Η Γλώσσα SQL

Μέρος β’

SQL -- Πλήρης Μορφή

select [**distinct**] *target_list*
from *tuple_variable_list*
[**where** *predicate*]
[**group by** *grouping_attributes*]
[**having** *group_condition*]
[**order by** *target_list_subset*]

Διαγραφή - Deletion

- **Εντολή DELETE**

  delete from relation_name **where**-qualification

  SEMANTICS:

  – Εκτέλεσε την ανάλογη SELECT εντολή και έπειτα διάγραψε τις πλειάδες του αποτελέσματος από τη Σχέση

Παραδείγματα Διαγραφής (1)

- Delete all account records at the Perryridge branch

  delete from *account*
  where *branch-name* = ‘Perryridge’

- Delete all accounts at every branch located in Needham city.

```
delete from account
where branch-name in (select branch-name
                      from branch
                      where branch-city = ‘Needham’)
delete from depositor
where account-number in
      (select account-number
      from branch, account
      where branch-city = ‘Needham’
        and branch.branch-name = account.branch-name)
```

Παραδείγματα Διαγραφής (2)

- Delete the record of all accounts with balances below the average at the bank.

delete from *account*
where *balance* < (**select avg** *(balance)*
from *account)*

- Πρόβλημα: όσο σβήνουμε tuples από το *account*, αλλάζει το μέσο υπόλοιπο λογαριασμού
- Λύση στην SQL:

1. Πρώτα υπολογίζεται το **avg** balance και βρίσκονται όλα τα tuples προς διαγραφή
2. Μετά σβήνουμε όλα τα tuples που βρήκαμε παραπάνω

Εισαγωγή - Insertion

- **Εντολή INSERT**

insert into relation_name **select**-statement

ή

insert into relation_name **values** (value_list)

Παραδείγματα Εισαγωγής (1)

- Πρόσθεσε νέο tuple στο *account*

  insert into *account*
  values (‘A-9732’, ‘Perryridge’,1200)

  ή ισοδύναμα:

  insert into *account (branch-name, balance, account-number)*
  values (‘Perryridge’, 1200, ‘A-9732’)

- Πρόσθεσε νέο tuple στο *account* με *balance* ίσο με null

  insert into *account*
  values (‘A-777’,‘Perryridge’, *null*)

Παραδείγματα Εισαγωγής (2)

- Provide as a gift for all loan customers of the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account

```
insert into account
    select loan-number, branch-name, 200
    from loan
    where branch-name = 'Perryridge'

insert into depositor
    select customer-name, loan-number
    from loan, borrower
    where branch-name = 'Perryridge'
        and loan.account-number = borrower.account-number
```

- Το select from where αποτιμάται πριν τα αποτελέσματά του να εισαχθούν στη σχέση (αλλιώς queries σαν αυτό

  insert into *table*1 **select** * **from** *table*1

  θα δημιουργούσαν πρόβλημα)

Ενημερώσεις - Update

- **Εντολή UPDATE**

  update relation_name
  set target_list
  where qualification

  SEMANTICS:

  – Εκτέλεσε την ανάλογη SELECT, διάγραψε τις πλειάδες και τέλος, κάνε εισαγωγή των νέων

Modification of the Database – Updates

- Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.
  - Write two **update** statements:

update *account*
set *balance* = *balance* ∗ 1.06
where *balance* > 10000

update *account*
set *balance* = *balance* ∗ 1.05
where *balance* ≤ 10000

  - Η σειρά έχει σημασία
  - Μπορεί να γίνει καλύτερα με το case statement

Case Statement για Ενημερώσεις υπό όρους

- Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

```
update account
set balance =  case
        when balance <= 10000 then balance *1.05
        else   balance * 1.06
        end
```

Data Definition Language (DDL)

Επιτρέπει τον ορισμό σχέσεων αλλά και πληροφοριών για κάθε σχέση:

- Το σχήμα κάθε σχέσης
- Τα πεδία τιμών κάθε γνωρίσματος
- Περιορισμούς ακεραιότητας
- Δείκτες που διατηρούνται για κάθε σχέση
- Πληροφορίες για ασφάλεια και εξουσιοδότηση για κάθε σχέση
- Τη δομή αποθήκευση στον δίσκο για κάθε σχέση

SQL Data Definition

- Η Γλώσσα Ορισμού (DDL) της SQL χρησιμοποιείται για
  - CREATE
  - DROP
  - ALTER

Τύποι πεδίου τιμών στην SQL

- **char(n):** Συμβολοσειρά σταθερού μήκους *n*.
- **varchar(n) :** Συμβολοσειρά μεταβλητού μήκους με μέγιστο μήκος *n*.
- **int :** Integer (machine-dependent).
- **smallint :** Small integer
- **numeric(p,d):** Δεκαδικός αριθμός σταθερής υποδιαστολής, με p ψηφία και n ψηφία δεξιά από την υποδιαστολή
- **real, double precision:** Κινητής υποδιαστολής και διπλής ακρίβειας αριθμοί (machine-dependent)
- **float(n):** Κινητής υποδιαστολής , με ακρίβεια τουλάχιστον *n* ψηφίων

- Οι τιμές null επιτρέπονται σε όλους τους τύπους πεδίου τιμών
- Αν ένα γνώρισμα οριστεί **not null** τότε απαγορεύονται οι τιμές null για το γνώρισμα αυτό

Τύποι Date/Time στην SQL

- **date.** Ημερομηνίες που περιέχουν (4 digit) year, month and date
  - E.g. **date** '2001-7-27'
- **time.** Χρόνος σε ώρες, λεπτά, δευτερόλεπτα
  - E.g. **time** '09:00:30'
- **timestamp**: ημερομηνία και ώρα
  - E.g. **timestamp** '2001-7-27 09:00:30.75'
- **Interval**: χρονική περίοδος
  - E.g. Interval '1' day
  - Η αφαίρεση ανάμεσα σε τιμές date/time/timestamp δίνουν interval
  - Τιμές interval μπορούν να προστεθούν σε date/time/timestamp
- Εξαγωγή τιμών από μεμονωμένα πεδία του date/time/timestamp
  - E.g. **extract** (**year from** r.starttime)
- cast string σε date/time/timestamp
  - E.g. **cast** <string-valued-expression> **as date**

Ορισμός νέων τύπων πεδίου τιμών

- **To create domain** στη SQL-92 ορίζει νέο τύπο (από άλλους υπάρχοντες)

  create domain *person-name* **char**(20) **not null**

  create domain *Dollars* **numeric**(12, 2)

  create domain *Euros* **numeric**(12,2)

- Δεν μπορούμε να συγκρίνουμε μια τιμή τύπου Dollars με μια τιμή τύπου Euros.
  - However, we can convert type as below
    (**cast** *r.A* **as** *Pounds*)
    (Should also multiply by the dollar-to-pound conversion-rate)

Περιορισμοί Πεδίου Τιμών

- Το **check** clause στην SQL-92 επιτρέπει επιπλέον περιορισμούς:
  - Έλεγχος ότι ένα πεδίο τιμών επιτρέπει μόνο τιμές που είναι μεγαλύτερες από κάποιο ελάχιστο.

