



Declarative SQL vs. PL SQL

Database System Concepts, 5th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use





Declarative SQL

- Δηλωτικές ή μη διαδικαστικές (Declarative or nonprocedural) – Ο χρήστης δηλώνει ποια δεδομένα απαιτούνται χωρίς να τον ενδιαφέρει ο τρόπος που θα ανακτηθούν αυτά τα δεδομένα (SQL)

select P=Predicate (κατηγορημα)
 from P= Συνδυασμός από λογικές και αριθμητικές εκφράσεις
 where P P=Εμφωλιασμένο Υποερώτημα

Γενική δομή:

select ...
 from ...
 where

? (select ...
 from ...
 where ...);

υποερώτηση



- ? : exists/non exists
- ? : In/ Not In
- ? : Some, Any, All
- ? : Unique / Non Unique

.....

Insert into R
 Select from
 where P

Update R
 where P

Delete R
 where P

Υπολογισμός της υποερώτησης για κάθε γραμμή (πλειάδα) της εξωτερικής ερώτησης





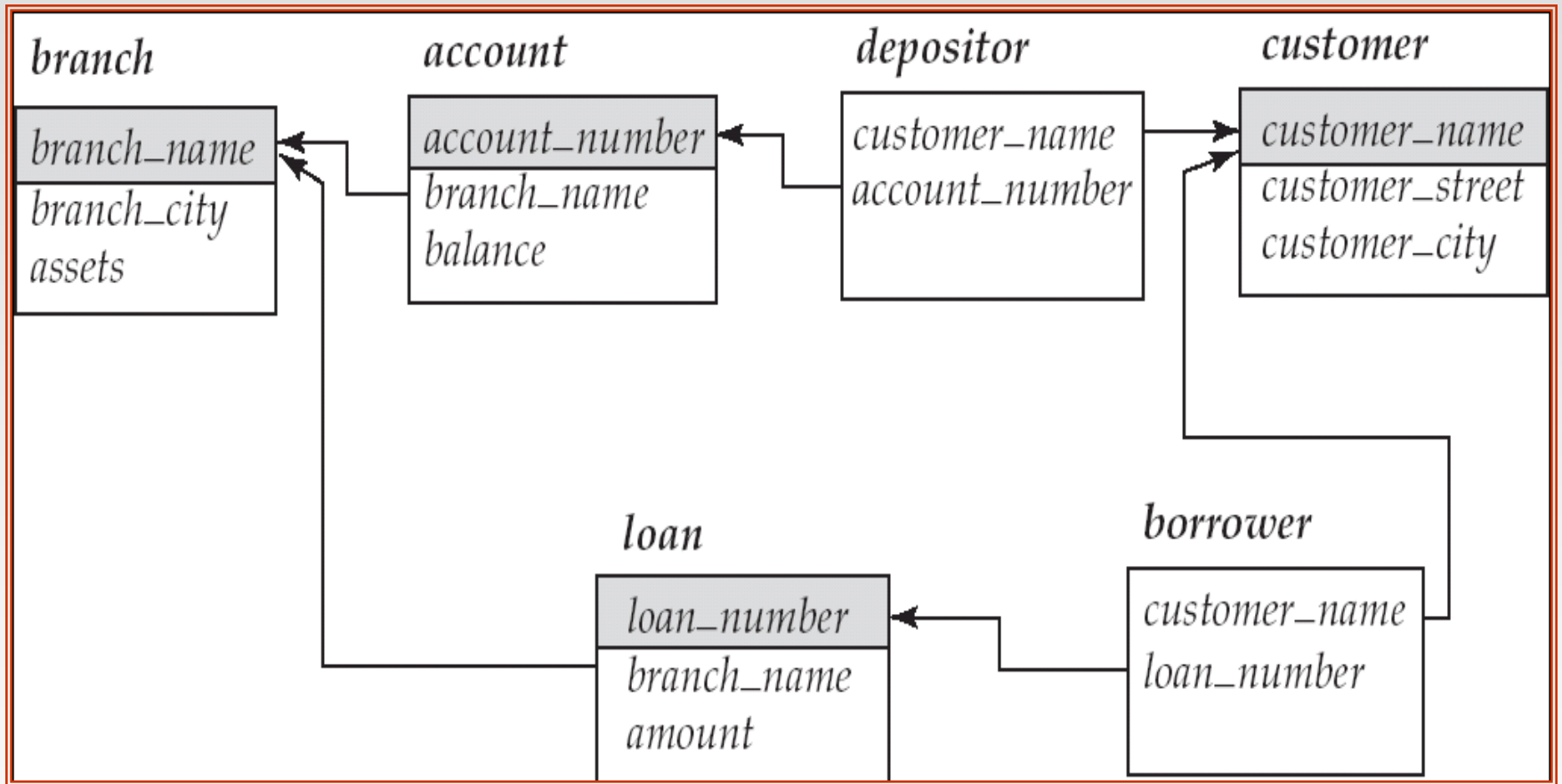
PL SQL

- Σκανδάλες (Triggers)
- Ενσωματωμένη (Embedded) SQL
- Δυναμική (Dynamic) SQL
- Συναρτήσεις και Διαδικαστικές Δομές (Functions and Procedural Constructs**)
- Αναδρομικά Ερωτήματα (Recursive Queries**)
- Προχωρημένα Θέματα SQL**





PL SQL





Trigger (Ειδοποίηση)

- “Υποθέστε ότι ο Jones κάνει αναλήψεις από κάποιο λογαριασμό που έκανε αρνητικό το υπόλοιπο του λογαριασμού του. Ας πούμε ότι το t δηλώνει την εγγραφή του λογαριασμού με αρνητική τιμή balance”
- Οι ενέργειες που θα πρέπει να κάνει αυτόματα το τραπεζικό σύστημα είναι οι εξής:
 - (1) Να εισαχθεί μία νέα εγγραφή s στη σχέση loan με
