ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και υλοποίηση εφαρμογής σε μια μικρή επιχείρηση με VISUAL STUDIO και C#

Του φοιτητή
Νικολέττη Χαράλαμπου
Αρ. Μητρώου: 06/3150

Επιβλέπων καθηγητής
κ. Κουμπή Βασιλάντα

Θεσσαλονίκη 2011

ΠΡΟΛΟΓΟΣ

Είναι κοινή διαπίστωση, ότι ένα από τα μεγαλύτερα προβλήματα των επιχειρήσεων, είναι ο χειρισμός του όγκου εργασίας που αφορά την συντήρηση των δεδομένων που προκύπτουν από τις καθημερινές συναλλαγές τους. Συναλλαγές δηλαδή που έχουν να κάνουν είτε με αγοραπωλησίες, είτε μισθοδοσίες, είτε με οποιοδήποτε επιχειρησιακό στοιχείο, που θεωρείται σημαντικό και πρέπει να καταγραφεί .Το πρόβλημα, μάλιστα, καθίσταται ακόμα πιο περίπλοκο όταν απαιτείται επαναφορά των δεδομένων.

Αναντίρρητα, άμεση λύση στο πρόβλημα, δίνει η ανάπτυξη λογισμικών συστημάτων, τα οποία μπορούν να υποστηρίξουν τις πιο πάνω διαδικασίες ανάλογα με τις εκάστοτε ανάγκες. Κατ΄ επέκταση, απαιτείται ένα λογισμικό, το οποίο να μπορεί να διαχειρίζεται επιχειρησιακά ζητήματα που έχουν να κάνουν με υπολογισμούς και κυρίως με αποθήκευση και ανάκτηση πληροφορίας.
Ως εκ τούτου, με τη χρήση τέτοιων συστημάτων επιτελείται αυτοματοποίηση των επιχειρησιακών διεργασιών, μειώνοντας σημαντικά το φόρτο εργασίας, τα έξοδα και τις απώλειες μιας επιχείρησης και ότι άλλο αυτά συνεπάγονται.

Συνακόλουθα, σκοπός της παρούσας πτυχιακής εργασίας είναι η ανάλυση, σχεδίαση και ανάπτυξη μιας τέτοιας εφαρμογής, η οποία θα χρησιμοποιηθεί σε μια επιχείρηση με στόχο την αυτοματοποίηση των εταιρικών διαδικασιών. Διαδικασίες, δηλαδή που έχουν να κάνουν με τη διαχείριση πελατών και προμηθευτών, την καταμέτρηση και καταγραφή των αποθεμάτων, την έκδοση τιμολογίων και αποδείξεων καθώς επίσης και την παροχή διαφόρων άλλων χρήσιμων αναφορών.

ΠΕΡΙΛΗΨΗ

Στόχος της παρούσας εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός ολοκληρωμένου dedicated συστήματος σε μια εταιρεία κοπής και επεξεργασίας μαρμάρων.
Η ανάγκη αυτή προέκυψε, εξαιτίας της επέκτασης της εταιρείας και των δυσκολιών που παρουσιάστηκαν, τόσο στη διαχείριση των διαφόρων συναλλαγών της, όσο και στο θέμα της άμεσης και γρήγορης εξυπηρέτησης των πελατών της.

Επιπρόσθετα, ο λόγος που επιλέχθηκε η εφαρμογή αυτή να απευθύνεται σε μια τέτοια εταιρεία, είναι για να καταστεί εφικτή, μελλοντικά, η χρησιμοποίηση της και από άλλες εταιρείες και επιχειρήσεις με παρόμοιες δραστηριότητες και απαιτήσεις, αφού βεβαίως γίνονται, κάθε φορά, οι απαραίτητες αλλαγές και τροποποιήσεις, ανάλογα με τα δεδομένα και τις ανάγκες της κάθε εταιρείας. Αξιοσημείωτο είναι σαφέστατα και το γεγονός ότι οι μετατροπές που θα απαιτούνται είναι ελάχιστες, στοιχείο που καθιστά την εφαρμογή αυτή ιδιαίτερα εύχρηστη.

Εκτός από τα πιο πάνω, η εργασία αποσκοπεί επιπλέον στην εκμάθηση της γλώσσας προγραμματισμού C# , του SQL Server αλλά και των Crystal Reports στο φιλικό περιβάλλον του Visual Studio και ταυτόχρονα στην σχεδίαση και υλοποίηση μια μικρής ολοκληρωμένης εφαρμογής.

Επίσης, ο βασικός λόγος που επιλέχθηκε η εφαρμογή να γραφεί σε C#, είναι για επίτευξη της διεύρυνσης των γνώσεων μου στον προγραμματισμό, μαθαίνοντας έτσι ακόμη μια γλώσσα προγραμματισμού, αλλά παράλληλα και λόγω της ευρείας χρήσης της γλώσσας αυτής στη χώρα μου για Desktop programming. Κατά συνέπεια, θεωρήθηκε πως, αν το πρακτικό μέρος της πτυχιακής μου εργασίας υλοποιηθεί σε αυτή τη γλώσσα προγραμματισμού, θα διευκολύνει σε πολύ μεγάλο βαθμό τη δραστηριοποίηση και την ευελιξία μου, στα πλαίσια της προσεχούς επαγγελματικής μου αποκατάστασης.

Εν συνεχεία, λαμβάνοντας υπόψιν τη λειτουργικότητα που πρέπει να διέπει την εφαρμογή αυτή, σ'ένα πρωταρχικό στάδιο, σε συνεργασία με τα στελέχη της εταιρείας, έγινε καταγραφή και ανάλυση των αναγκών και των απαιτήσεων της

εταιρείας, τόσο για τα δεδομένα που πρέπει να συντηρούνται, όσο και για τους στόχους λογισμικού, τις υπηρεσίες δηλαδή που πρέπει να παρέχει το σύστημα.
Ακολούθως, με βάση τα πιο πάνω δεδομένα και αξιολογώντας τις καταστάσεις, καθορίστηκε ως κατάλληλη Μεθοδολογία για υλοποίηση του συστήματος η «Ανάπτυξη με Προτυποποίηση».

Στα κεφάλαια που ακολουθούν, γίνεται σύντομη περιγραφή των προγραμμάτων που χρησιμοποιήθηκαν και περιγράφονται οι απαιτήσεις του συστήματος για την υποστήριξη των προγραμμάτων αυτών.
Επίσης γίνεται ανάλυση των στόχων λογισμικού, όπως αυτοί προέκυψαν μετά από συζητήσεις με την εν λόγω εταιρεία.
Έπειτα, ακολουθεί η επεξήγηση της εφαρμογής σε τρεις φάσεις: i) επεξήγηση της βάσης και ανάλυση του διαγράμματος της βάσης , ii) γενική περιγραφή του κώδικα και iii) επεξήγηση του User Interface, με παραπομπές σε τμήματα κώδικα που βρίσκονται πίσω από αυτό.
Τέλος, αναφέρονται οι τεχνικές που χρησιμοποιήθηκαν, ώστε να οδηγήσουν σε ένα θετικό αποτέλεσμα όσον αφορά την Ποιότητα και Αξιοπιστία του λογισμικού.

ABSTRACT

The aim of this thesis is the design and development of an integrated, dedicated information system for a company specialized in marble stone manufacturing.

The new challenges that occurred, due to the recent expansion of operations of the company, cover the areas of information and transactions management in order to enable a high level customer experience.

Throughout the design process of the application has been taken into account the possibility of future usage of the software from other businesses of the same market sector. For this reason design decisions have been taken in order to facilitate future modifications and adaptations in new requirement with the least future development cost.

By undertaking this thesis I familiarized myself with the programming language C#, the Microsoft SQL Server, the Crystal Report and the development environment of Microsoft Visual Studio. I also gained an on hands experience of the development cycle of a complete software system.

The choice of C# was made in order to expand my knowledge in programming by learning one of the most widely used programming language for developing desktop applications. The skills that I acquired will help me greatly in my future professional career.

The first stage of the development process was to define the functionality of the application. Interviews were contacted with the personnel of the company in order to gain insights of the needs and requirements of the business. After the interviews the data collected were analyzed in order to specify the data models and the functionality that was needed to be offered by the application. Also through the analysis of the data was decided that the most appropriate development model was the “Development by prototyping”.

In the first section of the thesis there are descriptions of the software tools that were used and the system requirements that are needed for them. In the next section there is an analysis of the functionality of the application. Then follows the presentation of the application in three parts: i) the database schema and the database diagram presentation ii) general description of the code iii) the user interface with references to the code behind it.

Finally there is a presentation of the development techniques that were used in order to ensure a high quality to the end product.

ΕΥΧΑΡΙΣΤΙΕΣ

Στο σημείο αυτό, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες και την ευγνωμοσύνη μου προς την κ. Κουμπή Βασιλάντα, τόσο για την εμπιστοσύνη που μου επέδειξε δίνοντας την συγκατάθεση της να ασχοληθώ στην πτυχιακή μου εργασία με το θέμα που επιθυμούσα, παρέχοντας μου έτσι την δυνατότητα να διευρύνω τις γνώσεις μου στο αντικείμενο που με ενδιαφέρει άμεσα, όσο και για την αμέριστη βοήθεια και την μεγάλη προθυμία της να αναλάβει την καθοδήγηση μου και την παρακολούθηση της πτυχιακής μου εργασίας μέχρι τέλους, παρά την διακοπή της συνεργασίας της με το τμήμα.

Επίσης επιθυμώ να ευχαριστήσω και όλους όσους συνέλαβαν, γενικότερα, με την υποστήριξη τους στην ολοκλήρωση της πτυχιακής μου εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Είναι αναμφισβήτητο γεγονός, ότι μια από τις βασικότερες δυσκολίες των επιχειρήσεων είναι η καταγραφή και η αρχειοθέτηση των καθημερινών τους συναλλαγών. Η δυσκολία δε, γίνεται μεγαλύτερη όταν χρειαστεί οι πληροφορίες να ανακληθούν.

Με έναυσμα, λοιπόν, την πτυχιακή μου εργασία και αφετηρία την πιο πάνω διαπίστωση, θέλησα να ασχοληθώ με την ανάλυση, την σχεδίαση και την ανάπτυξη ενός λογισμικού συστήματος, με το οποίο να καθίσταται εφικτή και συνάμα εύκολη η διαχείριση επιχειρησιακών ζητημάτων που έχουν να κάνουν, όχι μόνο με υπολογισμούς, αλλά κατά κύριο λόγο με την αποθήκευση και την ανάκτηση πληροφοριών.

Κατ' επέκταση, αξιοποιώντας τη δυνατότητα που παρέχει η ανάπτυξη των λογισμικών συστημάτων, η πτυχιακή αυτή εργασία καταπιάνεται με το σχεδιασμό και την υλοποίηση ενός ολοκληρωμένου dedicated συστήματος για μια εταιρεία η οποία δραστηριοποιείται στον τομέα κοπής και επεξεργασίας μαρμάρων.

Βεβαίως, το θετικό είναι ότι το πρόγραμμα αυτό θα εξυπηρετήσει, όχι μόνο την συγκεκριμένη επιχείρηση, αλλά μπορεί κάλλιστα να χρησιμοποιηθεί και από άλλες επιχειρήσεις παρόμοιας δραστηριοποίησης, αφού σαφώς γίνουν οι απαραίτητες, αλλά σημειωτέο ελάχιστες αλλαγές, ώστε η εφαρμογή να ανταποκρίνεται στις απαιτήσεις της εκάστοτε επιχείρησης.

Παράλληλα, στα πλαίσια της υλοποίησης της εργασίας αυτής, επιδιώκεται η εκμάθηση της γλώσσας C#, του SQL Server, αλλά και των Crystals Reports στο φιλικό περιβάλλον του Visual Studio, ώστε να αποκτήσω ακόμη ένα γνωστικό εφόδιο, το οποίο μου είναι απόλυτα χρήσιμο στην επαγγελματική μου αποκατάσταση, καθώς η συγκεκριμένη γλώσσα προγραμματισμού χρησιμοποιείται ευρέως στη χώρα μου.

Σαφέστατα, απαραίτητη προϋπόθεση για να μπορέσει να δομηθεί το εν λόγω λογισμικό σύστημα, ήταν η συνεργασία με τα στελέχη της εταιρείας ώστε να καταγραφούν οι απαιτήσεις και οι ανάγκες της εταιρείας, ενώ, εν συνεχεία, με την αξιολόγηση των δεδομένων αυτών, επιλέχθηκε ως κατάλληλη μεθοδολογία για υλοποίηση του συστήματος η ανάπτυξη με προτυποποίηση.

Στα κεφάλαια που ακολουθούν, γίνεται σύντομη παρουσίαση των προγραμμάτων που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, καθώς επίσης και των απαιτήσεων συστήματος για την υποστήριξη αυτών. Στη συνέχεια, αναλύονται οι στόχοι λογισμικού όπως αυτοί προέκυψαν από τις απαιτήσεις της εν λόγω εταιρείας και γίνεται σύντομη περιγραφή του μοντέλου ανάπτυξης που επιλέχθηκε, με βάση τις πιο πάνω απαιτήσεις. Ακολούθως, γίνεται μια λεπτομερής παρουσίαση της βάσης δεδομένων που δημιουργήθηκε καθώς και του διαγράμματος που παράχθηκε από αυτήν. Έπειτα, γίνεται μια γενική περιγραφή του κώδικα της εφαρμογής όσον αφορά την χρησιμότητα των κλάσεων και των projects που δημιουργήθηκαν. Ακολουθεί η ανάλυση του τρόπου με τον οποίο σχεδιάστηκε το User Interface, με παραπομπές σε τμήματα κώδικα που βρίσκονται πίσω από αυτό. Τέλος, παρουσιάζονται οι τεχνικές και οι μεθοδολογίες που χρησιμοποιήθηκαν ώστε να διατηρηθεί σε υψηλά επίπεδα η ποιότητα και αξιοπιστία του λογισμικού.

Εν συντομία, η πτυχιακή αυτή εργασία, αναλυτική παρουσίαση της οποίας υπάρχει στα κεφάλαια που ακολουθούν, δεν αποτελεί απλώς ένα θεωρητικό κείμενο, αλλά με την πρακτική της εφαρμογή, δίνει άμεσες λύσεις στη συγκεκριμένη επιχείρηση, και ταυτόχρονα η όλη πορεία για την εκπόνηση της συνέβαλε σε μεγάλο βαθμό στη διεύρυνση των γνώσεων μου.

ΚΕΦΑΛΑΙΟ 1: Απαιτήσεις συστήματος

Εισαγωγή:

Στο κεφάλαιο αυτό θα γίνει μια σύντομη παρουσίαση των προγραμμάτων που χρησιμοποιήθηκαν για τη σχεδίαση και ανάπτυξη της εφαρμογής. Επίσης θα αναφερθούν και οι απαιτήσεις συστήματος για την υποστήριξη των προγραμμάτων.

Τα προγράμματα που χρησιμοποιήθηκαν

Τα προγράμματα που χρησιμοποιήθηκαν για την παρούσα πτυχιακή εργασία είναι τα πιο κάτω:

- Microsoft Visual Studio 2008 – C sharp (C#)
- Microsoft SQL Server 2005
- Crystal Reports 2008

Η εφαρμογή αυτή είναι Windows Application, οπότε μπορεί να υποστηριχτεί μόνο σε λειτουργικό σύστημα Windows, αρκεί να πληρούνται οι πιο κάτω απαιτήσεις συστήματος όσον αφορά το υλικό και το λογισμικό.

Απαιτήσεις συστήματος ανά πρόγραμμα:

Απαιτήσεις συστήματος για Visual Studio 2008:

Windows Seven, ή Windows Vista, ή Windows XP με Service Pack 2 or above, ή Windows Server 2003 με Service Pack 1 or above, ή Windows Server 2008

Ελάχιστες Απαιτήσεις υλικού:

- Επεξεργαστής 1,6 GHz, μνήμη 384 MB, οθόνη 1024x768, σκληρός δίσκος 504 RPM.

- 1,22 GB διαθέσιμου χώρου στο σκληρό δίσκο για την ελάχιστη εγκατάσταση ή 2GB διαθέσιμου χώρου στο δίσκο για την πλήρη εγκατάσταση.

Προτεινόμενες Απαιτήσεις υλικού:

- Επεξεργαστής 2,2GHz ή ταχύτερος, μνήμη 1024 MB ή περισσότερη, οθόνη 1280x1024, σκληρός δίσκος 7200 RPM ή ταχύτερος (για Windows Vista συνιστάται επεξεργαστής 2,4 GHz και μνήμη 768 MB).
- 1,22 GB διαθέσιμου χώρου στο σκληρό δίσκο για την ελάχιστη εγκατάσταση ή 2GB διαθέσιμου χώρου στο δίσκο για την πλήρη εγκατάσταση.

Απαιτήσεις συστήματος για Microsoft SQL Server 2005 Express Edition:

- Επεξεργαστής Pentium III ή περισσότερο
- Ελάχιστη ταχύτητα Επεξεργαστή 500 MHz (συνιστάται τουλάχιστον 1GHz) και να είναι το ελάχιστο 192 MB
- Συνιστάται 512 MB RAM ή περισσότερο

Απαιτήσεις συστήματος για Crystal Reports 2008:

- Intel Pentium III ή νεότερος επεξεργαστής
- Τουλάχιστον 256 MB RAM (συνιστάται 512 MB ή περισσότερο)
- 300 MB διαθέσιμος χώρος στο σκληρό δίσκο (συνιστάται 600 MB ή περισσότερο)

Τέλος, αξίζει να σημειωθεί, ότι στον υπολογιστή στον οποίο θα τρέχει η παρούσα εφαρμογή, απαιτείται η εγκατάσταση του SQL Server 2005 ή και νεότερη έκδοση, για να ενσωματωθεί η βάση της εφαρμογής σε αυτόν.
Μάλιστα, αν η εφαρμογή τρέχει σε ένα δίκτυο από υπολογιστές, η εγκατάσταση του SQL Server απαιτείται μόνο σε ένα από τους υπολογιστές που συγκροτούν το δίκτυο και οι υπόλοιποι θα ενώνονται με αυτόν.
Επίσης απαιτείται η εγκατάσταση του .NET framework 3.5 ή και νεότερη έκδοση.

Σύντομη περιγραφή των προγραμμάτων που χρησιμοποιήθηκαν

Visual Studio 2008:

Το Visual Studio δημιουργήθηκε για το λειτουργικό σύστημα Windows, έκανε πρεμιέρα την άνοιξη του 1995 και μετά από την πρώτη έκδοση έχει ακολουθήσει πληθώρα από νέες και βελτιωμένες εκδόσεις. Προς το τέλος του 2007 κυκλοφόρησε η έκδοση 9.0 ή διαφορετικά Visual Studio 2008 το οποίο υποστηρίζει την έκδοση 3.5 του .NET Framework.

Το Visual Studio 2008 προσφέρει ένα ευρύ φάσμα από εργαλεία, χρήσιμα για όλες τις φάσεις ανάπτυξης λογισμικού, συμπεριλαμβανομένης της δημιουργίας, της δοκιμής, της ανάπτυξης, της ενοποίησης και τέλος της διαχείρισης. Επιπλέον, επιτρέπει στους προγραμματιστές να δημιουργήσουν εφαρμογές διαφόρων ειδών, είτε πρόκειται για απλές εφαρμογές, είτε για ιστοσελίδες.

Επίσης υποστηρίζει διάφορες γλώσσες προγραμματισμού, όπως :

- Visual C++
- Visual C#
- ASP.NET
- Visual Basic.NET

Συνάμα προσφέρει εργαλεία ανάπτυξης εφαρμογών, όπως:

- **Σύστημα έργου** - Χρησιμοποιείται για τη διαχείριση των δεδομένων, τα οποία απαιτούνται για την ανάπτυξη εφαρμογών.
- **Επεξεργασία κώδικα** - Εργαλεία για δημιουργία και τροποποίηση κειμένου και κώδικα.
- **Επανασχεδιασμός προγράμματος και διόρθωση σφαλμάτων** Εργαλεία για τη βελτίωση κώδικα και τον εντοπισμό και την επίλυση λογικών σφαλμάτων.
- **Δημιουργία αναφορών**
- **Πλατφόρμα** - Εργαλεία για τη δημιουργία εφαρμογών για τις τεχνολογίες Office, Windows CE, .NET και Windows.

Για προχωρημένους - Εργαλεία για το σχεδιασμό, την υλοποίηση, την αξιοποίηση, την ανάλυση την επαλήθευση και την αξιολόγηση της προόδου ανάπτυξης των εφαρμογών.

Τι είναι το .NET Framework :

Το .NET Framework είναι μια βιβλιοθήκη για υπολογιστές με λειτουργικό σύστημα Windows. Παρέχει στους προγραμματιστές τη δυνατότητα να χρησιμοποιήσουν διάφορες εξελιγμένες λειτουργίες και η εφαρμογή τους να είναι απόλυτα συμβατή με άλλα συστήματα τα οποία υποστηρίζουν και έχουν εγκατεστημένο το .NET Framework.

Η αρχιτεκτονική του χωρίζεται στα τέσσερα πιο κάτω τμήματα:

1) Common Language Runtime (CLR)

2) Βιβλιοθήκες (Class Libraries)

3) Γλώσσες Προγραμματισμού (C#, VC++, VB.NET, Jscript.NET)

4) ASP.NET

Σκοπός του .NET Framework είναι η επίτευξη τριών στόχων:

Κατά πρώτο λόγο, πρέπει να κάνει τις Windows εφαρμογές πιο αξιόπιστες βελτιώνοντας την ασφάλεια τους. Κατά δεύτερο να απλουστεύσει την ανάπτυξη Web εφαρμογών και υπηρεσιών (Web Services), οι οποίες θα τρέχουν και σε φορητές συσκευές και τρίτο να παρέχει ένα σύνολο βιβλιοθηκών, οι οποίες θα μπορούν να λειτουργήσουν με πολλές γλώσσες.

Συμπληρωματικά, το .NET Framework προορίζεται για να χρησιμοποιείται από τις περισσότερες νέες εφαρμογές, που δημιουργούνται για την πλατφόρμα των Windows.

Η αντικειμενοστρεφής γλώσσα C-sharp:

Η γλώσσα C# αναπτύχθηκε το 2000 και σχετίζεται άμεσα με την C, C++ και Java. Αυτό δεν είναι τυχαίο, καθώς αυτές είναι οι τρεις πιο διαδεδομένες και χρησιμοποιημένες γλώσσες προγραμματισμού στον κόσμο. Άλλωστε η σύνταξη της C# έγινε με κατανοητή υποδομή παρεμφερής της C, C++ και Java, έτσι ώστε να προσφέρει ένα βατό δρόμο μετανάστευσης από αυτές προς εκείνη.

Επιπρόσθετα, η C# εστίασε στις ελλείψεις που προϋπήρχαν στη γλώσσα προγραμματισμού Java και με συγκεκριμένες βελτιώσεις και καινοτομίες έλυσε τα προβλήματα αυτά.

Αξίζει να σημειωθεί πως η C# προσπαθεί να συνθέσει την C++ και την Java σε αντίθεση με την Java που ήταν απλά θυγατέρα της C++, γεγονός που καθιστά τη C# «συνδυαστικό» αντικαταστάτη των δύο αυτών σύγχρονων γλωσσών προγραμματισμού.

Επίσης η C# χρησιμοποιεί την Αντικειμενοστρεφή Τεχνολογία, την οποία κληρονόμησε από την C++. Με τον όρο Αντικειμενοστρεφή (object-oriented) δείχνει ένα μηχανισμό, ένα τρόπο σκέψης στην ανάπτυξη λογισμικού, που είναι στραμμένος σε Αντικείμενα (objects), τα οποία συγκροτούν τις βασικές δομικές μονάδες της τεχνολογίας αυτής. Το αντικείμενο είναι το βασικό συστατικό της σχεδίασης λογισμικού, είναι η γενίκευση της σύνθετης μεταβλητής και περιλαμβάνει από εγγραφές μέχρι συναρτήσεις, διαδικασίες και μεθόδους.

Το αντικείμενο έχει τις ακόλουθες βασικές ιδιότητες:

Κατάσταση: Έχει να κάνει με τις τιμές του αντικειμένου σε επίπεδο μεταβλητών μνήμης.

Συμπεριφορά: Είναι ο τρόπος που ανταποκρίνεται το αντικείμενο σε κλήσεις από το περιβάλλον του.

Ταυτότητα: Είναι η διάκριση ενός αντικειμένου από όλα τα άλλα που ορίζονται στον ίδιο τύπο δεδομένων.

Σήμερα, η γλώσσα προγραμματισμού C# θεωρείται ένα βασικό εργαλείο λογισμικού ανάπτυξης εφαρμογών και αποτελεί σημαντικότατο εφόδιο για την ειδικότητα των μηχανικών λογισμικού, που θέλουν να εφοδιαστούν με την ικανότητα σχεδιασμού και ανάπτυξης Πληροφοριακών Συστημάτων, είτε σε τοπικό δίκτυο παραγωγής, είτε σε διαδικτυακό επίπεδο στον παγκόσμιο ιστό.

C# και Σχεσιακές Βάσεις Δεδομένων (RDBMS):

Σχετικά με τις κλασικές βάσεις δεδομένων που βασίζονται σε πολύπλοκες ή μη δομές δεδομένων, η C# στο σημείο αυτό, ακολουθεί κανονικά τη Java με random access τεχνικές και αντίστοιχες κλάσεις.
Η C# για σχέσεις με Σχεσιακές βάσεις δεδομένων - RDBMS χρησιμοποιεί το **ADO.NET** με μια πληθώρα δυνατοτήτων, που θεωρείται όχι μόνο απλά ισοδύναμο του JDBC[1] αλλά υπερτερεί σε σχέση με αυτό σε παρεχόμενες ευκολίες, προσπελάσεις και ασφάλεια.

Το **ADO.NET** συγκροτείται από δύο βασικά συστατικά, τον **Data Provider** ο οποίος περιλαμβάνει κάθε διοίκηση στις πηγές των δεδομένων, αλλά και προσπελάσεις αυτών και το **DataSet** το οποίο λειτουργεί αποκλειστικά σε επίπεδο μνήμης σαν μια αποθήκη δεδομένων από διάφορες πηγές. Επίσης το DataSet, οποτεδήποτε χρειαστεί μπορεί να σταλεί για κανονική αποθήκευση των δεδομένων αυτών.

Data Provider

Ο Data Provider αποτελείται από τις ακόλουθες στοιχειώδεις δραστηριότητες:

Connection: Παρέχει τη σύνδεση με το Data Source[2] (τη σύνδεση με την βάση ουσιαστικά), έλεγχο **transaction** και δημιουργία **command components**.

Ορισμένες χρήσιμες παράμετροι για να ορίσουμε την κλάση SqlConnection είναι οι εξής:

[1] JDBC – είναι το αντίστοιχο εργαλείο του ADO.NET, που χρησιμοποιείται από την Java

[2] Data Source – Προσδιορίζει το όνομα ή τη διεύθυνση της βάσης

- **Application Name ->** Το όνομα της εφαρμογής που χρησιμοποιείται για τη σύνδεση.
- **Data Source ->** Το όνομα ή τη διεύθυνση του Sql Server.
- **Database ->** Το όνομα της βάσης δεδομένων.
- **Integrated Security ->** Καθορίζει την ασφάλεια της σύνδεσης.
- **Κωδικός** για σύνδεση στον Sql Server.
- **UserID** για σύνδεση στον Sql Server.

Command: Περιέχει τη διαχείριση δεδομένων και εντολές **ερωταποκρίσεων** προς τη βάση **(queries).** Παρέχει επίσης υποστήριξη για τη διαχείριση παραμετρικών ερωταποκρίσεων.

Data Reader: Παρέχει μια **μέθοδο ανάγνωσης** των αποτελεσμάτων μιας εκτέλεσης ερωταπόκρισης.

Data Adapter: Αποτελεί μια **γέφυρα** ανάμεσα στα δεδομένα της βάσης και το DataSet.

DataSet

Όπως αναφέρθηκε και πιο πάνω, ένα ιδιαίτερα χρήσιμο component για επικοινωνία με τη βάση δεδομένων είναι το DataSet. Ορισμένες από τις βασικές λειτουργίες του DataSet είναι οι εξής:

1) DataSet.Tables.add(ΌνομαΠίνακα) -> Προσθήκη Πίνακα
2) DataSet.Tables[ΌνομαΠίνακα] -> Ανάκτηση ενός συγκεκριμένου πίνακα
3) DataSet.Tables.Remove[ΌνομαΠίνακα] -> Διαγραφή συγκεκριμένου πίνακα
4) DataTable.Columns.Add(ΌνομαΣτήλης) -> Προσθήκη στήλης πίνακα
5) DataTable.Columns[ΌνομαΣτήλης] -> Ανάκτηση συγκεκριμένης στήλης
6) DataTable.Columns.Remove(ΌνομαΣτήλης) -> Διαγραφή μιας συγκεκριμένης στήλης
7) DataTable.Rows.Add(DataRow) - > Προσθήκη γραμμής πίνακα
8) DataTable.Rows[RowIndex] -> Ανάκτηση μιας συγκεκριμένης γραμμής
9) DataTable.Rows.Remove(DataRow) -> Διαγραφή συγκεκριμένης γραμμής

10) DataRow[ΌνομαΣτήλης] -> Επεξεργασία δεδομένων ανά γραμμή

SQL Server 2005:

Η έρευνα και η ανάπτυξη στο χώρο των βάσεων δεδομένων είναι ενεργή εδώ και αρκετές δεκαετίες, από τη στιγμή που η αυτοματοποιημένη αποθήκευση και διαχείριση μεγάλου όγκου πληροφοριών έγινε αναγκαία.

Ο SQL Server βγήκε για πρώτη φορά το 1989 από την Microsoft σε συνεργασία με την Sybase και αποτελεί ένα ισχυρό Σχεσιακό Μοντέλο Διαχείρισης Βάσεων Δεδομένων (RDBMS - Relational Database Management System) με πολλές δυνατότητες και το οποίο χρησιμοποιείται ευρύτατα για τη διαχείριση μεγάλων Βάσεων Δεδομένων και αναπτύσσεται από την Microsoft .

Είναι σχεδιασμένος να λειτουργεί σε πλατφόρμες, οι οποίες κυμαίνονται από φορητούς υπολογιστές έως και μεγάλους Servers πολλαπλών επεξεργαστών.

Χρησιμοποιείται συνήθως ως σύστημα υποστήριξης για ιστοσελίδες ή ακόμα και για αποθήκευση δεδομένων σε τοπικά δίκτυα, όπου αυτός μπορεί να υποστηρίξει χιλιάδες ταυτόχρονους χρήστες.

Η κύρια μονάδα αποθήκευσης στοιχείων είναι μια βάση δεδομένων, η οποία αποτελείται από μια συλλογή πινάκων και κώδικα.

Οι κύριες γλώσσες που χρησιμοποιούνται είναι η T_SQL και η ANSI SQL.

