ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΕΙΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

Π. ΒΑΣΙΛΕΙΑΔΗΣ, Σ. ΣΚΙΑΔΟΠΟΥΛΟΣ

4.1 ΓΕΝΙΚΑ

Γύρω στα 1986, η προσπάθεια συσχετισμού αντικειμένων και βάσεων δεδομένων, ήταν συναρπαστική. Από την πλευρά των βάσεων δεδομένων υπήρχε η τάση επέκτασης του σχεσιακού μοντέλου, που μόλις πριν από λίγο καιρό είχε καθιερωθεί. Από την πλευρά των γλωσσών προγραμματισμού υπήρχε η τάση πρόσθεσης χαρακτηριστικών όπως διάρκεια στα δεδομένα. Από τη συνάντηση των δύο τάσεων προέκυψε μια ποικιλία απόψεων: κάποιοι πίστευαν ότι ο συνδυασμός χαρακτηριστικών από τις αντικειμενοστρεφείς γλώσσες προγραμματισμού και τις βάσεις δεδομένων θα δημιουργούσε ένα νέο είδος συστημάτων βάσεων δεδομένων. Την ίδια στιγμή, κάποιοι άλλοι πίστευαν ότι, με τη βοήθεια των αντικειμένων, ο κόσμος θα έπρεπε να προσανατολιστεί σε εργαλεία (toolkits) που θα βοηθούσαν τους προγραμματιστές να κατασκευάσουν εξειδικευμένα συστήματα διαχείρισης βάσεων δεδομένων. Στη συνέχεια του κεφαλαίου αυτού, θα παρουσιάσουμε το πώς οι διαφορετικές αυτές προσεγγίσεις κατέληξαν στη δημιουργία των αντικειμενοστρεφών συστημάτων βάσεων δεδομένων. Προηγουμένως, όμως, θα κάνουμε μια μικρή εισαγωγή στις έννοιες του προσανατολισμού στα αντικείμενα.

Προσανατολισμός στ' Αντικείμενα (Object Orientation) σημαίνει μια "μοντελοποίηση του λογισμικού (software) και κάποιες αρχές ανάπτυξης που διευκολύνουν την κατασκευή σύνθετων συστημάτων από ξεχωριστά συστατικά" [Kh93].

Τι είναι *"Αντικείμενο"* (Object); "Αντικείμενο είναι μια ξεχωριστή οντότητα, που προσπαθεί να μοντελοποιήσει και να προσεγγίσει όσον το δυνατό καλύτερα, το φυσικό κόσμο." [CK86].

Ένα αντικείμενο μπορεί να διενεργήσει ένα σύνολο δραστηριοτήτων. Το σύνολο, αυτό, των δραστηριοτήτων του, καθορίζει και τη *συμπεριφορά* του (object's behaviour). Ο ορισμός της συμπεριφοράς ενός αντικειμένου αποτελείται από τρία μέρη:

- *τη διαπροσωπεία* (interface)
- *τον κώδικα* (code)
- *τα δεδομένα* (data)

Η διαπροσωπεία ενός αντικειμένου αποτελείται από ένα σύνολο εντολών που κάθε μία επιτελεί μία συγκεκριμένη εργασία. Ένα αντικείμενο ζητά από ένα άλλο να εκτελέσει μία εντολή στέλνοντάς του ένα *μήνυμα*. Ο έλεγχος μεταβιβάζεται από το αντικείμενο *αποστολέα* (sender) στο αντικείμενο *παραλήπτη* (receiver), μέχρι το δεύτερο να ολοκληρώσει την ζητηθείσα εντολή. Στη συνέχεια, ο έλεγχος επιστρέφει στον αποστολέα. Ένα μήνυμα μπορεί να περιέχει πληροφορία για τον αποστολέα, με τη μορφή ορισμάτων. Το αντικείμενο παραλήπτης, από την πλευρά του, μπορεί να επιστρέψει μια τιμή πίσω στο αιτούν αντικείμενο. Κάθε μήνυμα λαμβάνει ένα όνομα το οποίο ονομάζεται *επιλογέας* (selector), και είναι το όνομα αυτό που αποστέλλεται στο αντικείμενο παραλήπτη.

Ένα άλλο σύνηθες φαινόμενο είναι το αντικείμενο που λαμβάνει ένα μήνυμα να στέλνει με τη σειρά του ένα άλλο μήνυμα σε ένα τρίτο αντικείμενο κ.ο.κ., όπως φαίνεται στο Σχήμα 4.1.

Σχήμα 4.1 Ανταλλαγή μηνυμάτων μεταξύ αντικειμένων

Ο κώδικας του κάθε μηνύματος εκτελείται κάθε φορά που ένα αντικείμενο λαμβάνει το εν λόγω μήνυμα. Ο κώδικας που σχετίζεται με κάθε μήνυμα ονομάζεται *μέθοδος*. Όταν ένα αντικείμενο λαμβάνει ένα μήνυμα, καθορίζει ποια μέθοδος πρέπει να εκτελεστεί και της περνάει τον έλεγχο. Ένα αντικείμενο έχει τόσες μεθόδους, όσα και μηνύματα. Εν γένει, το όνομα της μεθόδου είναι το ίδιο με το όνομα του μηνύματος.

Τα δεδομένα ενός αντικειμένου χρησιμοποιούνται για να διατηρούν την πληροφορία γύρω από αυτό. Τα αντικείμενα διατηρούν μεταβλητές, οι οποίες υλοποιούν τα δεδομένα και ονομάζονται *instance variable*[1]. Είναι σαφές ότι αυτού του είδους οι μεταβλητές ζουν όσο και το αντικείμενο.

Ένα από τα χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού είναι και το γεγονός, ότι ένα αντικείμενο μπορεί να έρθει σε επικοινωνία με ένα άλλο, μόνο μέσω του interface του δεύτερου. Η λογική και τα δεδομένα κάθε αντικειμένου αποκρύπτονται. Το interface ενθυλακώνει τον κώδικα και τα δεδομένα κάθε αντικειμένου, γι' αυτό και το χαρακτηριστικό αυτό, του αντικειμενοστρεφούς προγραμματισμού, ονομάζεται *ενθυλάκωση* (encapsulation)

Είναι χαρακτηριστικό φαινόμενο, το γεγονός ότι πολλά αντικείμενα επιδεικνύουν την ίδια ακριβώς συμπεριφορά και διαφέρουν μόνο στην τιμή των instance variables. Για να μην επαναλαμβάνουμε τη συγγραφή του κώδικα για τις μεθόδους, τα δεδομένα και το interface, φροντίζουμε να ομαδοποιήσουμε τα αντικείμενα σε *κλάσεις*. Κλάση είναι μια ομάδα από αντικείμενα με παρόμοια σημασιολογία και, εν γένει, παρόμοια δομή και λειτουργίες. Είναι χαρακτηριστικό, ότι στον αντικειμενοστρεφή προγραμματισμό, δεν ορίζουμε αντικείμενα,

[1]Το όνομα instance variables είναι σύμβαση της Smalltalk. Στη C++ οι μεταβλητές αυτές ονομάζονται data members. Στο εξής, εμείς θα χρησιμοποιούμε την ορολογία της Smalltalk, εκτός από τις σαφείς αναφορές στη C++.

αλλά τις κλάσεις στις οποίες ανήκουν. Ένα *στιγμιότυπο* (instance) είναι ένα αντικείμενο που ανήκει στην κλάση αυτή, και έχει τα δικά του δεδομένα.

Δομούμε τις κλάσεις σε *ιεραρχίες κλάσεων*, έτσι ώστε να κατασκευαστεί ένα δέντρο από κλάσεις. Κάθε κλάση κληρονομεί από την πατρική της όλα τα χαρακτηριστικά της δεύτερης, στα οποία, αν υπάρχει ανάγκη, προσθέτει και τα ιδιαίτερα δικά της. Το φαινόμενο αυτό ονομάζεται *κληρονομικότητα.* Με τον τρόπο αυτό, μπορούμε να ομαδοποιούμε αλλά και να εξειδικεύουμε τα αντικείμενα με μια δενδρική δομή εύκολα αντιληπτή σημασιολογικά, η οποία ταυτοχρόνως μας επιτρέπει να εξοικονομούμε σημαντικό τμήμα κώδικα (αφού δε χρειάζεται να ορίσουμε ξανά τις λειτουργίες και τη δομή μιας θυγατρικής κλάσης -αρκεί να έχει περιγραφεί ο πατέρας της).

Τέλος, άλλο ένα χαρακτηριστικό του αντικειμενοστρεφούς προγραμματισμού είναι το γεγονός ότι διαφορετικά αντικείμενα αντιδρούν με διαφορετικό τρόπο στο ίδιο μήνυμα. (Έχουν, δηλαδή, διαφορετική υλοποίηση της μεθόδου για το ίδιο μήνυμα). Το φαινόμενο ονομάζεται *πολυμορφισμός* (polymorphism).

Κλείνοντας αυτή τη σύντομη εισαγωγή για τον αντικειμενοστρεφή προγραμματισμό, συνοψίζουμε τα βασικά του χαρακτηριστικά, ως εξής:

- Ένα αντικείμενο αποτελείται από δεδομένα, λογική και το interface που τα ενθυλακώνει και που καθορίζει τη συμπεριφορά του αντικειμένου.
- Τα αντικείμενα επικοινωνούν με μηνύματα.
- Μια μέθοδος είναι η υλοποίηση ενός μηνύματος.
- Αντικείμενα ομοειδή ανήκουν στην ίδια κλάση.
- Ο μόνος τρόπος για να υπάρξει επικοινωνία με ένα αντικείμενο είναι μέσω του interface του.
- Δύο αντικείμενα μπορεί να υλοποιούν το ίδιο μήνυμα μέσω διαφορετικών μεθόδων.

4.2 ΒΑΣΙΚΕΣ ΑΡΧΕΣ ΕΝΟΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΜΟΝΤΕΛΟΥ ΔΕΔΟΜΕΝΩΝ (Object-Oriented Data Model)

Σύμφωνα με το [Mai89] η έννοια *μοντέλο δεδομένων* (data model) έχει διττή σημασία: σημαίνει αφενός ένα συγκεκριμένο σχήμα ή γενικά, περιγραφή των metadata σε μια εφαρμογή, και αφετέρου σημαίνει τον τρόπο με τον οποίο σχήματα ή περιγραφές δεδομένων εκφράζονται σε ένα σύστημα. Για τη δεύτερη ερμηνεία, προς την οποία και εμείς θα προσανατολιστούμε στη συνέχεια, υπάρχουν επίσης, διάφορες παραλλαγές -εμείς θα υιοθετήσουμε τον ορισμό που δίνεται στο [Di93]:

"Ένα μοντέλο δεδομένων είναι ένα σύνολο από λογικά εργαλεία για την περιγραφή της αναπαράστασης της πληροφορίας σε δεδομένα. Περιλαμβάνει δε, ζητήματα σχετικά με:

- τύπους και δομές δεδομένων (types and data structures)
- λειτουργίες (operations)
- περιορισμούς ακεραιότητας (integrity constraints)"[2]

Στην ενότητα αυτή, θα ασχοληθούμε με ζητήματα που αφορούν το αντικειμενοστρεφές μοντέλο δεδομένων. Κυρίως θα βασιστούμε στο μοντέλο που προτείνεται στο [Di93], το οποίο με τη σειρά του βασίζεται στο μοντέλο που διαφαίνεται να υπάρχει στο [At+89]. Επιπλέον,

[2] Αξίζει να σημειωθεί, όμως, ότι παρ' όλα αυτά, στο ίδιο σύγγραμμα, ο Dittrich παρατηρεί «...(σ.σ. ένα μοντέλο) είναι λιγότερο ένα μοντέλο αυτό καθεαυτό, και περισσότερο ένα πλαίσιο για να συλλάβουμε μοντέλα....»

όπου κρίνεται αναγκαίο, θα αναφέρουμε τις ενστάσεις και τους προβληματισμούς άλλων επιστημόνων.

Η γενική ιδέα είναι ότι ένα αντικειμενοστρεφές μοντέλο δεδομένων προσπαθεί να ενώσει ιδέες που διέπουν αντικειμενοστρεφή συστήματα, με καθιερωμένες ιδέες από ισχύοντα μοντέλα δεδομένων.

4.2.1 Δομή των Δεδομένων

Ένα αντικειμενοστρεφές μοντέλο δεδομένων βασίζεται στη γενική ιδέα του *αντικειμένου* (object) και της *κλάσης* (class).

Σύμφωνα με το [Di93], ένα αντικείμενο μπορεί να θεωρηθεί σαν μία τετράδα:

<OID, τιμή, κατάσταση, κλάση>

και μια κλάση σαν μία εντεκάδα:

<όνομα κλάσης, τύπος της τιμής των στιγμιοτύπων, τύπος της κατάστασης των στιγμιοτύπων, {μηνύματα στιγμιοτύπων}, {μέθοδοι στιγμιοτύπων}, τιμή της κλάσης, κατάσταση της κλάσης, {μηνύματα της κλάσης}, τύπος της τιμής της κλάσης, τύπος της κατάστασης της κλάσης, {μέθοδοι της κλάσης}>

Στη συνέχεια θα αναλύσουμε τα προαναφερθέντα στοιχεία (όχι πάντα με τη σειρά που αναφέρθηκαν) εισάγοντας και επιπλέον βασικά στοιχεία ενός αντικειμενοστρεφούς μοντέλου δεδομένων.

OID - Ταυτότητα αντικειμένου (Object identity)

Σαφής ορισμός για την έννοια της *ταυτότητας αντικειμένου* (OID - object identifier ή object identity), δεν υπάρχει. Θα μπορούσε να αναφέρει κανείς τον ορισμό που δίνουν οι Copeland και Khoshafian [CK86]:

"Ταυτότητα είναι η ιδιότητα ενός αντικειμένου που ξεχωρίζει το κάθε αντικείμενο από όλα τα άλλα".

Για να υλοποιηθεί η έννοια της ταυτότητας αντικειμένου, κάνουμε την παραδοχή, ότι υπάρχει ένας (θεωρητικά άπειρος) αριθμός *αναγνωριστικών* (identifiers), τα οποία ικανοποιούν τις παρακάτω συνθήκες:

- ένα αναγνωριστικό σχετίζεται με κάθε μη βασικό αντικείμενο (nonbase object).
- το αναγνωριστικό σχετίζεται με το αντικείμενο κατά τη διάρκεια της δημιουργίας του αντικειμένου, και παραμένει συσχετισμένο με το αντικείμενο, ανεξάρτητα από τις αλλαγές στην κατάσταση του αντικειμένου.
- η αντιστοιχία μεταξύ αναγνωριστικών και αντικειμένων στη Βάση Δεδομένων πρέπει να είναι ένα προς ένα. Η ιδιότητα αυτή ονομάζεται *συνέπεια* (consistency). Το αναγνωριστικό καθορίζει μονοσήμαντα το αντικείμενο, το οποίο με τη σειρά του παραμένει άρρηκτα συνδεδεμένο μαζί του, καθ' όλη τη διάρκεια της ύπαρξής του.

Θα μπορούσε κανείς να πει, ότι η έννοια της ταυτότητας αντικειμένου είναι μια σύλληψη, ώστε το κάθε αντικείμενο να έχει μια οντότητα ανεξάρτητη της τιμής του. Έτσι, η ταύτιση και η ισότητα αντικειμένων είναι έννοιες ξέχωρες μεταξύ τους.

Ο τρόπος με τον οποίο η ταυτότητα καθορίζει μονοσήμαντα ένα αντικείμενο, φαίνεται ολοκάθαρα σε δύο ιδιότητες των αντικειμένων: το μοίρασμα αντικειμένων (object sharing) και τις ενημερώσεις αντικειμένων (object updates) [At+89].

- *Μοίρασμα Αντικειμένων*: δύο αντικείμενα μπορούν να μοιράζονται ένα συστατικό. Έστω ότι η κλάση Person έχει ως χαρακτηριστικά όνομα, ηλικία και ένα σύνολο από παιδιά. Ας

υποθέσουμε ότι ο Πέτρος και η Μαρία έχουν από ένα παιδί με όνομα Γιάννη και ηλικία 15 χρονών. Έχουμε δηλαδή τα εξής αντικείμενα:

```
( Πέτρος, 40, { ( Γιάννης, 15, { } ) } )
( Μαρία, 41, { ( Γιάννης, 15, { } ) } )
```

Είναι δυνατόν, ο Πέτρος και η Μαρία να έχουν το ίδιο παιδί, ή να πρόκειται για δύο διαφορετικά παιδιά. Η παραπάνω περιγραφή είναι ικανή να αποδώσει την πραγματικότητα, από μόνη της, και στις δύο περιπτώσεις. Στην πρώτη περίπτωση, αρκεί τα δύο αντικείμενα να μοιράζονται το κοινό στοιχείο {(Γιάννης, 15, { })}, ενώ στη δεύτερη, αρκεί να έχουμε δύο ίσα αλλά μη ταυτοτικά αντίγραφα του αντικειμένου {(Γιάννης, 15, { })}.

