ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Σύστημα κράτησης αεροπορικών θέσεων (ARS) με βάση δεδομένων XML

Σπουδαστής:
Νίκος Καλλιανός

Επιβλέπων:
Dr. Νίκος Παπαδάκης

Μια πτυχιακή για την εκπλήρωση απαιτήσεων πτυχίου

στο

Research Group Name
Τμήμα Μηχανικών Πληροφορικής

25 Σεπτεμβρίου 2016

Declaration of Authorship

Ο Νίκος Καλλιανός, δηλώνω οτι η πτυχιακή με τίτλο, "Σύστημα κράτησης αεροπορικών θέσεων (ARS) με βάση δεδομένων XML" και η δουλεία που παρουσιάζεται σε αυτή ειναι δική μου. Βεβαιώνω:

- Η εργασία εκπονήθηκε για τη λήψη πτυχίου.
- Οπου υπάρχει κέιμενο που έχει δημοσιευτεί και πάρθηκε απο άλλο Ιδρυμα ή Πανεπιστήμιο, εχει αναφερθεί στη βιβλιογραφία.
- Όπου έχω αναφόρες με την δημοσιευμένη εργασία των άλλων, αυτή είναι με σαφή αποδοση .
- Όπου έχω εισηγμένες από την εργασία των άλλων ,η πηγή είναι πάντα δεδομένη . Με την εξαίρεση των εν λόγω τιμών , αυτή η πτυχιακή είναι εντελώς δική μου δουλειά .
- Έχω αναγνωρίσει όλες τις κύριες πηγές βοήθειας .
- Όταν η εργασία βασίζεται στο έργο που επιτέλεσα ο ίδιος από κοινού με άλλους, έχω καταστήσει σαφές τι ακριβώς έγινε από τους άλλους και που έχω συμβάλλει ο ίδιος.

Signed: _____

Date: _____

"*Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.*"

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Περίληψη

Faculty Name
Τμήμα Μηχανικών Πληροφορικής

πτυχίου

Σύστημα κράτησης αεροπορικών θέσεων (ARS) με βάση δεδομένων XML

by Νίκος Καλλιανός

This thesis has been designed in C# .NET technology and consists of a XML backend which acts as the database instead of traditional entity relationship model. The objective of the project is to design an offline computerized Reservation System GUI application which enables the customers to search and book flights. In addition, provides functions such as flight schedule information, seat availability, refund and cancellation of tickets, as well as administration panel for maintaining flight data.

Acknowledgements

I would like to thank my supervisor Dr. Nikos Papadakis who has encouraged and inspired my dissertation topic as well as for his help whenever I needed.
Last but not least I owe an enormous debt of gratitude to my parents Konstantinos and Ourania and to my siblings Alexandros and Mantalena.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Στο κεφάλαιο αυτό γίνεται περιγραφή σε εισαγωγικες έννοιες της αποθήκευσης των δεδομένων και των βάσεων δεδομένων

1.1 Εισαγωγη

Στην 21ο αιώνα τα πάντα έχουν περάσει στην ψηφιακή μορφή τους. Βιβλιοθήκες,ηλεκτρονικά εισητήρια,ηλεκτρονικές συναλλαγές Η ευρεία διαδοση και χρηση της κυρίως της επιστήμης της Πληροφορικής αλλα και των επικοινωνιών, έχουν καταστήσει την πληροφορία ως ένα από τα βασικότερα και πολυτιμότερα αγαθά. Είναι κοινός τόπος σήμερα η εκτίμηση ότι το αγαθό της πληροφορίας είναι επιθυμητό στους εργαζόμενους αλλά και τους εκπαιδευόμενους, ώστε να είναι πιο αποδοτικοί, ανταγωνιστικοί και παραγωγικοί στην εργασία τους. Αυτό σημαίνει οτι γίνεται αποδοτικότερη λειτουργία και προγραμματισμός σε κάθε τύπου διαχείρηση. Έτσι, οι χρήστες προσδοκούν πολλά οφέλη από τη χρήση ενός ορθά πληροφοριακού συστήματος οπως βελτίωση της παραγωγικής διαδικασίας, ταχύτερη εξύπηρετηση ενός πελάτη καθως και ορθολογικές αποφάσεις.

Αυτο επιτυγχάνεται μέσα απο τα λεγόμενα DataBase Management Systems (DBMS). Τα συστήματα βάσεων δεδομένων χρησιμοποιούνται για να μπορούμε να αποθηκεύσουμε, να επεξεργαστούμε αλλά και να εκμεταλλευτούμε αποδοτικά τον μεγάλο όγκο των πληροφοριας που αυξάνεται με

ραγδαίους ρυθμούς καθημερινά.
Στην παρούσα πτυχιακή θα παρουσιαστεί και θα αναλυθεί μια βάση δεδομένων όπου θα τηρούνται πληροφορίες για σύστημα κράτησης θέσεων σε μια αεροπορική εταιρεία.

Tip

Ο υπολογιστής για την επεξεργασία και ανακληση δεδομένων μπορει να εκτελεί εως και 16 πεντάκις εκατομμύρια πράξεις κινητής υποδιαστολής το δευτερόλεπτο.

1.2 Αποθήκευση Δεδομένων

1.2.1 Αρχεία

Ο πιο γνωστός τρόπος οργάνωσης δεδομένων με τη χρήση ηλεκτρονικών υπολογιστών είναι σε αρχεία. Ένα αρχείο μπορούμε να το χαρακτηρίσουμε σαν ένα σύνολο που αποτελείται από οργανωμένα ομοειδή στοιχεία. Ένα αρχείο αποτελείται από εγγραφές (records) και πεδία (fields).

1.2.2 Εγγραφές

Οι εγγραφές χαρακτηρίζονται από:
Μήκος της εγγραφής
Δομή της εγγραφής
Διάβασμα (δεδομένα:σκληρό δίσκο ⊠ κεντρική μνήμη)
Γράψιμο (δεδομένα:κεντρική μνήμη ⊠ σκληρό δίσκο)

1.2.3 Πεδία

Ένα πεδίο χαρακτηρίζεται από το είδος των δεδομένων που περιέχει:
Αλφαριθμητικό
Αριθμητικό
Αλφαβητικό
Ημερομηνίας
Δυαδικό
Λογικό
Σημειώσεων

1.2.4 Προβλήματα Οργάνωσης Αρχείων

Στα αρχικά στάδια της οργάνωσης αρχείων, ήταν πολύ συνηθισμένη πρακτική η δημιουργία ξεχωριστών εφαρμογών (προγραμμάτων) και ξεχωριστών αρχείων, όπως για παράδειγμα η δημιουργία ενός αρχείου πελατών και ενός άλλου ανεξάρτητου αρχείου για τις παραγγελίες των πελατών. Τα προβλήματα που προέκυψαν από την πρακτική αυτή είναι τα εξής:

- **Πλεονασμός των δεδομένων (data redundancy)** Υπάρχει η περίπτωση να έχουμε επανάληψη των ίδιων δεδομένων σε αρχεία διαφορετικών εφαρμογών. Για παράδειγμα, αν έχουμε ένα αρχείο πελατών και ένα αρχείο παραγγελιών αυτών των πελατών, είναι σχεδόν σίγουρο ότι θα υπάρχουν κάποια στοιχεία των πελατών που θα υπάρχουν και στα δύο αρχεία.

- **Ασυνέπεια των δεδομένων (data inconsistency)** Αυτό μπορεί να συμβεί όταν υπάρχουν τα ίδια στοιχεία των πελατών (πλεονασμός) και στο αρχείο πελατών και στο αρχείο παραγγελιών και χρειασθεί να γίνει κάποια αλλαγή στη διεύθυνση ή στα τηλέφωνα κάποιου πελάτη, οπότε είναι πολύ πιθανό να γίνει η διόρθωση μόνο στο ένα αρχείο και όχι και στο άλλο.

- **Αδυναμία μερισμού δεδομένων (data sharing)**Μερισμός δεδομένων σημαίνει δυνατότητα για κοινή χρήση των στοιχείων κάποιων αρχείων. Για παράδειγμα, ο μερισμός δεδομένων θα ήταν χρήσιμος αν με την παραγγελία ενός πελάτη μπορούμε να έχουμε πρόσβαση την ίδια στιγμή στο αρχείο πελατών για να δούμε το υπόλοιπο του πελάτη και μετά στο αρχείο της αποθήκης για να δούμε αν είναι διαθέσιμα τα προϊόντα που παρήγγειλε ο συγκεκριμένος πελάτης. Η αδυναμία μερισμού δεδομένων δημιουργεί καθυστέρηση στη λήψη αποφάσεων και στην εξυπηρέτηση των χρηστών

- **Αδυναμία προτυποποίησης**Έχει να κάνει με την ανομοιομορφία και με την διαφορετική αναπαράσταση και οργάνωση των δεδομένων στα αρχεία των εφαρμογών. Η αδυναμία αυτή δημιουργεί προβλήματα προσαρμογής των χρηστών καθώς και προβλήματα στην ανταλλαγή δεδομένων μεταξύ διαφορετικών συστημάτων.

1.2.5 Δεδομένα και Πληροφορίες

Η σημασία του όρου της Πληροφορίας ποικίλλει. Ο όρος αυτός χρησιμοποιείται σε διάφορες επιστήμες αλλα και ως έκφραση στην καθημερινή ζωή, αφου ο άνθρωπος συνεχώς έρχεται αντιμέτωπος με ειδήσεις, γεγονότα ,νέα και ιδέες. Ολα τα παραπάνω συνήθως απαιτούν μια οργάνωση

ειδικά όσο μεγαλώνει ο όγκος των δεδομένων. Για τον λόγο αυτό χρησιμοποιούνται οι βάσεις δεδομένων. Η επεξεργασία με τη χρήση των βάσεων δεδομένων υπήρξε πάντα ένα σημαντικό αντικείμενο στον τομέα των πληροφοριακών συστημάτων.
Με τον όρο πληροφορία αναφερόμαστε συνήθως σε ειδήσεις, γεγονότα και έννοιες που αποκτάμε από την καθημερινή μας επικοινωνία και τα θεωρούμε ως αποκτηθείσα γνώση, ενώ τα δεδομένα μπορούν να είναι μη κατάλληλα επεξεργασμένα και μη ταξινομημένα σύνολα πληροφοριών. Ένας ορισμός για το τι είναι δεδομένα και το τι είναι πληροφορία είναι ο εξής:

- **Δεδομένα (data)** είναι ενα μη αξιολογημένο σύνολο όπως γράμματα, αριθμοί, σύμβολα κ.ά. στα οποία μπορούμε να αποδώσουμε κάποια σημασία. Προέρχεται απο το πλυθηντικο του datum δηλαδη ένα μονό(single)κομμάτι της πληροφοριας. Ο όρος των δεδομένων συχνα διαχωρίζει την δυαδική πληροφορία αναγνώσιμη απο τους υπολογιστές, απο την ανθρώπινα αναγνώσιμη σε κείμενο. Για παράδειγμα κάποιες εφαρμογές κάνουν διάκριση μεταξυ data αρχείων (που περιέχουν binary data) και μεταξύ text αρχείων (που περιέχουν ASCII χαρακτηρες).
  Στα Συστήματα Βασεων Δεδομένων τα δεδομένα είναι τα αρχεία που αποθηκευουν την πληροφορία της βάσης δεδομένων, ενώ τα υπόλοιπα αρχεία όπως τα αρχεία ευρετηρίου (indexing) καθως και λεξικά δεδομένων αποθηκεύουν πληροφοριες διαχείρησης όπως τα λεγόμενα metadata.

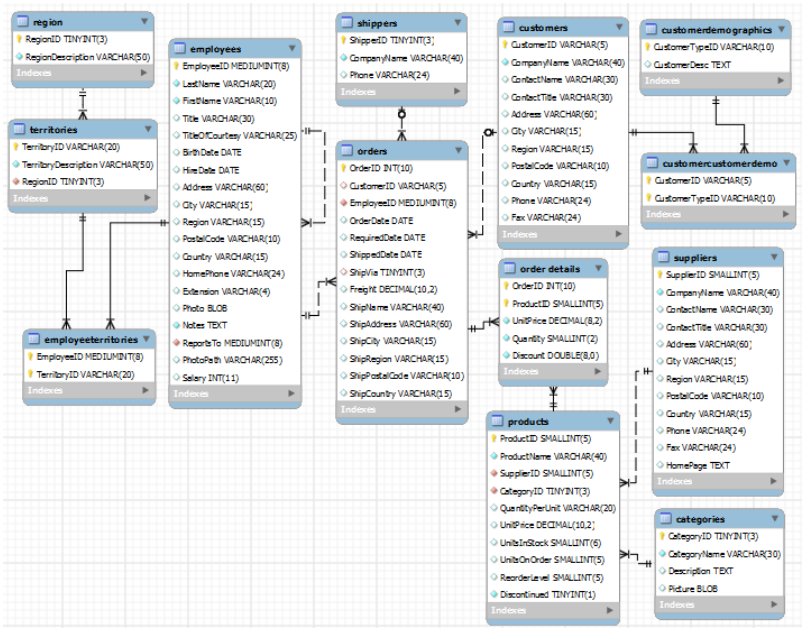
- **Πληροφορία (information)**. Η πληροφορία δεν είναι δυνατό να υφίσταται απο μόνη της. Ενα στοιχείο που χρειάζεται είναι τον "μεσον" με το οποίο θα γίνει εν τελει γνωστή. Το μέσο αυτό φυσικά δεν είναι αλλο απο τα δεδομένα τα οποία οταν επεξεργάζονται αποδίδουν την πληροφορία. Μεσω της τελευταίας βάσει προπαρασκευασμενης γνώσης μπορούμε να εξάγουμε τα συμπεράσματα μας.

  Tip

  Τα δεδομένα για να μας δίνουν πληροφορία πρέπει να τα συσχετίσουμε.

1.3 Βάσεις δεδομένων

Ως Συστήματα βάσης δεδομένων νοούνται ως ένα ηλεκτρονικό σύστημα τήρησης εγγραφών, δηλαδή, ένα σύστημα για υπολογιστές, που ο γενικός σκοπός του είναι να τηρεί πληροφορίες με συστηματκό τρόπο και να δίνει αυτές τις πληροφορίες όταν του ζητούνται. Οι πληροφορίες που τηρούνται σε ένα τέτοιο σύστημα μπορεί να είναι οτιδήποτε έχει σημασία για το άτομο ή τον οργανισμό που εξυπηρετεί το συγκεκριμένο σύστημα, με άλλα λόγια οτιδήποτε χρειάζεται για την υποβοήθηση των εργασιών



Σχήμα 1.1: Βάση Δεδομένων

αυτού του ατόμου ή οργανισμού.
Η ίδια η βάση δεδομένων μπορεί να θεωρηθεί ένα είδος ηλεκτρονικής αρχειοθήκης, ένα χώρος για την αποθήκευση μιας συλλογής ηλεκτρονικών αρχείων δεδομένων.
Ο χρήστης του συστήματος έχει στη διάθεση του ορισμένα βοηθήματα για να εκτελεί στα αρχεία Β.Δ. διάφορες εργασίες, στις οποίες συγκαταλέγονται, μεταξύ άλλων, και οι εξής:

- Η προσθήκη νέων κενών αρχείων στη βάση δεδομένων.
- Η εισαγωγή νέων δεδομένων στα υπάρχοντα αρχεία.
- Η ανάκληση δεδομένων από υπάρχοντα αρχεία.
- Η ενημέρωση δεδομένων σε υπάρχοντα αρχεία.
- Η διαγραφή δεδομένων από υπάρχοντα αρχεία.
- Η αφαίρεση υπαρχόντων αρχείων, κενών ή όχι, από τη βάση δεδομένων.

Ένας τηλεφωνικός κατάλογος, για παράδειγμα, θεωρείται βάση δεδομένων, καθώς αποθηκεύει και οργανώνει σχετιζόμενα τμήματα πληροφορίας, όπως είναι το όνομα και ο αριθμός τηλεφώνου. Ένα δεύτερο παράδειγμα είναι οι πίνακες δημογραφικών/στατιστικών στοιχείων. Ωστόσο οι σύγχρονες βάσεις δεδομένων υλοποιούνται κυρίως ψηφιακά, με τυποποιημένες μεθόδους σε ηλεκτρονικούς υπολογιστές, ελέγχονται από λογισμικό DBMS και συνιστούν αντικείμενο επιστημονικής και τεχνικής μελέτης από το αντίστοιχο ακαδημαϊκό πεδίο.

Τα πλεονεκτήματα ενός συστήματος βάσης δεδομένων, σε σύγκριση με τις παραδοσιακές μεθόδους συντήρησης (ντοσιέ χαρτί και μολύβι), είναι τα ακόλουθα:

- **Οικονομία** και βέλτιστη οργάνωση. Δεν υπάρχει η ογκώδης στοιχειοθεσία φακέλων.
- **Ευκολότερη** και με ακρίβεια εισαγωγή δεδομένων.
- **Πραγματικός έλεγχος** των δεδομένων σε ταξινόμηση, ανάλυση και σύνοψη των δεδομένων
- **Επαναχρησιμοποίηση** ωστε να μη χρειάζεται να εισαχθούν ξανά τα αρχεία.
- **Ταχύτητα**.Το σύστημα μπορεί να κάνει ανάκληση και αλλαγή δεδομένων ταχύτερα από τον άνθρωπο)
- **Άμεση πληροφόρηση** (Ενημερωμένες πληροφορίες είναι διαθέσιμες ανα πάσα στιγμή).

1.4 Μοντέλα αποθήκευσης δεδομένων

Ένα μοντέλο δεδομένων είναι ενα αφηρημένο μοντέλο που οργανώνει τα στοιχεία των δεδομένων και τυποποιεί(standarization) πώς σχετίζονται μεταξύ τους και με τις ιδιότητες του πραγματικού κόσμου. Για παράδειγμα, ένα μοντέλο δεδομένων μπορεί να προσδιορίσει ότι ένα στοιχείο δεδομένων που αντιπροσωπεύει ένα αυτοκίνητο περιλαμβάνουν μια σειρά από άλλα στοιχεία τα οποία με τη σειρά τους αντιπροσωπεύουν το χρώμα, το μέγεθος και τον ιδιοκτήτη του αυτοκινήτου.
Ένα μοντέλο δεδομένων καθορίζει ρητά τη δομή των δεδομένων. Τα μοντέλα δεδομένων καθορίζονται σε μια σημειογραφία μοντελοποίησης δεδομένων, η οποία είναι συχνά γραφική μορφή.

Ένα μοντέλο δεδομένων μπορεί να μερικές φορές αναφέρεται ως μια δομή δεδομένων , ειδικά στο πλαίσιο των γλωσσών προγραμματισμού. Τα μοντέλα δεδομένων συχνά συμπληρώνονται από τα μοντέλα λειτουργίας , ιδίως στο πλαίσιο των μοντέλων της επιχείρησης.

Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

Σχήμα 1.2: Το επίπεδο μοντέλο

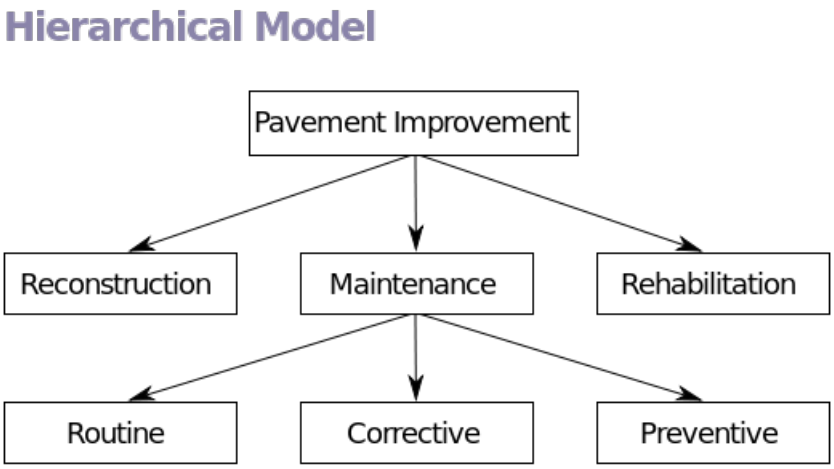
1.4.1 Μοντέλο Επίπεδης Αποθήκευσης

Αυτή η μορφή πίνακα αποτελεί τον πρόδρομο για το σχεσιακό μοντέλο.
Το συγκεκριμένο δεν μπορεί να χαρακτηριστεί απολύτως ως ένα μοντέλο δεδομένων. Το επίπεδο (Flat ή πίνακα) μοντέλο αποτελείται από μια μόνο δισδιάστατη διάταξη στοιχείων, όπου όλα τα μέλη μιας δεδομένης στήλης υποτίθεται ότι είναι παρόμοιες τιμές, και όλα τα μέλη μιας σειράς θεωρείται ότι σχετίζονται μεταξύ τους.
Σε έναν πίνακα 2x2 κάθε σειρά θα έχει το ειδικό κωδικό που σχετίζονται με έναν μεμονωμένο χρήστη. Στήλες του πίνακα έχουν συχνά έναν τύπο που συνδέονται με αυτά, τον ορισμό τους ως στοιχεία του χαρακτήρα, την ημερομηνία ή την ώρα πληροφοριών, ακέραιοι, ή αριθμούς κινητής υποδιαστολής.

1.4.2 Ιεραρχικό Μοντέλο

Σε αυτό το μοντέλο δεδομένων η διάταξη είναι οργανωμένη με μια δέντροειδής δομή, κατι που συνεπάγεται μια ενιαία ανοδική σύνδεση σε κάθε εγγραφή για να περιγράψει το μονοπατι , και ένα πεδίο ταξινόμησης για να κρατήσει τα αρχεία σε μια συγκεκριμένη σειρά σε κάθε λίστα ίδιο επίπεδο.
Μια ιδέα είναι να σκεφτούμε τις διατάξεις σε χαμηλότερο επίπεδο (μικρότερες και πιο πολυπλυθείς),όπως οργανώνονται, σε μια ιεραρχία από διαδοχικές μονάδες υψηλότερου επιπέδου. Για παράδειγμα, οι μαθητές είναι σε τάξεις, οι τάξεις είναι στα σχολεία, τα σχολεία βρίσκονται σε σχολικές περιοχές, σχολικές περιοχές βρίσκονται σε κράτη. Στη συνέχεια μπορούμε να περιγράψουμε τα αποτελέσματα για κάθε μαθητή ως ένα άθροισμα των αποτελεσμάτων για κάθε φοιτητή, για την τάξη του, για το σχολείο, για την περιοχή και για το κράτος.
Στο ιεραρχικό μοντέλο βάσης δεδομένων καθε εγγραφή-παιδί έχει μόνο ένα γονέα, ενώ κάθε εγγραφή-γονέα μπορεί να έχει ένα ή περισσότερα παιδία. Για να ανακτηθούν τα δεδομένα από μια ιεραρχική βάση δεδομένων πρέπει να γίνει διέλευση ξεκινώντας από τον κόμβο ρίζα ολόκληρο

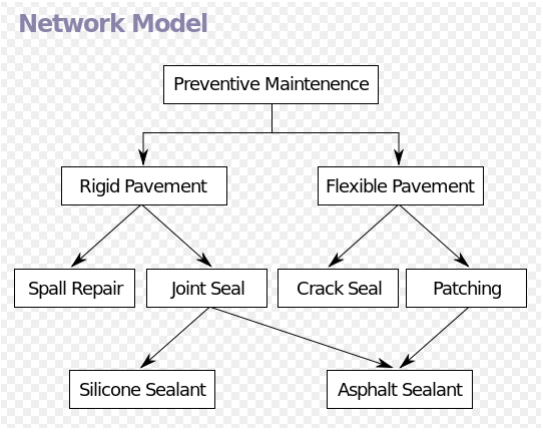
το δέντρο. Το μοντέλο αυτό έχει αναγνωριστεί ως το πρώτο μοντέλο βάση δεδομένων που δημιουργήθηκε από την IBM το 1960



Σχήμα 1.3: Το ιεραρχικό μοντέλο

1.4.3 Δικτυωτό μοντέλο

Το δικτυωτό μοντέλο δίκτυο αποτελεί επεκταση της ιεραρχικής δομής,ενώ ηταν το πιο δημοφιλές πριν αντικατασταθεί από το σχεσιακό μοντέλο. Επιτρέπει N:N συσχετισεις σε μια δέντροειδή δομή η οποία επιτρέπει με τη σειρά της πολλαπλά δέντρα-γονείς. Το μοντέλο δίκτυου οργανώνει τα δεδομένα χρησιμοποιώντας δύο θεμελιώδεις έννοιες, που ονομάζονται αρχεία και σύνολα. Οι εγγραφές περιέχουν πεδία (τα οποία μπορούν να οργανωθούν ιεραρχικά, όπως στη γλώσσα προγραμματισμού COBOL). Τα σετ ορίζουν τις 1:N σχέσεις μεταξύ των εγγραφών: ένας ”ιδιοκτήτης” - πολλά ”μέλη”. Ένα αρχείο μπορεί να είναι ιδιοκτήτης σε οποιοδήποτε αριθμό των συνόλων, και ένα μέλος σε οποιοδήποτε αριθμό των συνόλων.



Σχήμα 1.4: Το δικτυωτό μοντέλο

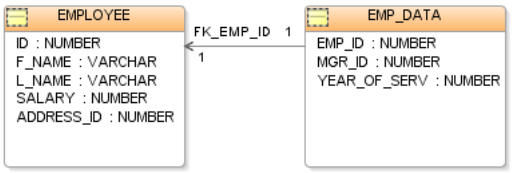
Ένα σετ αποτελείται από κυκλικές συνδεδεμένες λίστες (βλ. δομές δεδομένων), όπου μια εγγραφή, το σύνολο "ιδιοκτήτης" ή ο γονέας, εμφανίζεται μία φορά σε κάθε κύκλο, και ένα δεύτερο τύπο εγγραφής, τα υπόλοιπα παιδιά της ρίζας ή το παιδί, μπορεί να εμφανιστεί πολλές φορές σε κάθε κύκλο. Με αυτόν τον τρόπο μια ιεραρχία μπορεί να δημιουργηθεί μεταξύ οποιωνδήποτε δύο τύπων εγγραφών, π.χ., τύπου Α είναι ο ιδιοκτήτης του Β. Ταυτόχρονα, ενα άλλο σετ μπορεί να οριστεί όπου το Β είναι ο ιδιοκτήτης του Α. Έτσι, όλα τα σετ αποτελούνται απο ενα ολοκληρωμένο κατευθυνόμενο γραφήμα (ο διαχειριστης της βασης καθορίζει μια κατεύθυνση μέσα στο γράφο). Η πρόσβαση στις εγγραφές είναι είτε διαδοχική (συνήθως σε κάθε τύπο εγγραφής) ή με την μετακίνηση στις κυκλικά συνδεδεμένες λίστες.

1.4.4 Αντεστραμμένο Ευρετήριο

Σε ένα ανεστραμμένο αρχείο ή αντεστραμμένο ευρετήριο, τα περιεχόμενα των δεδομένων που χρησιμοποιούνται ως κλειδιά σε έναν lookup table (πίνακα αναζήτησης), και οι τιμές του πίνακα είναι δείκτες για τη θέση ένός δεδομένου στοιχείου περιεχομένου. Πρόκειται για την ίδια λογική δομή που έχουν και τα σύγχρονα ευρετήρια των βάσης δεδομένων, η οποία θα μπορούσε να χρησιμοποιήσει μόνο το περιεχόμενο από ένα συγκεκριμένες στήλες στον πίνακα αναζήτησης.
Το ανεστραμμένο μοντέλο δεδομένων του αρχείου μπορεί να τοποθετήσει ευρετήρια σε ένα δεύτερο σύνολο αρχείων δίπλα στο υπάρχον επίπεδο αρχείο της βάσης δεδομένων, προκειμένου να έχουν πρόσβαση σε εγγραφές σε αυτά τα αρχεία.
Η εγγραφοστρεφής (document oriented) βάση δεδομένων που ονομάζεται Clusterpoint, χρησιμοποιεί ανεστραμμένο μοντέλο μεθόδου ευρετηρίου για τη γρήγορη αναζήτηση για XML ή JSON αντικείμενα. Η εγγραφοστρεφής βάση Clusterpoint, έχει τη δυνατότητα συνδυασμένης εκτέλεσης ερωτήματος SQL, αναζήτησης, και εκτέλεσης κώδικα JavaScript στο εσωτερικό της κατανεμημένης βάσης δεδομένων. Τα δεδομένα και το ανεστραμμένο ευρετήριο μπορεί να διανεμηθεί σε ένα μεγάλο αριθμό απο servers ώστε να παρέχει δισεκατομμύρια αντικειμένα (objects) δεδομένων στην ίδια βάση δεδομένων Clusterpoint. Ομοίως με τις σχεσιακές βάσεις δεδομένων, η βάση Clusterpoint, υποστηρίζει κατανεμημένες συναλλαγές ACID -απο τα αρχικά Ατομικότητα, Συνέπεια, απομόνωση-Isolation, μονιμότητα- βάση δεδομένων για την συνέπεια της βάσης δεδομένων του εγγράφου, όπου το ανεστραμμένο τα στοιχεία του ευρετηρίου αμέσως ενημερώνεται διαμέσου XML ή JSON ενημερώσεων περιεχομένου του εγγράφου. Η μέθοδος αυτή χρησιμοποιείται για να υποστηρίξει ογκώδη πραγματικού χρόνου δεδομένα, analytics(όπως τα γνωστά google analytics) καθώς και για data mining.

1.4.5 Σχεσιακό Μοντέλο

Το σχεσιακό μοντέλο πρωτοεμφανιστηκε από τον E.P. Codd το 1970 ούτως ώστε τα συστήματα διαχείρισης βάσεων δεδομένων να γίνουν ανεξάρτητα από τρίτες εφαρμογές. Πρόκειται για ένα μαθηματικό μοντέλο που ορίζεται από την άποψη της λογικής του "κατηγορήματος" (γνωρισμα) και τη θεωρία των συνόλων, ενώ χρησιμοποείται σε όλες τις κατηγορίες υπολογιστικών συστημάτων. Δηλαδη servers, mainframes, middleframe, και βεβάιως τα γνωστά μας PC (microcomputers). Αυτά που αναφέρονται γενικά ως σχεσιακές βάσεις δεδομένων στην πραγματικότητα εφαρμόζουν ένα μοντέλο που είναι μόνο μια προσέγγιση με το μαθηματικό μοντέλο του Codd.
Τρεις είναι οι βασικοί όροι που χρησιμοποιούνται εκτενώς στο μοντέλο της σχεσιακής βάσης δεδομένων: Σχέσεις, Ιδιότητες, και οι Τομείς. Μια Σχέση είναι ένας πίνακας με στήλες και γραμμές. Οι ονομάζομενες στήλες της σχέσης που ονομάζεται ιδιότητες, και είναι το σύνολο τιμών είναι τα χαρακτηριστικά που επιτρέπεται να λαμβάνουν.
Η βασική δομή δεδομένων του σχεσιακού μοντέλου είναι ο πίνακας, όπου οι πληροφορίες σχετικά με μια συγκεκριμένη οντότητα (για παράδειγμα, ένας εργαζόμενος) τα στοιχεία που τον απαρτίζουν αντιπροσωπεύονται σε γραμμές, (ονομάζονται επίσης πλειάδες) και στήλες. Έτσι, η "σχέση" σε μια "σχεσιακή βάση δεδομένων" αναφέρεται στους διάφορους πίνακες στη βάση δεδομένων δηλαδή μια σχέση είναι ένα σύνολο από πλειάδες.