    create domain *hourly-wage* **numeric(5,2)**
    constraint *value-test* **check**(*value* > = 4.00)
  - Το **constraint** *value-test* είναι προαιρετικό, αλλά χρήσιμο για να δείχνει που ενδεχομένως γίνεται μια παραβίαση.
- Μπορεί να είναι αρκετά σύνθετο
  - **create domain** *AccountType* **char**(10)
    constraint *account-type-test*
    check (**value in** ('Checking', 'Saving'))
  - **check** (*branch-name* **in** (**select** *branch-name* **from** *branch*))

Create Table

- Μια SQL σχέση ορίζεται με την εντολή **create table**:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - r το όνομα της σχέσης
  - Κάθε A_i είναι το όνομα ενός γνωρίσματος στο σχήμα της σχέσης r
  - D_i είναι ο τύπος δεδομένων των πεδίων τιμών του γνωρίσματος A_i

- Παράδειγμα:

  create table *branch*
  (*branch-name* char(15) **not null,**
  branch-city char(30),
  assets integer)

Περιορισμοί ακεραιότητας στο Create Table

- **not null**
- **primary key** $(A_1, ..., A_n)$
- **check** *(P)*, όπου *P* ένα predicate

Παράδειγμα: Ορισμός *branch-name* ως πρωτεύοντος κλειδιού του *branch* και έλεγχος ότι οι τιμές των *assets* είναι θετικές

create table *branch*
(branch-name char(15)**,**
branch-city char(30)
assets integer,
primary key *(branch-name)*,
check *(assets* >= 0))

Ο ορισμός **primary key** σε ένα γνώρισμα αυτόματα ορίζει και **not null** από την SQL-92 και μετά

Αναφορική Ακεραιότητα στην SQL

- Κύρια / Υποψήφια κλειδιά και αναφορική ακεραιότητα ορίζονται στην SQL με την εντολή **create table**:
  - Το **primary key** clause για γνωρίσματα του κύριου κλειδιού.
  - Το **unique key** clause για γνωρίσματα του υποψήφιου κλειδιού.
  - Το **foreign key** clause για γνωρίσματα του εξωτερικού κλειδιού και το όνομα της σχέσης που αναφέρεται από το εξωτερικό κλειδί
- Ένα εξωτερικό κλειδί αναφέρεται στο κύριο κλειδί της αναφερόμενης σχέσης

  foreign key (*account-number*) **references** *account*
- Είναι δυνατή και η σύντμηση

  account-number **char** (10) **references** *account*
- Επίσης, μπορεί να γραφεί σε πλήρη μορφή (ρητή αναφορά σε γνώρισμα)
  - Πάντα αναφέρεται σε κύριο ή υποψήφιο κλειδί

    foreign key (*account-number*) **references** *account*(*account-number*)

Αναφορική Ακεραιότητα στην SQL – Παράδειγμα

create table *customer*
(*customer-name* char(20)**,**
customer-street char(30),
customer-city char(30),
primary key (*customer-name))*

create table *branch*
(branch-name char(15)**,**
branch-city char(30),
assets integer,
primary key *(branch-name))*

create table *account*
(*account-number* char(10)**,**
branch-name char(15),
balance integer,
primary key (*account-number),*
foreign key (*branch-name)* **references** *branch)*

create table *depositor*
(*customer-name* char(20)**,**
account-number char(10)**,**
primary key *(customer-name, account-number),*
foreign key *(account-number)* **references** *account,*
foreign key *(customer-name)* **references** *customer*

Drop Table

- Η εντολή:

  drop table *branch*

  διαγράφει τον πλήρη πίνακα *branch* και τον ορισμό του

  (δεν είναι δυνατή πλέον η χρήση του για ενημερώσεις, κλπ.)

Alter table

- Η εντολή **alter table** προσθέτει νέα γνωρίσματα σε υπάρχουσα σχέση

  alter table *r* **add** *A D*

  όπου *A* το όνομα του attribute που θα προστεθεί στη σχέση *r* και *D* το domain του *A*.

  – Όλες οι τιμές για τα νέα γνωρίσματα είναι αρχικά *null*

  – Αργότερα μπορούν να αλλάξουν με την εντολή update

- Η εντολή **alter table** χρησιμοποιείται και για την αφαίρεση attributes

  alter table *r* **drop** *A*

  όπου *A* το όνομα γνωρίσματος της σχέσης *r*

  – Δεν υποστηρίζεται από τις περισσότερες βάσεις

Περιορισμοί στο Σχεσιακό Μοντέλο

- Όταν υποστηρίζεται ένας περιορισμός ακεραιότητας, αρκετές **ενέργειες πρέπει να γίνουν** από το DBMS. Αυτές είναι, η **διαδοχική τροποποίηση (cascade delete)** και η **διαδοχική ενημέρωση (cascade update.)**
- Μερικά DBMSs υποστηρίζουν άμεσα τα παραπάνω (Access)
- Άλλα DBMSs απαιτούν από τον χρήστη να γράψει **triggers** (διαδικασίες) για την υποστήριξη αυτών (SQL Server)
- **Περιορισμοί Πεδίων** υποστηρίζονται μερικώς (ενίοτε και από **strong-typing** ή άλλους ισχυρούς μηχανισμούς)
- **Περιορισμοί που ορίζονται από τον Χρήστη** έχουν περιληφθεί σε νεώτερες εκδόσεις της τυποποίησης της SQL (SQL-92) με **assertions** (ειδικοί κανόνες)

Assertions - Παραδείγματα

- **Κάθε Υπάλληλος είναι είτε Άνδρας ή Γυναίκα**

  CREATE ASSERTION gender **CHECK** Sex="m" **OR** Sex="f"

- **Οι Μισθοί πρέπει να ξεπερνούν το 1,000,000**

  CREATE ASSERTION salary_bound **CHECK** Salary > 1000000

  Αν κάποιος επιχειρήσει την παρακάτω εισαγωγή:

  insert into EMPLOYEE (**select** Name="tom",.., Salary=1200000, …)

  η εισαγωγή ΑΛΛΑΖΕΙ εμπεδώνοντας τον κανόνα assert σε:

  insert into EMPLOYEE (**select** Name="tom",.., Salary=12000..
  where 1200000 > 1000000)

Assertions – Παραδείγματα (2)

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

create assertion *sum-constraint* **check**
(not exists (select * from *branch*
where (select sum(*amount*) **from** *loan*
where *loan.branch-name* =
branch.branch-name)
>= **(select sum**(*amount*) **from** *account*
where *loan.branch-name* =
branch.branch-name)))

Triggers

- Το **trigger** είναι μια εντολή που εκτελείται αυτόματα από το σύστημα σαν παρενέργεια (εκπυρσοκρότηση) μιας τροποποίησης της Βάσης Δεδομένων
- Για να σχεδιαστεί ένας μηχανισμός trigger χρειάζεται να
  - Γραφούν οι συνθήκες εκτέλεσης του trigger
  - Γραφούν οι πράξεις που γίνονται όταν το trigger εκτελείται.
- Τα Triggers επίσημα ορίστηκαν με την τυποποίηση της SQL το 1999, αλλά προϋπήρχαν και σε παλαιότερα DBMS
- Τα Triggers ΔΕΝ ΣΥΝΙΣΤΑΝΤΑΙ (εκτός αν είναι απολύτως απαραίτητα) διότι μπορεί να έχουν αναπάντεχα αποτελέσματα όταν δεν έχουν σχεδιαστεί καλά