Μετά από την πρώτη έκδοση του SQL Server, ακολούθησε μια σειρά από νέες, πιο εξειδικευμένες και με περισσότερες δυνατότητες εκδόσεις. Οι πιο πρόσφατες από αυτές είναι :

- SQLServer 2000 version 8.0 (Χρονολογία - 2000)
- SQLServer 2000 version 8.0- 64 bit Edition (Χρονολογία - 2003)
- SQLServer 2005 version 9.0 (Χρονολογία - 2005)
- SQLServer 2008 version 10.0 (Χρονολογία - 2008)
- SQL Azure version 10.25 (Χρονολογία - 2010)
- SQLServer 2008 R2 version 10.5 (Χρονολογία - 2010)

Ο **SQL Server 2005** με κωδική ονομασία «Yukon» κυκλοφόρησε τον Οκτώβριο του 2005 και είναι διάδοχος του SQL Server 2000. Πέραν από την υποστήριξη των σχεσιακών δεδομένων περιλαμβάνει εγγενή υποστήριξη για τη διαχείριση των

XML δεδομένων. Για το σκοπό αυτό ορίζεται ένας XML τύπος δεδομένων που μπορεί να χρησιμοποιηθεί είτε σε στήλες των πινάκων της βάσης είτε literals[3] σε Ερωτήματα (Queries).

Επίσης μερικά από τα σημαντικά γνωρίσματα που προστέθηκαν στην έκδοση αυτή είναι:

- ο **έλεγχος συναλλαγών** (transaction control).
- ο αυξημένος **έλεγχος εξαιρέσεων** και αντιμετώπισης λαθών (exception and error handling).
- μεγαλύτερη **διοικητική υποστήριξη.**
- **"MARS"** (Multiple Active Results Sets) Μια μέθοδος που επιτρέπει τη χρήση των συνδέσεων της βάσης για πολλαπλούς σκοπούς.
- **DMVs** (Dynamic Management Views) Οι προβολές αυτές είναι ειδικευμένες λειτουργίες που επιστρέφουν πληροφορίες για την κατάσταση του server, οι οποίες μπορούν να χρησιμοποιηθούν για την παρακολούθηση της σωστής του λειτουργίας.
- **CLR** (Common Language Runtime) Είναι ένα νέο module μέσω του οποίου ενσωματώνεται το .NET μέσα στον SQL Server. Με το module αυτό οι αποθηκευμένες διαδικασίες μπορούν να γραφούν σε οποιαδήποτε γλώσσα .NET συμπεριλαμβανομένου και της C#. Αυτό σημαίνει ότι ο SQL Server έχει όλες τις βιβλιοθήκες και τα πλεονεκτήματα του .NET, αυτόχθονα μέσα στο περιβάλλον του, τα οποία μπορεί να καλέσει οποιαδήποτε στιγμή.

Σχετικά με τον έλεγχο συναλλαγών θα γίνει εκτενέστερη περιγραφή σε επόμενο κεφάλαιο λόγω της χρήσης του κατά την υλοποίηση της εφαρμογής.

[3]literals – με τον όρο αυτό εννοούμε κάθε αναπαράσταση σε string, η τιμή του οποίου δεν πρέπει να είναι δεσμευμένη λέξη της Sql

Βασικοί τύποι δεδομένων που υποστηρίζονται από το SQL Server και χρησιμοποιήθηκαν στην εφαρμογή:

- int (για ακέραιους αριθμούς).
- decimal (για δεκαδικούς αριθμούς με δυνατότητα ορισμού του αριθμού των επιθυμητών ακέραιων και δεκαδικών ψηφίων).
- varchar (συμβολοσειρά/string με δυνατότητα ορισμού του επιθυμητού μεγέθους χωρητικότητας σε bytes).
- nvarchar (συμβολοσειρά/string με δυνατότητα ορισμού του επιθυμητού μεγέθους χωρητικότητας σε bytes - υποστήριξη συμβολοσειρών με Unicode χαρακτήρες).
- datetime (ημερομηνία και ώρα).
- bit (είναι αντίστοιχο του boolean, μπορεί να πάρει τις τιμές True και False).
- Varbinary (ο τύπος αυτός χρησιμοποιείται για να αποθηκεύσει τιμές που επιστρέφονται από μια μέθοδο κρυπτογράφησης).

Στο σημείο αυτό, αξίζει να σημειωθεί ότι δημιουργήθηκαν πίνακες, οι οποίοι λειτουργούν ως «τύποι», για την περιγραφή δεδομένων σε άλλους πίνακες και θα αναλυθούν σε επόμενο κεφάλαιο.

Τέλος, η έκδοση αυτή του SQL Server υπόσχεται αυξημένη ευελιξία, αξιοπιστία και ασφάλεια στις εφαρμογές της βάσης δεδομένων, καθώς επίσης ευκολότερη δημιουργία και ανάπτυξη, μειώνοντας έτσι την πολυπλοκότητα και την άγνοια που γεννιέται κατά διαχείριση των βάσεων.

Crystal Reports 2008:

Η εφαρμογή Crystal Reports αποτελεί ένα πανίσχυρο εργαλείο δημιουργίας επαγγελματικών αναφορών, το οποίο συνεργάζεται με όλες τις γνωστές εφαρμογές διαχείρισης βάσεων δεδομένων.
Βασίζεται σε ευέλικτη τεχνολογία και παρέχει ολοκληρωμένο έλεγχο για πρόσβαση σε δεδομένα και δυνατότητα παρουσίασης τους.
Υποστηρίζει πλήρως μεταβλητές, γραμματοσειρές Unicode, με πλήρη υποστήριξη της ελληνικής γλώσσας.

Παρέχει τη δυνατότητα ανάκτησης και μορφοποίησης ενός συνόλου αποτελεσμάτων από μια βάση δεδομένων ή άλλη πηγή/προέλευσης δεδομένων. Επίσης, περιέχει στοιχεία για τη μετατροπή των αρχικών δεδομένων με τα οποία τροφοδοτείται σε εκθέσεις ποιότητας, γραφήματα και διαγράμματα.

Τέλος, πρόσφατες δημοσκοπήσεις έχουν δείξει ότι το πενήντα τοις εκατόν των εφαρμογών που αναπτύσσονται χρησιμοποιούν Crystal Reports για αναφορές. Αυτό, δεν είναι τυχαίο γιατί το Crystal Reports της εταιρείας Business Object είναι η πλέον διαδεδομένη λύση για **reporting**, **analysis** και **information delivery**, καθώς υπάρχουν περισσότεροι από 35 οδηγοί για πρόσβαση σε βάσεις, ή οποιεσδήποτε άλλες πηγές δεδομένων και περισσότερες από 100 επιλογές **formatting** με έτοιμες παραμέτρους, διαγράμματα και hyperlinks, με αποτέλεσμα οι αναφορές που δημιουργούνται να είναι εύκολες και γρήγορες στη χρήση τους.

Υποστηριζόμενες Βάσεις Δεδομένων:

- Βάσεις Δεδομένων όπως PostgreSql, Sybase, IBM DB2, Ingres, Microsoft Access , Microsoft SQL Server, MySQL και Oracle.
- Spreadsheets όπως Microsoft Excel.
- Αρχεία κειμένου.
- HTML αρχεία XML.
- Groupware εφαρμογές όπως το Lotus Notes, Microsoft Exchange και Novel GroupWise.

Επιπλέον υποστηρίζεται οποιαδήποτε άλλη πηγή δεδομένων, η οποία είναι προσβάσιμη μέσω μιας υπηρεσίας Ιστού, ODBC, JDBC ή OLAB.

Επίλογος:

Συμπερασματικά, τα προγράμματα που αναφέρθηκαν στο κεφάλαιο αυτό, συνδέονται αρμονικά μεταξύ τους και ως εκ τούτου με την ορθή χρήση και αξιοποίηση τους, είναι εφικτό να υλοποιηθούν οι επιδιωκόμενοι στόχοι λογισμικού, για τους οποίους θα γίνει εκτενής αναφορά στο κεφάλαιο που ακολουθεί.

ΚΕΦΑΛΑΙΟ 2: Καταγραφή στόχων λογισμικού

Εισαγωγή:

Στο κεφάλαιο αυτό γίνεται περιγραφή των στόχων λογισμικού, δηλαδή, των ενεργειών και διαδικασιών, οι οποίες πρέπει να υποστηρίζονται από την εφαρμογή, όπως αυτές προέκυψαν από τις απαιτήσεις της εν λόγω εταιρείας.

Οι ενέργειες / διεργασίες που θα υποστηρίζονται από την εφαρμογή

Η εφαρμογή, εν συντομία, θα διαχειρίζεται ενέργειες που έχουν να κάνουν με τους πελάτες και προμηθευτές της εταιρείας, τα αποθέματα, και τις συναλλαγές της, όπως έκδοση τιμολογίων και αποδείξεων, ενώ παράλληλα θα παρέχει στους χρήστες μια σειρά από χρήσιμες αναφορές και εργαλεία που έχουν να κάνουν με σημαντικά θέματα της εταιρείας. Η χρήση των εργαλείων αυτών έχει σκοπό να προσφέρει επιπλέον δυνατότητες στη διαχείριση του όλου συστήματος.

Αναλυτικότερα...

Διαχείριση Πελατών:

Ο χρήστης θα έχει τη δυνατότητα εισαγωγής των βασικών/σημαντικών στοιχείων ενός νέου πελάτη στο σύστημα και συγκεκριμένα στη βάση του συστήματος.
Επίσης από το DataGridView (πίνακα) στο οποίο εμφανίζονται όλοι οι πελάτες με τα στοιχεία τους, ο χρήστης θα μπορεί να επιλέξει ένα από αυτούς και να κάνει μια ή περισσότερες από τις πιο κάτω ενέργειες :

1) Ενημέρωση των στοιχείων του Πελάτη, είτε σε περίπτωση μιας λανθασμένης εισαγωγής, είτε σε περίπτωση αλλαγής κάποιου από τα στοιχεία του συγκεκριμένου πελάτη.

2) Προβολή του ιστορικού του Πελάτη, δηλαδή αναλυτική εμφάνιση όλων των συναλλαγών που έχει κάνει ο συγκεκριμένος Πελάτης με την εταιρεία σε ένα DataGridView.
3) Προβολή του Ισοζυγίου(balance) όλων των λογαριασμών του Πελάτη.

4) Διαγραφή ενός συγκεκριμένου Πελάτη.

Διαχείριση Προμηθευτών:

Ο χρήστης θα έχει τη δυνατότητα εισαγωγής των βασικών στοιχείων ενός νέου Προμηθευτή στη βάση του συστήματος.
Επίσης από το DataGridView (πίνακα) στο οποίο εμφανίζονται όλοι οι Προμηθευτές με τα στοιχεία τους, ο χρήστης θα μπορεί να επιλέξει έναν από αυτούς και να κάνει μια ή περισσότερες από τις πιο κάτω ενέργειες :

1) Ενημέρωση των στοιχείων του Προμηθευτή, είτε σε περίπτωση μιας λανθασμένης εισαγωγής, είτε σε περίπτωση αλλαγής κάποιου από τα στοιχεία του συγκεκριμένου Προμηθευτή.

2) Προβολή του ιστορικού της εταιρείας με τον συγκεκριμένο Προμηθευτή, δηλαδή αναλυτική εμφάνιση όλων των συναλλαγών που έχει κάνει η εταιρεία με τον Προμηθευτή αυτό (στην ουσία με την συγκεκριμένη εταιρεία προμήθευσης) σε ένα DataGridView.

3) Προβολή του Ισοζυγίου(balance) όλων των λογαριασμών της εταιρείας για τον συγκεκριμένο Προμηθευτή.

4) Διαγραφή ενός συγκεκριμένου Προμηθευτή.

Διαχείριση Αποθεμάτων:

Ο χρήστης θα έχει τη δυνατότητα εισαγωγής των βασικών στοιχείων ενός νέου αποθέματος/προϊόντος στη βάση του συστήματος.
Επίσης από το DataGridView (πίνακα) στο οποίο εμφανίζονται όλα τα αποθέματα/προϊόντα με τα στοιχεία τους, ο χρήστης θα μπορεί να επιλέξει ένα από αυτά και να κάνει μια ή περισσότερες από τις πιο κάτω ενέργειες:

1) Ενημέρωση των στοιχείων του προϊόντος, είτε σε περίπτωση μιας λανθασμένης εισαγωγής, είτε σε περίπτωση αλλαγής κάποιου από τα στοιχεία του συγκεκριμένου προϊόντος.
2) Διαγραφή ενός συγκεκριμένου προϊόντος.

Τέλος, με την έκδοση ενός τιμολογίου και ανάλογα με τον τύπο του (τιμολόγιο από πώληση/αγορά), θα πρέπει ενημερώνεται αυτόματα ο πίνακας των Αποθεμάτων για τη νέα διαθέσιμη ποσότητα των συγκεκριμένων προϊόντων που βρίσκονται στο εκάστοτε τιμολόγιο.
Ο κύριος λόγος που θα γίνεται αυτό είναι για την πλήρη αυτοματοποίηση του συστήματος ώστε να δαπανάται ο ελάχιστος χρόνος από πλευράς εταιρείας για τη διαχείριση του.

Διαχείριση Τιμολογίων και Προσφορών:

Καταρχήν θα υπάρχουν δύο είδη/τύποι τιμολογίων, τα τιμολόγια που προέκυψαν από αγορές της εταιρείας και τα τιμολόγια που προέκυψαν από πωλήσεις της εταιρείας.
Οι προσφορές θα απευθύνονται από την εταιρεία προς τους πελάτες της.

Ο χρήστης θα έχει τη δυνατότητα εισαγωγής των βασικών στοιχείων ενός νέου τιμολογίου στο σύστημα και συγκεκριμένα στη βάση του συστήματος. Θα μπορεί να μεταφέρεται στον πίνακα Πελατών ή Προμηθευτών, ώστε να επιλέγει αυτόματα από εκεί σε ποιον αναφέρεται το συγκεκριμένο τιμολόγιο. Επίσης για κάθε νέα εισαγωγή προϊόντος στο τιμολόγιο, ο χρήστης θα μπορεί και πάλι να επιλέξει αυτόματα το προϊόν από μια λίστα στην οποία θα εμφανίζονται όλα τα προϊόντα που βρίσκονται στην βάση. Με αυτόν το τρόπο δεν θα χρειάζεται κάθε φορά να θυμάται τα ονόματα και τις τιμές πώλησης όλων των προϊόντων που διαθέτει η εταιρεία. Σε περίπτωση που το προϊόν αυτό δεν υπάρχει στο πίνακα αποθεμάτων, ο χρήστης θα έχει τη δυνατότητα να περάσει manual το προϊόν στο τιμολόγιο. Μετά από την έκδοση του τιμολογίου, θα του δίνεται μάλιστα και η δυνατότητα να αποθηκεύσει αυτόματα και πάλι, το προϊόν αυτό στη βάση του συστήματος μαζί με μερικά από τα βασικά χαρακτηριστικά του.

Κατά τη διάρκεια εισαγωγής προϊόντων στο τιμολόγιο, σε περίπτωση λάθους, ο χρήστης θα μπορεί να ανανεώσει ή και να διαγράψει κάποια από τα προϊόντα τιμολόγιου που ήδη έχουν περαστεί.
Σε περίπτωση εισαγωγής τιμολογίου αγοράς ο χρήστης θα περάσει manual τον αριθμό τιμολόγιου, ο οποίος θα αναγράφεται στο τιμολόγιο που έλαβε η εταιρεία από τον προμηθευτή, ενώ σε περίπτωση τιμολογίου πώλησης το σύστημα θα δίνει αυτόματα ένα μοναδικό αριθμό τιμολογίου κάθε φορά.

Επίσης από το DataGridView(πίνακα) στο οποίο εμφανίζονται όλα τα τιμολόγια, ο χρήστης θα μπορεί να επιλέξει ένα από αυτά και να κάνει μια ή περισσότερες από τις πιο κάτω ενέργειες :

1) Προβολή λεπτομερειών τιμολογίου
2) Ενημέρωση τιμολογίου
3) Εκτύπωση τιμολογίου
4) Ακύρωση τιμολογίου

Όσα αναφέρθηκαν πιο πάνω για τα τιμολόγια, ισχύουν ταυτόχρονα και για τις προσφορές, με τη διαφορά ότι δεν γίνεται ενημέρωση αποθεμάτων γιατί μια προσφορά δεν εκφράζει ούτε αγορά ούτε πώληση της εταιρείας.
Τέλος θα παρέχεται στο χρήστη η δυνατότητα αναζήτησης ενός εγγράφου (τιμολογίου, προσφοράς) με βάση τον αριθμό εγγράφου.

Διαχείριση Πληρωμών:

Στις πληρωμές, κατ'αναλογία με τους δυο τύπους τιμολογίου θα υπάρχουν δύο τύποι πληρωμών.
Πληρωμές οι οποίες αναφέρονται σε τιμολόγια αγορών της εταιρείας (έξοδα) και σε πληρωμές οι οποίες θα αναφέρονται σε τιμολόγια πωλήσεων της εταιρείας (έσοδα).

Ο χρήστης θα έχει τη δυνατότητα εισαγωγής των βασικών στοιχείων μιας νέας πληρωμής/απόδειξης στο σύστημα και συγκεκριμένα στη βάση του συστήματος.

Κατά την εισαγωγή της νέας πληρωμής, ο χρήστης θα μπορεί να μεταφερθεί στον πίνακα των τιμολογίων και να επιλέξει σε ποιο τιμολόγιο αναφέρεται η συγκεκριμένη πληρωμή. Αφού επιλέξει το τιμολόγιο, μερικά από τα βασικά στοιχεία της πληρωμής, όπως αριθμός τιμολογίου, συνολικό ποσό κ.α θα ενημερώνονται αυτόματα. Επιπρόσθετα, για κάθε πληρωμή θα δίνεται ένας μοναδικός κωδικός από το σύστημα, ο αριθμός απόδειξης.

Τέλος, από το DataGridView (πίνακα) στο οποίο εμφανίζονται όλες οι πληρωμές με τα βασικά τους στοιχεία, ο χρήστης θα μπορεί να επιλέξει μια από αυτές και να κάνει μια ή περισσότερες από τις πιο κάτω ενέργειες:

1) Ενημέρωση πληρωμής
2) Εκτύπωση πληρωμής
3) Ακύρωση πληρωμής

Εργαλεία Εφαρμογής

Στα εργαλεία εφαρμογής ανάλογα με το είδος του χρήστη θα δίνονται τα ανάλογα δικαιώματα διαχείρισης :

Διαχείριση Χρηστών:

Καταρχήν, οι χρήστες θα είναι άμεσα συνδεδεμένοι με όλους τους πίνακες της βάσης για λόγους που θα αναφερθούν σε επόμενο κεφάλαιο.
Στο σύστημα θα μπορούν να υπάρχουν περισσότεροι του ενός ενεργοί χρήστες και ανάλογα με τα δικαιώματα που θα τους δίνονται στη διαχείριση του συστήματος θα χωρίζονται σε δύο τύπους:

1) Οι χρήστες με πλήρη δικαιώματα στη διαχείριση, οι οποίοι θα είναι οι Administrator του συστήματος και
2) Οι χρήστες με ελλιπή δικαιώματα στη διαχείριση, οι οποίοι θα είναι οι απλοί Users.

Η βασική διαφορά των δύο τύπων χρηστών, είναι οι δυνατότητες που θα τους δίνονται στη διαχείριση μερικών από τα εργαλεία της εφαρμογής.

Ο κάθε χρήστης θα χαρακτηρίζεται κυρίως από το «Username», τον «κωδικό» του, αν είναι «ενεργός» ή όχι και από τον τύπο του (απλός χρήστης ή Administrator).

Δικαιώματα δημιουργίας νέου χρήστη, ενημέρωσης και διαγραφής χρηστών θα έχουν μόνο οι Administrator του συστήματος.

Διαχείριση Στοιχείων Εταιρείας :

Στη διαχείριση στοιχείων εταιρείας, οι χρήστες θα μπορούν να αλλάξουν άμεσα μερικές από τις γενικές πληροφορίες που αφορούν την εταιρεία και να τις αποθηκεύσουν. Μερικές από αυτές είναι το όνομα της εταιρείας, η διεύθυνση και το τηλέφωνο της εταιρείας, το λογότυπο εταιρείας κ.α.

Σκοπός του εργαλείου αυτού είναι να αποφευχθεί στο μέλλον η πιο κάτω διαδικασία :

Αν οι πληροφορίες για την εταιρεία είναι προκαθορισμένες με κώδικα σε μια κλάση της εφαρμογής, τότε κάθε φορά που θα άλλαζαν κάποια από τα στοιχεία της, ο προγραμματιστής θα έπρεπε να πηγαίνει στην εταιρεία για να διαμορφώνει κατάλληλα τον κώδικα, κάτι το οποίο δεν θα χρειάζεται να γίνεται τώρα, αφού ο χρήστης θα μπορεί να το κάνει εύκολα από μόνος του χάρη στο εργαλείο αυτό.

Διαχείριση Κατηγοριών Αποθεμάτων:

Ο χρήστης θα μπορεί να ορίσει μια νέα κατηγορία αποθέματος ή και να διαχειριστεί τις υπάρχουσες κατηγορίες. Σκοπός του εργαλείου αυτού, είναι η ευκολότερη διαχείριση των αποθεμάτων από μέρους των χρηστών.

Διαχείριση Τραπεζών:

Ο χρήστης θα μπορεί να εισάγει και να επεξεργάζεται, τα στοιχεία των τραπεζών που τυχόν εμπλέκονται στις συναλλαγές της εταιρείας.

Τέλος αξίζει να σημειωθεί, πως για κάθε νέα εισαγωγή/καταχώρηση των πιο πάνω, το σύστημα *αυτόματα* καταχωρεί στη βάση ένα μοναδικό ID για κάθε εγγραφή, την τρέχουσα ημερομηνία και ώρα της κάθε καταχώρησης, καθώς επίσης και τον κωδικό του χρήστη που έκανε την καταχώρηση.

Επιπλέον στο σύστημα, θα μπορούν να είναι συνδεδεμένοι από διαφορετικούς υπολογιστές περισσότεροι του ενός χρήστες ταυτόχρονα, αν αυτό χρειαστεί και

όλες οι αλλαγές που αυτοί θα πραγματοποιούν θα αποθηκεύονται σε μια κοινή βάση. Περαιτέρω εξήγηση των πιο πάνω θα γίνει και στα επόμενα κεφάλαια.

Αναφορές από το σύστημα

Το σύστημα θα παρέχει στο χρήστη διάφορες αναφορές για σημαντικά θέματα της εταιρείας όπως :

- Συνολικά έσοδα-έξοδα για συγκεκριμένες ημερομηνίες.
- Εμφάνιση των πελατών που έχουν οφειλές προς την εταιρεία, το ποσό οφειλής και τα βασικά τους στοιχεία.
- Εμφάνιση οφειλών της εταιρείας προς τους προμηθευτές, το ποσό οφειλής και τα βασικά στοιχεία των προμηθευτών αυτών.
- Συνολικά έσοδα από συγκεκριμένο πελάτη και για συγκεκριμένες ημερομηνίες.
- Συνολικά έξοδα για συγκεκριμένο προμηθευτή και για συγκεκριμένες ημερομηνίες.

Επίλογος:

Η πλήρης κάλυψη των στόχων λογισμικού, όπως αυτοί τίθενται από τον πελάτη και αποφασίζονται από κοινού με την ομάδα ανάπτυξης είναι, αν όχι το κυριότερο, ένα από τα σημαντικότερα σημεία της διαδικασίας ανάπτυξης.
Στη συνέχεια, αφού γίνει η μελέτη και αξιολόγηση των πιο πάνω, η ομάδα ανάπτυξης είναι σε θέση να επιλέξει το κατάλληλο «Μοντέλο Ανάπτυξης Λογισμικού».

.

ΚΕΦΑΛΑΙΟ 3: Περιγραφή του Μοντέλου Ανάπτυξης Λογισμικού

Εισαγωγή:

Πολύ σημαντικό ρόλο κατά την ανάπτυξη ενός λογισμικού συστήματος, έχει η «Μεθοδολογία Ανάπτυξης Λογισμικού» που θα χρησιμοποιηθεί. Ανάλογα με τις απαιτήσεις και αξιολογώντας τις καταστάσεις κάθε φορά, επιλέγεται το κατάλληλο Μοντέλο Ανάπτυξης.
Στο κεφάλαιο αυτό περιγράφονται οι λόγοι επιλογής του συγκεκριμένου Μοντέλου.

«Μοντέλο Κύκλου Ζωής» που χρησιμοποιήθηκε:

Για την ανάπτυξη της παρούσας εφαρμογής, επιλέχθηκε η Μεθοδολογία **«Ανάπτυξη με Προτυποποίηση»** (Development by Prototyping). Η μεθοδολογία αυτή ανήκει στην κατηγορία των Εξελικτικών Μεθοδολογιών/Ταχείας ανάπτυξης Εφαρμογής(Rapid Application Development - RAD).
Πρωτότυπο είναι ένα μερικώς ανεπτυγμένο προϊόν, το οποίο μπορεί να χρησιμοποιηθεί ως μέσο επικοινωνίας μεταξύ πελάτη και δημιουργού.
Στόχος της μεθοδολογίας αυτής είναι η συνεργασία με τον πελάτη και η άμεση δημιουργία μιας πρώτης έκδοσης της εφαρμογής, ξεκινώντας πάντα από μια αρχική περιγραφή και πάντοτε με πολύ καλά κατανοητές απαιτήσεις.

Μερικά από τα χαρακτηριστικά του:

Με το μοντέλο αυτό γίνονται αναθεωρήσεις, διορθώσεις και εμπλουτισμός σε κάθε φάση μέχρι την συμφωνία με τον πελάτη. **Στόχος** της διαδικασίας αυτής είναι η μείωση των κινδύνων και της αβεβαιότητας.
Γίνονται συνεχώς **επικυρώσεις** για την ορθή υλοποίηση των προδιαγραφών και **επαληθεύσεις** όσον αφορά την ορθότητα των λειτουργιών.
Το σύστημα δηλαδή αναπτύσσεται προσθέτοντας σταδιακά νέα χαρακτηριστικά, με αποτέλεσμα το αρχικό πρωτότυπο να εξελιχθεί στο τελικό σύστημα που θα εγκατασταθεί.

Πλεονεκτήματα του Μοντέλου:

- Παρέχει τη δυνατότητα στον πελάτη να αλλάξει γνώμη πριν υπογράψει
- Αντιμετωπίζει την ασάφεια στις απαιτήσεις λόγω της συχνής επικοινωνίας πελάτη – δημιουργού
- Μείωση του χρόνου ανάπτυξης
- Τα αρχικά πρωτότυπα χρησιμοποιούνται για εξοικείωση από τους πελάτες - χρήστες
- Μεγαλύτερη πιθανότητα ανάπτυξης φιλικού προς το χρήστη λογισμικού
- Ο πελάτης εμπλέκεται στην ανάπτυξη του προϊόντος
- Αυξανόμενη σταδιακά ικανοποίηση του πελάτη
- Επικοινωνία χρηστών / ομάδας ανάπτυξης

Επίλογος:

Ανακεφαλαιώνοντας, αφού γίνει η εκτενής ανάλυση των απαιτήσεων και η καταγραφή των στόχων λογισμικού, η ομάδα ανάπτυξης είναι σε θέση να επιλέξει το κατάλληλο Μοντέλο Ανάπτυξης.
Ακολούθως όλα είναι έτοιμα για να ξεκινήσει η διαδικασία ανάπτυξης.

ΚΕΦΑΛΑΙΟ 4: Περιγραφή της Βάσης Δεδομένων

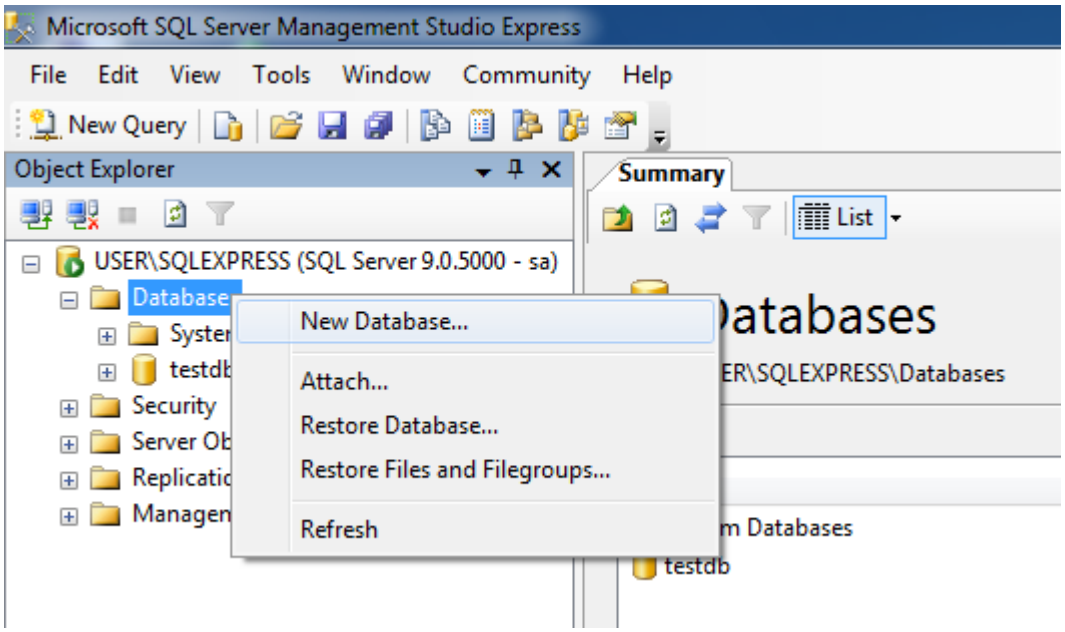
Εισαγωγή:

Στο κεφάλαιο αυτό, θα γίνει αναλυτική περιγραφή της δημιουργίας της Βάσης Δεδομένων. Θα γίνει παρουσίαση όλων των πινάκων της βάσης, των συσχετίσεων μεταξύ των πινάκων, καθώς επίσης και των πεδίων του κάθε πίνακα με τους τύπους δεδομένων που το καθένα υποστηρίζει.

Όπως έχει ήδη αναφερθεί, η βάση υλοποιήθηκε με τον **SQL Server 2005**, γιατί είναι ένα σύστημα το οποίο παρέχει πολλές χρήσιμες δυνατότητες και λειτουργίες. Μερικές από αυτές θα αναφερθούν στη συνέχεια του κεφαλαίου.