Σχήμα 1.5: Πίνακας εργαζόμενος στο σχεσιακό μοντέλο

Οι στήλες απαριθμήσει τα διάφορα χαρακτηριστικά της οντότητας (το όνομα του εργαζομένου, τη διεύθυνση ή τον αριθμό τηλεφώνου, για παράδειγμα) και μια σειρά είναι χαρακτηριστικά που αποτυπόνωνται κ στο πραγματικο κόσμου (ένα συγκεκριμένος εργαζόμενος θα έχει πάντα αυτά τα χαρακτηριστικά) που αντιπροσωπεύεται από τη σχέση. Έτσι, κάθε πλειάδα του πίνακα των εργαζομένων αντιπροσωπεύει διάφορα χαρακτηριστικά ενός μεμονωμένου υπαλλήλου.
Όλες οι σχέσεις (και ως εκ τούτου οι πίνακες) σε μια σχεσιακή βάση δεδομένων, πρέπει να τηρούν ορισμένους βασικούς κανόνες για να χαρακτηριστούν ως τέτοιες. Πρώτον, η διάταξη των στηλών δεν παίζει ρόλο σε έναν πίνακα. Δεύτερον, δεν μπορούν να υπάρχουν πανομοιότυπες πλειάδες (δήλ. σειρές) σε ένα πίνακα. Και τρίτον, κάθε πλειάδα θα περιέχει μία μόνο τιμή για κάθε ένα από τα γνωρίσματά της.
Μια σχεσιακή βάση δεδομένων περιέχει πολλούς πίνακες, παρόμοιους με εκείνους του επίπεδου μοντέλο βάσης δεδομένων. Ένα από τα δυνατά σημεία του σχεσιακού μοντέλου είναι ότι, κατ'αρχήν, σε οποιαδήποτε τιμή

σε δύο διαφορετικές εγγραφές (που ανήκουν στον ίδιο πίνακα ή σε διαφορετικούς πίνακες), υπονοεί μια σχέση μεταξύ αυτών των δύο εγγραφών. Ωστόσο, προκειμένου να επιβάλλει σαφείς περιορισμούς ακεραιότητας, οι σχέσεις στις μεταξύ εγγραφές μέσα στους πίνακες μπορεί επίσης να οριστεί ρητά, με τον προσδιορισμό ή μη εντοπισμό γονέα-παιδιού σχέσεις χαρακτηρίζονται από την ανάθεση (1: 1, 1: N, N: N). Οι πίνακες μπορούν επίσης να έχουν ένα και μοναδικό καθορισμένο χαρακτηριστικό ή ένα σύνολο χαρακτηριστικών που μπορεί να λειτουργήσει ως "κλειδί", το οποίο μπορεί να χρησιμοποιηθεί για να προσδιορίσει μοναδικά κάθε πλειάδα του πίνακα. Ένα κλειδί που μπορεί να χρησιμοποιηθεί για να προσδιορίσει επακριβώς μια σειρά σε ένα πίνακα ονομάζεται πρωτεύον κλειδί. Τα κλειδιά χρησιμοποιούνται συνήθως για να ενωθούν ή να συνδυαστούν στοιχεία από δύο ή περισσότερους πίνακες. Για παράδειγμα, ένας πίνακας Υπάλληλος μπορεί να περιέχει μια στήλη που ονομάζεται Θέση η οποία περιέχει μια τιμή που να ταιριάζει με το κλειδί του πίνακα Τοποθεσία. Τα πρωτεύοντα κλειδιά είναι επίσης κρίσιμα για τη δημιουργία ευρετηρίων, που διευκολύνουν την γρήγορη ανάκτηση των δεδομένων από μεγάλους πίνακες. Κάθε στήλη μπορεί να είναι ένα κλειδί, ή πολλαπλές στήλες μπορούν να ομαδοποιηθούν σε ενα συνδυασμένο κλειδί. Δεν είναι αναγκαίο να καθοριστούν όλα τα κλειδιά των προτέρων. Μια στήλη μπορεί να χρησιμοποιηθεί ως ένα βασικό κλειδί ακόμη και αν αρχικά δεν προορίζεται να είναι ως το βασικό.

Ένα κλειδί που έχει ένα νόημα του, πραγματικού κόσμου που σημαίνει - όπως το όνομα ενός ατόμου, το ISBN ενός βιβλίου, ή το σειριακό αριθμό του αυτοκινήτου- μερικές φορές ονομάζεται και "φυσικό"(natural) κλειδί. Εάν το φυσικό κλειδί δεν είναι κατάλληλο (σκεφτείτε ονόματα απο πολλούς ανθρώπους που έχουν όνομα Γιώργος), ένα αυθαίρετο ή υποκατάστατο κλειδί μπορεί να εκχωρηθεί (με παρόμοιο τρόπο όπως π.χ. το ID των εργαζομένων).

Στην πράξη, οι περισσότερες βάσεις δεδομένων έχουν και τα δύο ειδών παράγομενα και φυσικά κλειδία, γιατί τα παραγόμενα κλειδιά μπορούν να χρησιμοποιηθούν εσωτερικά για να δημιουργήσουν δεσμούς μεταξύ των πλειάδων που δεν μπορούν να "σπάσουν", ενώ τα φυσικά (natural) κλειδία, μπορούν να χρησιμοποιηθούν, λιγότερο αξιόπιστα, για αναζητήσεις και για την ενοποίηση με άλλες βάσεις δεδομένων. (Για παράδειγμα, οι εγγραφές σε δύο ανεξάρτητα ανεπτυγμένες βάσεις δεδομένων θα μπορούσαν να συνδυαστούν με τον αριθμό κοινωνικής ασφάλισης, εκτός εάν οι αριθμοί κοινωνικής ασφάλισης είναι λαθος, ελλιπείς, ή έχουν αλλάξει).

Η συνηθέστερη γλώσσα ερωτημάτων που χρησιμοποιείται με το σχεσιακό μοντέλο είναι το Structured Query Language (SQL). Αλλες παραλλαγές είναι οι SchemeQL, LSQL, ScalaQL και ScalaQuery, SqlStatement, ActiveRecord σε γλώσσα Ruby, HaskellDB, και βεβαίως η γλώσσα που θα δούμε σε αυτήν τη πτυχιακή η LINQ σε .Net.

1.4.6 Μετασχεσιακά Μοντέλα

Τα αποτελέσματα που προσφέρουν ένα πιο γενικό μοντέλο δεδομένων από το σχεσιακο, ορισμένες φορές ταξινομούνται ως μετα-σχεσιακά. Εναλλακτικός όρος είναι και ο «υβριδικές βάσεις δεδομένων", "Object-enhanced RDBMS" και άλλα. Το μοντέλο δεδομένων των εν λόγω πινάκων περιλαμβάνει τις σχέσεις, αλλά δεν περιορίζεται από την αρχή του Codd η οποία απαιτεί οι πληροφορίες στη βάση δεδομένων ρητά να καθορίζονται από την άποψη των τιμών και με κανέναν άλλον τρόπο.
Μερικές από αυτές τις επεκτάσεις στο σχεσιακό μοντέλο ενσωματώνουν έννοιες από τις τεχνολογίες πριν υπάρξει το σχεσιακό μοντέλο. Για παράδειγμα, επιτρέπουν αναπαράσταση ενός κατευθυνόμενου γράφηματος με τα δέντρα στους κόμβους για παραδειγμα στο GraphDB. Ορισμένα μετασχεσιακά προϊόντα παρατείνουν σχεσιακά συστημάτα με μη-σχεσιακά χαρακτηριστικά. Άλλοι έφτασαν στην πολύ στον ίδιο χώρο με την προσθήκη σχεσιακή χαρακτηριστικά να προ-σχεσιακών συστημάτων. Παραδόξως, αυτό επιτρέπει τα προϊόντα που είναι ιστορικά προ-σχεσιακές, όπως PICK και παρωτίτιδα, να κάνει μια εύλογη απαίτηση να είναι μετασχεσιακή.

1.4.7 Διάστατο Μοντέλο

Πρόκειται για μια εξειδικευμένη προσαρμογή του σχεσιακού μοντέλου που χρησιμοποιήθηκε για την αναπαράσταση των δεδομένων σε αποθήκες δεδομένων με τέτοιο τρόπο ώστε τα δεδομένα μπορούν να συνοψιστούν εύκολα χρησιμοποιώντας ηλεκτρονική αναλυτική επεξεργασία, ή OLAP ερωτήματα (Online Analytical processing-είναι η προσέγγιση απάντησης σε πολυερωτήματα με τρόπο ταχύ).
Στο τρισδιάστατο μοντέλο, ένα σχήμα βάσης δεδομένων αποτελείται από ένα μεγάλο πίνακα γεγονότων που περιγράφονται χρησιμοποιώντας τις διαστάσεις και τους κανόνες. Μια διάσταση παρέχει το πλαίσιο ενός γεγονότος (όπως ο οποίος συμμετείχε, πότε και πού, και το είδος του) και χρησιμοποιείται σε ερωτήματα που χρησιμοποιούνται για να ομαδοποιήσουν σχετιζόμενα γεγονότα.

Σχήμα 1.6: Το διάστατο μοντέλο

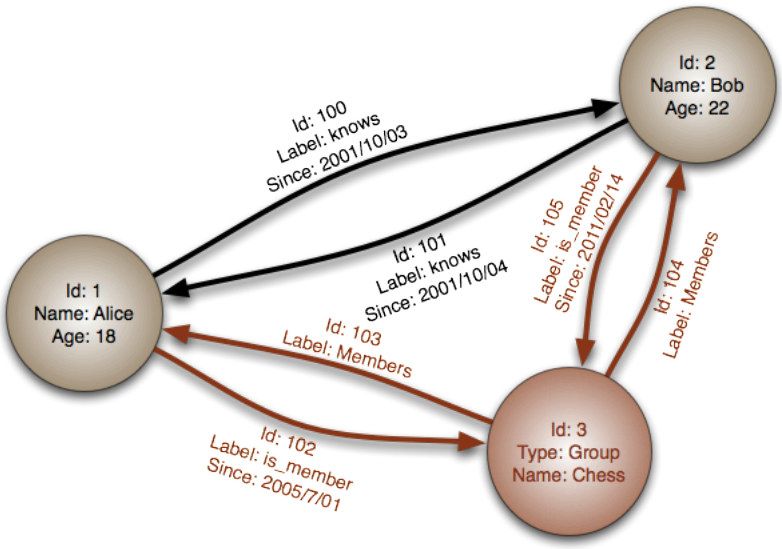
Οι διαστάσεις τείνουν να είναι διακριτές (discrete) και συχνά ιεραρχικές. Για παράδειγμα, η "θεση" θα μπορούσε να περιλαμβάνει το κτήριο, κράτος και χώρα. Ένα μέτρο είναι μια ποσότητα που περιγράφει τα γεγονότα όπως τα έσοδα. Είναι σημαντικό ότι τα μέτρα μπορούν να αθροιστούν -για παράδειγμα, τα έσοδα από διαφορετικές τοποθεσίες μπορούν να προστεθούν όλα μαζι.
Σε ένα OLAP ερωτημα οι διαστάσεις επιλέγονται και τα γεγονότα κατηγοροιοποιούνται και προστίθεται μαζί ούτως ώστε να δημιουργήσουν μια σούμα (summary).

Το διάστατο μοντέλο εφαρμόζεται συχνά στην κορυφή του σχεσιακού μοντέλου χρησιμοποιώντας ένα σχήμα αστέρα, που αποτελείται από ένα κανονικοποιημένο πίνακα που περιέχει τα πραγματικούς πίνακες και τους περιβάλλοντες αποκανονικοποιημένους πίνακες που περιέχουν κάθε διάσταση. Μια εναλλακτική φυσική εφαρμογή, που ονομάζεται snowflake schema(σχήμα νιφάδας), ομαλοποιεί πολυεπίπεδες ιεραρχίες εντός μιας διάστασης σε πολλαπλούς πίνακες.
Μια αποθήκη δεδομένων μπορεί να περιέχει σχήματα πολλαπλών διαστάσεων που μοιράζονται πίνακες διαφόρων διαστάσεων, ενω επιτρέπει στους πίνακες να χρησιμοποιούνται μαζί.Η υψηλή απόδοση του έχει καταστήσει το διάστατο μοντέλο ως την πιο δημοφιλή δομή της βάσεων δεδομένων για OLAP.

1.4.8 Μοντέλο Γράφου

Μια βάση δεδομένων μοντέλου γράφου είναι μια βάση που χρησιμοποιεί δομές γράφου για σημασιολογικά ερωτήματα ωστε η αποθήκευση των δεδομένων να γίνεται με κόμβους. Η βασική ιδέα του συστήματος είναι η γραφική αναπαράσταση (ή ακμή ή σχέση), η οποία σχετίζεται άμεσα τα στοιχεία δεδομένων στην αποθήκευση.
Αυτό αντιτίθεται με τις συμβατικές σχεσιακές βάσεις δεδομένων, όπου οι συνδέσεις μεταξύ των στοιχείων εφαρμόζουν τη λογική ad-hoc και βασίζονται στα ίδια τα δεδομένα, και τα σχετιζόμενα στοιχέια που συγκεντρώθηκαν από την αναζήτηση για αυτά τα δεδομένα εντός του αποθηκευτικού χώρου. Οι βάσεις δεδομένων γράφου σχεδιάστηκαν για να επιτρέπουν απλή και γρήγορη ανάκτηση των πολύπλοκων ιεραρχικών δομών, ενώ στον αντίποδα μια σχεσιακή βάση δεδομένων θα χρησιμοποιούσε ένα περίπλοκο ερώτημα για την επίτευξη του ίδιου στοχου, αλλα με πολύ μικρότερο performance.
Οι βάσεις γράφων είναι παρόμοιες με εκείνες που χρονολογούντα το 1970 ονοματι μοντέλο δίκτυου βάσεων δεδομένων, όπου και οι δύο έχουν σχεδιαστεί για να εκπροσωπούν γενικά γράφους, αλλά το μοντέλο δικτύου βάσεων δεδομένων λειτουργούν σε χαμηλότερο αφαιρετικό επίπεδο και δεν είναι εύκολη η διάσχιση πάνω στο γράφο.

Η ανάκτηση δεδομένων από μια βάση δεδομένων γράφου απαιτεί διαφορετικά κόνσεπτ και γενικότερα μια γλώσσα ερωτημάτων που δεν είναι κοινή. Μέχρι σήμερα, καμία άλλη ενιαία γλώσσα ερωτημάτων γράφου δεν έχει αυξηθεί στο προσκήνιο με τον ίδιο τρόπο όπως η SQL το κατάφερε στις σχεσιακές βάσεις δεδομένων. Κάποιες προσπάθειες τυποποίησης έχουν προηγηθεί οδηγώντας σε συστήματα όπως το Gremlin που λειτουργέι που λειτουργεί με μια ποικιλία με μηχανές γράφων καθώς και το SPARQL σύστημα.



Σχήμα 1.7: Μοντέλο Γράφου

1.4.9 Μοντέλο Πολλαπλών Τιμών

Όπως έχουμε συμπεράνει ως τώρα απο την ανάλυση των μοντελων βάσεων δεδομένων, η προφανής εντύπωση που μας έχει δωθεί ειναι ότι το σχεσιακό μοντέλο κυριαρχέι στην πλειονότητα των περιπτώσεων. Το μοντέλο αυτό, χρησιμοποιείται ευρέως στον ακαδημαϊκο χώρο, συχνα με τη διαπίστωση ότι είναι ο μόνος και μοναδικός τρόπος για να σχεδιάστει μια βάση δεδομένω.
Το μοντέλο πολλαπλών τιμών δημιουργεί τις προοπτικές για μια νέα προσέγγιση παράλληλα δείχνωντας και τα προτερήματα του. Με την στάθερη ανάπτυξη της γλώσσας ερωτημάτων NoSQL, αρχίζει και εντατικοποιείται ενω όλο και περισσότεροι το χρησιμοποιούν αντί του παραδοσιακού ER μοντέλου.

Ίσως ένας από τους λόγους για τους οποίους δεν είναι ευρύτερα γνωστό είναι ότι σπάνια κάνει ”ντόρο”, διότι εν γένει τα έργα βάσης δεδομένων πολλαπλών τιμών τείνουν να παραδοδίνονται πάντα στην ώρα τους και εντός του προϋπολογισμού. Οι προγραμματιστές που εργάζονται με αυτό το μοντέλο είναι συνήθως τόσο ενθουσιασμένοι που είναι εκπληκτικό το ότι έχει παραμείνει ανήκουστο από πολλούς επαγγελματίες της βάσης δεδομένων.

Ένα από τα βασικά χαρακτηριστικά του μοντέλου πολλαπλών τιμών είναι ότι επιτρέπει πληροφορίες που πρέπει να αποθηκεύονται στη βάση δεδομένων σε μια μορφή που σχετίζεται στενότερα με τις οντότητες του πραγματικού κόσμου, σε σύκριση με τον τρόπο που τα δεδομένα που αντιπροσωπεύονται σε μια σχεσιακή βάση δεδομένων.
Αυτό γίνεται λόγω των λιγότερο αυστηρών κανόνον σε αντιδιαστολή με το σχεσιακό μοντέλο που κάνει πιο πολύπλοκο το σχεδιασμό της βάσης δεδομένων, αυξάνει την πολυπλοκότητα εφαρμογής και συνεπώς αυξάνει

το κόστος ανάπτυξης. Σς γενικές γραμμές ότι μπορεί να γίνει σε μια σχεσιακή βάση δεδομένων, μπορεί να γίνει και σε πολλαπλών τιμών, αλλά η ικανότητα να αποθηκεύουν πολυδιάστατα δεδομένα καθιστά αυτό το μοντέλο τόσο ισχυρό.
Υπάρχουν πολλοί τρόποι για να αντιπροσωπεύτουν τα δεδομένα, αλλά η πιο κοινή είναι οι απλοι δισδιάστατοι πίνακες, που ο καθένας περιέχει δεδομένα ενός συγκεκριμένου τύπου. Παράδειγμα, ένα απλό σύστημα επεξεργασίας πωλήσεων θα μπορούσε να έχει τρεις πίνακες αποθήκευσης των δεδομένων αποθήκευσης (STOCK), τα στοιχεία των πελατών (CUSTOMERS) και οι παραγγελίες (SALES). Κάθε γραμμή του πίνακα αναφέρεται ως μια εγγραφή και το μοναδικό αναγνωριστικό για μια σειρά (σε αυτή την περίπτωση το part number) αναφέρεται ως το κλειδί ή αλλιως record id.

Part no	Description	Quantity	Price
003	Pen, blue	87	1.70
004	Pen, green	88	1.70
011	Pencil, black	171	0.30
013	Pencil, blue	13	0.30
051	Scissors, small	17	2.27
111	Desk lamp	3	17.50

Σχήμα 1.8: Το μοντέλο πολλαπλής τιμής

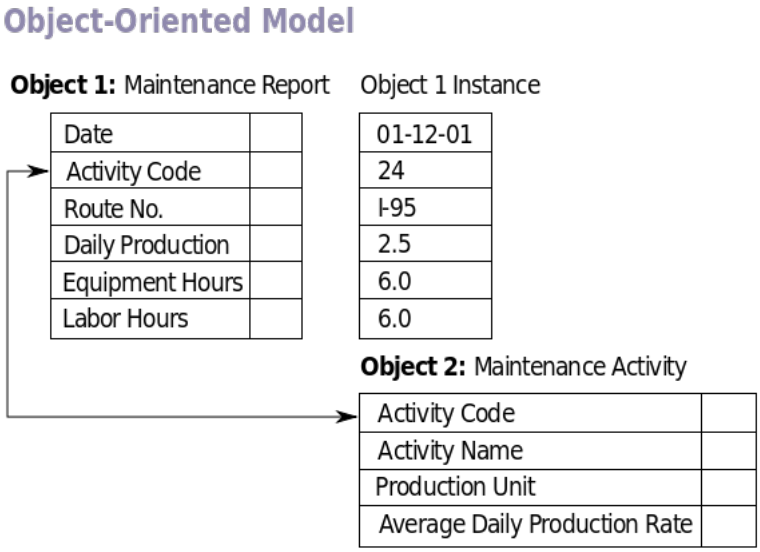
1.4.10 Αντικειμενοστραφές μοντέλο Βασης

Αυτού του είδους οι βάσεις, αποθηκεύουν αντικείμενα του πραγματικού κόσμου σε τύπους δεδομένων όπως ακέραιοι, συμβολοσειρές ή κινητής υποδιαστολής. Τα αντικείμενα αποθυηκεύονται σε αντικειμενοστραφείς γλώσσες όπως Smalltalk, C++, Java και άλλες. Τα αντικείμενα βασικά αποτελούνται:

- Ιδιοτητες Οι ιδιότητες είναι δεδομένα που υποδηλώνουν τα χαρακτηριστικά ενός αντικειμένου. Τα δεδομένα αυτά μπορεί να είναι απλά όπως π.χ. ακέραιοι, ή να έχουν αναφορά (reference) σε ένα πιο περίπλοκο αντικείμενο.

- **Μέθοδοι**Οι μέθοδοι καθορίζουν τη συμπερίφορά των αντικειμένων και είναι αυτό που επίσημα σε άλλες γλώσσες λέγεται συνάρτηση (ή διαδικασια).
  Μόλις τη δεκατία του 90 οι αντικειμενοστραφείς γλώσσες ενσωματώθηκαν στην τεχνολογία των βάσεων δεδομένων δημιουργώντας το νέο μοντέλο γνωστό ως Βασεις Αντικειμένων. Αυτό σκοπεύει ωστε να αποφευχθει η σύχγυση με τις παραδοσιακές αντικειμενοστραφείς γλώσσες, δηλαδή η επιβάρυνση της μετατροπής πληροφοριών μεταξύ της αναπαράστασης στην βάση (για παράδειγμα τις γραμμές στους πίνακες) και την αναπαράσταση στην εφαρμογή (συνήθως ως αντικείμενα). Περαιτέρω, ο τύπος συστήματος που χρησιμοποιείται σε μια συγκεκριμένη εφαρμογή μπορεί

να δηλωθεί απευθείας στη βάση επιτρεποντας στη βαση να ενισχύσει την ακεραιότητα των δεδομένων. Οι αντικειμενοστραφείς βάσεις δεδομένων εισάγουν τις βασικές ιδέες του προγραμματισμού αντικειμένων, όπως η ενθυλάκωση και πολυμορφισμός, στον κόσμο των βάσεων δεδομένων.



Σχήμα 1.9: Το αντικειμενοστραφές μοντέλο

1.4.11 Εναλλακτική μοντελοποίση βασεων δεδομένων

Υπάρχουν εναλλακτικά μοντέλα βάσεων δεδομένων όπως αυτό που θα δούμε σε αυτήν την πτυχιακή και είναι σε γλώσσα XML. Η υλοποίηση αυτή περιλαμβάνει ένα έγγραφο σε δεδροειδή η γράφου μορφή, που περιέχει όλες τις οντότητες αλλα στην πραγματικότητα δεν είναι ενα συστημα διαχείρησης βασης δεδομενων DBMS. Στο επόμενο κεφάλαιο θα δούμε αναλυτικη περιγραφη της γλώσσας αυτής καθώς και της LINQ σε περιβάλλον C.

Κεφάλαιο 2

XML και Βάσεις Δεδομένων

Το κεφάλαιο αυτό δίνει μια περιγραφή στο πως μπορεί να χρησιμοποιηθεί η XML σαν βάση δεδομένων. Περιγράφει τις διαφορές μεταξύ δεδομενοκεντρικών (datacentric) και εγγραφοκεντρικών (documentcentric) δεδομένων που επηρεάζουν την χρήση των βάσεων δεδομένων, πώς η XML συνδέεται με τις σχεσιακές βάσεις δεδομένων και ποιές είναι οι native XML βάσεις δεδομένων και πως μπορούν να χρησιμοποιηθούν. Επίσης γίνεται εκτενής ανάλυση στην προγραμματιστική γλώσσα που υποστηρίζει την XML τη LINQ.

2.1 Τι είναι η XML

Η XML είναι η συντομογραφία από τα αρχικά Extensible Markup Language. Πρόκειται για ένα ανοιχτό και δημοφιλές πρότυπο για τη σήμανση κειμένου με έναν τρόπο που να αναγνώσιμο και από τον άνθρωπο και από τη μηχανή. Με τον όρο "σήμανση κειμένου" εννοούμε ότι τα δεδομένα στα αρχεία κειμένου έχουν μορφοποιηθεί ώστε να περιλαμβάνουν σύμβολα που βγάζουν νόημα στο τι είναι τα δεδομένα. Η σύνταξη της XML είναι σε παρόμοιο στυλ με την HTML, τη γλώσσα σήμανσης του World Wide Web (WWW). Τα δεδομένα σε ένα αρχείο XML μπορούν να οργανωθούν σε ιεραρχίες, έτσι ώστε οι σχέσεις μεταξύ των στοιχείων να είναι οπτικά εμφανείς. Τα δεδομένα και η δομή πάντα παρουσιάζονται μαζί.
Το βασικό δομικό στοιχείο ενός εγγράφου XML είναι το στοιχείο (element), που ορίζεται με ετικέτες ανοιγματος (<) και κλεισίματος (>). Ένα στοιχείο πάντα έχει μια αρχή και ένα τελικό tag. Όλα τα στοιχεία σε ένα έγγραφο XML που περιέχεται σε ένα κυρίως στοιχείο γνωστό ως το στοιχείο ρίζας (root). Η XML μπορεί επίσης να υποστηρίξει τα ένθετα στοιχεία ή τα στοιχεία τα στοιχεία. Αυτή η ικανότητα επιτρέπει XML για την υποστήριξη ιεραρχικών δομών. Τα ονόματα των στοιχείων περιγράφουν το περιεχόμενο του στοιχείου, και η δομή περιγράφει τη σχέση μεταξύ των στοιχείων. Πέρα από απλό κείμενο XML, υπάρχουν σχετικές τεχνολογίες όπως τα XML σχήματα (XSD) και τα XML Transformations (XSL). Οι τεχνολογίες αυτές συμπληρώνουν τα έγγραφα κειμένου XML προσθέτοντας αξία στους τομείς της επικύρωσης και της επεξεργασίας.

Ένα έγγραφο XML θεωρείται ότι είναι «καλά σχηματισμένο» (δηλαδή, μπορεί να διαβαστεί και να κατανοηθεί από έναν XML parser) εάν η μορφή

του είναι σύμφωνη με την προδιαγραφή XML, αν είναι σωστά επισημαίνεται επάνω, και αν τα στοιχεία είναι σωστά ένθετα. Η XML υποστηρίζει επίσης την δυνατότητα να καθορίσει τα χαρακτηριστικά για τα στοιχεία και να περιγράφουν τα χαρακτηριστικά των στοιχείων στο να ξεκινούν απ'την ετικέτα του ενός στοιχείου.
Για παράδειγμα τα έγγραφα XML μπορεί να είναι πολύ απλά όπως το παρακάτω:

```
<?xml version="1.0" standalone="yes"?>

<conversation>

<greeting>Hello, world!</greeting>

<response>Hello from the planet</response>

</conversation>
```

Η δύναμη της XML βασίζεται στην απλότητά. Μπορεί να πάρει μεγάλα κομμάτια των πληροφοριών και να τα ενοποιήσει σε ένα έγγραφο XML - ουσιαστικά κομμάτια που παρέχουν τη δομή και την οργάνωση στις πληροφορίες.

Tip

Ένα έγκυρο έγγραφο είναι πάντα ένα καλά διαμορφωμένο(well-formed) έγγραφο, αλλά το αντίστροφο δεν ισχύει πάντα.

2.2 XML και Βάσεις Δεδομένων

Κανονικά τα XML documents δεν είναι αυστηρά δεδομενο-κεντρικά η εγραφο-κεντρικά, αλλά κάπου στη μέση. Ετσι, ενω αυτή η υποδιαίρεση είναι σωστό να ξεχωρίσουμε τις διαφορές μεταξύ XML native και XML enabled database ανάλογα με τις ανάγκες που θέλουμε να επεξεργαστούμε τα δεδομένα.
Υπάρχουν δύο κύριοι τύποι των βάσεων δεδομένων XML:

- XML-enabled
- Native XML (NXD)

XML- Enabled Database

Η XML βάση δεδομένων με δυνατότητα δεν είναι παρά η επέκταση που προβλέπεται για τη μετατροπή των εγγράφων XML. Αυτό είναι η σχεσιακή βάση δεδομένων, όπου τα δεδομένα είναι αποθηκευμένα σε πίνακες που αποτελούνται από γραμμές και στήλες. Οι πίνακες περιέχουν σειρά εγγραφών, τα οποία με τη σειρά τους αποτελούνται από πεδία.

XML- Native Database

Μια native βάση δεδομένων XML βασίζεται στο περιεχόμενο (container) και όχι μορφή πίνακα. Μπορεί να αποθηκεύσει μεγάλη ποσότητα εγγράφων XML και δεδομένων. τα περιεχόμενα απο native βάσεις δεδομένων XML μπορούν να ανακτηθούν μέσω ερωτημάτων Xquery ή LINQ όπως θα παρουσιαστεί παρακάτω στην πτυχιακή αυτή. Μια native βάση δεδομένων XML έχει πλεονέκτημα σε σχέση με τη βάση δεδομένων XML-enabled. Είναι ιδιαίτερα ικανή για την αποθήκευση, την αναζήτηση και τη διατήρηση του εγγράφου XML συγκριτικά με την XML-enabled βάση δεδομένων.
Το ακόλουθο παράδειγμα δείχνει μια βάση δεδομένων XML:

```
<?xml version="1.0"?>
<contact-info>
<contact1>
<name>Maria</name>
<company>Database</company>
<phone>(01) 123-4567</phone>
</contact1>

<contact2>
<name>Nikos</name>
<company>Computer</company>
<phone>(01) 789-4567</phone>
</contact2>
</contact-info>
```

Εδώ, ένας πίνακας των contact-info που δημιουργείται κρατά τις εγγραφες των επαφών (contact 1 και contact2), το οποίο με τη σειρά τους αποτελείται από τρεις οντότητες - το όνομα, η εταιρεία και το τηλέφωνο.

Προβολή αρχείων XML

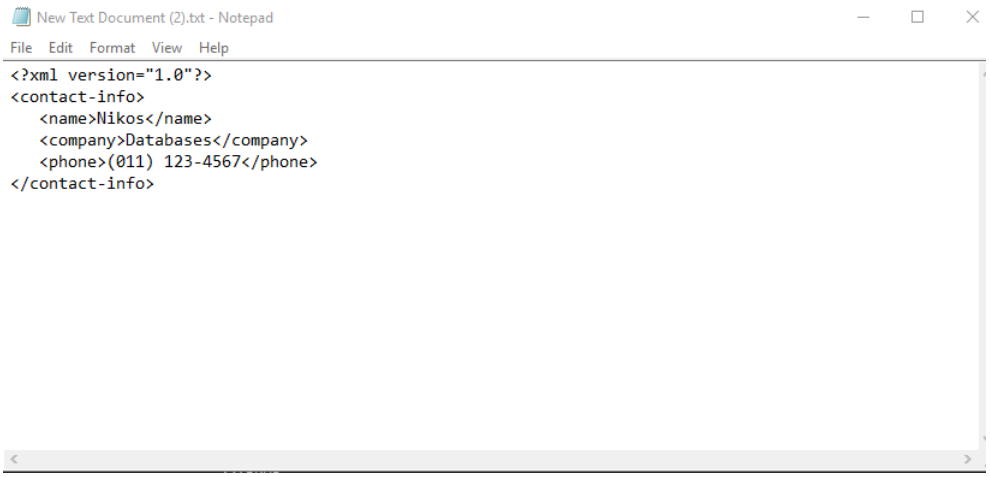
Ένα έγγραφο XML μπορεί να προβληθεί χρησιμοποιώντας ένα απλό πρόγραμμα επεξεργασίας κειμένου ή οποιοδήποτε πρόγραμμα περιήγησης (Chrome, Firefox, IE, Opera κλπ). Τα περισσότερα από τα μεγάλα προγράμματα περιήγησης υποστηρίζουν την XML. Για να δουμε τα XML αρχεία μπορούμε να τα ανοίξουμε με το πρόγραμμα περιήγησης κάνοντας διπλό κλικ στο έγγραφο XML (αν είναι ένα τοπικό αρχείο) ή πληκτρολογώντας τη διαδρομή URL στη γραμμή διευθύνσεων (εάν το αρχείο βρίσκεται στον server), με τον ίδιο τρόπο όπως και ανοίγουμε άλλα είδη αρχείων στο πρόγραμμα περιήγησης. Τα αρχεία XML αποθηκεύονται με την επέκταση ".xml".

