Αντικειμενοστραφείς Γλώσσες Προγραμματισμού για Βάσεις Δεδομένων

Database System Concepts

Δομή Αντικειμένων

- Ένα **αντικείμενο** αντιστοιχεί σε μία **οντότητα** σε ένα E-R
- Το αντικειμενοστραφές μοντέλο βασίζεται στην **ενσωμάτωση** δεδομένων και κώδικα που σχετίζονται με ένα αντικείμενο, σε μία μόνο μονάδα της οποίας τα περιεχόμενα δεν φαίνονται στον εξωτερικό κόσμο
- Όλες οι **αλληλεπιδράσεις** μεταξύ ενός αντικειμένου και του υπόλοιπου συστήματος γίνεται μέσω **μηνυμάτων**
- Η διασύνδεση μεταξύ ενός αντικειμένου και του υπόλοιπου συστήματος ορίζεται από ένα σύνολο από **επιτρεπόμενα** μηνύματα
- Γενικά ένα αντικείμενο έχει σχετικά με αυτό:
  - Ένα σύνολο από **μεταβλητές** που περιέχουν τα δεδομένα για το αντικείμενο. Οι μεταβλητές αντιστοιχούν στις **ιδιότητες** του μοντέλου E-R
  - Ένα σύνολο από **μηνύματα** στα οποία αποκρίνεται το αντικείμενο. Κάθε μήνυμα μπορεί να έχει 0,1 ή περισσότερες παραμέτρους
  - Ένα σύνολο από **μεθόδους**, καθεμία εκ των οποίων είναι ένα σώμα από κώδικα που χειρίζεται ένα μήνυμα. Μία μέθοδος επιστρέφει μία τιμή σε απόκριση του μηνύματος

Αντικειμενοστραφείς Γλώσσες

Χρησιμοποιούμε την ιδέα του προσανατολισμού στο αντικείμενο σε μία γλώσσα που χρησιμοποιούμε για να χειριζόμαστε τη βάση δεδομένων.

- Μία επιλογή είναι να επεκταθεί μία γλώσσα χειρισμού δεδομένων, όπως η SQL, προσθέτοντας περίπλοκους τύπους και προσανατολισμό στα αντικείμενα. Τα συστήματα που παρέχουν αντικειμενοστραφείς επεκτάσεις σε σχεσιακά συστήματα ονομάζονται συστήματα σχεσιακών αντικειμένων.

- Μία άλλη επιλογή είναι να πάρουμε μία υπάρχουσα γλώσσα προγραμματισμού (C++, Java) και να την επεκτείνουμε ώστε να χειρίζεται την βάση δεδομένων. Τέτοιες γλώσσες ονομάζονται **μόνιμες γλώσσες προγραμματισμού (persistent programming)**

Μόνιμες γλώσσες προγραμματισμού

- Οι γλώσσες βάσεων δεδομένων διαφέρουν από τις παραδοσιακές γλώσσες προγραμματισμού στο ότι χειρίζονται κατευθείαν δεδομένα που είναι μόνιμα, δηλαδή τα δεδομένα που συνεχίζουν να υπάρχουν ακόμα και αφού τερματίσει το πρόγραμμα που τα δημιούργησε
- Μία σχέση σε μία βάση δεδομένων και οι εγγραφές μιας σχέσης είναι παραδείγματα μόνιμων δεδομένων
- Αντίθετα, τα μόνα μόνιμα δεδομένα που χειρίζονται οι παραδοσιακές γλώσσες προγραμματισμού είναι τα αρχεία (files)
- Μία **μόνιμη γλώσσα προγραμματισμού** είναι μία γλώσσα προγραμματισμού που έχει επεκταθεί με κατάλληλες δομές ώστε να μπορεί να χειρίζεται **μόνιμα** δεδομένα.
- Οι μόνιμες γλώσσες προγραμματισμού **διαφέρουν** από τις γλώσσες με ενσωματωμένη (embedded) SQL
- ODMG C++ (Object Database Management Group)
  - C++ OML (Object Manipulation Language)
  - C++ ODL (Object Definition Language)
- Ομοίως για ODMG Java