- *Ενημερώσεις Αντικειμένων*: αν υποθέσουμε ότι ο Πέτρος και η Μαρία έχουν το ίδιο παιδί, τότε όλες οι αλλαγές στο παιδί της Μαρίας θα πρέπει να εφαρμόζονται και στο παιδί του Πέτρου.

Είναι βασικό να επισημανθεί, ότι από άποψη υλοποίησης, η ταυτότητα ενός αντικειμένου είναι μια έννοια που έχει να κάνει κυρίως με το σύστημα, παρά με το χρήστη. Τα αναγνωριστικά που εκχωρούνται είναι συνήθως αόρατα στους χρήστες, και πάντως σίγουρα μη διαχειρίσιμα, καθώς ο χρήστης δεν μπορεί να τα αλλάξει. Για να μπορεί ο χρήστης να διαχειριστεί τα αντικείμενα, τα συστήματα εισάγουν την έννοια του *ονόματος* (name) του αντικειμένου. Ένας χρήστης, αν θέλει, μπορεί να αναθέσει ένα όνομα σε ένα αντικείμενο, και να το διαχειρίζεται μέσω αυτού. Το όνομα του αντικειμένου μπορεί να ζει μαζί με το αντικείμενο -σε κάποιες υλοποιήσεις- και σε κάποιες άλλες μπορεί να είναι μια "προσωρινή μεταβλητή" -πράγμα που σημαίνει ότι μπορούμε ακόμα και να μεταφέρουμε ένα όνομα από ένα αντικείμενο σε ένα άλλο (κάτι που προφανώς δεν μπορεί να γίνει με την ταυτότητα σε καμία περίπτωση).

Ενθυλάκωση (encapsulation)

Η ιδέα της ενθυλάκωσης προκύπτει από την ανάγκη διάκρισης μεταξύ καθορισμού και υλοποίησης μιας εφαρμογής καθώς και από την ανάγκη για δομή κατά ενότητες (modularity).

Η έννοια της ενθυλάκωσης στις γλώσσες προγραμματισμού προκύπτει από τους *αφηρημένους τύπους δεδομένων* (abstract data types). Κάθε αντικείμενο έχει ένα *τμήμα διαπροσωπείας* (interface part) και ένα *τμήμα υλοποίησης* (implementation part). Το τμήμα διαπροσωπείας είναι ο ορισμός των λειτουργιών που μπορεί να επιτελέσει το αντικείμενο. Το τμήμα υλοποίησης έχει ένα *τμήμα δεδομένων* (data part) και ένα *συναρτησιακό τμήμα* (procedural part). Το τμήμα δεδομένων παριστάνει την κατάσταση (state) των δεδομένων και το συναρτησιακό τμήμα την υλοποίηση των διαφόρων λειτουργιών. Δηλαδή,

```
αντικείμενο = τμήμα διαπροσωπείας + τμήμα υλοποίησης
```

όπου

```
τμήμα υλοποίησης = (τμήμα δεδομένων + συναρτησιακό τμήμα)
```

Η μεταφορά της έννοιας της ενθυλάκωσης στις Βάσεις Δεδομένων καθορίζει ότι ένα αντικείμενο ενσωματώνει πρόγραμμα και δεδομένα. Στις σχεσιακές βάσεις, κάθε οντότητα παριστάνεται με μια πλειάδα (tuple), ενώ οι εφαρμογές τρέχουν χρησιμοποιώντας κάποια προστακτική γλώσσα με ενσωματωμένες DML εντολές (ή μια γλώσσα τέταρτης γενεάς). Οι εφαρμογές είναι αποθηκευμένες σε ένα σύστημα αρχείων (file system), ανεξάρτητο της βάσης.

Σ' ένα αντικειμενοστρεφές σύστημα κάθε οντότητα καθορίζεται σαν ένα τμήμα δεδομένων και ένα τμήμα υλοποίησης, όπου υλοποιούνται οι διάφορες εφαρμογές που έχουν σχέση με την οντότητα.

Ένα καλό παράδειγμα, για τη διαφορά ανάμεσα στο σχεσιακό και το αντικειμενοστρεφές μοντέλο, είναι η οντότητα "Εργοστάσιο_Αυτοκινήτων" και η υλοποίηση της εφαρμογής "Αύξησε_Παραγωγή_Φορτηγών". Σε μια σχεσιακή βάση, η εφαρμογή "Αύξησε_Παραγωγή_Φορτηγών" θα ήταν ένα εξωτερικό πρόγραμμα. Αντίθετα, σε μια αντικειμενοστρεφή βάση, η οντότητα "Εργοστάσιο_Αυτοκινήτων" θα αποτελούνταν από το τμήμα διαπροσωπείας (το μέσο επικοινωνίας με το χρήστη), το τμήμα δεδομένων (το οποίο θα έμοιαζε αρκετά με την υλοποίηση της οντότητας στο σχεσιακό μοντέλο) και το συναρτησιακό τμήμα, που ανάμεσα στα άλλα, θα είχε και την εφαρμογή "Αύξησε_Παραγωγή_Φορτηγών".

Καμιά λειτουργία, πέραν αυτών που καθορίστηκαν στο τμήμα διαπροσωπείας δεν μπορεί να επιτελεστεί. Η παραπάνω θέση αποτελεί ένα από τα πιο σημαντικά στοιχεία για την έννοια της ενθυλάκωσης. Ο μόνος τρόπος για να αποταθούμε στα διάφορα αντικείμενα, και στις instance variables τους, είναι μέσω των μηνυμάτων που δηλώθηκαν στο τμήμα διαπροσωπείας.

Αν υποθέσουμε ότι έχουμε μια κλάση "Άνθρωπος", με μια μέθοδο "Παρουσιάσου", και ότι για το αντικείμενο "Γιώργος", που τυχαίνει να είναι στιγμιότυπο της κλάσεως "Άνθρωπος", υπάρχει κάπου η κλήση:

```
Γιώργος Παρουσιάσου
```

Όταν το αντικείμενο "Γιώργος" λάβει το παραπάνω μήνυμα αναλαμβάνει την εκτέλεση της μεθόδου "Παρουσιάσου".

Η ενθυλάκωση ενθαρρύνει ιδιαίτερα την ιδέα της "λογικής ανεξαρτησίας δεδομένων": μπορούμε να αλλάξουμε την υλοποίηση ενός τύπου, χωρίς να φανεί αυτό, στη διαπροσωπεία. Έτσι, οι διάφορες εφαρμογές προστατεύονται από τις αλλαγές στα χαμηλότερα επίπεδα υλοποίησης.

Συμπερασματικά, η ενθυλάκωση συγκεντρώνει τα κάτωθι χαρακτηριστικά:

- ενσωματώνει δεδομένα και λειτουργίες σ' αυτά,
- αποκρύπτει τον τρόπο υλοποίησης των μεθόδων, οπότε απεξαρτά τη διαχείριση του αντικειμένου από τη δομή του,
- επιβάλει τη διαχείριση του κάθε αντικειμένου, αυστηρά και μόνο από τις δικές του μεθόδους, μέσω των αντίστοιχων μηνυμάτων.

Τύποι και κλάσεις (Types/Classes)

Τύποι. Η ιδέα του τύπου συνίσταται στην σύνοψη μιας ομάδας τιμών. Χρησιμοποιούμε τους τύπους για να εκφράσουμε ομάδες από αντικείμενα με *δομικές* ομοιότητες. Είναι πολύ βασικό να σημειωθεί ότι ο τύπος αναφέρεται αποκλειστικά και μόνο στη δομή των αντικειμένων.

Ο τύπος αντιστοιχεί στην ιδέα του *αφηρημένου τύπου δεδομένων* (abstract data type - ADT). Όταν λέμε αφηρημένος τύπος δεδομένων, εννοούμε ότι ο εν λόγω τύπος έχει οριστεί από το χρήστη, αφενός σε ότι αφορά τη δομή του, αλλά και σε ότι αφορά τις εξειδικευμένες λειτουργίες που μπορεί ο τύπος να εκτελέσει.

Ένας τύπος αποτελείται από δύο μέρη:

- το τμήμα διαπροσωπείας
- το τμήμα υλοποίησης.

Το *τμήμα διαπροσωπείας* είναι και το μόνο ορατό στους χρήστες, και αποτελείται από μια λίστα από λειτουργίες, και την *υπογραφή* τους (signature). Η υπογραφή αποτελείται από το όνομα της λειτουργίας, τη λίστα των παραμέτρων εισόδου (input parameters), και τον τύπο του αποτελέσματος. Πρόκειται για τα μηνύματα που ο προγραμματιστής δικαιούται να χρησιμοποιεί, προκειμένου να διαχειρίζεται τα αντικείμενα του συγκεκριμένου τύπου.

Το *τμήμα υλοποίησης* αποτελείται από το *τμήμα δεδομένων* (data part) όπου περιγράφεται η εσωτερική δομή του αντικειμένου, και το *λειτουργικό τμήμα* (operation part), όπου υπάρχουν οι μέθοδοι που υλοποιούν τις λειτουργίες του τμήματος διαπροσωπείας.

Με τη χρήση των τύπων, έχουμε ισχυρό έλεγχο ορθότητας των προγραμμάτων. Η ορθότητα του προγράμματος ελέγχεται κυρίως στον χρόνο μετάφρασης. Γενικά, στα συστήματα που βασίζονται στη χρήση τύπων, οι τύποι δεν είναι "πολίτες α' κατηγορίας" (first class citizens), έχουν ειδικό καθεστώς, και δε γίνεται να τροποποιηθούν στο χρόνο εκτέλεσης (run-time).

Κλάσεις. Η ιδέα της κλάσης έχει να κάνει με αντικείμενα που παρουσιάζουν τις ίδιες *σημασιολογικές* ιδιότητες. Το σύνολο των αντικειμένων που ανήκουν σε μια κλάση, το ονομάζουμε *έκταση* (extension) της κλάσης.

Οι κλάσεις δε χρησιμοποιούνται τόσο για τον έλεγχο του προγράμματος, όσο για την κατασκευή και διαχείριση νέων αντικειμένων. Οι κλάσεις είναι "πολίτες α' κατηγορίας", και μπορεί κανείς να τις διαχειριστεί κατά το χρόνο εκτέλεσης (π.χ. μπορεί μια κλάση να περαστεί σαν παράμετρος).

Αξίζει, επίσης, να σημειωθεί ότι ο ρόλος των κλάσεων είναι να αντικαταστήσουν το παραδοσιακό σχήμα της Βάσης Δεδομένων, με μια ιεραρχία από κλάσεις .

Κάθε κλάση συνήθως σχετίζεται άμεσα με ένα τύπο. Είναι λογικό να θεωρήσουμε ότι "..αφού τα αντικείμενα που ανήκουν σε μια κλάση έχουν σημασιολογική ομοιότητα, είναι λογικό να αναμένει κανείς ότι θα έχουν και δομική ομοιότητα..." [Ki91]. Αυτό, βέβαια, δε σημαίνει ότι θα ήταν λάθος να έχουμε αντικείμενα που έχουν διαφορετική δομή μέσα στην ίδια κλάση, αρκεί βέβαια να το επιτρέπει η σημασιολογία της εφαρμογής.

Η βασική διαφορά ανάμεσα σε τύπους και κλάσεις συνίσταται λοιπόν, στο γεγονός ότι οι μεν τύποι υπάρχουν λόγω δομικής ομοιότητας κάποιων αντικειμένων, ενώ οι κλάσεις λόγω της σημασιολογικής ομοιότητάς τους.

Το γεγονός αυτό έχει επίδραση και πάνω στο τι αποτελεί τον πληθυσμό των τύπων και των κλάσεων. Οι τύποι έχουν ως στιγμιότυπα *τιμές*, ενώ οι κλάσεις έχουν ως στιγμιότυπα *αντικείμενα*.

Η διαφορά τιμών και αντικειμένων είναι πολύ σημαντική. Οι τιμές είναι απλές, *βασικές* (simple ή base values) δομές δεδομένων, που βασίζονται σε *βασικούς* τύπους (base types), τους οποίους παρέχει το σύστημα (π.χ. string, integer, κ.λ.π.) ή *σύνθετες* (composite). Οι τιμές, σκοπό έχουν να αναπαραστήσουν τη δομή των αντικειμένων. Οι σύνθετες τιμές κατασκευάζονται από το χρήστη για να αναπαραστήσουν αντικείμενα σύνθετης δομής. Οι τιμές είναι αναλλοίωτες. Αυτό σημαίνει, πως οποιαδήποτε μεταβολή σε μια τιμή θα έχει ως αποτέλεσμα την παραγωγή μιας νέας τιμής. Για να μεταβάλλουμε, δε, την τιμή που κατασκευάστηκε με βάση ένα δεδομένο τύπο, θα πρέπει να χρησιμοποιήσουμε απλώς, τους αντίστοιχους τελεστές του τύπου αυτού.

Τα αντικείμενα από την άλλη πλευρά, υπάρχουν για να αναπαραστήσουν οντότητες του πραγματικού κόσμου. Χαρακτηρίζονται μονοσήμαντα από μια ταυτότητα αντικειμένου, η δομή τους βασίζεται σε ένα τύπο και παρουσιάζουν συμπεριφορά. Μια αλλαγή σε ένα αντικείμενο είναι αλλαγή στην τιμή του, που σημαίνει ότι το αντικείμενο παραμένει το ίδιο ως σημασιολογική οντότητα (διατηρεί τον ταυτότητά του), αλλά η τιμή του αλλάζει. Για να μεταβάλλουμε την τιμή ενός αντικειμένου, πρέπει να χρησιμοποιήσουμε τα μηνύματα που μας προσφέρει η διαπροσωπεία του. Η υλοποίηση της μεταβολής γίνεται από τις ενθυλακωμένες μεθόδους, οι οποίες χρησιμοποιούν τους τελεστές του τύπου του αντικειμένου για να μεταβάλλουν την τιμή του. Κάτι αντίστοιχο συμβαίνει και όταν προσπαθούμε να προσπελάσουμε απλώς, την τιμή του αντικειμένου.

Τιμή (value) και κατάσταση (state) αντικειμένου

Με βάση τα παραπάνω, είναι τώρα ξεκάθαρο, ότι η *κατάσταση* του αντικειμένου είναι ο τύπος βάσει του οποίου δομήθηκε το αντικείμενο. Δεδομένου δε, ότι, εν γένει, τα αντικείμενα μιας κλάσης είναι ομοειδή σε ότι αφορά τη δομή τους, ο τύπος που δομεί τα αντικείμενα μιας κλάσης είναι ο ίδιος με τον "τύπο της κατάστασης των στιγμιοτύπων", στον ορισμό της κλάσης που δόθηκε παραπάνω.

Η *τιμή* ενός αντικειμένου, είναι ένα διάνυσμα τιμών. Κάθε στοιχείο του διανύσματος παρουσιάζει και την τιμή μιας από τις instance variables του αντικειμένου. Είναι σαφές ότι ένα στοιχείο του διανύσματος μπορεί να είναι μια σύνθετη τιμή. Επιπλέον, η σύνθετη αυτή τιμή μπορεί να αναπαριστά την τιμή ενός δεδομένου και υπαρκτού αντικειμένου.

Ακόμα και οι κλάσεις μπορούν να θεωρηθούν αντικείμενα. Έτσι, είναι απολύτως φυσικό να έχουν και αυτές κατάσταση και τιμή. Στον ορισμό μιας κλάσης, οι έννοιες "τύπος της τιμής των στιγμιοτύπων" και "τύπος της κατάστασης των στιγμιοτύπων" αναφέρονται στις αντίστοιχες έννοιες σε ότι αφορά τα στιγμιότυπα μιας κλάσης. Οι έννοιες "τύπος της τιμής της κλάσης" και "τύπος της κατάστασης της κλάσης" αναφέρονται στην κλάση αυτή καθεαυτή, αντιμετωπιζόμενη ως αντικείμενο.