Το ακόλουθο αρχείο δείχνει πως μπορεί να φανεί με διαφορους τρόπους:

```
<?xml version="1.0"?>
```

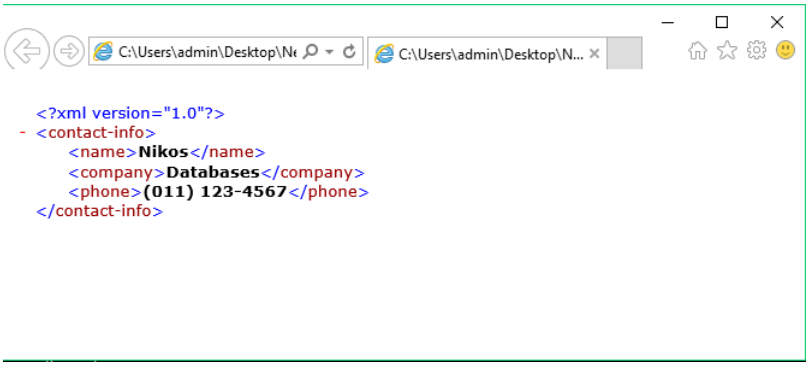
```
<contact-info>
<name>Nikos</name>
<company>Databases</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

Επεξεργαστής κειμένου



Σχήμα 2.1: Αρχείο xml σε απλό επεξεργαστή κειμένου

Web browser

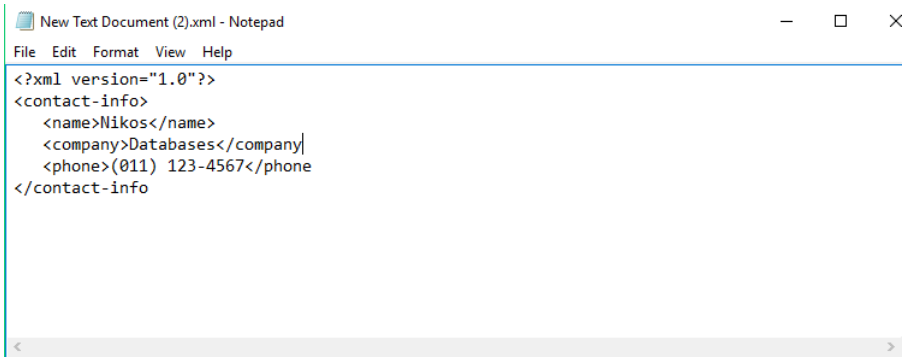


Σχήμα 2.2: Αρχείο xml σε web browser

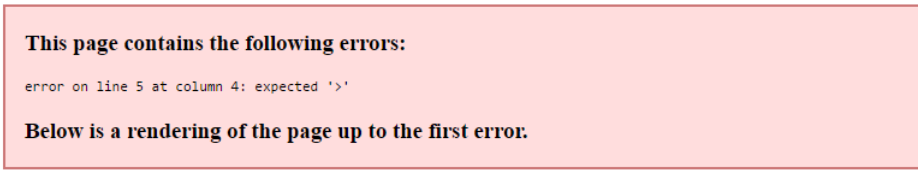
Σφάλματα - κακό markup

Αν απο τον XML κώδικα XML έχει κάποιες λείπουν καποια tags, στη συνέχεια, εμφανίζεται ένα μήνυμα λάθους στο πρόγραμμα περιήγησης.

Αν προσπαθήσουμε να ανοίξουμε το ακόλουθο αρχείο XML στον Chrome το ακόλουθο μηνυμα εμφανίζεται:



Σχήμα 2.3: Λαθη στο αρχείο XML



Nikos

Σχήμα 2.4: Παράδειγμα κακού markup

2.2.1 Μοντελοποίηση και XML

Ενα μοντέλο εγγράφου (ή αλλιώς document model) προσδιορίζει ένα σύνολο από στοιχεία (elements) καθώς και ιδιοτήτων (attributes) τα οποία εμφανίζονται σε ένα έγγραφο XML. Ένα μοντέλο εγγράφου ή πιο επίσημα και γενικά γνωστό ώς data model περιγράφει τη λογική δομή και ιεραρχία ενός σέτ δεδομένων.
Επίσης υποδηλώνει ποιές πληροφορίες περιέχει ένα σέτ δεδομένων σε όρους ονομάτων πεδίων, ποια δεδομένα περιέχει κάθε πεδίο, καθώς και τη συσχέστιση μεταξύ των πεδίων και των άλλων σέτ πεδιων.
Ενα μοντέλο δεδομένων είναι σημαντικό για τους κάτωθι λόγους:

- Πρέπει να υπάρχει ένα κατάλληλο πρότυπο δομής ώστε να είναι εύκολη η ανάκτηση του σε σύνθετες δομές π.χ. ενα Λεξικό
- Μείωση του κόστους γιατί δεν χρείαζεται συχνή συντήρηση
- Εξασφάλιση ότι το XML έγγραφο ιδιαίτερα όταν είναι μεγάλο πρέπει να τηρεί ένα επίπεδο ποιότητας ώς προς τη δομή και τα δεδομένα που περιέχει.
- Τα XML αρχεία γράφονται απο ανθρώπους και πρέπει να είναι εύκολα αναγνώσιμα από άλλους ανθρώπους/

Υπάρχουν τρείς κύριες τεχνολογίες που χρησιμοποιούνται για να δημιουργήσουμε ένα μοντέλο δεδομένων στα XML έγγραφα μας. Αυτές είναι τα:

DTD

Το DTD (Document Type Definition) ή αλλιώς Προσδιοριστής Τύπου Δεδομένων είναι μια τεχνολογία που είναι μέρος των XML parsers.

```
<?xml version="1.0"?><!DOCTYPE books [
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT authors (author)+>
<!ELEMENT subject (#PCDATA)>
<!ATTLIST subject class CDATA "">
<!ELEMENT book (title,authors,subject)>
<!ATTLIST book
    bookid CDATA #REQUIRED
    pubdate CDATA #REQUIRED>
]>
<books name="My books">
    <book bookid="1" pubdate="03/01/2002">
        <title>Java Web Services</title>
        <authors>
```

Σχήμα 2.5: Μορφή Εγγράφου DTD

Ο συγκεκριμενος τρόπος αποθήκευσης έχει κάποια μειονεκτήματα. Δηλαδή:

- Τα DTDs χρησιμοποιούν εξειδικευμένο συντακτικό διαφορετικό απ'αυτο της XML κατι που τα καθιστά δύσκολα ιδιαίτερα σε νέους χρήστες.
- Τα DTDs δεν επιτρέπουν στους χρήστες να προσδιορίσουν ποιόν τύπο δεδομένων ένα στοιχείο μπορεί να περιέχει.
- Τα DTDs έχουν έναν σταθερό μή επεκτάσιμο τύπο περιεχομένου που δεν επιτρέπει στους χρήστες να δημιουργήσουν νέα στοιχεία elements και νέες ιδιότητες (attributes).
- Δεν υποστηρίζουν namespaces.

XDR

Τα XDR ή XML Data Reduced είναι ένα XML "λεξικό" που επινοήθηκε απο τη microsoft. Το XDR έχει τη διαφορά ότι δεν περιγράφει μόνο τα δεδομένα αλλα και τον τύπο των δεδομένων. Κύριο μειονέκτημα είναι η μειωμένη υποστήριξη.

XSD

Πρόκειται για ένα σχετικά νέο πρότυπο που λανσαρίστηκε απο το W3C consortium και σταδιακά αρχίζει να καθιερώνεται.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
Project Management
-->
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Database">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="DIVISION">
               <xs:complexType>
                  <xs:attribute name="DIVNUM">
                     <xs:simpleType>
                        <xs:restriction base="ID">
                           <xs:minInclusive value="1"/>
                           <xs:pattern value="00000"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:attribute>
                  <xs:attribute name="DIVNAME" type="NAME">
                  </xs:attribute>
                  <xs:attribute name="DIVADDR" type="SHORT_TEXT">
                  </xs:attribute>
               </xs:complexType>
            </xs:element>
```

Σχήμα 2.6: Μορφή Εγγράφου XDR

Στην πτυχιακή αυτή θα ασχοληθούμε με μια δομή στην γλώσσα .NET που μοιάζει κατα πολύ με την HTML. Παρακάτω φαίνονται μερικά παραδείγματα δομών XML.

Παραδειγματα well formed XML

Εστω η παρακάτω δομή

```
def Person = {
name: String
age:  Integer
}
```

Μια επιλογή θα μπορούσε να είναι αυτή:

```
<Person
name="Kostas"
age="23"
/>
```

Η αυτή:

```
<Person>
<name>Kostas</name>
<age>23</age>
</Person>
```

Μια δομή πιο σύνθετη όπως η παρακάτω

```
<Person>
<name>Kostas</name>
<age>23</age>
</Person>
```

```
<?xml version="1.0" standalone="yes" ?>
- <Sample.xml>
  - <xs:schema id="Sample.xml" xmlns=""
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    - <xs:element name="Sample">
      - <xs:complexType>
        - <xs:sequence>
          - <xs:element name="Book" minOccurs="0" maxOccurs="unbounded">
            - <xs:complexType>
              - <xs:sequence>
                  <xs:element name="Number1" type="xs:int" minOccurs="0" />
                  <xs:element name="String2" type="xs:string" minOccurs="0" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="name" type="xs:string" />
        </xs:complexType>
      </xs:element>
    </xs:schema>
  - <Sample name="First">
    - <Book>
        <Number1>1</Number1>
        <String2>aaaaaaaaaaa</String2>
      </Book>
    - <Book>
        <Number1>2</Number1>
        <String2>bbbbbbbbbb</String2>
      </Book>
    - <Book>
        <Number1>3</Number1>
        <String2>ccccccccccc</String2>
      </Book>
    </Sample>
  </Sample.xml>
```

Figure 1.
Sample.xml with in-line XSD schema

Σχήμα 2.7: Μορφή Εγγράφου XSD

Βλέποντας την αναπαράσταση βασισμένη σε ιδιοτητες βλέπουμε ότι δεν δουλεύει.

```
<Person
name=<FullName first="Mad" last="Goose" />
age="30"
/>
```

Ένα καλύτερο σχέδιο είναι η παρουσία δύο διαφορετικών μορφών είναι συντακτικά σωστή. Μερικές φορές είναι απλώς πιο βολικό για να διαβάσετε τα χαρακτηριστικά / εγγραφής, αλλά οι δύο μορφές θα μπορούσαν να έχουν γίνει ισοδύναμες:

```
<!-- kanoniki morfi -->

<para>
<font>Arial</font>
<content>Hello</content>
</para>

<!-- syntomi morfi -->

<para font="Arial">Hello</para>

<para font="Arial" content="Hello"/>

<para content="Hello"/>
```

```
<font>Arial</font>
</para>
```

Μεικτό περιεχόμενο

Μια κακή πρακτική είναι αυτή:

```
<p>Den <em>einai</em> sosto!</p>
```

Ενώ η παρακάτω δείχνει το ιδιο κείμενο με τη σωστή γραφή

```
<p>
<text value="1 "/>
<em>
<text value="2"/>
</em>
<text value="3"/>
</p>
```

"Μικτό περιεχόμενο" είναι ακριβώς το αποτέλεσμα δύο κομμάτια της συντακτικής γραφής και βασίζεται στην τεχνοτροπία του κάθε σχεδιαστή. Η XML δεν επιτρέπει "ελεύθερο κείμενο" οπως παρακατω

```
<!-- den tha doulepsei -->
<hobbies>
Sewing
Cup Stacking
Parking
</hobbies>
```

Αντ' αυτου θα μπορούσαμε να το περιγράψουμε όπως κάτωθι:

```
<!-- sosto -->
<hobbies>
<text value="Sewing"/>
<text value="Cup Stacking"/>
<text value="Parking"/>
</hobbies>
```

Τύποι regular expression

Όπως προειπώθηκε, υπάρχουν πολλοί τροποι να αναπαρασταθεί ενα XML εγγραφο. Εδω παρουσιάζεται πώς με την XML μπορεί να αναπαραστεθεί ένας συγκεκριμένος τύπος δεδομένων.

```
type Boat = {
Name:  String
Color: String
Class: String
}
```

Για να αναπαρασταθεί αυτό μεσω του DTD γίνεται κάπως ετσι:

```
<!ELEMENT Boat (Name, Color, Class)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Color (#PCDATA)>
<!ELEMENT Class (#PCDATA)>
```

Το πρόβλημα με το DTD είναι ότι απαιτεί απο όλα τα πεδία να εμφανίζονται σε μία συγκεκριμένη σειρά. Επίσης το DTD προσθέτει επιπλέον αχρείαστες παραμέτρους στους τύπους δεδομένων.

```
<!ELEMENT Boat ((Name, Color, Class)|(Name, Class,
    Color)
|(Color, Name, Class)|(Color, Class, Name)
|(Class, Name, Color)|(Class, Color, Name))>
```

Με αυτη την πρακτική έχουμε περιττές ετικέτες XML, όπου, στη χειρότερη περίπτωση, το έγγραφο είναι διπλάσιο από το μέγεθος που πρέπει να είναι. Για τους προηγουμενους λόγους, τα attributes στα DTDs δεν είναι χρήσιμα σε γενικές γραμμές.

Πλεονασμός στα δεδομένα

Παρακάτω φαίνεται παράδειγμα επαναχρησιμοποίησης στην XML στο παρακάτω παράδειγμα Τηλεφωνικού Καταλόγου:

```
def PhoneNumber = {
AreaCode: String
Number: String
}
def TContact = {
Name: String
HomePhone: PhoneNumber
WorkPhone: PhoneNumber
}
```

```
<Contact>
<Name>Wolverine</Name>

<HomePhone>
<PhoneNumber> <---pleonasmos--->
<AreaCode>123</AreaCode>
<Number>555-1234</Number>
</PhoneNumber>
</HomePhone>

<WorkPhone>
<PhoneNumber> <---pleonasmos--->
<AreaCode>123</AreaCode>
<Number>555-4321</Number>
```

```
</PhoneNumber>
</WorkPhone>
</Contact>
```

Στα παραπάνω στοιχεία, οι επιπλέον <PhoneNumber> tags είναι εντελώς περιττές. Στις περισσότερες γλώσσες προγραμματισμού, δεν χρειάζεται να "επαναλάβεται ο εαυτός" οπως εδώ. Το πρόβλημα είναι ότι ακόμα κι αν το HomePhone είναι ένα πεδίο και το PhoneNumber είναι ένας τύπος δεδομένων, η XML δεν μπορεί να "δει" τη διαφορά.

Γενικά ένα αρχείο XML είναι ένα format αρχείου για αποστολή δεδομένων από ένα σύστημα στο άλλο. Η για αποθήκευση μικρών σέτ δεδομένων. Δηλαδή στην πράξη είναι δύσκολο να γίνει χειρισμός μιας βάσης με 500mb XML έγγραφα το καθένα. Υπάρχουν κάποιες περιπτώσεις που οι γηγενείς xml βάσεις δεδομένων έχουν πράγματι πολύ μεγάλο πλεονέκτημα έναντι των RDBs όταν το περιεχόμενο προς αποθήκευση είναι ημιδομημένο.

Tip

Μια βάση δεδομένων *δεν* είναι για την αποθήκευση δεδομένων, αλλά για να φιλτραρει δεδομένα σύμφωνα με συγκεκριμένα κριτήρια (ερωτήματα).

2.2.2 XML vs RDBMS

Όταν οι άνθρωποι μιλούν για βάσεις δεδομένων, σκεφτονται συνήθως συστήματα αποθήκευσης που αποθηκεύουν τεράστιες ποσότητες δεδομένων, συχνά με gigabyte ή terabyte φάσμα. Μια βάση δεδομένων είναι δυνητικά πολύ μεγαλύτερη από το ποσό της διαθέσιμης μνήμης RAM στο διακομιστή που αποθηκεύει. Δεδομένου ότι κανείς δεν χρειάζεται πάντα όλα τα δεδομένα σε μια βάση δεδομένων μονομιας, οι βάσεις δεδομένων πρέπει να βελτιστοποιούνται για γρήγορη ανάκτηση των υποσύνολα των δεδομένων τους: αυτό είναι που η δήλωση SELECT είναι, και σχεσιακές βάσεις δεδομένων, βελτιστοποιούν τη μορφή εσωτερικής αποθήκευσης τους για γρήγορη ανάκτηση των εν λόγω υποσύνολων.

Η XML, όμως, δεν ταιριάζει πραγματικά αυτές τις απαιτήσεις. Λόγω της ένθετης δομής των tags, είναι αδύνατο να προσδιοριστεί ο τόπος όπου το αρχείο είναι αποθηκευμένο μια ορισμένη τιμή (από την άποψη της ένα byte offset σε ένα αρχείο), χωρίς να διανυστεί το δέντρο του εγγράφου. Μια σχεσιακή βάση δεδομένων έχει ευρετήρια που δινει σημαντικό πλεονέκτημα στην απόδοση της τάξης του O(logn). Σε ένα αρχείο XML,δεν μπορούμε να έχουμε απόδοση καλύτερη απο καθόλου καλύτερη από O(n),

Πίνακας 2.1: Σύγκριση XML και RDBMS

	Σχεσιακές Βάσεις	XML
Δομή	Πίνακας	Ιεραρχικό Δέντρο, Γράφος
Σχήμα	Σταθέρο χωρίς αλλαγές	Προσαρμόσιμο, "Αυτοπεριγραφική" γλώσσα
Ερωτήματα	Εύκολια και ποικιλία ερωτηματων (queries)	Πιο δύσχρηστα ερωτήματα, ειδικά σε πολύπλοκες δεντροειδείς δομές
Ordering	Οχι.	Εμμεσο.
Εφαρμογή	Εσωτερική	Add-on

αν δεν χρησιμοποιήσετε ευρετήρια, αλλά στη συνέχεια, θα πρέπει να ξαναχτίσουμε το σύνολο του δείκτη για κάθε εισαγωγή.

Ένα XML αρχείο, είναι μια βάση δεδομένων με την ευρύτερη έννοια του όρου. Δηλαδή πρόκειται για μία συλλογή από δεδομένα. Αυτό μας δίνει την αίσθηση ότι πρόκειται για ένα απλό αρχείο όπως τα κοινά αρχεία. Αλλά σαν "XML" αρχείο, έχει κάποια επιπλέον πλεονεκτήματα. Για παράδειγμα, είναι *αυτοπεριγραφικό* (πρόκειται για σημειολογική-markup-γλώσσα όπου περιγράφει την δομή και τον τύπο των δεδομένων), είναι *Φορητή(Unicode)*και μπορεί να περιγράψει τα δεδομένα σε δομή δέντρου ή γράφου.

Τι κοινό έχει μια RDBMS και ένα έγγραφο XML?

Και ταέγγραφα XML και οι Σχεσιακές Βάσεις Δεδομένων αποθηκεύουν δεδομένα. Και οι δύο έχουν καθιερωμένες τεχνικές (ερωτήματα) για την εξαγωγή των δεδομένων που περιέχουν.

Τι κάνει μια σχεσιακή βάση δεδομένων καλύτερα;

Οι Σχεσιακές βάσεις δεδομένων είναι καλύτερες για το χειρισμό μεγάλων όγκων δεδομένων μέσα σε ένα σύστημα. Έχουν συστήματα διαχείρισης που μπορούν αποτελεσματικά να διατηρούν μεγάλες ποσότητες δομημένων δεδομένων. Αυτά τα δεδομένα μπορούν να ενημερωθουν διασφαλίζοντας την ακεραιότητα της βάσης δεδομένων και το περιεχόμενο μπορεί να εξαχθεί πολύ γρήγορα, όταν ρυθμιστεί σωστά. Δεν υπάρχει ισοδύναμο σύστημα διαχείρισης XML, και η χρήση των XML έγγραφων για την αποθήκευση και τη διατήρηση μεγάλου όγκου δεδομένων μπορεί να αποδειχθεί αναποτελεσματική και αναξιόπιστη. Η XML αριθμεί μια έλλειψη χαρακτηριστικών σχέτιζόμενη με τις κλασσικές βάσεις όπως επαρκής αποθήκευση, ευρετήρια (indexes), ασφάλεια, ακεραιότητα δεδομένων, πρόσβαση πολλαπλών χρηστών (multiuser environment) και ερωτήματα πάνω σε πολλαπλές βάσεις.

Τι κάνει η XML καλύτερα;

Για να καταστεί δυνατή η επικοινωνία δεδομένων μεταξύ διαφορετικών συστημάτων που έχουν διαφορετικά περιβάλλοντα ανάπτυξης και υλοποιήσεις, απαιτείται μια κοινή μορφή δεδομένων. Η Extensible Markup Language (XML), η οποία είναι μια πλατφόρμα ανεξάρτητη,μπορεί να χρησιμοποιηθεί για τη μεταφορά δεδομένων μεταξύ διαφορετικών συστημάτων με διαφορετικές εφαρμογές και περιβάλλοντα. Τα XML έγγραφα περιέχουν μαζι τα δεδομένα και τη σχέση δόμησης των δεδομένων με τέτοιο τρόπο ώστε να είναι εύκολα αναγνώσιμα και στον χρήστη και στη μηχανή. Ένα έγγραφο XML μπορεί να διαβιβάζονται ηλεκτρονικά από το ένα μέσο στο άλλο και όλες οι πληροφορίες που μεταφέρονται με αυτό ειναι αυτοπεριγραφικές (self-described). Οι παραδοσιακές βάσεις δεδομένων, αν και αυτο-περιγράφονται με τη χρήση κατάλληλων εργαλείων, δεν μεταδίδονται γενικά στο σύνολό τους από το ένα μέσο στο άλλο. Γι'αυτο η xml είναι γλώσσα που μεταφέρεται μεταξύ των server. Από τη στιγμή που η XML είναι ιδανική για να μεταφέρει αυτο-περιγραφικα data feeds, έχει γίνει το βασικό πρότυπο στις Service Oriented Architectures (SOA) και στα Web Services.

Μια άλλη ερώτηση είναι στο εαν η XML και οι τεχνολογίες της απαρτίζουν μια βάση όπως ενα "κανονικο" DBMS. Η XML εντέλει έχει πολλά από τα χαρακτηριστικά που βρίσκουμε και στις κλασσικές βάσεις δεδομένων:

- **Αποθήκευση** Έγγραφα XML, διαγραμματα δεδομένων (schema) όπως τα DTDs, XML Schemas, RELAX NG.
- **Γλώσσες ερωτημάτων**XQuery, XPath, LINQ, XQL, XML-QL, QUILT
- **Προγραμματιστικά interface** SAX, DOM, JDOM.

Στόν αντίποδα,
Έτσι, ενώ είναι πιθανό να χρησιμοποιήσουμε ένα αρχείο ή έγγραφα XML σαν μια βάση δεδομένων σε περιβάλλοντα με μικρό όγκο δεδομένων λιγους χρήστες και κανονικές απαιτήσεις ταχύτητας, αυτό δεν ενδύκνειται σε μεγάλα περιβάλλοντα χρηστών.

Ένα παράδειγμα που ένα αρχείο XML είναι ιδανικό να "συμπεριφερθεί" σαν βάση δεδομένων, είναι το αρχείο συστήματος .ini -που περιέχει πληροφορίες διαμόρφωσης εφαρμογών. Η XML επιτρέπει να υπαρχουν φωλιασμένες (nested) δομές και αυτό πλεονεκτεί παρολο που δύσκολα μια τέτοια δομή όπως αυτή που περιγράφηκε παραπάνω μπορεί να λεχθεί οτι είναι βάση δεδομένων αφού η ανάγνωση και η εγγραφή γίνεται *γραμμικά* και μόνο όταν η εφαρμογή ξεκινάει και τερματίζει (αντλεί πληροφορίες προφανώς από το αρχείο .ini).

Πιο εξειδικευμένα σετ δεδομενων που προσεγγίζουν κοντά τις βάσεις δεδομένων, είναι οι προσωπικές λίστες επαφών (ονόματα, τηλεφωνικοί

αριθμοί, διευθύνσεις κτλ) τα bookmark κάποιου web browser, περιγραφές mp3 tags κτλ.

2.2.3 Δεδομένα και XML Έγγραφα

Ίσως ο σημαντικότερος παράγοντας για να χρησιμοποιήσουμε μια βάση δεδομένων είναι ο βάσιμος σκοπός αν αποθηκεύουμε *δεδομένα* ή αν αποθηκεύουμε *έγγραφα*. Για παράδειγμα, η XML χρησιμοποιείται απλά σαν μεταφορά δεδομένων μεταξύ της βάσης? Η μπορεί να χρησιμοποιηθεί και σαν ενσωματωμένη γλώσσα στην περίπτωση των XTHML ή DocBook εγγράφων; Αυτό είναι σημαντικό επειδή όλα τα δεδομενοστρεφή έγγραφα μοιράζονται εναν αριθμό απο χαρακτηριστικά όπως γίνεται και με όλα τα εγγραφοστρεφή έγγραφα και αυτά επηρεάζουν στο πως τα έγγραφα XML αποθηκέυονται μέσα στη βάση δεδομένων. Οι επόμενες δύο υποενότητες εξετάζουν αυτά τα χαρακτηριστικά:

2.2.4 Δεδομενοστρεφή έγγραφα

Τα δεδομενοστρεφή έγγραφα είναι αυτά που χρησιμοποιούν την XML σαν μεταφορέα δεδομένων ή αλλιώς εστιάζουν μόνο σε κομμάτια της συνολικής δομής και όχι στο σύνολο. Ετσι, με αυτή την έννοια, δεν είναι σημαντικό στην εφαρμογή ή στην βάση ότι τα δεδομένα είναι αποθηκευμένα οπωσδήποτε σε ένα XML αρχείο. Παραδείγματα από δεδομενοστρεφή έγγραφα είναι οι παραγγελίες πωλήσεων, τα δρομολόγια πτήσεων, τα επιστημονικά δεδομένα, και η καταγραφή αποθεμάτων στόκ. Χαρακτηρίζονται απο κανονική δομή και ομοιόμορφο περιεχόμενο.
Δεδομένα αυτού του είδους που βρίσκονται στα δεδομενοστρεφή έγγραφα μπορούν να προέρχονται και μέσα απο την βάση(οπου θέλουν να εκτεθούν σαν XML) και έξω απο αυτή (όπου στη περίπτωση αυτή θέλουμε να τα αποθηκεύσουμε στη βάση δεδομένων). Δηλαδή αντί να ενδιαφερόμαστε για όλο το σύνολο του τιμολογίου. ενδιαφερόμαστε μόνο για πληροφορίες πχ το ποσό των χρημάτων που θα πληρώναμε.

Για παράδειγμα το παρακάτω αρχείο παραγγελιών είναι δεδομενοκεντρικό:

```
<SalesOrder SONumber="12345">
<Customer CustNumber="543">
<CustName>ABC Industries</CustName>
<Street>123 Main St.</Street>
<City>Chicago</City>
<State>IL</State>
<PostCode>60609</PostCode>
</Customer>
<OrderDate>981215</OrderDate>
<Item ItemNumber="1">
<Part PartNumber="123">
```

```
<Description>
<p><b>Turkey wrench:</b><br />
Stainless steel, one-piece construction,
lifetime guarantee.</p>
</Description>
<Price>9.95</Price>
</Part>
<Quantity>10</Quantity>
</Item>
<Item ItemNumber="2">
<Part PartNumber="456">
<Description>
<p><b>Stuffing separator:<b><br />
Aluminum, one-year guarantee.</p>
</Description>
<Price>13.27</Price>
</Part>
<Quantity>5</Quantity>
</Item>
</SalesOrder>
```

Το παραπάνω είναι ένα παράδειγμα δεδομενοκεντρικού εγγράφου. Ένα άλλο τέτοιο παράδειγμα είναι πχ μια σελίδα στο amazon που δείχνει πληροφορίες για ένα βιβλίο. Αν και η σελίδα μπορεί να είναι μεγάλη σε κείμενο, η δομή από αυτό το κείμενο είναι ομοιόμορφη αφού όλες οι σελίδες στη συγκεκριμένη σελίδα θα έχουν παρόμοιο ύφος παρουσιάζοντας όλες βιβλία ενώ κάθε κομμάτι της σελίδας έχει συγκεκριμένο όριο κειμένου (ας πούμε μια περιγραφή βιβλίου). Για τον λόγο αυτό, η σελίδα μπορεί να αντλήσει τα δεδομένα από ένα μικρό και απλό δεδομενοστρεφή XML αρχείο το οποίο περιέχει πληροφορίες σχετικά με το βιβλίο και γίνεται ανάκτηση από τη βάση. Γενικά, κάθε ιστοσελίδα που κατασκευάζει δυναμικά έγγραφα HTML μπορεί καθε φορά να έχει ένα πρότυπο βάσης και να αντλεί δεδομένα απο κει. Για παράδειγμα το παρακάτω περιγράφει μια πτηση:

```
<FlightInfo>
<Airline>ABC Airways</Airline> provides <Count>
    three</Count>
non-stop flights daily from <Origin>Dallas</Origin
    > to
<Destination>Fort Worth</Destination>. Departure
    times are
<Departure>09:15</Departure>, <Departure>11:15</
    Departure>,
and <Departure>13:15</Departure>. Arrival times
    are minutes
later.
</FlightInfo>
```

```
A                                :
```

```
<Flights>
<Airline>ABC Airways</Airline>
<Origin>Dallas</Origin>
<Destination>Fort Worth</Destination>
<Flight>
<Departure>09:15</Departure>
<Arrival>09:16</Arrival>
</Flight>
<Flight>
<Departure>11:15</Departure>
<Arrival>11:16</Arrival>
</Flight>
<Flight>
<Departure>13:15</Departure>
<Arrival>13:16</Arrival>
</Flight>
</Flights>
```

2.2.5 Εγγραφοστρεφή έγγραφα

Το αρχείο με επίκεντρο τη *χρήση* των δεδομένων XML είναι τα δεδομένα που χρησιμοποιούνται πάντα στην πλήρη μορφή τους. Αν θέλουμε να χρησιμοποιήσουμε τα δεδομένα, τότε μπορούμε πάντα θα το ανακτήσσουμε ως μία οντότητα ή μπορούμε να το αποθηκεύσετε ως μία οντότητα. Δεν μας ενδιαφέρει για τα δεδομένα εσωτερικά XML δηλαδή οι πληροφορίες μέσα στο ”πακέτο” αρχείο, μας ενδιαφέρει μόνο η μορφή στο σύνολο της. Ας πούμε, αν έχουμε ένα τιμολόγιο το οποίο μπορεί να εκτυπωθεί σε ένα φύλλο χαρτιού. Αυτό το αρχείο που περιέχει τα δεδομένα, πάντα θα αντιμετωπίζονται σαν ένα αρχείο, δηλαδή, σαν συνολικά οι πληροφορίες θα περιέχονται σε ένα φύλλο χαρτιού (αρχείο).

2.2.6 Δεδομένα, Έγγραφα και Βάσεις Δεδομένων

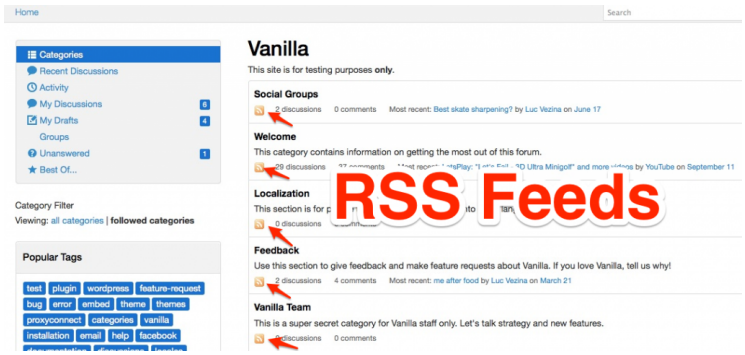
Στην πράξη, η διάκριση μεταξύ των δεδομένοστρεφών και των αρχείοστρεφών έγγραφων δεν είναι πάντα σαφής. Για παράδειγμα, ένα έγγραφοστρεφες αρχείο, όπως ένα τιμολόγιο, θα μπορούσε να περιέχει ακανόνιστα δομημένα δεδομένα. Διαφορετικά ένα έγγραφοστεφές αρχείο, όπως ένα εγχειρίδιο χρήστη, μπορεί να περιέχει τακτικά δομημένα δεδομένα (συχνά μεταδεδομένα), όπως είναι το όνομα ενός συγγραφέα και πχ ημερομηνία επανάληψης ενός μαθήματος. Άλλα παραδείγματα περιλαμβάνουν νομικά και ιατρικά έγγραφα, τα οποία περιέχουν διακριτά κομμάτια των δεδομένων, όπως ημερομηνίες, ονόματα, και διαδικασίες, και *συχνά πρέπει να αποθηκεύονται ως πλήρη έγγραφα για νομικούς λόγους*. Παρά το γεγονός αυτό, χαρακτηρίζοντας τα δεδομένα σε μια απο τις δυο κατηγορίες θα μας βοηθήσει να αποφασίσουμε τι είδους βάση δεδομένων θα χρήσιμοποιήσουμε. Κατά γενικό κανόνα, τα δεδομένα αποθηκεύονται σε μια παραδοσιακή βάση δεδομένων, όπως μια σχεσιακή, ή ιεραρχική βάση

δεδομένων. Αυτό μπορεί να γίνει από τρίτους middleware ή με ενσωματωμένες δυνατότητες για να ίδιας της βάσης δεδομένων. Στην τελευταία περίπτωση, η βάση δεδομένων λέγεται ότι είναι XML-enabled. Τα έγγραφα αποθηκεύονται σε μητρική βάση δεδομένων XML (μια βάση δεδομένων ειδικά σχεδιασμένη για την αποθήκευση XML) ή ένα περιεχόμενο σύστημα διαχείρισης (μία εφαρμογή που σχεδιάστηκε για τη διαχείριση εγγράφων και χτισμένο στην κορυφή μιας εγγενή βάση δεδομένων XML

2.2.7 Χρήσεις της XML

Η XML και η δομή των tags της έχει ευρεία χρήση. Μερικές από τις κοινότερες χρήσεις της είναι:

- **RSS feeds** Το ακρωνύμιο RSS (απο το Rich Site Summary) δηλαδή σε ελεύθερη μετάφραση σύνοψη πολλαπλής διανομής, είναι η ανταλλαγή πληροφοριών βασισμένη σε ένα πρότυπο συνήθως την γλώσσα XML.