 - s[loan_number]=t[account_number]
 - s[branch_name]=t[branch_name]
 - s[amount]= - t[balance].....σιγά που δεν θα τον άφηνε να τραβήξει χρήματα...του τα χρεώνει τώρα σαν δάνειο.....
 - (2) Εισάγουμε μια νέα εγγραφή u στη σχέση borrower με
 - u[customer_name]=“Jones”
 - u[loan_number]=t[account_number]
 - (3) Ορίζουμε στον πίνακα account το t[balance]=0





Trigger (Ειδοποίηση) (Συν.)

- **Create trigger** overdraft-trigger **on** account
for update
as
if inserted.balance < 0
begin
 insert into borrower
 (**select** customer_name, account_number
 from depositor, **inserted**
 where inserted.account_number=depositor.account-number);
 insert into loan **values**
 (**inserted**.account_number, **inserted**.branch_name, - **inserted**. balance);
 update account **Set** balance=0
 where account.account_number=**inserted**.account-number)

Κώδικας Trigger σε MS-SQL Server





Ενσωματωμένη (Embedded) SQL

- Το SQL πρότυπο (standard) ορίζει ενσωματώσεις της γλώσσας SQL σε μία πλειάδα γλωσσών προγραμματισμού όπως C, Java, και Cobol.
- Η γλώσσα στην οποία τα SQL queries ενσωματώνονται είναι γνωστή με το όνομα **host language**, και οι SQL δομές που επιτρέπονται μέσα σε μία host language αποτελούν την επανομαζόμενη *embedded* SQL.
- Η βασική μορφή αυτών των γλωσσών ακολουθεί το System R το οποίο ενσωματώνει SQL μέσα σε PL/I.
- Η **EXEC SQL** εντολή χρησιμοποιείται για να δηλώσει στον προεπεξεργαστή (preprocessor) αίτημα ενσωματωμένης SQL