Μια πολύ λεπτή διάκριση γίνεται ανάμεσα στον "τύπο της τιμής των στιγμιοτύπων" και τον "τύπο της κατάστασης των στιγμιοτύπων". Θα έλεγε κανείς ότι θα πρέπει να συμπίπτουν και άρα δεν έχουν λόγο ύπαρξης και οι δύο μαζί. Ενίοτε όμως, η τιμή ενός αντικειμένου δεν βασίζεται αυστηρά στην κατάστασή του, αλλά σε ένα υπερσύνολό της, το οποίο συμπληρώνεται από το συνδυασμένο αποτέλεσμα κάποιων μεθόδων. Για παράδειγμα, έστω η κλάση "Άνθρωπος" με διάφορα instance variables, ένα εκ των οποίων είναι και το "Όνομα", το οποίο εκφράζεται σαν ένα string. Μπορεί η διαπροσωπεία του αντικειμένου να καθορίζει ότι στο διάνυσμα της τιμής του αντικειμένου πρέπει να υπάρχει και μια τιμή που να λέγεται "Επώνυμο". Η τιμή μπορεί να μην υπάρχει δηλωμένη στην κατάσταση του αντικειμένου (στον τύπο, δηλαδή, που το υλοποιεί), αλλά να προκύπτει με βάση το αποτέλεσμα μιας μεθόδου (η οποία, για παράδειγμα, επεξεργάζεται το string με το όνομα του αντικειμένου).

Σύνθετοι τύποι και τιμές

Είναι σαφές μέχρι τώρα, ότι σε ένα αντικειμενοστρεφές μοντέλο δεδομένων θα πρέπει να υπάρχουν βασικοί τύποι, παρεχόμενοι από το σύστημα, και σύνθετοι, κατασκευαζόμενοι από το χρήστη. Οι τιμές που προκύπτουν με βάση αυτούς τους τύπους είναι και αυτές με τη σειρά τους, βασικές ή σύνθετες. Για να κατασκευάσουμε μια σύνθετη τιμή χρειαζόμαστε *κατασκευαστές* (constructors), τους οποίους το μοντέλο δεδομένων θα πρέπει να παρέχει. Υπάρχουν πολλά είδη κατασκευαστών:

- *Σύνολα* (sets), τα οποία είναι βασικά, γιατί είναι ένας φυσικός τρόπος να αναπαριστώνται ομάδες του πραγματικού κόσμου.
- *Πλειάδες* (tuples), των οποίων η σημασία έγκειται στο ότι, είναι ο πιο φυσικός τρόπος να αναπαραστήσουμε τις ιδιότητες μιας οντότητας.
- *Λίστες* (lists) ή *πίνακες* (arrays), τα οποία είναι βασικά, στο να αναπαριστούν την διάταξη που υπάρχει στη φυσική πραγματικότητα.
- *Πολυσύνολα* (bags), που είναι σύνολα που επιτρέπουν τις διπλοεγγραφές.

Οι τύποι αντικειμένων πρέπει να είναι ορθογώνιοι: ο οποιοσδήποτε κατασκευαστής θα πρέπει να μπορεί να εφαρμοστεί σε οποιαδήποτε τιμή ή ακόμα και στον εαυτό του. Έτσι,

μπορεί να προκύπτουν τιμές ιδιαίτερης πολυπλοκότητας. Για παράδειγμα μπορεί να έχουμε τιμές του τύπου:

```
tuple [ set { string }, set { integer }, bag{ string } ]
```

(Ας σημειωθεί ότι στο σχεσιακό μοντέλο οι κατασκευαστές δεν είναι ορθογώνιοι - ο κατασκευαστής συνόλων -για παράδειγμα- μπορεί να εφαρμοστεί μόνο σε πλειάδες).

Σύνθετα αντικείμενα και δομές αντικειμένων

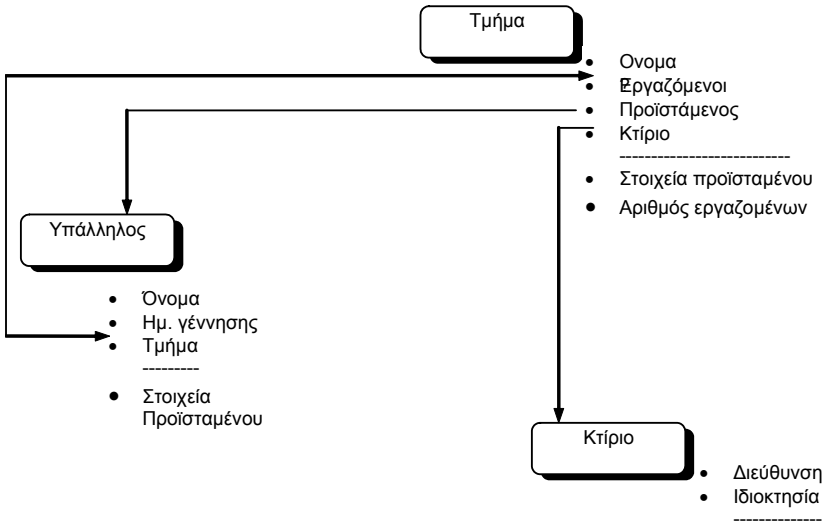
Είναι σαφές μέχρι τώρα, πώς με βάση τις τιμές και τις καταστάσεις κατασκευάζουμε τα αντικείμενα. Τα αντικείμενα μπορεί να είναι "δομές αντικειμένων" (object structures) ή "σύνθετα αντικείμενα" (complex objects).

Για να κατασκευάσουμε *δομές αντικειμένων* που ανταποκρίνονται στις φυσικές οντότητες του πραγματικού κόσμου, και εκφράζουν την πολυπλοκότητά τους, χρειαζόμαστε τη δυνατότητα να έχουμε ένα είδος αναφοράς από ένα δοθέν αντικείμενο σε ένα άλλο (π.χ. μέσω της ταυτότητας ή του ονόματος). Η *αναφορά* (reference) υλοποιείται σαν μια επιπρόσθετη βασική τιμή, η οποία να μπορεί να υπάρξει ως μέρος της τιμής ολόκληρου του δοθέντος αντικειμένου. Μπορούμε έτσι, να υλοποιούμε προγραμματιστικά συσχετίσεις αντικειμένων από διαφορετικές ή ίδιες κλάσεις. Εννοείται ότι η αναφορά μπορεί να γίνεται ακόμα και σε ένα αντικείμενο τύπου set, bag, κ.λ.π. το οποίο με τη σειρά του αναφέρεται σε άλλα αντικείμενα.

Είναι σημαντικό να σημειώσουμε ότι η αναφορά μπορεί να είναι και κατά τις δύο φορές. Οι αναφορές αυτές ονομάζονται *αντίστροφοι δείκτες* (inverses). Στην περίπτωση των αντίστροφων δεικτών έχουμε ένα είδος περιορισμού ακεραιότητας. Όταν αλλάξει η τιμή στην μία πλευρά του δείκτη, τότε αυτόματα ενημερώνεται και η τιμή της άλλης πλευράς.

Στο Σχήμα 4.2 έχουμε ένα τμήμα το οποίο έχει κάποιους εργαζόμενους, κάποιον προϊστάμενο και ένα κτίριο στο οποίο στεγάζεται. Οι εργαζόμενοι στο τμήμα είναι ένα σύνολο υπαλλήλων. Η μοντελοποίησή τους γίνεται έχοντας μια αναφορά (συμβολιζόμενη με ένα απλό βέλος) από το κάθε τμήμα σε ένα σύνολο που περιέχει αντικείμενα της κλάσης υπάλληλος. Η εν λόγω αναφορά έχει και την αντίστροφή της: για κάθε υπάλληλο έχουμε και μια αναφορά στο τμήμα στο οποίο εργάζεται. Αν αποφασίσουμε κάποια στιγμή ότι ο υπάλληλος Χ αλλάζει τμήμα, τότε αυτόματα και το παλιό του τμήμα σταματά να τον έχει στο σύνολο των υπαλλήλων του. Επίσης, παρατηρούμε ότι μπορούμε να μεταχειριστούμε τις αναφορές ισοδύναμα με τις instance variables. Αυτό σημαίνει, π.χ. ότι μπορούμε να έχουμε μεθόδους που να τις χρησιμοποιούν.

Πολλές φορές η συσχέτιση αυτή δύο ή περισσοτέρων αντικειμένων αναπαραριστά μια *σχέση σύνθεσης* ("part_of"). Βάσει αυτού του γεγονότος, είναι θεμιτό να έχουμε την ανάγκη διαχείρισης του συνόλου της σύνθεσης των αντικειμένων (π.χ. την ανάγκη της διαγραφής και του αντικειμένου και των αντικειμένων που το "συνθέτουν"). Έτσι, χρειαζόμαστε να θεωρήσουμε το *σύνθετο αντικείμενο* διαφορετικά. Στην τιμή του πλέον, δεν έχουμε αναφορές στα άλλα αντικείμενα που το συνθέτουν, αλλά απ' ευθείας περικλείουμε τα αντικείμενα που το συνθέτουν.



Σχήμα 4.2 Αναφορές μεταξύ κλάσεων

Ιεραρχία τύπων/ κλάσεων (Class or Type Hierarchies)

Η ιεραρχία τύπων/κλάσεων σχετίζεται άμεσα με την έννοια της *κληρονομικότητας* (inheritance). Η κληρονομικότητα βασίζεται στην ιδέα του να μπορούμε να δώσουμε εξειδικευμένα χαρακτηριστικά σε κάποια αντικείμενα, χωρίς αυτά να χάνουν τις γενικότερες ιδιότητες που θα πρέπει να τα διακρίνουν.

Για να το επιτύχουμε αυτό κατασκευάζουμε ένα δέντρο κλάσεων. Κάθε κλάση που είναι κόμβος του δέντρου "κληρονομεί", από την πατρική της κλάση όλα τα δομικά και λειτουργικά χαρακτηριστικά της -τις instance variables και τις μεθόδους, δηλαδή. Στα χαρακτηριστικά αυτά, κάθε κλάση νομιμοποιείται να προσθέσει τα δικά της χαρακτηριστικά και λειτουργίες. Με τον τρόπο αυτό η θυγατρική κλάση (ή *υποκλάση*) αποτελεί μια εξειδίκευση της πατρικής κλάσης (ή *υπερκλάσης*), (ή αντίστροφα, η πατρική κλάση αποτελεί μια γενίκευση των θυγατρικών της κλάσεων).

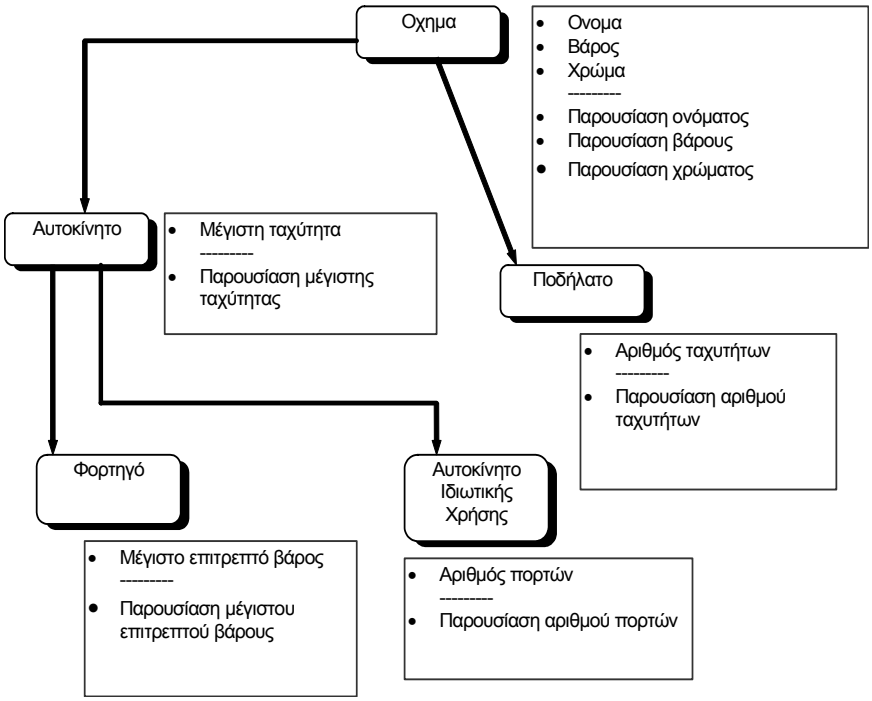
Είναι δυνατόν, μια κλάση να κληρονομεί χαρακτηριστικά και λειτουργίες από περισσότερες της μίας κλάσεις. Τότε έχουμε το φαινόμενο της *πολλαπλής κληρονομικότητας* (multiple inheritance).

Η κληρονομικότητα βοηθά στην καλύτερη αναπαράσταση του φυσικού κόσμου, εκεί όπου υπάρχουν πραγματικά ιεραρχίες, με αποτέλεσμα να έχουμε μια πιο ακριβή και καλύτερα δομημένη περιγραφή του σχήματος. Σε κάθε περίπτωση πάντως, βοηθά και στην εξοικονόμηση και επαναχρησιμοποίηση κώδικα (code reusability).

Στο παράδειγμα του Σχήματος 4.3, έχουμε ορίσει την κλάση "Όχημα" με το όνομα, το βάρος του και το χρώμα του σαν instance variables και τις μεθόδους "Παρουσίαση ονόματος", "Παρουσίαση βάρους" και "Παρουσίαση χρώματος". Όλες οι θυγατρικές κλάσεις, όπως "Ποδήλατο", "Αυτοκίνητο", "Φορτηγό", "Αυτοκίνητο Ιδιωτικής Χρήσης" κληρονομούν τα παραπάνω χαρακτηριστικά και λειτουργίες. Επιπλέον η κάθε μία από αυτές τις κλάσεις, ορίζει και τα δικά της ιδιαίτερα χαρακτηριστικά και λειτουργίες.

Στο εν λόγω παράδειγμα, η κλάση "Όχημα" είναι πολύ πιθανό να κατασκευάστηκε για λόγους γενίκευσης. Αυτό σημαίνει ότι είναι πιθανό, η κλάση αυτή να μην έχει στιγμιότυπα στην πράξη, αλλά να κατασκευάστηκε για να αποτελέσει μια γενίκευση από την οποία κληρονομούν ιδιότητες (χαρακτηριστικά και λειτουργίες) οι υπόλοιπες κλάσεις. Είναι επίσης

πιθανόν, κάποιες ιδιότητες να μην είναι σαφώς υλοποιημένες, αλλά απλά να δηλώνονται. Σε αυτή την περίπτωση, έχουμε να κάνουμε με μια *αφηρημένη κλάση (abstract class)*. Ας σημειωθεί ότι οι κλάσεις αυτού του τύπου δεν είναι, εν γένει, "τεχνητές". Αυτό σημαίνει ότι μοντελοποιούν οντότητες που υπάρχουν και στην πραγματικότητα (π.χ. οχήματα υπάρχουν παντού στην καθημερινή μας ζωή). Το γεγονός ότι δεν έχουν στιγμιότυπα, ερμηνεύεται απλά, από το γεγονός ότι κάποιες ιδιότητές τους δεν είναι σαφώς υλοποιημένες. Επιπλέον, όμως, τα στιγμιότυπά τους εξειδικεύονται έτσι, ώστε να αποτελούν στοιχεία των θυγατρικών τους κλάσεων. Η χρήση τους θα αναδειχτεί ακόμα περισσότερο στην ενότητα "Πολυμορφισμός (polymorphism), υπέρβαση (overriding), υπερφόρτωση (overloading) και late binding".