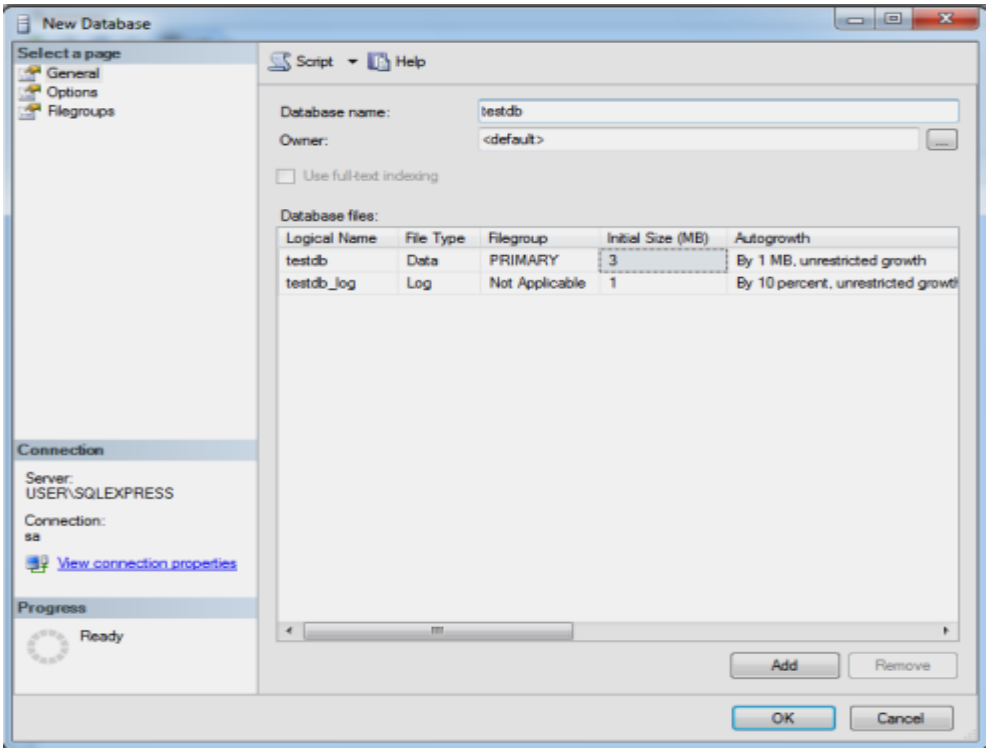
Παράδειγμα δημιουργίας βάσης στον SQL Server:

Για τη δημιουργία μιας καινούργιας βάσης, πατάμε δεξί κλικ στο φάκελο **«Database»** και από το παράθυρο που εμφανίζεται επιλέγουμε **«New Database…».**



Εικόνα 1: «Δημιουργία Βάσης 1/2»

Στη συνέχεια εμφανίζεται ένα νέο παράθυρο όπως φαίνεται πιο κάτω:

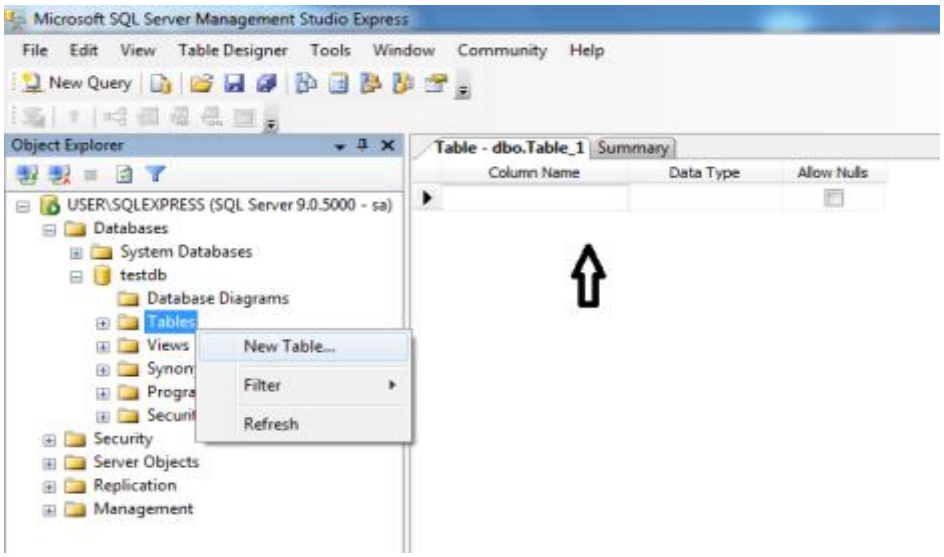


Εικόνα 2: «Δημιουργία Βάσης 2/2»

Στο πεδίο **«Database name»** γράφουμε ένα όνομα για τη βάση μας και ακολούθως πατάμε το κουμπί **«OK».** Μ' αυτό τον τρόπο πραγματοποιείται η δημιουργία της βάσης.

Το επόμενο βήμα είναι η δημιουργία των πινάκων της βάσης.

Παράδειγμα δημιουργίας πίνακα:



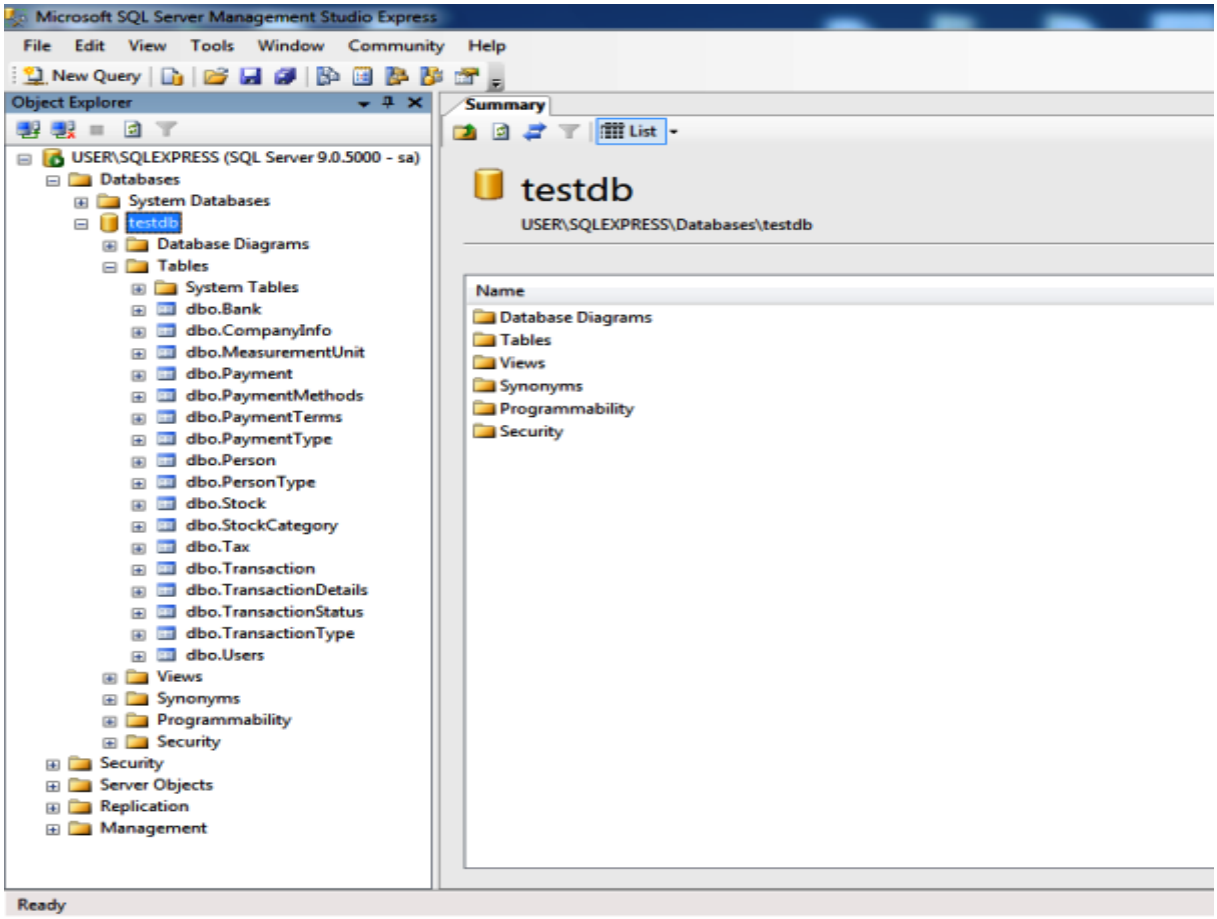
Εικόνα 3: «Δημιουργία πίνακα»

Στο φάκελο **«Tables»** που βρίσκεται κάτω από τη βάση μας, πατάμε δεξί κλικ και επιλέγουμε **«New Table…».** Τότε εμφανίζεται η εικόνα δεξιά που δείχνει το βέλος.

Στο textbox κάτω από τη στήλη **«Column Name»** μπορούμε να δώσουμε τα ονόματα των πεδίων του πίνακα και δεξιά από κάθε πεδίο, στη στήλη **«Data Type»**, τον τύπο δεδομένων που θα αποθηκεύει το κάθε πεδίο. Στη στήλη **«Allow Nulls»** δίνεται η δυνατότητα επιλογής ή όχι για το αν κάποιο πεδίο θα δέχεται και κενές - Null - τιμές. Αφού συμπληρωθούν τα επιθυμητά πεδία, επιλέγουμε ποιο από αυτά είναι το κύριο κλειδί του πίνακα, πατάμε **«Save»** και δίνουμε ένα όνομα για τον πίνακα που μόλις δημιουργήσαμε.

Η βάση της παρούσας εφαρμογής, αποτελείται από *δεκαεπτά πίνακες*, από τους οποίους οι εννιά είναι «βοηθητικοί» έχουν δηλαδή δημιουργηθεί για να περιγράφουν δεδομένα/τύπους των *οκτώ βασικών πινάκων* της βάσης.

Οι πίνακες της βάσης φαίνονται στην πιο κάτω εικόνα:



Εικόνα 4: «Οι πίνακες της Βάσης»

Μετά τη δημιουργία των πινάκων της βάσης, ακολουθεί η διαδικασία συσχέτισης των μεταξύ τους σχέσεων. Ουσιαστικά, στο σημείο αυτό ορίζονται τα ξένα κλειδιά του κάθε πίνακα, δηλαδή πεδία, τα οποία αντιστοιχούν σε κύρια κλειδιά/πεδία άλλων πινάκων της βάσης, καθορίζοντας έτσι τη μεταξύ τους συσχέτιση.

Σημειωτέον, ότι η διαδικασία αυτή είναι απαραίτητο να γίνεται, γιατί μια βάση χωρίς συσχέτιση μεταξύ των πινάκων οι οποίοι τη δομούν, είναι μια βάση χωρίς λειτουργικότητα.

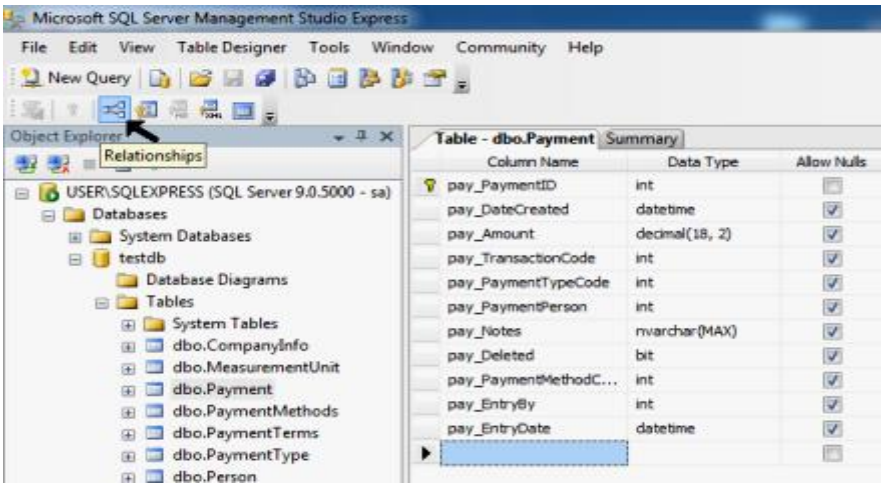
Παράδειγμα δημιουργίας συσχέτισης:

Ας υποθέσουμε ότι θέλουμε από την **«Εικόνα 4»** να συσχετίσουμε τους πίνακες **Payment** και ***PaymentMethods***. Ο πίνακας Payment αναφέρεται στις πληρωμές της εταιρείας και ο πίνακας PaymentMethods στις μεθόδους πληρωμής (π.χ Επιταγή, Μετρητά, Πιστωτική κάρτα, ...).

Εύκολα, λοιπόν, μπορεί κανείς να διακρίνει ότι το κύριο κλειδί του πίνακα ***PaymentMethods*** είναι ξένο κλειδί προς τον πίνακα **Payment .** Συνεπώς, για να υπάρχει μια λειτουργικότητα μεταξύ τους πρέπει να γίνει και ο απαραίτητος συσχετισμός τους.

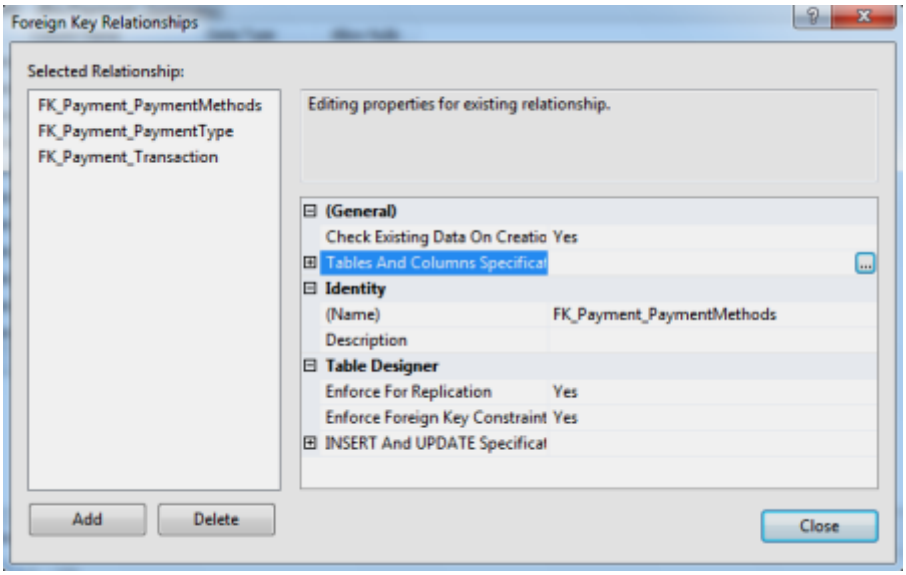
Ως εκ τούτου, για να οριστεί η συσχέτιση αυτή ακολουθείται η εξής διαδικασία :

Στον πίνακα Payments, όπως αυτός φαίνεται στην «**Εικόνα 4»**, πατάμε «δεξί κλικ», επιλέγουμε **«Modify»** και εμφανίζεται η ακόλουθη εικόνα :



Εικόνα 5: «Δημιουργία Συσχέτισης 1/3»

Ακολούθως κάνουμε κλικ στην επιλογή **«Relationships»** που δείχνει το βέλος και εμφανίζεται το πιο κάτω παράθυρο. Πατάμε **«Add»** για τη δημιουργία νέας συσχέτισης και επιλέγουμε το **«Tables And Columns Specification».**

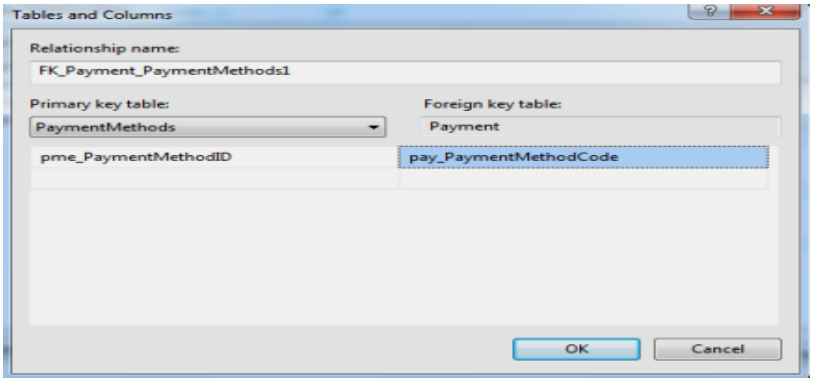


Εικόνα 6: «Δημιουργία Συσχέτισης 2/3»

Στο παράθυρο που εμφανίζεται (Εικόνα 7), δίνουμε ένα όνομα για τη νέα συσχέτιση και στη συνέχεια από το property **«Primary key table»** επιλέγουμε τον πίνακα από τον οποίο θα πάρουμε το κύριο κλειδί. Στο παράδειγμά μας είναι ο Πίνακας ***PaymentMethods*** και το κύριο κλειδί του είναι το πεδίο **pme_PaymentMethodID.**

Στη συνέχεια στο **«Foreign key table»,** όπου είναι προεπιλεγμένος ο Πίνακας **Payment**, επιλέγουμε το πεδίο του πίνακα, στο οποίο αντιστοιχεί το ξένο κλειδί από τον πίνακα PaymentMethods και είναι το **pay_PaymentMethodCode**.

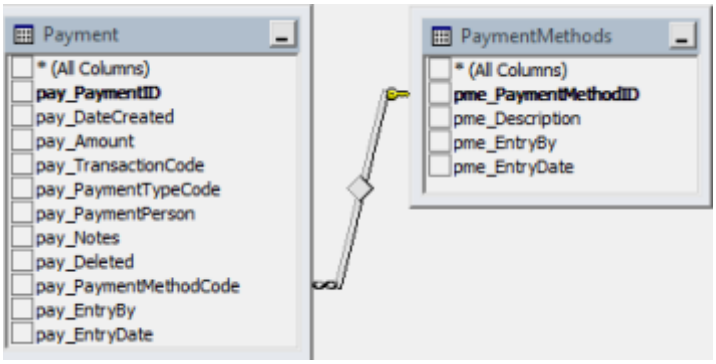
Τέλος πατάμε το κουμπί **«OK»** και η συσχέτιση των δύο πινάκων ολοκληρώνεται.



Εικόνα 7: «Δημιουργία Συσχέτισης 3/3»

Η σωστή δημιουργία της συσχέτισης, μπορεί να επαληθευτεί με τον εξής τρόπο:

Στο φάκελο **«Views»** (Εικόνα 4) πατάμε «δεξί κλικ», και επιλέγουμε **«New View…»**, στο παράθυρο που εμφανίζεται επιλέγουμε για ποιους πίνακες επιθυμούμε να δούμε τη συσχέτιση (Payment, PaymentMethods), πατάμε **«Add»** και ακολούθως **«Close»**. Το αποτέλεσμα της πιο πάνω διαδικασίας φαίνεται στην εικόνα που ακολουθεί.



Εικόνα 8: «Database View»

Εφόσον, λοιπόν, έχει παρουσιαστεί βήμα προς βήμα ο βασικός τρόπος με τον οποίο δομείται μια βάση και πως ορίζονται οι αλληλοσυσχετίσεις μεταξύ των πινάκων, απομένει η ανάλυση του κάθε πίνακα της βάσης, η οποία θα παρουσιαστεί εκτενώς στη συνέχεια.

Ανάλυση των πινάκων της βάσης

Σ΄ αυτό το υποκεφάλαιο, θα γίνει ανάλυση όλων των πινάκων της βάσης, των πεδίων και των αλληλοσυσχετίσεων του κάθε πίνακα, καθώς θα γίνει αναφορά στη λειτουργία και τη χρησιμότητα του καθενός εξ΄αυτών.

Στο σημείο αυτό, καλό είναι να επισημανθεί ότι έγινε προσπάθεια τήρησης σωστής ονοματολογίας των πεδίων των πινάκων, ώστε κατά τον προγραμματισμό και την αλληλεπίδραση του προγράμματος με τη βάση, να είναι ξεκάθαρο το που ανήκει το κάθε πεδίο. Έτσι τα ονόματα των πεδίων του κάθε πίνακα αρχίζουν με τρία γράμματα, τα οποία συμβολίζουν τον πίνακα στον οποίο το καθένα ανήκει.
Επίσης τα ξένα κλειδιά του κάθε πίνακα έχουν για ονομασία, το όνομα του πίνακα στον οποίο αντιστοιχούν, του πίνακα δηλαδή στον οποίο είναι κύρια κλειδιά.

Οι *δεκαεπτά πίνακες* της βάσης *(8 βασικοί και 9 βοηθητικοί)* είναι οι κάτωθι:

1. Person PersonType
2. Payment PaymentType PaymentMethods
3. Stock StockCategory MeasurementUnit Tax
4. Bank
5. Company Info
6. Transaction TransactionType TransactionStatus PaymentTerms
7. TransactionDetails
8. Users

Πίνακας Person:

Στον πίνακα αυτό θα αποθηκεύονται όλα τα πρόσωπα που συναλλάσσονται με την εταιρεία. Τα πρόσωπα αυτά είναι οι **Πελάτες** και οι **Προμηθευτές**. Πιο συγκεκριμένα, προμηθευτές είναι άλλες εταιρείες οι οποίες προμηθεύουν με προϊόντα την εν λόγω εταιρεία.

Ως εκ τούτου, στον πίνακα αυτό θα υπάρχουν δύο τύποι Person (ατόμων) όπως αυτοί προαναφέρθηκαν. Κύριο κλειδί στον πίνακα αυτό είναι το ID του Person και πιο συγκεκριμένα το πεδίο **per_PersonID,** ενώ ξένο κλειδί είναι το πεδίο **per_PersonTypeCode** που αντιστοιχεί στο κύριο κλειδί του πίνακα **PersonType** και συγκεκριμένα στο πεδίο **pty_PersonTypeID.** Το πεδίο **per_PersonTypeCode** προσδιορίζει αν το άτομο είναι Πελάτης ή Προμηθευτής.
Τα πεδία και οι τύποι των πεδίων του πίνακα Person φαίνονται στην επόμενη εικόνα :

Table - dbo.Person

Column Name	Data Type	Allow Nulls
per_PersonID	int	☐
per_Name	nvarchar(100)	☑
per_Address	nvarchar(150)	☑
per_Telephone	varchar(50)	☑
per_Fax	varchar(50)	☑
per_Email	varchar(50)	☑
per_Website	varchar(50)	☑
per_PersonTypeCode	int	☑
per_EntryBy	int	☑
per_EntryDate	datetime	☑
per_Notes	nvarchar(MAX)	☑
per_Deleted	bit	☑

Εικόνα 9: «Πίνακας Person»

Βοηθητικός Πίνακας PersonType:

Ο Πίνακας αυτός δημιουργήθηκε για να προσδιορίζει τον τύπο του ατόμου (Πελάτης/Προμηθευτής) στον πίνακα **Person.**
Τα πεδία και οι τύποι των πεδίων του πίνακα **PersonType** φαίνονται στην επόμενη εικόνα :

Table - dbo.PersonType

Column Name	Data Type	Allow Nulls
pty_PersonTypeID	int	☐
pty_Description	varchar(50)	☑
pty_EntryBy	int	☑
pty_EntryDate	datetime	☑
		☐

Εικόνα 10: «Πίνακας PersonType»

Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **pty_PersonTypeID**. Για τις διάφορες τιμές του **pty_PersonTypeID** (τιμές από 1-2), το πεδίο **pty_Description** παίρνει αντίστοιχα τις εξής τιμές:

- CUSTOMER / Πελάτης
- SUPPLIER / Προμηθευτής

Πίνακας Payment:

Στον πίνακα **Payment** αποθηκεύονται όλες οι πληρωμές της εταιρείας. Οι πληρωμές μπορεί να είναι είτε έσοδα της εταιρείας (από πελάτες), είτε έξοδα της εταιρείας (προς προμηθευτές).
Ως εκ τούτου, είναι φανερό ότι στον πίνακα Payment υπάρχουν δύο τύποι πληρωμών όπως αυτοί αναφέρθηκαν προηγουμένως. Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **pay_PaymentID** και ξένα κλειδιά είναι τα εξής:

- **pay_TransactionCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Transaction,** που είναι το πεδίο **tra_TransactionID** και προσδιορίζει για πιο τιμολόγιο συγκεκριμένα έγινε η πληρωμή.

- **pay_PaymentTypeCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **PaymentType**, που είναι το πεδίο **pty_PaymentTypeID** και προσδιορίζει τον τύπο πληρωμής.
- **pay_PaymentMethodsCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **PaymentMethods,** που είναι το πεδίο **pme_PaymentMethodsID** και προσδιορίζει την μέθοδο πληρωμής (Τοις Μετρητοίς, Επιταγή, Πιστωτική Κάρτα, Τραπεζική Μεταφορά, Άλλος).
- **pay_BankCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Bank**, που είναι το πεδίο **bnk_BankID** και προσδιορίζει την τράπεζα που αναμιγνύεται, σε περίπτωση που η πληρωμή γίνει με «Επιταγή». Τα πεδία και οι τύποι των πεδίων του πίνακα **Payment** φαίνονται στην επόμενη εικόνα :

Table - dbo.Payment

Column Name	Data Type	Allow Nulls
pay_PaymentID	int	☐
pay_DateCreated	datetime	☑
pay_Amount	decimal(18, 2)	☑
pay_TransactionCode	int	☑
pay_PaymentTypeCode	int	☑
pay_EntryBy	int	☑
pay_EntryDate	datetime	☑
pay_PaymentMethodCode	int	☑
pay_Deleted	bit	☑
pay_Notes	nvarchar(MAX)	☑
pay_DocumentNumber	nvarchar(15)	☑
pay_ChequeNo	nvarchar(50)	☑
pay_BankCode	int	☑
		☐

Εικόνα 11: «Πίνακας Payment»

Βοηθητικός Πίνακας PaymentType:

Σκοπός του πίνακα αυτού είναι να προσδιορίζει τον τύπο πληρωμής (έσοδα/έξοδα) στον πίνακα Payment.

Τα πεδία και οι τύποι των πεδίων του πίνακα **PaymentType** φαίνονται στην επόμενη εικόνα :

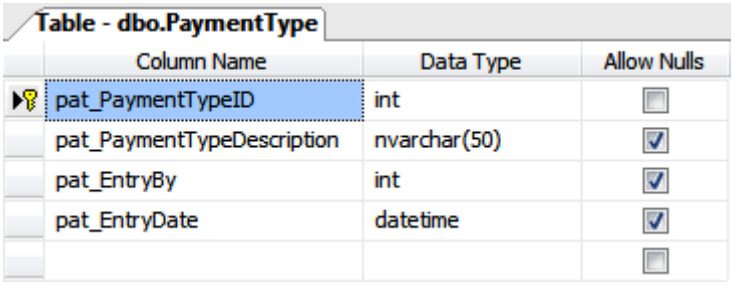


Table - dbo.PaymentType

Column Name	Data Type	Allow Nulls
pat_PaymentTypeID	int	☐
pat_PaymentTypeDescription	nvarchar(50)	☑
pat_EntryBy	int	☑
pat_EntryDate	datetime	☑
		☐

Εικόνα 12: «Πίνακας PaymentType»

Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **pat_PaymentTypeID.** Για τις διάφορες τιμές του **pat_PaymentTypeID** (τιμές από 1-2) το πεδίο **pat_PaymentTypeDescription** έχει αντίστοιχα τις εξής τιμές:

- Customers – δηλαδή έσοδα (πληρωμές από πελάτες).
- Supplier – δηλαδή έξοδα (πληρωμές προς τους προμηθευτές).

Βοηθητικός Πίνακας PaymentMethods:

Ο Πίνακας αυτός δημιουργήθηκε για να προσδιορίζει την μέθοδο πληρωμής στον πίνακα Payment. Τα πεδία και οι τύποι των πεδίων του πίνακα **PaymentType** φαίνονται στην επόμενη εικόνα :

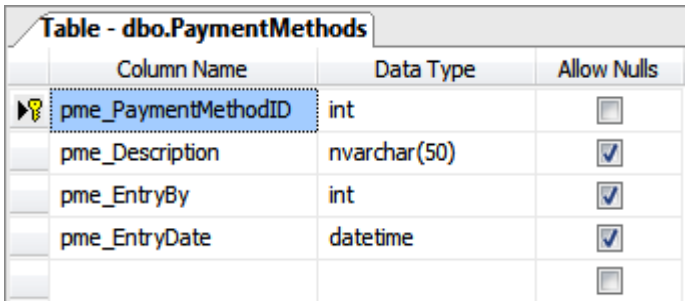


Table - dbo.PaymentMethods

Column Name	Data Type	Allow Nulls
pme_PaymentMethodID	int	☐
pme_Description	nvarchar(50)	☑
pme_EntryBy	int	☑
pme_EntryDate	datetime	☑
		☐

Εικόνα 13: «Πίνακας PaymentMethods»

Κύριο κλειδί του πίνακα **PaymentMethods** είναι το πεδίο **pme_PaymentMethodsID.** Για τις διάφορες τιμές του **pme_PaymentMethodsID** (τιμές από 1-5), το **πεδίο pme_Description** παίρνει αντίστοιχα τις τιμές

- Τοις Μετρητοίς
- Επιταγή
- Πιστωτική κάρτα
- Τραπεζική μεταφορά

- Άλλος

Πίνακας Stock:

Στον πίνακα αυτό θα αποθηκεύονται όλα τα αποθέματα της εταιρείας, η ημερομηνία παραλαβής, η τιμή αγοράς και πώλησης και άλλα σημαντικά στοιχεία, όπως αυτά φαίνονται πιο κάτω με τους αντίστοιχους τύπους τους.
Κύριο κλειδί στον πίνακα αυτό είναι το πεδίο **sto_StockID** και ξένα κλειδιά είναι τα εξής:

- **sto_MeasurementUnitCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **MeasurementUnit,** που είναι το πεδίο **mun_MeasurementUnitID** και προσδιορίζει τη μονάδα μέτρησης του κάθε προϊόντος (Μέτρα, Τετρ. Μέτρα, Ανά κομμάτι, Κιλά).
- **sto_SupplierCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Person**, που είναι το πεδίο **per_PersonID** και συγκεκριμένα στο ID του προμηθευτή από τον οποίο αγόρασε το απόθεμα η εταιρεία.
- **sto_CategoryStock ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **StockCategory,** που είναι το πεδίο **cat_CategoryID** και συγκεκριμένα στο ID της κατηγορίας αποθεμάτων στην οποία αντιστοιχεί το συγκεκριμένο απόθεμα.
- **sto_TaxCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Tax**, που είναι το πεδίο **tax_ID** και συγκεκριμένα στον φόρο προστιθέμενης αξίας του εκάστοτε προϊόντος.

