διάσχισε Γραμμικά τον Πίνακα που παράγεται στο βήμα 1 αποτιμώντας την έκφραση <condition> σε κάθε πλειάδα.
π.χ., WHERE E.Dno=D.Dnumber and D.Dname="Research"

Συνέχεια επόμενη διαφάνεια ...

Εννοιολογική Εκτέλεση Ενός SQL Μπλοκ

TEMP1		
Dno, SSN, Sal		
1, E1, 10000		
2, E2, 12000		
1, E3, 15000		
2, E4, 20000....		

Έστω το ενδιάμεσο αποτέλεσμα: **TEMP1(DNO, SSN, Salary)**

3. **GROUP BY:** Διάσχισε γραμμικά το TEMP1, ομαδοποιώντας τα αποτελέσματα βάσει του grouping attribute(s) και υπολογίζοντας για κάθε ομάδα τα αιτούμενα συναθροιστικά αποτελέσματα:

- MAX, MIN, COUNT, SUM: Υλοποιούνται με μια μεταβλητή ανά ομάδα
- AVG = SUM / COUNT

π.χ., **GROUP BY DNO**

TEMP2	
Dno, Avg(Sal)	
1, 12500	
2, 16000	

Έστω το ενδιάμεσο αποτέλεσμα: **TEMP2(DNO, AVG(salary))**

4. **HAVING:** Διάσχισε γραμμικά το TEMP2 εφαρμόζοντας το <group condition> πάνω σε κάθε πλειάδα.

π.χ., **HAVING AVG(Salary)>15000**

5. **ORDER BY:** Ταξινόμησε τα αποτελέσματα βάσει της συνθήκης

π.χ., **ORDER BY AVG(Salary)**

6. **SELECT:** Πρόβαλε τα αιτούμενα γνωρίσματα (από το TEMP2) εφαρμόζοντας τα σχετικά aliases στα απλά γνωρίσματα.

π.χ., **SELECT DNO** (θυμηθείτε ότι τα γνωρίσματα του ORDER BY δεν χρειάζεται να εμφανίζονται στο SELECT)

Ομαδοποίηση σε SQL (GROUP BY)



Query 20: Για κάθε department, ανάκτησε το department number, τον αριθμό των employees στο εν λόγω department, και τον μέσο μισθό τους.

```
Q20: SELECT DNO, COUNT (*), AVG (SALARY)
      FROM EMPLOYEE
      GROUP BY DNO
```

Πως διαφέρει από το AVG (SALARY) SUM(SALARY)/COUNT();*

• Εννοιολογική Εκτέλεση Επερώτησης:

- Η βάση εκτελεί μια γραμμική διέλευση του πίνακα Employee.
- Για κάθε πλειάδα, βρίσκει το **DNO** της πλειάδας, βάσει του οποίου αυξάνει τους ακόλουθους μετρητές:

- TUPLE_COUNT[DNO],**
- SALARY_SUM[DNO],**
- SALARY_COUNT_NONULLS[DNO],**

- Στο τέλος τυπώνει για κάθε **DNO** τα ακόλουθα: **Dno, A, B/C**

AVG(SALARY) = αγνοεί διπλότυπα

SUM(SALARY)/COUNT(); => λαμβάνει υπόψη όλες τις γραμμές στον παρονομαστή (NULL ή όχι)*

Ομαδοποίηση σε SQL (GROUP BY)



```
SELECT DNO, COUNT(*) AS Count,
       AVG(SALARY) AS AVG_Salary
FROM EMPLOYEE
GROUP BY DNO
```

*Εάν δεν υπάρχει
GROUPBY, το
SELECT μπλοκ εδώ
δεν δουλεύει (λόγω της
ύπαρξης του DNO).*

Column is invalid in the select list
because it is not contained in either an
aggregate function or the GROUP BY
clause

EMPLOYEE

ssn	Fname	Minit	Lname	Dno	salary
1	John	B	Smith	2	NULL
2	Franklin	T	Wong	5	10000
3	Alicia	J	Wong	2	40000
4	Jennifer	NULL	Zelaya	4	40000
5	Ramesh	NULL	Wallace	2	30000
6	Joyce	A	Narayan	2	20000
7	Ahmad	T	Wong	2	56000
8	James	E	Jabbar	2	15000
1	John	B	Smith	2	NULL