  Ένας χρήστης μπορεί να ενημερώνεται απο ιστοσελίδες που υποστηρίζουν ροές rss για ειδήσεις, γεογονότα, χρηματιστηριο κλπ. Με αυτή την έννοια λέμε ότι ο χρήστης λαμβάνει μεταδεδομένα (metadata)

- **Datasets**Όλοι μας λιγο πολύ έχουμε ”συναντήσει” ένα XML dataset αφού στο διαδίκτυο υπάρχουν δισεκατομμύρια. Αλλιώς λέγονται και ώς μεταδεδομένα. Αυτά μπορει να έχουν Γεωχωρικες πληροφορίες (κτηματολογιο) δηλαδη καθιστούν δυνατή την εξεύρεση και χρήση τους. Ενα πολύ συχνό παράδειγμα ειναι τα ID tags της βιβλιοθήκης τραγουδιών (playlist) που έχουμε αποθηκευμένη στον υπολογιστή μας.

- **Εγγραφα** Η XML χρησιμοποιείται σε ένα μεγάλο πλήθος εγγράφων. Οι χρήσεις της μπορει να έιναι σε επεξεργαστή κειμένου, σε λογιστικό φύλλο EXCEL, βάση δεδομενων ACCESS, SQL, LINQ, XML. Βλέπουμε πως η απλή της δομή ειναι αναγνώσιμη απο παντου και αυτό την κάνει εξαιρετικά πρακτική!

- **Sitemaps** Το sitemap χρησιμοποιείται από crawlers των μηχανών αναζήτησης για να μάθει παρούσες σελίδες οι οποίες έχουν αλλάξει πρόσφατα.
- **XML Web Services** Πρόκειται για τα πρωτοκολλα WSDL, SOAP, RDF, Ajax των web services ή αλλιως διαδικτυακών υπηρεσιών, είναι οι υπηρεσίες (συνήθως κάποιου συνδυασμου προγραμματισμού και δεδομένων, αλλά ενδεχομένως συμπεριλαμβανομένων και ανθρωπίνων πόρων) που διατίθενται από το διακομιστή Web μιας επιχειρησης (bussiness server) για τους χρήστες του ιντερνετ ή άλλα web συνδεδεμένα προγράμματα.

2.2.8 Γιατι XML

Κεφάλαιο 3

Η γλώσσα LINQ to XML

Στο κεφάλαιο αυτό παρουσιάζεται η γλώσσα περιγραφής LINQ. Η LINQ to XML είναι μία προσέγγιση του προγραμματισμού με την XML. Παρέχει τις δυνατότητες τροποποίησης του εγγράφου στη μνήμη του Μοντέλου Αντικειμένου Εγγράφου (DOM), και υποστηρίζει LINQ εκφράσεις ερώτηματων. Παρά το γεγονός ότι αυτές οι εκφράσεις ερώτηματων είναι συντακτικά διαφορετικοί από το XPath, παρέχουν παρόμοια λειτουργικότητα.

3.1 Εισαγωγή

Η LINQ (Language Integrated Query) είναι ένα σύνολο χαρακτηριστικών για να επεκτείνει την ικανότητα ερώτηματων της γλώσσας C# με την παροχή ενός IntelliSence και σύνταξη τονίζοντας χαρακτηριστικά. Ως «LINQ» είναι ενσωματωμένη στη γλώσσα C#, ως εκ τούτου, ο εντοπισμός σφαλμάτων των ερωτημάτων του «LINQ» είναι δυνατόν να χρησιμοποιηθεί με το ενσωματωμένο πρόγραμμα εντοπισμού σφαλμάτων. Η LINQ μπορεί να χρησιμοποιηθεί για την αναζήτηση σχεδόν κάθε των δεδομένων, συμπεριλαμβανομένης της XML DOM. Αυτό το κεφάλαιο εξηγεί πώς η LINQ μπορεί να χρησιμοποιηθεί για να αλληλεπιδράσει με τα έγγραφα XML μεσω των ερωτηματων.

3.1.1 Τι είναι η C# XML LINQ;

Η C# LINQ to XML μπορεί να θεωρηθεί μια ομάδα κλάσεων, που βρίσκονται στη βιβλιοθήκη, ή αλλιώς namespace του «System.Xml.Linq», το οποίο μπορεί να χρησιμοποιηθεί για να μας επιτρέψει να δημιουργήσουμε αποτελεσματικά, να τροποποιήσουμε, να διαγράψουμε και να ανακτήσουμε τα δεδομένα ενός αρχείου ”XML”, χρησιμοποιώντας ερωτήματα LINQ. Τα παρακάτω είναι σημαντικές λειτουργίες που παρέχονται από αυτές τις κατηγορίες:

- Φόρτωση του περιεχομένου XML στη μνήμη από αρχείο ή απο stream.
- Σειριοποίηση των δεδομένων σε μορφή XML και αποθηκεύση σε ένα αρχείο ή stream.
- Δημιουργία δέντρων XML, συμπεριλαμβανομένων των κόμβων, στοιχείων(elements), ιδιοτήτων(attributes),κλπ.

- Ερωτήματα queries σε XML (κόμβους) με τη χρήση ερωτημάτων LINQ.
- Τροποποίηση και διαγραφή του περιεχομένου XML στη μνήμη (πλήρες δέντρο, κόμβους, στοιχεία, χαρακτηριστικά, κ.λπ.).
- Επικύρωση του περιεχομένου XML χρησιμοποιώντας το XML Schema Definition (XSD).
- Ο μετασχηματισμός XML εγγράφων από τη μία μορφή στην άλλη χρησιμοποιώντας μια Extensible Style Γλώσσα Μετασχηματισμού(XSLT).
- Αποθήκευση XML σε μια ποικιλία τύπων εξόδου (αρχείο, XmlWriter, κλπ)

Χαρακτηριστικά της LINQ σε σχέση με την SQL

Η γλώσσα LINQ (LINQ σε SQL,XML, Parallel LINQ κ.λπ) είναι πολύ καλή όπου εφαρμόζεται κατάλληλα, αλλά είναι πιο δύσκολη στην κατανοήση και χρειάζεται περισσότερη εμπειρία απο τη πλευρά του χρήστη. Επίσης, μπορεί να εφαρμόστεί μόνο σε ορισμένες περιοχές του κώδικα. Πλεονεκτήματα και μειονεκτήματα της LINQ συνοψίζονται παρακάτω:

Πλεονεκτήματα

- Οι πίνακες δημιουργούνται αυτόματα στην κλάση
- Οι στήλες δημιουργούνται αυτόματα στις ιδιότητες(attributes)
- Τα ερωτήματα μπορούν να είναι δυναμικά
- Οι συσχετίσεις γίνονται αυτόματα στην κλάση
- Επιτρέπει επεκτασιμότητα και τα δέντρα έκφρασεων επιτρέπουν συνέπεια σε ερωτήματα πολλαπλών πηγών
- Ευρύ φάσμα των default operators, ενώ μπορούν εύκολα να προστεθούν και άλλα για LINQ to Objects
- Τα της χαρακτηριστικά της γλώσσα που αρχικά ήταν κατά κύριο λόγο για τη LINQ, τώρα εφαρμόζονται ευρέως και αλλού (lambda expressions)

Μειονεκτήματα

- Πάντα θα υπάρχουν κάποια πράγματα που μπορούν να γίνουν στην SQL αλλά όχι σε LINQ
- Επαναλαμβανομενοι κόμβοι (πλεονασμος δεδομένων)
- Κάποιοι τελεστές ερωτημάτων είναι ελλιπείς, ιδιαίτερα τα ισοδύναμα OrderBy - π.χ. εύρεση του στοιχείου με τη μέγιστη τιμή ενός ακινήτου

- Μικρά σύνολα δεδομένων χρειάζονται περισσότερο χρόνο για να κάνουν build το ερώτημα από το εκτελέσουν
- Ο εντοπισμός σφαλμάτων μπορεί να είναι πολύ δύσκολος

3.2 LINQ to XML

Στο κεφάλαιο αυτό παρουσιάζονται μερικές πτυχές της γλώσσας LINQ που χρησιμοποιήθηκαν για τη πτυχιακη. Το πρότυπο LINQ είναι μια προσθήκη στη .NET (προσθέτει περισσότερα αρχεία DLL βασικά) που επιτρέπει στον προγραμματιστή να θέτει ερωτήματα στα δεδομένα, όπως θα μπορούσε να χρησιμοποιηθεί για να κάνει με την τυπική σύνταξη SQL.

Έτσι, μπορεί να υπήρχε ένα ερώτημα σε μια βάση δεδομένων ή ένα ερώτημα SQL, όπως αυτό:

```
1SELECT * from Books WHERE QuatityInStock > 50 AND
    Price > 50.00
```

το παρακάτω είναι ισοδύναμο ως έγκυρο ερώτημα της LINQ to XML:

```
1var result =
2from b Books
3where b.QuatityInStock > 50 AND Price > 50.00
4select b;
```

Παραττηρούμε πόσο παρόμοια είναι αυτά τα δύο ερωτήματα και είναι εύκολα αντιληπτό ποσο ισχυρή γλώσσα είναι. Έτσι, αυτό είναι βασικά αυτό που η LINQ μας επιτρέπει να κάνουμε. Η LINQ εισάγει επίσης πολλές έννοιες που προέρχονται πρωτογενώς από άλλες λειτουργικές γλώσσες προγραμματισμού, όπως η Haskell, LISP. Μερικές από αυτές τις νέες έννοιες είναι Αναδρομική επεξεργασία σε μια ακολουθία, ανώνυμες συνάρτησεις (μέθοδοι στην NET) που μπορούν να κληθούν σε μια ακολουθία.

Τελεστές στην LINQ

Η LINQ έχει μια μεγάλη ποικιλία απο τελεστές (Standard Query Operators). Αυτοί είναι:

- Restriction operators
- Projection operators
- Partitioning operators
- Join operators
- Concatenation operator
- Ordering operators

- Grouping operators
- Set operators
- Conversion operators
- Equality operator
- Element operators
- Generation operators
- Quantifiers
- Aggregate operators

3.2.1 Τελεστής Περιορισμου-Restriction (WHERE)

Ο χειριστής Περιορισμού φιλτράρει μια ακολουθία βασίσμένη σε ένα κατηγόρημα. Στην επίσημη σημασιολογία ένα κατηγόρημα είναι μια έκφραση του σημασιολογικού τύπου του σετ. Ένας ισοδύναμος όρος είναι ότι θεωρείται ως λειτουργία-δείκτης των συνόλων, δηλαδή τις λειτουργίες από μια οντότητα σε μια τιμή αλήθειας (bool).

Στη λογική των κατηγορημάτων, ένα κατηγόρημα μπορεί να πάρει το ρόλο είτε ως ιδιοκτησία (property) ή μιας σχέσης μεταξύ των οντοτήτων.

Η επίσημη **LINQ** σημειολογία αναφέρει:
"Το παρακάτω παράδειγμα δηλώνει μια τοπική μεταβλητή κατηγόρημα που λαμβάνει ένας Customer και επιστρέφει bool. Στη τοπική μεταβλητή έχει εκχωρηθεί μια ανώνυμη μέθοδος που επιστρέφει true αν ο συγκεκριμένος πελάτης βρίσκεται στο Λονδίνο. Το κατηγόρημα χρησιμοποιείται στη συνέχεια για να βρούμε όλους τους πελάτες στο Λονδίνο."

```
1Func<Customer, bool> predicate = c => c.City == "
    London";
2IEnumerable<Customer> customersInLondon = customers
    .Where(predicate);
```

Ας εξετάσουμε αυτό το απλό παράδειγμα. Το τι θα καταλήξουμε είναι μια έκφραση που θα ήταν κάτι σαν:

```
1IEnumerable<Customer> customersInLondon = customers
    .Where(c => c.City
2== "London");
```

Που θα μπορούσε επίσης να είναι γραμμένο σε έναν πιο συμβατικό τρόπο (δηλαδή εκείνων που είναι πιο εξοικειωμένοι με τον SQL θα ήθελαν πιθανώς):

```
1IEnumerable<Customer> customersInLondon =
2from c in customers
3where c.City == "London"
4select c;
```

Ένας ακόμη πιο απλουστευμένος τρόπος είναι ο κάτωθι:

```
IEnumerable<Customer> customersInLondon =
from c in customers
where c.City == "London"
select c;
```

Όπου μπορει να γίνει:

```
var customersInLondon =
from t in customers
where it.City == "London"
select it;
```

Αυτός είναι ο τρόπος για το πώς η LINQ χρησιμοποιεί τα κατηγορήματα. Βασικά ο ευκολότερος τρόπος σκέψης για το τι είναι κατηγορήματα είναι να τα παρομοιάσουμε ως φίλτρα, που θα επιστρέψουν True ή False, και ως εκ τούτου θα "φιλτράρει" το IEnumerable <T> πηγή δεδομένων ώστε η έκφραση που εφαρμόζεται να περιέχει μόνο τα στοιχεία που ταιριάζουν με το φίλτρο (κατηγόρημα).

Μεταβλητές var

Ο κυριότερος λόγος για μεταβλητή τύπου var είναι ότι η LINQ επιτρέπει να δημιουργήσουμε εξ ολοκλήρου νέων τύπων δεδομένων "έργο", χωρίς να χρειάζεται να δημιουργήσουμε οι ίδιοι τον νεο αυτό τύπο. Αυτό είναι γνωστό ως "projection".
Παράδειγμα 1

```
IEnumerable<Item> iSmall =
from it in _itemList
where it.UnitPrice < 50.00M
select it;
\\
```

Παραδειγμα 2

```
var iSmall =
from it in _itemList
where it.UnitPrice < 50.00M
select it;
```

Στο παράδειγμα 2 παραπάνω, μπορούμε απλά να χρησιμοποιήσετε var αντί να εκχωρήσουμε το αποτέλεσμα του ερωτήματος. Αυτό λειτουργεί, ακόμη και για τα πιο περίπλοκα αποτελέσματα ενός ερωτήματος. Επίσης, αν αλλάξουμε το ερώτημα, δεν χρείαζεται να αλλάξουμε το var, καθώς προσαρμόζεται στους νέους απαιτούμενους τύπους αυτόματα. Η μεταβλητή τύπου var μπορεί να εξοικονομήσει χρόνο και να αποδειχθεί πολύτιμη.

Τελεστής Προβολής-Projection (SELECT)

Σε μια έκφραση ερωτήματων(query), το select clause καθορίζει το είδος των τιμών που θα παραχθούν όταν εκτελεστεί το ερώτημα. Το αποτέλεσμα βασίζεται στην αξιολόγηση όλων των προηγούμενων και για τυχόν

εκφράσεις στον ίδιο τον όρο select. Το select πρεπει να τερματίζει με τον τελεστή select ή group.

```
List<int> Scores = new List<int>() { 97, 92, 81, 60
    };
IEnumerable<int> queryHighScores =
from score in Scores
where score > 80
select score;
```

Ο τύπος της ακολουθίας που παράγεται από τον όρο select καθορίζει το αποτέλεσμα του ερώτηματος της μεταβλητής queryHighScores. Στην απλούστερη περίπτωση το select clause απλα καθορίζει το εύρος μιας μεταβλητής.

Τελεστής Κατάτμησης-Partition (TAKE/SKIP)

Ο τελεστής επιστρέφει ενα αποτέλεσμα σύμφωνα με ορισμένα κριτήρια. Το ακόλουθο παράδειγμα δημιουργεί μια ακολουθία από τα 10 πιο ακριβά προϊόντα:

```
IEnumerable<Product> MostExpensive10 =
products.OrderByDescending(p => p.UnitPrice)
.Take(10);
```

Ενώ ο τελεστής SKIP, παραλείπει ένα δεδομένο αριθμό στοιχείων από μια ακολουθία και στη συνέχεια αποδίδει το υπόλοιπο της ακολουθίας.

```
public static IEnumerable<TSource> Skip<TSource>(
this IEnumerable<TSource> source,
int count);
```

Τελεστής TAKEWHILE

Ο τελεστής TakeWhile σταματά όταν η συνθήκη είναι ψευδής, όπου συνεχίζει και να βρει όλα τα στοιχεία που ταιριάζουν στην συνθήκη. Παράδειγμα σε σύγκριση με το where:

```
var intList = new int[] { 1, 2, 3, 4, 5, -1, -2 };
Console.WriteLine("Where");
foreach (var i in intList.Where(x => x <= 3))
Console.WriteLine(i);
Console.WriteLine("TakeWhile");
foreach (var i in intList.TakeWhile(x => x <= 3))
Console.WriteLine(i);
```

Δίνει

```
Where
1
2
3
-1
-2
```

```
TakeWhile
1
2
3
```

Τελεστής Ένωσης (JOIN)

Παράδειγμα join δύο ακολουθιών:

```
var customersxml = @"<Customers>
<Customer CustomerID='alc'>Alice</Customer>
</Customers>";

var ordersxml = @"<Orders>
<Order OrderID='001' CID='alc'>apple</Order>
</Orders>";
```

```
1 var result = new XElement("Result",
2 from customer in
3 XElement.Parse(customersxml).
4 Elements("Customer")
5 join order in XElement.Parse(ordersxml).
6 Elements("Order")
7 on (string)customer.Attribute("CustomerID")
8 equals (string)order.Attribute("CID")
9
10 select new XElement("Join", customer, order));
```

Δίνει έξοδο το:

```
<Result>
<Join>
<Customer CustomerID="alc">Alice</Customer>
<Order OrderID="001" CID="alc">apple</Order>
</Join>
</Result>
```

Τελεστής Σύνδεσης-Concatenate (CONCAT)

Ο Concat τελεστής χρησιμοποιείται για να ενώσει δύο ακολουθίες.

```
1 IQueryable<String> custQuery =
2 (from cust in db.Customers
3 select cust.Phone)
4 .Concat
5 (from cust in db.Customers
6 select cust.Fax)
7 .Concat
8 (from emp in db.Employees
9 select emp.HomePhone);
```

Τελεστής Σειράς-Order (ORDERBY/THENBY)

Η οικογένεια OrderBy / ThenBy των τελεστών order τοποθετεί σε σειρά μια ακολουθία σύμφωνα με ένα ή περισσότερα κριτήρια.

```
var thenByResult = studentList.OrderBy(s => s.
    StudentName)
.ThenBy(s => s.Age);
```

Το παραπάνω ταξινομεί τη StudentList βάσει του ονόματος και κατόπιν βάσει της ηλικίας.

```
var thenByDescResult = studentList.OrderBy(s => s.
    StudentName).
ThenByDescending(s => s.Age);
```

Το παραπάνω ταξινομεί τη StudentList βάσει του ονόματος και μετά βάσει φθίνουσας σειράς της ηλικίας.

Τελεστής Ομαδοποίησης-Group (GROUPBY)

Ο τελεστής GroupBy ομαδοποιεί τα στοιχεία μιας ακολουθίας βασισμένος σε συγκεκριμένα κριτήρια.

```
var itemNamesByCategory =
from i in _itemList
group i by i.Category into g
select new { Category = g.Key, Items = g };
```

Αυτό το παράδειγμα παίρνει όλα τα αντικείμενα του τύπου Item και απλά τα ομαδοποιεί βάσει Κατηγορίας.

Τελεστές (Distinct / Union / Intersect / Except)

Ο τελεστής Distinct (Διάκρισης) εξαλείφει διπλά στοιχεία από μια ακολουθία. Για παράδειγμα αν υπάρχουν τέσσερα ονόματα σε έναν πίνακα Γιάννης, Πέτρος, Γιάννης, Κώστας, η εντολή αυτή θα βγάλει ώς έξοδο το Γιάννης, Πέτρος, Κώστας.

```
var itemCategory = (from i in _itemList
select i.Category)
.Distinct()
```

Σε αυτό το παράδειγμα παίρνει το μοναδικό όνομα των Κατηγοριών για το Items.

Ο τελεστής Union(Ένωση) παράγει το σύνολο ένωση των δύο ακολουθιών

```
var  unn = (from i in _itemList select i.ItemName)
.Distinct()
.Union((from o in _orderList select o.OrderName)
.Distinct());
```

Όπου το παράδειγμα αυτό παίρνει μια μοναδική λίστα τύπου List όλων των ItemName και, στη συνέχεια, συνενώνει αυτό το αποτέλεσμα, με έναν μοναδική λίστα List όλων των orderName.

Ο τελεστής Intersect(Τομή) παράγει το σύνολο τομής των δύο ακολουθιών.

```
var inter = (from i in _itemList select i.ItemID)
.Distinct()
.Intersect((from o in _orderList select o.OrderID)
.Distinct());
```

Όπου το παράδειγμα αυτό παίρνει μια μοναδική λίστα τύπου List όλων των Itemid και στη συνέχεια κάνει τομή αυτό το αποτέλεσμα, με μία μοναδική List όλων των τιμών OrderID. Το αποτέλεσμα είναι μια List των ακεράιων που είναι κοινά τόσο στο Item όσο και στο Order.

Ο τελεστής Except (Εξαίρεση) παράγει το σύνολο διαφοράς μεταξύ των δύο ακολουθιών.

```
var inter = (from i in _itemList select i.ItemID)
.Distinct()
.Intersect((from o in _orderList select o.OrderID)
.Distinct());
```

Όπου το παράδειγμα αυτό παίρνει μια μοναδική λίστα τύπου List όλων των τιμών ItemID και μετά τέμνει το αποτέλεσμα με μια μοναδική λίστα List απο όλες τις τιμές OrderId. Το αποτέλεσμα είναι μια λίστα List απο ακεραίους που είναι κοινές με τις λίστες Item και το Order.

Τελεστές Μετατροπής (ToSequence / ToArray / ToList/)

Οι τελεστές Set είναι οι παρακάτω:

- ToSequence. Ο τελεστής Ακολουθίας επιστρέφει το όρισμα σαν IEnumerable <T>.

```
public static IEnumerable<T> ToSequence<T>(
this IEnumerable<T> source);
```

Παράδειγμα:

```
var query = _itemList.ToSequence()
.Where(i => i.Category.Equals(
"Entertainment"));
```

Όπου στο παράδειγμα αυτό πάιρνει μια List και τη μετατρέπει σε μια ακολουθία ενω ανακτά όλα τα στοιχεία των οποίων η κατηγορία (Category) είναι Entairtainment.

- ToArray. Ο τελεστής αυτός δημιουργεί έναν πίνακα απο μία ακολουθία

```
var query =_itemList.ToArray()
.Where(i => i.Category.Equals("Food"));
```

Όπου στο παράδειγμα παίρνει μια List απο Items κατόπιν τη μετατρέπει σε έναν πίνακα, και επειτα ανακτά όλα τα στοιχεία των οποίων η κατηγορία (Category) είναι Food.

- ToList Ο τελεστης αυτός δημιουργεί μια List<T> απο μία ακολουθία.

```
var query = _itemList.ToList()
.Where(i => i.ItemID > 5).Reverse();
```

Όπου στο παράδειγμα παίρνει μια List απο Items και τη μετατρέπει σε List και επειτα ανακτά όλα τα στοιχεία των οποίων το ItemID είναι μεγαλύτερο του 5 και το αντιστρέφει, έτσι οι μεγαλύτεροι αριθμοί ξεκινούν από την αρχή.

- ToDictionary. Ο τελεστής αυτός δημιουργεί ένα Dictionary<K,E> απο μία ακολουθία.

```
var scoreRecords = new [] { new {Name = "Alice",
    Score = 50},
new {Name = "Bob" , Score = 40},
new {Name = "Cathy", Score = 45}
};
var scoreRecordsDict = scoreRecords
.ToDictionary(sr => sr.Name);
```

Όπου στο παράδειγμα δημιουργεί ένα νέο 2D πίνακα και τον μετατρέπει σε ένα κλειδί/τιμή ζευγαριού του αντικειμένου Dictionary.

- Equal Τελεστής. Ο τελεστής αυτός ελέγχει αν δύο ακολουθίες είναι ίσες.

```
int[] scoreRecords1 = new [] { 10,20 };
int[] scoreRecords2 = new [] { 10,20 };

var eq2 = scoreRecords1.EqualAll(scoreRecords2);
```

Στο παράδειγμα δημιουργεί δύο νέους πίνακες, οι οποίοι έχουν τα ίδια στοιχεία, και ως τέτοια, συγκρίνονται με τη χρήση της EqualAll και θεωρούνται ότι είναι ίσοι μεταξύ τους.

Τελεστές Στοιχείων (First / FirstOrDefault /)

- Ο τελεστής First επιστρέφει το πρώτο στοιχείο μιας ακολουθίας.

Ο τελεστής First απαριθμεί την ακολουθία πηγής και επιστρέφει το πρώτο στοιχείο για το οποίο επιστρέφει true. Εάν δεν έχει καθοριστεί, η First απλώς επιστρέφει το πρώτο στοιχείο της ακολουθίας.

```
1string itemName = "Lord Of The Rings DVD";
2Item itm = _itemList
3.First(i => i.ItemName == itemName);
```

Στο παράδειγμα πάιρνει το πρώτο στοιχείο της List από ένα Item που ταιρίαζει στο κατηγόρημα i.ItemName where itemName="Lord Of The Rings DVD".

- FirstOrDefault όπως και στη προηγούμενη περίπτωση επιστρέφει το πρώτο στοιχείο αλλά σε περίπτωση που δεν βρεθεί, επιστρέφει την προεπιλεγμένη τιμή.

```
1string itemName = "A Non existence Element";
2Item itm = _itemList
3.FirstOrDefault(i => i.ItemName == itemName);
```

Δηλαδή εδώ παίρνει το πρώτο ή προεπιλεγμένο στοιχείο που ταιρίαζει με τη συνθήκη. Στο συγκεκριμένο παράδειγμα το itemName δεν ταιρίαζει οπότε επιστρέφει null.

- Last Ο τελεστής Last επιστρέφει το τελευταίο στοιχείο μιας ακολουθίας ακριβώς όπως ο Φιρστ μόνο που δεν είναι το πρώτο αλλα το τελευταίο.

.

- ElementAt Ο τελεστής αυτός επιστρέφει το στοιχείο σε ενα δωσμένο index(δείκτη)σε μία ακολουθία

```
1Item thirdMostExpensive =
2_itemList.OrderByDescending(i => i.UnitPrice)
3.ElementAt(2);
```

Στο παράδειγμα επιστρέφει το τρίτο ακριβότερο στοιχείο των παραγγελιών(Η αρίθμηση ξεκινά απο το 0,1,2,3 όπως και στους πίνακες)

- ElementAtOrDefault Ακριβώς όπως το παραπάνω μόνο που αν δεν υπάρχει στοιχείο υπαρκτό με τη τιμή, επιστρεφει null

Τελεστές ποσότητας (Any / All / Contains)

Οι τελεστές αυτοί χωρίζονται σε τρεις:

- Any Ο τελεστης αυτός ελέγχει εναν οποιοδήποτε στοιχείο απο μια ακολουθία ικανοποιέι μια συνθήκη

```
bool b = _itemList
.Any(i => i.UnitPrice >= 400);
```

Όπου στο παράδειγμα απλά επιστρέφει bool αν υπάρχει οποιδήποτε Item στοιχειο με UnitPrice μεγαλύτερη του 400.

- All Οπως παραπάνω ο τελεστής αυτός ελέγχει, αντι για ένα, αν όλα τα στοιχεία μιας ακολουθίας ικανοποιούν μια συνθήκη.

- Contains Ο τελεστής αυτός κάνει έλεγχο για το άν μια ακολουθία περιέρχει ένα συγκεκριμένο στοιχείο.

```
bool b = _itemList
.Contains(_itemList[0]);
```

Όπου στο παράδειγμα απλά επιστρέφει bool αληθή τιμή αν η ItemList[0] υπάρχει.

Τελεστές Συνόλου (Count / LongCount / Sum /)

- Count Ο τελεστής αυτός μετράει (απαριθμεί) τα στοιχεία μιας ακολουθίας.

```
var foodCat = (from i in _itemList
where i.Category.Equals("Food")
select i).Count();
```

Όπου στο παράδειγμα απλά επιστρέφεται το μέγεθος (πόσα είναι) τα φαγητά κατηγορίας Food.

- LongCount Αυτός ο τελεστής λειτουργει το ίδιο με τον παραπάνω,μόνο που επιστρέφει μεγάλο αριθμό (long)

- Sum Αυτός ο τελεστής υπολογίζει το άθροισμα sum απο μια ακολουθία αριθμητικών στοιχείων.

```
var totals = (from i in _itemList select i.
    UnitPrice)
.Sum();
```

Στο παράδειγμα αθροίζονται όλα τα στοιχεία των UnitPrice.

- Min Αυτός ο τελεστής υπολογίζει το μικρότερο στοιχείο απο μια ακολουθία αριθμητικών στοιχείων.

```
var minimum = (from i in _itemList select i.
    UnitPrice)
.Min();
```

Όπου στο παράδειγμα αυτό αναζητεί και ανακτά τη μικρότερη UnitPrice τιμή.

- Min Αυτός ο τελεστής υπολογίζει ομοίως με παραπανω, το μεγαλύτερο στοιχείο απο μια ακολουθία αριθμητικών στοιχείων.
- Aggregate

Παράδειγμα 1

```
var num[] = new[]{1,2,3,4}
var sum = nums.Aggregate((a,b)=> a+b);
//eksodos (1+2+3+4)
```

Στο παράδειγμα προσθέτει 1 και 2 για να γίνει 3. Στη συνέχεια προσθέτει 3 (αποτέλεσμα του προηγούμενου) και 3 (επόμενο στοιχείο στη σειρά) να γίνει 6. Στη συνέχεια, προσθέτίθεται το 6 και 4 για σαν αποτέλεσμα γίνεται 10.

Παράδειγμα 2

```
var chars = new []{"a","b","c", "d"};
var csv = chars.Aggregate( (a,b) => a + ',' + b);
//eksodos a,b,c,d
```

Αυτό λειτουργεί με τον ίδιο τρόπο. Συνενώνεται με το το a και β να κάνει a, b. Στη συνέχεια συνενώνονται a, b με ένα κόμμα και c για να γίνει εντέλει b, c. και ούτω καθεξής.

- Fold Η fold μας επιτρέπει να φτίαξουμε μια δική μας συνάρτηση με στοιχεία.

```
double[] doubles = { 2,4,6,8,10 };
double product = doubles.Fold((runningProduct,
nextFactor) => runningProduct * nextFactor);
```

Στο παράδειγμα έχουμε έναν πίνακα που θέλουμε να πάρουμε το προϊον. Μπορούμε απλά να χρησιμοποιήσουμε τη fold για να ”στριμώξουμε” μια ένθετη συνάρτηση(runningProduct * nextFactor) ώστε να γίνει ο υπολογισμός μας.

Δυναμικά LINQ ερωτήματα

Στα παραπάνω είδαμε ότι θέταμε εμείς τις μεταβλητές και κατόπιν επεξεργαζόμασταν τα ερωτήματα. Στη πτυχιακή αυτή και στην εφαρμογή που αναπτύχθηκε, τα δεδομένα προέρχονται απο ένα User Interface που λαμβάνονται οι τιμές απο τον χρήστη.

Για παράδειγμα μπορούμε στη μετάβλητη var priceVar που μπορεί να προέρχεται απο ένα TextBox της γλώσσας C#, να γράψουμε

```
var iSmall = from it in _itemList
where it.UnitPrice < priceVal
select it;
```

Έτσι, με αυτό τον τρόπο, ελέγχουμε το έρωτημα.

IQuerable διεπαφή

Χρησιμοποιείται για την εξοικονόμηση πόρων όταν έχουμε μια απομακρυσμένη βάση δεδομένων. Δηλαδη στο παράδειγμα παρακάτω άν έχουμε έναν πίνακα Προϊόντα και θέλουμε να επιστρέψουμε αυτά που το κόστος είναι μεγαλύτερο από 25 δολλάρια σε μία απομακρυσμένη βάση δεδομένων:

Αν το κάνουμε με την IEnumerable

```
IEnumerable<Product> products = myDB.GetProducts()
    ;
var productsOver25 = products
.Where(p => p.Cost >= 25.00);
```

Η βάση φορτώνει όλα τα προϊόντα ενώ το λογισμικό στο τερματικό μας βγάζει το σχετικό αποτέλεσμα. Η μεγάλη διαφορά είναι ότι γίνεται ένα "ολόκληρο" SELECT * FROM που απο τη σκοπία της απόδοσης μπορεί να φιλτράρει 20.000 εγγραφές αντι για 2. Για το λόγο αυτό χρησιμοποιείται το κάτωθι

```
IQueryable<Product> products = myDB.
    GetQueryableProducts();
var productsOver25 = products.Where(p => p.Cost >=
    25.00);
```

Ο κώδικας μοιάζει ίδιος αλλα εκτελουνται υποχιλιαπλάσια λιγότερες λειτουργίες. Αυτός ειναι κ ο σκοπός της απόδοδης.

3.2.2 Προγραμματίζοντας με τη LINQ to XML

Η LINQ to XML έχει σχεδιαστεί για να είναι ένα ελαφρύ API προγραμματισμού XML. Αυτό ισχύει τόσο από εννοιολογική άποψη, δίνοντας έμφαση σε ένα απλό, εύκολο στη χρήση μοντέλο προγραμματισμού, και από τη σκοπιά της μνήμης και απόδοσης.