Παράδειγμα Trigger στην SQL:1999

```
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
        insert into borrower
           (select customer-name, account-number
            from depositor
            where nrow.account-number =
                        depositor.account-number);
        insert into loan values
           (n.row.account-number, nrow.branch-name,
                                              – nrow.balance);
        update account set balance = 0
         where account.account-number = nrow.account-number
end
```

Χρήση των triggers για Εξωτερικές πράξεις

- Π.χ., κάνε νέα παραγγελία για ένα προϊόν του οποίου η ποσότητα στην αποθήκη λιγοστεύει μετά από μια τροποποίηση.

```
create trigger reorder-trigger after update of amount on inventory
referencing old row as orow, new row as nrow
for each row
        when nrow.level < = (select level
                                from minlevel
                                where minlevel.item = orow.item)
                                    and orow.level > (select level
                                                        from minlevel
                                                        where minlevel.item = orow.item)
    begin
                        insert into orders
                              (select item, amount
                                from reorder
                                where reorder.item = orow.item)
    end
```

Δημιουργία Ευρετηρίων στην SQL

- Στις περισσότερες περιπτώσεις μια Σχέση Βάσης αντιστοιχεί σε αποθηκευμένο αρχείο
- Η εντολή **create index** χρησιμοποιείται για τον προσδιορισμό ευρετηρίου (index) που έχει ονομασία (index name)

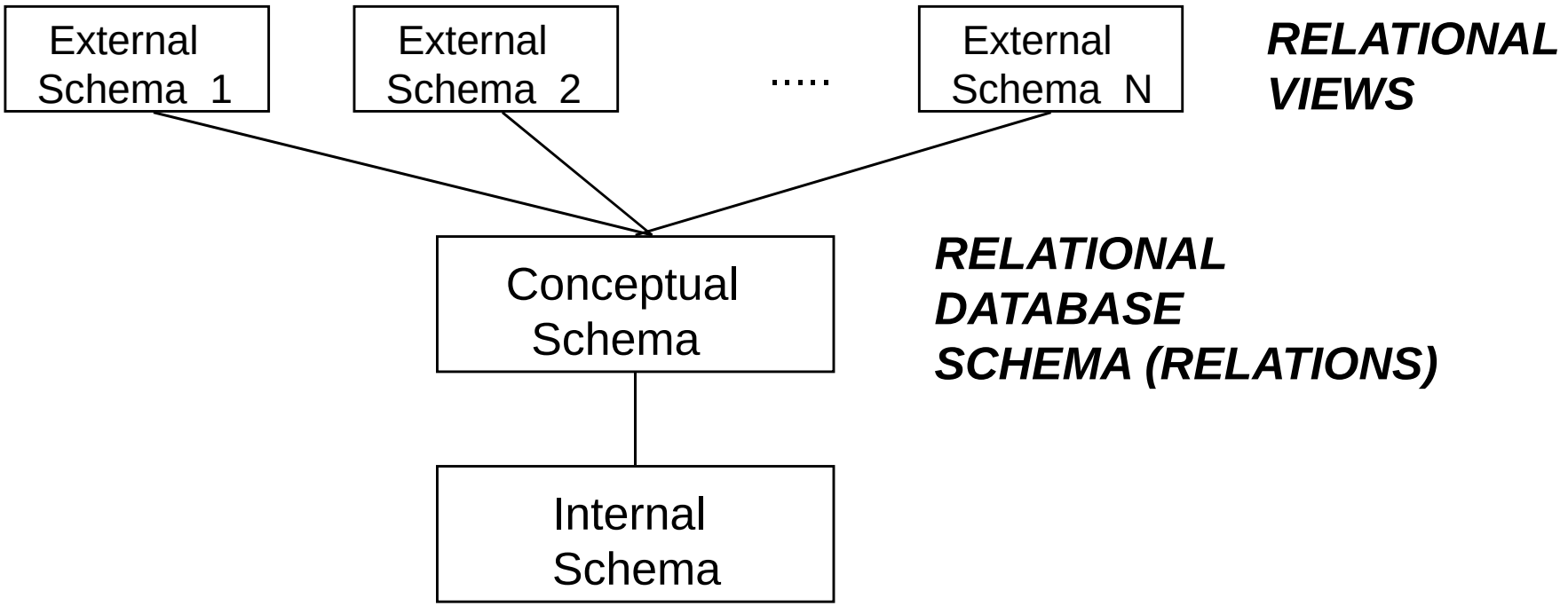
  create index LN_INDEX on EMPLOYEE (Name);
- Για τον προσδιορισμό ενός περιορισμού ευρετηρίου κλειδιού στο γνώρισμα (γνωρίσματα) κλειδιού χρησιμοποιείται το UNIQUE

  create unique index S_IND on EMPLOYEE (SSN);
- Για τον προσδιορισμό ενός ευρετηρίου συστάδων (clustering index) χρησιμοποιείται το CLUSTER

  create index D_IN on EMPLOYEE(DNumber) **cluster**;
- Στα περισσότερα DBMS, κάποια παραλλαγή των B+-tree χρησιμοποιείται για την υλοποίηση ευρετηρίων

Σχεσιακές Όψεις και SQL

- Η αρχιτεκτονική 3-επιπέδων στα DBMS, έχει σαν αποτέλεσμα *διαφορετικοί* χρήστες να βλέπουν *διαφορετικά* εξωτερικά σχήματα (όψεις) - αυτό επιτυγχάνεται στο σχεσιακό μοντέλο με την έννοια του **view**



Σχεσιακές Όψεις – Ορισμός (1)

- Μια **όψη (view)** είναι μια Σχέση που ΔΕΝ είναι στοιχείο του Εννοιολογικού Σχήματος αλλά είναι προσβάσιμη ως μια **ιδεατή (virtual)** σχέση
  - Ο χρήστης δεν μπορεί να ξεχωρίσει αν μια Σχέση είναι Σχέση Βάσης ή είναι Όψη
  - Η Όψη ΔΕΝ αποθηκεύεται (materialized) στην Βάση Δεδομένων
  - Τα περιεχόμενα της Όψης προσδιορίζονται από τον **(αποθηκευμένο) ορισμό της** σαν μια συνάρτηση των περιεχομένων της
  - Μια Όψη ορίζεται σε Σχέσεις Βάσης ή σε άλλες Όψεις με κάποιο query
  - Πράξεις (queries, ενημερώσεις) σε Όψεις μεταφράζονται σε Πράξεις στις ορίζουσες (την Όψη) Σχέσεις Βάσης
  - ΔΕΝ υπάρχουν περιορισμοί στα Queries σε Όψεις
  - Ελάχιστες ενημερώσεις σε Όψεις επιτρέπονται
  - Μια Όψη αλλάζει **δυναμικά** με τη Βάση δεδομένων