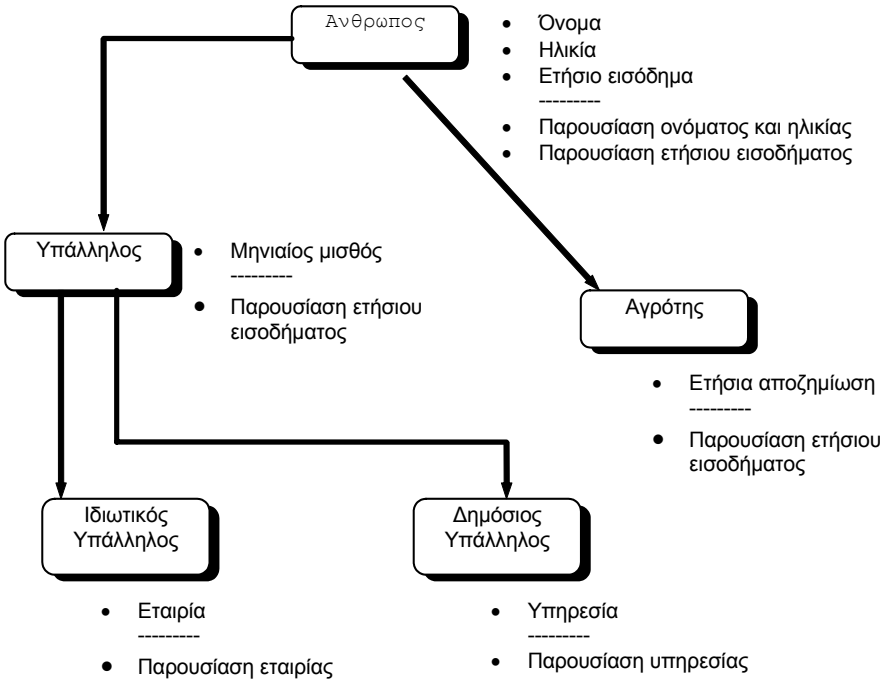
Σχήμα 4.3 Ιεραρχία κλάσεων

Πολυμορφισμός (polymorphism), υπέρβαση (overriding), υπερφόρτωση (overloading) και late binding

Πολλές φορές υπάρχει η ανάγκη να χρησιμοποιήσουμε το ίδιο όνομα για διαφορετικές λειτουργίες. Ας υποθέσουμε το παράδειγμα του Σχήματος 4.4. Στο παράδειγμα αυτό, έχουμε ορίσει την κλάση "Άνθρωπος" με το όνομα, την ηλικία του και το ετήσιο εισόδημά του σαν instance variables και τις λειτουργίες "Παρουσίασε όνομα" και "Παρουσίασε ηλικία" και "Παρουσίαση ετήσιου εισοδήματος". Όλες οι θυγατρικές κλάσεις, όπως "Υπάλληλος", "Αγρότης", "Δημόσιος Υπάλληλος", "Ιδιωτικός Υπάλληλος" κληρονομούν τα παραπάνω χαρακτηριστικά και λειτουργίες. Επιπλέον, η κάθε μία από αυτές τις κλάσεις, ορίζει και τα δικά της ιδιαίτερα χαρακτηριστικά και λειτουργίες.

Έστω ότι υλοποιούμε το ετήσιο εισόδημα στην κλάση "Άνθρωπος" σαν ένα ακέραιο αριθμό. Επιπλέον, έστω ότι στην εν λόγω κλάση η υλοποίηση της μεθόδου "Παρουσίαση ετήσιου εισοδήματος" έγκειται στην επιστροφή της τιμής της instance variable "Ετήσιο εισόδημα". Είναι δυνατόν, στην κλάση "Υπάλληλος", η οποία κληρονομεί τα χαρακτηριστικά και τις μεθόδους της κλάσης "Άνθρωπος", να ορίσουμε εκ νέου τη μέθοδο "Παρουσίαση

ετήσιου εισοδήματος" έτσι ώστε να επιστρέφει το αποτέλεσμα του μηνιαίου μισθού του υπαλλήλου επί 12. Επιπλέον, αν θέλουμε, μπορούμε να ορίσουμε ότι η εν λόγω μέθοδος, όταν υλοποιείται στην κλάση "Αγρότης", επιστρέφει το άθροισμα του ετήσιου εισοδήματος (που είναι χαρακτηριστικό που κληρονομείται από την πατρική κλάση "Άνθρωπος") με την αποζημίωση.



Σχήμα 4.4 Υπερφόρτωση μηνυμάτων

Υπερφόρτωση (overloading) ονομάζεται το φαινόμενο δύο μηνύματα σε διαφορετικές κλάσεις, να συμφωνούν στη διαπροσωπεία τους. Στην περίπτωσή μας, ακόμα και αν δεν ορίζαμε ξανά τη μέθοδο "Παρουσίαση ετήσιου εισοδήματος" θα μπορούσαμε να έχουμε το φαινόμενο της υπερφόρτωσης, λόγω κληρονομικότητας (αν υποθέσουμε ότι η κληρονομικότητα υλοποιείται αντιγράφοντας λειτουργίες και χαρακτηριστικά, από την πατρική κλάση στις θυγατρικές της).

Υπέρβαση (overriding) ονομάζουμε το γεγονός ότι κάποιο μήνυμα υλοποιείται με διαφορετική μέθοδο στον κορυφή της ιεραρχίας, απ' ότι σε κάποιο χαμηλότερο επίπεδο.

Αξίζει να σημειωθεί ότι ένα σύστημα για να μπορέσει να υποστηρίξει την υπέρβαση, δε συνδέει τα ονόματα των λειτουργιών με τα αντίστοιχα προγράμματα στο χρόνο μετάφρασης, αλλά στο χρόνο εκτέλεσης. Έτσι, τα ονόματα των λειτουργιών "επιλύονται" (resolved) - μεταφράζονται σε διευθύνσεις- στο χρόνο εκτέλεσης. Αυτή η καθυστερημένη μετάφραση, ονομάζεται *late binding*.

Έτσι, αν για παράδειγμα, ζητήσουμε για κάθε άνθρωπο την παρουσίαση του ετήσιου εισοδήματός του, μέσω του μηνύματος "Παρουσίαση ετήσιου εισοδήματος", τότε το σύστημα θα πρέπει να αποφασίσει στο χρόνο εκτέλεσης, ποια από τις τρεις υλοποιήσεις του εν λόγω μηνύματος θα χρησιμοποιήσει, για κάθε αντικείμενο. Είναι σαφές, ότι το κριτήριο επιλογής αποτελεί η κλάση στην οποία ανήκει το κάθε αντικείμενο.

Τελευταία συναφής έννοια είναι αυτή του *πολυμορφισμού* (polymorphism). Πολυμορφισμός είναι το γεγονός ότι διαφορετικά αντικείμενα (ή κλάσεις) αντιδρούν με διαφορετικό τρόπο στο ίδιο μήνυμα.

Τέλος, να σημειωθεί ότι σύμφωνα με την κλασική αντιμετώπιση, θα έπρεπε να βρούμε άλλα ονόματα, για τα ομώνυμα μηνύματα, ή εναλλακτικά να υλοποιήσουμε μία και μόνο συνάρτηση με μια εντολή case στον κώδικά της, όπου θα έπρεπε να γίνεται έλεγχος τύπων για το ποια συνάρτηση θα πρέπει να εκτελεστεί κάθε φορά. Το πλεονέκτημα που κερδίζουμε με την υπερφόρτωση, είναι η ευκολία στον προγραμματισμό εφαρμογών, καθώς μπορούμε να σχεδιάζουμε τις εφαρμογές εύκολα και να τις ανανεώνουμε ακόμα ευκολότερα.

4.2.2 Λειτουργίες στα Δεδομένα

Η άλγεβρα για αντικειμενοστρεφείς βάσεις δεδομένων δεν έχει αποσαφηνιστεί πλήρως. Παρ' όλα αυτά, μάλλον έχει υπάρξει ένα ελάχιστο πεδίο συμφωνίας γύρω από τους αλγεβρικούς τελεστές, οι οποίοι είναι απαραίτητοι σ' ένα μοντέλο σύνθετων αντικειμένων (ανεξάρτητα από το ποιο από τα προαναφερθέντα μοντέλα θα είναι αυτό). Είναι σαφές, ότι η υλοποίηση των τελεστών βασίζεται σημαντικά στα δομικά χαρακτηριστικά του μοντέλου. Έτσι, ένας τελεστής με ίδια λειτουργικότητα, μπορεί να υλοποιείται με διαφορετικό τρόπο σε διαφορετικά μοντέλα.

Λειτουργίες για την ταυτότητα αντικειμένου

Ένα μοντέλο δεδομένων, -με την ίδια λογική που οι αντικειμενοστρεφείς γλώσσες το κάνουν- οφείλει να εφοδιάζει τον προγραμματιστή με τελεστές για ελέγχους σύγκρισης και ταύτισης της ταυτότητας των αντικειμένων. Δύο αντικείμενα μπορεί να έχουν μία από τις παρακάτω σχέσεις μεταξύ τους:

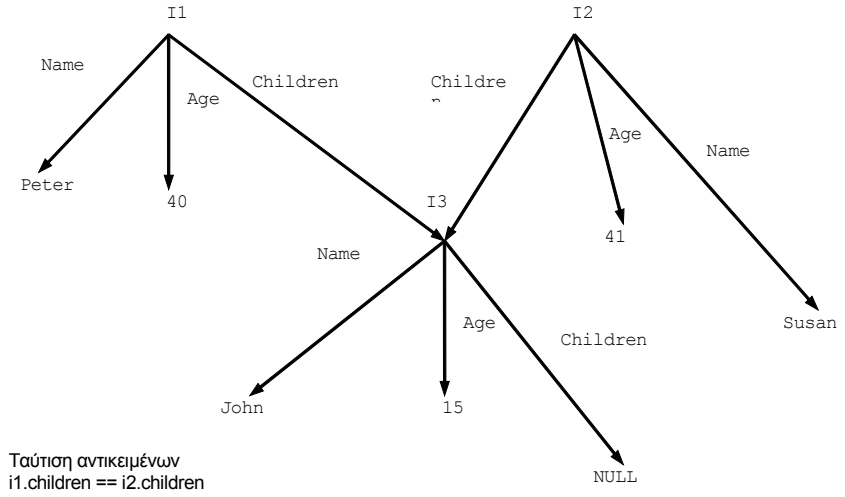
- *Ταύτιση* (identical objects): Ο τελεστής ταύτισης (X==Y) ελέγχει αν δύο αντικείμενα X και Y έχουν την ίδια ταυτότητα αντικειμένου (οπότε και ταυτίζονται μεταξύ τους) (βλ. Σχήμα 4.5)
- *Ρηχή Ισότητα* (shallow equal objects): Δύο αντικείμενα είναι ρηχά ίσα μεταξύ τους, αν ανήκουν στην ίδια κλάση, και οι instance variables τους ταυτίζονται, με την έννοια που δώσαμε ακριβώς πριν στην ταύτιση. (Σημ.: οι instance variables είναι και αυτές αντικείμενα, στη γενική περίπτωση ενός σύνθετου αντικειμένου). (βλ. Σχήμα 4.6)
- *Βαθειά Ισότητα* (deep equal objects): η βαθειά ισότητα, στην ασθενή της εκδοχή, απαιτεί από δύο αντικείμενα:
    - να ανήκουν στην ίδια κλάση
    - οι τιμές των instance variables να είναι ίσες

  Στην ισχυρή της εκδοχή, η βαθειά ισότητα απαιτεί επιπλέον, οι γράφοι των δύο αντικειμένων να είναι ισομορφικοί. Για παράδειγμα, στο Σχήμα 4.7, ισχύει:

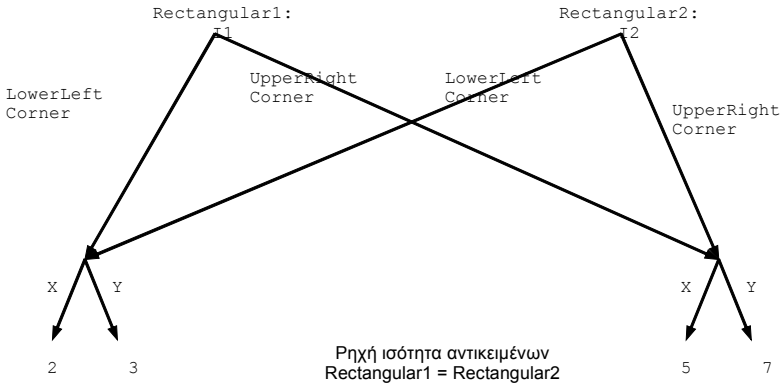
```
ARRAY1 deep-equal ARRAY2 (και στις δύο εκδοχές)
ARRAY1 deep-equal ARRAY3 (στην ασθενή εκδοχή)
ARRAY1 not deep-equal ARRAY3 (στην ισχυρή εκδοχή)
```

Επιπλέον, υπάρχουν και τελεστές που παράγουν ταυτοτικά ίσα (όπως π.χ. στην Smalltalk X:=Y), ρηχά ίσα (shallow copy) και βαθειά ίσα αντικείμενα (deep copy). Πέραν τούτου, υπάρχει και η δυνατότητα *συγχώνευσης* (merging) δύο ταυτοτικών αντικειμένων και η

δυνατότητα *αντιμετάθεσης* (swapping) των ταυτοτήτων δύο αντικειμένων (π.χ. στην Smalltalk: X become: Y).



Σχήμα 4.5 Ταύτιση αντικειμένων



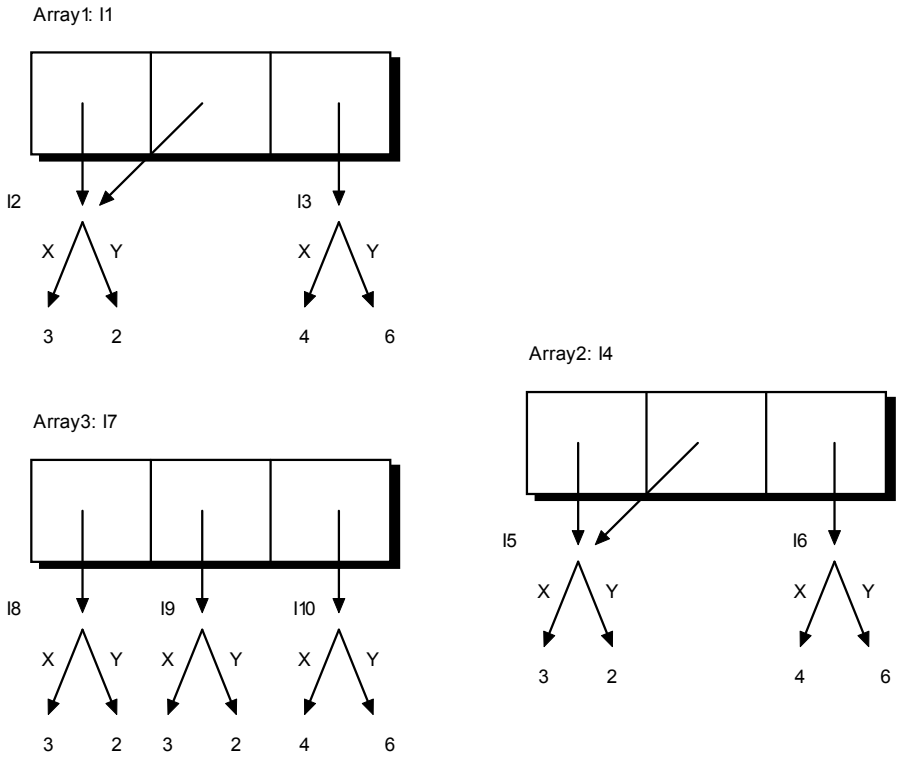
Σχήμα 4.6 Ρηχή ισότητα αντικειμένων

Όταν αναφερόμαστε σε *αθροιστικά* (aggregate) αντικείμενα, η ισότητα, όπως αυτή παρουσιάστηκε στις τρεις μορφές της, επεκτείνεται ως κάτωθι (βλ. και [BBKV87])[3]:

- *Ταύτιση* (identical objects): Παραμένει ως ισότητα αναγνωριστικών για αντικείμενα.
- *Ρηχή Ισότητα* (Shallow Equality):
    - Δύο ατομικά αντικείμενα είναι ρηχά ίσα, αν δείχνουν το ίδιο στοιχείο στο χώρο των βασικών τιμών (base values).
    - Δύο set είναι ρηχά ίσα, αν όλα τα στοιχεία τους *ταυτίζονται*.

[3] Ας μην παρανοήσει ο αναγνώστης τις οποιεσδήποτε μεταβάσεις από το ένα μοντέλο στο άλλο. Ο σκοπός αυτού του κεφαλαίου είναι να παρουσιάσει ιδέες και προτάσεις από το χώρο των OODBS, και όχι να προτείνει νέες. Οι μεταβάσεις, λοιπόν, αυτόν τον σκοπό εξυπηρετούν.

  - Δύο tuples είναι ρηχά ίσα, αν οι τιμές που παίρνουν σε κάθε πεδίο τους (attribute) *ταυτίζονται*.
- *Βαθειά Ισότητα* (Deep Equality):
  - Δύο ατομικά αντικείμενα είναι βαθειά ίσα, αν και μόνο αν, είναι ρηχά ίσα.
  - Δύο αντικείμενα τύπου set είναι βαθειά ίσα, αν κάθε στοιχείο του ενός είναι βαθειά ίσο προς ένα στοιχείο του άλλου.
  - Δύο tuples είναι ρηχά ίσα, αν οι τιμές που παίρνουν σε κάθε πεδίο είναι βαθειά ίσες.



Σχήμα 4.7 Διαφορετικές μορφές ισότητας αντικειμένων

Να υπενθυμίσουμε επίσης το γεγονός, ότι η υποστήριξη σύνθετων αντικειμένων, απαιτεί και την ταυτόχρονη υποστήριξη τελεστών, για τη διαχείριση αυτών των αντικειμένων. Οι διάφορες λειτουργίες πάνω σε ένα σύνθετο αντικείμενο, πρέπει να διαδίδονται και σε όλα τα συστατικά του. Για παράδειγμα, τα παραπάνω ισχύουν για την περίπτωση της *βαθιάς αντιγραφής* (deep copy), σε αντίθεση με την περίπτωση της *ρηχής αντιγραφής* (shallow copy), όπου τα συστατικά του νέου αντικειμένου, απλά δεικτοδοτούν κατάλληλα τα συστατικά του αρχικού (root) αντικειμένου.