EXEC SQL <embedded SQL statement > END_EXEC

Σημείωση: Αυτό διαφοροποιείται από γλώσσα σε γλώσσα (για παράδειγμα, στη γλώσσα Java η ενσωμάτωση (embedding) γίνεται ως εξής:

```
# SQL { .... }; )
```





Example Query

- Μέσα από μία *host language*, βρες τα ονόματα και τις πόλεις των πελατών που έχουν σε κάποιο λογαριασμό υπόλοιπο μεγαλύτερο από την τιμή της μεταβλητής *amount*.
- Προδιαγράφουμε το query σε SQL και δηλώνουμε έναν *cursor* γι' αυτό
EXEC SQL

```
declare c cursor for  
select customer_name, customer_city  
from depositor, customer, account  
where depositor.customer_name = customer.customer_name  
       and depositor.account_number = account.account_number  
       and account.balance > :amount  
  
END_EXEC
```





Embedded SQL (Συν.)

- Η εντολή **open** προκαλεί την έναρξη εκτέλεσης του ερωτήματος
`EXEC SQL open c END_EXEC`

- Η εντολή **fetch** τοποθετεί τις τιμές ενός tuple του αποτελέσματος σε μεταβλητές της οικοδέσποινας γλώσσας (host language).

`EXEC SQL fetch c into :cn, :cc END_EXEC`

Επαναλαμβανόμενες κλήσεις της **fetch** τοποθετούν ένα-ένα τα tuples του αποτελέσματος

- Μία δεσμευμένη μεταβλητή SQLSTATE (της κλάσης SQLCA) παίρνει την τιμή '02000' για να υποδηλώσει ότι δεν υπάρχουν διαθέσιμα άλλα δεδομένα για να γίνουν fetch, δηλαδή να φορτωθούν στις μεταβλητές *cn* και *cc*.
- Η εντολή **close** προκαλεί στο database σύστημα να διαγράψει την προσωρινή σχέση η οποία έχει αποθηκεύσει το αποτέλεσμα του ερωτήματος.

`EXEC SQL close c END_EXEC`

Σημ: Οι παραπάνω λεπτομέρειες ποικίλουν από γλώσσα σε γλώσσα. Π.χ., στην Java δεν χρειάζονται επαναλαμβανόμενες κλήσεις της *fetch*, αφού η ίδια η γλώσσα ορίζει αυτόματα ειδικούς Java iterators που κάνουν την παραπάνω δουλειά.





Ενημερώσεις δια μέσω Cursors

- Μπορούμε να ενημερώσουμε το tuple που γίνεται fetched από τον cursor δηλώνοντας ότι ο cursor αυτός είναι για update:

```
declare c cursor for  
  select *  
  from account  
  where branch_name = 'Perryridge'  
for update
```

- Για να ενημερώσουμε το tuple στην τρέχουσα θέση του cursor *c*, γράφουμε:

```
update account  
  set balance = balance + 100  
  where current of c
```





Δυναμική SQL

- Επιτρέπει σε προγράμματα να δομούν και να αποστέλλουν SQL queries κατά το χρόνο εκτέλεσής τους (at run time).
- Παράδειγμα χρήσης δυναμικής SQL μέσα από πρόγραμμα C.

```
char * sqlprog = "update account  
                set balance = balance * 1.05  
                where account_number = ?"
```

```
EXEC SQL prepare dynprog from :sqlprog;
```

```
char account [10] = "A-101";
```

```
EXEC SQL execute dynprog using :account;
```

- Ένα δυναμικό SQL πρόγραμμα περιέχει ένα ?, ο οποίος είναι ένας place holder για μία τιμή (value) η οποία μας προμηθεύεται όταν ένα SQL πρόγραμμα εκτελείται.
- *** Dynamic SQL is a programming technique that enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL because **the full text of a SQL statement may be unknown at compilation. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.**





Διαδικαστικές Επεκτάσεις & Stored Procedures

- Η SQL παρέχει μία **module (transact SQL)** γλώσσα η οποία:
 - Επιτρέπει τον ορισμό procedures στην SQL, με if-then-else εντολές, for και while επαναλήψεις (loops), κ.τ.λ.