Τα πεδία και οι τύποι των πεδίων του πίνακα **Stock** φαίνονται στην επόμενη εικόνα :

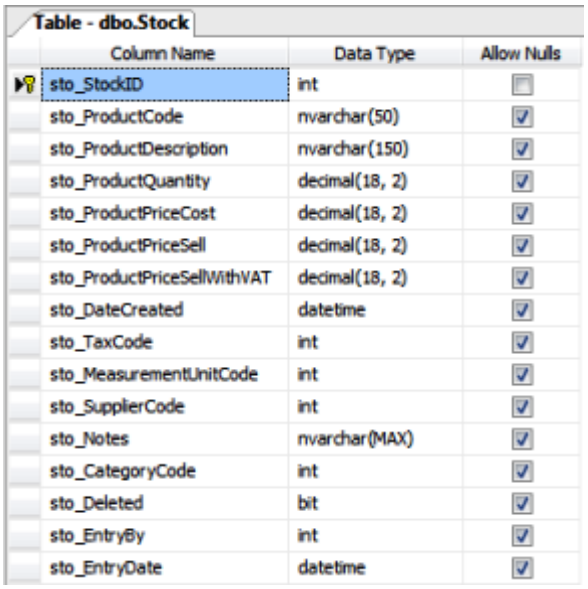


Table - dbo.Stock

Column Name	Data Type	Allow Nulls
sto_StockID	int	☐
sto_ProductCode	nvarchar(50)	☑
sto_ProductDescription	nvarchar(150)	☑
sto_ProductQuantity	decimal(18, 2)	☑
sto_ProductPriceCost	decimal(18, 2)	☑
sto_ProductPriceSell	decimal(18, 2)	☑
sto_ProductPriceSellWithVAT	decimal(18, 2)	☑
sto_DateCreated	datetime	☑
sto_TaxCode	int	☑
sto_MeasurementUnitCode	int	☑
sto_SupplierCode	int	☑
sto_Notes	nvarchar(MAX)	☑
sto_CategoryCode	int	☑
sto_Deleted	bit	☑
sto_EntryBy	int	☑
sto_EntryDate	datetime	☑

Εικόνα 14: «Πίνακας Stock»

Βοηθητικός Πίνακας StockCategory:

Σκοπός του πίνακα **StockCategory** είναι να προσδιορίζει την κατηγορία αποθέματος στην οποία ανήκει το κάθε προϊόν. Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **cat_CategoryID** και τα υπόλοιπα πεδία και τύποι των πεδίων του φαίνονται στην επόμενη εικόνα :

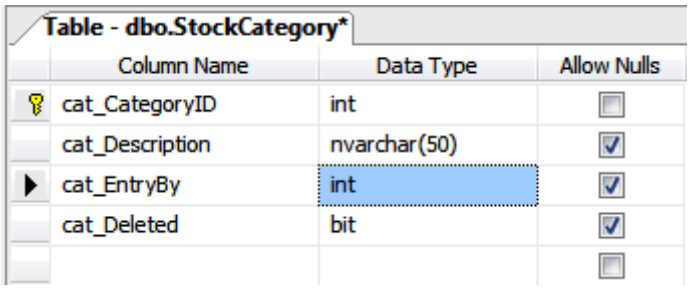


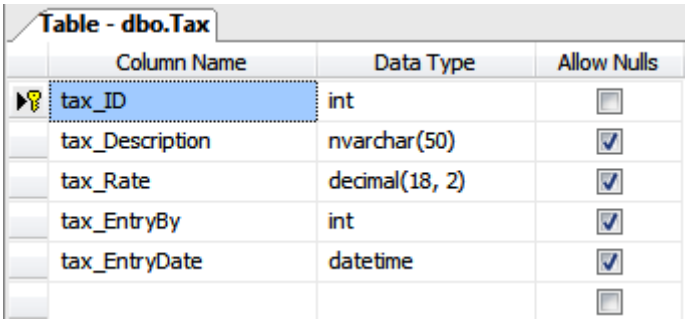
Table - dbo.StockCategory*

Column Name	Data Type	Allow Nulls
cat_CategoryID	int	☐
cat_Description	nvarchar(50)	☑
cat_EntryBy	int	☑
cat_Deleted	bit	☑
		☐

Εικόνα 15: «Πίνακας StockCategory»

Βοηθητικός Πίνακας Tax:

Ο Πίνακας αυτός αποσκοπεί στον προσδιορισμό της τιμής προστιθέμενης αξίας του κάθε προϊόντος. Τα πεδία και οι τύποι των πεδίων του πίνακα **Tax** φαίνονται στην επόμενη εικόνα :



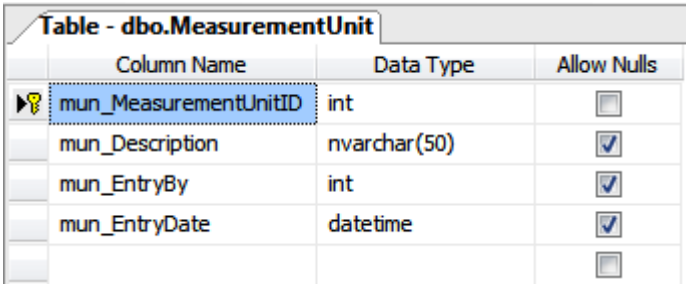
Εικόνα 16: «Πίνακας Tax»

Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **tax_ID.** Για τις διάφορες τιμές του **tax_ID** (τιμές από 1-2), το πεδίο **tax_Rate** έχει αντίστοιχα τις πιο κάτω τιμές :

- **tax_Rate = 0.00 ,** δηλαδή το προϊόν δεν έχει Φ.Π.Α.
- **tax_Rate = 0.15**

Βοηθητικός Πίνακας MeasurementUnit:

Ο σκοπός της δημιουργίας του πίνακα αυτού, είναι να προσδιορίζει την μονάδα μέτρησης του κάθε προϊόντος. Τα πεδία και οι τύποι των πεδίων του πίνακα **MeasurementUnit** φαίνονται στην επόμενη εικόνα :



Εικόνα 17: «Πίνακας MeasurementUnit»

Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **mun_MeasurementUnitID.** Για τις διάφορες τιμές του **mun_MeasurementUnitID** (τιμές από 1-4), το πεδίο **mun_Description** παίρνει αντίστοιχα τις τιμές

- Μέτρα
- Τετραγωνικά Μέτρα
- Ανά κομμάτι
- Κιλά (kg)

Πίνακας Bank:

Η βασική λειτουργία του πίνακα αυτού, είναι να κρατά πληροφορίες σχετικά με τις τράπεζες που τυχόν αναμιγνύονται στις διάφορες συναλλαγές της εταιρείας.
Τα πεδία και οι τύποι των πεδίων του πίνακα **Bank** φαίνονται στην επόμενη εικόνα:

Table - dbo.Bank

Column Name	Data Type	Allow Nulls
bnk_BankID	int	☐
bnk_Name	nvarchar(100)	☑
bnk_PhoneNo	varchar(20)	☑
bnk_FaxNo	varchar(20)	☑
bnk_Address	nvarchar(250)	☑
bnk_Email	varchar(50)	☑
bnk_Website	varchar(50)	☑
bnk_ContactPerson	nvarchar(60)	☑
bnk_Deleted	bit	☑
bnk_EntryDate	datetime	☑
bnk_EntryBy	int	☑
		☐

Εικόνα 18: «Πίνακας Bank»

Πίνακας CompanyInfo:

Στον πίνακα **CompanyInfo,** αποθηκεύονται όλα τα στοιχεία της εταιρείας, όπως Όνομα, Διεύθυνση, Τηλέφωνο, Email, Λογότυπο κ.α.
Τα πεδία και οι τύποι των πεδίων του πίνακα **CompanyInfo** φαίνονται στην επόμενη εικόνα:

Table - dbo.CompanyInfo

Column Name	Data Type	Allow Nulls
com_CompanyID	int	☐
com_Name	nvarchar(100)	☑
com_Phone	varchar(15)	☑
com_Fax	varchar(15)	☑
com_AddStreetName	nvarchar(50)	☑
com_AddNumber	varchar(5)	☑
com_AddCity	nvarchar(50)	☑
com_AddCountry	nvarchar(50)	☑
com_Email	nvarchar(50)	☑
com_Website	nvarchar(50)	☑
com_TaxNo	varchar(50)	☑
com_VatNo	varchar(50)	☑
com_LogoPath	varchar(100)	☑
		☐

Εικόνα 19: «Πίνακας CompanyInfo»

Ο Πίνακας αυτός, χρησιμεύει στην άντληση πληροφοριών για τα στοιχεία της εταιρείας μέσω αυτού και τη χρησιμοποίηση τους στα διάφορα έγγραφα της εταιρείας.

Πίνακας Transaction:

Στον πίνακα **Transaction,** αποθηκεύονται όλα τα τιμολόγια και οι προσφορές τα οποία προκύπτουν από τις συναλλαγές της εταιρείας.
Αναλυτικότερα:

- Τα τιμολόγια μπορεί να προέρχονται είτε από αγορές (από προμηθευτές), είτε από πωλήσεις της εταιρείας (προς πελάτες).
- Οι προσφορές απευθύνονται προς τους πελάτες της εταιρείας.

Εν ολίγοις, όπως προκύπτει από τα προαναφερθέντα, είναι φανερό ότι στον πίνακα **Transaction** υπάρχουν τρεις τύποι συναλλαγών (Αγορά, Πώληση Προσφορά). Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **tra_TransactionID** και ξένα κλειδιά είναι τα εξής:

- **tra_PersonCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Person**, που είναι το πεδίο **per_PersonID,** και συγκεκριμένα στο ID του πελάτη ή του προμηθευτή στον οποίο ανήκει το τιμολόγιο ή η προσφορά.
- **tra_TransactionTypeCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **TransactionType**, που είναι το πεδίο **trt_TransactionTypeID** και προσδιορίζει τον τύπο της συναλλαγής (Αγορά, Πώληση, Προσφορά).
- **tra_TransactionStatusCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **TransactionStatus**, που είναι το πεδίο **trs_TransactionStatusID** και προσδιορίζει την κατάσταση του τιμολογίου. (Ανοιχτό, Εξοφλημένο, Άκυρο).
- **tra_PaymentMethodCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **PaymentMetdods**, που είναι το πεδίο **pme_PaymentMethodID** και προσδιορίζει τη μέθοδο πληρωμής που συμφωνήθηκε για το συγκεκριμένο τιμολόγιο (Τοις Μετρητοίς, Επιταγή, Πιστωτική Κάρτα, Τραπεζική Μεταφορά, Άλλος).
- **tra_PaymentTermsCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **PaymentTerms,** που είναι το πεδίο **pte_PaymentTermsID** και προσδιορίζει τους όρους πληρωμής που συμφωνήθηκαν για το συγκεκριμένο τιμολόγιο (Εξοφλημένο, Πληρωμή κατά την παράδοση, Πληρωμή σε μέρες).

Τα πεδία και οι τύποι των πεδίων του πίνακα αυτού φαίνονται στην πιο κάτω εικόνα:

Table - dbo.Transaction

Column Name	Data Type	Allow Nulls
tra_TransactionID	int	☐
tra_DocumentNumber	nvarchar(50)	☑
tra_PersonCode	int	☑
tra_DateCreated	datetime	☑
tra_TransactionTypeCode	int	☑
tra_TransactionStatusCode	int	☑
tra_PaymentMethodCode	int	☑
tra_PaymentTermsCode	int	☑
tra_PaymentInDays	varchar(2)	☑
tra_SubTotal	decimal(18, 2)	☑
tra_Discount	decimal(18, 2)	☑
tra_TaxAmount	decimal(18, 2)	☑
tra_AmountPayable	decimal(18, 2)	☑
tra_AmountRemaining	decimal(18, 2)	☑
tra_ExternalNotes	nvarchar(MAX)	☑
tra_InternalNotes	nvarchar(MAX)	☑
tra_EntryBy	int	☑
tra_EntryDate	datetime	☑
tra_Deleted	bit	☑
tra_CancellationReason	nvarchar(MAX)	☑
		☐

Εικόνα 20: «Πίνακας Transaction»

Βοηθητικός Πίνακας TransactionType:

Η ενσωμάτωση του πίνακα αυτού, έγινε με στόχο, μέσω αυτού, να προσδιορίζεται το είδος της συναλλαγής στον πίνακα **Transaction**. Κύριο κλειδί στον πίνακα **TransactionType** είναι το πεδίο **trt_TransactionTypeID** και για τις διάφορες τιμές του (τιμές από 1-3), το πεδίο **trt_Description** παίρνει αντίστοιχα τις τιμές:

- Αγορά
- Πώληση
- Προσφορά

Τα πεδία και οι τύποι των πεδίων του πίνακα αυτού φαίνονται στην πιο κάτω εικόνα:

Table - dbo.TransactionType

Column Name	Data Type	Allow Nulls
trt_TransactionTypeID	int	☐
trt_Description	varchar(50)	☑
trt_EntryBy	int	☑
trt_EntryDate	datetime	☑
		☐

Εικόνα 21: «Πίνακας TransactionType»

Βοηθητικός Πίνακας TransactionStatus:

Ο Πίνακας αυτός δημιουργήθηκε με σκοπό τον προσδιορισμό της κατάστασης των τιμολογίων στον πίνακα **Transaction**. Έχει για κύριο κλειδί το πεδίο **trs_TransactionStatusID**, και για τις διάφορες τιμές του (τιμές από 1-3), το πεδίο **trs_Description** παίρνει αντίστοιχα τις τιμές:

- Ανοιχτό
- Εξοφλημένο
- Άκυρο

Τα πεδία και οι τύποι των πεδίων του πίνακα αυτού φαίνονται στην πιο κάτω εικόνα:

Table - dbo.TransactionStatus

Column Name	Data Type	Allow Nulls
trs_TransactionStatusID	int	☐
trs_Description	nvarchar(50)	☑
trs_EntryBy	int	☑
trs_EntryDate	datetime	☑
		☐

Εικόνα 22: «Πίνακας TransactionType»

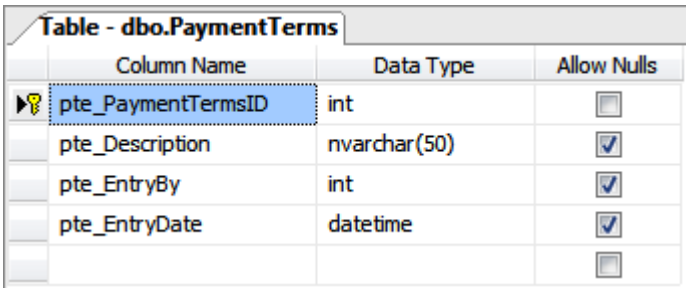
Βοηθητικός Πίνακας PaymentTerms:

Η δημιουργία του πίνακα **PaymentTerms**, έχει στόχο τον προσδιορισμό των όρων πληρωμής στον πίνακα **Transaction**. Κύριο κλειδί του πίνακα αυτού είναι το πεδίο **pte_PaymentTermsID** και για τις διάφορες τιμές του (τιμές από 1-3), το πεδίο **pte_Description** παίρνει αντίστοιχα τις τιμές:

- Εξοφλημένο
- Πληρωμή κατά την παράδοση

- Πληρωμή σε μέρες

Τα υπόλοιπα πεδία του πίνακα **PaymentTerms** καθώς και οι τύποι τους φαίνονται στην επόμενη εικόνα:



Εικόνα 23: «Πίνακας PaymentTerms»

Πίνακας TransactionDetails:

Ο Πίνακας αυτός δημιουργήθηκε με βασικό σκοπό την αποθήκευση των λεπτομερειών που αφορούν μια συναλλαγή (τιμολόγιο/προσφορά). Τέτοιες λεπτομέρειες είναι το όνομα προϊόντος, περιγραφή προϊόντος, ποσότητα προϊόντος, η τιμή προϊόντος κ.α.
Κύριο κλειδί του πίνακα αυτού, είναι το πεδίο **trd_TransactionDetailsID** και ξένα κλειδιά είναι τα εξής:

- **trd_TransactionCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Transaction**, που είναι το πεδίο **tra_TransactionID,** και προσδιορίζει σε ποια συναλλαγή ανήκουν οι λεπτομέρειες αυτές.

- **trd_ProductCode ->** το πεδίο αυτό αντιστοιχεί στο κύριο κλειδί του πίνακα **Stock,** που είναι το πεδίο **sto_StockID,** και προσδιορίζει σε ποιο προϊόν αναφέρεται συγκεκριμένα η κάθε εγγραφή του πίνακα αυτού.

Τα πεδία και οι τύποι των πεδίων του πίνακα αυτού φαίνονται στην πιο κάτω εικόνα:

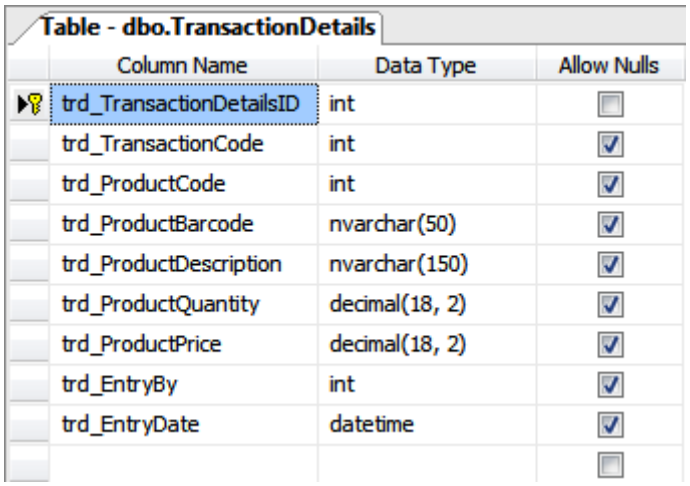


Table - dbo.TransactionDetails

Column Name	Data Type	Allow Nulls
trd_TransactionDetailsID	int	☐
trd_TransactionCode	int	☑
trd_ProductCode	int	☑
trd_ProductBarcode	nvarchar(50)	☑
trd_ProductDescription	nvarchar(150)	☑
trd_ProductQuantity	decimal(18, 2)	☑
trd_ProductPrice	decimal(18, 2)	☑
trd_EntryBy	int	☑
trd_EntryDate	datetime	☑
		☐

Εικόνα 24: «Πίνακας TransactionDetails»

Πίνακας Users:

Στον πίνακα **Users,** θα αποθηκεύονται όλα τα στοιχεία των χρηστών του συστήματος. Κύριο κλειδί του πίνακα **Users**, είναι το πεδίο **usr_UserID** και είναι ξένο κλειδί στους υπόλοιπους πίνακες της βάσης για λόγους που θα αναφερθούν στη συνέχεια του κεφαλαίου.

Τα υπόλοιπα πεδία του πίνακα αυτού καθώς και οι τύποι τους, φαίνονται στην επόμενη εικόνα:

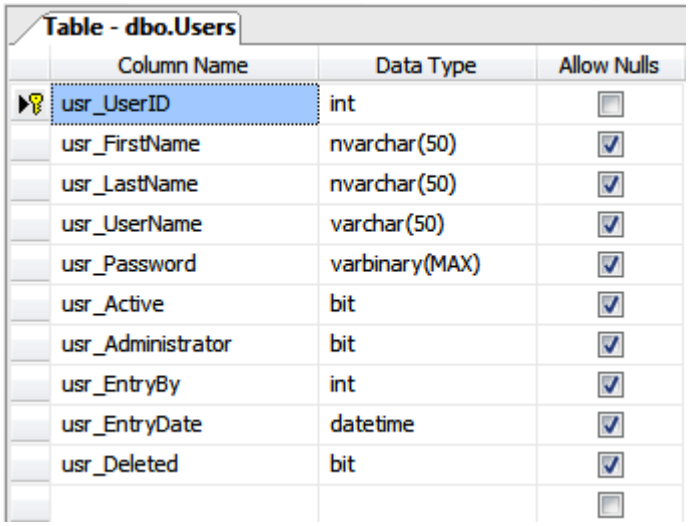


Table - dbo.Users

Column Name	Data Type	Allow Nulls
usr_UserID	int	☐
usr_FirstName	nvarchar(50)	☑
usr_LastName	nvarchar(50)	☑
usr_UserName	varchar(50)	☑
usr_Password	varbinary(MAX)	☑
usr_Active	bit	☑
usr_Administrator	bit	☑
usr_EntryBy	int	☑
usr_EntryDate	datetime	☑
usr_Deleted	bit	☑
		☐

Εικόνα 25: «Πίνακας Users»

Στο σημείο αυτό αξίζει να σημειωθούν τα πιο κάτω:

Όπως φαίνεται και από τις πιο πάνω εικόνες, στους πλείστους πίνακες υπάρχουν τα πεδία **EntryBy**, **EntryDate**, και **Deleted**. Ο σκοπός των πεδίων αυτών και η επεξήγηση της υλοποίησης τους φαίνεται πιο κάτω:

- **EntryBy:** Για κάθε αλλαγή που θα πραγματοποιείται σε κάποιο πίνακα, αυτόματα θα αποθηκεύεται από το σύστημα και το ID του χρήστη που έκανε την αλλαγή. Απώτερος σκοπός, είναι σε περίπτωση κάποιου λάθους ή σε περίπτωση κακόβουλης ενέργειας μέσα στη βάση του συστήματος, τα στελέχη της εταιρείας να μπορούν να δουν από ποιο χρήστη/υπάλληλο προκλήθηκε. Επίσης για να γίνει εφικτή η επίτευξη της προαναφερθείσας λειτουργίας, το κύριο κλειδί του πίνακα **Users** πρέπει να είναι ξένο κλειδί στους πίνακες αυτούς.

  Επεξήγηση Υλοποίησης

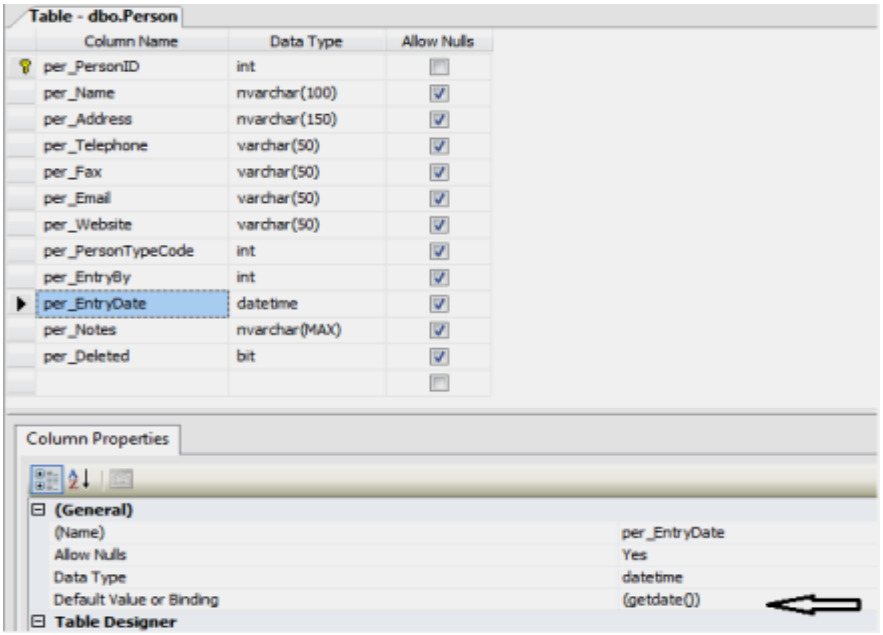
  Η πιο πάνω διαδικασία γίνεται προγραμματιστικά. Κατά την είσοδο κάποιου χρήστη στο σύστημα, αποθηκεύεται το ID του σε μια μεταβλητή, με αποτέλεσμα όταν αυτός πραγματοποιήσει μια αλλαγή μέσα στη βάση (Εισαγωγή, Ενημέρωση, Διαγραφή), τότε μέσω του Query που θα εκτελεστεί, θα αποθηκευτεί και το ID του.

- **EntryDate:** Για κάθε αλλαγή που πραγματοποιείται σε κάποιο πίνακα, αυτόματα αποθηκεύεται από το σύστημα και η τρέχουσα ημερομηνία και ώρα. Σκοπός είναι τόσο η ευκολία των χρηστών στην ανεύρεση των καταχωρημένων πληροφοριών με πάσα ακρίβεια και η εξοικονόμηση χρόνου, αφού δεν θα απαιτείται η καταχώρηση αυτών των πληροφοριών από μέρους τους, όσο και η γνώση του τι έγινε και πότε.

  Επεξήγηση Υλοποίησης:

  Καταρχήν, ο τύπος του πεδίου στο οποίο θα αποθηκευτεί η πιο πάνω πληροφορία πρέπει να είναι **Datetime.**

  Ας θεωρήσουμε ότι είμαστε στον πίνακα **Person** (Εικόνα 9). Όταν κάνουμε κλικ στο πεδίο **per_EntryDate** εμφανίζεται η πιο κάτω εικόνα :



Εικόνα 26: «Επεξήγηση πεδίου EntryDate»

Από το menu **«Column Properties»** και συγκεκριμένα στο property **«Default Value or Binding»** γράφουμε **getdate(),** έτσι όταν πραγματοποιηθεί μια αλλαγή μέσα στη βάση (Εισαγωγή, Ανανέωση, Διαγραφή), το σύστημα αποθηκεύει από μόνο του την τρέχουσα ημερομηνία και ώρα.

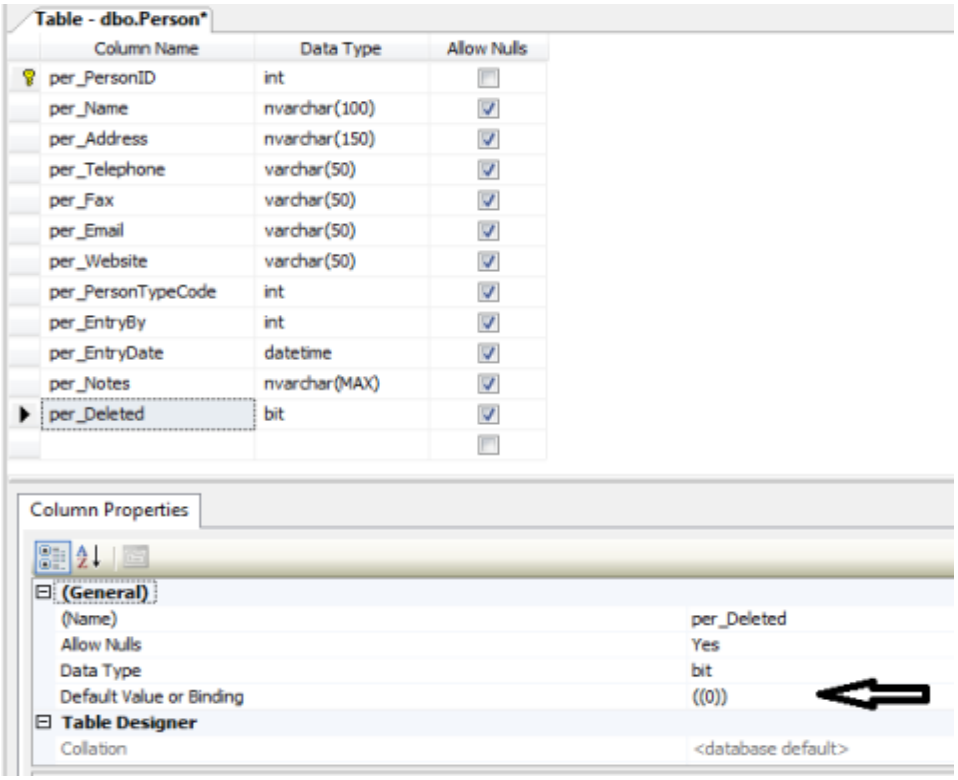
- **Deleted:** Σκοπός του πεδίου αυτού, είναι σε περίπτωση διαγραφής μιας εγγραφής από τη βάση, να μην χάνεται εντελώς η πληροφορία από τη βάση, αλλά να μπαίνει στο πεδίο αυτό η τιμή **«True»**. Έτσι, σε περίπτωση που ο χρήστης για δικούς του λόγους, θελήσει να επαναφέρει αυτή την πληροφορία/εγγραφή να μπορεί να το κάνει.

  Επίσης όταν γίνει μια νέα εισαγωγή σε κάποιο πίνακα, στο πεδίο αυτό δίνεται αυτόματα από το σύστημα (by default) η τιμή **«False».**

  Επεξήγηση Υλοποίησης:

  Εν πρώτοις, ο τύπος του πεδίου αυτού πρέπει να είναι **bit** και μπορεί να πάρει τις τιμές «0» και «1» που αντιστοιχούν στις λογικές τιμές «**True»** ή **False.**

  Έστω ότι είμαστε στον πίνακα **Person** (Εικόνα 9). Όταν κάνουμε κλικ στο πεδίο **per_Deleted** εμφανίζεται η πιο κάτω εικόνα :



Εικόνα 27: «Επεξήγηση πεδίου Deleted»

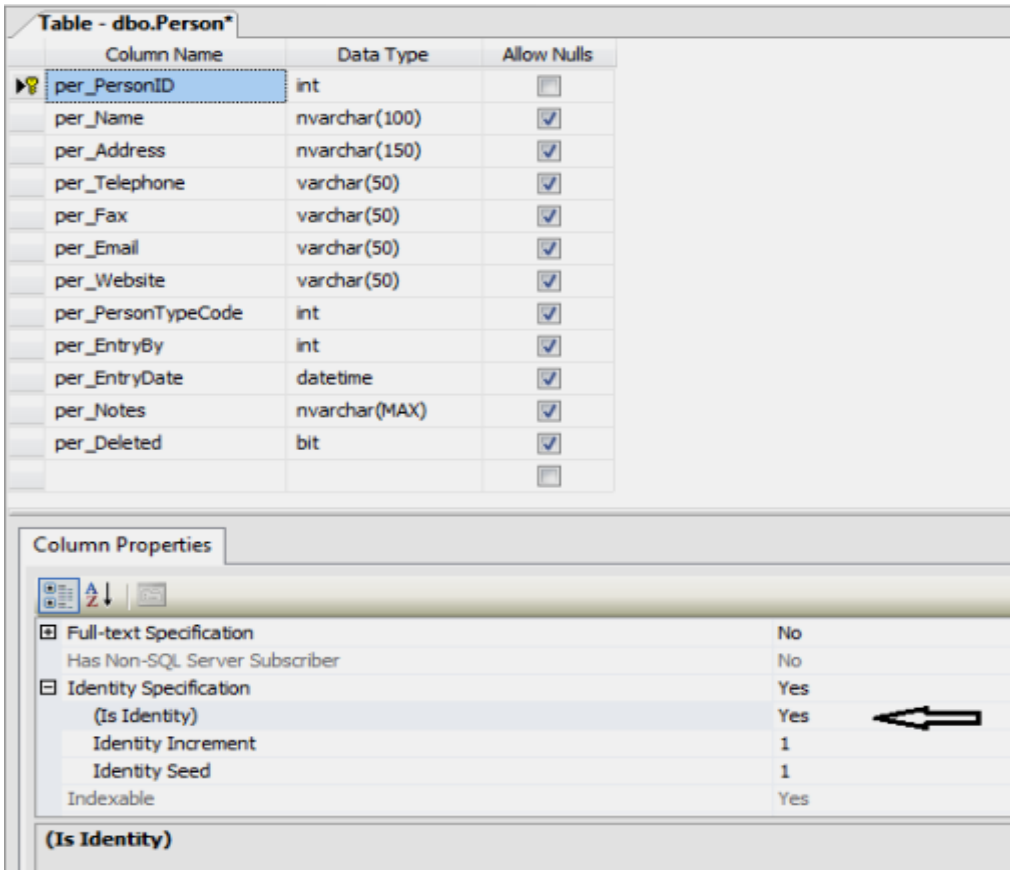
Από το menu **«Column Properties»** και συγκεκριμένα στο property **«Default Value or Binding»** γράφουμε **(0),** έτσι όταν πραγματοποιηθεί μια νέα εισαγωγή στη βάση, το σύστημα αποθηκεύει από μόνο του στο πεδίο αυτό την τιμή «**False».**

- **ID εγγραφής:**

  Τέλος, από το σύστημα και σε όλους τους πίνακες, δίνεται αυτόματα και το μοναδικό ID της κάθε εγγραφής, ξεκινώντας από το «1» και προσθέτοντας ένα για κάθε νέα εγγραφή του πίνακα.