Κατά τη δημιουργία δέντρου, και κατ' αντιστοιχία στο W3C DOM, όταν δημιουργούμε ένα δέντρο XML, έχουμε δημιουργήσει το δέντρο XML με ένα τρόπο από κάτω προς τα πάνω. Για παράδειγμα, χρησιμοποιώντας XmlDocument (η εφαρμογή DOM από τη Microsoft), το παρακάτω θα ήταν ένας τυπικός τρόπος για να δημιουργήσουμε ένα δέντρο XML:

```
XmlDocument doc = new XmlDocument();
XmlElement name = doc.CreateElement("name");
name.InnerText = "Patrick Hines";
XmlElement phone1 = doc.CreateElement("phone");
phone1.SetAttribute("type", "home");
```

```
phone1.InnerText = "206-555-0144";
XmlElement phone2 = doc.CreateElement("phone");
phone2.SetAttribute("type", "work");
phone2.InnerText = "425-555-0145";
XmlElement street1 = doc.CreateElement("street1");
street1.InnerText = "123 Main St";
XmlElement city = doc.CreateElement("city");
city.InnerText = "Mercer Island";
XmlElement state = doc.CreateElement("state");
state.InnerText = "WA";
XmlElement postal = doc.CreateElement("postal");
postal.InnerText = "68042";
XmlElement address = doc.CreateElement("address");
address.AppendChild(street1);
address.AppendChild(city);
address.AppendChild(state);
address.AppendChild(postal);
XmlElement contact = doc.CreateElement("contact");
contact.AppendChild(name);
contact.AppendChild(phone1);
contact.AppendChild(phone2);
contact.AppendChild(address);
XmlElement contacts = doc.CreateElement("contacts")
    ;
contacts.AppendChild(contact);
doc.AppendChild(contacts);
```

Παρατηρούμε ότι το contacts ξεκινάει απο κάτω πρός τα επάνω.

Σε αντίθεση με το πρότυπο W3C, η LINQ to XML χρησιμοποιεί άλλο τρόπο για την κατασκεύη δέντρου XML. Η LINQ to XML υποστηρίζει αυτή την προσέγγιση για την κατασκευή ενός δέντρου XML, αλλά υποστηρίζει επίσης μια εναλλακτική προσέγγιση που αναφέρεται ως functional construction (συναρτησιακή κατασκευή). Παρακάτω φαίνεται το πώς θα κατασκευάστει το ίδιο δέντρο XML με τη χρήση LINQ to XML:

```
1XElement contacts =
2new XElement("contacts",
3new XElement("contact",
4new XElement("name", "Patrick Hines"),
5new XElement("phone", "206-555-0144",
6new XAttribute("type", "home")),
7new XElement("phone", "425-555-0145",
8new XAttribute("type", "work")),
9new XElement("address",
10new XElement("street1", "123 Main St"),
11new XElement("city", "Mercer Island"),
12new XElement("state", "WA"),
13new XElement("postal", "68042")))
14);
```

Παρατηρούμε ότι ο κωδικας είναι πιο συμπαγής για να κατασκευάστει το δέντρο XML και δείχνει την ευκολότερη δομή που παρέχει η LINQ to XML.
Το έγγραφο XML που βγάζει σαν έξοδο ο παρακάτω κώδικας είναι το κάτωθι:

```
<?xml version="1.0" standalone="yes"?>
<contacts>
<contact>
<name>Patrick Hines</name>
<phone>206-555-0144</phone>
<address>
<street1>123 Main St</street1>
<city>Mercer Island</city>
<state>WA</state>
<postal>68042</postal>
</address>
</contact>
</contacts>
```

Ονοματολογία XML

- Το έγγραφο αρχίζει με μια οδηγία επεξεργασίας: <xml ...?>. Αυτή είναι η δήλωση XML. Αν και δεν απαιτείται, η παρουσία του αναγνωρίζει ρητά το έγγραφο ως έγγραφο XML και υποδεικνύει την έκδοση της XML στο οποίο εχει γραφτεί.

- Δεν χρειάζεται η δήλωση του τύπου εγγράφου. Σε αντίθεση με SGML, XML δεν απαιτεί μια δήλωση τύπου εγγράφου. Ωστόσο, μια δήλωση τύπου εγγράφου μπορούν να παρέχονται, και ορισμένα έγγραφα θα απαιτήσει ένα προκειμένου να γίνει κατανοητό με σαφήνεια.

- Τα κενά elements (στοιχεία), έχουν βελτιωμένο συντακτικό. Δηλαδή αν έχουμε το στοιχείο <student> χωρίς την ετικέτα κλεισίματος <student/>, είναι έγκυρο να χρησιμοποιηθεί στην LINQ σε XML.

Σήμανση XML(markup)

Τα έγγραφα XML αποτελούνται από τη σήμανση(markup) και το περιεχομένο (content). Υπάρχστοιχεία, αναφορές οντότητα, σχόλια, οδηγίες επεξεργασίας, σημειώνονται τα τμήματα, και οι δηλώσεις τύπου εγγράφου. Οι ακόλουθες ενότητες εισάγουν κάθε μία από αυτές τις έννοιες σήμανσης:

- **Elements** Τα Στοιχεία είναι η πιο κοινή μορφή markup (σήμανσης). Το συντακτικό οριοθετείται από αγκύλες, όπου μέσα σ'αυτες προδιορίζεται η φύση του περιεχομένου που τις περιβάλλουν. Ορισμένα στοιχεία μπορεί να είναι άδεια, οπότε δεν έχουν περιεχόμενο. Εάν ένα στοιχείο δεν είναι κενό, αρχίζει με την έναρξη της ετικέτας, <element> και τελειώνει με μια ετικέτα τέλους, </ element>.

- **Attributes** Τα Χαρακτηριστικά είναι τα ζεύγη ονόματος-τιμής που συμβαίνουν στίς ετικέτες μετά το όνομα του στοιχείου. Για παράδειγμα,

```
<Div class= "content">
```

είναι ένα στοιχείο div με την κατηγορία χαρακτηριστικό που έχει την τιμή content. Στην XML, όλες οι τιμές χαρακτηριστικών πρέπει να αναφέρονται.

- **Entity References** Προκειμένου να εισαχθεί η σήμανση σε ένα έγγραφο, κάποιοι χαρακτήρες έχουν δεσμευτεί για να προσδιορίσουν την έναρξη της σήμανσης. Η αριστερή αγκύλη, <, για παράδειγμα, προσδιορίζει την αρχή ενός νέου στοιχείου ή ετικέτας τέλους. Για να εισαγάγουμε αυτούς τους χαρακτήρες στο έγγραφό μας ως περιεχόμενο, πρέπει να υπάρχει ένας εναλλακτικός τρόπος για να τους εκπροσωπήσει. Στην XML, οι οντότητες (entities) χρησιμοποιούνται για να αντιπροσωπεύσουν αυτούς τους ειδικούς χαρακτήρες.
Κάθε οντότητα θα πρέπει να έχει ένα μοναδικό όνομα. Για παράδειγμα, η οντότητα lt εισάγει ένα σύμβολο < σε ένα έγγραφο. Έτσι, το στοιχειο <element> μπορεί να αναπαρασταθεί σε ένα έγγραφο XML ως &lt?element>. < παράγει την αριστερή αγκύλη, < > παράγει την δεξια αγκύλη, > & παράγει το συμπλεκτικό σύμβολο & ' παράγει το εισαγωγικό σύμβολο (an apostrophe), ' " παράγει το διπλό εισαγωγικό σύμβολο,

- Comments Τα σχόλια αρχίζουν με <! - και τελειώνουν με ->. Σχόλια μπορούν να περιέχουν οποιοδήποτε κείμενο εκτός απο δεσμευμένες λέξεις. Μπορούμε να τοποθετήσουμε σχόλια μεταξύ tags οπουδήποτε στο έγγραφό μας. Τα σχόλια δεν αποτελούν μέρος του περιεχομένου του κώδικα ενός εγγράφου XML.

- CDATA Σε ένα έγγραφο XML, ένα τμήμα CDATA καθοδηγεί τον parser να αγνοήσει τις περισσότερες σήμανσης χαρακτήρες.

```
<![CDATA[

*p = &q;

b = (i <= 3);

]]>
```

Δηλαδή, σε έναν πηγαίο κώδικα σε ένα έγγραφο XML, θα μπορούσε να περιέχει χαρακτήρες που ο XML parser θα αναγνωρίσει κανονικά ως tag (<και , για παράδειγμα). Για να αποφευχθεί αυτό, μπορεί να χρησιμοποιηθεί ένα τμήμα CDATA.

Κλάση XName

Στη LINQ η κλάση που εκπροσωπεί τα ονόματα XML είναι η XName και βρίσκεται στο πακέτο System.Xml.Linq. Τα ονόματα XML εμφανίζονται συχνά σε όλο το LINQ API, και όπου απαιτείται το όνομα XML, υπάρχει μια παράμετρος XName. Ωστόσο, μπορείτε σπάνια συνεργάζονται άμεσα με XName. XName περιέχει μια σιωπηρή μετατροπή από κορδόνι Τα ονοματα XML, συχνά ένα σύνθετο θέμα σε APIs προγραμματισμού XML, που εκπροσωπούνται με απλό τρόπο στη LINQ to XML. Ένα όνομα XML αντιπροσωπεύεται από ένα χώρο ονομάτων XML (που αναφέρεται επίσης ως ένα XML namespace URL) και ένα τοπικό όνομα. Ένα XML namespace εξυπηρετεί τον ίδιο σκοπό ότι ένα namespace κάνει τα προγράμματά που είναι φτιαγμένα σε .NET Framework περιβάλλον, να μας επιτρέπει να δώσουμε μοναδικά ονόματα στις κλάσεις σας. Αυτό βοηθά στο να διασφαλιστεί ότι δεν θα υπάρξει διένεξη ονόματος με άλλους χρήστες. Για παράδειγμα, εάν θέλουμε να δημιουργήσουμε ένα στοιχείο XML με τα όνοματα επαφών, πιθανότατα θα το δημιουργήσουμε μέσα σε ένα XML namespace όπως http://myCompany.com/ContactList. Δηλαδή:

```
"{NamepaceURI}LocalName";
```

όπου αν δεν υπάρχει namespace μπορούμε να χρησιμοποιήσουμε τα ονόματα παρακάτω. Ένα όνομα με XML namespace είναι όπως το παρακάτω:

```
{http://myCompany.com}contacts
```

Και μπορούμε να το χρησιμοποιήσουμε:

```
XElement contacts = new XElement
("{http://myCompany.com}contacts",...
);
```

Ο κώδικας αυτός ισοδύναμος με:

```
XElement contacts = new XElement(
XName.Get("{http://myCompany.com}
contacts"), ...);
```

Ένας άλλος τρόπος που δεν χρησιμοποιούμε namespaces είναι:

```
string myNs = "{http://mycompany.com}";
```

και

```
XElement contacts =
new XElement(myNs+"contacts",
new XElement(myNs+"contact",
new XElement(myNs+"name", "Patrick Hines"),
new XElement(myNs+"phone", "206-555-0144",
new XAttribute("type", "home")),
new XElement(myNs+"phone", "425-555-0145",
new XAttribute("type", "work")),
new XElement(myNs+"address",
new XElement(myNs+"street1", "123 Main St"),
new XElement(myNs+"city", "Mercer Island"),
```

```
12new XElement(myNs+"state", "WA"),
13new XElement(myNs+"postal", "68042")
14)
15)
16);
```

το αποτέλεσμα του εγγράφου XML είναι:

```
<contacts xmlns="http://mycompany.com">
<contact>
<name>Patrick Hines</name>
<phone type="home">206-555-0144</phone>
<phone type="work">425-555-0145</phone>
<address>
<street1>123 Main St</street1>
<city>Mercer Island</city>
<state>WA</state>
<postal>68042</postal>
</address>
</contact>
</contacts>
```

Φόρτωση εγγράφου XML

Μπορείτε να φτιάξουμε τη δομή XML σε ένα δέντρο XML ώστε να μπορείτε ώστε να μπορεούμε να το χειριστούμε απευθείας. Η LINQ παρέχει πολλαπλές πηγές εισόδου, συμπεριλαμβανομένου εξωτερικού αρχείου, XmlReader ή TextReader, ή μέσω κειμένου.

- Εισαγωγή μέσω string : Μπορεούμε να χρησιμοποιήσουμε τη μέθοδο Parse. Εδώ είναι ένα παράδειγμα της μεθόδου Parse:

```
XElement contacts = XElement.Parse(
@"<contacts>
<contact>
<name>Patrick Hines</name>
<phone type=""home"">206-555-0144</phone>
<phone type=""work"">425-555-0145</phone>
<address>
<street1>123 Main St</street1>
<city>Mercer Island</city>
<state>WA</state>
<postal>68042</postal>
</address>
<netWorth>10</netWorth>
</contact>
</contacts)
```

- Φόρτώση απο εξωτερικό αρχείο με κατάληξη .xml

```
XElement contacts = XElement.Load(@"c:\contacts.xml
    ");
```

ή όταν βρίσκεται στο default directory

```
XElement contacts = XElement.Load("contacts.xml");
```

3.2.3 Διάσχιση XML δέντρων

Όταν έχουμε το XML διαθέσιμο στη μνήμη, το επόμενο βήμα είναι συχνά για να πλοηγηθείτε στα στοιχεία XML που θέλετε να εργαστείτε. Language Integrated Query παρέχει ισχυρές δυνατότητες για να κάνει ακριβώς αυτό, όπως περιγράφεται στο κεφάλαιο 3, "Υποβολή ερωτημάτων XML με XLinq", αυτή η ενότητα περιγράφει τις πιο παραδοσιακές προσεγγίσεις για το περπάτημα μέσα από ένα δέντρο XML.

Αναζητώντας κόμβους-παιδία

Η LINQ παρέχειπολλές μεθόδους για αναζήτηση παιδών του δέντρου. Εδώ μπορούμε να χρησιμοποιήσουμε την Content() μέθοδο. Για παράδειγμα αν φορτώσουμε στη μνήμη το ακόλουθο αρχείο επαφών:

```
<contact>
Met in 2005.
<name>Patrick Hines</name>
<phone>206-555-0144</phone>
<phone>425-555-0145</phone>
<--contacts-->
</contact>
```

και διασχίζοντας με την εντολή

```
1foreach (c in contact.Content()) {
2Console.WriteLine(c);
3}
```

έχουμε:

```
Met in 2005.
<name>Patrick Hines</name>
<phone>206-555-0144</phone>
<phone>425-555-0145</phone>
<--contacts-->
```

Το πρώτο παιδί ήταν το string "Met in 2005", το δεύτερο τρίτο και τέταρτο ήταν XElement name και phone αντίστοιχα, ενώ το τελευταίο ονομάζεται XComment. Παρατηρούμε ότι τα πάντα είναι δέντρα ακόμη και τα σχόλια.
Αν θέλουμε να γίνουμε πιο συγκεκριμένοι, μπορούμε να ζητήσουμε τα περιεχόμενα από εναν XElement συγκεκριμένο τύπο. Για παράδειγμα μπορεί να θελήσουμε να πάρουμε ένα παιδί XElement για την contact XElement επαφή μόνο. Γι'αυτη τη περίπτωση υποδηλώνουμε το παραμετροποιημένο τύπο:

```
foreach (c in contact.Content<XElement>()) {
Console.WriteLine(c)
}
```

(Η αντί για Content<XElement>() μπορούσαμε να γραψουμε σκέτο Elements())
Η διαφόρά με τα παραπάνω είναι οτι εμφανίζονται μόνο τα XElement παιδία:

```
<name>Patrick Hines</name>
<phone>206-555-0144</phone>
<phone>425-555-0145</phone>
```

Content(), Content<T>, Elements(), Elements(XName), and Element(XName),είναι οι βασικοί τύποι διάσχισης ενός XML δέντρου.

3.2.4 Χειρισμός XML δέντρου

Η LINQ παρέχει ένα πλήρες σύνολο μεθόδων για το χειρισμό XML. Μπορούμε να εισάγουμε, να διαγράψουμε, να αντιγράψουμε και να ενημερώσουμε το περιεχόμενο XML.
Σημείωση: σε όλες τις περιπτώσεις σώζουμε το XML δέντρο με την εντολή

```
contacts.Save("contacts.xml");
```

Εισαγωγή XML δέντρου

Μπορούμε εύκολα να είσαγουμε περιεχόμενο σε ένα υπάρχον XML δέντρο. Αυτό γίνεται με την εντολή Add. Δηλαδή:

```
XElement mobilePhone = new
XElement("phone", "206-555-0168");
contact.Add(mobilePhone);
```

Ετσι ο κόμβος phone τοποθετείται τελευταίο παιδί (από κάτω προς τα πάνω). Αν θέλουμε να προσθέσουμε το κόμβο σε συγκεκριμένο σημείο γραφουμε AddAfterThis() ή AddBeforeThis().
Δηλαδή σάν παράδειγμα αν μετα το mobilePhone παραπάνω θέλαμε να βάλουμε κ ένα cellPhone θα γράφαμε:

```
XElement mobilePhone = new XElement("phone", "
    1234567");
XElement firstPhone = contact.Element("phone");
firstPhone.AddAfterThis(mobilePhone);
```

Διαγραφή XML δέντρου

Για να σβήσουμε έναν XML κόμβο απλά πηγαίνουμε μέσω διάσχισης στο δέντρο και καλούμε την μέθοδο Remove(). Δηλαδή για να σβησουμε το πρώτο στοιχείο phone σε μια λίστα επαφών contact:

```
contact.Element("phone").Remove();
```

Επίσης μπορούμε να χρησιμοποιήσουμε και τη RemoveContent() όπως κάτωθι:

```
contacts.Element("contact")
.Element("address").RemoveContent();
```

Ενημέρωση XML δέντρου

Για να κάνουμε ενημέρωση διασχίζουμε το δέντρο και στα περιεχόμενα που θέλουμε να αντικαταστήσουμε χρσιμοποιούμε τη ReplaceContents() μέθοδο. Για παράδειγμα μπορούμε να κάνουμε το εξής για ένα στοιχείο address:

```
contact.Element("address")
.ReplaceContent(
new XElement("street", "123 Lane"),
new XElement("city", "London"),
new XElement("state", "CH"),
new XElement("country", "UK"),
new XElement("postalCode", "LN3E2")
);
```

Μπορούμε να χρησιμοποιήσουμε και τη SetElement() αντί της ReplaceContent().

Λάθη κατα το χειρισμό XML δέντρων

Πρέπει να έχουμε υπ'οψη ότι όταν χειριζόμαστε ένα ερώτημα XML δουλεύει μόνο σε αμεση πρόσβαση στο παιδί-φύλλο του από το να δοκιμάσουμε διασχιση μονομιάς. Για παράδειγμα θέλοντας να διαγράψουμε όλα στοιχεία απο μια contacts λίστα επαφών:

```
foreach (var phone in contacts
.Descendants("phone")
) {
phone.Remove();
}
```

Το παραπάνω θα εμφανίσει μήνυμα λάθους NullReferenceException. Πρέπει πάντα να χρησιμοποιούμε διασχίζοντας ολοκληρη την ακολουθία την συνάρτηση ToList() ή την ToArray(). Ετσι, στην προκειμένη περίπτωση, θα διασχίσει την λίστα και θα σβήσει όλα τα στοιχεία με όνομα phone.

```
foreach (var phone in contacts.
Descendants("phone").ToList()
) {
phone.Remove();
}
```

Το ίδιο ακριβώς συμβαίνει και με την συνάρτηση Remove().

3.3 XAttributes

Η προσθήκη ενός XAttribute είναι παρόμοια με την προσθήκη ενός απλού XElement. Στο παρακάτω XML, παρατηρούμε ότι κάθε αριθμός

τηλεφώνου έχει ένα χαρακτηριστικό τύπο που δηλώνει αν αυτό είναι ένα σπίτι, δουλειά, ή αριθμός κινητού τηλεφώνου:

```
<contacts>
<contact>
<name>Patrick</name>
<phone type="home">1234</phone>
<phone type="work">0123</phone>
</contact>
```

Ενώ δημιουργείται ως:

```
XElement contact =
new XElement("contact",
new XElement("name", "Patrick Hines"),
new XElement("phone",
new XAttribute("type", "home"),
"206-555-0144"
...
)
);
```

Ακριβώς όπως μπορούμε να χρησιμοποιήσουμε τη SetElement να ενημερώσουμε, να προσθέσουμε ή να διαγράψουμε στοιχεία με απλούς τύπους, μπορούμε να κάνουμε το ίδιο με τη μέθοδο της setAttribute (XName, αντικείμενο) σε XElement. Εάν υπάρχει το χαρακτηριστικό, θα πρέπει να ενημερωθεί. Εάν το χαρακτηριστικό δεν υπάρχει, θα προστεθεί.

3.3.1 Δουλεύοντας με άλλους τύπους στη LINQ

Η LINQ παρέχει ένα πλήρες σύνολο διαφορετικών τύπων κόμβων XML που εμφανίζονται στην XML. Για να φανεί αυτό, μπορούμε να δημιουργήσουμε ένα έγγραφο που χρησιμοποιεί το σύνολο των διαφορετικών τύπων κόμβου XML:

```
XDocument xdoc = new XDocument(
new XDeclaration("1.0", "UTF-8", "yes"),
new XDocumentType(),
new XProcessingInstruction("myApp", "My App Data"),
new XComment("My comment"),
new XElement("rootElement",
new XAttribute("myAttribute", "att"),
1234,
new XCData("Text with a <left> bracket"),
"mystring"
)
);
```

Ετσι, λαμβάνουμε ώς έξοδο το παρακάτω

```
<?xml version="1.0" standalone="yes"?>
<!--DOCTYPE-->
<?myApp My App Data?>
<!--My comment-->
```

```
<rootElement myAttribute="att">
1234<![CDATA[Text with a <left> bracket]]>mystring
</rootElement>
```

XLinq καθιστά εύκολο να διαχειριστει στοιχεία XML και χαρακτηριστικά, αλλά και άλλα είδη κόμβων XML είναι έτοιμα και διαθέσιμα όταν τα χρειαζόμαστε.

3.4 Ερωτήματα XML στη LINQ

Η μεγαλύτερη διαφοροποίηση της LINQ to XML και άλλα APIs προγραμματισμού XML είναι ότι είναι γλώσσα επερωτήσεων Language Integrated Query. Παρέχει συνεπή ερωτήματα ανάμεσα σε διαφορετικά μοντέλα δεδομένων σε ένα ενιαίο ερώτημα. Αυτή η ενότητα περιγράφει πώς να χρησιμοποιήσουμε τη Language Integrated Query με XML. Πέντε είναι οι κοινοί τελεστές ερωτημάτων στην LINQ:

- Where
- Select
- SelectMany
- OrderBy
- GroupBy

Η δημιουργία καθενός XElement με το Select Standard Query Operator λειτουργεί όπως θα περίμενε κανείς, όταν κάνει μια μετατραπεί σε XML, αλλά τι γίνεται αν χρειαστεί να δημιουργήσετε πολλαπλά στοιχεία μέσα στο ίδιο Select; Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να έχουμε πολλές contacts και τα στοιχεία επικοινωνίας απευθείας κάτω από τη ρίζα <contacts> και όχι βάσει μεμονωμένου <contact> στοιχείου. Σαν αυτό:

```
<contacts>
<!-- contact1 -->
<name>Patrick Hines</name>
<phone type="home">206-555-0144</phone>
<phone type="work">425-555-0145</phone>
<address>
<address>
<state>WA</state>
</address>
</address>
<!-- contact2 -->
<name>Gretchen Rivas</name>
<address>
<address>
<state>WA</state>
</address>
</address>
```

```
<!-- contact3 -->
<name>Scott MacDonald</name>
<phone type="home">925-555-0134</phone>
<phone type="mobile">425-555-0177</phone>
<address>
<address>
<state>CA</state>
</address>
</address>
</contacts>
```

Χειρίζοντας null σε μια ακολουθία

Όταν γράφουμε ένα μετασχηματισμό σε XML χρησιμοποιώντας συναρτησιακή κατασκευή, που ορισμένες φορές αντιμετωπίζουμε καταστάσεις όπου ένα στοιχείο είναι προαιρετικο, και δεν θέλουμε να δημιουργήσουμε κάποιο μέρος του XML εάν το στοιχείο δεν υπάρχει. Για παράδειγμα, το ακόλουθο είναι ένα ερώτημα που παίρνει τα names και τα phone numbers βάζοντας τους αριθμούς τηλεφώνου κάτω από ένα στοιχείο<phoneNumbers>.

```
1new XElement("contacts",
2from c in contacts.Elements("contact")
3select new XElement("contact",
4c.Element("name"),
5new XElement("phoneNumbers", c.Elements("phone"))
6)
7);
```

Αν η επαφή δεν έχει phone numbers, το στοιχείο θα συνεχίσει να υπάρχει αλλά δεν δεν υπάρχουν phone παιδιά στοιχεία. Για να λύσουμε αυτό το πρόβλημα χρησιμοποιούμε τον τελεστή Any (παρουσιάστηκε αναλυτικά στην ενότητα 3.2.1) Η γλώσσα LINQ δεν έχει πρόβλημα με κενά null στοιχεία, έτσι ο τελεστής Any μας επιτρέπει να παραλείψουμε το phoneNumber αν η συγκεκριμένη contact δεν έχει τηλεφωνικούς αριθμούς.

Συνάρτηση που επιστρέφει το XElement με την Any()

```
1static XElement GetPhoneNumbers(XElement c) {
2if (c.Elements("phone").Any())
3return
4new XElement("phoneNumbers", c.Elements("phone"));
5else
6return null;
7}
```

Επεκτάσεις ερωτημάτων

Οι XML συγκεκριμένες επεκτάσεις ερώτηματων παρέχει ερώτηματα που θα περίμενε κανείς όταν εργάζεται σε μια δομή δεδομένων δέντρου

XML. Αυτές οι XML συγκεκριμένες επεκτάσεις ερωτημάτων είναι ανάλογες με τους την XPath. Για παράδειγμα, η μέθοδος Elements είναι ισοδύναμη με το χειριστή XPath * (αστερίσκος). Οι ακόλουθες ενότητες περιγράφουν κάθε ένα από τις XML-συγκεκριμένες επεκτάσεις ερωτήματος με τη σειρά.

- Στοιχεία και περιεχόμενο Ο χειριστής ερώτηματος Elements επιστρέφει τα παιδιά για κάθε XElement σε μια ακολουθία XElements (IEnumerable <XElement>). Για παράδειγμα, για να πάρουμε τα στοιχεία παιδια για κάθε contact στη λίστα επαφών, μπορούμε να κάνουμε το εξής:

```
foreach (XElement x in contacts.Elements("contact")
.Elements())
{
Console.WriteLine(x);
}
```

Σημειώστε ότι οι δύο Elements() μέθοδοι σε αυτό το παράδειγμα είναι διαφορετικές, αν και κάνουν δύο ταυτόσημα πράγματα. Τα πρώτα Elements είναι καλώντας τους μέθοδο XElement μέθοδο Elements(), η οποία επιστρέφει ένα IEnumerable <XObject> και περιέχει τα στοιχεία του παιδιού στις XElement επαφές. Η δεύτερη μέθοδος Elements() ορίζεται ως μια μέθοδος επέκτασης για IEnumerable <XObject>. Επιστρέφει μια ακολουθία που περιέχει τα στοιχεία παιδιά του κάθε XElement στη λίστα. Τα αποτελέσματα της παραπάνω ερώτημα μοιάζουν με το παρακάτω:

```
<name>Patrick Hines</name>
<phone type="home">206-555-0144</phone>
<phone type="work">425-555-0145</phone>
<address>
<street1>123 Main St</street1>
<city>Mercer Island</city>
<state>WA</state>
<postal>68042</postal>
</address>
<netWorth>10</netWorth>
<name>Gretchen Rivas</name>
<phone type="mobile">206-232-4444</phone>
<address>
<street1>123 Main St</street1>
<city>Mercer Island</city>
<state>WA</state>
<postal>68042</postal>
</address>
<netWorth>11</netWorth>
<name>Scott MacDonald</name>
<phone type="home">925-555-0134</phone>
<phone type="mobile">425-555-0177</phone>
<address>
```

```
<street1>345 Stewart St</street1>
<city>Chatsworth</city>
<state>CA</state>
<postal>92345</postal>
</address>
<netWorth>500000</netWorth>
```

Εάν θέλουμε όλα τα παιδία με ένα συγκεκριμένο όνομα, μπορούμε να χρησιμοποιήσουμε το Elements(XName) overload. Για παράδειγμα:

```
1foreach (XElement x in contacts.Elements("contact")
    .Elements("phone"))
2{
3Console.WriteLine(x);
4}
```

Θα επεστρεφε:

```
<phone>206-555-0144</phone>
<phone>425-555-0145</phone>
<phone>925-555-0134</phone>
<phone>425-555-0177</phone>
```

- Descendants και Ancestors Οι τελεστές ερωτημάτων Descendants(απόγονοι) και Ancestors (προγονοι) μας επιτρέπουν να πραγματοποιούμε ερωτήματα κάτω και πάνω από το δέντρο XML, αντίστοιχα. Απόγονοι χωρίς παραμέτρους δίνουν το περιεχόμενο-παιδί ενός XElement και, στη συνέχεια, το περιεχόμενο του κάθε παιδιού κάτω των κόμβων φύλλων (η XML υποδέντρο). Προαιρετικά, μπορούμε να καθορίσουμε ενα XName (Descendants (XName)) και να ανακτήσουμε το σύνολο των απογόνων με ένα συγκεκριμένο όνομα, ή να καθορίσουμε έναν τύπο (Descendants <T>) και να ανακτήσουμε το σύνολο των απογόνων ενός συγκεκριμένου τύπου LINQ (για παράδειγμα, XComment).Για παράδειγμα, για να πάρουμε όλους τους τηλεφωνικούς αριθμούς απο την contact μας μπορούμε να κάνουμε το εξής:

```
contacts.Descendants("phone");
```

Οι Descendants και οι Ancestors δεν συμπεριλαμβάνουν τον τρέχοντα κόμβο. ΑΝ χρησιμοποιήσουμε την Descendants() στο στοιχείο ρίζας, θα πάρουμε όλο το XML δέντρο εκτός απο το στοιχείο ρίζας. Εαν θελήσουμε όλο το δέντρο και τη ρίζα μαζί θα πρέπει να χρησιμοποιήσουμε την SelfAndDescendants.

- Οι Ancestors(πρόγονοι) και οι SelfAndAncestors (ριζα και πρόγονοι) δουλεύουν με παρόμοιο τρόπο όπως οι Descendants και SelfAndDescendants, με τη διαφορά διασχίζουν το δέντρο απο πάνω αντί για προς τα κάτω. Για παράδειγμα, μπορούμε να ανακτήσουμε τον πρώτο αριθμό τηλεφώνου στο δέντρο contacts XML και, στη συνέχεια, να εκτυπώσουμε τους προγόνους του:

```
XElement phone = contacts.Descendants("phone")
.First();
foreach (XElement a in phone.Ancestors()) {
Console.WriteLine(a.Name);
};
```

Το αποτέλεσμα είναι:

```
contact
contacts
```

Αν εκτελέσουμε το ίδιο ερώτημα με την SelfAncestors η έξοδος θα δείξει επίσης το phone:

```
XElement phone = contacts.Descendants("phone")
.First();
foreach (XElement a in phone.SelfAndAncestors()) {
Console.WriteLine(a.Name);
};
```

Δηλαδή

```
Phone
contact
contacts
```

Οι συναρτήσεις ερωτημάτων Descendants και Ancestors μπορούν κατα πολύ να βελτιώσουν τη διάσχιση δέντρου στην XML.

Attributes

Οι ιδιότητες (Attributes) των XML ερωτημάτων καλούνται σε ένα IEnumerable<XElement> και επιστρέφουν μια ακολουθία από ιδιότητες (IEnumerable<XAttribute>). Προαιρετικά μπορούμε να καθορίσουμε ένα XName να επιστρέψουμε μόνο ιδιότητες με αυτό το όνομα. Για παράδειγμα μπορούμε να πάρουμε μια λίστα απο διακριτούς distinct τύπους τηλεφωνικών αριθμών που είναι στη λίστα επαφών contact:

```
contacts.Descendants("phone").
Attributes("type").
Select(t => t.Value)
.Distinct();
```

όπου θα επιστρέψει

```
home
work
mobile
```

3.4.1 ElementsBeforeThis, ElementsAfterThis, ContentBeforeThis, ContentAfterThis

Εάν βρισκόμαστε σε ένα συγκεκριμένο στοιχείο, και θέλουμε να ανακτήσουμε όλα τα στοιχεία του παιδιού ή του περιεχόμενου πριν από το

συγκεκριμένο αυτό στοιχείο ή τα στοιχεία του παιδιού ή το περιεχόμενο μετά από αυτό το συγκεκριμένο στοιχείο.Η ElementsBeforeThis επιστρέφει ένα IEnumerable <XElement> που περιέχει το στοιχείο αδελφό πριν από αυτό το στοιχείο. ElementsAfterThis επιστρέφει τα στοιχεία αδελφό μετά από αυτό το στοιχείο. Η επέκταση ContentBeforeThis επιστρέφει τα προηγούμενα αδέλφια οποιουδήποτε τύπου (π.χ., string, XComment, XElement, κλπ). Κατά συνέπεια, επιστρέφει ένα IEnumerable <object>. Ομοίως, ContentAfterThis επιστρέφει τα αδέλφια οποιουδήποτε τύπου.