- Οι Stored Procedures
 - Μπορούν να αποθηκεύσουν *διαδικασίες* μέσα σε μία ΒΔ
 - Και εν συνεχεία να τις εκτελέσουν χρησιμοποιώντας μία **call** εντολή
 - Επιτρέπουν επίσης σε εξωτερικές εφαρμογές να συνεργάζονται με τη ΒΔ χωρίς να χρειάζεται να γνωρίζουν εσωτερικές λεπτομέρειες της ΒΔ.





Functions και Procedures

- Η SQL:1999 υποστηρίζει functions και procedures
 - Οι Functions/procedures μπορεί να αναπτυχθούν είτε σε SQL ή σε μία εξωτερική γλώσσα προγραμματισμού
 - Οι Functions είναι ιδιαίτερα χρήσιμες με ειδικούς τύπους δεδομένων όπως εικόνες και γεωμετρικά αντικείμενα
 - ▶ Παράδειγμα: functions που ελέγχουν αν επικαλύπτονται πολύγωνα ή που συγκρίνουν εικόνες για να ελέγξουν ομοιότητα (similarity)
 - Κάποια Συστήματα ΒΔ υποστηρίζουν **table-valued functions**, οι οποίες επιστρέφουν μία σχέση (relation) ως αποτέλεσμα
- Η SQL:1999 υποστηρίζει επίσης ένα πλούσιο σύνολο από αναγκαίες δομές, όπως:
 - Loops, if-then-else, assignment
- Πολλές ΒΔ έχουν ιδιόκτητες procedural extensions σε SQL οι οποίες διαφέρουν από την SQL:1999





SQL Functions

- Όρισε μία συνάρτηση η οποία να παίρνει ως παράμετρο το όνομα ενός πελάτη, και να επιστρέφει το πλήθος των λογαριασμών που κατέχει ο συγκεκριμένος πελάτης.

```
create function account_count (customer varchar(20))  
returns integer  
begin  
    declare a_count integer;  
    select count (*) into a_count  
    from depositor  
    where depositor.customer_name = customer  
    return a_count;  
end
```





Table Functions

- Η SQL:2003 περιέχει συναρτήσεις οι οποίες επιστρέφουν ως αποτέλεσμα μία ολόκληρη σχέση (relation)
- Παράδειγμα: Επέστρεψε όλους τους λογαριασμούς που ανήκουν σε έναν συγκεκριμένο πελάτη.

```
create function accounts_of (customer char(20)  
    returns table (account_number char(10),  
                    branch_name char(15),  
                    balance numeric(12,2))
```

return table

```
(select account_number, branch_name, balance  
from account  
where exists (  
    select *  
    from depositor  
    where depositor.customer_name = customer  
    and depositor.account_number = account.account_number ))
```





Table Functions (Συν.)

- Βρες τα ονόματα και τις διευθύνσεις κάθε πελάτη ο οποίος έχει περισσότερους από έναν λογαριασμούς.

```
create function names_streets (customer varchar(20)  
    returns table (customer_name char(20),  
                    customer_street char(20),  
                    customer_city char(30))
```

return table

```
(select customer_name, customer_street, customer_city  
from customer  
where account_count (customer) > 1)
```





Table Functions (Συν.)