Για το Department DNO=2
 $(40K + 30K + 20K + 56K + 15K) / 5$

Q20a

Dno	Count	Avg_Salary
2	7	32200
4	1	40000
5	1	10000

Με NULLS

Χωρίς
NULLS

Ομαδοποίηση σε SQL (GROUP BY)



```
SELECT DNO, COUNT(*) AS Count,  
       AVG(SALARY) AS AVG_Salary  
FROM EMPLOYEE  
GROUP BY DNO, SSN
```

-- Σωστό, Χωρίς Ιδιαίτερο Νόημα ωστόσο
-- Σημειώστε ότι προβάλλεται μόνο το DNO

EMPLOYEE

Q20b

ssn	Fname	Minit	Lname	Dno	salary
1	John	B	Smith	2	NULL
2	Franklin	T	Wong	5	10000
3	Alicia	J	Wong	2	40000
4	Jennifer	NULL	Zelaya	4	40000
5	Ramesh	NULL	Wallace	2	30000
6	Joyce	A	Narayan	2	20000
7	Ahmad	T	Wong	2	56000
8	James	E	Jabbar	2	15000
1	John	B	Smith	2	NULL

Dno	Count	Avg_Sal...
2	2	NULL
5	1	10000
2	1	40000
4	1	40000
2	1	30000
2	1	20000
2	1	56000
2	1	15000

Ομαδοποίηση σε SQL (GROUP BY)



- **Επισημάνσεις:**

1. Το SELECT περιλαμβάνει Aggregates (MIN, MAX, κτλ) ή γνωρίσματα που εμφανίζονται και στο GROUP BY, π.χ.,
 - **ΟΡΘΟ(χωρίς ιδιαίτερη λογική):** SELECT **dno** FROM EMPLOYEE GROUP BY **dno, ssn** -- προηγούμενο παράδειγμα
 - **ΟΡΘΟ(χωρίς ιδιαίτερη λογική):** SELECT **dno, ssn** FROM EMPLOYEE GROUP BY **dno, ssn**
 - **ΛΑΘΟΣ:** SELECT **dno, ssn** FROM EMPLOYEE GROUP BY **dno** -- αυτό διότι το **ssn** δεν προβάλλεται στο ενδιάμεσο αποτέλεσμα του Group By

2. Εάν υπάρχει **NULL** στο **γνώρισμα ομαδοποίησης** τότε τα ομαδοποιημένα αποτελέσματα εμφανίζονται σε 1 μια νέα επιπλέον ομάδα (ANSI_NULLS OFF)

* **ANSI_NULLS = ON: Πολλαπλά** NULL ομάδες

salary	Count
NULL	2
10000	1
15000	1
20000	1
30000	1
40000	2
56000	1

Ομαδοποίηση σε SQL (GROUP BY)



- Query 21: Για κάθε project, ανάκτησε το project number, project name, και τον αριθμό των employees που δουλεύουν πάνω στο εν λόγω project.

WORKS_ON

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

PROJECT

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

```
Q21:      SELECT      PNUMBER, PNAME, COUNT (*)
           FROM        PROJECT, WORKS_ON
           WHERE       PNUMBER=PNO
           GROUP BY    PNUMBER, PNAME
```

- **Εξάσκηση:** Περιγράψτε εννοιολογικά πως θα εκτελεστεί η πιο πάνω επερώτηση (με βάσει την λογική που περιγράψαμε νωρίτερα)

Επιλογή μετά από Ομαδοποίηση (HAVING)



- Ο όρος **HAVING**, χρησιμοποιείται όταν θέλουμε να επιλέξουμε ένα **υποσύνολο** ενός ομαδοποιημένου αποτελέσματος βάσει συνθήκης.
- Π.χ., επιλογή των DNO με μέσο μισθό πάνω από 35,000

Dno	Count	Avg_Salary
2	7	32200
4	1	40000
5	1	10000

Επισημάνσεις

- Το **HAVING** έχει αντίστοιχη λειτουργία με το **WHERE**, μόνο που η **συνθήκη επιλογής** είναι πάνω σε **ομάδες** παρά σε επί μέρους **πλειάδες**.
- Τα γνωρίσματα που εμφανίζονται στο **HAVING** πρέπει οπωσδήποτε να εμφανίζονται στο **GROUP BY** (θυμηθείτε τα βήματα της εννοιολογικής εκτέλεσης επερωτήσεων)
 - Π.χ., **HAVING AVG(Salary)>35000**

Επιλογή μετά από Ομαδοποίηση (HAVING)



- **Query 22:** Για κάθε **project** πάνω στο οποίο δουλεύουν περισσότεροι από **2 employees**, ανάκτησε το **project number**, **project name**, και τον **αριθμό των employees** που δουλεύουν πάνω στο project αυτό.