  Επεξήγηση Υλοποίησης:

  Καταρχήν, ο τύπος του πεδίου στο οποίο αποθηκεύεται η πιο πάνω πληροφορία πρέπει να είναι **int.** Ας υποθέσουμε ότι είμαστε και πάλι στον πίνακα Person (Εικόνα 9), όταν κάνουμε κλικ στο πεδίο **per_PersonID** εμφανίζεται η πιο κάτω εικόνα :

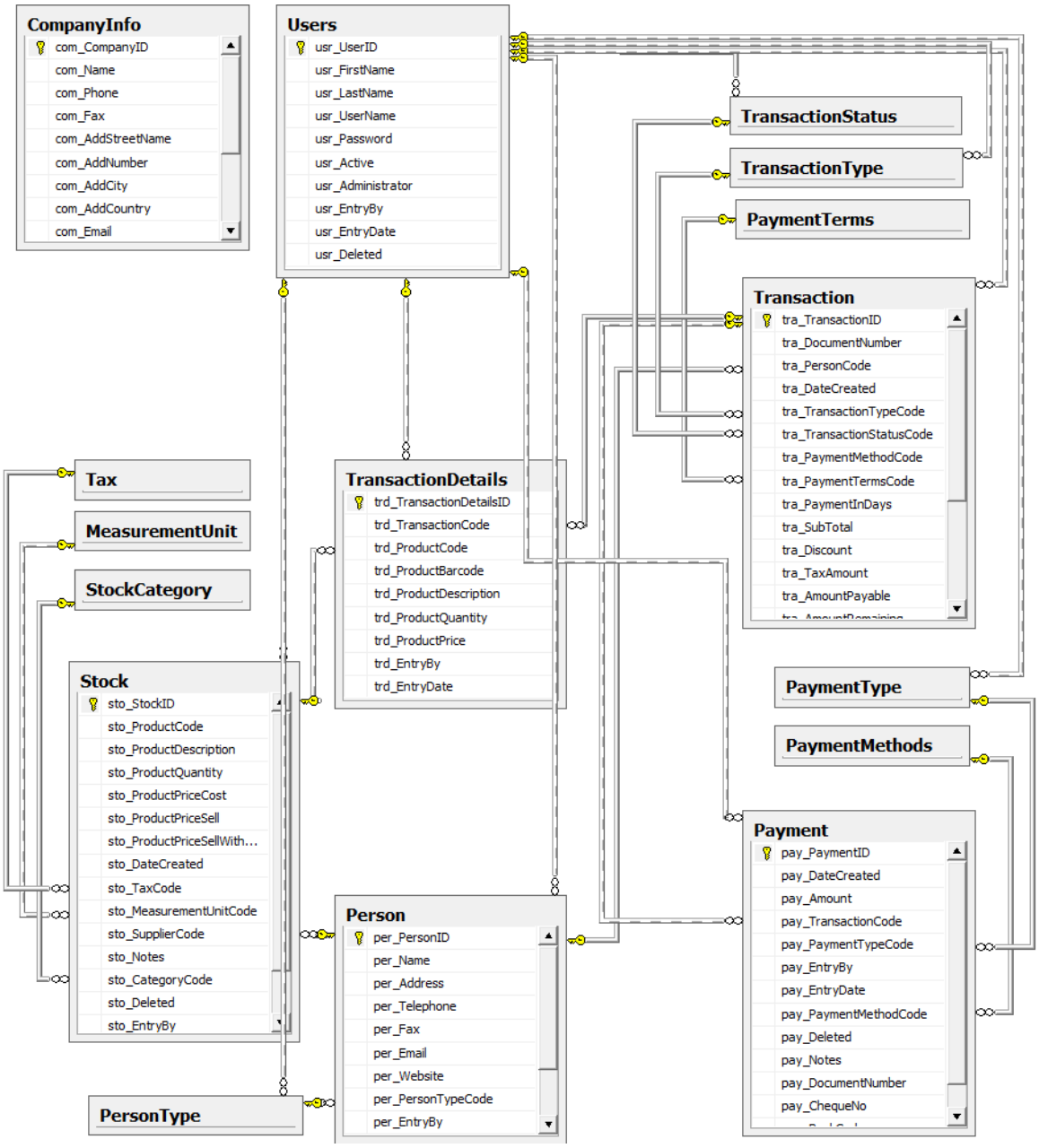


Εικόνα 28: « Property Identity»

Από το menu **«Column Properties»** και συγκεκριμένα στο property **«Identity Specification»,** στην επιλογή **(Is Identity)** επιλέγουμε **«Yes».**

Διάγραμμα Βάσης:

Οι συσχετίσεις των πινάκων της βάσης, όπως αυτές περιγράφηκαν πιο πάνω, φαίνονται και στο πιο κάτω διάγραμμα:



Εικόνα 29: «Διάγραμμα Βάσης»

Συνοψίζοντας, οι συσχετίσεις των πινάκων της βάσης είναι οι εξής:

Ο **Πίνακας Person** έχει για ξένα κλειδιά τα κύρια κλειδιά των Πινάκων **PersonType** και **Users** τα οποία συμβολίζουν αντίστοιχα:

- τον τύπο του ατόμου στον πίνακα αυτό (πελάτης ή προμηθευτής)
- τον χρήστη ο οποίος έκανε την συγκεκριμένη καταχώρηση στον πίνακα

Ο **Πίνακας Payment** έχει για ξένα κλειδιά, τα κύρια κλειδιά των Πινάκων **PaymentType**, **PaymentMethods**, **Transaction** και **Users** τα οποία συμβολίζουν αντίστοιχα:

- τον τύπο της Πληρωμής (έσοδα ή έξοδα)
- τη μέθοδο πληρωμής (Τοις μετρητοίς, Επιταγή, ...)
- τον αριθμό τιμολογίου για το οποίο έγινε η πληρωμή
- τον χρήστη ο οποίος έκανε την συγκεκριμένη καταχώρηση στον πίνακα

Ο **Πίνακας Transaction** έχει για ξένα κλειδιά τα κύρια κλειδιά των Πινάκων **TransactionType**, **TransactionStatus**, **PaymentTerms**, **Person** και **Users** τα οποία συμβολίζουν αντίστοιχα:

- τον τύπο συναλλαγής (αγορά, πώληση, προσφορά)
- την κατάσταση τιμολογίου (ανοιχτό, εξοφλημένο, άκυρο)
- τη συμφωνία πληρωμής
- το άτομο στο οποίο αναφέρεται η συναλλαγή
- τον χρήστη ο οποίος έκανε την συγκεκριμένη καταχώρηση στον πίνακα

Ο **Πίνακας Stock** έχει για ξένα κλειδιά τα κύρια κλειδιά των Πινάκων **StockCategory**, **MeasurementUnit**, **Tax**, **PersonType** και **Users** τα οποία συμβολίζουν αντίστοιχα:

- την κατηγορία του εκάστοτε αποθέματος
- τη μονάδα μέτρησης του κάθε προϊόντος (Μέτρα, Τετρ. Μέτρα, ...)
- τον φόρο προστιθέμενης αξίας του προϊόντος
- τον προμηθευτή που προμήθευσε το συγκεκριμένο προϊόν
- τον χρήστη που έκανε την συγκεκριμένη καταχώρηση στον πίνακα

Ο **Πίνακας TransactionDetails** έχει για ξένα κλειδιά τα κύρια κλειδιά των Πινάκων **Transaction**, **Stock** και **Users** τα οποία συμβολίζουν αντίστοιχα:

- το τιμολόγιο στο οποίο αναφέρονται οι λεπτομέρειες
- το συγκεκριμένο απόθεμα
- τον χρήστη που έκανε την συγκεκριμένη καταχώρηση στον πίνακα

Τέλος, οι πίνακες **TransactionType**, **PaymentType** και **PersonType** έχουν για ξένο κλειδί το κύριο κλειδί του πίνακα **Users,** το οποίο συμβολίζει ποιος χρήστης έκανε την συγκεκριμένη καταχώρηση στους πιο πάνω πίνακες.

Επίλογος:

Η δημιουργία της βάσης σε τέτοιου είδους συστήματα, προηγείται της συγγραφής του κώδικα, γιατί όλος ο κώδικας έχει να κάνει με τη διαχείρισή της. Επομένως η ακριβής γνώση της δομής της βάσης, είναι απαραίτητη πριν την έναρξη ανάπτυξης του κώδικα. Βέβαια κατά τη διάρκεια της ανάπτυξης, μπορούν να γίνονται διάφορες αλλαγές και βελτιώσεις στην βάση, οι οποίες πριν δεν είχαν εντοπισθεί. Τέλος, αφού ολοκληρωθεί η βάση σε ένα πρώτο στάδιο, ακολουθεί η σχεδίαση του User Interface παράλληλα με την ανάπτυξη του κώδικα.

ΚΕΦΑΛΑΙΟ 5: Γενική περιγραφή ανάπτυξης κώδικα

Εισαγωγή:

Στο κεφάλαιο αυτό, γίνεται μια γενική περιγραφή του τρόπου με τον οποίο έχει αναπτυχθεί η εφαρμογή όσον αφορά το κομμάτι του κώδικα. Περιγράφεται το πως διαχωρίστηκε σε τρία projects καθώς επίσης και το περιεχόμενο του καθενός από αυτά.

Διαχωρισμός του κώδικα σε τρία projects (Solution)

Για λόγους ποιότητας, εύκολης ανάπτυξης και αποσφαλμάτωσης ο κώδικας διαχωρίστηκε σε τρία projects:

- Classes
- Enums
- Marmasoft

To project Classes

Στο project αυτό, βρίσκονται όλες οι κλάσεις της εφαρμογής. Οι κλάσεις αυτές, δημιουργήθηκαν κυρίως, κατ'αντιστοιχία των πινάκων που περιέχει η Βάση Δεδομένων της εφαρμογής. Για τη διαχείριση του κάθε πίνακα δηλαδή δημιουργήθηκε και η αντίστοιχη κλάση.
Η ονοματολογία που χρησιμοποιήθηκε για τις κλάσεις, παραπέμπει άμεσα στον πίνακα τον οποίο διαχειρίζεται η κάθε μια από αυτές. Οι εν λόγω κλάσεις παρουσιάζονται πιο κάτω :

- clsBank
- clsCommon
- clsCompany
- clsMeasurementUnit
- clsPayment

- clsPaymentMetdods
- clsPaymentTerms
- clsPerson.
- clsQuery
- clsStock.
- clsStockCategory
- clsTax
- clsTransaction
- clsTransactionDetails
- clsTransactionStatus
- clsUsers

Κάθε κλάση από τις πιο πάνω, πλην των κλάσεων που αναφέρονται σε «πίνακες τύπους» περιέχει :

- **τις κύριες μεταβλητές της** - οι μεταβλητές αυτές αντιστοιχούν στα πεδία του κάθε πίνακα.

- **τη μέθοδο Load()** – η μέθοδος αυτή ανάλογα με την παράμετρο που δέχεται όταν κληθεί παρουσιάζει «δύο συμπεριφορές»:
  - είτε φορτώνει όλες τις εγγραφές του πίνακα σε ένα DataGridView.
  - είτε φορτώνει μια συγκεκριμένη εγγραφή του πίνακα, δίνοντας τις τιμές του κάθε πεδίου της, στις αντίστοιχες μεταβλητές της κλάσης αυτής. Αυτό εξυπηρετεί στην περαιτέρω διαχείριση της συγκεκριμένης εγγραφής.

- **τη μέθοδο Save() –** η μέθοδος **Save** παρουσιάζει επίσης «δύο συμπεριφορές» κατά την κλήση της. Γίνεται έλεγχος και αν αναφερόμαστε:
  - σε νέα εγγραφή, γίνεται κανονική αποθήκευση/προσθήκη της εγγραφής στον συγκεκριμένο πίνακα.
  - σε υπάρχουσα εγγραφή, γίνεται ενημέρωση (Update) της εγγραφής αυτής.

- **τη μέθοδο Delete() –** η μέθοδος αυτή χρησιμοποιείται για τη διαγραφή μιας εγγραφής από κάποιο πίνακα. Ουσιαστικά, δεν γίνεται διαγραφή αλλά μια ανανέωση (Update) της συγκεκριμένης εγγραφής, αλλάζοντας τη τιμή του πεδίου Deleted από «False» σε «True». Ο λόγος που συμβαίνει αυτό, είναι σε περίπτωση διαγραφής μια εγγραφής η οποία συσχετίζεται άμεσα με δεδομένα άλλου πίνακα να μην υπάρξει πρόβλημα.

Οι κλάσεις **clsStock, clsPayment** και **clsTransaction** διαθέτουν επιπλέον τις εξής μεθόδους:

Κλάση clsStock:

Μέθοδος Restock() – η μέθοδος αυτή καλείται κατά την διάρκεια έκδοσης/αποθήκευσης ή και ανανέωσης ενός τιμολογίου και παρουσιάζει τις εξής συμπεριφορές:

- αν το τιμολόγιο είναι τύπου **«Purchase»,** δηλαδή αναφέρεται στην αγορά προϊόντων από την εταιρεία, τότε η μέθοδος **Restock()** κάνει ανανέωση του πίνακα Stock προσθέτοντας τα νέα προϊόντα.
- αν το τιμολόγιο είναι τύπου **«Sale»,** δηλαδή αναφέρεται σε μια πώληση της εταιρείας, τότε η μέθοδος **Restock()** κάνει ανανέωση του πίνακα Stock αφαιρώντας τα προϊόντα που έχουν πωληθεί.

Κλάση clsPayment:

Μέθοδος GenerateDocumentNumber() – κατά την αποθήκευση μιας νέας πληρωμής/απόδειξης, καλείται η μέθοδος αυτή και δημιουργεί έναν μοναδικό αριθμό εγγράφου.

Κλάση clsTransaction:

- **Μέθοδος LoadByDocCode() –** η μέθοδος αυτή χρησιμεύει στην αναζήτηση ενός συγκεκριμένου τιμολογίου ή προσφοράς μέσα από το DataGridView στο οποίο εμφανίζονται οι συναλλαγές αυτές. Δέχεται σαν παράμετρο τον **«αριθμό εγγράφου»** και κάνει αναζήτηση εκτελώντας ένα Query προς τη

βάση. Η πιο πάνω διαδικασία έχει ως αποτέλεσμα την εμφάνιση μόνο του συγκεκριμένου εγγράφου στο DataGridView για προφανείς λόγους.

- **Μέθοδος GenerateDocumentNumber() –** κατά την αποθήκευση ενός τιμολογίου πώλησης ή προσφοράς, καλείται η μέθοδος αυτή και ανάλογα με τον τύπο της συναλλαγής, δημιουργεί έναν μοναδικό «αριθμό εγγράφου».

Οι κλάσεις οι οποίες δημιουργήθηκαν για τη διαχείριση των **«πινάκων τύπων»** περιέχουν :

- **τις κύριες μεταβλητές** τους που αντιστοιχούν στα πεδία του εκάστοτε πίνακα
- **τη μέθοδο Load(),** η οποία έχει την ίδια λειτουργία με τη μέθοδο Load() όπως περιγράφηκε στις προηγούμενες κλάσεις και χρησιμοποιείται κυρίως για την φόρτωση των περιεχομένων του εκάστοτε πίνακα σε combobox που τοποθετήθηκαν στα διάφορα Tabs και φόρμες.

Στο project αυτό, πέραν από τις κλάσεις για τη διαχείριση των πινάκων, υπάρχουν και οι κλάσεις **clsCommon** και **clsQuery.**

Κλάση clsCommon:

Στην κλάση **clsCommon** υπάρχουν διάφορες βοηθητικές μέθοδοι, οι οποίες ανακτούν δεδομένα που η χρήση τους είναι πολύ συχνή μέσα στον κώδικα και είναι επίσης κοινά και αναγκαία για τις υπόλοιπες κλάσεις. Η κλάση αυτή δημιουργήθηκε κυρίως για να αποφευχθεί η επαναλαμβανόμενη συγγραφή των ίδιων τμημάτων κώδικα.

Μέθοδοι της κλάσης clsCommon:

- **LoggedInUserID() -** επιστρέφει το ID του συνδεδεμένου χρήστη.

- **MyConnectionString() -** επιστρέφει το ConnectionString που είναι απαραίτητο για τη σύνδεση με τη βάση.

- **GetDBServerName() -** επιστρέφει το όνομα του Server.

- **GetDBUserName() -** επιστρέφει το UserName που χρησιμοποιείται για τη σύνδεση με τον SQL Server.

- **GetDBPassword() -** επιστρέφει το password που χρησιμοποιείται για την σύνδεση με τον SQL Server.

- **GetDBName() -** επιστρέφει το όνομα της βάσης.

Επιπλέον διαθέτει τις μεθόδους **isNumeric()** και **ValidateCharInput()**, οι οποίες καλούνται για να επιβεβαιωθεί η εγκυρότητα των στοιχείων (ως προς τον τύπο τους) που έδωσε ο χρήστης σε κάποια από τα πεδία εισαγωγής.

Κλάση clsQuery:

Η clsQuery είναι μια βοηθητική κλάση, η οποία δημιουργήθηκε για να αποφευχθεί η επαναλαμβανόμενη συγγραφή ιδίων τμημάτων κώδικα, τα οποία έχουν να κάνουν με τη σύνδεση με τη βάση και την εκτέλεση ερωταποκρίσεων προς αυτήν.
Η κλάση αυτή διαθέτει τις εξής μεθόδους:

- **clsQuery() ->** Η μέθοδος αυτή δέχεται σαν παράμετρο ένα **Query** και το αρχικοποιεί στο command.

- **Execute() ->** Με τη μέθοδο αυτή, γίνεται η σύνδεση με τη βάση και εκτελείται το Query το οποίο αρχικοποιήθηκε με την μέθοδο «clsQuery()». Τα αποτελέσματα που θα επιστραφούν αποθηκεύονται σε ένα DataTable.

- **ExecuteNonQuery ->** Με τη μέθοδο αυτή, όπως και στη μέθοδο «Execute()», γίνεται η σύνδεση με τη βάση και εκτελείται το Query, με τη

διαφορά, ότι τα Queries που εκτελούνται με αυτή τη μέθοδο δεν επιστρέφουν αποτελέσματα.

Το project Enums

Στο project **Enums** βρίσκεται η κλάση **clsEnums** στην οποία δημιουργήθηκαν όλα τα **enums** που χρησιμοποιούνται στην εφαρμογή. Ουσιαστικά είναι τύποι, τους οποίους οι τιμές αντικατοπτρίζουν τιμές οι οποίες είναι αποθηκευμένες στη βάση.

Τι είναι Enum:

Η λέξη κλειδί **«enum»** χρησιμοποιείται για να δηλώσει μια **«απαρίθμηση».** Ένας τύπος απαρίθμησης, παρέχει έναν αποτελεσματικό τρόπο για να καθοριστεί ένα σύνολο από σταθερές που μπορούν να αποδοθούν σε μια μεταβλητή. Κάθε τύπος απαρίθμησης, έχει ένα υποκείμενο τύπου, το οποίο μπορεί να είναι οποιουδήποτε τύπου εκτός από **«char»**. Ο προεπιλεγμένος τύπος (default), που υποστηρίζεται είναι ο **«int»** και εξ'ορισμού η πρώτη απαρίθμηση έχει τιμή «0» ενώ η τιμή κάθε διαδοχικής απαρίθμησης αυξάνεται κατά «1».

Για παράδειγμα, ας θεωρηθεί ότι πρέπει να οριστεί μια μεταβλητή της οποίας η αξία θα αποτελέσει μια ημέρα της εβδομάδας. Υπάρχουν μόνο επτά σημαντικές αξίες τις οποίες η μεταβλητή αυτή θα μπορούσε να αποθηκεύει.
Προκειμένου να οριστούν οι τιμές αυτές, μπορεί να γίνει χρήση ενός τύπου απαρίθμησης, που έχει δηλωθεί χρησιμοποιώντας τη λέξη-κλειδί **enum.** Πιο κατανοητό θα γίνει στο επόμενο παράδειγμα.

Παράδειγμα χρήσης του enum:

Όπως ήδη έχει αναφερθεί σε προηγούμενο κεφάλαιο, στον πίνακα **Person** αποθηκεύονται δύο τύποι ατόμων, οι πελάτες και οι προμηθευτές. Για το λόγο αυτό δημιουργήθηκε ο Πίνακας **PersonType** που ουσιαστικά περιγράφει τον τύπο του ατόμου στον πίνακα Person.

Στον πίνακα **PersonType** τώρα, για **pty_PersonTypeID=1** το πεδίο **pty_Description** έχει την τιμή **«CUSTOMER»** ενώ για **pty_PersonTypeID=2** το πεδίο **pty_Description** έχει την τιμή **«SUPPLIER».**

Με βάση τα πιο πάνω μπορεί να δημιουργηθεί ένα **enum,** που να αντικατοπτρίζει τις τιμές των πεδίων του πίνακα **PersonType** όπως φαίνεται πιο κάτω**:**

```
namespace Enums
{
   public class clsEnums
   {
       public enum PersonType
       {
           Customer = 1,
           Supplier = 2
       }
   }
}
```

Που χρησιμεύει αυτό:

Κατά τη συγγραφή του κώδικα, και ειδικότερα όταν υπάρχουν πολλοί τύποι δεδομένων, είναι δύσκολο να θυμάται κανείς τις τιμές που αντιστοιχούν στον κάθε ένα από αυτούς. Έτσι με τη δημιουργία των **enums** μπορούν να ανακτηθούν οι τιμές αυτές πολύ εύκολα όπως φαίνεται στο επόμενο παράδειγμα :

```
txtPersonType.Tex=
Convert.ToString(Convert.ToInt32(Enums.clsEnums.PersonType.Customer));
```

Στο πιο πάνω απόσπασμα κώδικα, «καλείται» από το project **Enums** η κλάση **clsEnums** και ακολούθως το **enum PersonType,** έτσι δίνεται σε ένα textbox σε μορφή string η τιμή του τύπου Customer.

To project Marmasoft

Στο project αυτό βρίσκονται όλες οι φόρμες της εφαρμογής (User Interface) καθώς επίσης και ο κώδικας που εκτελείται πίσω από αυτές για κάθε αλληλεπίδραση του χρήστη με το σύστημα.

Οι φόρμες οι οποίες δομούν το User Interface:

- Η φόρμα για την είσοδο χρήστη **(frmLogin).**
- Η αρχική φόρμα **(frmWelcomeScreen).**
- Η *κύρια φόρμα* για τη διαχείριση της εφαρμογής**(frmMain)**. Από την φόρμα αυτή **ο χρήστης μπορεί να διαχειριστεί**:
    - Τους Πελάτες
    - Τους Προμηθευτές
    - Τα αποθέματα
    - Τις πληρωμές
    - Τις αναφορές
    - Όλες τις συναλλαγές (Αγορές, Πωλήσεις, Προσφορές)

καθώς επίσης να μεταφερθεί στις φόρμες:

- frmSettings (Ρυθμίσεις)
- frmTransactionDetails (Λεπτομέρειες Συναλλαγής)
- frmCustomReportGenerator (Προσαρμοσμένη Αναφορά)

- **frmSettings ->** Η φόρμα από την οποία ο χρήστης μπορεί να διαχειριστεί όλα τα στοιχεία της εταιρείας, τους χρήστες της εφαρμογής, τις εμπλεκόμενες τράπεζες καθώς και τις κατηγορίες αποθεμάτων.

- **frmTransactionDetails ->** Η φόρμα στην οποία ο χρήστης μπορεί να διαχειριστεί τις λεπτομέρειες των συναλλαγών της εταιρείας (Αγορές, Πωλήσεις, Προσφορές).

- **frmCustomReportGenerator ->** Στη φόρμα αυτή, ο χρήστης μπορεί να δημιουργήσει δυναμικά μια αναφορά, σχετικά με δεδομένα που συντηρούνται στη βάση, και να την εξάγει σε excel, είτε για να την εκτυπώσει, είτε για να την αποθηκεύσει στον υπολογιστή για άλλη χρήση.

Τέλος, υπάρχουν και οι φόρμες **frmTransactionReport** και **frmPaymentReport,** στις οποίες γίνεται προεπισκόπιση των αναφορών σχετικά με τις συναλλαγές (τιμολόγια, προσφορές) και τις πληρωμές της εταιρείας

αντίστοιχα, πριν από την αποθήκευση ή εκτύπωσή τους. Περαιτέρω επεξήγηση των φορμών θα γίνει στο επόμενο κεφάλαιο.

Application configuration file:

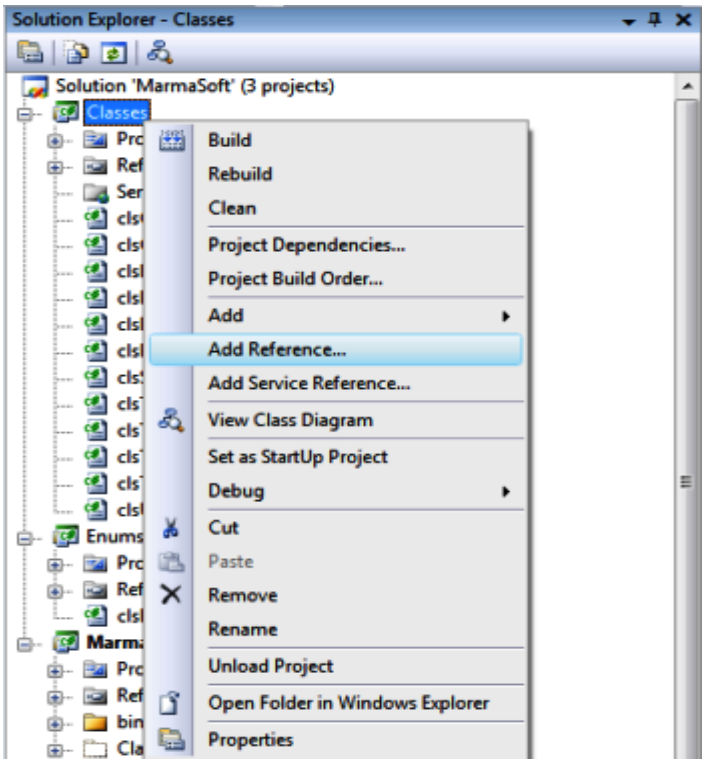
Πέραν από τις φόρμες που σχεδιάστηκαν, στο project αυτό υπάρχει ένα **application configuration file (app.config).** Το app.config είναι ένα XML αρχείο στο οποίο μπορεί να γίνει αποθήκευση δεδομένων ρύθμισης παραμέτρων, οι οποίες χρησιμοποιούνται από την εφαρμογή. Οι πληροφορίες αυτές αποθηκεύονται κυρίως ως ζευγάρια «τιμή-κλειδί» και ένα παράδειγμα τέτοιων παραμέτρων είναι το «connection string» που χρησιμοποιείται από την εφαρμογή για την σύνδεση με την βάση δεδομένων.

Για την επεξεργασία του αρχείου αυτού και ουσιαστικά για την αλλαγή των τιμών που περιέχει, μπορεί να χρησιμοποιηθεί οποιοδήποτε πρόγραμμα επεξεργασίας κειμένου (συμπεριλαμβανομένου και του Notepad). Η εφαρμογή θα φορτώσει τις τιμές από το αρχείο αυτό, χωρίς να χρειαστεί να αλλάξει ή να μεταγλωττιστεί ο πηγαίος κώδικας κάθε φορά που αλλάζουν τα δεδομένα του.

Το όφελος από τη χρήση του αρχείου αυτού, έγκειται στο γεγονός ότι φιλοξενεί παραμέτρους συχνά χρησιμοποιούμενες στον κώδικα, οι οποίες μπορεί για διάφορους λόγους να πρέπει να αλλάξουν. Έτσι ο προγραμματιστής μπορεί απλά να αλλάξει τις τιμές των παραμέτρων στο αρχείο αυτό, αποφεύγοντας τη χρονοβόρα διαδικασία αλλαγής της κάθε μιας από αυτές ξεχωριστά στον πηγαίο κώδικα της εφαρμογής.

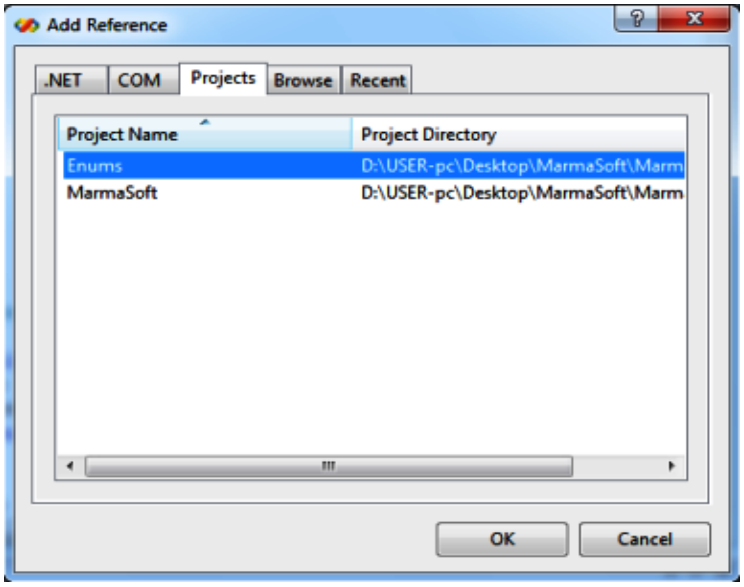
Παράδειγμα συσχέτισης των projects:

Για να καταστεί δυνατή η επικοινωνία/αλληλεπίδραση μεταξύ των τριών projects είναι απαραίτητο να γίνει η μεταξύ τους σύνδεση.
Ας θεωρηθεί, ότι σε μια από τις κλάσεις του **project Classes** παρουσιάστηκε η ανάγκη χρήσης ενός από τα **enum** που βρίσκονται **στο project Enums.** Για να γίνει εφικτό αυτό, πρέπει πρώτα να γίνει η πιο κάτω διαδικασία για την σύνδεση των δύο projects :



Εικόνα 30: «Σύνδεση των projects 1/2»

Από τον **«Solution Explorer»** του Visual Studio (Εικόνα 28), «δεξί-κλικ» στο project **«Classes»** και από το menu που εμφανίζεται πατάμε στην επιλογή «**Add Reference…».** Στη συνέχεια, από το νέο παράθυρο που εμφανίζεται (Εικόνα 29), αφού επιλεγεί πρώτα το **«tab Projects»**, επιλέγουμε το **project «Enums»** και πατάμε «OK».