Object-Relational Μοντέλα Δεδομένων

- Επεκτείνουμε τα σχεσιακά μοντέλα δεδομένων και τις σχεσιακές δομές χρησιμοποιώντας αντικειμενοστραφή προσανατολισμό
- Επιτρέπουμε σε πεδία εγγραφών να έχουν περίπλοκους τύπους (complex types), π.χ. μη-ατομικές τιμές όπως εμφωλιασμενες σχέσεις (nested relations).
- Καθώς το μοντέλο μας επεκτείνεται, διατηρούμε τις σχεσιακές ιδιότητες, ιδιαίτερα ότι έχει να κάνει με την προσπέλαση των δεδομένων
- Ανερχόμενη συμβατότητα με υπάρχοντες σχεσιακές γλώσσες προγραμματισμού.

Περίπλοκοι Τύποι Δεδομένων

- Κίνητρο:
  - Επιτρεπτά μη ατομικά πεδία (non-atomic domains) (ατομικό ≡ αδιαίρετο)
  - Παράδειγμα μη-ατομικών πεδίων: σύνολο ακεραίων ή σύνολο εγγραφών (tuples)
  - Επιτρέπει πιο διαισθητική μοντελοποίηση εφαρμογών με περίπλοκα δεδομένα
- Διαισθητικός Ορισμός:
  - Επιτρέπει σχέσεις μέσα σε σχέσεις
  - Διατηρεί τη μαθηματική θεμελίωση του σχεσιακού μοντέλου
  - Προφανώς, παραβιάζει την 1NF.

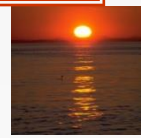
Παράδειγμα Εμφωλιασμένης Σχέσης

- Παράδειγμα: πληροφοριακό σύστημα βιβλιοθήκης
- Κάθε βιβλίο έχει:
  - Τίτλο,
  - Σύνολο συγγραφέων,
  - Εκδότη, και
  - Ένα σύνολο από λέξεις-κλειδιά
- Η Non-1NF σχέση *books*

title	*author-set*	*publisher*	*keyword-set*
		(*name*, *branch*)	
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

1 NF (flat books)

title	*author*	*pub-name*	*pub-branch*	*keyword*
Compilers	Smith	McGraw-Hill	New York	parsing
Compilers	Jones	McGraw-Hill	New York	parsing
Compilers	Smith	McGraw-Hill	New York	analysis
Compilers	Jones	McGraw-Hill	New York	analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web



4NF Αποσύνθεση Εμφωλιασμένων Σχέσεων

- Εξαλείφει την *ατεχνία-αδεξιότητα* των *flat-books* υποθέτοντας ότι ισχύουν οι ακόλουθες πολλών-τιμών εξαρτήσεις (multivalued dependencies):
  - *title* $\twoheadrightarrow$ *author*
  - *title* $\twoheadrightarrow$ *keyword*
  - *title* $\twoheadrightarrow$ *pub-name, pub-branch*
- Αποσυνθέτει το *flat-book* σε 4NF χρησιμοποιώντας τα σχεσιακά σχήματα:
  - (*title, author*)
  - (*title, keyword*)
  - (*title, pub-name, pub-branch*)