WORKS_ON		
<u>Essn</u>	<u>Pno</u>	Hours

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

*Can DNO=2 be added to HAVING?
YES! Why wouldn't we do that?*

We want to filter early on in the WHERE ...

```
Q22: SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT (*) > 2
```

Ο βασικός λόγος που κάνουμε group by βάσει και του PNAME είναι επειδή θέλουμε το PNAME στο αποτέλεσμα του SELECT, εναλλακτικά δεν υπάρχει λόγος ύπαρξης του στο GROUP BY

Executes once, not multiple times ... could be used with COUNT() as CNT; then CNT>2*

Αλλάζοντας την Κατάσταση μιας Βάσης με SQL



- Μέχρι στιγμής είδαμε τα ακόλουθα:
 - Ορισμός Δομής Βάσης με **SQL-DDL (CREATE)**
 - Μεταβολή Δομής Βάσης με **SQL-DDL (ALTER/DROP)**
 - Ορισμός Επερωτήσεων με **SQL-DML (SELECT...)**
- Ήρθε η ώρα να δούμε πως μπορούμε να **μεταβάλουμε την κατάσταση μιας βάσης με εισαγωγές, διαγραφές, και ενημερώσεις πλειάδων σε SQL-DML.**
- Θα μελετήσουμε τις εντολές **INSERT, DELETE, and UPDATE** της SQL:99 αλλά και κάποιες εξειδικευμένες εντολές της **TSQL.**

Εισαγωγή Δεδομένων στην SQL (INSERT)



- Η εντολή **INSERT** εισάγει μια ή περισσότερες πλειάδες σε μια υφιστάμενη σχέση

```
INSERT [INTO] <table-name> [(<column-list>)]  
VALUES (<data values>) [, (<data values>)] [, ...n];
```

- Π.χ., **INSERT INTO EMPLOYEE(surn,name,ssn,age)
VALUES ('Name', 'Lastname', 748797, 34);**

- **Επισημάνσεις:**

- Εάν δεν οριστεί το (**<column-list>**) τότε πρέπει:

1. Να οριστούν τα (**<data values>**) για **ΌΛΑ** τα γνωρίσματα, **ΚΑΙ**

2. Να δοθούν τα (**<data values>**) με την ίδια σειρά όπως αυτά ορίστηκαν κατά την **CREATE TABLE** εντολή

- Π.χ., **INSERT INTO EMPLOYEE VALUES (748797, 'Name','Lastname',34);**

Εισαγωγή Δεδομένων στην SQL (INSERT)



- **Εισαγωγή Τιμών για Συγκεκριμένα Πεδία:**
 - Εάν το επιτρέπουν οι **περιορισμοί** του πίνακα, τότε μπορεί να γίνει **εισαγωγή υποσυνόλου πεδίων**, π.χ.,
 - **INSERT INTO EMPLOYEE(ssn) VALUES (7);**
 - Σε αυτές τις περιπτώσεις τα υπόλοιπα πεδία **είτε** μένουν **NULL** ή αυτά παίρνουν τις αντίστοιχες **DEFAULT** τιμές τους.
- **Επισημάνσεις για TSQL:**
 - Πεδία τύπου **IDENTITY** σε TSQL ΔΕΝ πρέπει να μην δηλώνονται ρητά στο INSERT εφόσον αυτά θα συμπληρωθούν αυτόματα.
 - `INSERT INTO dbo.Tab(name) VALUES ('SomeName')` -- Εισαγωγή Δεδομένων.
 - `SELECT @@IDENTITY;` -- Εξέγερση Τελευταίας Εισαχθείσας IDENTITY (@@ : system function)
 - **Εισαγωγή Πολλαπλών Εγγραφών:** Η TSQL'08 (ΟΧΙ σε παλιότερες εκδόσεις) προσφέρει την δυνατότητα εισαγωγής πολλαπλών πλειάδων με μια εντολή INSERT, δηλ.,
INSERT INTO EMPLOYEE VALUES
('a','b',7,34), ('a','b',8,30), ('a','b',9,24), ('a','b',10,22);

Εισαγωγή Δεδομένων στην SQL (INSERT)



- **Παράδειγμα:** Υποθέστε ότι θέλετε να δημιουργήσετε ένα **νέο πίνακα** ο οποίος θα έχει για κάθε department το **name**, **number of employees**, και το **άθροισμα των salaries**.