Εικόνα 31: «Σύνδεση των project 2/2»

Επίλογος:

Στο κεφάλαιο αυτό, έγινε παρουσίαση του τρόπου με τον οποίο διαχωρίστηκε ο κώδικας της εφαρμογής σε τρία projects, καθώς επίσης και του βασικού περιεχομένου των projects αυτών. Στο επόμενο κεφάλαιο ακολουθεί η περιγραφή του User Interface, καθώς και παραπομπές σε περιγραφή κώδικα που βρίσκεται πίσω από αυτό.

ΚΕΦΑΛΑΙΟ 6 : Γενική περιγραφή του User Interface

Εισαγωγή:

Ένα από τα σημαντικότερα τμήματα κατά την υλοποίηση μιας εφαρμογής είναι το User Interface. Πρέπει να δοθεί μεγάλη έμφαση κατά τον σχεδιασμό του γιατί είναι αυτό που θα δώσει την πρώτη εντύπωση στον χρήστη.
Συνεπώς, πρέπει να είναι σωστά σχεδιασμένο, ώστε να είναι όσο το δυνατό περισσότερο φιλικό ως προς το χρήστη και να μην δημιουργούνται απορίες μόνο με το κοίταγμά του. Να είναι δηλαδή ένα ωραίο και ευχάριστο περιβάλλον, του οποίου οι ενέργειες που μπορούν να εκτελεστούν να είναι απλές και ξεκάθαρες.

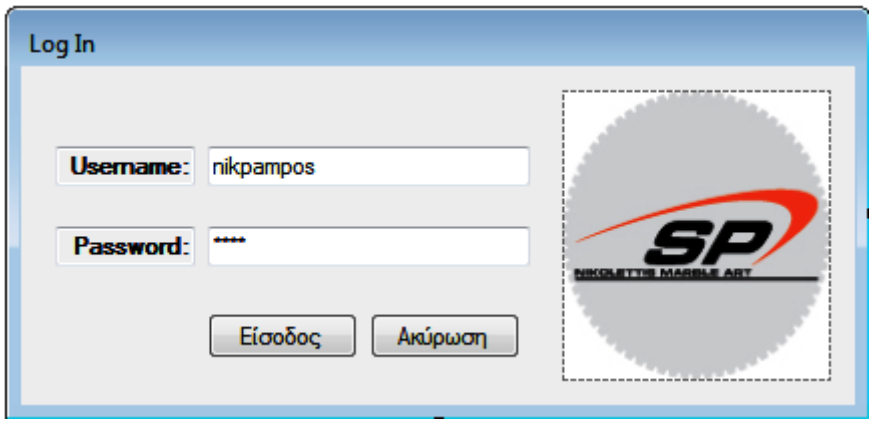
Στο κεφάλαιο αυτό παρουσιάζεται ο τρόπος σχεδίασης του User Interface της εφαρμογής ώστε να τηρούνται τα πιο πάνω, καθώς υπάρχουν και παραπομπές σε μερικά τμήματα κώδικα που βρίσκονται πίσω από αυτό.

Οι φόρμες της εφαρμογής

Το User Interface της εφαρμογής δομείται κυρίως από επτά φόρμες, οι οποίες θα παρουσιαστούν και θα αναλυθούν πιο κάτω:

frmlogin (Log In):

Είναι η φόρμα στην οποία ο χρήστης πρέπει να εισάγει τα προσωπικά του στοιχεία για να μπορέσει να εισέλθει στο σύστημα.



Εικόνα 32: «Φόρμα frmLogin»

Στη φόρμα αυτή τοποθετήθηκαν δύο «textbox» στα οποία ο χρήστης μπορεί να εισάγει τα στοιχεία του (**Username** και **Password**) καθώς επίσης και δύο ενδεικτικές «ετικέτες» που καθορίζουν την ακριβή τοποθεσία του καθενός.

Επίσης επιλέχθηκε, για λόγους ασφάλειας, οι χαρακτήρες που πληκτρολογούνται μέσα στο textbox για το Password, να εμφανίζονται σαν αστεράκια, για να μην μπορεί να δει τον κωδικό ο οποιοσδήποτε. Στο δεξιό μέρος της φόρμας τοποθετήθηκε ένα **PictureBox** στο οποίο βρίσκεται το λογότυπο της εταιρείας και στο κέντρο τα κουμπιά **«Log In»** και **«Cancel».**

Αν πατηθεί το κουμπί **«Cancel»** η εφαρμογή κλείνει (βλ. Παράρτημα-Διαδικασία 1), ενώ όταν πατηθεί το κουμπί **«Log In»,** γίνεται έλεγχος αν είναι σωστά τα στοιχεία του χρήστη. Στην περίπτωση που είναι ορθά η εφαρμογή ανοίγει, αλλιώς εμφανίζεται κατάλληλο μήνυμα που προτρέπει το χρήστη να εισάγει ξανά τα στοιχεία του (βλ. Παράρτημα-Διαδικασία 2).

frmWelcomeScreen:

Μετά την εισαγωγή του χρήστη στο σύστημα εμφανίζεται η πιο κάτω φόρμα:

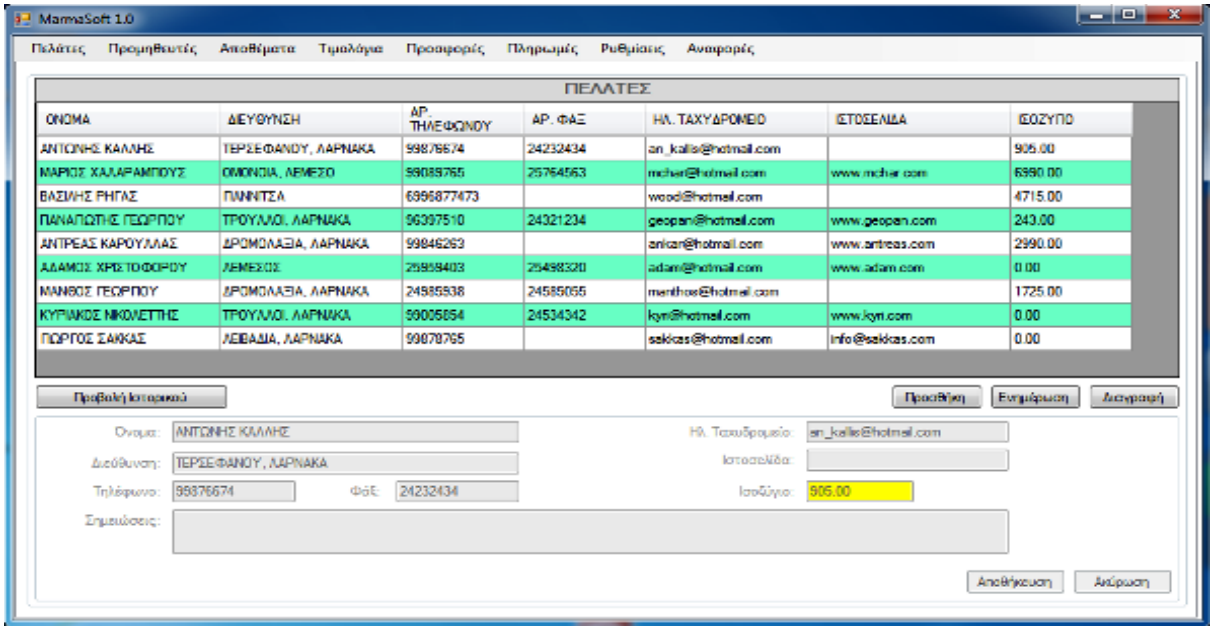
Εικόνα 33: «Φόρμα frmWelcomeScreen»

Στη φόρμα αυτή, τοποθετήθηκαν μέσα σε **PictureBox** εικονίδια, που με το πάτημά τους οδηγούν στα αντίστοιχα Tabs της κύριας φόρμας (**frmMain**) της εφαρμογής. Μόλις ο χρήστης δηλαδή πατήσει σε οποιοδήποτε PictureBox, η φόρμα αυτή κλείνει και οδηγείται στο αντίστοιχο tab της φόρμας frmMain. Αντίθετα, αν κάνει κλικ στο εικονίδιο που βρίσκεται κάτω δεξιά η εφαρμογή κλείνει.
Στα PictureBox ορίστηκαν τα εξής Events:

- **MouseEnter ->** Όταν ο δείκτης του ποντικιού εισέλθει σε κάποιο από τα PictureBox, καλείται το event αυτό και το Style της φωτογραφίας αλλάζει.
- **MouseLeave ->** Το event αυτό, καλείται όταν εξέλθει ο δείκτης του ποντικιού από το PictureBox και η φωτογραφία επαναφέρεται στο αρχικό της Style.
- **Click ->** Όταν ο χρήστης πατήσει σε κάποιο από τα PictureBox, η φόρμα αυτή κλείνει, ανοίγει η κύρια φόρμα της εφαρμογής και οδηγείται στο αντίστοιχο tab.

frmMain (Marmasoft):

Είναι η κύρια φόρμα της εφαρμογής, από την οποία ο χρήστης μπορεί να πλοηγηθεί σε όλες τις φόρμες και να διαχειριστεί όλα τα ζητήματα της εταιρείας:



Εικόνα 34: «Φόρμα frmmain – Tab Persons»

Η φόρμα αυτή, είναι στην ουσία ένα **TabControl** και με τη βοήθεια ενός **MenuStrip,** αποτελούμενο από διάφορα **ToolStripMenuItems** (Πελάτες, Προμηθευτές, Αποθέματα, Τιμολόγια, Πληρωμές, Εργαλεία, Αναφορές, Προσφορές), ο χρήστης μπορεί να μεταφερθεί, είτε στα **tabs** που αυτή περιέχει, είτε στις υπόλοιπες φόρμες της εφαρμογής. Τα tabs της frmMain είναι τα εξής:

- **Persons**
- **Payments**
- **Stock**
- **Transactions**

Σε όλα τα tabs και φόρμες, η σχεδίαση είναι παρόμοια με την πιο πάνω εικόνα, με μερικές τροποποιήσεις ανάλογα με τις εκάστοτε ανάγκες και απαιτήσεις. Σε όλα τα tabs τοποθετήθηκαν δύο GroupBox. Στο μεγάλο Groupbox τοποθετήθηκε ένα **DataGridView** στο οποίο θα εμφανίζονται τα σημαντικά πεδία του κάθε πίνακα, ανάλογα με το **Item** που επέλεξε ο χρήστης, ενώ πιο κάτω από αυτό βρίσκονται τα αναγκαία κουμπιά για τη διαχείριση του κάθε πίνακα. Όπως θα δούμε και στη συνέχεια, τα κουμπιά **«Προσθήκη», «Ανανέωση»** και **«Διαγραφή»** τοποθετήθηκαν σε όλα τα tabs και φόρμες και κάτω από αυτά τοποθετήθηκε ένα δεύτερο **Groupbox (grpDetails).**

Στο Groupbox αυτό, για κάθε πεδίο του πίνακα υπάρχει το αντίστοιχο «textbox» με ενδεικτικά «labels». Επιπλέον στο grpDetails υπάρχει και ένα textbox στο οποίο ο χρήστης μπορεί να γράψει σημειώσεις για την κάθε εγγραφή του πίνακα αν αυτό χρειαστεί, καθώς επίσης και τα κουμπιά **«Αποθήκευση», «Ακύρωση».** Λεπτομερής ανάλυση των πιο πάνω γίνεται στη συνέχεια του κεφαλαίου.

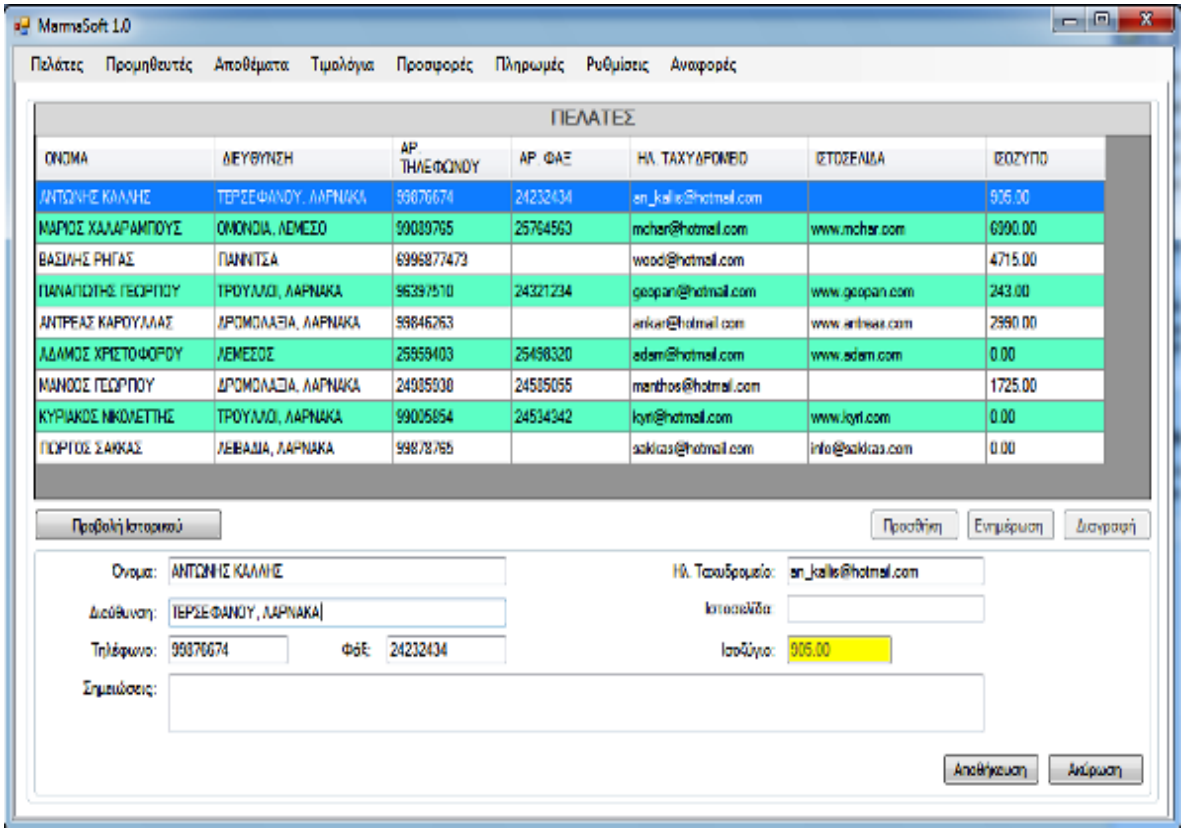
Ενέργειες που εκτελούνται σε όλα τα tabs

Ας θεωρηθεί ότι επιλέχθηκε το «**Item Πελάτες»** και βρισκόμαστε στο **tab Persons** (Εικόνα 32). Οι Πελάτες και οι Προμηθευτές έχουν τα ίδια σημαντικά στοιχεία/πεδία, γι'αυτό όταν ο χρήστης επιλέξει από το **MenuStrip** το **ToolStripMenuItem** «Πελάτες» ή «Προμηθευτές» οδηγείται πάντοτε στο **tab**

Persons της **frmMain**, με τη διαφορά ότι ανάλογα με το πιο **Item** επιλέχθηκε, στο **DataGridView** φορτώνονται τα αντίστοιχα δεδομένα.
Αν ο χρήστης, δηλαδή, επιλέξει το «Item Πελάτες», το DataGridView θα εμφανιστούν τα στοιχεία όλων των Πελατών, ενώ αν επιλέξει το «Item Προμηθευτές» θα εμφανιστούν όλα τα στοιχεία των Προμηθευτών (βλ. Παράρτημα-Διαδικασία 5-8).

Όπως φαίνεται και από την «Εικόνα 32», αρχικά το **grpDetails** σε όλα τα tabs είναι μη διαθέσιμο (Enabled False). Πατώντας το κουμπί **«Προσθήκη»**, το grpDetails γίνεται διαθέσιμο για το χρήστη (Enabled True) και τα υπόλοιπα κουμπιά που βρίσκονται έξω από αυτό γίνονται μη διαθέσιμα (βλ. Παράρτημα-Διαδικασία 9). Ο χρήστης μπορεί να εισάγει στα textbox, τα στοιχεία για μια νέα εγγραφή του πίνακα και να πατήσει το κουμπί **«Αποθήκευση»** (βλ. Παράρτημα-Διαδικασία 10-11). Αν ο χρήστης πατήσει το κουμπί **«Ακύρωση»** ή **«Escape**» (Παράρτημα-Διαδικασία 12), το tab επανέρχεται στην αρχική του κατάσταση.

Όταν ο χρήστης επιλέξει οποιαδήποτε εγγραφή από το DataGridView, αμέσως τα στοιχεία της εγγραφής εισάγονται στα αντίστοιχα textbox που βρίσκονται μέσα το grpdetails (βλ. Παράρτημα-Διαδικασία 13). Στη συνέχεια, αν πατηθεί το κουμπί **«Ενημέρωση»** ή το κουμπί **«Enter»** το grpDetails γίνεται διαθέσιμο και ο χρήστης μπορεί να διαχειριστεί την εγγραφή αυτή, αλλάζοντας την τιμή σε οποιοδήποτε από τα πεδία της (βλ. Εικόνα 33). Ακολούθως μπορεί να πατήσει το κουμπί **«Αποθήκευση»,** για να γίνει αποθήκευση των αλλαγών στη βάση και τότε το tab θα επανέλθει στην αρχική του κατάσταση, ή να πατήσει το κουμπί **«Ακύρωση»** για να μην γίνει αποθήκευση των αλλαγών και πάλι το tab θα επανέλθει στην αρχική του κατάσταση.
Ο χρήστης μπορεί να διαγράψει μια εγγραφή, επιλέγοντας την και πατώντας το κουμπί «Διαγραφή» (βλ. Παράρτημα-Διαδικασία 14-15).



Εικόνα 35: «Tab Persons after Selection Changed»

Στο σημείο αυτό, αξίζει να σημειωθεί ότι :

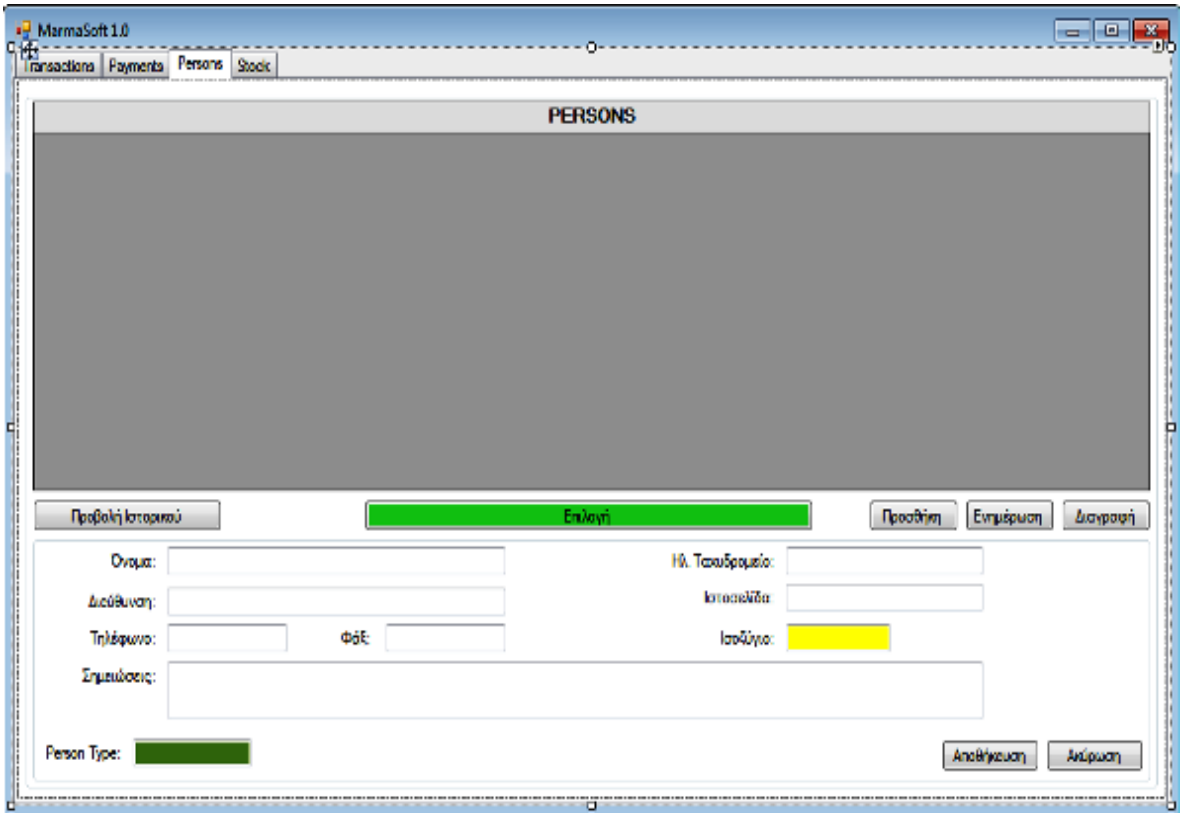
- Σε όλα τα textbox που είναι μη διαθέσιμα για το χρήστη και τα οποία θα λαμβάνουν αυτόματα τιμές από το σύστημα, ορίστηκε το κίτρινο χρώμα.
- Ο χρήστης για κάθε ενέργεια που κάνει (σωστή ή λανθασμένη) ενημερώνεται με κατάλληλα μηνύματα, που το βοηθούν, είτε στην επιβεβαίωση της σωστής εκτέλεσης κάποιας διαδικασίας, είτε στη σωστή χρήση του συστήματος.

Ιδιαίτερα χαρακτηριστικά ανά tab

Πέραν από τα κοινά χαρακτηριστικά που διαθέτουν τα tabs, το κάθε ένα από αυτά έχει τις δικές του ιδιαιτερότητες γι’ αυτό θα γίνει ανάλυση του καθενός ξεχωριστά.

Tab Persons:

Όπως ήδη έχει αναφερθεί, αν επιλεγεί το «Item Πελάτες» ή το «Item Προμηθευτές» ανοίγει το tab Persons και στο DataGridView φορτώνονται τα ανάλογα δεδομένα.



Εικόνα 36: «Tab Persons»

Όπως φαίνεται και από την πιο πάνω εικόνα, στο Tab αυτό υπάρχουν τρία ιδιαίτερα χαρακτηριστικά:

Κουμπί «Προβολή Ιστορικού»:

Ο χρήστης μπορεί να επιλέξει από το DataGridView ένα συγκεκριμένο Πελάτη ή Προμηθευτή και να πατήσει το κουμπί **«Προβολή Ιστορικού».** Αυτόματα μεταφέρεται στο «tab Transactions», όπου στο DataGridView του tab αυτού, θα εμφανιστούν όλες οι συναλλαγές (τιμολόγια) που αφορούν το συγκεκριμένο άτομο (βλ. Παράρτημα-Διαδικασία 16-18). Μάλιστα, στο tab αυτό του δίνεται και η επιλογή για κλείσιμο της Προβολής Ιστορικού και επιστροφή πίσω στο tab Persons.

Κουμπί «Επιλογή»

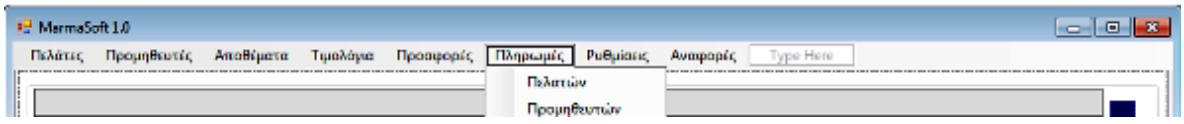
Το κουμπί **«Επιλογή»,** όταν η εφαρμογή τρέχει είναι κρυφό. Εμφανίζεται μόνο, αν κατά τη διάρκεια έκδοσης ενός τιμολογίου, πατηθεί το ανάλογο κουμπί στο «tab Transaction» για επιλογή του ατόμου στο οποίο αναφέρεται η συναλλαγή (τιμολόγιο ή προσφορά). Αφού επιλεγεί το συγκεκριμένο άτομο, το κουμπί γίνεται και πάλι κρυφό και ο χρήστης επιστρέφει στο «tab Transactions».

Τέλος, το label **«Person Type»** μαζί με το αντίστοιχο textbox είναι κρυφά. Το textbox τοποθετήθηκε, ώστε να παίρνει τιμή κατά την επιλογή, είτε του «item Πελάτες», είτε του «item Προμηθευτές». Η τιμή του χρησιμοποιείται κατά την εμφάνιση των ανάλογων δεδομένων στο DataGridView του tab αυτού (βλ. Παράρτημα-Διαδικασία 5).

Tab Payments:

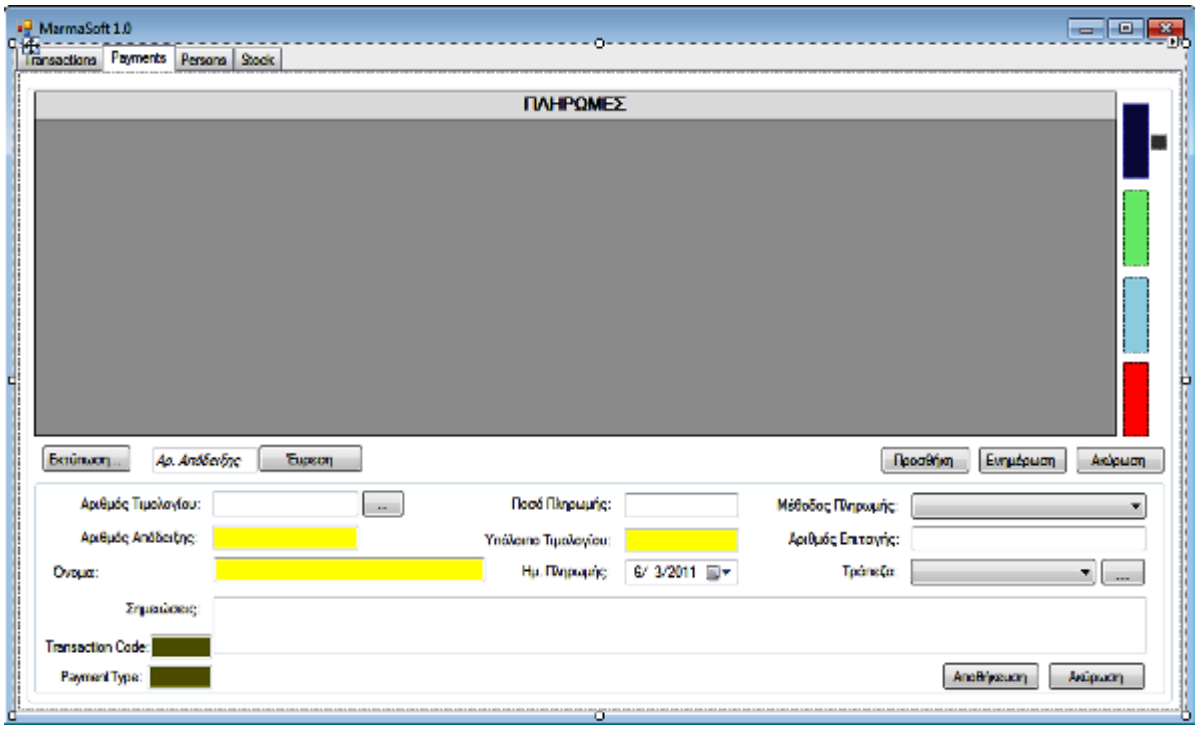
Υπάρχουν δύο τύποι πληρωμών. Αν ο χρήστης επιλέξει το «Item Πληρωμές» (βλ. Εικόνα 35), εμφανίζονται δύο επιλογές:

- Πληρωμές Πελατών
- Πληρωμές Προμηθευτών



Εικόνα 37: «Item Πληρωμές»

Επιλέγοντας ένα από τα δύο ο χρήστης, μεταφέρεται στο tab Payments, στο DataGridView του οποίου εμφανίζονται οι ανάλογες πληροφορίες. Το tab Payments φαίνεται πιο κάτω:



Εικόνα 38: «Tab Payments»

Ιδιαίτερα χαρακτηριστικά του tab Payment:

Τα textbox με κίτρινο χρώμα υποδηλώνουν δεδομένα/πεδία του πίνακα, για τα οποία ο χρήστης δεν έχει τη δυνατότητα τροποποίησης του περιεχομένου τους κατά την ανανέωση μιας εγγραφής, αλλά ούτε και δυνατότητα εισαγωγής περιεχομένου κατά τη διαδικασία προσθήκης νέας εγγραφής.

Πως παίρνουν τιμές τα στοιχεία αυτά:

Για την πλήρη αυτοματοποίηση του συστήματος και κυρίως για τη διευκόλυνση και εξοικονόμηση χρόνου από το χρήστη, προστέθηκε το κουμπί δίπλα από το textbox με ένδειξη **«Αριθμός Τιμολογίου».** Πατώντας το κουμπί αυτό ο χρήστης, μεταφέρεται αυτόματα στο «tab Transaction», από το οποίο μπορεί να επιλέξει το τιμολόγιο για το οποίο αναφέρεται η πληρωμή/απόδειξη. Όταν ο χρήστης επιλέξει το τιμολόγιο μεταφέρεται πάλι πίσω στο «tab Payments» και τα textbox με ενδείξεις **«Αριθμός Τιμολογίου», «Όνομα»** (πελάτη ή προμηθευτή) και **«Υπόλοιπο»**(τιμολογίου) ενημερώνονται αυτόματα (βλ. Παράρτημα-Διαδικασία 19-20).

Επιπλέον κατά την προσθήκη μιας νέας πληρωμής, μέσα στο textbox με ένδειξη **«Αριθμός Απόδειξης»** εμφανίζεται μήνυμα **«ΝΕΑ ΣΥΝΑΛΛΑΓΗ»** και όταν

ο χρήστης πατήσει το κουμπί **«Αποθήκευση»**, δημιουργείται αυτόματα από το σύστημα ένας αριθμός απόδειξης**.**

Επίσης για λόγους ευκολίας, τοποθετήθηκε ένα DateTimePicker, από το οποίο ο χρήστης μπορεί να επιλέξει την ημερομηνία εγγράφου χωρίς να χρειαστεί να την γράψει ο ίδιος.

Στο textbox με ένδειξη **«Ποσό»,** ο χρήστης μπορεί να εισάγει το ποσό πληρωμής και στη συνέχεια από ένα «combobox» να επιλέξει την μέθοδο πληρωμής (βλ. Παράρτημα-Διαδικασία 21). Αν η μέθοδος πληρωμής που επιλέχθηκε είναι **«Επιταγή»,** τότε εμφανίζεται ένα κρυφό textbox στο οποίο ο χρήστης μπορεί να εισάγει τον αριθμό επιταγής, και ένα κρυφό combobox από το οποίο μπορεί να επιλέξει την τράπεζα που αναμιγνύεται στη συναλλαγή αυτή.