- Τρόπος χρήσης

select *

from table (*accounts_of* ('Smith'))





SQL Procedures

- Η συνάρτηση *account_count* μπορούσε να γραφτεί και ως διαδικασία (procedure) ως εξής:

```
create procedure account_count_proc (in customer varchar(20),  
                                     out a_count integer)
```

```
begin
```

```
  select count(*) into a_count  
  from depositor  
  where depositor.customer_name = customer
```

```
end
```

- Οι Procedures μπορούν να κληθούν είτε από μία SQL procedure ή από μία embedded SQL, χρησιμοποιώντας την εντολή **call**.

```
  declare a_count integer;  
  call account_count_proc( 'Smith', a_count);
```

Procedures και functions μπορούν να κληθούν επίσης από dynamic SQL

- Η SQL:1999 επιτρέπει σε περισσότερες από μία function/procedure να έχουν το ίδιο όνομα (**overloading**), εφόσον ο αριθμός των ορισμάτων (arguments) διαφέρει, ή τουλάχιστον διαφέρουν οι τύποι των ορισμάτων τους.





Διαδικαστικές Δομές

- Η ενιαία δήλωση: **begin ... end**,
 - Μπορεί να περιέχει πολλαπλά SQL statements μεταξύ **begin** και **end**.
 - Καθώς και τοπικές μεταβλητές (Local variables)
- **While** και **repeat** statements:

```
declare  $n$  integer default 0;  
while  $n < 10$  do  
    set  $n = n + 1$   
end while
```

```
repeat  
    set  $n = n - 1$   
until  $n = 0$   
end repeat
```





Διαδικαστικές Δομές (Συν.)

- **For** loop
 - Επιτρέπει επαναλήψεις πάνω στα αποτελέσματα ενός query
 - Παράδειγμα: Βρες το σύνολο των υπολοίπων στο υποκατάστημα Perryridge

```
declare n integer default 0;  
for r as  
    select balance from account  
    where branch_name = 'Perryridge'  
do  
    set n = n + r.balance  
end for
```





Διαδικαστικές Δομές (Συν.)

- Εντολές συνθήκης (**if-then-else**)

Π.χ. Για να κατατάξουμε το συνολικό υπόλοιπο κάθε πελάτη σε ποια από τις παρακάτω κατηγορίες ανήκει ($balance < 1000$, ≥ 1000 and < 5000 , ≥ 5000)

```
if r.balance < 1000
  then set l = l + r.balance
elseif r.balance < 5000
  then set m = m + r.balance
else set h = h + r.balance
end if
```

- Η SQL:1999 υποστηρίζει επίσης την εντολή **case** με παρόμοιο τρόπο όπως στη C
- Επισήμανση συνθηκών εξαίρεσης (exception conditions), και δηλώσεις χειρισμού εξαιρέσεων (exception handlers)

```
declare out_of_stock condition
declare exit handler for out_of_stock
begin
...
.. signal out-of-stock
end
```

- Ο handler εδώ είναι το **exit** – προκαλεί έξοδο από το **begin..end** σώμα





Αναδρομή (Recursion) στην SQL

- SQL:1999 επιτρέπει τον ορισμό των recursive views
- Παράδειγμα: Βρες όλα τα employee-manager ζευγάρια, όπου ο employee δίνει αναφορά στον manager άμεσα ή έμμεσα (δηλαδή ο manager του manager, ο manager του manager του manager, κ.τ.λ.)

```
with recursive empl (employee_name, manager_name ) as (  
    select employee_name, manager_name  
    from manager  
    union  
    select manager.employee_name, empl.manager_name  
    from manager, empl  
    where manager.manager_name = empl.employee_name)  
select *  
from empl
```





SQL Server – Simple Recursive Query Example

Create a table called “Area”:

```
1 CREATE TABLE dbo.Area(  
2   AreaID int NOT NULL,  
3   AreaName varchar(100) NOT NULL,  
4   ParentAreaID int NULL,  
5   AreaType varchar(20) NOT NULL  
6 CONSTRAINT PK_Area PRIMARY KEY CLUSTERED  
7 ( AreaID ASC  
8 ) ON [PRIMARY])  
9 GO
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)  
VALUES(1, 'Canada', null, 'Country')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)  
VALUES(2, 'United States', null, 'Country')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)  
VALUES(3, 'Saskatchewan', 1, 'State')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)  
VALUES(4, 'Saskatoon', 3, 'City')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)  
VALUES(5, 'Florida', 2, 'State')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)  
VALUES(6, 'Miami', 5, 'City')
```

In this post I use the common example of a table with **countries, states, and cities** and where we want to get **a list of all cities** in a **single country**. Enjoy!