Λειτουργίες για σύνολα, πολυσύνολα και λίστες

Στην ενότητα αυτή θα παρουσιαστούν λειτουργίες που αφορούν πιο σύνθετα αντικείμενα, όπως σύνολα, πολυσύνολα, λίστες κ.λ.π. (βλ. και [Kh93]). Οι λειτουργίες αυτές είναι:

- *Ένωση* (Union): για τα set S1 και S2

```
Union(S1, S2) = {s| s ∈ S1 OR s ∈ S2}
```

- *Τομή* (Intersection): για τα set S1 και S2

```
Intersection(S1, S2) = {s| s ∈ S1 AND s ∈ S2}
```

- *Διαφορά* (Difference): για τα set S1 και S2

```
Difference(S1, S2) = {s| s ∈ S1 AND NOT (s ∈ S2)}
```

Γενική Επιλογή (General Selection)

Η σημασιολογία της *Γενικής Επιλογής* βασίζεται στον τελεστή *λ(x)* και στην λογική έκφραση *e*. Ο τελεστής *λ* σημαίνει ότι εφαρμόζουμε την έκφραση *e*, για κάθε στοιχείο του διανύσματος *x*. Ισχύει:

```
GeneralSelection(S1, S2, ...,Sn,λ(x1, .., xn)e) = {e(s1, ...,
     sn) |s1 ∈ S1 AND s2 ∈ S2 AND ... AND sn ∈ Sn}
```

Για παράδειγμα, αν θέλουμε να λάβουμε όλα τα ονόματα και τους αριθμούς κυκλοφορίας για τα αυτοκίνητα με τιμή άνω των 3,000,000 δρχ:

```
GeneralSelection(Car, λ(χ)χ.price > 3,000,000 -> [Name: x.Name,
     Id: x.Id])
```

Φώλιασμα (Nest)

Το *Φώλιασμα* εξυπηρετεί την ανάγκη της ομαδοποίησης, βάσει κάποιων χαρακτηριστικών. Ισχύει:

```
Nest(S, ai) = {[a1 : s.a1, ..., ai : Sti, ..., an : s.an]|
                ∀ r, ∃ s(r ∈ Sti AND s ∈ S AND s.ai = r)}
```

Για παράδειγμα, αν έχουμε το set:

```
     Car =   {[Name: "Skarabaios", Id: "YAT 5700"],
              [Name: "Skarabaios", Id: "IB 1919"],
              [Name: "Froso", Id: "XZ 2739"]}
```

τότε:

```
Nest(Car, Name) =
         {[Name: "Skarabaios", Id: {"YAT 5700", "IB 1919"}],
          [Name: "Frosso", Id: "XZ 2739"]}
```

Απο-Φώλιασμα(UnNest)

Το *Απο-Φώλιασμα* αναιρεί το αποτέλεσμα που έχει η επίδραση ενός τελεστή Φωλιάσματος σε ένα set. Ισχύει:

```
UnNest(S, Ai) = {[A1 : s.A1, A2 : s.A2, ..., Ai : t, ...] |s ∈ S
     AND t ∈ s.Ai}
```

Άνοιγμα(Flatten)

Το *Άνοιγμα* είναι μια λειτουργία η οποία ενοποιεί σε ένα set, τα set των αντικειμένων που ανήκουν στο ίδιο "ευρύτερο" set. Ισχύει:

```
Flatten(S) = {t | ∃ s ∈ S AND t ∈ s}
```

Αν για παράδειγμα, έχω το set των set που αποτελούνται από τα αντικείμενα χ1, χ2, ..., χ10, και είναι:

```
S = {{χ1, χ2, χ3}, {χ4, χ5, χ6, χ7, χ8}, {χ9}, {χ10}}
```

τότε

```
Flatten(S) = {χ1, χ2, χ3, χ4, χ5, χ6, χ7, χ8, χ9, χ10}
```

4.2.3 Περιορισμοί Ακεραιότητας

Οι περιορισμοί σκοπό έχουν να εξασφαλίσουν την ακεραιότητα των δεδομένων. Στις αντικειμενοστρεφείς βάσεις δεδομένων οι περιορισμοί ακεραιότητας που συνήθως μπορούν να οριστούν είναι οι ακόλουθοι:

- *Περιορισμοί ακεραιότητας κλειδιού* (Key constraints).
- *Υπαρξιακός περιορισμός ακεραιότητας* (Existential constraint) (όταν σε κάποιο σύνθετο αντικείμενο, υπάρχει αναφορά σε κάποιο άλλο αντικείμενο, τότε το αντικείμενο στο οποίο γίνεται η αναφορά, υπάρχει).
- *Περιορισμοί ακεραιότητας μη κενού χαρακτηριστικού* (NOT NULL constraints).
- *Περιορισμός δομικής συνέπειας* (structural consistency) (το σχήμα μιας αντικειμενοστρεφούς βάσης δεδομένων είναι δομικά συνεπές αν η ιεραρχία των κλάσεων είναι ένας κατευθυνόμενος ακυκλικός γράφος (DAG), αν δεν υπάρχουν συγκρούσεις στον ορισμό χαρακτηριστικών (attributes) και μεθόδων και αν οι τύποι και οι υπογραφές των μεθόδων είναι συμβατοί. Ένα αντικείμενο, δε, είναι δομικά συνεπές, αν η τιμή του ανταποκρίνεται στον τύπο της κλάσης του).
- *Περιορισμός συνέπειας συμπεριφοράς* (Behavioral consistency) (μια βάση είναι συνεπής στη συμπεριφορά της αν κάθε μέθοδος σέβεται την υπογραφή της και αν ο κώδικάς της δεν παράγει λάθη στο χρόνο εκτέλεσης (run-time errors) ή ανεπιθύμητα αποτελέσματα).
- *Περιορισμός διατήρησης της συνέπειας του αντίστροφου δείκτη* (Inverse link consistency).
- *Περιορισμός διαχωρισιμότητας* (Disjointness Constraint) (ένα αντικείμενο δε μπορεί να ανήκει σε περισσότερες από μία κλάσεις).
- *Περιορισμός επικάλυψης* (Covering constraint) (δεν είναι δυνατό, μια κλάση να έχει στιγμιότυπα, τα οποία να μην ανήκουν σε κάποια από τις υποκλάσεις της).

4.3 ODMG-93

Η επιτυχία των σχεσιακών συστημάτων διαχείρισης βάσεων δεδομένων βασίστηκε αφενός στην απλότητα του σχεσιακού μοντέλου δεδομένων και στην ανεξαρτησία των δεδομένων από τις εφαρμογές, και αφετέρου στην τυποποίηση. Η αποδοχή του SQL standard, το οποίο βασίστηκε στο μοντέλο δεδομένων και τη γλώσσα που πρότεινε η IBM (η μεγαλύτερη εταιρεία που αναμίχθηκε από νωρίς στην ανάπτυξη τέτοιων συστημάτων), πρόσφερε μεταφερσιμότητα των δεδομένων και των εφαρμογών, γεγονός που έκανε το σχεσιακό μοντέλο δεδομένων και την SQL τόσο δημοφιλή.

Στην περίπτωση των αντικειμενοστρεφών συστημάτων δεδομένων, η κατάσταση είναι διαφορετική, καθώς η "παιδική τους ηλικία" συνοδεύτηκε από το βασικό πρόβλημα της ανυπαρξίας ενός καλά ορισμένου μοντέλου δεδομένων. Κάθε σύστημα προσέφερε το δικό του μοντέλο δεδομένων. Τα μοντέλα δεδομένων αυτά έμοιαζαν σημαντικά μεταξύ τους, όμως η

ομοιότητα δεν είναι αρκετή: στο επίπεδο του μοντέλου δεδομένων χρειάζεται ταύτιση, και όχι απλή ομοιότητα στα βασικά στοιχεία. Το ίδιο συμβαίνει και στο επίπεδο των γλωσσών προγραμματισμού και των ερωτήσεων. Κάθε σύστημα, βασιζόμενο στο δικό του μοντέλο δεδομένων, προσέφερε και τη δική του γλώσσα. Αν και οι γλώσσες προγραμματισμού της βάσης δεδομένων βασίζονταν στις ίδιες κλασικές γλώσσες τρίτης γενιάς (C++, Smalltalk, κ.λ.π.) δεν υπήρχε ταύτιση στα επιπλέον στοιχεία που χρειάζονται για τη διαχείριση των αντικειμένων μιας βάσης δεδομένων.

Με βάση τα παραπάνω, ήταν φανερή η ανάγκη για την ύπαρξη τουλάχιστον ενός κοινού παρονομαστή, που να επιτρέπει τη μεταφερσιμότητα εφαρμογών και δεδομένων. Το Object Database Management Group (ODMG) είναι μια επιτροπή που, στις αρχές της δεκαετίας του 1990, πρότεινε ένα κοινό interface (ODMG-93) στους διάφορους κατασκευαστές αντικειμενοστρεφών συστημάτων βάσεων δεδομένων, με βάση το οποίο να συγγράφονται μεταφέρσιμες εφαρμογές από το ένα σύστημα στο άλλο. Η μεταφερσιμότητα (portability) των εφαρμογών συνίσταται στη μεταφερσιμότητα του σχήματος της βάσης, του binding της γλώσσας προγραμματισμού, της γλώσσας διαχείρισης δεδομένων και της γλώσσας ερωτήσεων.

Το ODMG δεν είναι μια επίσημη επιτροπή τυποποίησης. Συμμετέχουν όμως, σ' αυτήν, εταιρείες που κατέχουν το 80% της αγοράς των αντικειμενοστρεφών συστημάτων βάσεων δεδομένων. Επιπλέον, οι εταιρείες αυτές έχουν δεσμευτεί να παρέχουν το κοινό αυτό interface από τις αρχές του 1995. Κομμάτια του standard αποτελούν στοιχεία ήδη υλοποιημένα σε κάποια από τα υπάρχοντα αντικειμενοστρεφή συστήματα βάσεων δεδομένων (π.χ. η query language βασίζεται στην query language του O_2).

Το 1993, τα επτά μέλη του ODMG με δικαίωμα ψήφου ήταν οι εταιρείες Object Design (ObjectStore), Objectivity, ONTOS, O_2 Technology, POET Software, Servio Corporation (GemStone), και Versant Technology. Επιπλέον όλες οι εταιρείες που συμμετέχουν ως Reviewer Members έχουν δεσμευτεί να υλοποιήσουν το ODMG-93. Σήμερα, στα 1997, οι εταιρείες αυτές έχουν παραμείνει σχεδόν οι ίδιες (αν και πολλές έχουν αλλάξει τα ονόματά τους) και είναι οι εξής: GemStone Systems, IBEX Computing, O2 Technology, Object Design, Objectivity, POET Software, UniSQL, Versant Object Technology. Πρακτικά, αποχώρησε η Ontos και προσχώρησε η IBEX Computing (που διακινεί το Itasca) και η UniSQL. Στην παρουσίαση του ODMG-93 standard θα βασιστούμε κυρίως στο [Ca95] και κατά δεύτερο λόγο στα [Ki94], [ODMG94]. Όλα τα σχήματα και τα παραδείγματα είναι από το [Ca94] και το [Ca95].

4.3.1 Αρχιτεκτονική του ODMG-93

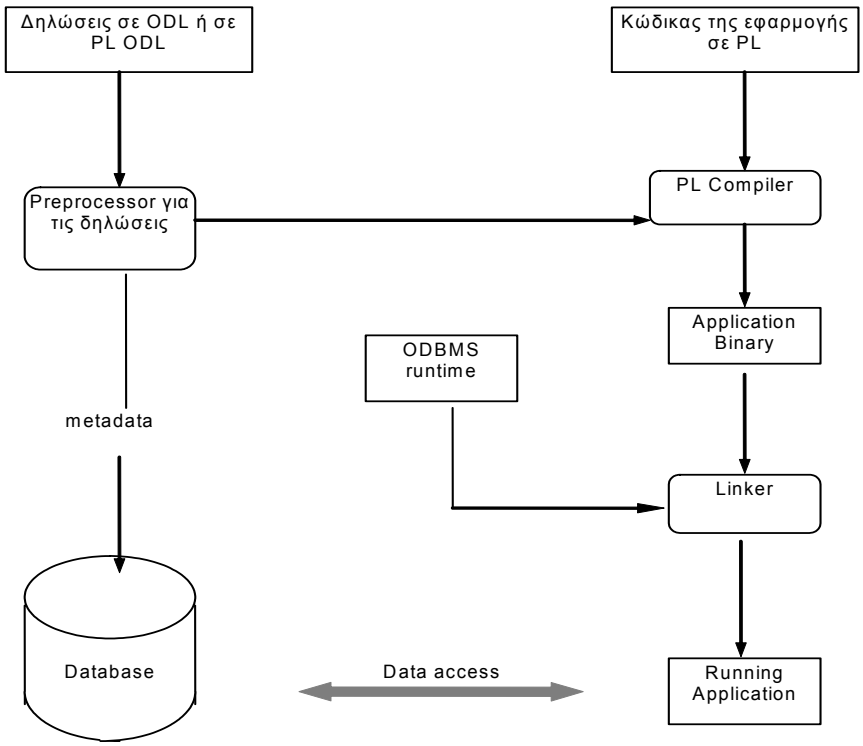
Τα βασικά στοιχεία της αρχιτεκτονικής του ODMG-93 είναι τα εξής:

- *Μοντέλο Αντικειμένων* (Object Model)
- *Γλώσσα Ορισμού Αντικειμένων* (Object Definition Language - ODL)
- *Γλώσσα Ερωταποκρίσεων Αντικειμένων* (Object Query Language - OQL)
- *Language Bindings (C++, Smalltalk)*

Η περιγραφή του ODMG-93 περιλαμβάνει ένα μοντέλο αντικειμένων που είναι μια επέκταση στο μοντέλο αντικειμένων του OMG [OMG97], μια γλώσσα ορισμού αντικειμένων (ODL) που παρέχει ένα μηχανισμό ανεξάρτητο από τις γλώσσες προγραμματισμού για να εκφράσει κανείς το σχήμα μιας βάσης, μια γλώσσα ερωταποκρίσεων αντικειμένων (OQL) που παρέχει ένα δηλωτικό τρόπο προσπέλασης των δεδομένων, σαν μια επέκταση της SQL, και

δυο επεκτάσεις αντικειμενοστρεφών γλωσσών προγραμματισμού. Η μια επέκταση είναι αυτή για τη C++ και η άλλη για τη Smalltalk.

Στο Σχήμα 4.8 φαίνεται ο τρόπος, με βάση τον οποίο θα χρησιμοποιείται ένα αντικειμενοστρεφές σύστημα βάσεων δεδομένων, στα πλαίσια της προδιαγραφής του ODMG--93. Ένας διαχειριστής της βάσης γράφει δηλώσεις για το σχήμα της, οι οποίες συγγράφονται στην ODL, ή σε μια επέκταση της γλώσσας προγραμματισμού (PL ODL). Ο προγραμματιστής εφαρμογών γράφει και ένα πρόγραμμα που υλοποιεί την εφαρμογή. Ο κώδικας του προγράμματος αυτού γράφεται σε μια γλώσσα προγραμματισμού (PL), όπως για παράδειγμα, η C++, η οποία επεκτείνεται για να αποτελέσει την γλώσσα προγραμματισμού μιας βάσης δεδομένων. Οι δηλώσεις περνούν από κάποιο προεπεξεργαστή (preprocessor), ο οποίος αφενός παράγει το σχήμα της βάσης και αφετέρου φτιάχνει μια περιγραφή του σχήματος στη γλώσσα προγραμματισμού της εφαρμογής. Ο κώδικας της εφαρμογής και η περιγραφή αυτή μεταφράζονται, δημιουργείται ένα εκτελέσιμο αρχείο της εφαρμογής, το οποίο αφού συνδυαστεί με τη μηχανή του αντικειμενοστρεφούς συστήματος βάσεων δεδομένων (μέσα από ένα πρόγραμμα σύνδεσης -linker) δημιουργεί την τελική εφαρμογή, η οποία και επεξεργάζεται τα δεδομένα της βάσης δεδομένων.