Για παράδειγμα αν θέλουμε όλους τους αριθμούς τηλεφώνων απο την contacts:

```
IEnumerable<XElement> phones =
contacts.Elements("contact")
.Elements("phone");
```

Αυτό θα μπορούσε να επαναγραφτεί ως:

```
IEnumerable<XElement> phones =
XElementSequence.Elements
(contacts.Elements("contact"), "phone");
```

3.4.2 Μετασχηματισμός XML

Ο μετασχηματισμός είναι ένα πολύ σημαντικό σενάριο χρήσης στην XML. Είναι τόσο σημαντικό το γεγονός ότι είναι ένα κρίσιμο χαρακτηριστικό σε δύο βασικές τεχνολογίες της XML: XQuery και XSLT. Στην LINQ to XML, ο κύριος παράγοντας της μετατροπής αυτής σε XML είναι η λειτουργική κατασκευή (functional construction). Σε αυτή, μπορούμε να συμπληρώσουμε κομμάτια της XML με τη χρήση συνδυασμών των ερωτημάτων και των λειτουργιών, όπως απαιτείται. Για παράδειγμα, μπορούμε να μετατρέψουμε τη μορφή της λίστας επαφών σε μια λίστα πελατών. Αρχίζοντας αντιστροφα, με τον κατάλογο των πελατών βλέπούμε κάτι σαν αυτό:

```
<Customers>
<Customer>
<Name>Patrick Hines</Name>
<PhoneNumbers>
<Phone type="home">206-555-0144</Phone>
<Phone type="work">425-555-0145</Phone>
</PhoneNumbers>
</Customer>
</Customers>
```

Με το γνωστό τρόπο συναρτησιακής κατασκευής κατασκευάσαμε το XML

```
new XElement("Customers",
new XElement("Customer",
new XElement("Name", "Patrick Hines"),
new XElement("PhoneNumbers",
new XElement("Phone",
```

```
6new XAttribute("type", "home"),
7"206-232-2222"),
8new XElement("Phone",
9new XAttribute("type", "work"),
10"425-555-0145")
11)));
```

Για να μετατρέψουμε τη λίστα επαφών γράφουμε:

```
1new XElement("Customers",
2from c in contacts.Elements("contact")
3select new XElement("Customer",
4new XElement("Name", (string) c.Element("name")),
5new XElement("PhoneNumbers",
6from ph in c.Elements("phone")
7select new XElement("phone", (string) ph,
8ph.Attribute("type")
9))));
```

Παρατηρούμε πως ο μετασχηματισμός ευθυγραμμίζεται με τη δομή του εγγράφου προορισμού μας. Μπορούμε να ξεκινήσουμε με τη δημιουργία του εξωτερικού, στοιχείου ρίζα του XML εγγραφου-προορισμού

```
new XElement("Customers", ...
```

Θα χρειαστεί να δημιουργήσουμε ένα XElement Customer που αντιστοιχεί σε κάθε επαφή στο αρχικό μας XML. Για να το κάνουμε αυτό, θα ανακτήσουμε όλα τα στοιχεία contacts σύμφωνα με τις επαφές, γιατί θα πρέπει να επιλέξουμε με τη select για τη κάθε επαφή.

```
... from c in contacts.Elements("contact")...
```

Για κάθε contact εκτελείται η select μέσα στο loop

```
select new XElement("Customer",
```

Τώρα κατασκευάζουμε το <Customer> μέρος του XML προορισμού. Αρχίζουμε δημιουργώντας ένα XElement Customer

```
1select new XElement("Customer",
2new XElement("Name", (string) c.Element("name")),
```

Το παιδί <PhoneNumbers> είναι πιο περίπλοκο διότι οι αριθμοί τηλεφώνου στην λίστα επαφών απεκονείζονται απευθείας κάτω απο την επαφή

```
<contact><phone>...</phone><phone>...</phone></
    contact>
```

Για να υπερκεραστεί αυτό το εμπόδιο εκτελουμε ερώτημα στους αριθμούς τηλεφώνου και τους τοποθετούμε σαν παιδία κάτω απο το στοιχείο <PhoneNumbers>

```
1...
2new XElement("PhoneNumbers",
3from ph in c.Elements("phone")
4select new XElement("phone", (string) ph,
5ph.Attribute("type")
6))
```

Σε αυτόν τον κώδικα, εκτελούμε ερώτημα στους αριθμούς τηλεφώνου της επαφής, c.Elements ("phone"), για κάθε phone. Μπορούμε επίσης να δημιουργήσουμε ένα νέο XElement που ονομάζεται Phone με ίδιο τύπο χαρακτηριστικού όπως το αρχικό τηλεφώνο με την ίδια τιμή.

3.4.3 Χρησιμοποιώντας Εκφράσεις Ερωτημάτων

Το παρακάτω δείχνει μερικά απλά παραδείγματα για τη χρήση εκφράσεων ερώτηματων με την LINQ to XML. Αυτό το ερώτημα ανακτά όλες τις επαφές από το Λονδίνο, τις ταξινομεί με βάση το όνομα, και στη συνέχεια να επιστρέφει ως συμβολοσειρά (το αποτέλεσμα αυτού του ερωτήματος είναι IEnumerable <string>).

```
1from     c in contacts.Elements("contact")
2where    (string) c.Element("address").Element("
    state") == "LO"
3orderby  (string) c.Element("name")
4select   (string) c.Element("name");
```

Ενώ το παρακάτω ανακτά τις επαφές απο το Λονδίνο που έχουν κωδικό περιοχής 4420 ταξινομημένα κατ'όνομα. Το αποτέλεσμα αυτού του ερωτήματος είναι IEnumerable <XElement>

```
1from     c in contacts.Elements("contact"),
2ph in c.Elements("phone")
3where    (string) c.Element("address")
4.Element("state") == "LO" &&
5ph.Value.StartsWith("4420")
6orderby  (string) c.Element("name")
7select   c;
```

Εδώ είναι ένα άλλο παράδειγμα για ανάκτηση των επαφών που έχουν μια καθαρή αξία μεγαλύτερη από το μέσο όρο της καθαρής.

```
1from c in contacts.Elements("contact"),
2average = contacts.Elements("contact").
3Average(x => (int) x.Element    (""))
4where (int) c.Element("netWorth") > average
5select c;
```

3.4.4 Συνδυασμός XML και αλλων Μοντέλων Δεδομένων

Η LINQ παρέχει πολλές δυνατότητες συνδυασμού μεταξύ πολλών και διαφορετικών μοντέλων δεδομένων. Παρέχει επίσης την δυνατότητα να συνδυάσει και ταιριάξει LINQ μοντέλα δεδομένων και άλλα APIs μέσα σ'ενα ενιαίο ερώτημα. Αυτη η ενότητα παρουσιάζει δύο απλά παραδείγματα σεναριων χρήσης μεικτών μοντέλων.
Παρακάτω φαίνεται μια απλή δομή πελατών απο το Λονδίνο που βγαζει ως έξοδο ενα XML αρχείο.

```
XElement londonCustomers =
new XElement("Customers",
from c in db.Customers
where c.City == "London"
select new XElement("Customer",
new XAttribute("CustomerID", c.CustomerID),
new XElement("Name", c.ContactName),
new XElement("Phone", c.Phone)
));
```

Εξοδος:

```
<Customers>
<Customer CustomerID="AROUT">
<Name>Mark Harrington</Name>
<Phone>(171) 555-0188</Phone>
</Customer>
<Customer CustomerID="BSBEV">
<Name>Michelle Alexander</Name>
<Phone>(171) 555-0112</Phone>
</Customer>
<Customer CustomerID="CONSH">
<Name>Nicole Holliday</Name>
<Phone>(171) 555-0182</Phone>
</Customer>
<Customer CustomerID="EASTC">
<Name>Kim Ralls</Name>
<Phone>(171) 555-0197</Phone>
</Customer>
<Customer CustomerID="NORTS">
<Name>Scott Culp</Name>
<Phone>(171) 555-0173</Phone>
</Customer>
<Customer CustomerID="SEVES">
<Name>Deepak Kumar</Name>
<Phone>(171) 555-0117</Phone>
</Customer>
</Customers>
```

- Αναγνωση XML και ανανεωση της βασης: Μπορούμε επίσης αυτή τη πληροφορία να την εντάξουμε στην βάση μας. Στο εδώ παράδειγμα ας υποθέσουμε ότι παιρνουμε ενα σέτ απο updates πελατών στο γνωστό μας XML φορματ:

```
<customerUpdates>
<customerUpdate>
<custid>ALFKI</custid>
<phone>206-555-0103</phone>
</customerUpdate>
<customerUpdate>
<custid>EASTC</custid>
```

```
<phone>425-555-0143</phone>
</customerUpdate>
</customerUpdates>
```

Για να πραγματοποιήσουμε αυτη την ανανέωση στο XML έγγραφο μας, κάνουμε ερώτημα για κάθε ένα στοιχείο και καλούμε τη βάση να ανακτήσει την αντίστοιχη Customer εγγραφη. Επειτα, γίνεται το update στην Customer στήλη με έναν νέο αριθμό τηλεφώνου.

```
foreach (var cu in customerUpdates.Elements("
    customerUpdate")) {
Customer cust = db.Customers.
First(c => c.CustomerID == (string)cu.Element("
    custid"));
cust.Phone = (string)cu.Element("phone");
}
db.SubmitChanges();
```

3.4.5 Επίλογος

Σε αυτή την ενότητα παρουσιάστηκαν τα κυρίοτερα χαρακτηριστικά της LINQ με αναλυτικό τρόπο. Είδαμε ότι όσο εξοικειονόμαστε μπορούμε να αξιολογήσουμε τις δυνατότητες που αγγίζουν ή και ξεπερνούν σε ορισμένες περιπτώσεις τις παραδοσιακές βάσεις δεδομένων. Υπάρχει πλήθος βιβλιογραφίων στο διαδίκτυο. Ασφαλώς το καλύτερο ξεκίνημα είναι απο το επίσημο site της microsoft, LINQ to XML με πληθώρα παραδειγμάτων είναι αυτο:101 LINQ samples(πατηστε)

Κεφάλαιο 4

Εφαρμογή κρατήσεων Αεροπορικής Εταιρείας

Στο τελευταιο αυτό κεφάλαιο παρουσίαζεται μια απλή εφαρμογή κρατήσεων για μια αεροπορική εταιρεία, χρησιμοποιώντας τα ερωτήματα που είδαμε στα προηγούμενα κεφάλαια. Το γραφικό περιβάλλον έχει δημιουργηθεί σε .net 4.0 γλώσσα c# και στη σουίτα εφαρμογών Visual Studio 2015.

4.0.1 Ιστορία των Airline Reservation Systems

Η ιστορία των συστημάτων κρατήσεων αεροπορικών εταιρειών ξεκίνάει από τα τέλη της δεκαετίας του 1950, όταν η American Airlines απαιτούσε ένα σύστημα που θα επέτρέπε σε πραγματικό χρόνο πρόσβαση στα στοιχεία της πτήσης σε όλα τα γραφεία της, καθώς και την ενσωμάτωση και την αυτοματοποίηση των διαδικασιών κράτησης και έκδοσης εισιτηρίων της.

Ετσι, το πρώτο σύστημα ηλεκτρονικών κρατήσεων, Magnetronic Reservisor, εισήχθη το 1952. Πολλά χρόνια αργότερα, ένα νέο σύστημα, το Sabre (απο τα ακρωνύμια Semiautomatic Bussiness Research Environment) αναπτύχθηκε και ξεκίνησε το 1964. Το επαναστατικό στοιχείο του Sabre ήταν η

ικανότητά του να κρατήσει καταγραφή σε πραγματικό χρόνο, πρόσβιβάσιμη από πράκτορες σε όλο τον κόσμο. Πριν από αυτό, τα συστήματα δεν ήταν αυτόματα και απαιτούσαν κεντρικά κέντρα κράτησεων όπου ομάδες ανθρώπων σε ένα δωμάτιο κανόνιζαν τη διαχειρηση κρατήσεων μέσω καρτών, σε αυτή την περίπτωση, τα καθίσματα των αεροπλάνων.
Τις δεκαετίες 1960 και 1970 η αεροπορική βιομηχανία επινόησε και επένδυσε μεγάλη ερευνα σχετικά με το πώς θα βελτιώσουν τα Σύστηματα Κρατήσεων. Προς τα τέλη του 1970, οι αεροπορικές εταιρίες ήταν σε θέση να έχουν το δικο τους, ιδιωτικό σύστημα κρατήσεων εισιτηρίων. Η εταιρεία United Airlines ήταν υπεύθυνη για την ανάπτυξη του συστήματος κρατήσεων Apollo και η Delta για το Automated Accounts Systems (DATAS) το 1968 και λίγο μετά την ανάπτυξη του, οι ταξιδιωτικοι πράκτορες ήταν σε θέση να έχουν πρόσβαση και να κάνουν τη χρήση του συστήματος.

4.0.2 Ορίσμός ενός ARS

Πρίν προχωρήσουμε στον ορίσμο του Συστήματός Κρατήσεων ή αλλιως Computerized Reservation SYstem είναι απαραίτητο να επεξηγηθούν μερικοί όροι. Αυτοί είναι:

- Computerized Ενα σύστημα που διαχειρίζεται διεργασίες (process).
- Reservation Οπως υποδηλώνει το όνομα, είναι η τήρηση κράτησης πληροφοριών. Αυτό μπορεί να γίνει με τρόπους όπως εγγραφές (records) και δεδομένα. Είναι δηλαδή η εισαγωγή δεδομένων με οργανωμένη και συνεπή τρόπο, ώστε να γίνεται εύκολα και αποδοτικά η ανάκτηση τους.
- Σύστημα Είναι το υπεύθυνο κεντρικό σύστημα με τη μονάδα επεξεργασίας που μπορεί να χειρίζεται το σύνολο των μεθόδων και διαδικασιών ώστε να εκτελέσει μια λειτουργία, χωρίς πολλές φορές να χρειαστεί να παρέμβει ο ανθρωπος.

Το Airline Reservation System είναι ένα σύστημα που είναι ένα ηλεκτρονικό σύστημα τήρησης πληροφοριών που επιτρέπει στη πλευρά του χρηση να κάνει Αναζήτηση, Ανάκτηση και Κράτηση και απο την business πλευρά της αεροπορικής εταιρείας να συντονίζει τα δρομολόγια και να έχει έλεγχο των πληροφοριών των κρατήσεων και των πτήσεων.

4.0.3 Αρχιτεκτονικές Σχεδίασης ARS

Υπάρχουν διάφορες αρχιτεκτονικές σχεδίασης λογισμικού κρατήσεω. Οι δύο κυριότερες είναι:

- Υπηρεσιοστρεφείς(Service Oriented): Αυτός ο τύπος της αρχιτεκτονικής χρησιμοποιείται σε κατανεμημένα συστήματα οπου οι ταξιδιωτικοί πράκτορες αναφέρονται ως υπηρεσίες (services). Αυτό το είδος της αρχιτεκτόνικής συνήθως υλοποιείται με XML, WDSL(Web

Service Description Language), SOAP(Simple Access Object Protocol) και UDDI (Universal Description Discovery and Integration). Ετσι, οι χρήστες έχουν πρόσβαση στό web service από οπουδήποτε και μπορούν να κάνουν αναζήτηση για πτήσεις, χωρίς να χρειάζεται να πηγαίνουν στο αεροδρόμιο.

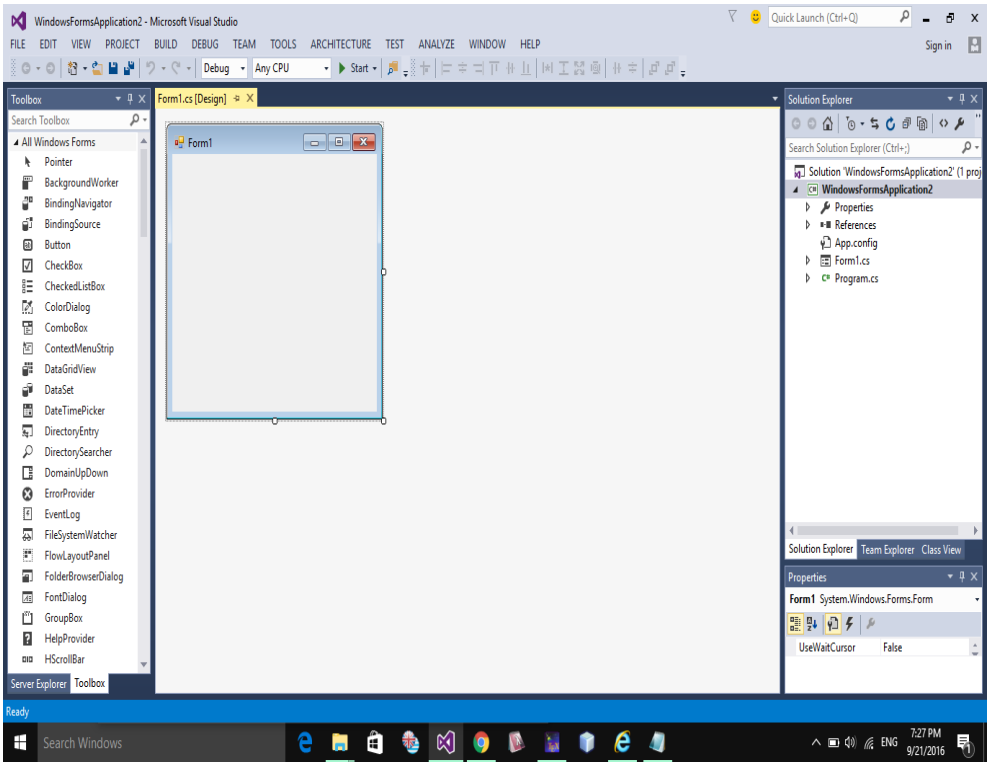
- Παραδοσιακή Λογισμικοστρεφή Υπηρεσία(Service Oriented): Αυτό το είδος της αρχιτεκτονικής είναι ο τύπος οπου περιλαμβάνει το λογισμικό μιας desktop εφαρμογής πράκτορα εισιτηρίων που δεν είναι συνδεδεμένο online: δέν έχει διαδικτυακή υπηρεσία-πρόσβαση και απαιτεί από τους πελάτες να μετακινηθούν προς το χώρο της έκδοσης εισιτηρίων (συνήθως κτίριο ή κιόσκι) για να κάνουμε κράτηση. Οι περισσότερες υπανάπτυκτες χώρες εξακολουθούν να κάνουν χρήση αυτής της αρχιτεκτονικής, αλλά αυτό απαιτεί πολύ χρόνο και προσπάθεια τόσο από τον εκδότη εισιτηρίων όσο και τον πελάτη, ενώ αντίθετα στην αρχιτεκτονική service oriented υπηρεσία μπορούμε να κάνουμε κράτηση πτήσης με ένα μόνο κλικ μακριά.

4.1 Παρουσίαση της εφαρμογής

Η εφαρμογή είναι φτιαγμένη με το λογισμικό Visual Studio. Το Microsoft Visual Studio είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) από τη Microsoft. Χρησιμοποιείται για την ανάπτυξη προγραμμάτων ηλεκτρονικών υπολογιστών για Windows, καθώς και για ιστοσελίδες, διαδικτυακές εφαρμογές και web services.Το Visual Studio χρησιμοποιεί πλατφόρμες ανάπτυξης λογισμικού της Microsoft, όπως τα Windows API, τα Windows Forms, το Windows Presentation Foundation, το Windows Store και το Microsoft Silverlight. Μπορεί να παράγει τόσο εγγενή (native) κώδικα και διαχειριζόμενο (managed) κώδικα.

4.1.1 Γραφικό Περιβάλλον

Για να δημιουργήσουμε το γραφικό περιβάλλον ανοίγουμε την εφαρμογή Visual Studio και επιλέγουμε File > New Project και κατόπιν επιλέγουμε New WinForms. Η οθόνη που εμφανίζεται είναι η κάτωθι:

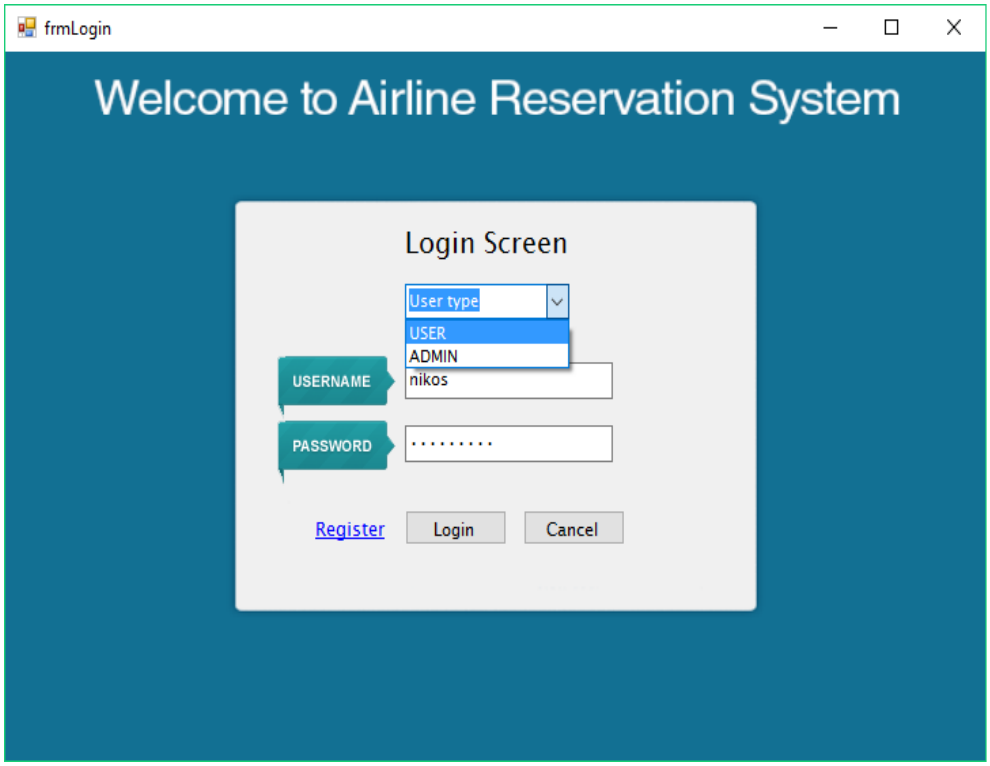


Η εργαλειοθήκη (Toolbox) εμφανίζει εικονίδια από μια πληθώρα στοιχείων ελέγχου (controls) καθώς και άλλα στοιχεία που μπορούμε να προσθέσουμε στην εφαρμογή μας. Μερικά απο τα πιο κοινά στοιχεία ειναι το Button, το TextBox, το CheckBox, το RichTextBox, το Label, το PictureBox, το RadioButton το Tooltip. Παρακάτω παρουσιάζονται τα παράθυρα της εφαρμογής και η λειτουργία της:

Οθόνη Login

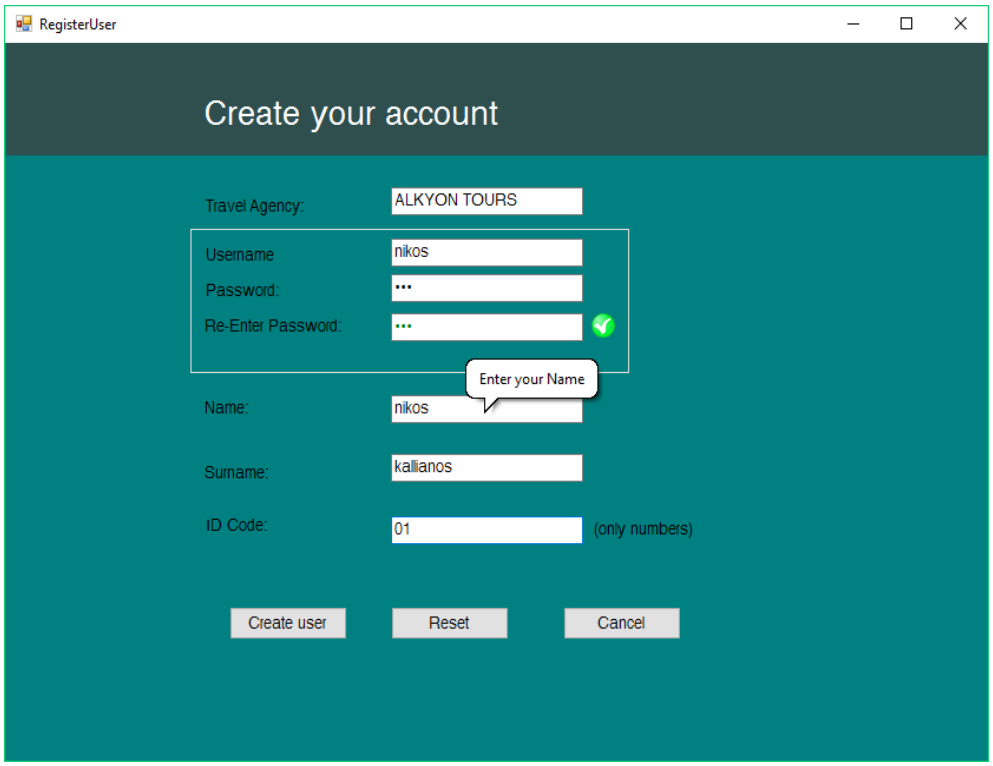
Στην εισαγωγική οθόνη έχουμε 2 επιλογές:

- User Ο χρήστης μπαίνει σαν απλός χρήστης πληκτρολογώντας το όνομα χρήστη και τον κωδικό (όπου είναι ο ταξιδιωτικός πράκτορας)
- Admin Ο χρήστης μπαίνει σαν administrator για να ρυθμίσει παραμέτρους της εφαρμογής (πτήσεις κ.α)

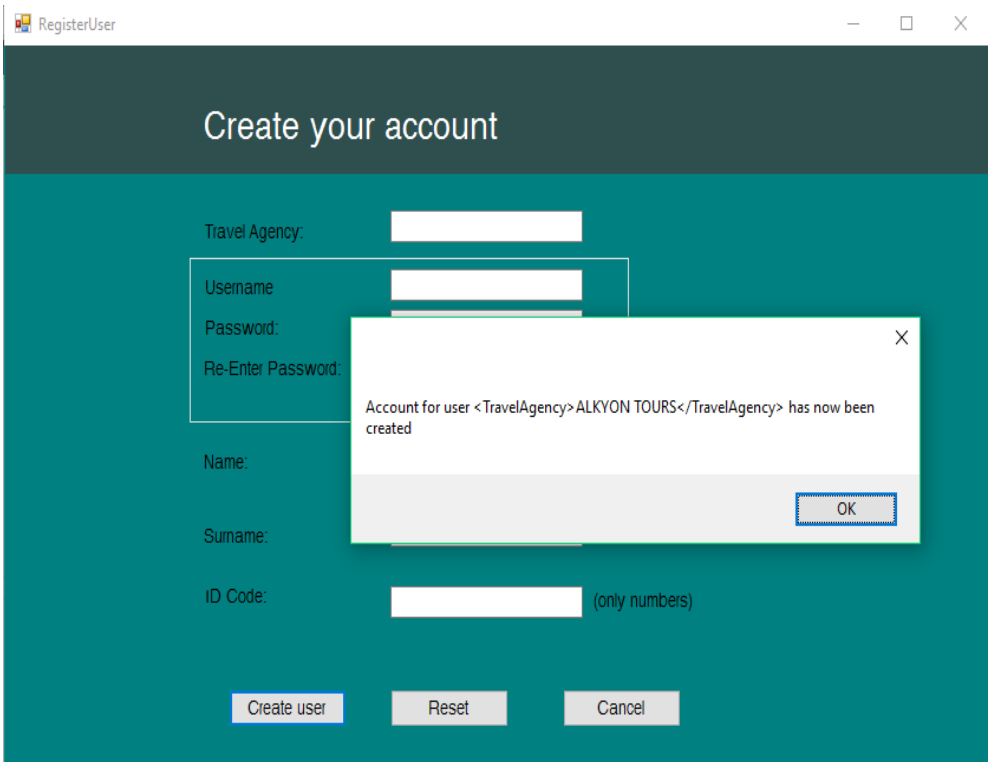


Οθόνη register

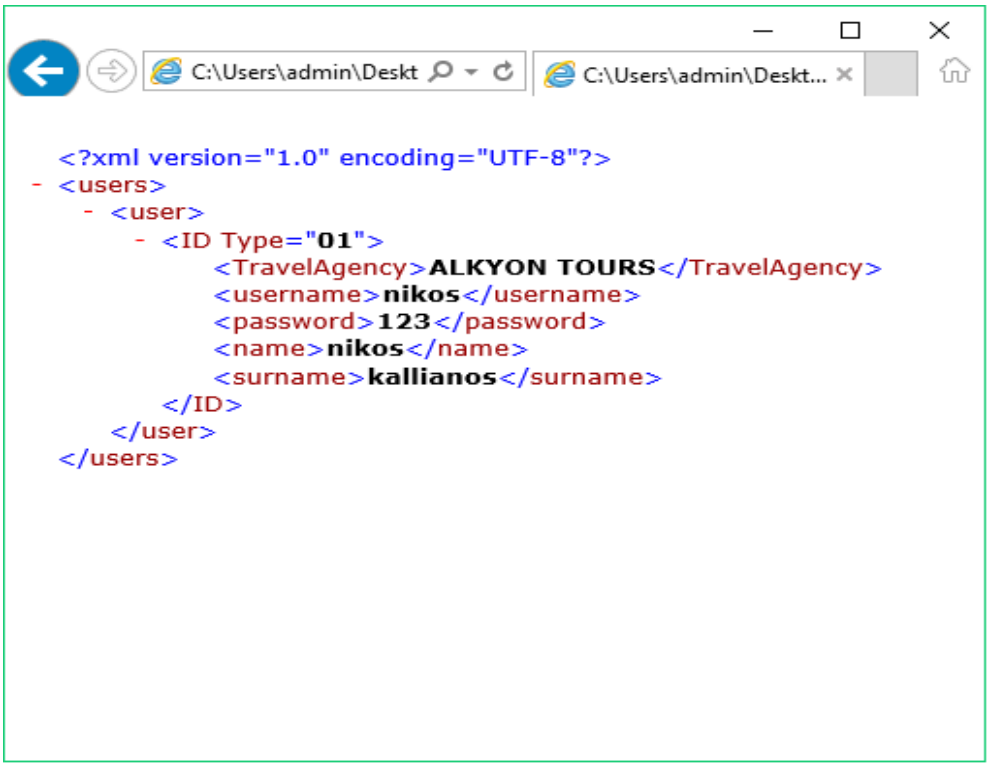
Ο χρήστης μπορεί να κάνει register πληκτρολογώντας τα στοιχεία του όπως φαίνεται παρακάνω και να δημιουργήσει έναν νέο χρήστη. Χρησιμοποιούνται baloontips στα textfields



Κατόπιν εμφανίζεται μήνυμα ότι ο χρήστης ALKYON TOURS δημιουργήθηκε.



Το έγγραφο XML που αντιστοιχεί είναι το παρακάτω. Το όνομα χρήστη είναι nikos και ο κωδικός 123 στη προκειμένη.