Δεξιά από το DataGridView, τοποθετήθηκαν τέσσερα **Panel** με διαφορετικά χρώματα, και σε όλα ορίστηκε το **Click Event**. Όταν ο χρήστης κάνει «rollover» πάνω από οποιοδήποτε Panel, ενημερώνεται με **ToolTipText** για τη χρήση του καθενός από αυτά, ενώ όταν κάνει κλικ σε κάποιο από τα Panel γίνονται τα εξής:

- Σκούρο μπλε Panel - εμφανίζονται όλες οι πληρωμές στο DataGridView
- Πράσινο Panel - εμφανίζονται μόνο οι πληρωμές σε μετρητά
- Ελαφρύ Μπλε Panel - εμφανίζονται μόνο οι πληρωμές με επιταγή
- Κόκκινο Panel - Εμφανίζονται οι πληρωμές που ακυρώθηκαν

Στο αριστερό μέρος του tab, και κάτω από το DataGridView, τοποθετήθηκε το κουμπί **«Εκτύπωση»**. Δίνεται έτσι η δυνατότητα στο χρήστη, να επιλέξει από το DataGridView οποιαδήποτε Πληρωμή/Απόδειξη και να την εκτυπώσει. Επίσης κατά την αποθήκευση ή ανανέωση μιας πληρωμής, εμφανίζεται κατάλληλο μήνυμα στο χρήστη με δυνατότητα εκτύπωσης.

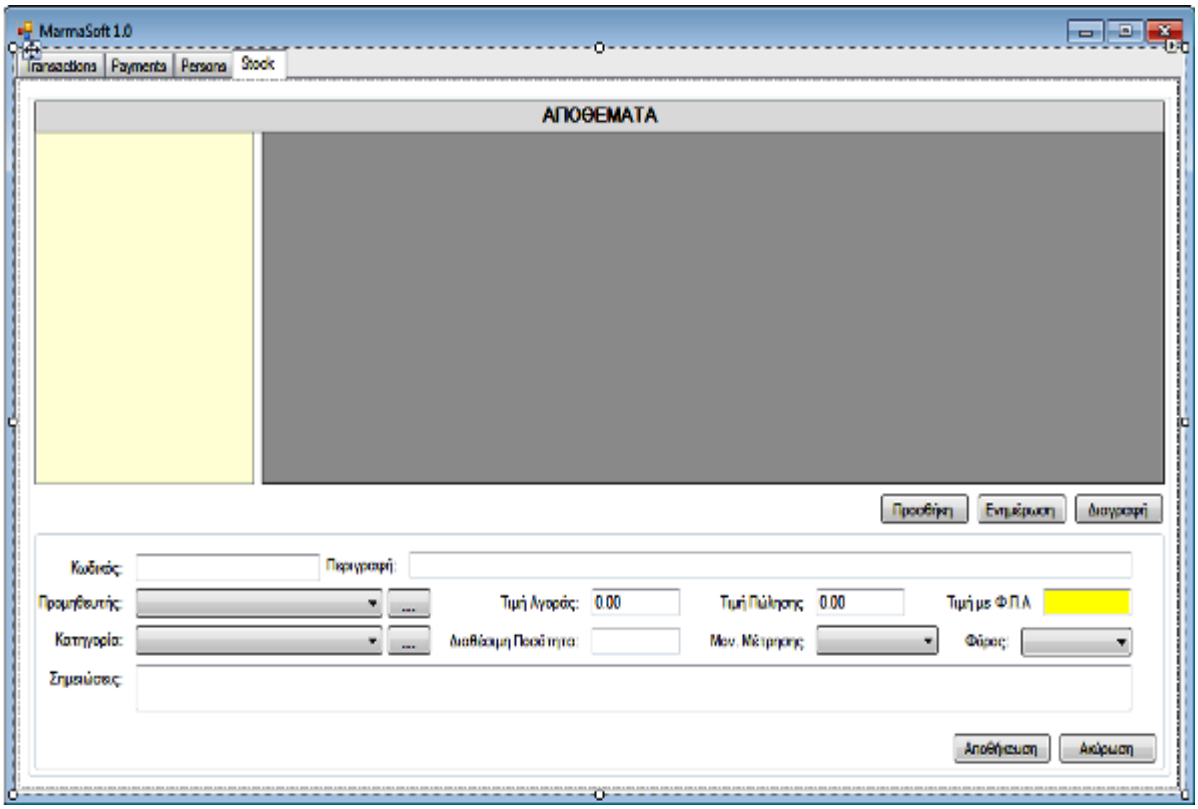
Τέλος, ακόμη μια σημαντική υπηρεσία που προσφέρεται είναι η εξής:

Κατά την αποθήκευση μια νέας πληρωμής ή κατά την ανανέωση μιας πληρωμής, η οποία ήδη υπάρχει στη βάση, ενημερώνεται αυτόματα και ο πίνακας Transaction για το νέο υπόλοιπο, καθώς επίσης και για την κατάσταση του συγκεκριμένου τιμολογίου που προέκυψε (Ανοιχτό ή Εξοφλημένο - βλ. Παράρτημα-Διαδικασία 22).

Επίσης τα textbox με ενδείξεις **«Transaction Code»** και **«Payment Type»** είναι κρυφά και χρησιμοποιούνται για προγραμματιστικούς λόγους (βλ. Παράρτημα-Διαδικασία 19).

Tab Stock:

Το tab Stock δημιουργήθηκε για την καταμέτρηση και διαχείριση όλων των αποθεμάτων της εταιρείας και φαίνεται πιο κάτω:



Εικόνα 39: «Tab Stock»

Ιδιαίτερα χαρακτηριστικά του tab Stock:

Όπως φαίνεται και από την εικόνα στο «tab Stock», ο χρήστης μπορεί να δώσει ένα κωδικό για το νέο προϊόν, την περιγραφή του, την τιμή αγοράς, τη διαθέσιμη ποσότητα και την τιμή πώλησης.

Για λόγους αυτοματοποίησης, εύκολης διαχείρισης και εξοικονόμησης χρόνου από τον χρήστη έγιναν τα εξής:

- Μόλις δοθεί η τιμή πώλησης του προϊόντος, υπολογίζεται αυτόματα από το σύστημα η τιμή πώλησης με Φ.Π.Α.

- Επίσης τοποθετήθηκαν τέσσερα combobox (Προμηθευτής, Κατηγορία, Μον. Μέτρησης, Φόρος), στα οποία κατά τη διάρκεια που το tab Stock κάνει Load(), φορτώνονται τα ανάλογα δεδομένα. Έτσι ο χρήστης δεν χρειάζεται να γράφει αυτά τα στοιχεία κάθε φορά, απλά μπορεί να τα επιλέγει από τα εν λόγω combobox.

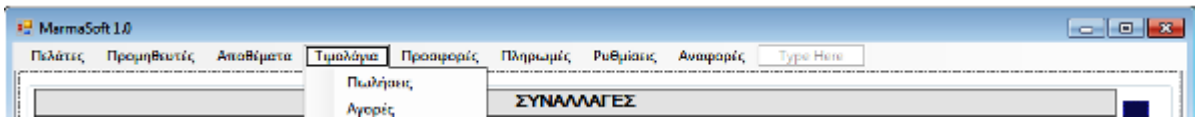
Επειδή το Φ.Π.Α δεν είναι το ίδιο για όλα τα προϊόντα, ο χρήστης από το combobox με ένδειξη **«Φόρος»** μπορεί να επιλέξει αν το συγκεκριμένο προϊόν έχει φόρο η όχι, και αν ναι το ποσοστό του Φ.Π.Α.

- Αριστερά από το DataGridView, τοποθετήθηκε ένα **TreeView**, στο οποίο εμφανίζονται όλες οι κατηγορίες αποθεμάτων σε μορφή δέντρου (βλ. Παράρτημα-Διαδικασία 23). Ο χρήστης μπορεί να επιλέξει οποιαδήποτε κατηγορία από αυτές και αμέσως θα εμφανιστούν στο DataGridView μόνο τα προϊόντα που ανήκουν στην επιλεχθείσα κατηγορία. Επιπρόσθετα, στη συνέχεια μπορεί να επανεμφανίσει όλα τα προϊόντα απ' όλες τις κατηγορίες.

Tab Transactions:

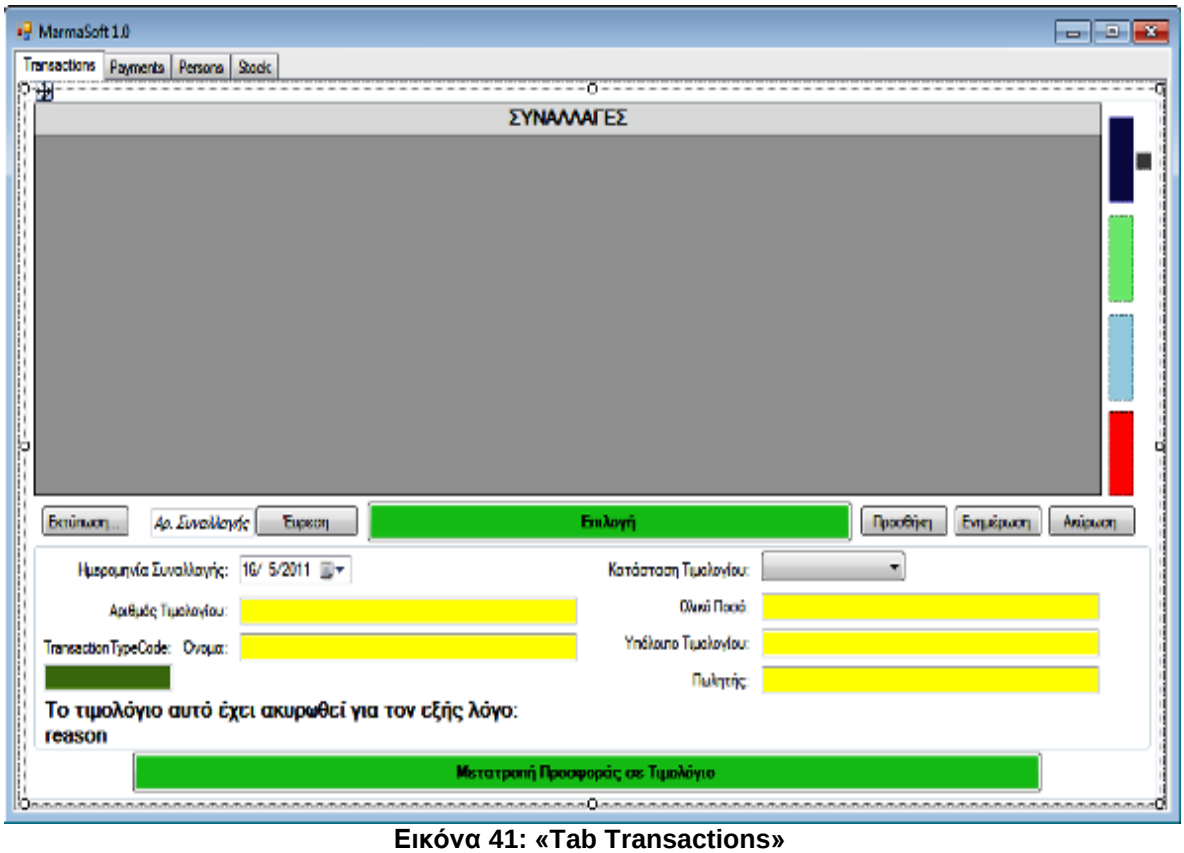
Όπως έχει ήδη αναφερθεί, υπάρχουν δύο τύποι συναλλαγών (Τιμολόγια και Προσφορές). Επίσης τα τιμολόγια χωρίζονται σε δύο τύπους:

- Τιμολόγια από πωλήσεις της εταιρείας
- Τιμολόγια από αγορές της εταιρείας



Εικόνα 40: «Item Τιμολόγια»

Οποιοδήποτε Item και να επιλέξει ο χρήστης από τα πιο πάνω, οδηγείται στο **tab Transactions,** στο DataGridView του οποίου εμφανίζονται τα ανάλογα δεδομένα. Το tab φαίνεται στην επόμενη εικόνα :



Εικόνα 41: «Tab Transactions»

Ιδιαίτερα χαρακτηριστικά του tab Transactions:

Όπως φαίνεται και από την εικόνα, ο χρήστης, όπως και στις πληρωμές, μπορεί να επιλέξει μια εγγραφή από το DataGridView και να πατήσει **«Εκτύπωση».**

Κουμπί «Επιλογή»:

Το κουμπί **Επιλογή** είναι κρυφό και η λειτουργία του εξηγήθηκε στο tab Payments.

Κουμπί «Μετατροπή Προσφοράς σε Τιμολόγιο»:

Το κουμπί αυτό, είναι επίσης κρυφό και γίνεται ορατό, μόνο στην περίπτωση που ο χρήστης επέλεξε το **«Item Προφορές»** για να μεταφερθεί στο **tab Transactions**. Έτσι, του δίνεται η δυνατότητα, σε περίπτωση που μια προσφορά συμφωνήθηκε με τον πελάτη, να μπορεί να την μετατρέψει σε τιμολόγιο ώστε να μην χρειαστεί να ξαναγράψει τις ίδιες πληροφορίες.

Επιπλέον τοποθετήθηκε ένα textbox, στο «text» του οποίου αναγράφεται **«Αρ. Συναλλαγής»** και δίπλα του ένα κουμπί με ένδειξη **«Εύρεση»,** το οποίο είναι μη διαθέσιμο. Μόλις ο χρήστης πληκτρολογήσει έναν αριθμό συναλλαγής, το κουμπί «Εύρεση» γίνεται διαθέσιμο και του δίνεται η δυνατότητα για αναζήτηση της συναλλαγής. Αν ο αριθμός συναλλαγής υπάρχει, τότε εμφανίζεται στο DataGridView μόνο η συγκεκριμένη συναλλαγή, το textbox γίνεται μη διαθέσιμο και η ονομασία του κουμπιού από **«Εύρεση»** γίνεται **«Προβολή όλων».** Έτσι ο χρήστης μπορεί και πάλι να επαναφέρει στο DataGridView όλες τις συναλλαγές. Σε περίπτωση που ο «Αρ. Συναλλαγής» δεν υπάρχει, εμφανίζεται κατάλληλο μήνυμα για ενημέρωση του χρήστη (βλ. Παράρτημα-Διαδικασία 24).

Δεξιά από το DataGridView, τοποθετήθηκαν τέσσερα **Panel** με διαφορετικά χρώματα και σε όλα ορίστηκε το **Click Event**. Όταν ο χρήστης κάνει rollover πάνω από οποιοδήποτε Panel, ενημερώνεται με ToolTipText για τη χρήση του καθενός από αυτά, ενώ όταν κάνει κλικ σε κάποιο από τα Panel γίνονται τα εξής:

- Σκούρο μπλε Panel - εμφανίζονται όλα τα τιμολόγια ή προσφορές.
- Πράσινο Panel - εμφανίζονται μόνο οι εκκρεμείς συναλλαγές (βλ. Παράρτημα-Διαδικασία 25).
- Ελαφρύ Μπλε Panel - εμφανίζονται μόνο τα εξοφλημένα τιμολόγια.
- Κόκκινο Panel - Εμφανίζονται τα τιμολόγια που ακυρώθηκαν.

Σε αντίθεση με τα υπόλοιπα tabs:

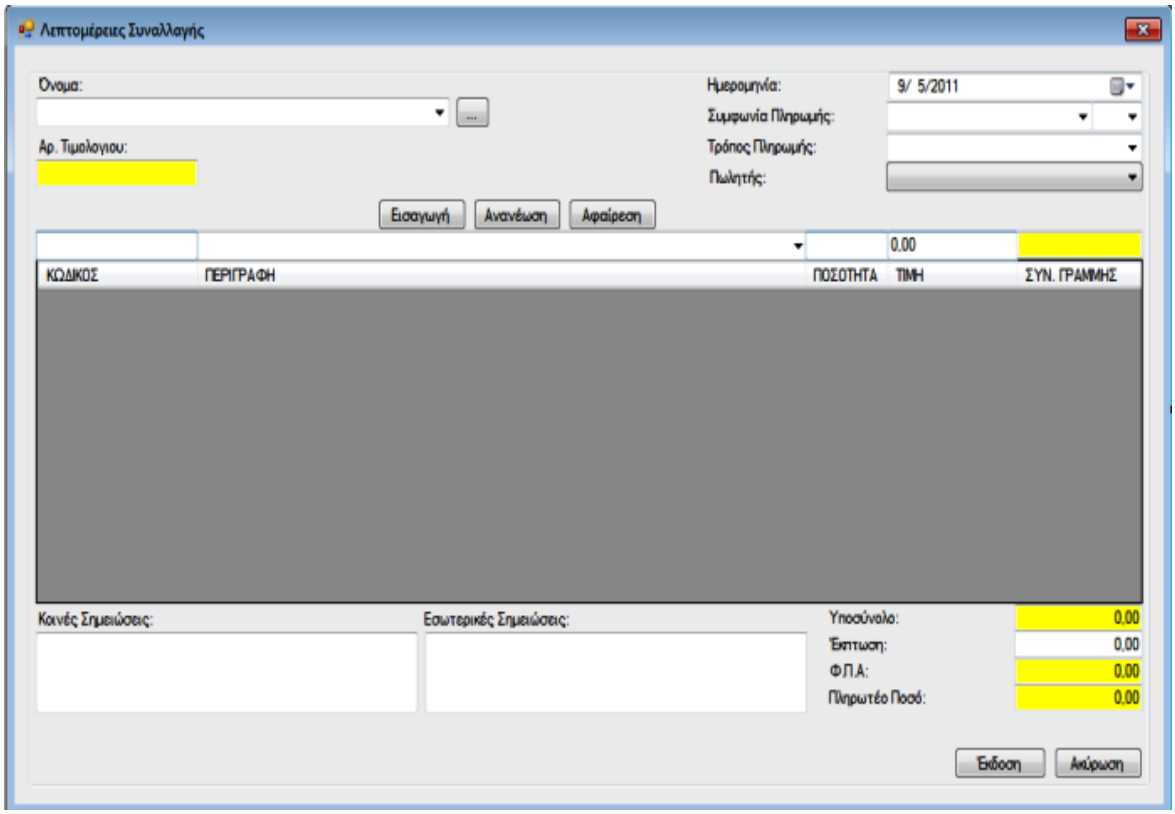
- Το **grpDetails** δε γίνεται ποτέ διαθέσιμο, απλά τοποθετήθηκε για να μπορεί ο χρήστης να βλέπει μερικές από τις λεπτομέρειες μιας συναλλαγής. Μπορεί να οδηγηθεί στη φόρμα **frmTransactionDetails** πατώντας **«διπλό κλικ»** σε μια εγγραφή και να δει όλα τα στοιχεία της.
- Όταν πατηθεί το κουμπί **«Προσθήκη»** ο χρήστης οδηγείται στη φόρμα «frmTransactionDetails». Στη φόρμα αυτή μπορεί να εισάγει όλα τα στοιχεία ενός νέου τιμολογίου ή προσφοράς.
- Όταν επιλεχθεί μια συναλλαγή και πατηθεί το κουμπί **«Ενημέρωση»**, ο χρήστης οδηγείται στη φόρμα «frmTransactionDetails» από την οποία μπορεί να κάνει οποιαδήποτε αλλαγή όσον αφορά τη συγκεκριμένη συναλλαγή.

- Όταν επιλεχθεί μια συναλλαγή και πατηθεί το κουμπί **«Ακύρωση»**, εμφανίζεται ένα **InputDialog** στο οποίο ο χρήστης μπορεί να γράψει το λόγο που οδήγησε στην ακύρωση της συναλλαγής.

Τέλος, σε labels, στο κάτω αριστερά μέρος του tab αυτού, τοποθετήθηκαν τα **κρυφά μηνύματα «Το τιμολόγιο αυτό έχει ακυρωθεί για τον εξής λόγο:»** και **«reason»**. Έτσι, σε περίπτωση που ο χρήστης επιλέξει μια συναλλαγή η οποία είναι **άκυρη,** θα εμφανίζονται τα μηνύματα αυτά και στο label **«reason»** θα αναγράφεται ο λόγος που οδήγησε στην ακύρωση της συναλλαγής. Απώτερος σκοπός είναι η ενημέρωση του χρήστη.

frmTransactionDetails (Λεπτομέρειες Συναλλαγής):

Ουσιαστικά, η όλη διαχείριση ενός τιμολογίου ή μιας προσφοράς γίνεται από αυτήν τη φόρμα. Η φόρμα παρουσιάζεται πιο κάτω:



Εικόνα 42: «Φόρμα frmTransactionDetails»

Ανάλογα με το είδος της συναλλαγής, στη φόρμα αυτή παρουσιάζονται μερικές τροποποιήσεις.

Όταν το είδος συναλλαγής είναι **«Τιμολόγιο πώλησης»** ή **«Προσφορά»,** στο combobox κάτω από το label «Όνομα», φορτώνονται όλα τα ονόματα των Πελατών της εταιρείας, δίνοντας στο χρήστη τη δυνατότητα επιλογής του ονόματος του πελάτη στον οποίο αναφέρεται η συναλλαγή. Σε αντίθετη περίπτωση, όπου το είδος της συναλλαγής είναι **«Τιμολόγιο Αγοράς»** στο combobox αυτό φορτώνονται τα ονόματα όλων των προμηθευτών.

Ο χρήστης μπορεί επίσης να επιλέξει το όνομα του Πελάτη ή του Προμηθευτή, πατώντας στο κουμπί που βρίσκεται δίπλα από το combobox. Σκοπός του κουμπιού αυτού είναι, σε περίπτωση που ο χρήστης δεν είναι σίγουρος για το όνομα του ατόμου που συναλλάσσεται, να μπορεί να οδηγηθεί στο **«tab Persons».** Από το tab αυτό μπορεί να δει περισσότερα στοιχεία για τα άτομα που συναλλάσσονται με την εταιρεία, ώστε να είναι βέβαιος για την επιλογή του.

Επίσης κατά την προσθήκη ενός νέου «Τιμολογίου Πώλησης» ή «Προσφοράς», στο textbox με ένδειξη «Αρ. Τιμολογίου» ή «Αρ. Προσφοράς» αντίστοιχα, εμφανίζεται το μήνυμα **«ΝΕΑ ΠΩΛΗΣΗ»** ή **«ΝΕΑ ΠΡΟΣΦΟΡΑ»,** ανάλογα με το είδος της συναλλαγής και έχει κίτρινο χρώμα. Αυτό συμβαίνει γιατί δεν χρειάζεται ο χρήστης να δώσει τον αριθμό συναλλαγής, επειδή δίνεται αυτόματα από το σύστημα κατά την αποθήκευση της συναλλαγής (βλ. Παράρτημα-Διαδικασία 26).

Αντίθετα, σε περίπτωση «Τιμολογίου Αγοράς», στο textbox δεν εμφανίζεται κάποιο μήνυμα και είναι λευκό (Enabled True). Έτσι, ο χρήστης μπορεί να γράψει τον αριθμό τιμολογίου, που αναγράφεται στο τιμολόγιο που πήρε η εταιρεία από τον Προμηθευτή.

Στη συνέχεια, από ένα DataTimePicker μπορεί να επιλέξει την ημερομηνία εγγράφου, και ακολούθως από τα combobox, τα οποία βρίσκονται κάτω από αυτό να επιλέξει τη συμφωνία πληρωμής και τον τρόπο πληρωμής που συμφωνήθηκε.

Στο combobox με ένδειξη **«Συμφωνία Πληρωμής»,** υπάρχουν τρία είδη συμφωνίας για επιλογή:

- Εξοφλημένο (σε περίπτωση που ήδη εξοφλήθηκε)
- Πληρωμή κατά την παράδοση
- Πληρωμή σε μέρες

Σε περίπτωση επιλογής της συμφωνίας **«Πληρωμή σε μέρες»,** το combobox που βρίσκεται δεξιά από αυτό, γίνεται διαθέσιμο και ο χρήστης μπορεί να επιλέξει από εκεί τον αριθμό των ημερών που συμφωνήθηκαν.

Επίσης σε περίπτωση που η συμφωνία Πληρωμής είναι **«Εξοφλημένο»**, κατά την αποθήκευση του τιμολογίου δημιουργείται αυτόματα από το σύστημα η αντίστοιχη Πληρωμή/Απόδειξη.

Αντίθετα, σε περίπτωση που η συναλλαγή είναι **«Προσφορά»** τα labels **«Συμφωνία Πληρωμής»** και **«Τρόπος Πληρωμής»** μαζί με τα αντίστοιχα textbox είναι μη ορατά για προφανείς λόγους.

Ακολούθως ο χρήστης μπορεί να προχωρήσει στην διαδικασία εισαγωγής των προϊόντων της συναλλαγής. Στο σημείο αυτό για την υποστήριξη της πιο πάνω διαδικασίας έγιναν τα εξής:

Από ένα combobox στο οποίο βρίσκονται όλα τα αποθέματα της εταιρείας, ο χρήστης μπορεί να επιλέξει το επιθυμητό προϊόν. Αποτέλεσμα της διαδικασίας αυτής είναι να εμφανιστεί στο textbox με ένδειξη **«ΚΩΔΙΚΟΣ»,** ο κωδικός του προϊόντος και στο textbox με ένδειξη **«ΤΙΜΗ»,** η τιμή πώλησης του προϊόντος αυτού (αν αυτά υπάρχουν στη βάση του συστήματος). Στο textbox με ένδειξη **«ΠΟΣΟΤΗΤΑ»** υπάρχει πάντοτε by default η τιμή «1», ενώ όταν ο χρήστης αλλάξει την τιμή αυτή, υπολογίζεται αυτόματα από το σύστημα η συνολική τιμή προϊόντος, βάση της ποσότητας του, και μπαίνει στο textbox με ένδειξη **«ΣΥΝ. ΓΡΑΜΜΗΣ»** το οποίο είναι κίτρινο για τους λόγους που προαναφέρθηκαν.

Ακολούθως, ο χρήστης μπορεί να πατήσει το κουμπί **«Εισαγωγή»,** για να εισαχθεί η συγκεκριμένη εγγραφή στο DataGridView που βρίσκεται πιο κάτω από αυτά. Σε αντίθετη περίπτωση, όπου το προϊόν δεν υπάρχει στη βάση, ο χρήστης μπορεί να γράψει την ονομασία του στο textbox με ένδειξη **«ΠΕΡΙΓΡΑΦΗ»,** να δώσει ένα μοναδικό κωδικό για το προϊόν αυτό, καθώς και την τιμή πώλησης του. Μάλιστα, κατά την έκδοση/αποθήκευση του τιμολογίου, του δίνεται η δυνατότητα να αποθηκεύσει το προϊόν αυτό στη βάση και συγκεκριμένα στο πίνακα Stock.

Ο χρήστης μπορεί να επιλέξει ένα από τα προϊόντα που ήδη έχει εισάγει στο DataGridView και να πατήσει το κουμπί **«Ανανέωση»**. Αμέσως τα στοιχεία της εγγραφής αυτής μεταφέρονται στα textbox πάνω από το DataGridView, όπου ο χρήστης μπορεί να κάνει τις αλλαγές που επιθυμεί και να πατήσει ξανά **«Εισαγωγή».**

Επίσης, μπορεί να επιλέξει μια από τις εγγραφές που έχει εισάγει και να πατήσει το κουμπί **«Διαγραφή»,** ώστε να αφαιρεθεί η εγγραφή αυτή από το DataGridView.

Ενώ ο χρήστης εισάγει ή αφαιρεί προϊόντα στο DataGridView, για λόγους αυτοματοποίησης και εξοικονόμησης χρόνου και με τη βοήθεια των **Event «RowsAdded»** και **«RowsRemoved»** αντίστοιχα, γίνονται τα εξής:

- Στο textbox με ένδειξη **«Υποσύνολο»,** υπολογίζεται αυτόματα το άθροισμα των τιμών όλων των προϊόντων που έχουν εισαχθεί στο DataGridView.
- Στο textbox με ένδειξη **«Φ.Π.Α»**, υπολογίζεται αυτόματα ο φόρος προστιθέμενης αξίας για όλα τα προϊόντα που εισήχθησαν.
- Στο textbox με ένδειξη **«Πληρωτέο Ποσό»**, υπολογίζεται το ακριβές ποσό που πρέπει να πληρωθεί με βάση τα πιο πάνω (βλ. Παράρτημα-Διαδικασία 27).

Τέλος, στο textbox με ένδειξη **«Έκπτωση»**, ο χρήστης έχει τη δυνατότητα εισαγωγής ενός ποσού έκπτωσης και με τη βοήθεια του **TextChanged Event,** καλείται και πάλι η μέθοδος για τον υπολογισμό των πιο πάνω.

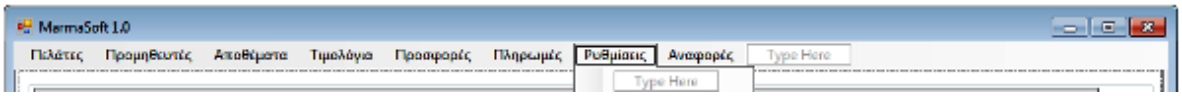
Στο κάτω αριστερά μέρος της φόρμας, τοποθετήθηκαν τα textbox με ενδείξεις **«Κοινές Σημειώσεις»** και **«Εξωτερικές Σημειώσεις»**. Με αυτά παρέχεται η δυνατότητα στο χρήστη να κρατήσει σημειώσεις για κάτι σημαντικό που σχετίζεται με την συναλλαγή αυτή. Οι «Εσωτερικές Σημειώσεις» είναι ορατές μόνο στο χρήστη/χρήστες του συστήματος, ενώ οι «Κοινές Σημειώσεις» γίνονται γνωστές σε όλα τα συναλλασσόμενα πρόσωπα, καθώς κατά την εκτύπωση του τιμολογίου ή της προσφοράς τυπώνονται και αυτές στο έγγραφο.
Αν ο χρήστης επιλέξει **«Ακύρωση»**, ακυρώνονται όλες οι αλλαγές και η φόρμα αυτή κλείνει.

Επίσης, όταν η συναλλαγή αφορά **«Τιμολόγιο Αγοράς»,** ο πίνακας Stock ανανεώνεται αυτόματα, προσθέτοντας τα νέα προϊόντα, ενώ όταν η συναλλαγή αφορά **«Τιμολόγιο Πώλησης»** ο πίνακας Stock ανανεώνεται αφαιρώντας τα προϊόντα που έχουν πωληθεί (βλ. Παράρτημα-Διαδικασία 28).