Σχήμα 4.8 Η χρήση ενός αντικειμενοστρεφούς συστήματος βάσεων δεδομένων, σύμφωνα με το ODMG

4.3.2 Το Μοντέλο Αντικειμένων του ODMG

Το μοντέλο δεδομένων του ODMG συνοψίζεται ως εξής:

- Η βασικές οντότητες είναι το *αντικείμενο* (object) και η *βασική τιμή* (literal). Κάθε αντικείμενο έχει ένα μοναδικό αναγνωριστικό. Μια βασική τιμή δεν έχει αναγνωριστικό.

- Η *κατάσταση* (state) των αντικειμένων χαρακτηρίζεται από τις τιμές που έχουν αυτά για κάποιες *ιδιότητες* (properties). Οι ιδιότητες μπορεί να είναι είτε *χαρακτηριστικά* (attributes) του αντικειμένου, είτε *αναφορές* (relationships) ανάμεσα στο αντικείμενο και τα άλλα αντικείμενα. Γενικά, οι τιμές των ιδιοτήτων ενός αντικειμένου μπορούν να αλλάξουν στο χρόνο.
- Η *συμπεριφορά* των αντικειμένων χαρακτηρίζεται από μια ομάδα *λειτουργιών* (operations) που μπορεί να επιτελεσθεί από (ή σε) ένα αντικείμενο.
- Τα αντικείμενα και οι βασικές τιμές κατηγοριοποιούνται σε *τύπους* (types). Όλα τα στοιχεία ενός δεδομένου τύπου έχουν ένα κοινό *ενεργό πεδίο* καταστάσεων (δηλαδή, το ίδιο σύνολο ιδιοτήτων) και κοινή συμπεριφορά (δηλαδή, το ίδιο σύνολο λειτουργιών). Ένα αντικείμενο αποτελεί *στιγμιότυπο* (instance) του τύπου του.
- Μια βάση δεδομένων αποθηκεύει αντικείμενα τα οποία μπορούν να μοιράζονται πολλοί χρήστες και πολλές εφαρμογές. Η βάση δεδομένων στηρίζεται σε ένα σχήμα που δηλώνεται σε ODL και περιέχει στιγμιότυπα των τύπων που δηλώθηκαν στο σχήμα.

Η ορολογία που χρησιμοποιεί το ODMG δε συμβαδίζει απόλυτα με την ορολογία που έχουμε χρησιμοποιήσει μέχρι εδώ. Η "κατάσταση" ενός αντικειμένου στο ODMG είναι ο όρος για την "τιμή" του αντικειμένου, όπως τον έχουμε χρησιμοποιήσει μέχρι τώρα.

4.3.3 Γλώσσα Ορισμού Αντικειμένων (Object Definition Language - ODL)

Η ODL είναι μια γλώσσα που χρησιμοποιείται για να δηλώσει *διαπροσωπείες* (interfaces) σε τύπους αντικειμένων που ακολουθούν το μοντέλο αντικειμένων του ODMG-93 και έχει σκοπό να παρέχει μεταφερσιμότητα των σχημάτων των βάσεων δεδομένων. Η ODL αποτελεί την DDL για τύπους αντικειμένων. Καθορίζει τα χαρακτηριστικά των τύπων και τις υπογραφές των μεθόδων. Για τη διαχείριση των στιγμιότυπων των τύπων, και την υλοποίηση των μεθόδων, υπάρχουν οι γλώσσες διαχείρισης αντικειμένων (Object Manipulation Languages - OMLs). Το ODMG-93 δεν καθορίζει κάποια συγκεκριμένη OML. Περιγράφει, όμως, δύο interfaces για τη σύνδεση των αντικειμενοστρεφών συστημάτων βάσεων δεδομένων με C++ και Smalltalk.

Δεδομένου ότι δεν είναι σαφές που θα υλοποιηθεί το σχήμα που θα καθοριστεί στην ODL, η ODL πρέπει να είναι ανεξάρτητη από την οποιαδήποτε γλώσσα προγραμματισμού. Επιπλέον, πρέπει να μπορεί να ενσωματωθεί με ομαλό τρόπο σε γλώσσες όπως η Smalltalk και η C++, πράγμα που καθορίζεται στα αντίστοιχα bindings. Ακολουθεί το παράδειγμα ενός υποτυπώδους σχήματος σε ODL (με έντονα γράμματα φαίνονται οι δεσμευμένες λέξεις της ODL)

```
interface Person
(  extent people)
{
   attribute String name;
   attribute Struct Address { Unsigned Short number, String
        Street, String city_name} address;
   relationship Person spouse inverse Person::spouse;
   relationship Set<Person> children inverse Person::parents
        {order by birth_date }
   relationship List<Person> parents inverse
        Person::children;
```

```
    void birth (in String name);
    Boolean marriage (in String person_name) raises
          (no_such_person);
    Unsigned Short ancestors (out Set<Person>
          all_ancestors)raises (no_such_person);
    void move (in String new_address);
};

interface Employee: Person
(   extent employees
    key(name, id))
{
    attribute Short id;
    attribute Unsigned Short annual_salary;
};

interface City
(   extent cities
    key city_code)
{
    attribute Unsigned Short city_code;
    attribute String name;
    attribute Set<Person> population;
};
```

Στο παραπάνω παράδειγμα, ορίσαμε την κλάση `Person` που μοντελοποιεί ένα άνθρωπο. Τα χαρακτηριστικά της εν λόγω κλάσης είναι το όνομα (`name`), η διεύθυνση (`Address`) που αποτελεί μια σύνθετη τιμή, η/ο σύζυγος (`spouse`) που αποτελεί ένας αντίστροφος δείκτης (inverse) και οι αντίστροφοι δείκτες `children` και `parents`, που μοντελοποιούν τα παιδιά και τους γονείς αντίστοιχα. Ακόμα υπάρχουν οι συναρτήσεις `birth` (ημερομηνία γεννήσεως), `marriage` (αν κάποιος είναι παντρεμένος ή όχι), `ancestors` (που επιστρέφει όλους τους προγόνους κάποιου) και `move` (που αλλάζει τη διεύθυνση κάποιου. Η κλάση `Employee` κληρονομεί την κλάση `Person` και την εξειδικεύει προσθέτοντας ένα κωδικό (`id`) και ένα ετήσιο εισόδημα (`annual_salary`). Τέλος, η κλάση `City` έχει τα χαρακτηριστικά `city_code` (κωδικός), `name` (όνομα) και population, που είναι ο πληθυσμός της και είναι ένα σύνολο από αντικείμενα της κλάσης `Person`.

4.3.4 Object Query Language - OQL

Η OQL βασίστηκε στην ήδη υλοποιημένη γλώσσα ερωταποκρίσεων του συστήματος O_2. Η OQL μπορεί να χρησιμοποιηθεί και σαν ανεξάρτητη γλώσσα, αλλά και σαν υποσύνολο μιας από τις OML. Στη δεύτερη περίπτωση, υπάρχει μια συνάρτηση που εκτελεί την ερώτηση.

Ας υποθέσουμε ότι έχουμε τον τύπο `Person` με τα χαρακτηριστικά `name`, `birthdate`, `salary` και τη μέθοδο age και τον τύπο `Employee` που κληρονομεί τον τύπο `Person` και τον εξειδικεύει με την *αναφορά* (relationship) `subordinates` και τη μέθοδο `seniority`. Επίσης, κάποιο πρόσωπο είναι ο `Chairman` και μάλιστα υπάρχει κάποιο entry-point για αυτό το αντικείμενο. Τέλος, οι δύο τύποι έχουν τα extents `Persons` και `Employees`. Έστω

επίσης, ότι υπάρχει και ο τύπος `Department`, με το χαρακτηριστικό `name` και την αναφορά `employees`, η οποία τυχαίνει να είναι και αντίστροφη (στον τύπο `Employee` υπάρχει η αντίστροφη αναφορά `department`). Έστω η ερώτηση:

```
select distinct struct(name: x.name, hps:
           (select y
            from x.subordinates as y
            where y.age > 30))
from Employees x
```

η οποία επιστρέφει για κάθε υπάλληλο, το όνομά του και τους υφιστάμενούς του που έχουν ηλικία πάνω από 30. Η παραπάνω ερώτηση έχει αποτέλεσμα του τύπου

```
set<struct(name: string, hps: bag<Employee>)>
```

Παρατηρούμε τα εξής:

- Μπορούμε να ονομάζουμε τα χαρακτηριστικά του αποτελέσματος της ερώτησης
- Στο *from clause* μπορούμε να χρησιμοποιήσουμε πολλούς φορμαλισμούς. Μπορούμε να πούμε `from Employees x`, ή `from Employees as x`, ή ακόμα και `from x in Employees`.
- Μπορούμε να έχουμε φώλιασμα ερωτήσεων και μεταβλητών.
- Μπορούμε να χρησιμοποιούμε εκφράσεις μονοπατιών μέσα σε μια ερώτηση
- Μπορούμε να χρησιμοποιούμε μεθόδους οπουδήποτε μέσα σε μια ερώτηση

Η ερώτηση

```
Chairman
```

αν και δεν υπακούει στο γνωστό *SELECT-FROM-WHERE* φίλτρο της SQL, είναι έγκυρη και επιστρέφει το αντικείμενο Chairman.

Η ερώτηση

```
Employee(name: "Pat", id: 0, annual_salary: 100,000)
```

δημιουργεί ένα αντικείμενο της κλάσης Employee.

Πρέπει να επισημάνουμε ότι το αποτέλεσμα μιας ερώτησης μπορεί να έχει ή να μην έχει ταυτότητα αντικειμένου, ανάλογα με τον τύπο του.

Ας υποθέσουμε και μία κλάση Flowers με το χαρακτηριστικό name.

```
select p
from Persons p, Flowers f
where p.name = f.name
```

Η παραπάνω ερώτηση που βρίσκει όλους τους ανθρώπους που το όνομά τους είναι και το όνομα ενός άνθους, υλοποιεί μια σύνδεση, όπως θα γινόταν και στα σχεσιακά συστήματα.

Στην ερώτηση

```
select department, avg_salary: avg(select e.salary from
     partition x)
from Employees e
group by department: e.department
having count(select * from partition x) > 5
order by department.name
```

μπορούμε να παρατηρήσουμε ότι διατηρούνται τα group by, having και order by clauses της SQL, τα οποία χαρακτηρίζονται από τα εξής:

- Το clause *group by* χωρίζει τους υπαλλήλους σε *ομάδες* (partitions). Κάθε ομάδα χαρακτηρίζεται από το γεγονός ότι όλοι οι υπάλληλοι που ανήκουν σ' αυτό δουλεύουν στο ίδιο τμήμα και ονομάζεται partition (που είναι δεσμευμένη λέξη στην OQL). Επιπλέον το clause *having* περιορίζει τα τμήματα με τα οποία θα ασχοληθεί η ερώτηση, σ' αυτά που έχουν αριθμό υπαλλήλων πάνω από 5.
- Η συντόμευση * διατηρείται και στην OQL (όπως και στην SQL)
- Το *order by* λειτουργεί όπως και στην SQL

Τέλος, στην OQL υπάρχουν και διάφορα άλλα χαρακτηριστικά, τα οποία δεν κρίνεται σκόπιμο να αναφερθούν. Για μια πιο λεπτομερή παρουσίαση, ο αναγνώστης παραπέμπεται στο [Ca95].

4.3.5 C++ Binding

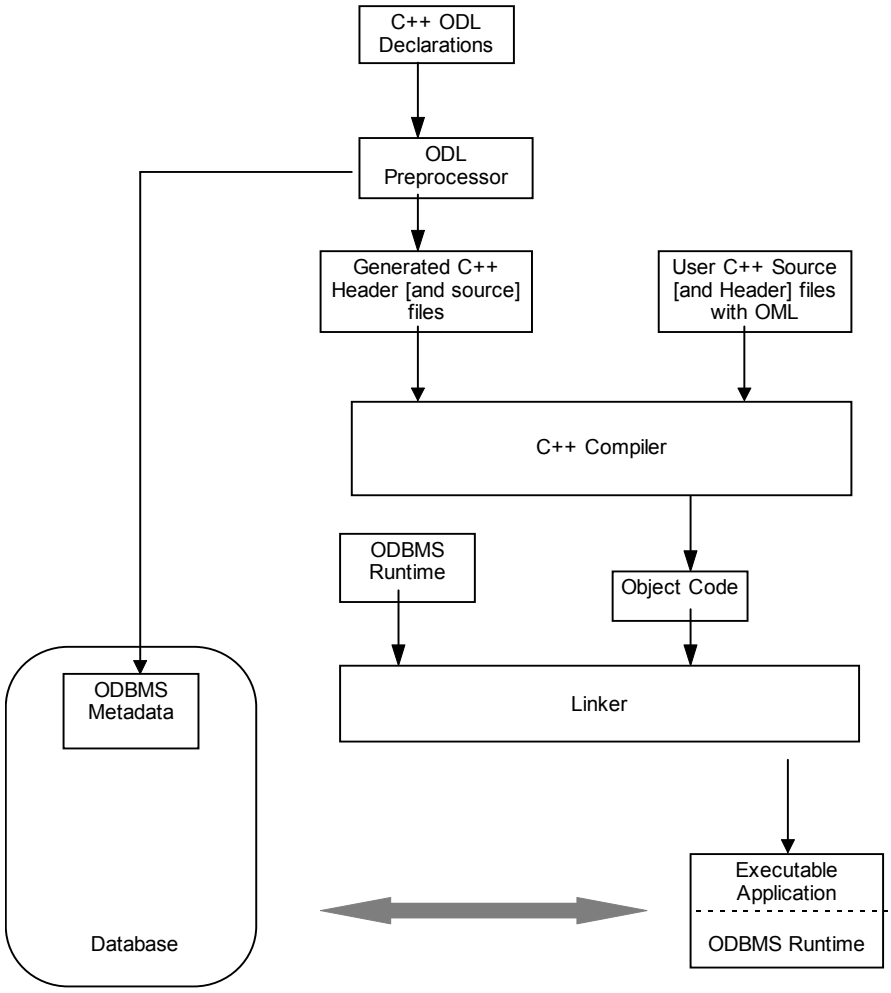
Στη συνέχεια, θα παρουσιάσουμε ένα από τα χρησιμοποιούμενα bindings, το C++ binding, το οποίο είναι και το πλέον χρησιμοποιούμενο. Όταν αναφερόμαστε στο C++ binding για ODL/OML πρέπει να έχουμε υπόψη μας δύο πράγματα: το C++ binding για ODL είναι μια βιβλιοθήκη κλάσεων και μια επέκταση στη γραμματική της C++. Η βιβλιοθήκη των κλάσεων παρέχει κλάσεις για να υλοποιηθεί το μοντέλο αντικειμένων του ODMG-93. Η επέκταση του συντακτικού της γλώσσας έχει να κάνει με την υποστήριξη αναφορών (relationships) μεταξύ των κλάσεων. Η OML χρησιμοποιείται για την ανάκτηση και την επεξεργασία δεδομένων από τη βάση, και ακολουθεί το συντακτικό της C++. Να σημειωθεί ότι οι ODL/OML δεν ασχολούνται με τη φυσική αποθήκευση των αντικειμένων, αλλά μόνο με τα λογικά χαρακτηριστικά τους. Ένα πρόσθετο σύνολο δομών, με όνομα *physical pragmas,* έχει καθοριστεί για να αναλάβει να δώσει στον προγραμματιστή κάποιο έλεγχο πάνω σε αυτά τα ζητήματα.

Τα bindings για τις διάφορες γλώσσες προγραμματισμού καθορίστηκαν με βάση την αρχή ότι ο προγραμματιστής πρέπει να έχει να κάνει με μόνο μία γλώσσα. Αποτελέσματα της αρχής αυτής είναι, ότι υπάρχει μόνο ένα σύστημα τύπων και για τη γλώσσα και για τη βάση δεδομένων, και ότι η ενσωμάτωση των χαρακτηριστικών που οφείλονται στο μοντέλο αντικειμένων της βάσης δεδομένων γίνεται ομαλά.