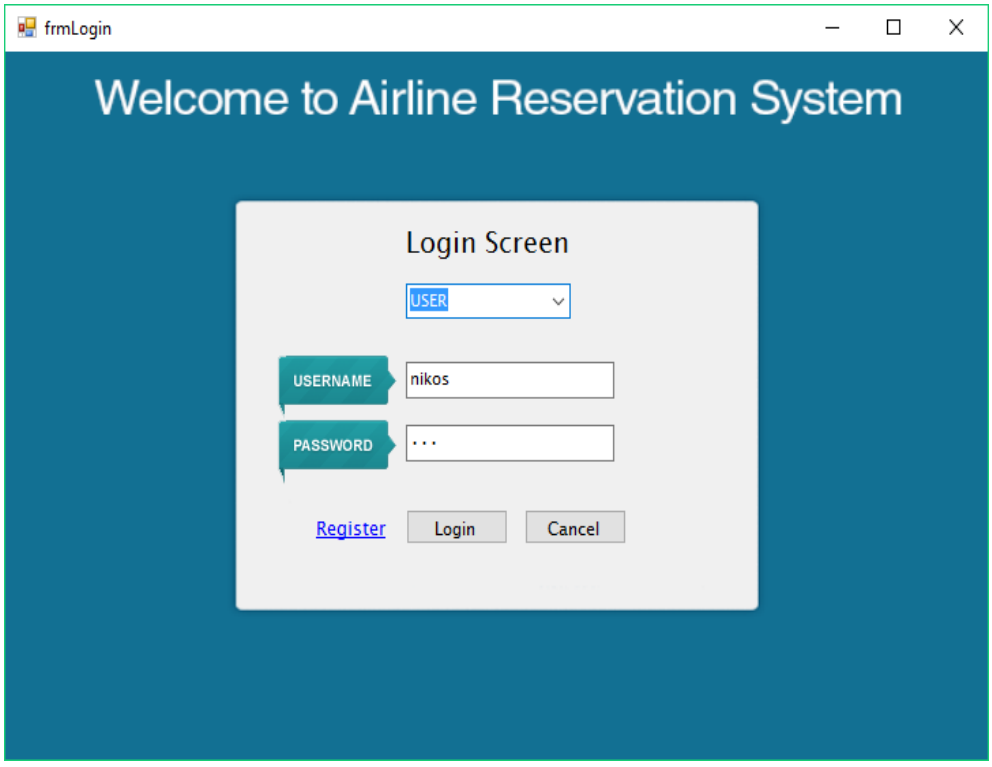
SQL:

```
Insert into users(TravelAgency,Username,password,
    name,surname,ID)
VALUES ('" & tbTravelAgency.Text & "','" &
    textBox1.Text & "','" & tbPassword.Text & "',
'" & tbName.Text & "','" &tbSurname.Text & "','" &
    tbIDCode.Text & "')"
```

Ενώ παρακάτω φαίνεται ο κώδικας που κατασκευάζει το έγγραφο XML. Όλα τα εγγραφα μας XML βρίσκονται στο φάκελο Debug του Visual Studio.

```
XDocument users = XDocument.Load("users.xml");
users.Element("users").Add(
new XElement("user",
new XElement("ID", new XAttribute("Type", tbIDCode.
    Text),
new XElement("TravelAgency", tbTravelAgency.Text),
new XElement("username",textBox1.Text),
new XElement("password", tbPassword.Text),
new XElement("name",tbName.Text),
new XElement("surname", tbSurname.Text)))
);
users.Save("users.xml");
```

Τώρα μπορούμε να κάνουμε login με τα στοιχεία που δώσαμε:



Η παρακάτω μέθοδος ValidateLogin έχει ένα ερώτημα που συγκρίνει τα string που δίνουμε στα textbox userName και password και μόνο αν είναι

ίδια με τα elements username και password του XML μπορούμε να προχωρήσουμε στο κουμπί login.

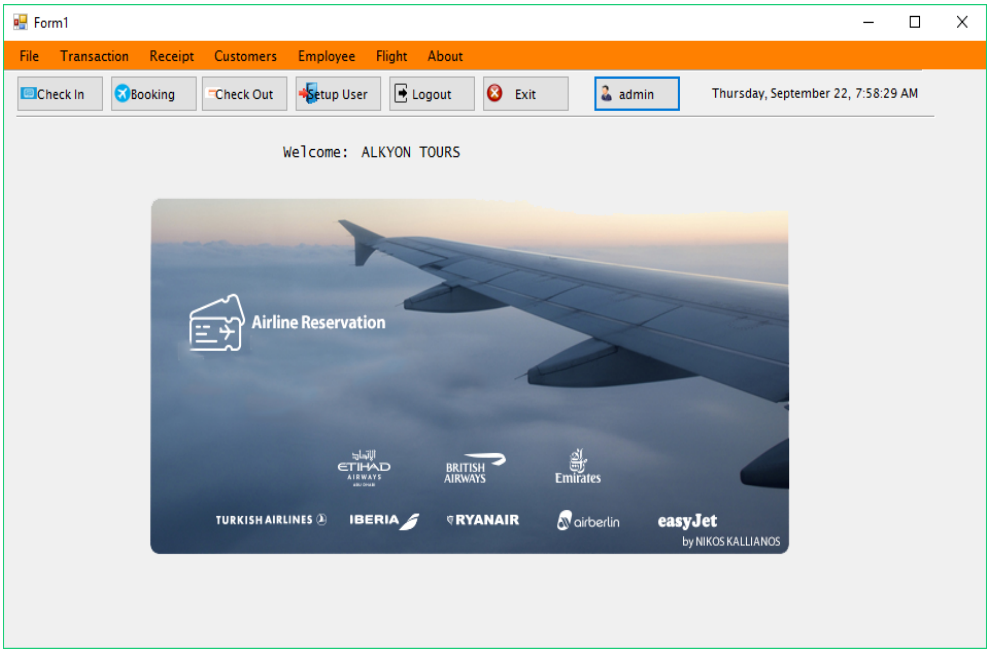
```
public bool ValidateLogin(string userName, string
    password)
{
bool validUser = false;
try
{
string fileName = System.IO.Path.Combine
(Application.StartupPath, "users.xml");
XDocument users = XDocument.Load(fileName);

XElement userElement = (from subitem in
(
from item in users.Descendants("user")select item)
where subitem.Element("username").
Value.ToLower() == userName.ToLower() &&
subitem.Element("password").Value == password
select subitem).First();
validUser = true;
}
catch (Exception ex)
{
if (ex != null) { }
}
return validUser;
}
```

Ο κώδικας που εκτελείται όταν πατάμε το button Login είναι ο εξής: Σημειοτέον έχουμε 3 προσπάθειες μέχρι να δώσουμε το σωστό κωδικό, αλλιώς εμφανίζεται μήνυμα λάθους:

```
private int m_attempts = 0;
private const int MAX_LOGIN_ATTEMPTS = 3;

private void btnLogin_Click(object sender,
   EventArgs e)
{
bool authenticated = true;
m_attempts++;

if (comboBox1.Text == "USER")
{
String user;
user = textBox1.Text;
String pass;
pass = textBox2.Text;
if (user != "" && pass != "")
{
authenticated = ValidateUser(user, pass);
//this.Close();
```

```
frmMain main = new frmMain();
main.Show();
}
else
if (!authenticated)
{
if (m_attempts == MAX_LOGIN_ATTEMPTS)
{
MessageBox.Show("You have reached the maximum limit
    of login attempts");
Application.Exit();
}
else
{
MessageBox.Show("Invalid Login. Try again",
String.Format("Invalid Login (attempt {0} of {1})",
m_attempts, MAX_LOGIN_ATTEMPTS));
}

}//end if

}//end if USER
else
if (comboBox1.Text == "ADMIN")
{
if(textBox1.Text =="admin"||textBox1.Text=="
   password")
{
frm_AdminMain frmAdminMain = new frm_AdminMain();
frmAdminMain.Show();
}
}//end if ADMIN
else
{
MessageBox.Show("Please select a choice", "ADMIN or
    USER", MessageBoxButtons.OK, MessageBoxIcon.
   Information);
}
}//end btnLogin
```

Οθόνη main



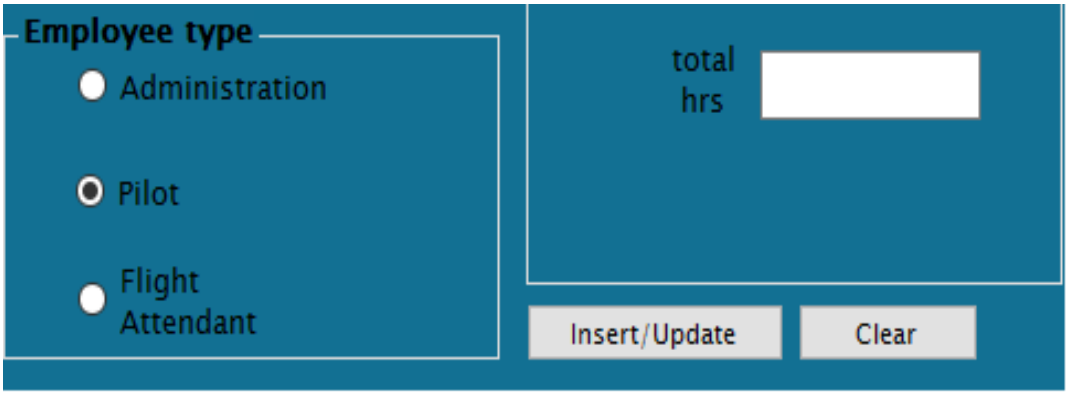
Έτσι, πατώντας το κουμπί Login βρισκόμαστε στη κεντρική οθόνη σαν χρήστης ALKYON TOURS όπως φαίνεται στο παράθυρο: Στήν οθόνη αυτή μπορούμε να κάνουμε διάφορες λειτουργίες όπως να προσθαφαιρέσουμε πελάτες εργαζόμενους, να κάνουμε κράτηση κτλ. Μερικά φαίνονται παρακάτω:

Οθόνη Employee

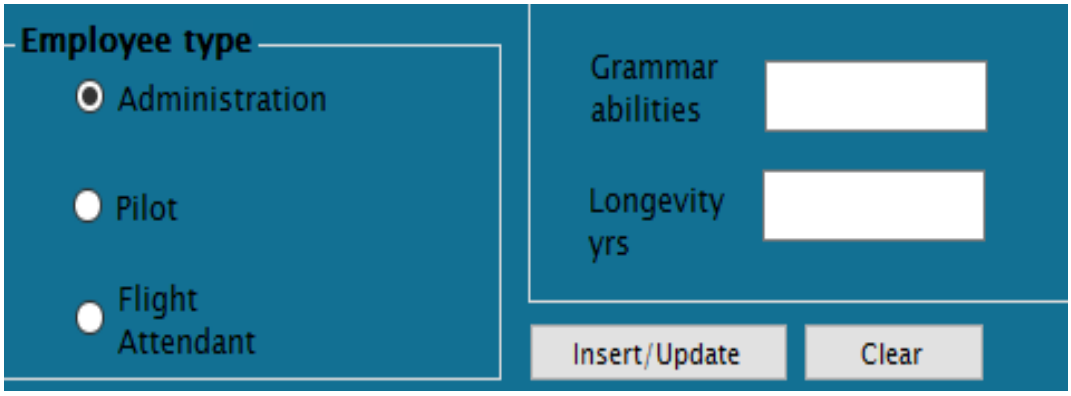
Στην οθόνη αυτή μπορούμε να προσθέσουμε τρείς τύπους εργαζομένων: Διοικητικούς, Πιλότους και Αεροσυνοδούς.

Προσθήκη(Insert)

Η προσθήκη κατηγορίας εργαζομένου γίνεται ανάλογα με τα radiobuttons ό που εμφανίζεται το αντίστοιχο panel και textbox. για παράδειγμα στον pilot εμφανίζεται το πεδίο Total Hrs(ωρες πτήσης)



ενώ για τους διοικητικούς εμφανίζονται τα πεδία Longevity years(ετη προυπηρεσιας) και Grammar Abilities(γραμματικές γνώσεις)

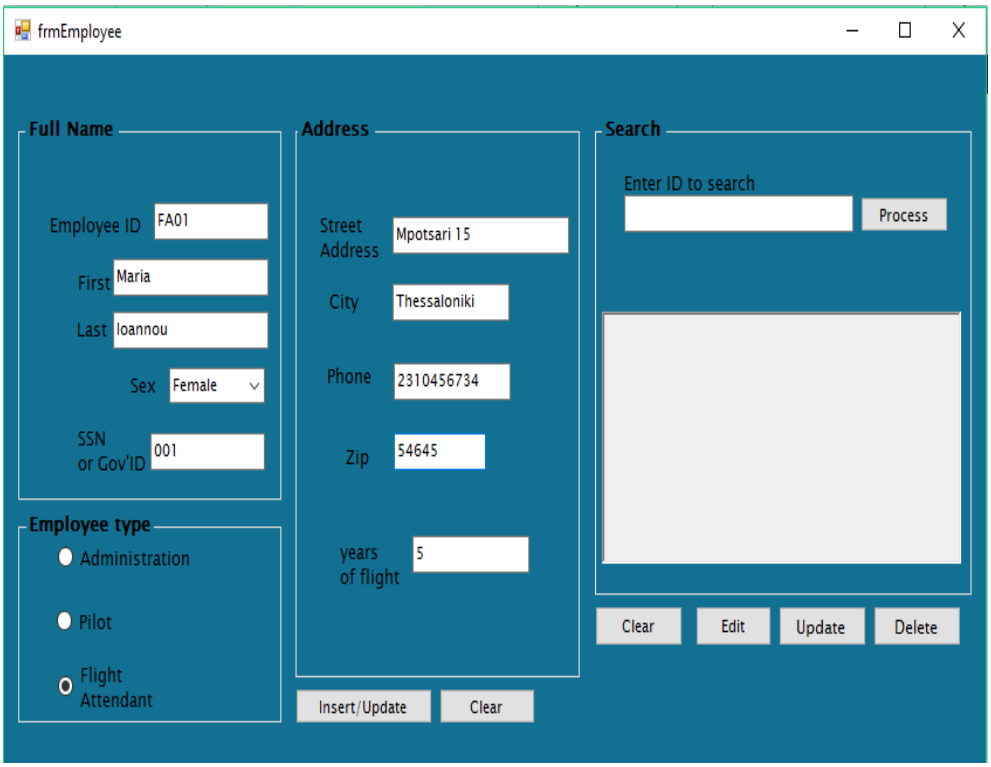


Ο διαχωρισμός των panels γίνεται με τον παρακάτω κώδικα: (Το panel3 εμφανίζεται όταν πατάμε το radiobutton FlightAttendant, το panel2 οταν πατάμε το radiobutton Administration, και το panel1 οταν επιλέγουμε το radiobutton Pilot)

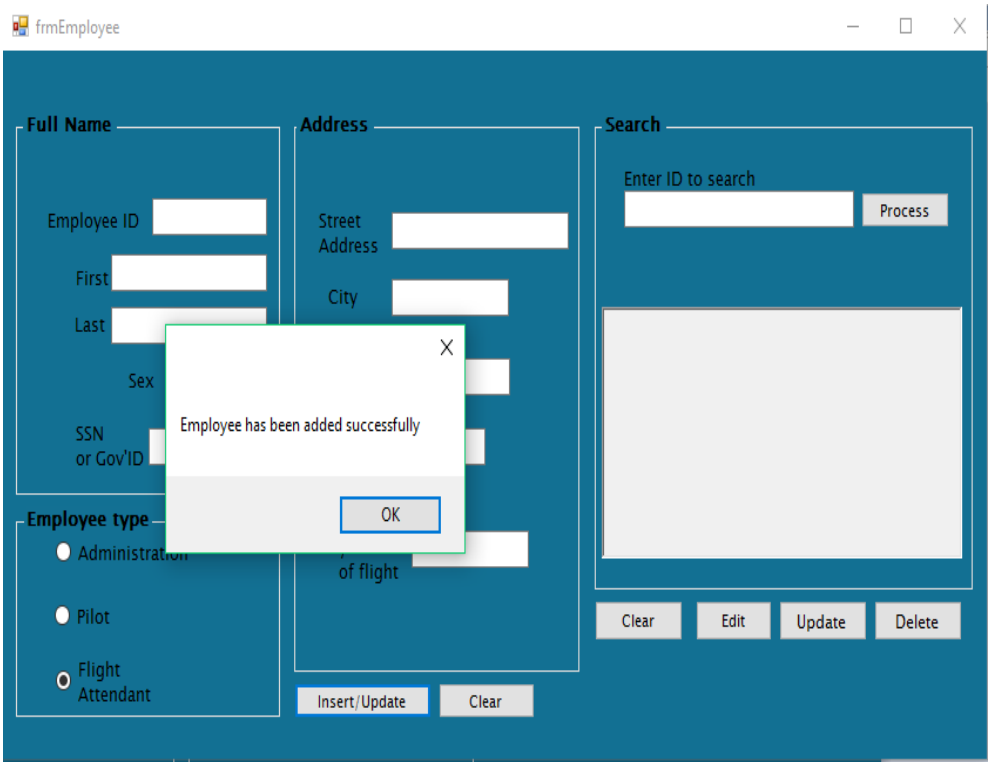
```
private void rbAdministration_CheckedChanged(
    object sender, EventArgs e)
{
if (rbAdministration.Checked)
{
panel2.Visible = true;
panel1.Visible = false;
rbAttendant.Checked = false;
rbPilot.Checked = false;
}
}
private void rbAttendant_CheckedChanged(object
    sender, EventArgs e)
{
if (rbAttendant.Checked)
{
panel3.Visible = true;
panel1.Visible = false;
panel2.Visible = false;
rbPilot.Checked = false;
}
}
private void rbPilot_CheckedChanged(object sender,
     EventArgs e)
{
if(rbPilot.Checked)
{
panel1.Visible = true;
panel3.Visible = false;
rbAttendant.Checked = false;
rbAdministration.Checked = false;
```

```
}
}
```

Παρακάτω φαίνεται η προσθήκη μιας αεροσυνοδού:



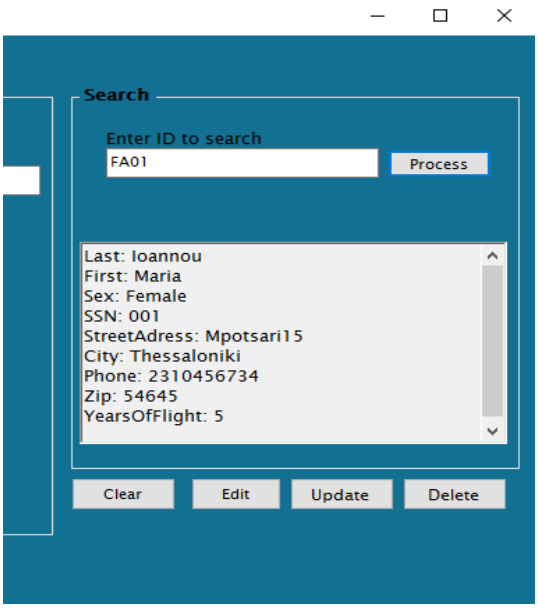
Πατώντας το κουμπί Insert/Update εμφανίζεται μήνυμα ότι η ο εργαζόμενος προστέθηκε.



Το έγγραφο XML που δημιουργήθηκε για 2 κατηγορίες εργαζομένων φαίνεται παρακάτω:

Επεξεργασία(Update)

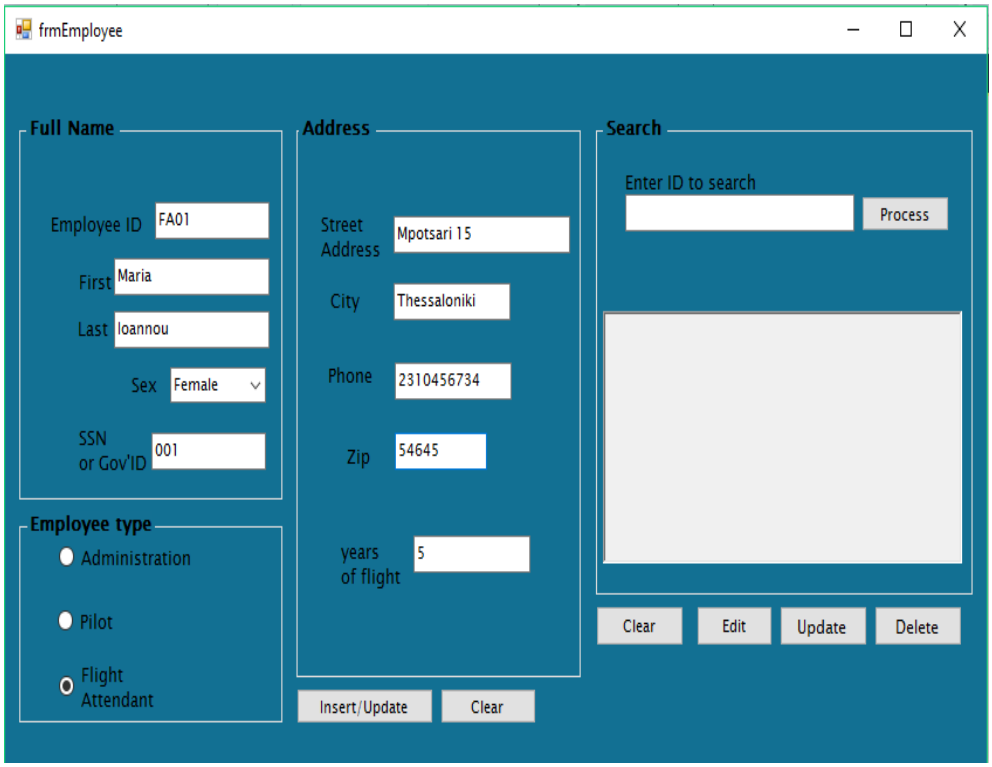
Μπορούμε να ανενεώσουμε τα στοιχεία ενός εργαζομένου, αναζητώντας τον απο το ID του. Παρακάτω αναζητάμε τον Employee με το ID FA01 που είναι η Ιωαννου Μαρία: Πατώντας το κουμπί Process τα αποτελέσματα εμφανίζονται σε ένα richcombobox.



Ο κώδικας του κουμπιού Process που εμφανίζει τα στοιχεία του richcombobox είναι: Το textBox4 είναι το πεδίο που συγκρίνει το ID του εργαζομένου στο attribute του εγγράφου XML. Η αναζήτηση γίνεται με το παρακάτω ερώτημα:

```
var persons = from person in xmlDoc.Descendants
("Employee")
where person.Attribute("ID").Value == textBox4.Text
select new
{
Last = person.Element("Last").Value.ToString(),
First = person.Element("First").Value.ToString(),
Sex = person.Element("Sex").Value.ToString(),
SSN = person.Element("SSN").Value.ToString(),
Address=person.Element("StreetAdress").Value.
    ToString(),
City = person.Element("City").Value.ToString(),
Phone = person.Element("Phone").Value.ToString(),
Zip = person.Element("Zip").Value.ToString(),
YOF=person.Element("YearsOfFlight").Value.ToString
    ()
};

foreach (var person in persons)
{
richTextBox1.Text = richTextBox1.Text + "Last: " +
    person.Last + "\n";
richTextBox1.Text = richTextBox1.Text + "First: " +
     person.First + "\n";
richTextBox1.Text = richTextBox1.Text + "Sex: " +
    person.Sex + "\n";
richTextBox1.Text = richTextBox1.Text + "SSN: " +
    person.SSN + "\n";
richTextBox1.Text = richTextBox1.Text + "
    StreetAdress: " + person.Address + "\n";
richTextBox1.Text = richTextBox1.Text + "City: " +
    person.City + "\n";
richTextBox1.Text = richTextBox1.Text + "Phone: " +
     person.Phone + "\n";
richTextBox1.Text = richTextBox1.Text + "Zip: " +
    person.Zip + "\n";
richTextBox1.Text = richTextBox1.Text + "
    YearsOfFlight: " + person.YOF + "\n";
}
```

Παρακάτω φαίνεται τι γίνεται όταν πατάμε το κουμπί Edit. Εμφανίζονται όλα τα στοιχεία παρμένα απο το έγγραφο XML Employees.xml.



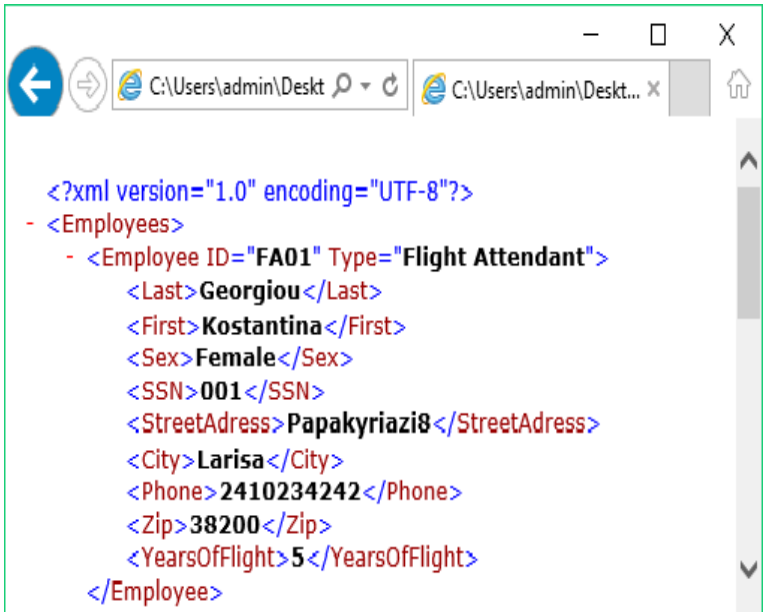
Μερικά απο τα ερώτηματα που βρίσκουν τα στοιχεία και τα εμφανίζουν φαίνονται παρακάτω: Στο κάτωθι ερώτημα, βρίσκει το επίθετο και το εμφανίζει στο textbox tbLast. Εχουμε χρησιμοποιήσει τον τελεστή Take() για να πάρουμε ένα μονο element απο την ακολουθία:
Φορτώνουμε το έγγραφο XML

```
XElement xml2 = XElement.Load("Employee.xml");
```

Και στη συνέχεια αναζητούμε τα επιμέρους Elements με το ερώτημα Select με σύνταξη LAMBDA expression:

```
var search = xml2.Descendants("Employee")
.Where(n => n.Attribute("ID").Value == textBox4.
    Text)
.Select(n => n).Take(1)
.Single().Element("Last");
tbLast.Text = search.Value.ToString();
tbLast.Select();
```

Με όμοιο τρόπο βρίσκουμε και τα υπόλοιπα Elements και τα εμφανίζουμε στα αντίστοιχα textbox.
Πατώντας το κουμπί Update τα στοιχεία στο έγγραφο XML gia to ID "FA01" βλέπουμε ότι άλλαξαν, ο εργαζόμενος από Μαρία Ιωάννου έγινε Κωσταντίνα Γεωργίου:



Ο κώδικας που εκτελέστηκε στο button Update είναι:

```
XElement temp = XElement.Load("Employee.xml");
var items=from item in temp.Descendants("Employee")
where item.Element("Last").Value != null || item.
    Element("First").Value != null||
item.Element("SSN").Value != null || item.Element("
    StreetAdress").Value != null||
item.Element("City").Value != null|| item.Element("
    Phone").Value != null||
item.Element("Zip").Value != null
select item;

foreach(XElement xe in xml3.Descendants("Employee")
    ) \{
xe.Element("Last").Value = tbLast.Text;
xe.Element("First").Value = tbFirst.Text;
xe.Element("SSN").Value = tbSSN.Text;
xe.Element("StreetAdress").Value = tbStreet.Text;
xe.Element("City").Value = tbCity.Text;
xe.Element("Phone").Value = tbPhone.Text;
xe.Element("Zip").Value = tbZip.Text;
...
}
temp.Save("Employee.xml");
MessageBox.Show("Records have been updated
    succesfully");
```

Διαγραφή(Delete)

Για να διαγράψουμε έναν εργαζόμενο αναζητούμε το ID του. Ο κώδικας που εκτελείται πατώντας το κουμπί delete είναι:

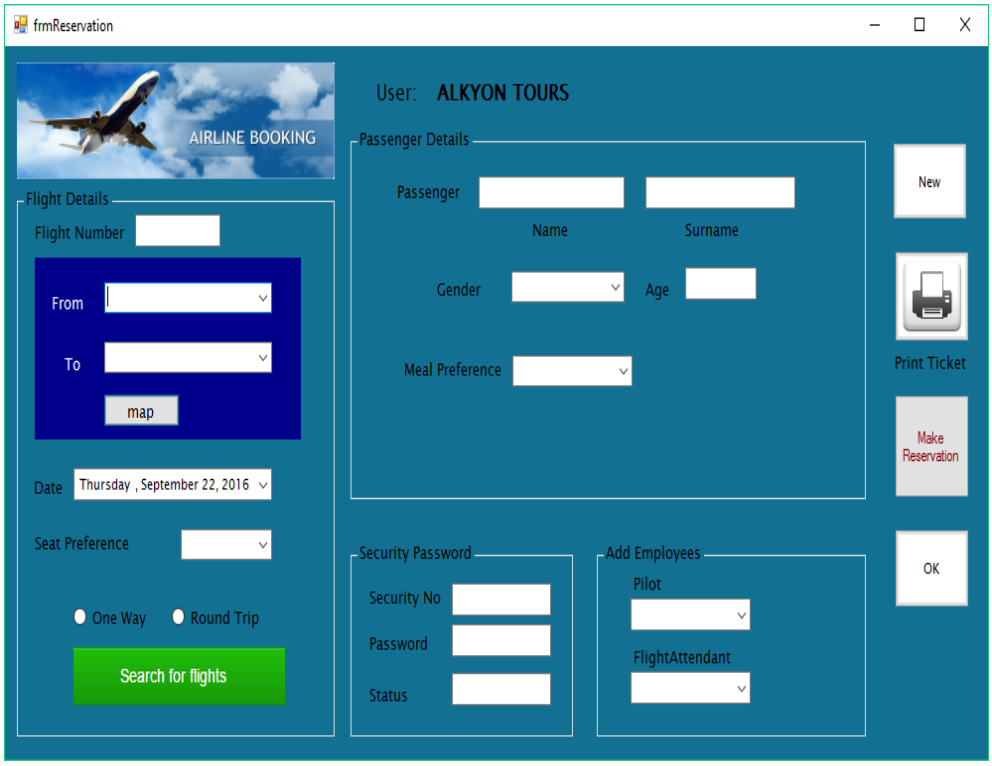
```
var deleteEmployee = (from employeeid in xmlDoc.
    Descendants("Employee")
where employeeid.Attribute("ID").Value == textBox4
    .Text
select employeeid).ToList();
foreach (XElement del in deleteEmployee)
{
del.Remove();
}
xmlDoc.Save("Employee.xml");
MessageBox.Show("Employee has been removed");
```

Οθόνη Κρατήσεων (Reservation)

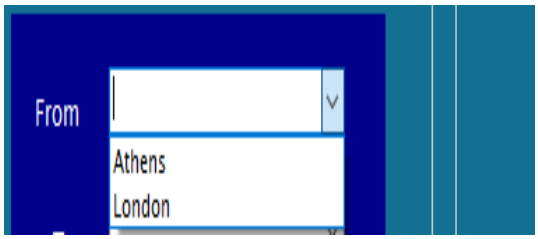
Η οθόνη reservation (υπενθυμίζεται ότι ο user που έκανε login ως nikos και password: 123 αντιστοιχεί στον χρήστη ALKYON TOURS και εμφανίζεται στο label19 της εφαρμογής) έχει διάφορες

```
XElement users = XElement.Load("users.xml");
var name2 = users.Descendants("ID")
.Elements("TravelAgency")
.Select(x => x).Take(1).Single();
label19.Text = name2.Value.ToString();
```

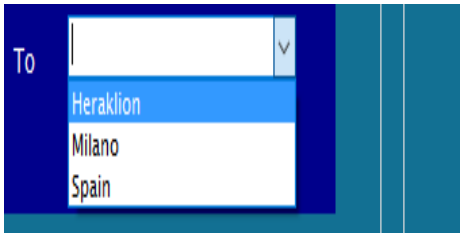
επιλογές που είναι επιλογή Από-Πρός, Ημερομηνία αναζήτησης πτήσεων, Αριθμό Θεσης - Πρώτη, Οικονομικη, Στοιχεία Επιβατη, Κωδικός Κράτησης, καθώς και προσθήκη Προσωπικού (πιλότοι-αεροσυνοδοί) και τέλος Εκτύπωση του εισητηρίου.