4NF Decomposition of *flat–books*

title	*author*
Compilers	Smith
Compilers	Jones
Networks	Jones
Networks	Frick

authors

title	*keyword*
Compilers	parsing
Compilers	analysis
Networks	Internet
Networks	Web

keywords

title	*pub-name*	*pub-branch*
Compilers	McGraw-Hill	New York
Networks	Oxford	London

books4

Προβλήματα στο 4NF Σχήμα

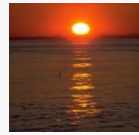
- Ο 4NF σχεδιασμός απαιτεί χρήση συνδέσμων (joins) στα queries.
- Ο 1NF σχεδιασμός (*flat-books)*:
  - Εξαλείφει την ανάγκη για join queries,
  - Χάνει την ένα-προς-ένα αντιστοίχηση μεταξύ tuples και documents.
  - Εμφανίζει περιττή επανάληψη (redundancy) πληροφορίας
- **Η αναπαράσταση με Nested relations είναι πολύ πιο φυσική εδώ (στο συγκεκριμένο παράδειγμα).**

title	*author-set*	*publisher*	*keyword-set*
		(name, branch)	
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

Nested Relation Books

Περίπλοκοι Τύποι στην SQL:1999

- Η SQL για να υποστηρίξει complex types περιέχει:
  - Τύπους Συλλογών (collections types) και τύπους μεγάλων αντικειμένων (large object types)
    - Nested relations: π.χ. collection types
  - Δομημένους Τύπους (Structured types)
    - Nested record structures: π.χ. σύνθετες ιδιότητες
  - Κληρονομικότητα
  - Προσανατολισμός στα αντικείμενα
    - object identifiers και object references
- Η περιγραφή μας βασίζεται κυρίως στο SQL:1999 πρότυπο
  - Δεν υλοποιείται κατά κόρον, αλλά
  - Κάποια κομμάτια της έχουν κάνει την παρουσία τους σε εμπορικές βάσεις δεδομένων



Δομημένοι Τύποι και Κληρονομικότητα στην SQL

- Structured types δηλώνονται και χρησιμοποιούνται στην SQL

```
create type Name as
    (firstname      varchar(20),
     lastname       varchar(20))

create type Address as
    (street         varchar(20),
     city           varchar(20),
     zipcode        varchar(20))
```

- Structured types χρησιμοποιούνται για δημιουργία πινάκων με σύνθετες ιδιότητες

```
create table customer (
    name        Name,
    address     Address,
    dateOfBirth date)
```

- Η αναφορά στα συστατικά των τύπων γίνεται χρησιμοποιώντας Dot notation:
- Π.χ. *name.firstname*

Δομημένοι Τύποι (Συν.)

- Row types ορισμένοι από το χρήστη, π.χ.

  create type *CustomerType* **as** (

  name Name,

  address Address,

  dateOfBirth **date**)

- Εν συνεχεία, δημιουργία πινάκων των οποίων οι γραμμές είναι ένας τύπος ορισμένος από το χρήστη

  create table *customer* **of** *CustomerType*

Μέθοδοι

- Μπορούμε σε έναν δομημένο τύπο να δηλώσουμε και μία μέθοδο.

  method *ageOnDate* (*onDate* **date**)

  returns interval year

- Το σώμα της μεθόδου δίνεται ξεχωριστά.

  create instance method *ageOnDate* (*onDate* **date**)

  returns interval year

  for *CustomerType*

  begin

  return *onDate* - **self**.*dateOfBirth*;

  end

- Μπορούμε να υπολογίσουμε τώρα την ηλικία κάθε πελάτη:

  select *name.lastname, ageOnDate* (**current_date**)

  from *customer*

Κληρονομικότητα

- Υποθέστε ότι για ανθρώπους (people) έχουμε ορίσει τον παρακάτω τύπο:

```
create type Person
    (name varchar(20),
    address varchar(20))
```

- Χρησιμοποιώ κληρονομικότητα (inheritance) για να ορίσω student και teacher types

```
create type Student
  under Person
  (degree       varchar(20),
   department   varchar(20))
create type Teacher
  under Person
  (salary       integer,
   department   varchar(20))
```