Όψεις

- Μηχανισμός για να κρύψεις δεδομένα από συγκεκριμένες ομάδες χρηστών.

  create view *v* **[(** *view_column_list* **)] as** <query expression>

  όπου:

  - <query expression> οποιαδήποτε έκφραση
  - *V το όνομα της όψης*
  - *view_column_list* τα νέα ονόματα των γνωρισμάτων

Παράδειγμα όψης

- A view consisting of branches and their customers

```
create view all-customer as
  (select branch-name, customer-name
   from depositor, account
   where depositor.account-number = account.account-number)
    union
  (select branch-name, customer-name
   from borrower, loan
   where borrower.loan-number = loan.loan-number)
```

- Find all customers of the Perryridge branch

```
select customer-name
        from all-customer
        where branch-name = 'Perryridge'
```

Παράδειγμα όψης (2)

- Μια Όψη είναι δυνατόν να έχει **διαφορετικά γνωρίσματα (νέα)** από το Εννοιολογικό Σχήμα
- Πληροφορίες για υποκαταστήματα

Create view branch-info (*branch-name, NoOfAccounts, AvgBalance*)
as **select** *branch-name, COUNT(*), AVG(balance)*
from *branch b, account a*
where *b.branch-name = a. branch-name*
group by *branch-name* ;

Ενημέρωση όψης

- Δημιούργησε ένα view όλων των πληροφοριών της σχέσης *loan*, κρύβοντας το γνώρισμα *amount*

  create view *branch-loan* **as**
  select *branch-name, loan-number*
  from *loan*

- Πρόσθεσε νέο tuple στο *branch-loan*

  insert into *branch-loan*
  values (‘Perryridge’, ‘L-307’)

  αυτή η εισαγωγή αντιστοιχεί με την εισαγωγή του

  (‘L-307’, ‘Perryridge’, *null*)

  στο *loan*

- Ενημερώσεις σε πιο πολύπλοκες όψεις είναι δύσκολο ή και αδύνατο να μεταφραστούν στο αρχικό σχήμα, οπότε απαγορεύονται
- Οι περισσότερες υλοποιήσεις της SQL επιτρέπουν ενημερώσεις μόνο σε απλές όψεις που ορίζονται σε μια μόνο σχέση

Σχεσιακές Όψεις - Εκτέλεση

- Όπως ήδη ειπώθηκε, το σύστημα μεταφράζει πράξεις σε όψεις σε πράξεις σε Σχέσεις Βάσης.
- Πώς;
  - **ΧΑΖΗ Προσπάθεια 1**: **Δημιούργησε (Materialize)** την Όψη (προσωρινός πίνακας) και **εκτέλεσε το query σε αυτήν.**
  - **ΧΑΖΗ Προσπάθεια 2:** Κράτα συνεχώς ενήμερη την Όψη -- δηλαδή, κράτησέ την δημιουργημένη (ονομάζεται, snapshot) και όποτε σχετικές ενημερώσεις γίνονται στις Σχέσεις Βάσης, τότε πέρασε αυτές και στην Όψη. **Εκτέλεσε το query στο snapshot**.
  - **ΣΩΣΤΗ Προσπάθεια:** **Άλλαξε το query** αντικαθιστώντας την Όψη με το query σε Σχέσεις Βάσης που την ορίζει, και **εκτέλεσε το query σε αυτές τις Σχέσεις Βάσης.**

Σχεσιακές Όψεις – Παραδείγματα Εκτέλεσης

- Για παράδειγμα, το query στην Όψη *branch-loan* :

select *branch-name*
from *branch-loan*
where *loan-number* = L-1212

Τροποποιείται ανάλογα και το παρακάτω query εκτελείται:

select *branch-name*
from *loan*
where *loan-number* = L-1212

Σχεσιακές Όψεις - Ενημερώσεις

- Γενικά, υπάρχουν σοβαροί περιορισμοί στις ενημερώσεις
- Μια ενημέρωση Όψης είναι αναμφίβολη αν μια μοναδική ενημέρωση στις Σχέσεις Βάσης μπορεί να εκπληρώσει το ζητούμενο αποτέλεσμα στην Όψη
- ΠΑΡΑΤΗΡΗΣΕΙΣ:
  - Μια όψη που ορίζεται σε μια (ακριβώς) Σχέση Βάσης είναι ενημερώσιμη αν τα γνωρίσματα της Όψης περιέχουν το Κύριο Κλειδί
  - Όψεις που ορίζονται σε πολλαπλές Όψεις είναι γενικά μη-ενημερώσιμες
  - Όψεις που περιέχουν Συναθροιστικές Συναρτήσεις είναι σίγουρα μη-ενημερώσιμες
- Οι Ενημερώσεις των Όψεων παραμένουν ένα ενδιαφέρον ανοικτό ερευνητικό θέμα / πρόβλημα (π.χ., Data Warehouses)

Σχεσιακές Όψεις - Σχόλια

- Το “WITH CHECK OPTION” χρησιμοποιείται σε ενημερώσιμες όψεις για να αντιμετωπίσει το πρόβλημα των γραμμών που εξαφανίζονται (*vanishing rows*)
- *Για παράδειγμα, θεωρήστε την Όψη:*

  CREATE VIEW V AS select * from R where A = “X”

  Έστω η ενημέρωση:

  UPDATE V set A = “Y”

  Η ενημέρωση θα προχωρήσει κανονικά, αλλά οι πλειάδες που ήταν πριν προσβάσιμες στην V θα εξαφανιστούν! (προφανώς, δεν θα ικανοποιούν πλέον την συνθήκη της WHERE-πρότασης)

Εμφύτευση της SQL σε Γλώσσα Προγραμματισμού

- Εντολές DML συχνά ***εμφυτεύονται σε προγράμματα γλώσσας προγραμματισμού*** **(host)**
- Η εμφυτευμένη SQL εντολή διακρίνεται από εντολές της Γλώσσας Προγραμματισμού με ειδικό τρόπο, π.χ. με πρόθεμα
- Υπάρχουν δυο τρόποι εμφύτευσης:

1.- *Επέκταση της Γλώσσας Προγραμματισμού (αλλαγή του Compiler)*

Ενδεικτικά, RIGEL, MODULA-R, Gemstone, Orion, κλπ.)

Ονομάζονται **database programming languages**

2.- *Χρήση ενός προ-επεξεργαστή της Γλώσσας για τις εντολές DML*

Ο προ-επεξεργαστής αντικαθιστά τις εντολές DML με κλήσεις στην host γλώσσα, οι οποίες **εκτελούνται**

Εμφυτευμένη (Embedded) SQL

- Η SQL μπορεί να εμφυτευτεί σε πληθώρα γλωσσών προγραμματισμού (Pascal, PL/I, Fortran, C, Cobol…).
- Η γλώσσα στην οποία εμφυτεύονται τα SQL queries λέγεται *host* language, και οι δομές SQL που επιτρέπονται στην host language λέγονται *embedded* SQL.
- EXEC SQL χρησιμοποιείται για να σηματοδοτήσει ένα embedded SQL request στον preprocessor

  EXEC SQL <embedded SQL statement > END-EXEC

  Note: μπορεί να διαφέρει ανάλογα με τη γλώσσα.