Ο χρήστης μπορεί να επιλέξει την αναχώρηση



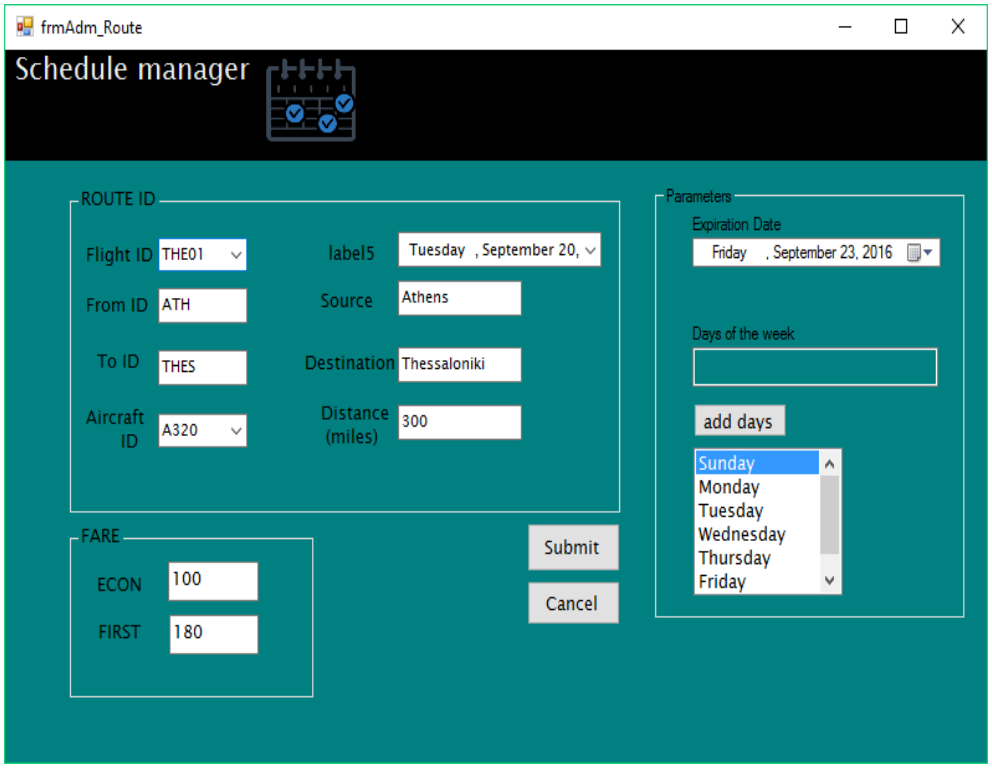
Και τον προορισμό



Ο κώδικας που συνδέεται με τα combobox είναι ο ακόλουθος (LAMBDA expression). Εχει χρησιμοποιηθεί ο τελεστής DISTINCT προκειμένου να μην εμφανίζονται πολλές φορές τα ίδια στοιχεία στα combobox. Για το combobox source:

```
cbFrom.Items.AddRange(obj.Descendants("Aircraft")
.Select(item => item.Attribute("Source").Value)
.Distinct()
.OrderBy(item => item)
.ToArray());
```

Ομοια και για το combobox Destination.
Τις πτήσεις τις ορίζει ο administrator. Δηλαδή κάνοντας login με όνομα "admin", και επιλέγοντας "Add Route", μπορούμε να ορίσουμε στοιχεία της πτήσης όπως φαίνεται στην εικόνα:



Στο συγκεκριμένο παράδειγμα δώσαμε δρομολόγιο κράτησης Athens - Thessaloniki με τιμή 100 ευρώ για την Οικονομική θέση και 180 για την Πρώτη. Επίσης έχουμε και επιλογή αυτόματης ακύρωσης επιλέγοντας οποια ημερομηνία θέλουμε στο Expiration Date. Αυτά τα στοιχεία αποθηκεύονται σε ενα έγγραφο XML με όνομα schedulemanager.xml

Αυτόματη ακύρωση κράτησης

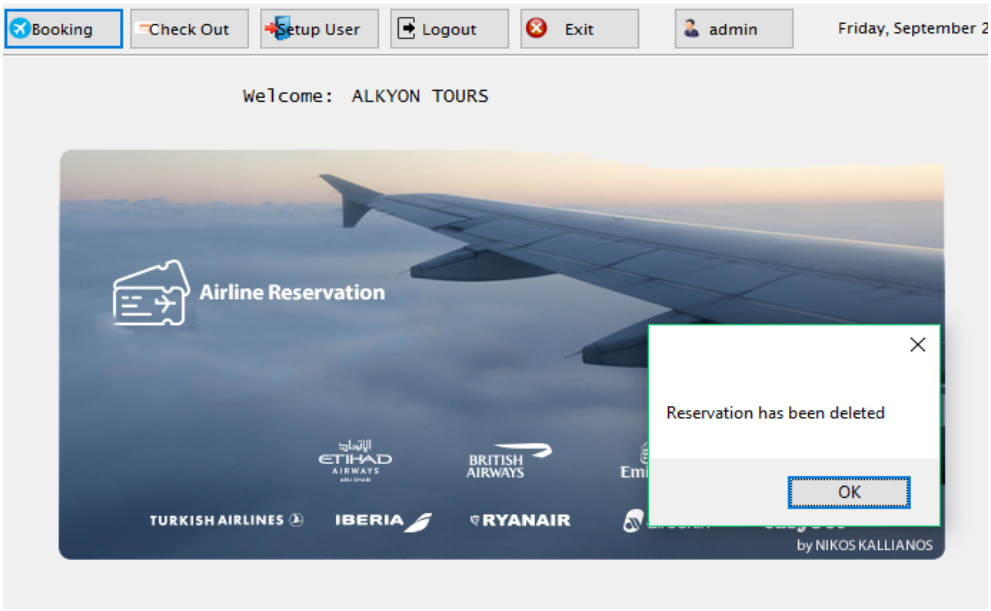
Η αυτόματη ακύρωση γίνεται όταν "συμφωνούν" οι ημερομηνίες που δίνουμε στο combobox και αυτής που εγγράφετα στο έγγραφο XML.
Ο κώδικας παρακάτω διαβάζει την ημερομηνία απο το XML, την παίρνει σαν string, και κατόπιν το string μετατρέπεται σε τύπου DateTime μεταβλητή μέσω της Convert(). Έπειτα h Compare() συγκρίνει τις ημερομηνίες και αν συμφωνούν διαγράφει όλα τα στοιχεία απο το XML.

```
XDocument xdoc = XDocument.Load("schedulemanager.
    xml");
var date = xdoc.Descendants("schedule")
.Elements("Aircraft").Elements("ExpiryDate")
.Select(x => x).Take(1).Single().Value;

DateTime inputDate = Convert.ToDateTime(date);

xdoc.Descendants("Aircraft").
Where(x => DateTime.Compare(DateTime.Parse
(x.Element("ExpiryDate")
.Value, null, DateTimeStyles.RoundtripKind).Date,
inputDate.Date)==0).
```

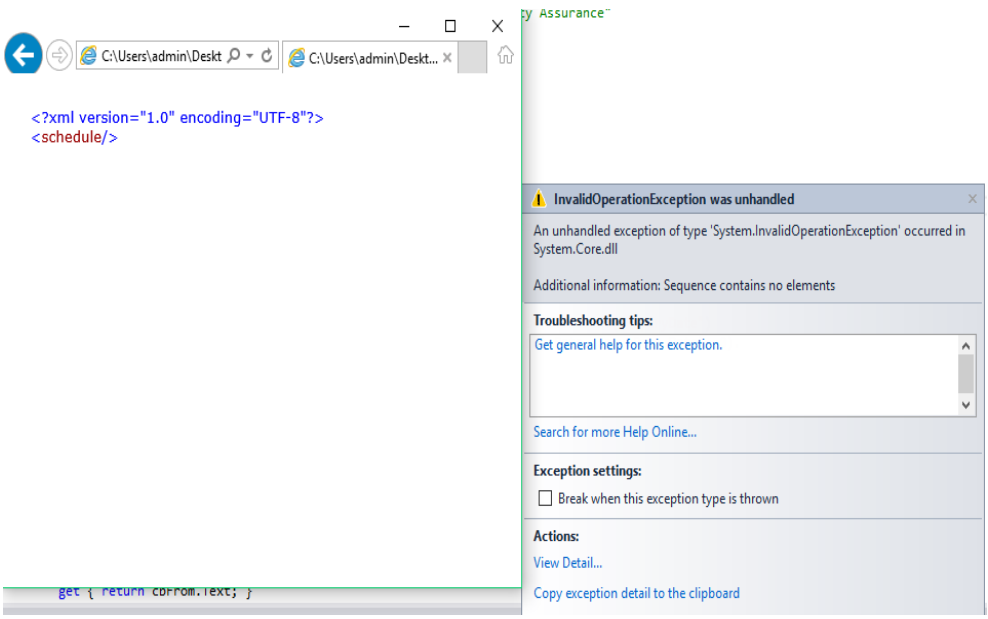
```
ToList().ForEach(x => x.Remove());
MessageBox.Show("Reservation has been deleted");
xdoc.Save("schedulemanager.xml");
```

Το παράδειγμα δείχνει αυτό ακριβώς: Ορίσαμε ημερομηνία ακύρωσης για την πτήση Athens - Heraklion την 23/9/2016 όπως φαίνεται στο XML:

Βλέπουμε ότι εμφανίζεται το μήνυμα "Reservation has been deleted".

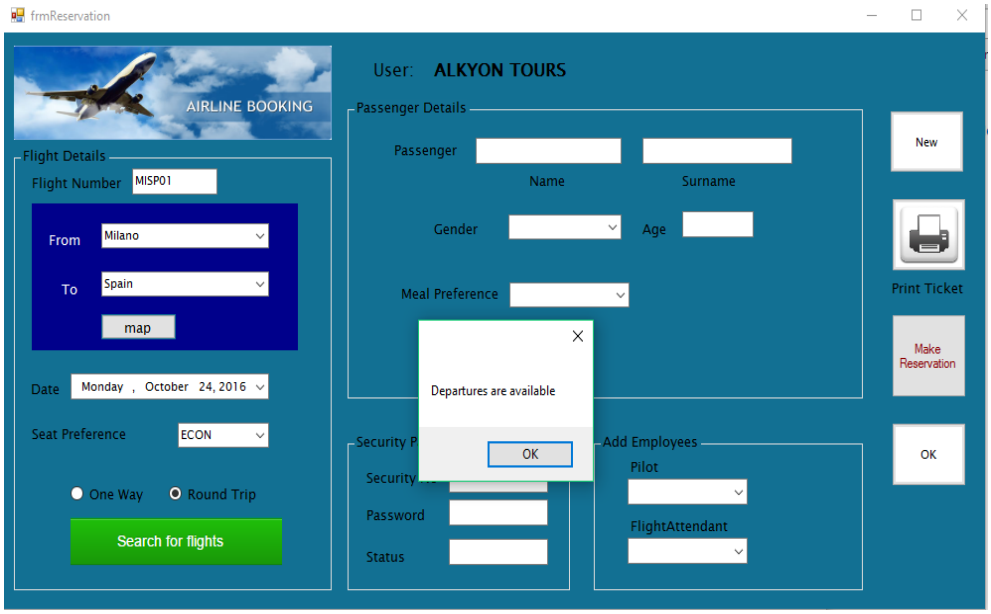


Το XML μας είναι κενό πια και στο Visual Studio εμφανίζεται το μήνυμα Sequence contains no Elements.

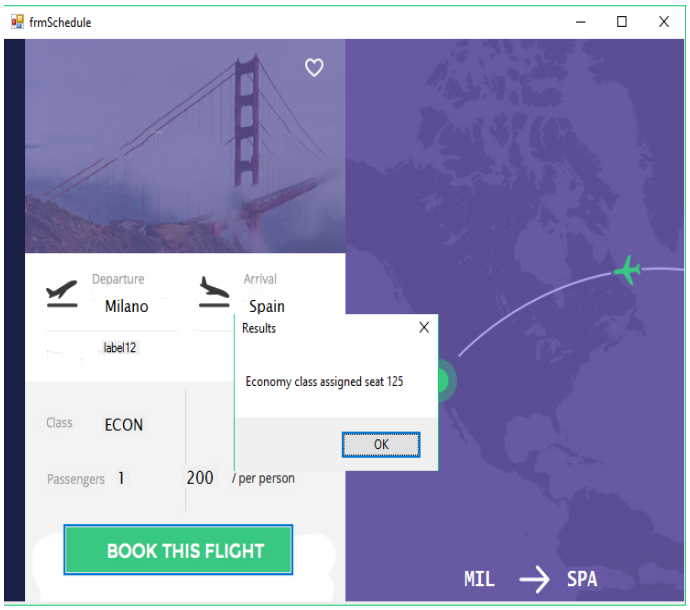


Κράτηση (Reservation)

Στην οθόνη κρατήσεων μπορούμενα επιλέξουμε την αναχώρηση και τον προορισμό και να δούμε αν υπάρχουν διαθέσιμα δρομολόγια. Το παράδειγμα παρακάτω δείχνει ένα δρομολόγιο Milano - Spain στην οικονομική θέση ECON. Επιλέγουμε το Search for Flights και εμφανίζεται το μήνυμα Departures are available. (Η συγκεκριμένη εφαρμογή συγκρίνει τις ημερομηνίες).



Στη συνέχεια εμφανίζεται η φόρμα frmSchedule που εμφανίζει την κράτηση. Αυτά διαβαζονται απο το XML schedulemanager.xml. Βλέπουμε ότι μας δίνει πληροφοριες οτι κρατήθηκε η θέση Οικονομική θέση 125 και αφού ειναι οικονομική, έχει τιμη 200 ευρώ (στην πρώτη κοστίζει 400 για τη συγκεκριμένη πτήση). Οι πληρόφορίες εμφανίζονται στα label.



Ο κώδικας που εκτελέιται πατώντας το Search For Flights ειναι ο παρακάτω. ΕΚτελείται το ερώτημα που ψάχνει ποιο αεροπλανο είνα για τη συγκεκριμένη πτήση. Υποθέτουμε οτι έχουμε 2 αεροπλανα για λόγους ευκολίας ένα AIRBUS A320 και ένα BOEING 747. Το A320 υποθέτουμε ότι έχει 140 θέσεις, 100 Οικονομικές ECON και 40 πρώτης θέσης FIRST, ενω το B747 160 Οικονομικές και 60 Πρώτες δηλαδή σύνολο 220.

```
var p = (from element in XDocument.Load("
    schedulemanager.xml").Descendants("Aircraft")
let type = element.Attribute("ID").Value
where type.Equals("B747")
select type == "B747" ? 220 :
type == "A320" ? 140 : 0).FirstOrDefault();
```

Έτσι, η μεταβλητή p παίρνει τη τιμή ανάλογα με το LINQ ερώτημα. Ο πίνακας με τις θέσεις του αεροπλάνου είναι ενας bool που παίρνει σαν τιμή τη μεταβλητή p και αρχικοποιείται σε false.

```
bool[] seats = new bool[p];

for (int i = 0; i < p; i++)
{
 seats[i] = false;
}
```

Στη συνέχεια το comboBox4 (που έχει τις κατηγορίες θέσεων FIRST και ECON) ανάλογα ποιό θα επιλεχθεί, εκτελείται και η ανάλογη συνάρτηση κράτησης θέσεων AssignFirstClass για το FIRST και AssignSecondClass για την ECON:

```
if (comboBox4.SelectedText.Equals("FIRST") || temp
    == dtArrival.Text)
{
if (totalAssignedFirstClass < 40)
{
MessageBox.Show("Departures are available");
frmScheduleResult frm = new frmScheduleResult(this
    );
frm.Show();
assignFirstClass();
}
}
else if (comboBox4.SelectedText.Equals("ECON") ||
    temp == dtArrival.Text)
if (totalAssignedSecondClass < 100)
{
MessageBox.Show("Departures are available");
frmScheduleResult frm = new frmScheduleResult(this
    );
frm.Show();
assignSecondClass();
}
else
```

```
if (temp != dtArrival.Text)
{
MessageBox.Show("There are no available departures
    ");
//goto finish;
return;
}
else if (totalAssignedSecondClass == 100 &&
    totalAssignedFirstClass < 40)
{
MessageBox.Show("Sorry, economy class is full. Do
    you want to get a ticket for first class? Y-N")
    ;
}
```

Η μεταβλητή temp είναι το αποτέλεσμα του ερωτήματος που βρίσκει την ημερομηνια απο το XML και την συγκρίνει με το dateTime combobox των ημερομηνιών.

```
XDocument manager = XDocument.Load("schedulemanager
    .xml");
var result = from reservat in manager.Descendants("
    Aircraft")
let ep = reservat.Attribute("mdy").Value
select ep;
var temp = result.ToString();
```

Εαν καλέσουμε την FIRST κατηγορία Πρώτης Θέσης τότε καλέιται η συναρτηση AssignFirstClass. Για τις πρώτες θέσεις έχουμε 40 διαθέσιμες οπότε και η rand είναι απο 1 εως 41. Ακριβώς η ίδια συνάρτηση καλείται και για τις πρώτες θέσεις μόνο που η rand φτάνει απο απο 41 εως 141 (100 συνολικά θέσεις δηλαδή).

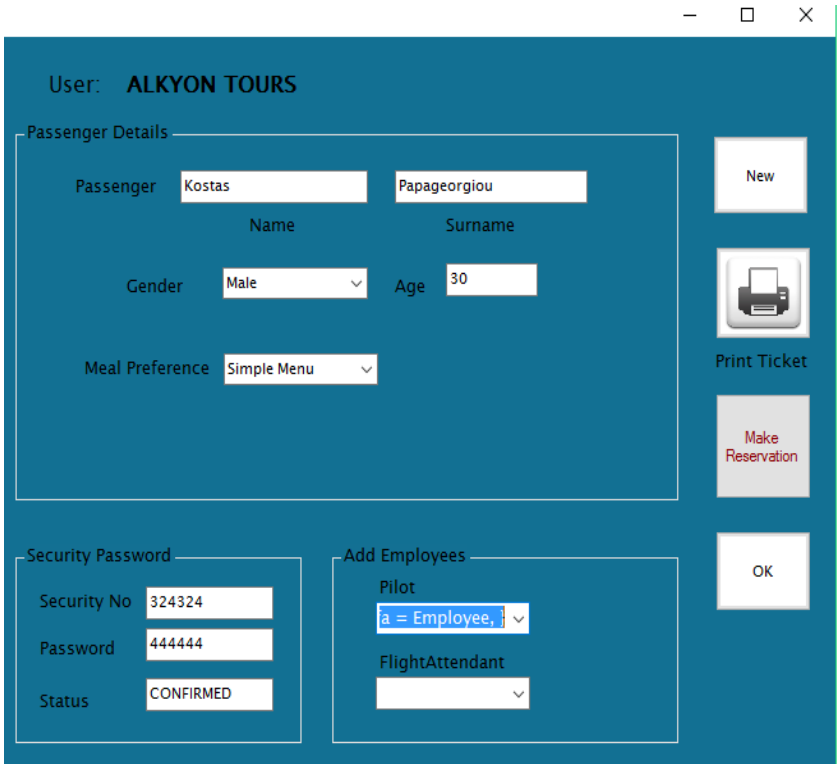
```
public void assignFirstClass()
{
  var p = GetAircraftSeat("A320");
  bool[] seats = new bool[p];
  bool noDuplicate = false;
  Random rand = new Random();
  int index = 0;
  while (!noDuplicate)
  {
    noDuplicate = true;
    index = rand.Next(1, 41);
    if (seats[index] == true)
    noDuplicate = false;
  }
  seats[index] = true;
  totalAssignedFirstClass++;
  MessageBox.Show(string.Format("First class
      assigned seat {0}", index), "Results",
      MessageBoxButtons.OK);
```

```
    XDocument rese = XDocument.Load("seat.xml");
    XDocument reservation = XDocument.Load("
        reservation.xml");

    reservation.Elements("Reservations").First().Add(
    new XElement
    (
    "Seat", new XAttribute("No", index)
    )
    );
    reservation.Save("reservation.xml");
}
```

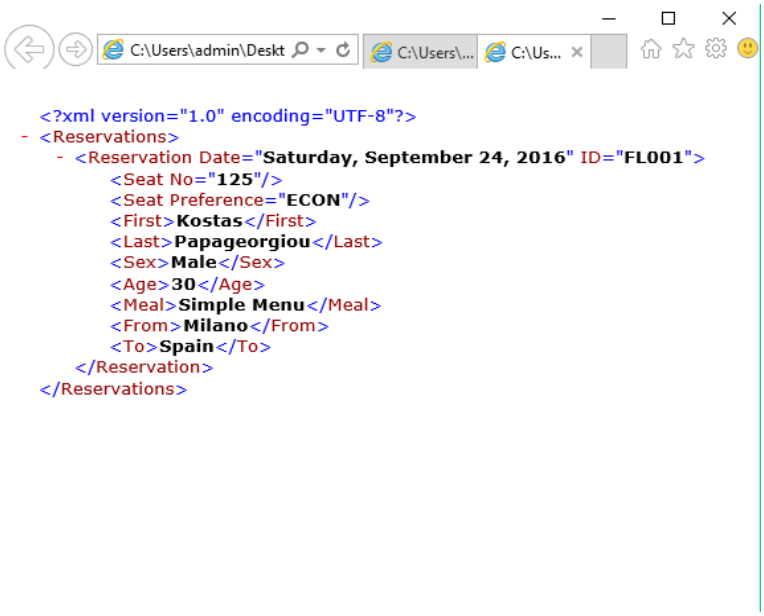
Με τη GetAircraftSeat πήραμε ξανά τις θέσεις του αεροπλάνου για να τις χρησιμοποιήσουμε στις συναρτήσεις AssignFirstClass και AssignSecondClass.

```
    public static int GetAircraftSeat(string
        aircraftModel)
    {
      var p = (from element in XDocument.Load("
          schedulemanager.xml").Descendants("Aircraft"
          )
      let type = element.Attribute("ID").Value
      where type.Equals(aircraftModel)
      select type == "B747" ? 100 :
      type == "A320" ? 140 : 0).FirstOrDefault();
      return p;
    }
```

Στη συνέχεια, δίνουμε τα στοιχεία του πελάτη. Μπορούμε να κάνουμε διάφορες επιλογές εδώ όπως γεύμα, προσθήκη αεροσυνοδού πιλότου κ.α.



Κατόπιν πατώντας MAKE RESERVATION δημιουργείται ένα αρχείο XML με όλα τα στοιχεία της κράτησης.

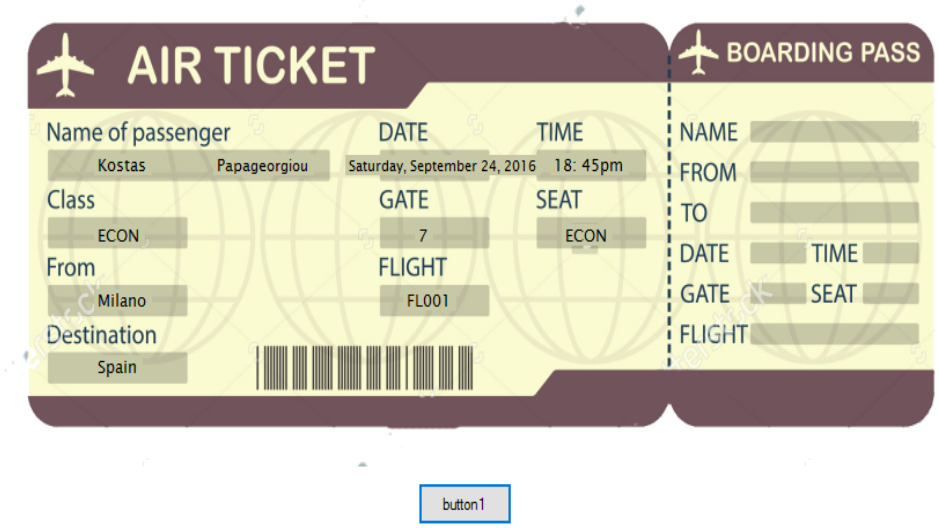


Μπορούμε να εκτυπώσουμε το εισητηριο μας σε μορφη PDF με όνομα output.pdf με path που έχουμε ορίσει εμείς. Ο κώδικας που αναζητούμε κάθε όνομα label μέσα στο XML reservation.xml είναι:

```
foreach (XElement xe in xmlTick.Nodes())
{
  var name2 = xmlTick.Descendants("Seat")
  .Attributes("Preference")
  .Select(x => x).Take(1).Single();
  label2.Text = name2.Value.ToString();

  var name3 = xmlTick.Descendants("Reservation")
  .Elements("First")
  .Select(x => x).Take(1).Single();
  label1.Text = name3.Value.ToString();

  var name4 = xmlTick.Descendants("Reservation")
  .Elements("Last")
  .Select(x => x).Take(1).Single();
  label10.Text = name4.Value.ToString();

  var name5 = xmlTick.Descendants("Reservation")
  .Elements("From")
  .Select(x => x).Take(1).Single();
  label3.Text = name5.Value.ToString();

  var name8 = xmlTick.Descendants("Reservation")
  .Elements("To")
  .Select(x => x).Take(1).Single();
  label4.Text = name8.Value.ToString();
```

```
  var name6 = xmlTick.Descendants("Reservation")
  .Attributes("Date")
  .Select(x => x).Take(1).Single();
  label6.Text = name6.Value.ToString();

  var name9 = xmlTick.Descendants("Reservation")
  .Attributes("ID")
  .Select(x => x).Take(1).Single();
  label5.Text = name9.Value.ToString();

  var name10 = xmlTick.Descendants("Seat")
  .Attributes("Preference")
  .Select(x => x).Take(1).Single();
  label9.Text = name10.Value.ToString();
}
}
```

Εντέλει το εισητήριο μας είναι το παρακάτω βασισμένο με τα στοιχεία που δώσαμε πριν:

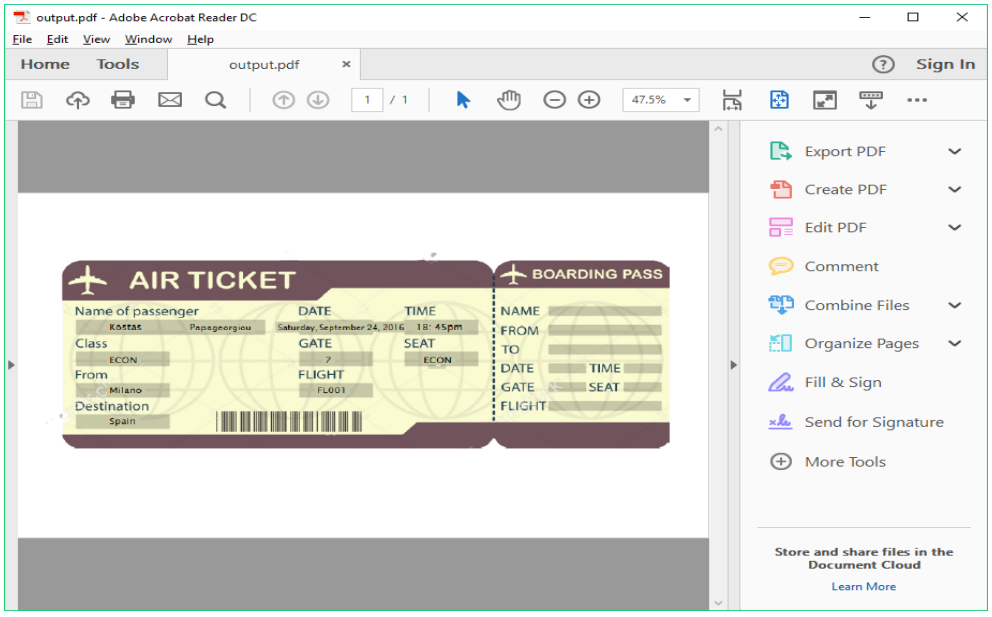


Πατώντας το button1 μπορούμε να στείλουμε το εισητήριο για εκτύπωση σε μορφη pdf με όνομα output.pdf Ο κώδικας για να εκτυπώσουμε το εισητήριο σε PDF είναι (Χρησιμοποιήθηκε το itextsharp):

```
private void button1_Click(object sender,
    EventArgs e)
{
 using (Bitmap b = new Bitmap(800, 370))
 {
    using (Graphics g = Graphics.FromImage(b))
    {
      g.CopyFromScreen(this.Location, new Point(0,
          0), this.Size);
```

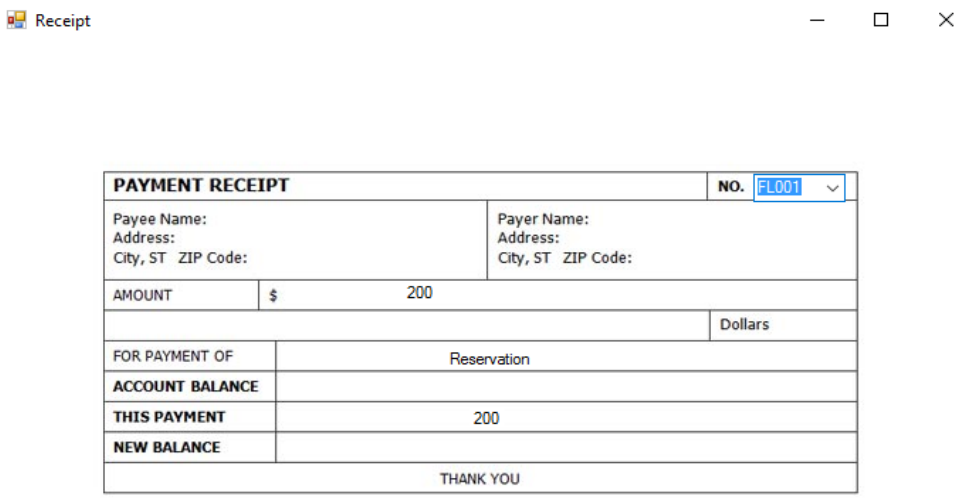
```
    }
    Document doc = new Document();
    iTextSharp.text.Image i = iTextSharp.text.Image
        .GetInstance(b, System.Drawing.Imaging.
        ImageFormat.Bmp);
    PdfWriter writer = PdfWriter.GetInstance(doc,
        new FileStream(@"C:\Users\admin\Desktop\
        output.pdf", FileMode.Create));
    doc.SetPageSize(new iTextSharp.text.Rectangle
    (this.Size.Width + doc.LeftMargin + doc.
        RightMargin,
    this.Size.Height + doc.TopMargin + doc.
        BottomMargin));
    doc.Open();
    doc.Add(i);
    doc.Close();
  }
}
```

Τελικά βλέπουμε το εισητήριο σε έγγραφο pdf, με το όνομα output.pdf απο εναν PDF reader (στη προκειμένη τον Adobe Reader)



Απόδειξη

Για τις πτήσεις υπάρχει απόδειξη πληρωμής. Επιλέγουμε απο το combobox (που διαβάζει το έγγραφο reservation.xml) το ID της πτήσης στην προκειμενη το FL001, και βλέπουμε την απόδειξη και το ποσό:



Ο παρακάτω κώδικας είναι πειραματικός για μελλοντική επέκταση και υπολογίζει την τιμή βάσει του πότε γίνεται η κράτηση. (δεν υποστηρίζεται απο τη συγκεκριμένη εφαρμογή)

```
public double calcFirstClassPrice()
{
XDocument price = XDocument.Load("schedulemanager.
    xml");
//vriskei thn hmerominia sto eggrafo XML
var date = price.Descendants("Aircraft")
.Attributes("mdy")
.Select(x => x).Take(1).Single();
//metatrepei se DateTime to apotelesma
DateTime convertedDate = Convert.ToDateTime(date);
TimeSpan ts = convertedDate - DateTime.Now;

//metatrepetai se double h timi apo to eggrafo XML
var priceToBeCalculated = (double)price.
    Descendants("schedule")
.Elements("Price_First")
.Select(x => x).Take(1).Single();
double FinalPrice = priceToBeCalculated;
if (ts.Days >30*2) // krathsh pano apo dyo mhnes
{
FinalPrice = priceToBeCalculated * 0.50;
}else if(ts.Days >30) //krathsh pano apo ena mhna
{
FinalPrice = priceToBeCalculated * 0.40;
}else if(ts.Days> 2* 7)//krathsh pano apo 2
    evdomades
{
FinalPrice = priceToBeCalculated * 0.25;
}
```

```
else if (ts.Days > 2 * 7)//krathsh pano apo 2
   evdomades
{
FinalPrice = priceToBeCalculated * 0.10;
}
return FinalPrice;
}
```

Επίλογος

Η εφαρμογή αυτή ήταν μια ματία στις δυνατότητες της LINQ to XML και στο πως μπορούμε να χρησιμοποιήσουμε αντί για παραδοσιακές λύσεις SQL. Η εφαρμογή μπορεί να επεκταθεί και άλλο προσθέτοντας δυνατότητες καθώς επιτελεί μόνο βασικές λειτουργίες της κράτησης για μία αεροπορική εταιρεία.

Î Î±ÏÎ¬ÏÏÎ·Î¼Î± Î±Í´

Βιβλιογραφία

https://en.wikipedia.org/wiki/Airline$_reservations_system$
$ProjectLINQwebsitehttp : //go.microsoft.com/fwlink/?LinkId =$
51461
$http : //stackoverflow.com/questions/7105505/linq - aggregate -$
$algorithm - explained$
$http : //stackoverflow.com/questions/1578778/using - iqueryable -$
$with - linq$
$https : //msdn.microsoft.com/en - us/library/mt693072.aspx$
$http : //www.xml.com/pub/a/98/10/guide0.html?page = 3AEN179$
$https : //el.wikipedia.org/wiki/\%CE\%92\%CE\%AC\%CF\%83\%CE\%B7_\%CE\%B4\%CE\%B5\%C$
$isa.teipir.gr/files/projects/database2.ppt$
$http : //www.rpbourret.com/xml/XMLAndDatabases.htm$
$http : //www.stylusstudio.com/images/screenshots/internal_dtd.gif$
$http : //dide.flo.sch.gr/Plinet/Tutorials/Tutorials-DataBasesTheory.html$
$http : //panacea.med.uoa.gr/topic.aspx?id = 492$
$https : //el.wikipedia.org/wiki/\%CE\%94\%CE\%B5\%CE\%B4\%CE\%BF\%CE\%BC\%CE\%AD$
$http : //msdn.microsoft.com/en - us/vstudio/bb688087.aspx$[]
$http : //cdn1.itpro.co.uk/sites/itpro/files/8/05//big_data_0.jpg$
$https : //en.wikipedia.org/wiki/Data_model$
$http : //www.codeproject.com/Articles/18116/LINQ-Introduction-$
$Part - Of$
$https : //www.rustemsoft.com/images/validateXML1.jpg$
$http : //sqlmag.com/site-files/sqlmag.com/files/archive/sqlmag.com/content/content/$
$http : //www.openqm - zumasys.com$
$http : //www.tutorialspoint.com/xml/xml_databases.htm$
$https : //social.msdn.microsoft.com/Forums/$