Πολλαπλή Κληρονομικότητα

- SQL:1999 και SQL:2003 δεν υποστηρίζουν multiple inheritance
- Αν το σύστημά μας υποστηρίζει multiple inheritance, μπορούμε να ορίσουμε έναν teaching assistant τύπο ως εξής:

  create type *Teaching Assistant*
  under *Student, Teacher*

- Για να αποφύγουμε το conflict μεταξύ των δύο *department* εμφανίσεων μπορούμε να γράψουμε:

  create type *Teaching Assistant*
  under
  Student **with** (*department* **as** *student_dept*),
  Teacher **with** (*department* **as** *teacher_dept*)

Περιορισμοί Συνέπειας (Consistency Requirements) για Υποπίνακες (Subtables)

- Περιορισμοί συνέπειας για υπό-πίνακες και σούπερ-πίνακες.
  - Κάθε tuple ενός supertable (π.χ. *people)* πρέπει να αντιστοιχεί τουλάχιστον σε ένα tuple καθενός από τα subtables (π.χ. *students* και *teachers)*
  - Επιπλέον περιορισμός της SQL:1999:

    Κάθε tuple στον πίνακα people το οποίο κληρονομείται στα tuples των υπό-πινάκων students και teachers γίνεται inserted στον πίνακα people μία φορά ασχέτως αν αντιστοιχεί σε περισσότερες εμφανίσεις μέσα στους υπό-πίνακες

Τύποι Array και Multiset στην SQL

- Παραδείγματα array και multiset δηλώσεων:

```
create type Publisher as
    (name          varchar(20),
     branch        varchar(20))
create type Book as
    (title         varchar(20),
     author-array  varchar(20) array [10],
     pub-date      date,
     publisher     Publisher,
     keyword-set   varchar(20) multiset )
create table books of Book
```

- Παρόμοιο με τα nested relation books, αλλά με array για τους συγγραφείς, αντί για set.

Collection Values

- Κατασκευή Array
  - **array** [‘Silberschatz’,`Korth’,`Sudarshan’]
- Multisets
  - **multisetset** [‘computer’, ‘database’, ‘SQL’]
- Για να δημιουργήσουμε ένα tuple του τύπου που ορίσαμε στη σχέση *books*: (‘Compilers’, **array**[`Smith’,`Jones’],
  Publisher (`McGraw-Hill’,`New York’),
  multiset [`parsing’,`analysis’])
- Για να ενθέσουμε το προηγούμενο tuple στη σχέση *books, γράφουμε:*

insert into *books*
values
(‘Compilers’, **array**[`Smith’,`Jones’],
Publisher (`McGraw-Hill’,`New York’),
multiset [`parsing’,`analysis’])

Queries πάνω σε Collections

- Για να βρούμε όλα τα βιβλία που περιέχουν ως keyword τη λέξη “database” γράφουμε,

  select *title*
  from *books*
  where ‘database’ **in** (**unnest**(*keyword-set*))

- Μπορούμε να προσπελάσουμε ατομικά στοιχεία ενός array χρησιμοποιώντας δείκτες
  - Π.χ.: Εάν ξέρουμε ότι ένα βιβλίο έχει 3 authors, μπορούμε να γράψουμε:

    select *author-array*[1], *author-array*[2], *author-array*[3]
    from *books*
    where *title* = `Database System Concepts’

Unnesting

- Ο μετασχηματισμός μίας nested relation σε μία φόρμα με λιγότερα (ή καθόλου) relation-valued attributes ονομάζεται **unnesting**.
- Π.χ.

select *title*, *A.author* **as** *author*, *publisher.name* **as** *pub_name*, *publisher.branch* **as** *pub_branch*, *K.keyword* **as** *keyword*

from *books* **as** *B*, **unnest**(*B.author_array*) **as** *A* (*author*), **unnest** (*B.keyword_set*) **as** *K* (*keyword*)

title	*author*	*pub-name*	*pub-branch*	*keyword*
Compilers	Smith	McGraw-Hill	New York	parsing
Compilers	Jones	McGraw-Hill	New York	parsing
Compilers	Smith	McGraw-Hill	New York	analysis
Compilers	Jones	McGraw-Hill	New York	analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web

flat-books

Nesting

- **Nesting** είναι το αντίθετο του unnesting και δημιουργεί collection-valued attributes
- Σημ: Η γλώσσα SQL:1999 δεν υποστηρίζει nesting