  π.χ. στην Java

  # SQL { …. } ;

Εμφύτευση της SQL

- SQL εντολές καλούνται μέσα από ένα πρόγραμμα μιας host γλώσσας (π.χ., C ή COBOL)
  - Οι εντολές της SQL μπορούν να αναφέρονται σε host μεταβλητές
  - Πρέπει να περιλαμβάνουν μια εντολή για *σύνδεσμο* με την σωστή βάση.
  - Οι SQL Σχέσεις είναι σύνολα εγγραφών, χωρίς προκαθορισμένο (*a priori)* όριο στον αριθμό των εγγραφών. ***Δεν υπάρχουν τέτοιες δομές στην C*** !
  - Για να αντιμετωπίσει αυτή την δυσκολία η SQL υποστηρίζει ένα μηχανισμό που λέγεται ***cursor***

Cursors

- Η εντολή **open** έχει ως αποτέλεσμα την αποτίμηση ενός query

  EXEC SQL **open** *c* END-EXEC

- Το **fetch** βάζει την τιμή ενός tuple του αποτελέσματος στη μεταβλητή της host language

  EXEC SQL **fetch** *c* **into** :*cn*, *:cc* END-EXEC

  Επαναλαμβανόμενες κλήσεις της **fetch** φέρνουν διαδοχικά tuples του αποτελέσματος

- Η εντολή **close** διαγράφει οποιαδήποτε προσωρινή σχέση της βάσης που κρατάει αποτελέσματα του query.

  EXEC SQL **close** *c* END-EXEC

Note: Οι λεπτομέρειες μπορεί να διαφέρουν ανά γλώσσα. Π.χ., στην Java έχουμε iterators

Παράδειγμα

Μέσω host language, βρες τα ονόματα και τις πόλεις των πελατών με περισσότερα στον λογαριασμό τους από τη μεταβλητή *amount*

- Specify the query in SQL and declare a *cursor* for it

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

Updates Through Cursors

- Can update tuples fetched by cursor by declaring that the cursor is for update

```
declare c cursor for
  select *
  from account
  where branch-name = 'Perryridge'
for update
```

- To update tuple at the current location of cursor

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

Εμφύτευση της SQL σε C -- Παράδειγμα

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_NOνομα[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
   SELECT S.Ονομα, S.age FROM ΝΑΥΤΙΚΟΣ S
   WHERE S.rating > :c_minrating
   ORDER BY S.Ονομα;
do {
   EXEC SQL FETCH sinfo INTO :c_Ονομα, :c_age;
   printf("%s is %d years old\n", c_NOνομα, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Δυναμική SQL

- Επιτρέπει σε προγράμματα να κατασκευάζουν και να στέλνουν SQL επερωτήσεις σε run time
- Παράδειγμα χρήσης δυναμικής SQL από ένα 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;
```

- Το δυναμικό SQL πρόγραμμα περιέχει ένα ?, που θα συμπληρωθεί από τον χρήστη όταν το SQL πρόγραμμα εκτελείται

API Βάσης Δεδομένων: Εναλλακτικός της Εμφύτευσης τρόπος πρόσβασης

- Αντί να αλλάζει ο compiler, προστίθεται μια βιβλιοθήκη με Κλήσεις στη Βάση Δεδομένων (API)
- Application Programming Interface
  - Ειδικές Διαδικασίες / Αντικείμενα
  - Περνά τις SQL εντολές σαν character strings από τη γλώσσα και παρουσιάζει τα αποτελέσματα με φιλικό (για τη γλώσσα) τρόπο
  - Το ***ODBC*** της Microsoft έχει γίνει το C/C++ standard στα Windows
  - Το ***JDBC*** της SUN είναι το ανάλογο για τη Java
  - Ανεξάρτητα του DBMS
    - » Ένας "οδηγός" παγιδεύει τις κλήσεις και τις μεταφράζει σε κώδικα για το DBMS
    - » Η Βάση μπορεί να είναι στο Δίκτυο

ODBC

- Open DataBase Connectivity(ODBC) standard
  - Standard για επικοινωνία εφαρμογής με database server
  - application program interface (API) για
    - » Άνοιγμα σύνδεσης με βάση,
    - » Αποστολή ερωτημάτων και ενημερώσεων,
    - » Λήψη αποτελεσμάτων
- Applications such as GUI, spreadsheets, etc. can use ODBC

ODBC (Cont.)

- Κάθε σύστημα που υποστηρίζει ODBC παρέχει μια βιβλιοθήκη "driver" που συνδέεται με το πρόγραμμα του πελάτη
- Όταν το πρόγραμμα του πελάτη κάνει μια κλήση στο ODBC API, ο κώδικας στη βιβλιοθήκη επικοινωνεί με τον server για να εκτελέσει την πράξει που θέλει ο χρήστης και να ανακτήσει τα αποτελέσματα
- Ανοίγει σύνδεση με τη βάση με SQLConnect(). Παράμετροι για SQLConnect:
  - connection handle,
  - the server to which to connect
  - the user identifier,
  - password

JDBC

- Το JDBC είναι ένα Java API για επικοινωνία με συστήματα βάσεων που υποστηρίζουν SQL
- Υποστηρίζει αναζήτηση, ενημέρωση και ανάκτηση δεδομένων
- Υποστηρίζει ανάκτηση metadata, όπως ερωτήσεις για υπάρχουσες σχέσεις, ονόματα και τύπους γνωρισμάτων
- Επικοινωνία με τη βάση:
  - Ανοίγει σύνδεση
  - Δημιουργεί ένα αντικείμενο “statement”
  - Εκτελεί ερωτήματα χρησιμοποιώντας το αντικείμενο statement και ανακτά τα αποτελέσματα
  - Έχει μηχανισμούς exception για διαχείριση σφαλμάτων

JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(   "jdbc:oracle:thin:@aura.bell-
          labs.com:2000:bankdb", userid, passwd);
       Statement stmt = conn.createStatement();
          … Do Actual Work ….
       stmt.close();
       conn.close();
   }
   catch (SQLException sqle) {
       System.out.println("SQLException : " + sqle);
   }
}
```

JDBC Code (Cont.)