- **Βήμα Α: Δημιουργία Πίνακα DEPTS_INFO**

```
U3A: CREATE TABLE DEPTS_INFO
      (DEPT_NAME          VARCHAR(10),
       NO_OF_EMPS        INTEGER,
       TOTAL_SAL         INTEGER);
```

- **Βήμα Β: Εισαγωγή Δεδομένων στον DEPTS_INFO**

```
U3B: INSERT INTO DEPTS_INFO (DEPT_NAME,
                             NO_OF_EMPS, TOTAL_SAL)
      SELECT DNAME, COUNT (*), SUM (SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME;
```

Σημείωση: Εδώ το INSERT είναι ολόκληρο SELECT μπλοκ

Εισαγωγή Δεδομένων στην SQL (INSERT)



- Σημειώστε ότι στο προηγούμενο παράδειγμα, ο πίνακας DEPTS_INFO **ΔΕΝ** θα είναι ενημερωμένος σε περίπτωση που μεταβάλλεται είτε ο πίνακας DEPARTMENT ή ο πίνακας EMPLOYEE.
- Σε ερχόμενες διαλέξεις θα μάθουμε την έννοια των **Όψεων (Views)** τα οποία μπορούν να διατηρούν «ενημερωμένο» ένα πίνακα που ορίζεται βάσει άλλων πινάκων.

Μαζική Εισαγωγή Δεδομένων σε TSQL (BULK INSERT)



- Μια εξαιρετικά χρήσιμη εντολή είναι η **BULK INSERT** η οποία εισάγει **μαζικά** δεδομένα από αρχεία κειμένου (αντίστοιχα η **BCP (windows command line utility)** εξαγάγει μαζικά δεδομένα):
 - Π.χ., `bcp "SELECT FirstName, LastName FROM TABLE ORDER BY LastName" queryout Contacts.txt -c -T`
- **Στον SQL Server:**
 - **BULK INSERT** OrdersBulk **FROM** 'c:\data.csv'
WITH (FIRSTROW=2, FIELDTERMINATOR=',', ROWTERMINATOR='\n')

Υποθέστε ότι έχουν προηγηθεί τα ακόλουθα:

 - **CREATE TABLE** OrdersBulk(CustomerID INT, CustomerName VARCHAR(32), OrderID INT, OrderDate SMALLDATETIME)
 - Το **data.csv** θεωρήστε ότι έχει την ακόλουθη δομή:

```
# CustomerID,CustomerName,OrderID,OrderDate
1,foo,5,20031101
3,blat,7,20031101
5,foobaz,22,20021101
```

BULK Insert: <http://msdn.microsoft.com/en-us/library/ms188365.aspx>

Μαζική Εισαγωγή Δεδομένων σε TSQL (OPENROWSET)



Η **OPENROWSET** επιτρέπει να αντλήσουμε δεδομένα από μια άλλη πηγή (π.χ., txt αρχείο, άλλη OLE-DB βάση δεδομένων, κτλ)

Παράδειγμα join με πίνακα άλλης βάσης

```
USE Northwind ;
GO
SELECT c.*, o.*
FROM Northwind.dbo.Customers AS c
  INNER JOIN OPENROWSET('Microsoft.Jet.OLEDB.4.0',
  'C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb';'admin';",
Orders)
  AS o
  ON c.CustomerID = o.CustomerID ;
GO
```

Μαζική Εισαγωγή Δεδομένων σε TSQL (OPENROWSET)



Η **OPENROWSET** επιτρέπει να αντλήσουμε δεδομένα από μια άλλη πηγή (π.χ., txt αρχείο, άλλη OLE-DB βάση δεδομένων, κτλ)

Παράδειγμα import από txt αρχείο.