SQL Server – Simple Recursive Query Example

If I do a select by AreaType “City”:
I get both Saskatoon and Miami:

	AreaID	AreaName	ParentAreaID	AreaType
1				
2	4	Saskatoon	3	City
3	6	Miami	5	City

However, what if I wanted to return all cities in Canada?

You can accomplish this by doing a recursive select which uses a common table expression (CTE).

```
1 select * from dbo.Area
2 where AreaType = 'City'
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(1, 'Canada', null, 'Country')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(2, 'United States', null, 'Country')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(3, 'Saskatchewan', 1, 'State')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(4, 'Saskatoon', 3, 'City')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(5, 'Florida', 2, 'State')
```

```
INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(6, 'Miami', 5, 'City')
```





SQL Server – Simple Recursive Query Example

```

WITH AreasCTE AS
(
--anchor select, start with the country of Canada, which will be the root element for our search
SELECT AreaID, AreaName, ParentAreaID, AreaType
FROM dbo.Area
WHERE AreaName = 'Canada'
UNION ALL
--recursive select, recursive until you reach a leaf (an Area which is not a parent of any other area)
SELECT a.AreaID, a.AreaName, a.ParentAreaID, a.AreaType
FROM dbo.Area a
INNER JOIN AreasCTE s ON a.ParentAreaID = s.AreaID
)
--Now, you will have all Areas in Canada, so now let's filter by the AreaType "City"
SELECT * FROM AreasCTE
where AreaType = 'City'

```

```

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(1, 'Canada', null, 'Country')

```

```

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(2, 'United States', null, 'Country')

```

```

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(3, 'Saskatchewan', 1, 'State')

```

```

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(4, 'Saskatoon', 3, 'City')

```

```

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(5, 'Florida', 2, 'State')

```

```

INSERT INTO dbo.Area(AreaID,AreaName,ParentAreaID,AreaType)
VALUES(6, 'Miami', 5, 'City')

```

Now we get back the following results for cities in Canada:

	AreaID	AreaName	ParentAreaID	AreaType
1				
2	4	Saskatoon	3	City





Advanced PL SQL Features**

- Δημιούργησε έναν πίνακα με το ίδιο σχήμα ενός υπάρχοντα:
create table temp_account like account
- SQL:2003 επιτρέπει subqueries να λαβαίνουν χώρα οπουδήποτε μία τιμή απαιτείται, δεδομένου ότι το subquery επιστρέφει μία τιμή μόνο. Αυτό εφαρμόζεται εξίσου καλά στα updates.
- SQL:2003 επιτρέπει subqueries στην πρόταση **from** να προσπελαίνουν attributes από άλλες relations χρησιμοποιώντας τη δομή **lateral** (πλευρική παράλληλη):
select customer_name, num_accounts
from customer, lateral (
 select count(*)
 from account
 where account.customer_name = customer.customer_name)
as this_customer (num_accounts)





Advanced SQL Features (Συν.)

- Η δομή **Merge** επιτρέπει μαζική επεξεργασία (batch processing) ενημερώσεων.
- Παράδειγμα: Η σχέση ***funds_received*** (***account_number***, ***amount***) έχει μία ομάδα / παρτίδα (batch) από καταθέσεις που πρέπει να προστεθούν στον κατάλληλο λογαριασμό (***account***) της σχέσης ***account***
merge into *account* as A
using (select *
 from *funds_received* as F)
on (A.*account_number* = F.*account_number*)
when matched then
 update set *balance* = *balance* + F.*amount*





ΤΕΛΟΣ ΚΕΦΑΛΑΙΟΥ

Database System Concepts, 5th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