- Update to database

```
try {
    stmt.executeUpdate(  "insert into account values
                                ('A-9732', 'Perryridge', 1200)");
} catch (SQLException sqle) {
    System.out.println("Could not insert tuple. " + sqle);
}
```

- Execute query and fetch and print results

```
ResultSet rset = stmt.executeQuery( "select branch_name, avg(balance)
                                     from account
                                     group by branch_name");
while (rset.next()) {
    System.out.println(
            rset.getString("branch_name") + "  " + rset.getFloat(2));
}
```

Μειονεκτήματα των Host Programming Languages

- Ανάμιξη των Διαδικαστικών και Δηλωτικών γλωσσών (το γνωστό *language mismatch* πρόβλημα)
- *Διαφορετικοί* προ-επεξεργαστές απαιτούνται για *διαφορετικές* γλώσσες
- Οι Σχέσεις *δεν αποτελούν πολίτες Α κατηγορίας* στην γλώσσα (π.χ., μια Σχέση δεν περνιέται σαν παράμετρος σε διαδικασία)
- Η HOST γλώσσα *πιθανόν να μην υποστηρίζει χρήσιμες δομές* (π.χ., η FORTRAN δεν υποστηρίζει εγγραφές)
- Οι ***database programming languages*** κέρδισαν πολύ έδαφος, ειδικά σε object-oriented DBMSs

Μειονεκτήματα της SQL

- Η SQL δεν υποστηρίζει ***strong typing, inheritance***, *etc.*
- Οι SQL πίνακες ΔΕΝ ΕΙΝΑΙ **ΣΧΕΣΕΙΣ** (επιτρέπουν διπλές πλειάδες)
- Οι SQL πίνακες **δεν εμφωλιάζονται** (δεν αποτελούν τιμές για άλλους πίνακες)
- Η SQL **δεν υποστηρίζει** πολλές πράξεις, όπως: *generalized restriction, division, forall*
- Η SQL **δεν υποστηρίζει** 3-V logic (Λογική 3 τιμών αληθείας)
- Η SQL **δεν υποστηρίζει συναρτησιακές εξαρτήσεις (*functional dependencies)***
- Η SQL **δεν υποστηρίζει** περιορισμούς ακεραιότητας σε Όψεις

ΠΑΡΟΛΑ ΑΥΤΑ, η SQL είναι η STANDARD ΓΛΩΣΣΑ

Query-by-Example (QBE)

- Μια Γλώσσα για ερωταποκρίσεις που αναπτύχθηκε στην IBM (από τον Moshe Zloof) και παρουσιάζεται σε ένα προϊόν (QMF) (που είναι εναλλακτικός τρόπος διεπαφής για το DB2)
- Ευκολότερη από την SQL για τον ***μέσο χρήστη***

  (ΟΠΤΙΚΗ και **ΔΙΣΔΙΑΣΤΑΤΗ)**
- **ΚΕΝΤΡΙΚΗ ΙΔΕΑ:** Το Σύστημα παρέχει στον χρήστη τη δυνατότητα να δει το **περίγραμμα** των Σχέσεων στη Βάση και ο Χρήστης συμπληρώνει τους πίνακες δίνοντας παραδείγματα για το πώς θέλει να είναι η απάντηση

QBE ΣΥΝΟΨΗ

- Οι Αρχές της Γλώσσας
  - Ο χρήστης ***δεν απαιτείται να θυμάται*** τα ονόματα των γνωρισμάτων και των σχέσεων
  - Στην διατύπωση της ερωταπόκρισης, ***δεν απαιτείται να τηρούνται ανελαστικοί κανόνες***
  - Στηρίζεται στον σχεσιακό λογισμό ***πεδίου*** (μεταβλητές είναι οι στήλες)
  - ***Σχεσιακά πλήρης*** διατύπωση

- Πως Λειτουργεί
  - Σύμβολα με "_" να προηγείται, είναι **μεταβλητές**
  - Σύμβολα χωρίς "_" να προηγείται είναι **σταθερές** (υποδηλώνουν μια συνθήκη για επιλογή - ***equality selection-condition***)
  - Το πρόσημο "**P.**" χρησιμοποιείται για να υποδειχθεί ποια γνωρίσματα θα **τυπωθούν** (υποδηλώνει μια προβολή - ***projection***)

QBE Σύνοψη – Η διαδικασία

- Διαδικασία ερωταπόκρισης
  - Πρώτα, ο χρήστης ***διαλέγει*** *τις σχέσεις (πίνακες)* που χρειάζεται για το query
  - Παρουσιάζονται τα ***περιγράμματα*** των πινάκων που διαλέχτηκαν
  - Ο χρήστης «πηγαίνει» στις ***κατάλληλες στήλες*** (με ειδικά πλήκτρα)
  - ***Τιμές-παραδείγματα*** (μεταβλητές), ***σταθερές***, κλπ., δακτυλογραφούνται
  - ***Άλλοι συγκριτικοί τελεστές*** (πέραν της ισότητας – που είναι αυτόματη για σταθερές τιμές) πρέπει να ***δακτυλογραφηθούν*** (όπως, **>, <,** κλπ.)
  - πιο ***πολύπλοκες συνθήκες*** μπαίνουν σε ένα **κουτί-συνθηκών** (***condition box)***
  - Συνθήκες στην ίδια σειρά *υποδηλώνουν το* Boolean **AND**
  - Συνθήκες σε διαφορετικές σειρές *υποδηλώνουν το* Boolean **OR**
  - **Η άρνηση (negation -** Boolean **NOT**) προσδιορίζεται με το σύμβολο " ¬ "
  - **Οι Συνενώσεις (JOINS)** εκφράζονται με τη χρήση ***κοινών παραδειγματικών τιμών*** σε πολλαπλούς πίνακες

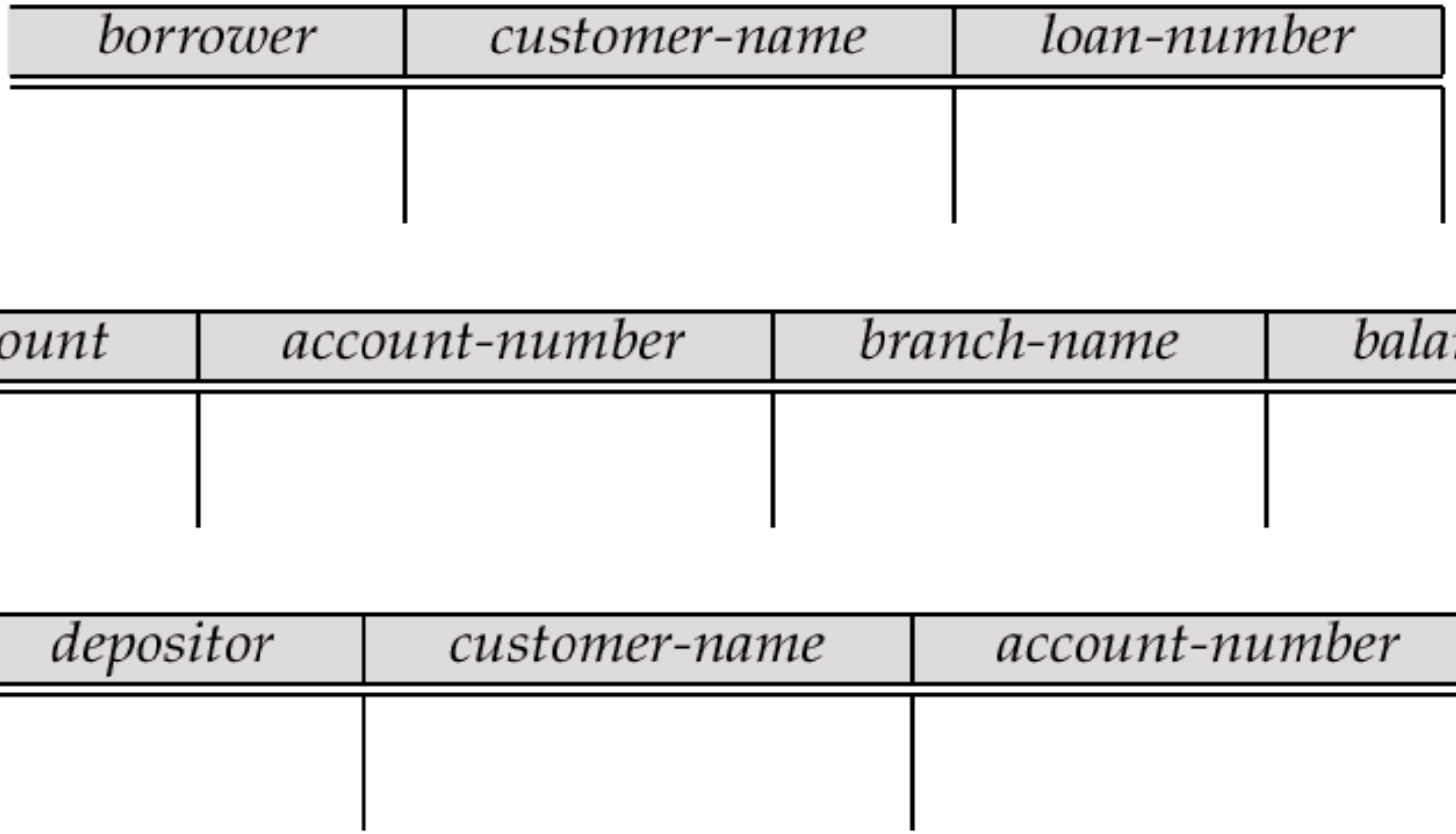
QBE περιγράμματα για το παράδειγμα της τράπεζας (1)