- Το Nesting γίνεται με τρόπο παρόμοιο της συνάρτησης συνάθροισης (aggregation), αλλά χρησιμοποιώντας τη συνάρτηση **colect**() στη θέση μιας συνάρτησης συνάθροισης, για να δημιουργήσουμε ένα multiset

- Για να κάνουμε nest μαζί authors και keywords:

select *title*, **collect** (*author*) **as** *author_set*,
Publisher (*pub_name, pub_branch*) **as** *publisher*,
collect (*keyword*) **as** *keyword_set*
from *flat-books*
group by *title*, *publisher*

title	*author-set*	*publisher* (*name, branch*)	*keyword-set*
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

Ταυτότητες (Identifiers) παραγόμενες από τον χρήστη

- Παράδειγμα:
- **create type** *Person*
  (*name* **varchar**(20)
  address **varchar**(20))
- ***Δεν κάνω ref τη στιγμή που δημιουργώ το type
- **ref using varchar**(20)
  create table *people* **of** *Person*
  ref is *person_id* **user generated**
- ******* Κάνω το ref τη στιγμή που δημιουργώ τον πίνακα
- Όταν δημιουργούμε ένα tuple, παρέχουμε μία μοναδική τιμή για τον identifier:
  - **insert into** *people* (*person_id, name, address*) **values**
    (‘01284567’, ‘John’, `23 Coyote Run’)

Ταυτότητα Αντικειμένου και Τύποι Αναφορών

- Ορίζουμε έναν τύπο *Department* με πεδία *name* και *head* με το δεύτερο να είναι αναφορά (reference) στον τύπο *Person*:

  create type *Department* (
  name **varchar** (20),
  head **ref** (*Person*))

***Κάνω το ref τη στιγμή που δημιουργώ το type

- Μπορούμε μετά να δημιουργήσουμε τον πίνακα *departments* ως εξής:

  create table *departments* **of** *Department*

- Για να δημιουργήσουμε ένα tuple με reference value, πρέπει πρώτα να δημιουργήσουμε το tuple με null reference και μετά να θέσουμε ξεχωριστά το reference:

  insert into *departments*
  values (`CEID', null)
  update *departments*
  set *head* = (**select** *p.person_id*
  from *people* **as** *p*
  where *name* = `Gallopoulos')
  where *name* = `CEID'

****Προσοχή!!! Ο πίνακας είναι ο people, person είναι το complex type.*

Path Expressions

- Βρες τα ονόματα και τις διευθύνσεις των επικεφαλής (heads) όλων των τμημάτων (departments):

  select *head –>name*, *head –>address*
  from *departments*

- Μία έκφραση όπως η “head–>name” ονομάζεται **path expression**
- Path expressions βοηθούν στο να αποφεύγονται τα explicit joins
  - Αν ο επικεφαλής ενός τμήματος δεν ήταν reference, για να βρεθούν οι διευθύνσεις έπρεπε να κάνουμε join τον πίνακα *departments* με το πίνακα *people*.
  - Κάνει τον χρήστη να μπορεί να εκφράσει το query πολύ ευκολότερα.

Σύγκριση

- **Persistent-programming-language-based OODBs**
  - Περίπλοκοι τύποι δεδομένων, ενσωμάτωση (integration) με γλώσσες προγραμματισμού (C, C++, Java).
  - *** ΔΥΣΧΡΗΣΤΗ
- **Object-relational SQL**
  - Περίπλοκοι τύποι δεδομένων, **ισχυρές γλώσσες ερωτημάτων**, υψηλή προστασία.
  - *** ΕΥΧΡΗΣΤΗ