Για να δηλωθούν τα διαρκή (persistent) δεδομένα στη C++, μια κλάση πρέπει να είναι υποκλάση της κλάσης Persistent_Object. Αυτό σημαίνει, ότι η κλάση αυτή μπορεί να έχει και

διαρκή (persistent) και μεταβατικά (transient) -μέσω κάποιων συναρτήσεων- αντικείμενα. Στο παρακάτω παράδειγμα, ακολουθεί η δήλωση ενός στιγμιότυπου (`profP`) μιας κλάσεως (`Professor`), το οποίο είναι διαρκές:

```
Ref<Professor> profP;
```



Σχήμα 4.9 Παραγωγή εφαρμογής με το C++ binding

Στο Σχήμα 4.9 φαίνεται ο τρόπος με τον οποίο παράγεται μια εφαρμογή, χρησιμοποιώντας το C++ binding για ODL/OML. Ένας διαχειριστής της βάσης γράφει δηλώσεις για το σχήμα της, οι οποίες συγγράφονται στην C++ ODL, η οποία είναι μια επέκταση της C++. Ο προγραμματιστής εφαρμογών γράφει και ένα πρόγραμμα που υλοποιεί την εφαρμογή σε C++, η οποία επεκτείνεται για να αποτελέσει την γλώσσα προγραμματισμού μιας βάσης δεδομένων. Οι δηλώσεις περνούν από κάποιο προεπεξεργαστή (preprocessor), ο οποίος αφενός παράγει το σχήμα της βάσης και αφετέρου φτιάχνει μια περιγραφή του σχήματος στη C++. Ο κώδικας της εφαρμογής και η περιγραφή αυτή μεταφράζονται, δημιουργείται ένα εκτελέσιμο αρχείο της εφαρμογής, το οποίο αφού συνδυαστεί με τη μηχανή του αντικειμενοστρεφούς συστήματος

βάσεων δεδομένων (μέσα από ένα πρόγραμμα σύνδεσης -linker) δημιουργεί την τελική εφαρμογή, η οποία και επεξεργάζεται τα δεδομένα της βάσης δεδομένων.

Όπως προαναφέρθηκε, η ODL καθορίζει το σχήμα μιας βάσης, ενώ η OML καθορίζει τη διαχείριση των αντικειμένων, των σχέσεων μεταξύ των κλάσεων, καθώς και των συλλογών (set, bag, list, array). Σε ότι αφορά τις *δοσοληψίες* (transactions), είναι υποχρεωτικό, κάθε λειτουργία διαχείρισης αντικειμένων να γίνεται μέσα από μια δοσοληψία. Οι δοσοληψίες αποτελούν στιγμιότυπα της ομώνυμης κλάσης και διαθέτουν λειτουργίες commit, abort και checkpoint. Δεν υποστηρίζονται long transactions, υποστηρίζονται όμως, nested transactions. Επιπλέον, η OML είναι υπεύθυνη και για τις λειτουργίες σύνδεσης και αποσύνδεσης με μία βάση.

Υπάρχει και μια απεικόνιση της OQL για C++. Οι ερωτήσεις μπορούν να δηλωθούν μέσα σε ένα πρόγραμμα μέσω μιας συναρτήσεως για ερωτήσεις, η οποία παίρνει strings για ορίσματα. Τα ορίσματα αυτά πρέπει να είναι έγκυρες ερωτήσεις, οι οποίες μεταφράζονται και εκτελούνται στο run-time. Το αποτέλεσμα επιστρέφεται στην παράμετρο "result".

4.4 ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΕΙΣ ΕΠΕΚΤΑΣΕΙΣ ΤΩΝ ΣΧΕΣΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (SQL/Object, SQL/Foundation)

Στην ενότητα αυτή θα εξετάσουμε τον τρόπο με τον οποίο οι παρουσία των αντικειμενοστρεφών βάσεων δεδομένων επηρέασε την εξέλιξη των κλασικών, σχεσιακών συστημάτων. Διάφορα σχεσιακά ΣΔΒΔ (Oracle, Informix κλπ.) άρχισαν σιγά σιγά, στα μέσα της δεκαετίας του '90 να παρέχουν αντικειμενοστρεφείς επεκτάσεις στη μηχανή τους. Η πιο βασική, όμως, έκφανση της επίδρασης που είχε η ύπαρξη των αντικειμενοστρεφών συστημάτων ήταν η υιοθέτηση αντικειμενοστρεφών επεκτάσεων στο καινούριο στάνταρτ της SQL, το SQL-3. Πιο συγκεκριμένα, υπάρχουν τέσσερις κύριοι τομείς στους οποίους συνοψίζεται αδρά, η επίδραση αυτή. Οι τομείς αυτοί είναι:

- Εισαγωγή τύπων που ορίζονται από το χρήστη -UDTs
- Κατηγορήματα για την κατασκευή τους -Type Predicate
- Συναρτήσεις που μπορούν να οριστούν από το χρήστη - User Defined Functions & Procedures
- Κατάργηση της πρώτης κανονικής μορφής (1NF) - Collection Types

Θα στηριχθούμε στα [KMN97], [MM96] για την παρουσίαση αυτή.

4.4.1 Τύποι Ορισμένοι από τον Χρήστη

Οι τύποι που μπορούν να οριστούν από το χρήστη (*User Defined Types -UDTs)* διαιρούνται με τη σειρά τους σε διάφορες κατηγορίες:

- Διακριτοί τύποι - Distinct Types
- Αφηρημένοι τύποι - Abstract Data Types (ADTs)
- Τύποι γραμμών / Τύποι αναφοράς - Row Types / Reference Types

Στη συνέχεια, θα εξετάσουμε μία μία αυτές τις κατηγορίες.

Διακριτοί Τύποι

Οι *διακριτοί τύποι* (distinct types) αποτελούν την πιο βασική έκφραση των ορισμένων από τον χρήστη τύπων. Ουσιαστικά, αποτελούν μετονομασία ενός βασικού τύπου πηγή (π.χ. `DECIMAL`, `REAL`, `INTEGER`). Στην συνέχεια, θα αναφερόμαστε στον τύπο βάσει του οποίου ορίζεται ένας διακριτός τύπος, ως *πηγαίο τύπο* (source type) ή *βασικό τύπο αναφοράς* του

διακριτού τύπου. Ο διακριτός τύπος με τον πηγαίο τύπο έχουν την ίδια εσωτερική δομή αλλά παρουσιάζουν εν γένει διαφορετική συμπεριφορά.

Για παράδειγμα, μπορούμε να δηλώσουμε δύο διακριτούς τύπους, παράγωγα του DECIMAL, που θα αναπαριστούν το αμερικάνικο (US_DOLLAR) και το καναδικό δολάριο (CND_DOLLAR).

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9.2)
CREATE DISTINCT TYPE CDN_DOLLAR AS DECIMAL (9.2)
```

Οι διακριτοί τύποι είναι εφοδιασμένοι από το σύστημα με τελεστές σύγκρισης. Ο ορισμός των τελεστών βασίζεται στους αντιστοίχους ορισμούς του πηγαίου τύπου.

Παρόλο που οι διακριτοί τύποι παρουσιάζουν την ίδια εσωτερική δομή με τον πηγαίο τύπο, δεν είναι άμεσα συγκρίσιμοι μαζί του (strongly typed). Για να μπορέσουμε να συγκρίνουμε ένα διακριτό τύπο με τον πηγαίο του, καθώς και διακριτούς τύπους μεταξύ τους (που έχουν βέβαια κοινή πηγή) χρησιμοποιούμε *συναρτήσεις μετατροπής* (casting).

Με χρήση των διακριτών τύπων μπορούμε να ορίζουμε σχέσεις. Για παράδειγμα:

```
CREATE TABLE SALES
         (ID INTEGER, US US_DOLLAR, CDN CDN_DOLLAR)
```

Οι μεταβλητές US, CND βασίζονται στον ίδιο πηγαίο τύπο αλλά ανήκουν σε διαφορετικούς διακριτούς τύπους. Ως εκ τούτου η ακόλουθη έκφραση είναι λανθασμένη.

```
SELECT * FROM SALES WHERE CDN > US
```

Για να πετύχουμε την σύγκριση χρησιμοποιούμε την συνάρτηση μετατροπής CND_DOLLAR η οποία μετατρέπει το όρισμα της σε καναδικά δολάρια.

```
SELECT * FROM SALES WHERE CDN > CDN_DOLLAR(US)
```

Αντίστοιχα, θα μπορούσαμε να χρησιμοποιήσουμε την συνάρτηση μετατροπής US_DOLLAR. Τέλος, ας σημειωθεί ότι οι διακριτοί τύποι δεν υποστηρίζουν κληρονομικότητα ούτε είναι δυνατό να ορίσουμε ένα διακριτό τύπο με πηγαίο έναν άλλο διακριτό τύπο.

Αφηρημένοι τύποι δεδομένων

Έκτος από τους διακριτούς τύπους δεδομένων μπορούμε να ορίσουμε και *αφηρημένους τύπους δεδομένων* (Abstract Data Types, ADTs). Οι αφηρημένοι τύποι δεδομένων αποτελούν οντότητες με συμπεριφορά και ενθυλακωμένη εσωτερική δομή. Ο ορισμός της δομής ενός ATD γίνεται σε SQL. Για παράδειγμα ορίζουμε τον ακόλουθο τύπο:

```
CREATE TYPE address
              (street char (30),
               city char (20),
               state char (2),
               zip integer);
```

Ας εξετάσουμε όμως κάθε ένα από τα χαρακτηριστικά των αφηρημένων τύπων δεδομένων.

Ενθυλάκωση (Encapsulation)

Όπως έχουμε ήδη αναφέρει, η εσωτερική δομή των αφηρημένων τύπων είναι ενθυλακωμένη. Η πρόσβαση στα πεδία του τύπου περιορίζεται σε συναρτήσεις που ορίζονται στο ορατό μέρος του κάθε αφηρημένου τύπου. Παράλληλα, δεν υπάρχει διάκριση μεταξύ *συναρτήσεων*, *αποθηκευμένων συναρτήσεων* (stored functions) και *ιδεατών πεδίων* (virtual attributes).

Οι συναρτήσεις με τις οποίες μας επιστρέφεται η τιμή ενός πεδίου (observer functions) ή με τις οποίες αναθέτουμε μια τιμή σε κάποιο πεδίο (mutator functions) δημιουργούνται αυτόματα. Για παράδειγμα:

```
street(address)     -> char(30)
city(address)       -> char(20)
state(address)      -> char(2)
zip(address)  -> integer

street(address, char(30))      -> address
city(address, char(20))  -> address
state(address, char(2))  -> address
zip(address, integer)          -> address
```

Η κλήση της `street(address)` θα επιστρέψει ένα πεδίο `char(30)` που αντιστοιχεί στο πεδίο `street` του τύπου `address`. Αντίστοιχα, η κλήση της συνάρτησης `street(address, char(30))` θα δημιουργήσει ένα στιγμιότυπο του τύπου `address` με τιμή πεδίου `address` την τιμή του `char(30)`.

Δημιουργία Στιγμιότυπων

Για να δημιουργήσουμε ένα νέο στιγμιότυπο χρησιμοποιούμε την *κατασκευαστική συνάρτηση* (constructor function). Κάθε φόρα που ορίζουμε ένα ADT δημιουργείται αυτόματα και μια κατασκευαστική συνάρτηση.

```
address() -> address
```

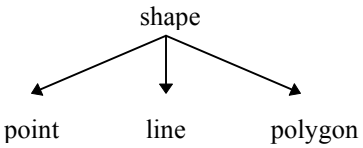
Με την κλήση της παραπάνω συνάρτησης κατασκευάζουμε ένα νέο στιγμιότυπο του τύπου `address` με κάθε πεδίο του αρχικοποιήμενο στην εξ' ορισμού τιμή.

Κληρονομικότητα

Σε αντίθεση με τους διακριτούς τύπους, οι αφηρημένοι τύποι μπορούν να οριστούν με βάση έναν ή περισσότερους διαφορετικούς αφηρημένους τύπους (subtyping). Έτσι ο νέος τύπος (υποτύπος) κληρονομεί τα αντίστοιχα πεδία, την συμπεριφορά και τις συναρτήσεις των τύπων ορισμού του (υπερτύποι). Τονίζουμε ότι υποστηρίζεται και πολλαπλή κληρονομικότητα.

```
CREATE TYPE shape …
   CREATE TYPE point UNDER shape …
   CREATE TYPE line UNDER shape …
```

```
CREATE TYPE polygon UNDER shape …
```



Στο παράδειγμα μας ο υπερτύπος είναι ο τύπος `shape` και οι υποτύποι είναι οι τύποι `point, line` και `polygon`.

Χρήση

Οι αφηρημένοι τύποι δεδομένων μπορούν να χρησιμοποιηθούν όπως και οι βασικοί τύποι της SQL. Έστω ότι ορίζουμε ένα ADT με όνομα `shape` ως εξής:

```
CREATE TYPE shape
    (refercing_system     INTEGER,
     tolerance       DECIMAL(8.2),
     geometry        BLOB(1M));
```

Με βάση τον παραπάνω τύπο μπορούμε να ορίσουμε και την σχέση:

```
CREATE TABLE real_estate_info
    (address   address,
     price     money,
     owner     char(40),
     property  shape);
```

Η διαχείριση των στιγμιότυπων γίνεται με την κλήση των αντιστοίχων συναρτήσεων ορισμού. Χρησιμοποιούμε σαν διαχωριστή πεδίου τις δύο τελείες (dot notation). Για παράδειγμα:

```
INSERT real_estate_info
   VALUES (US_DOLLAR(30000),'Joe Doe',spatial(10,12,15,20))

SELECT D_mark(price), owner
FROM real_estate_info
WHERE overlaps(property..geometry, square (5, 5, 25, 25))
```

Δυνατότητα Αντικατάστασης (Substitutability)

Για να κατανοήσουμε την έννοια της *δυνατότητας αντικατάστασης* θα χρησιμοποιήσουμε ένα παράδειγμα. Ας υποθέσουμε την ύπαρξη της ακόλουθης σχέσης:

```
CREATE TABLE real_estate_info
         (price money,
          owner CHAR (40),
```

```
        property shape);
```

Ακόμα, ας υποθέσουμε ότι κάνουμε τις ακόλουθες εισαγωγές:

```
INSERT INTO real_estate_info VALUES
        (US_DOLLAR(100000),'Mr. S. White',point(4, 4))
INSERT INTO real_estate_info VALUES
        (CDN_DOLLAR(400000),'Mr. W. Green',
         poly(point(4, 4), point(10, 10), point(12, 14)))
INSERT INTO SALES VALUES
        (S_FRANK(150000),'Mrs. D. Black',line(5, 5, 7, 8))
```

Τα δεδομένα φυλάσσονται ως εξής:

table real_estate_info

price	owner	property
<us-dollar> amount: 100000	Mr. S. White	*<point>*
<cnd-dollar> amount: 400000	Mr. W. Green	*<poly>*
<swiss-frank> amount: 150000	Mrs. D. Black	*<line>*

τύποι μονάδας

Παρατηρούμε ότι στην στήλη `price` εκτός από τις τιμές φυλάσσουμε και πληροφορία σχετική με την μονάδα μέτρησης. Ακόμα, αν έχουμε δηλώσει ένα πεδίο σαν τύπο Α, η τιμή του πεδίου αυτού μπορεί να είναι οποιαδήποτε στιγμιότυπο του τύπου Α ή κάποιου από τους υποτύπους του. Έτσι στο πεδίο `property` του τύπου `shape` μπορούμε να αναθέσουμε στιγμιότυπα των υποτύπων `point`, `poly` και `line`. Τα στοιχεία του υποτύπου βρίσκονται κατά τον χρόνο εκτέλεσης, γεγονός που απαιτεί καθυστέρηση στην αποτίμηση (late binding). Το βασικό χαρακτηριστικό της καθυστέρησης στην αποτίμηση είναι η αυτόματη κλήση των συναρτήσεων μετατροπής κατά την εκτέλεση μιας ερώτησης. Για παράδειγμα, κατά την εκτέλεση της ερώτησης

```
SELECT owner, dollar_amount(price)
FROM real_estate_info
WHERE dollar_amount(price)<US_DOLLAR(500000);
```

θα κληθούν αυτόματα οι συναρτήσεις μετατροπής `US_DOLLAR`, `CND_DOLLAR` και `SWISS_FRANK` ανάλογα με τον τύπο των δεδομένων της στήλης `price`.

Επώνυμοι Τύποι Γραμμής (Named Row Types)

Οι τύποι που έχουμε μέχρι τώρα εξετάσει αναφέρονται στον ορισμό νέων πεδίων. Υπάρχουν όμως και τύποι για την αναπαράσταση μιας ολόκληρης πλειάδας μια σχέσης -ενός συνόλου δηλαδή πεδίων. Οι τύποι αυτοί ονομάζονται *επώνυμοι τύποι γραμμής* (named row types). Γενικά, οι επώνυμοι τύποι γραμμής αποτελούν τύπους ορισμένους από τον χρήστη χωρίς ενθυλακωμένη εσωτερική δομή. Για παράδειγμα:

```
CREATE ROW TYPE account_t
         (acctno    INTEGER,
          cust      REF(customer_t),
          type      CHAR (1),
          opened    DATE,
          rate      DOUBLE PRECISION,
          balance   DOUBLE PRECISION);
```

Χρησιμοποιώντας τους επώνυμους τύπους γραμμής μπορούμε να ορίσουμε σχέσεις. Για παράδειγμα:

```
CREATE TABLE account OF account_t
         (PRIMARY KEY acctno);
```

Τύποι Αναφοράς (Reference types)

Οι επώνυμοι τύποι γραμμής μπορούν να συνδυαστούν με αναφορές, όπως στο πεδίο `cust` του τύπου γραμμής `account_t`. Οι τύποι αναφοράς παραπέμπουν σε δηλώσεις τύπων (στην συγκεκριμένη περίπτωση του τύπου `customer_t`) και χρησιμοποιούνται κυρίως για να μοντελοποιήσουν σχέσεις μεταξύ επώνυμων τύπων γραμμών.

Οι αναφορές μπορούν να έχουν *εμβέλεια* (scope).