This file can be described by the following format file (**data.format**):

9.0

4

1 SQLCHAR 0 40 "\t" 1 A SQL_Latin1_General_CP1_CI_AS

2 SQLCHAR 0 10 "\t" 2 B "" *numeric*

3 SQLCHAR 0 40 "\t" 3 C SQL_Latin1_General_CP1_CI_AS

4 SQLCHAR 0 40 "\r\n" 4 D SQL_Latin1_General_CP1_CI_AS

National language string

```
SELECT A,B,C,D
```

```
FROM OPENROWSET(BULK N'C:\Desktop\data.txt',  
FORMATFILE = 'C:\Desktop\data.format', FIRSTROW  
= 2,ROWS_PER_BATCH = 1) AS Document;
```

Διαγραφή Δεδομένων στην SQL (DELETE)



- Η εντολή **DELETE** διαγράφει πλειάδες από μια σχέση βάσει κάποιας ορισμένης συνθήκης:

```
DELETE [FROM] <table-name>  
[WHERE <condition>]
```

- Π.χ., DELETE FROM EMPLOYEE where Dno=5;
- **Επισημάνσεις**
 1. Εάν δεν οριστεί ο όρος WHERE, τότε σβήνονται **ΟΛΑ** ολα τα δεδομένα (πλειάδες) μιας σχέσης.
 - Π.χ., «DELETE FROM EMPLOYEE» (αντίστοιχο αποτέλεσμα με την TRUNCATE EMPLOYEE μόνο που θα καταγράφεται ένα log record για κάθε διαγραφή)
 - Το **DROP** από την άλλη σβήνει τόσο τα **δεδομένα** όσο και το σχήμα της βάσης από τον **κατάλογο του συστήματος**.

Διαγραφή Δεδομένων στην SQL (DELETE)



- **Επισημάνσεις**

2. Οι **κανόνες αναφορικής ακεραιότητας** επιβάλλονται από την βάση κατά την διαγραφή

- π.χ., το σύστημα απαγορεύει την διαγραφή ενός **EMPLOYEE** που αναφέρεται από την σχέση **WORKS_ON**.

3. Δεν υπάρχει η έννοια της **διαγραφής** από **πολλαπλούς πίνακες**. Κάθε διαγραφή **αναφέρεται σε 1 πίνακα**

```
U4A: DELETE FROM EMPLOYEE
      WHERE LNAME='Brown'
```

– Ωστόσο, εάν ορίζονται **Εντολές Ενεργοποίησης Αναφοράς**, τότε μια διαγραφή μπορεί να προκαλέσει μια ή περισσότερες αλυσιδωτές διαγραφές

- π.χ., εάν έχουμε **ON DELETE CASCADE** στην **WORKS_ON(ssn)** (με αναφορά στον **EMPLOYEE(ssn)**) τότε η διαγραφή ενός **EMPLOYEE** σβήνει και την αντίστοιχη πλειάδα από την **WORKS_ON**.

Διαγραφή Δεδομένων στην SQL (DELETE)



- Παράδειγμα διαγραφής μέσω Εμφωλευμένης Επερώτησης: Διάγραψε όλους τους Employees που δουλεύουν στο τμήμα του Research.
 - Θυμηθείτε ότι Employee(...,Dno) και Department(Dno,Dname,...)

```
U4C:  DELETE FROM EMPLOYEE
      WHERE DNO IN (
          SELECT DNUMBER
          FROM DEPARTMENT
          WHERE DNAME='Research'
      )
```

Με το IN αποφεύγουμε την συνένωση που δεν είναι εφικτή στα πλαίσια του DELETE

Ενημέρωση Δεδομένων στην SQL (UPDATE)



- Η εντολή **UPDATE** χρησιμοποιείται για να **ενημερώνει** την τιμή **προκαθορισμένων γνωρισμάτων** μιας **σχέσης** βάσει κάποιας συνθήκης.

UPDATE <table-name>

SET <column>=<value> [,<column>=<value>]

[**WHERE** <condition>]

Π.χ., UPDATE Employee SET Dno=5 WHERE sex='M';

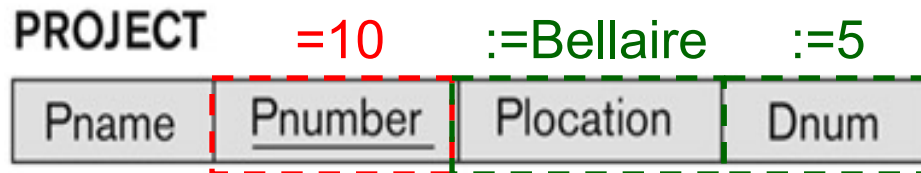
Επισημάνσεις:

- Το **WHERE** χρησιμοποιείται για να **προσδιοριστεί το σύνολο** εγγραφών που πρέπει να **ενημερωθεί**
- Το **SET** προσδιορίζει τα **γνωρίσματα** που πρέπει να ενημερωθούν και την νέα τους τιμή.
- Η ενημέρωση γίνεται σε 1 σχέση (όχι πολλές) ταυτόχρονα.
- **Οι κανόνες ακεραιότητας επιβάλλονται αυτόματα**

Ενημέρωση Δεδομένων στην SQL (UPDATE)



- **Παράδειγμα:** Ενημέρωσε το **PLocation** του Project με αριθμό **10** σε **'Bellaire'**. Επίσης, ενημέρωσε το **Dnum** του εν λόγω project number σε 5.



```
U5:      UPDATE PROJECT
         SET PLOCATION = 'Bellaire', DNUM=5
         WHERE PNUMBER=10
```

Ενημέρωση Δεδομένων στην SQL (UPDATE)



- **Σημείωση:** Το UPDATE μπορεί να δημιουργηθεί βάσει πολλών πινάκων αλλά η **τελική ενημέρωση** γίνεται μόνο σε 1 πίνακα.

- **Παράδειγμα:** Δώσε σε όλους τους employees στο 'Research' department μια αύξηση 10% στον μισθό.

```
U6:      UPDATE      EMPLOYEE
          SET         SALARY = SALARY *1.1
          WHERE      DNO IN
                    (SELECT DNUMBER
                     FROM   DEPARTMENT
                     WHERE  DNAME='Research')
```

Νέο salary → Παλαιό salary

Και πάλι, με το IN αποφεύγουμε την συνένωση που δεν είναι εφικτή στα πλαίσια της UPDATE

Ενημέρωση Δεδομένων στην SQL (INSERT/UPDATE/DELETE)



Γενικές Επισημάνσεις

- Μια **Ενημέρωση (UPDATE)** αποτυγχάνει εάν παραβιαστούν **κανόνες** και **περιορισμοί** που έχουν οριστεί, δηλ.,
 - Περιορισμοί, Αναφορικής Ακεραιότητας, CHECK, NULL, συμβατότητα-προς-τον-τύπο, κτλ.
 - Σε αυτές τις περιπτώσεις **ακυρώνονται ΟΛΕΣ** οι ενημερώσεις μέχρι το τελευταίο GO (όχι μόνο αυτή που προκάλεσε την παραβίαση).
- Η Ενημέρωση (UPDATE) **Primary Keys** πρέπει να γίνεται με μεγάλη **προσοχή** εφόσον αυτή ενδέχεται να προκαλέσει **αλυσιδωτές αλλαγές** στην **κατάσταση της βάσης**.
- Υπάρχει και το **UPSERT** ως συνδυασμός της **UPDATE + INSERT** σε μια ατομική πράξη.
 - UPSERT: Ενημερώνει ή εισάγει ταυτόχρονα, το οποίο είναι βολικό σε περιπτώσεις που δεν είμαι βέβαιος εκ των προτέρων.
 - Αναφέρετε ένα σενάριο όπου αυτό είναι χρήσιμο.

Γλώσσα Ορισμού Ελέγχων

Data Control Language (DCL)



- **GRANT:** Δίνει πρόσβαση σε κάποιο αντικείμενο της βάσης δεδομένων σε κάποιους χρήστες ή ρόλους
 - GRANT [INSERT | UPDATE | DELETE]
ON [TABLE | PROCEDURE | <object> TO [USER | ROLE];
- **DENY:** Απαγορεύει πρόσβαση σε κάποιο αντικείμενο της βάσης δεδομένων σε κάποιους χρήστες ή ρόλους
 - DENY [INSERT | UPDATE | DELETE]
ON [TABLE | PROCEDURE | <object> TO [USER | ROLE];
- **REVOKE:** Αναστέλλει την τελευταία εντολή πρόσβασης ή απαγόρευσης

*Χρήση μόνο στους
προσωπικούς σας
υπολογιστές*

SQL-DCL



```
40
41 -- GRANT
42 GRANT SELECT, INSERT, UPDATE, DELETE ON Employees TO almir
43
44 -- REVOKE
45 REVOKE INSERT ON Employees TO almir
46         christodoulos constantinides
47
48 -- DENY
49 DENY UPDATE ON Employees TO almir
```