branch	*branch-name*	*branch-city*	*assets*

customer	*customer-name*	*customer-street*	*customer-city*

loan	*loan-number*	*branch-name*	*amount*

QBE περιγράμματα για το παράδειγμα της τράπεζας (1)



Queries σε μια σχέση

- Find all loan numbers at the Perryridge branch.

loan	*loan-number*	*branch-name*	*amount*
	P._*x*	Perryridge	

- _x είναι μεταβλητή (προαιρετικό, μπορεί να παραληφθεί στο παραπάνω query)
- Το P. σημαίνει δείξε (display)
- οι διπλές εγγραφές αφαιρούνται by default
- Για να διατηρηθούν τα διπλότυπα χρησιμοποιούμε P.ALL

loan	*loan-number*	*branch-name*	*amount*
	P.ALL.	Perryridge	

Queries σε μια σχέση (2)

- Δείξε όλες τι λεπτομέρειες των δανείων
  - Method 1:

loan	*loan-number*	*branch-name*	*amount*
	P._x	P._y	P._z

  - Method 2: Shorthand notation

loan	*loan-number*	*branch-name*	*amount*
P.			

Queries σε μια σχέση (3)

- Find the loan number of all loans with a loan amount of more than $700

loan	*loan-number*	*branch-name*	*amount*
	P.		>700

- Find names of all branches that are not located in Brooklyn

branch	*branch-name*	*branch-city*	*assets*
	P.	¬ Brooklyn	

Queries σε πολλές σχέσεις (1)

- Find the names of all customers who have a loan from the Perryridge branch.

loan	*loan-number*	*branch-name*	*amount*
	_x	Perryridge	

borrower	*customer-name*	*loan-number*
	P._y	_x

Queries σε πολλές σχέσεις (2)

- Find the names of all customers who have both an account and a loan at the bank.

depositor	*customer-name*	*account-number*
	P._*x*	

borrower	*customer-name*	*loan-number*
	_*x*	

Άρνηση στην QBE

- Find the names of all customers who have an account at the bank, but do not have a loan from the bank.

depositor	*customer-name*	*account-number*
	P._x	

borrower	*customer-name*	*loan-number*
¬	_x	

¬ means “there does not exist”

Άρνηση στην QBE

- Find all customers who have at least two accounts.

depositor	*customer-name*	*account-number*
	P._*x* _*x*	_*y* ¬ _*y*

¬ means “not equal to”

Κουτί συνθηκών

- Επιτρέπει την έκφραση περιορισμών σε μεταβλητές πεδίου που δεν εκφράζονται (εύκολα) με πίνακες-περιγράμματα
- Περίπλοκες συνθήκες μέσα σε κουτιά συνθηκών
- E.g. Find the loan numbers of all loans made to Smith, to Jones, or to both jointly

borrower	*customer-name*	*loan-number*
	_n	P._*x*

conditions
_n = Smith **or** *_n* = Jones

Κουτί συνθηκών (2)

- Find all account numbers with a balance between $1,300 and $1,500

account	*account-number*	*branch-name*	*balance*
	P.		_x

conditions
_x ≥ 1300 _x ≤ 1500

- Find all account numbers with a balance between $1,300 and $2,000 but not exactly $1,500.

account	*account-number*	*branch-name*	*balance*
	P.		_x

conditions
_x = (≥ 1300 **and** ≤ 2000 **and** ¬ 1500)

Η σχέση Result

- Find the *customer-name*, *account-number*, and *balance* for alll customers who have an account at the Perryridge branch.
  - We need to:
    - » Join *depositor* and *account.*
    - » Project *customer-name*, *account-number* and *balance.*
  - To accomplish this we:
    - » Create a skeleton table, called *result*, with attributes *customer-name*, *account-number*, and *balance.*
    - » Write the query.

The Result Relation (Cont.)

- The resulting query is:

account	*account-number*	*branch-name*	*balance*
	_y	Perryridge	_z

depositor	*customer-name*	*account-number*
	_x	_y

result	*customer-name*	*account-number*	*balance*
P.	_x	_y	_z

Ordering the Display of Tuples

- AO = ascending order; DO = descending order.
- E.g. list in ascending alphabetical order all customers who have an account at the bank

depositor	*customer-name*	*account-number*
	P.AO.	

- When sorting on multiple attributes, the sorting order is specified by including with each sort operator (AO or DO) an integer surrounded by parentheses.
- E.g. List all account numbers at the Perryridge branch in ascending alphabetic order with their respective account balances in descending order.

account	*account-number*	*branch-name*	*balance*
	P.AO(1).	Perryridge	P.DO(2).

Aggregate Operations

- The aggregate operators are AVG, MAX, MIN, SUM, and CNT
- The above operators must be postfixed with “ALL” (e.g., SUM.ALL.or AVG.ALL._x) to ensure that duplicates are not eliminated.
- E.g. Find the total balance of all the accounts maintained at the Perryridge branch.

account	*account-number*	*branch-name*	*balance*
		Perryridge	P.SUM.ALL.

Aggregate Operations (Cont.)

- UNQ is used to specify that we want to eliminate duplicates
- Find the total number of customers having an account at the bank.

depositor	*customer-name*	*account-number*
	P.CNT.UNQ.	