```
CREATE TABLE account OF account_t
         (PRIMARY KEY acctno,
          SCOPE FOR cust IS customer);
```

Ας σημειωθεί ότι σε μια σχέση μπορούμε να ορίσουμε αναφορές μόνο σε μεταβλητές που βρίσκονται στο πρώτο επίπεδο (π.χ. `cust`). Επιπλέον μπορούμε να χρησιμοποιούμε τους τύπους αναφοράς για να εκφράσουμε ερωτήσεις:

```
SELECT a.acctno, a.cust->name
FROM account a
WHERE a.cust->address..city = "Hollywood"
   AND a.balance > 1000000;
```

Υποτύποι και κληρονομικότητα

Ένας επώνυμος τύπος γραμμής μπορεί να δηλωθεί σαν υπερτύπος από έναν ή περισσότερους επώνυμους τύπους γραμμής. Οι υποτύποι κληρονομούν τις μεταβλητές και τις συναρτήσεις από τους υπερτύπους τους.

Έστω ότι ορίζουμε ένα τύπο γραμμής για να αναπαραστήσουμε εργαζόμενους.

```
CREATE TYPE empoloyee
              (name CHAR (20), salary DECIMAL (10,2))
```

Βασισμένοι στον τύπο αυτό ορίζουμε δυο νέους για τους προϊστάμενους και τους φοιτητές που εργάζονται το καλοκαίρι.

```
CREATE TYPE manager UNDER employee
              (bonus DECIMAL (10,2));
CREATE TYPE summer_stundent UNDER employee
              (school VARCHAR (30));
```

Χρησιμοποιώντας τύπους και υποτύπους μπορούμε να ορίσουμε ιεραρχίες σε σχέσεις. Έστω:

```
CREATE TABLE employees OF employee;
CREATE TABLE managers OF manager UNDER employees;
CREATE TABLE summer_students OF summer_student
                              UNDER employees;
```

Πρακτικά, ιεραρχία στις σχέσεις σημαίνει ότι αν εκφράσουμε ερώτηση που αφορά μια δεδομένη σχέση, θα αναζητηθούν απαντήσεις και σε όλες τις σχέσεις που κληρονομούν από αυτήν. Έτσι, απευθύνοντας ερώτηση στην σχέση `employees` αυτόματα εξετάζονται και δεδομένα από τις σχέσεις `managers` και `summer_students`.

4.4.2 Τύποι Κατηγορημάτων (Type Predicate)

Ας ξαναδούμε την σχέση `real_estate_info`.

```
CREATE TABLE real_estate_info
    (price money,
     owner CHAR(40),
     property real_estate);
```

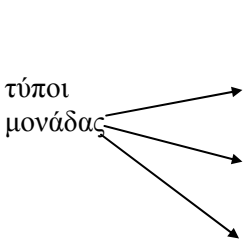


table real_estate_info

price	owner	property
<us-dollar> amount: 100000	Mr. S. White	*<point>*
<cnd-dollar> amount: 400000	Mr. W. Green	*<poly>*
<swiss-frank> amount: 150000	Mrs. D. Black	*<line>*

Είναι δυνατό μια ερώτηση, κατά τον χρόνο εκτέλεσης, να ελέγχει το είδος του τύπου που έχει ορίσει ο χρήστης:

```
SELECT price, owner, property
FROM real_estate_info
WHERE TYPE(price) IN US_DOLLAR;
```

Επίσης είναι δυνατό μια ερώτηση να μετατρέψει ένα τύπο σε ένα άλλο υποτύπο του (subtype specification):

```
SELECT TREAT (price AS US_DOLLAR), owner, property
FROM real_estate_info
WHERE TYPE (price) IN (US_DOLLAR);
```

4.4.3 Συναρτήσεις και Μέθοδοι ορισμένοι από τον χρήστη (User Defined Functions-Methods)

Ο χρήστης είναι δυνατό να ορίζει *συναρτήσεις, μεθόδους* και *αποθηκευμένες διαδικασίες* (stored procedures) βασισμένες σε κάποιο τύπο (ADTs, row types, κλπ.). Οι δηλώσεις τους γίνονται είτε σε SQL/PSM, είτε σε γλώσσες τρίτης ή τέταρτης γενιάς (3GLs ή 4GLs).

Διαχωρίζουμε τις συναρτήσεις που μπορεί ο χρήστης να ορίσει σε *εξωτερικές* (external) και σε *εσωτερικές* (internal). Οι εξωτερικές συναρτήσεις γράφονται και δηλώνονται σε μια από τις καθιερωμένες γλώσσες προγραμματισμού (ADA, C κλπ.). Οι συναρτήσεις αυτές προφανώς υπόκεινται στους περιορισμούς αλλά και εκμεταλλεύονται τις βιβλιοθήκες και την λειτουργικότητα της γλώσσας στην οποία ορίζονται. Παράλληλα, μπορούν να περιέχουν και ενσωματωμένη SQL. Αντίθετα, οι εσωτερικές συναρτήσεις είναι δηλωμένες εξ' ολοκλήρου σε SQL, εφοδιασμένη με συναρτήσεις ροής (loop, case, if, for, repeat, return κτλ.) και ανάθεσης (set) (βλέπε ακόμα και SQL/PSM στο Κεφάλαιο 3).

4.4.4 Συγκεντρωτικοί Τύποι (Collection Types)

Οι συγκεντρωτικοί τύποι αποτελούν δομές όπως σύνολα (sets), λίστες (lists), πίνακες (arrays) και πολυσύνολα (bags).

```
CREATE ROW TYPE employee
         (id        INTEGER,
          name      VARCHAR(30),
          address   address,
          manager   REF (employee),
          projects  SET (REF (project)),
          children  LIST (REF (PERSON)),
          hobbies   SET (VARCHAR (20)))
```

Στις ερωτήσεις τα στιγμιότυπα των συγκεντρωτικών τύπων συμπεριφέρονται σαν σχέσεις.

```
SELECT e.name
FROM employees e
WHERE 'travel' IN (SELECT *
                   FROM TABLE (e.hobbies) c)
   AND 5 > (SELECT count(*) FROM TABLE (e.children) d)
```

Επιπρόσθετα οι πίνακες και οι λίστες υποστηρίζουν και δεικτοδότηση για πιο γρήγορη πρόσβαση.

Ας εξετάσουμε μερικά παραδείγματα. Ας υποθέσουμε ότι έχουμε ορίσει δυο σχέσεις που αφορούν εργαζόμενους και τμήματα (`employees` και `departments` αντίστοιχα) ως εξής:

```
CREATE TABLE employees
         (id        INTEGER PRIMARY KEY,
          name      VARCHAR (30),
          address   ROW (street VARCHAR (40),
          city      CHAR(20),
          state     CHAR (2)
          zip       INTEGER,
          country   VARCHAR(30),
          salary    DECIMAL(10,2),
          deptno    INTEGER REFERENCES departments,
          manager   INTEGER RFEFERENCES employees,
          projects  SET (INTEGERS),
          children  LIST (person),
          hobbies   SET (VARCHAR(20)));
CREATE  TABLE departments
         (deptno    INTEGER PRIMARY KEY,
          manager   INTEGER REFERENCES employees,
          projects  TABLE (projno INTEGER),
          projname  CHAR (30),
          budget    DECIMAL (10,2));
```

Μετασχηματισμός συγκεντρωτικών τύπων σε πίνακες

Ερώτηση: Βρείτε όλους τους εργαζόμενους που δουλεύουν σε περισσότερα από δύο έργα.

```
SELECT name
FROM employees e
WHERE 2 < (SELECT count(*) FROM TABLE (e.projects) d)
```

Στο παραπάνω παράδειγμα, παρατηρούμε ότι μπορούμε να χρησιμοποιούμε σαν πίνακα, οποιαδήποτε attributes τύπου `SET` (εδώ συγκεκριμένα χρησιμοποιούμε το φορμαλισμό `TABLE (e.projects)`).

Μετασχηματισμός πινάκων σε συγκεντρωτικούς τύπους

Ερώτηση: Βρείτε όλους τους εργαζόμενους που δεν είναι προϊστάμενοι.

```
SET (TABLE employees
EXCEPT
(SELECT * FROM employees e WHERE e.id IN
   (SELECT ee.manager FROM employees ee)))
```

Στο παραπάνω παράδειγμα, παρατηρούμε ότι μπορούμε να κάνουμε και την αντίστροφη διαδικασία: να μετατρέπουμε, δηλαδή, αποτελέσματα τύπου πίνακα σε αποτελέσματα τύπου `SET`. Αυτό γίνεται μέσω της δεσμευμένης λέξης `SET`.

Ερωτήσεις σε συγκεντρωτικούς τύπους

```
BEGIN
 DECLARE hobbies_var SET (VARCHAR (20));
 SELECT hobbies INTO hobbies_var
 FROM employees e
 WHERE name = 'John Doe';

 SELECT ITEM ROW h
 FROM h IN hobbies_var
 WHERE h <= 'A';

 SELECT ITEM ROW h
 FROM h IN hobbies_var /* quantifiers are allowed */
 WHERE FOR SOME j IN (SELECT hobbies
                      FROM employees e
                      WHERE name = 'Mary Doe')
         (h = j);
END;
```

Στο παραπάνω παράδειγμα, παρατηρούμε τα εξής:

- Κατ' αρχήν, όπως παρουσιάζεται εκτενέστερα και στο κεφάλαιο 3, μπορούμε να γράφουμε προγράμματα σε μια υπολογιστικά πλήρη γλώσσα. Έτσι, για παράδειγμα, μπορούμε να ορίζουμε block εντολών, μεταβλητές, κλπ.
- Κατά δεύτερον, μπορούμε να εισάγουμε τιμές στις μεταβλητές τύπου `SET`, όπως φαίνεται από την πρώτη ερώτηση
- Ακόμα, έχουμε εναλλακτικούς φορμαλισμούς για την εκτέλεση συνδέσεων `(JOIN)` καθώς φαίνεται από την έκφραση `FOR SOME j IN (SELECT hobbies…)`

Unnesting.

Ερώτηση: Βρείτε τον κωδικό, τον προϋπολογισμό και το μέσο όρο μισθού για κάθε τμήμα.

```
SELECT d.deptno, d.budget, av.avg_sal
FROM departments d, (SELECT avg (salary) AS avg_sal
                     FROM e.deptno = d.deptno) AS av
```

ή όμοια

```
SELECT d.deptno, d.budget, AVG (e.salary) AS avg_sal
FROM departments d, employees e
WHERE d.deptno = e.deptno
GROUP BY d.deptno, d.budget
```

Στο παραπάνω παράδειγμα, παρατηρούμε ότι μπορούμε να εισάγουμε στο *from clause* μιας έκφρασης και ένα ολόκληρο query.

Flattening των αποτελεσμάτων

Ερώτηση: Βρείτε το σύνολο των εξωεργασιακών (sic!) δραστηριοτήτων του εργαζόμενου με κωδικό 12345.

```
THE (SELECT e.hobbies
     FROM employees e
     WHERE e.id = 12345)
```

Στο παραπάνω παράδειγμα, το αποτέλεσμα δεν είναι τύπου `ROW(SET(VARCHAR))`, αλλά `(SET(VARCHAR))`, λόγω της χρήσης του τελεστή `THE`.

Γενικά Παραδείγματα

Ερώτηση: Βρείτε τους εργαζόμενους που ο προϊστάμενος τους διαμένει σε μια συγκεκριμένη διεύθυνση.

```
SELECT e.id
FROM employees as e
WHERE row (row (123, University Avenue, Palo Alto, CA, 94301))
     = (SELECT mgr.address
        FROM employees as mgr
        WHERE e.manager = mgr.id)
```

Στο παραπάνω παράδειγμα μπορούμε να δούμε πώς συγκρίνουμε ένα αποτέλεσμα της μορφής `ROW` με ένα ολόκληρο πίνακα (θυμηθείτε ότι παλαιότερα, θα έπρεπε να χρησιμοποιήσουμε τελεστές όπως `IN`, `EXISTS`, κλπ)

Ερώτηση: Βρείτε τους προϊσταμένους και τα έργα με προϋπολογισμό μεγαλύτερο από 100000.

```
SELECT d.manager, multiset (SELECT *
                            FROM table (d.projects) as p
                            WHERE p.budget >100000)
FROM departments as d
```

Στο εν λόγω παράδειγμα, βλέπουμε πώς μπορούμε να έχουμε αποτέλεσμα τύπου `MULTISET`, αντί για τύπου `SET`.

Ερώτηση: Βρείτε πια από τις διευθύνσεις addr.1_v, addr.2_v, addr.3_v είναι πιο κοντά στην διεύθυνση 123, University Avenue, Palo Alto, CA, 94301 (δεδομένης μιας Boolean συνάρτησης κοντά, `(NEAR)`).

```
SELECT row addr
FROM (values (addr1_v, addr2_v, addr3_v)) as addr
WHERE near
   (row addr, row(123 University Avenue, Palo Alto, CA, 94301))
```

ή όμοια

```
SELECT row addr
FROM (multiset ( addr1_v, addr2_v, addr3_v)) as addr
WHERE near
   (row addr, row(123 University Avenue, Palo Alto, CA, 94301))
```

Στο συγκεκριμένο παράδειγμα, μπορούμε να δούμε αφενός τη χρήση συναρτήσεων που έχει φτιάξει ο χρήστης και αφετέρου ένα φορμαλισμό για την κατασκευή συνόλων από συγκεκριμένες τιμές (μέσω της έκφρασης `(values addr1_v, addr2_v, addr3_v) as addr`).

4.5 ΑΝΑΦΟΡΕΣ

[At+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik, "The Object-Oriented Database System Manifesto", In Banchilhon et. al. (eds), *Building an Object-Oriented Database System: The Story of O_2,Morgan Kaufmann, 1992.* Also in *Proc. Int. Conf. Deductive Object Oriented Databases, Kyoto, Japan, Dec. 1989.*

[BBKV87] F. Bancilhon, T. Briggs, S. Khoshafian, P. Valduriez, "FAD, a Powerful and Simple Database Language", *Proceedings of the 13th VLDB Conference, Brighton 1987.*

[Ca94] R. G. G. Cattell (edt), "The Object Database Standard: ODMG-93, Release 1.1", *Morgan Kaufmann Publishers, Inc., 1994*

[Ca95] R. G. G. Cattell (edt), "The Object Database Standard: ODMG-93, Release 1.2", *Morgan Kaufmann Publishers, Inc., 1995*

[CK86] G. Copeland, S. Khoshafian, "Identity and Versions for Complex Objects", "1986 International Workshop on Object-Oriented Database Systems", September 1986, Pacific Grove, California

[Di93] K. Dittrich, "Object-Oriented Data Model Concepts", NATA ASI OODBS 1993

[Kh93] S. Khoshafian, "Object-Oriented Databases", *John Willey and Sons, 1993*

[Ki91] M. Kifer, "A First Order Formalization of Object-Oriented Languages", *IEEE Data Engineering Bulletin, June 1991, Vol. 14, No. 2.*

[Ki94] W. Kim, “Observations on the ODMG-93 Proposal for an Object-Oriented Database Language”, *SIGMOD RECORD, Vol 23, No. 3, March 1994*

[KMN97] K. Kulkarni, N. Mattos, A. K. Nori, “Object-Relational Database Systems - Principles, Products and Challenges” Tutorials of 23rd Int’l VLDB Conference, Athens, Greece, 1997

[Mai89] D. Maier, "Why Isn't There an Object-Oriented Data Model?", *Technical Report CS/E-89-002, 2 May 1989, Computer Science and Enginnering, Oregon Graduate Center.*

[MM96] J. Melton, N. Mattos, “An overview of SQL3 - the Emerging New Generation of the SQL Standard”, Tutorials of 22nd Int’l VLDB Conference, Mumbai, India, 1996

[ODMG94] Object Database Management Group, "Response to the ODMG-93 Commentary Written by Dr. Won Kim of UniSQL, Inc.", *SIGMOD RECORD, Vol 23, No. 3, September 1994*

[OMG97] http://www.omg.org/corba/corbiiop.htm