Query Examples

- Find the average balance at each branch.

account	*account-number*	*branch-name*	*balance*
		P.G.	P.AVG.ALL._x

- The “G” in “P.G” is analogous to SQL’s **group by** construct
- The “ALL” in the “P.AVG.ALL” entry in the *balance* column ensures that all balances are considered
- To find the average account balance at only those branches where the average account balance is more than $1,200, we simply add the condition box:

conditions
AVG.ALL._x > 1200

Modification of the Database – Deletion

- Deletion of tuples from a relation is expressed by use of a D. command. In the case where we delete information in only some of the columns, null values, specified by –, are inserted.
- Delete customer Smith

customer	*customer-name*	*customer-street*	*customer-city*
D.	Smith		

- Delete the *branch-city* value of the branch whose name is "Perryridge".

branch	*branch-name*	*branch-city*	*assets*
	Perryridge	D.	

Modification of the Database – Insertion

- Insertion is done by placing the I. operator in the query expression.
- Insert the fact that account A-9732 at the Perryridge branch has a balance of $700.

account	*account-number*	*branch-name*	*balance*
I.	A-9732	Perryridge	700

Modification of the Database – Insertion (Cont.)

- Provide as a gift for all loan customers of the Perryridge branch, a new $200 savings account for every loan account they have, with the loan number serving as the account number for the new savings account.

account	*account-number*	*branch-name*	*balance*
I.	_x	Perryridge	200

depositor	*customer-name*	*account-number*
I.	_y	_x

loan	*loan-number*	*branch-name*	*amount*
	_x	Perryridge	

borrower	*customer-name*	*loan-number*
	_y	_x

Modification of the Database – Updates

- Use the U. operator to change a value in a tuple without changing *all* values in the tuple. QBE does not allow users to update the primary key fields.
- Update the asset value of the Perryridge branch to $10,000,000.

branch	*branch-name*	*branch-city*	*assets*
	Perryridge		U.10000000

- Increase all balances by 5 percent.

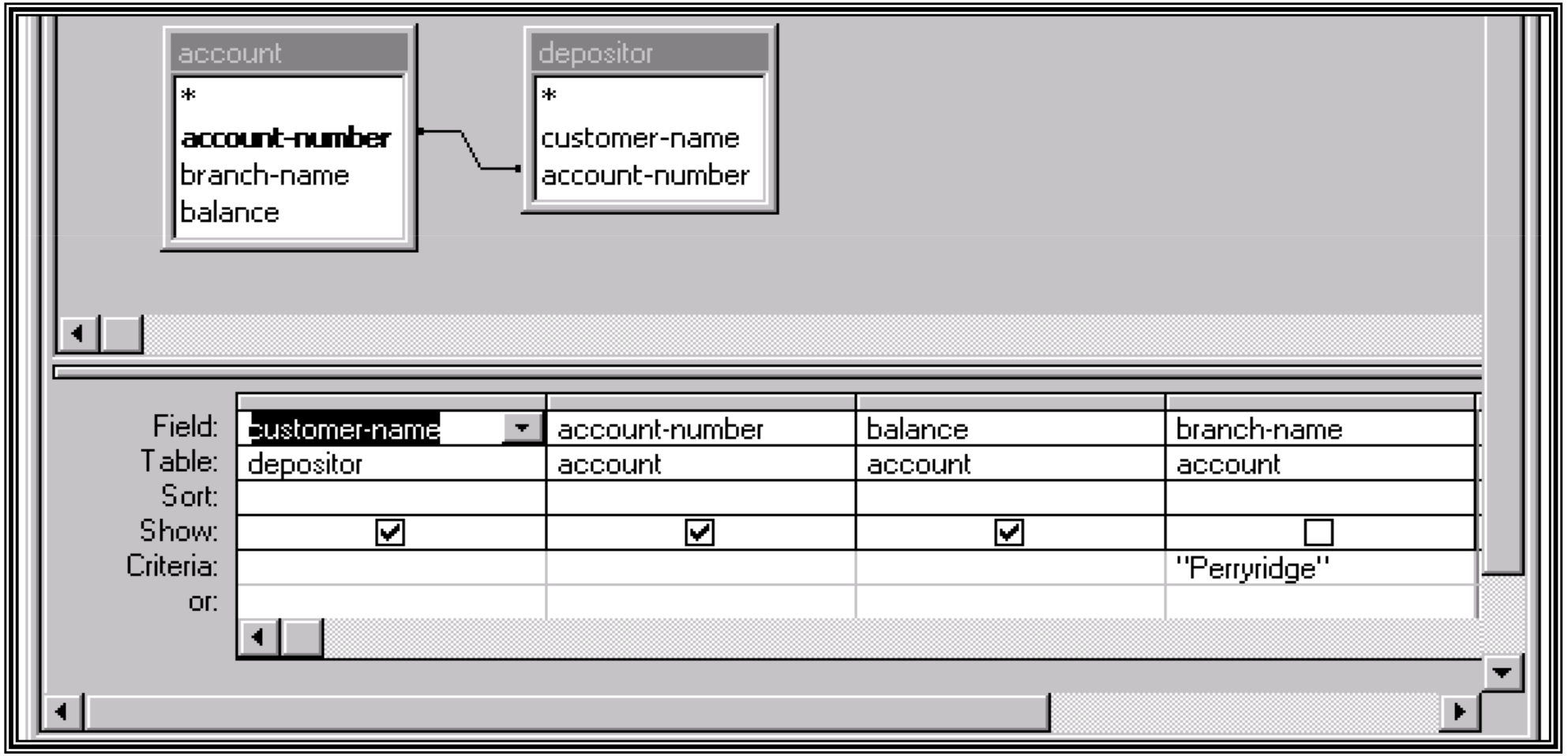
account	*account-number*	*branch-name*	*balance*
			U._*x* * 1.05

Microsoft Access QBE

- Microsoft Access supports a variant of QBE called Graphical Query By Example (GQBE)
- GQBE differs from QBE in the following ways
  - Attributes of relations are listed vertically, one below the other, instead of horizontally
  - Instead of using variables, lines (links) between attributes are used to specify that their values should be the same.
    - » Links are added automatically on the basis of attribute name, and the user can then add or delete links
    - » By default, a link specifies an inner join, but can be modified to specify outer joins.
  - Conditions, values to be printed, as well as group by attributes are all specified together in a box called the **design grid**

Παράδειγμα σε Microsoft Access QBE

- Example query: Find the *customer-name*, *account-number* and *balance* for all accounts at the Perryridge branch



Κλείνοντας

- Η QBE είναι μια κομψή και φιλική προς το χρήστη γλώσσα που βασίζεται στο σχεσιακό λογισμό πεδίου
- Είναι ιδιαίτερα εκφραστική (σχεσιακά πλήρης, αν και οι ενημερώσεις ληφθούν υπόψη).
- Απλές ερωταποκρίσεις είναι εξαιρετικά εύκολο να εκφραστούν στην QBE, με ένα ελάχιστο συντακτικό που πρέπει κανείς να θυμάται
- Η QBE Έχει επηρεάσει σε μεγάλο βαθμό τις γραφικές διευκολύνσεις για queries που σήμερα προσφέρονται σε πολλά προϊόντα, περιλαμβανομένης και της Microsoft Access.