Τέλος, κατά την έκδοση/αποθήκευση της συναλλαγής, δίνεται η δυνατότητα στο χρήστη για εκτύπωση του εγγράφου.

frmSettings (Ρυθμίσεις):

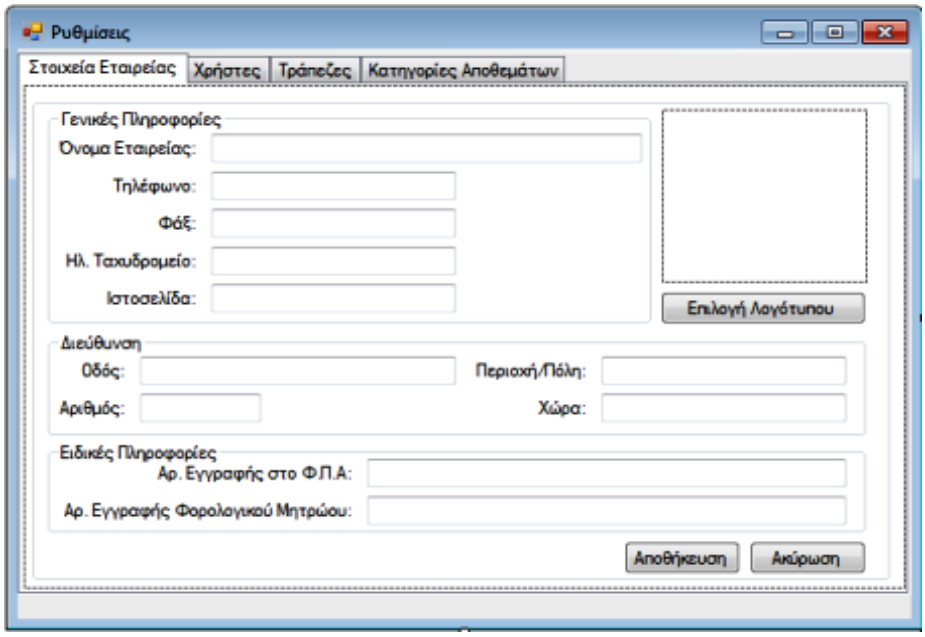
Μέσω του **«Item Εργαλεία»,** παρέχεται στο χρήστη η δυνατότητα διαχείρισης των εσωτερικών θεμάτων της εταιρείας.
Αν ο χρήστης λοιπόν, επιλέξει το **«Item Ρυθμίσεις»**:



Εικόνα 43: «Item Εργαλεία»

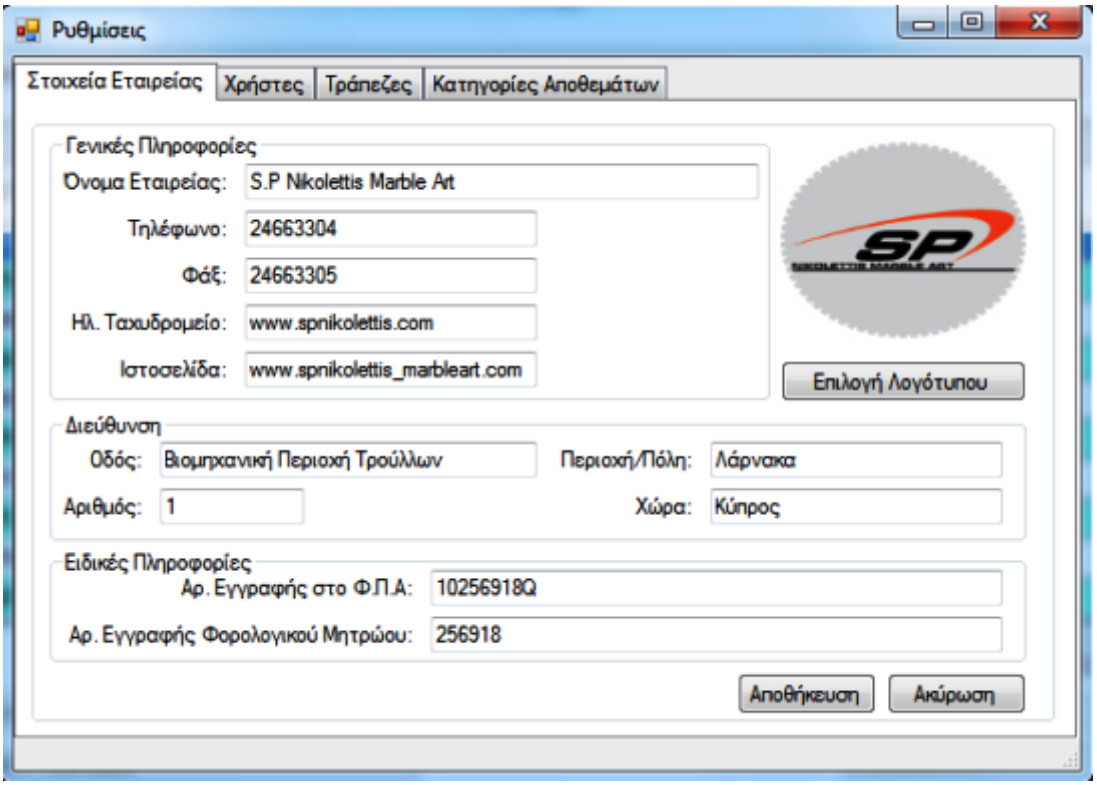
εμφανίζεται η **φόρμα Ρυθμίσεων** από την οποία ο χρήστης μπορεί να διαχειριστεί:

- Τα Στοιχεία της Εταιρείας
- Τους χρήστες της εφαρμογής
- Τις Τράπεζες
- Τις Κατηγορίες Αποθεμάτων



Εικόνα 44: «Φόρμα frmSettings»

Tab «Στοιχεία Εταιρείας»:



Εικόνα 45: « Tab Στοιχεία Εταιρείας»

Στο tab αυτό, τοποθετήθηκαν textbox τα οποία αντιστοιχούν στα πεδία του πίνακα **«CompanyInfo»** και έχουν σαν περιεχόμενο τα τρέχοντα στοιχεία της εταιρείας. Ο χρήστης μπορεί να αλλάξει οποιοδήποτε από αυτά και να πατήσει **«Αποθήκευση»**, είτε να πατήσει **«Ακύρωση»** για να επιστρέψει στην κύρια φόρμα της εφαρμογής, χωρίς να αποθηκευτούν οι αλλαγές.

Ιδιαίτερα χαρακτηριστικά:

Τα στοιχεία της εταιρείας χωρίστηκαν σε τρία Groupbox, όπως φαίνεται και από την πιο πάνω εικόνα:

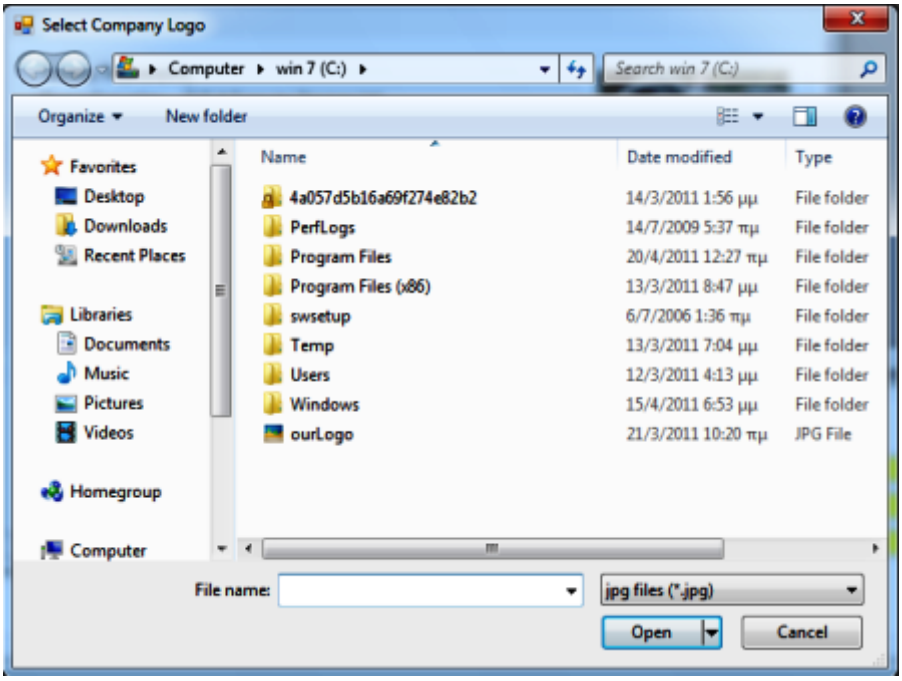
- Γενικές Πληροφορίες
- Διεύθυνση
- Ειδικές Πληροφορίες

Επίσης τοποθετήθηκε ένα PictureBox, στο οποίο ο χρήστης μπορεί να εισάγει το λογότυπο της εταιρείας με τη βοήθεια του κουμπιού **«Επιλογή Λογότυπου»,** που

βρίσκεται κάτω από αυτό, και πιο συγκεκριμένα με τη βοήθεια του εργαλείου **OpenFileDialog.**

Κουμπί «Επιλογή Λογότυπου»:

Με το πάτημα του κουμπιού αυτού, δημιουργείται ένα Instance του εργαλείου **OpenFileDialog** και εμφανίζεται στο χρήστη το πιο κάτω παράθυρο:

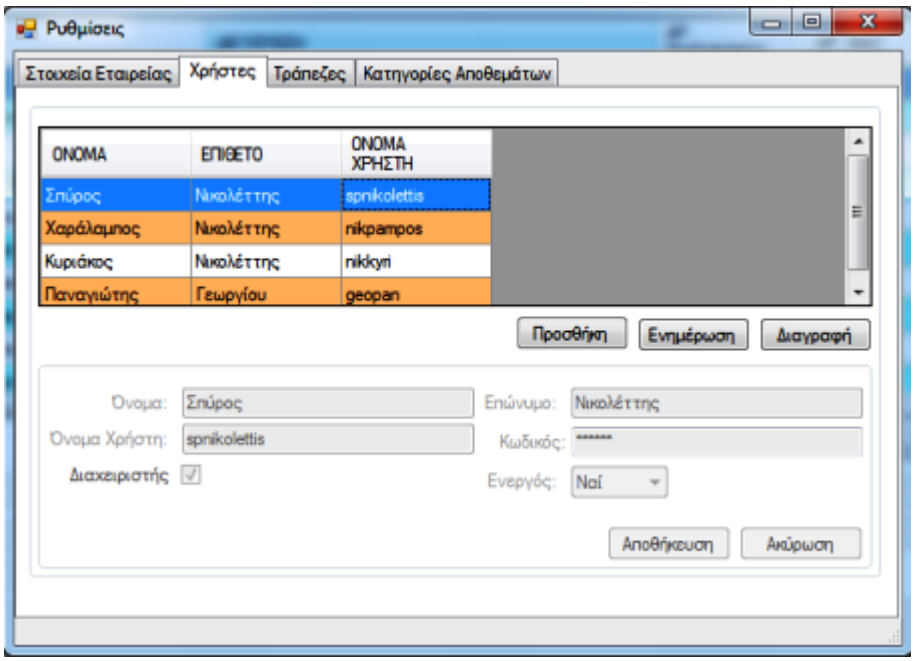


Εικόνα 46: «Select Company Logo»

Από το παράθυρο αυτό, ο χρήστης μπορεί να πλοηγηθεί στα αρχεία του υπολογιστή και να επιλέξει το λογότυπο της εταιρείας. Η διαδρομή του λογότυπου που επιλέχθηκε, αποθηκεύεται στο πίνακα **CompanyInfo** για περαιτέρω χρήση του κατά την έκδοση εγγράφων.

Tab «Χρήστες»:

Το tab αυτό, δημιουργήθηκε για την πλήρη διαχείριση των χρηστών του συστήματος, και έχουν πρόσβαση σε αυτό μόνο οι χρήστες με δικαιώματα Administrator. Κατά την εισαγωγή του χρήστη στο σύστημα, γίνεται έλεγχος και αν είναι Administrator, η επιλογή για τη μεταφορά του στο tab αυτό είναι διαθέσιμη, αλλιώς είναι μη διαθέσιμη (Enabled false).



Εικόνα 47: «Tab Χρήστες»

Εδώ, όπως και στα tabs της κύριας φόρμας, τοποθετήθηκε ένα DataGridView, στο οποίο φορτώνονται όλοι οι χρήστες του πίνακα Users. Τοποθετήθηκαν επίσης τα κουμπιά **«Προσθήκη», «Ενημέρωση»** και **«Διαγραφή»**, τα οποία έχουν ακριβώς την ίδια λειτουργικότητα με τα αντίστοιχα κουμπιά που περιγράφηκαν στα υπόλοιπα tabs.

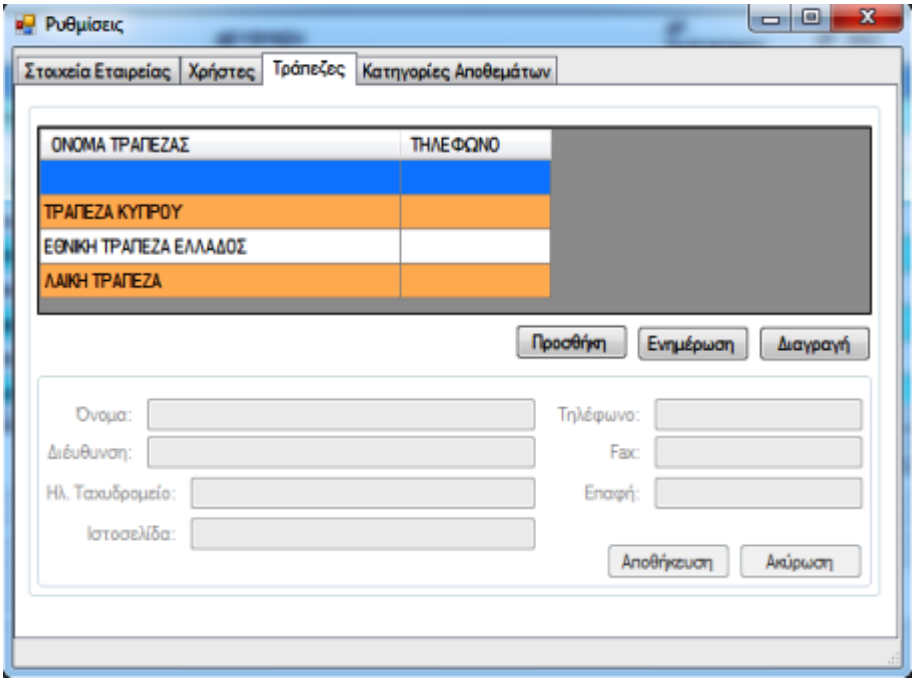
Τέλος σε ένα groupbox, τοποθετήθηκαν τα textbox με τις ανάλογες ενδείξεις, τα οποία αντιστοιχούν στα πεδία του πίνακα Users. Ο χρήστης μπορεί να εισάγει σε αυτά τα στοιχεία ενός νέου χρήστη ή να ενημερώσει τα στοιχεία ενός υπάρχοντος χρήστη. Επίσης τοποθετήθηκε ένα checkbox (Διαχειριστής) και ένα combobox (Ενεργός), από τα οποία ο Administrator μπορεί να επιλέξει, εάν κάποιος χρήστης είναι «Διαχειριστής» ή «Ενεργός» αντίστοιχα. Με το κουμπί «Ακύρωση» η φόρμα κλείνει και ο χρήστης οδηγείται στην κεντρική φόρμα διαχείρισης.

Tab «Τράπεζες»:

Σκοπός του tab αυτού, είναι η διαχείριση του πίνακα Bank στον οποίο συντηρούνται πληροφορίες για τις τράπεζες που αναμιγνύονται στις διάφορες συναλλαγές της εταιρείας. Η λειτουργικότητα των κουμπιών που φαίνονται στην

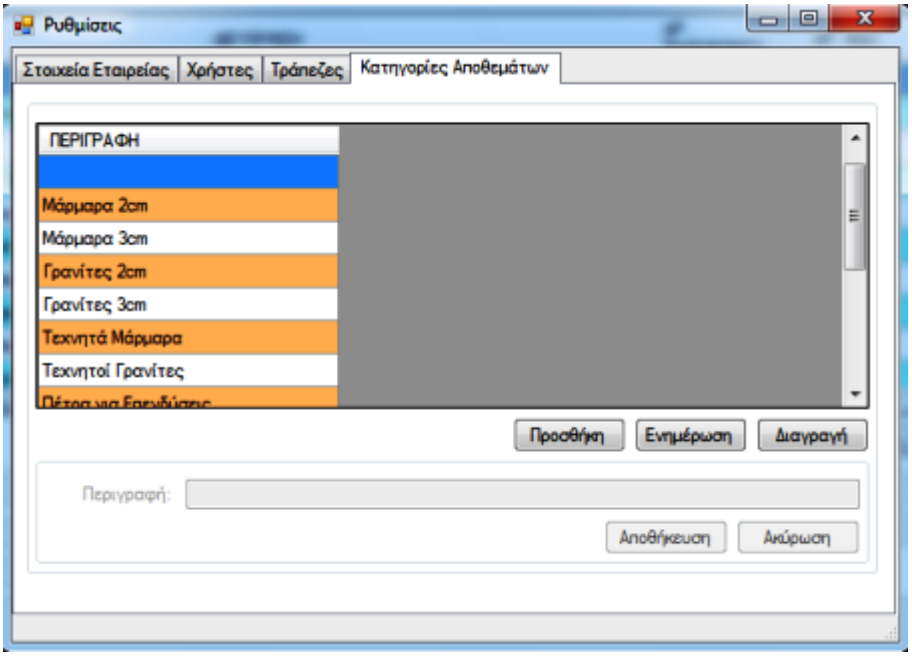
εικόνα, είναι ακριβώς η ίδια με αυτή που περιγράφηκε για τα αντίστοιχα κουμπιά άλλων φορμών.

Το **tab Τράπεζες** φαίνεται στην πιο κάτω εικόνα:



Εικόνα 48: «Tab Τράπεζες»

Tab «Κατηγορίες Αποθεμάτων»:



Εικόνα 49: «Tab Κατηγορίες Αποθεμάτων»

Στο tab αυτό τοποθετήθηκε ένα textbox, στο οποίο ο χρήστης μπορεί να εισάγει την περιγραφή μιας νέας ομάδας αποθεμάτων. Η χρήση των κατηγοριών αυτών περιγράφηκε στο «tab Stock».

Φόρμες αναφορών

frmTransactionReport και frmPaymentReport:

Οι φόρμες **«frmTransactionReport»** και **«frmPaymentReport»,** δημιουργήθηκαν για την εξαγωγή και την προεπισκόπιση αναφορών και κατ' επέκταση την εκτύπωση τους, όσον αφορά τις συναλλαγές (τιμολόγια πώλησης, προσφορές) και τις πληρωμές (από πελάτες) της εταιρείας αντίστοιχα.

Για την επίτευξη των πιο πάνω, στις φόρμες τοποθετήθηκε το **object «CrystalReportViewer»** και χρησιμοποιήθηκε το πρόγραμμα **«Crystal Reports 2008».** Με το πρόγραμμα αυτό, δημιουργήθηκαν δύο πρότυπα εγγράφων, που θα υποστηρίζουν τους πιο πάνω τύπους αναφοράς, όσον αφορά τα δεδομένα (πεδία) και τον τρόπο που αυτά θα εμφανίζονται στην εκάστοτε φόρμα αναφοράς.

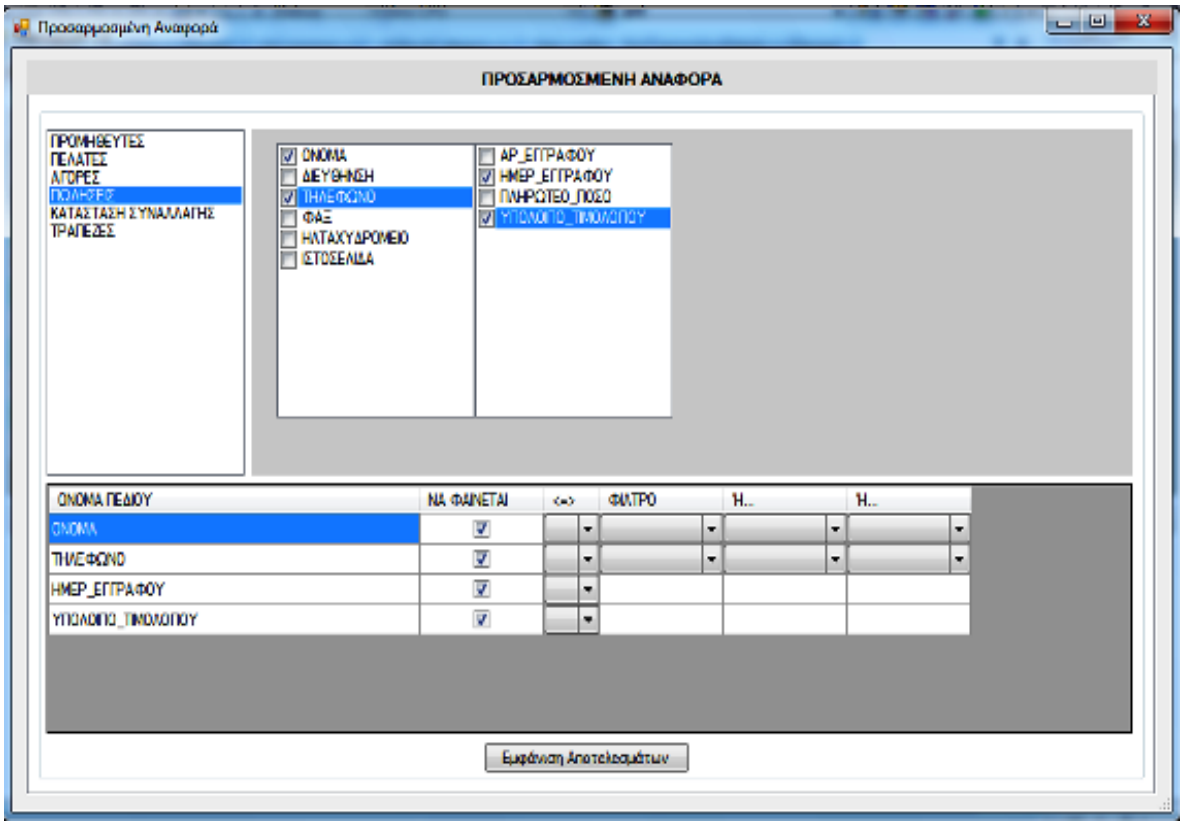
Τέλος, τα πρότυπα αυτά, τοποθετήθηκαν σε υποφάκελο, στο φάκελο «Debug» της εφαρμογής, απ' όπου τα «διαβάζει» το σύστημα και τα εμφανίζει στις αντίστοιχες φόρμες, με τα ανάλογα πάντα δεδομένα.

frmCustomReportGenerator (Προσαρμοσμένη αναφορά):

Σκοπός της φόρμας αυτής, είναι να παρέχεται στο χρήστη η δυνατότητα για δημιουργία δυναμικής αναφοράς. Να μπορεί να επιλέξει δηλαδή διάφορα δεδομένα από τη βάση (πεδία) και σε συνδυασμό με διάφορους περιορισμούς που μπορεί να θέσει να εμφανίσει μια αναφορά με τα δεδομένα αυτά.

Η φόρμα αποτελείται από δύο tab. Στο πρώτο tab, τοποθετήθηκε αριστερά ένα **listbox** με προκαθορισμένες τιμές, οι οποίες αντιστοιχούν στους βασικότερους πίνακες της βάσης. Ο χρήστης με «διπλό κλικ» μπορεί να επιλέξει ένα ή περισσότερα items από αυτά, και αμέσως στο **panel** που βρίσκεται δεξιά από το listbox εμφανίζεται ένα **CheckedListBox** το οποίο περιέχει τα βασικότερα πεδία του εκάστοτε πίνακα. Στη συνέχεια, ο χρήστης μπορεί να επιλέξει ένα η

περισσότερα από αυτά τα πεδία και αμέσως εμφανίζονται στο DataGridView που βρίσκεται από κάτω, δίνοντας του τη δυνατότητα να θέσει κάποιο περιορισμό για τα πεδία αυτά. Καθώς ο χρήστης κάνει την πιο πάνω διαδικασία και ανάλογα με τις επιλογές του χτίζεται ένα Query. Μόλις πατήσει το κουμπί **«Εμφάνιση Αποτελεσμάτων»,** το Query εκτελείται και ο χρήστης μεταφέρεται στο δεύτερο tab. Στο tab αυτό, τοποθετήθηκε ένα DataGridView στο οποίο εμφανίζονται τα αποτελέσματα, καθώς επίσης και ένα κουμπί με ένδειξη **«Εξαγωγή Αποτελεσμάτων σε Excel»** δίνοντας έτσι τη δυνατότητα στο χρήστη να κάνει εξαγωγή τον αποτελεσμάτων σε Excel για περαιτέρω διαχείριση.



Εικόνα 50: «Φόρμα frmCustomReportGenerator»

Επίλογος:

Στο κεφάλαιο αυτό παρουσιάστηκε ο τρόπος με τον οποίο σχεδιάστηκε το User Interface της εφαρμογής, ώστε να είναι φιλικό και εύχρηστο προς τον χρήστη. Στη συνέχεια, ακολουθεί η παρουσίαση των τεχνικών και μεθοδολογιών οι οποίες χρησιμοποιήθηκαν, ώστε να διατηρηθεί σε υψηλά επίπεδα η Ποιότητα και Αξιοπιστία λογισμικού.

ΚΕΦΑΛΑΙΟ 7: Ποιότητα και Αξιοπιστία Λογισμικού

Εισαγωγή:

Κατά την ανάπτυξη μιας εφαρμογής πρέπει να γίνει χρήση σωστών μεθοδολογιών και τεχνικών ώστε να διατηρηθεί σε υψηλά επίπεδα η Ποιότητα και Αξιοπιστία λογισμικού. Στο κεφάλαιο αυτό καθώς παρουσιάζονται οι τεχνικές και μεθοδολογίες που χρησιμοποιήθηκαν ώστε να οδηγήσουν σε ένα καλό αποτέλεσμα όσον αφορά τα πιο πάνω.

Μεθοδολογίες και τεχνικές που χρησιμοποιήθηκαν

Καταρχήν, η όλη ανάπτυξη της εφαρμογής βασίστηκε στην Μεθοδολογία «Ανάπτυξη με Προτυποποίηση» (Development by Prototyping). Με την Μεθοδολογία αυτή, μειώνονται οι κίνδυνοι της αβεβαιότητας και είναι ένας τρόπος ανάπτυξης που μπορεί να οδηγήσει εύκολα σε ένα ποιοτικό και αξιόπιστο λογισμικό, για λόγους που αναφέρθηκαν σε προηγούμενο κεφάλαιο (βλ. Κεφάλαιο 3).

Πέραν από την μεθοδολογία αυτή, κατά την ανάπτυξη του κώδικα χρησιμοποιήθηκαν διάφορες τεχνικές, οι οποίες συνέβαλαν στην διατήρηση της ποιότητας και αξιοπιστίας λογισμικού σε υψηλά επίπεδα και παρουσιάζονται πιο κάτω :

- Η εφαρμογή διαχωρίστηκε σε τρία projects (Solution), ώστε να είναι πιο εύκολη τόσο η ανάπτυξη του κώδικα όσο και η αποσφαλμάτωσή του (βλ. Κεφάλαιο 5).

- Ο κώδικας χωρίστηκε σε περιοχές **«# region»,** όπου σε κάθε περιοχή από αυτές δόθηκε ένα μοναδικό όνομα που να την χαρακτηρίζει. Οι περιοχές αυτές περιέχουν μεθόδους, οι οποίες έχουν παρόμοια χαρακτηριστικά ως

προς την συμπεριφορά τους. Αποτέλεσμα της πιο πάνω διαδικασίας είναι πιο τακτοποιημένος κώδικας και ακόμα πιο εύκολη αποσφαλμάτωση.

- Όλος ο κώδικας είναι γραμμένος μέσα σε «**try-catch statement»**, ώστε να γίνεται ορατό κάθε σφάλμα κατά την ανάπτυξη του και να γίνεται άμεση διαχείριση του.

- Χρησιμοποιήθηκε η τεχνική **«Generic Programming»,** δηλαδή, όπου ήταν δυνατόν, αναπτύχθηκαν μέθοδοι οι οποίες για διαφορετικές παραμέτρους ή διαφορετικές καταστάσεις παρουσιάζουν διαφορετική συμπεριφορά. Έτσι αποφεύγονται οι αλληλοεπικαλύψεις και η πολυπλοκότητα.

- Σε μεθόδους, όπου η εκτέλεσή τους, προκαλεί αλλαγές σε περισσότερους από ένα πίνακες χρησιμοποιήθηκε η τεχνική **«Transaction».** Με την τεχνική αυτή, ο προγραμματιστής διαβεβαιώνεται ότι οι αλλαγές θα γίνουν είτε σε όλους τους πίνακες είτε σε κανένα. Αναλυτικότερα, αν δεν υπάρξει σφάλμα κατά την εκτέλεση του κώδικα οι αλλαγές πραγματοποιούνται σε όλους τους πίνακες, αλλιώς αν υπάρξει οποιοδήποτε σφάλμα κατά την διάρκεια εκτέλεσης της μεθόδου, έστω και αν προηγήθηκε αλλαγή σε κάποιο πίνακα πριν από το σφάλμα αυτό, εντέλει, οι αλλαγές δεν γίνονται σε κανένα πίνακα γιατί τα δεδομένα αυτά είναι αλληλοσυσχετιζόμενα μεταξύ τους.

- Όλα τα δεδομένα που εισάγονται από τον χρήστη, ελέγχονται ως προς τον τύπο τους. Ελέγχεται η εγκυρότητα τους δηλαδή, και αν ο χρήστης εισάγει λανθασμένα δεδομένα, ενημερώνεται με κατάλληλο μήνυμα που τον προτρέπει για την διόρθωση τους, πριν αυτά αποθηκευτούν στη βάση ή προκαλέσουν σφάλμα κατά την εκτέλεση του κώδικα.

- Οι κωδικοί των χρηστών, κωδικοποιούνται πριν την αποθήκευση τους στη βάση και σε συνδυασμό με το γεγονός ότι, για κάθε αλληλεπίδραση του χρήστη με τη βάση του συστήματος, αποθηκεύεται και το ID του στη βάση, αποφεύγονται τυχόν κακόβουλες ενέργειες. Ενέργειες, δηλαδή, που έχουν να κάνουν είτε με την είσοδο μη εξουσιοδοτημένων χρηστών στο σύστημα, είτε με την πρόκληση ζημιών (π.χ παραποίηση δεδομένων, διαγραφή

δεδομένων) από εξουσιοδοτημένους χρήστες, στην οποία μπορεί να οδηγηθούν για δικούς τους λόγους.

- Ο χρήστης σε κάθε αλληλεπίδραση του με την εφαρμογή, ενημερώνεται με κατάλληλα μηνύματα, τα οποία είτε τον επιβεβαιώνουν για την ορθότητα της ενέργεια που μόλις εκτέλεσε, είτε τον ενημερώνουν σε περίπτωση λάθους, με αποτέλεσμα να επικυρώνεται η αξιοπιστία της εφαρμογής.

Επίλογος:

Οι τεχνικές και μεθοδολογίες που αναφέρθηκαν πιο πάνω, οδήγησαν στην εύκολη ανάπτυξη και αποσφαλμάτωση του κώδικα, στην επίτευξη των στόχων λογισμικού και στην ασφάλεια των δεδομένων που συντηρούνται, καθώς επίσης και στην ευκολία της μετέπειτα συντήρησής του.
Αδιαμφισβήτητα, σημαντικές παράμετροι, ως προς τη συμβολή τους στη διατήρηση της Ποιότητας και Αξιοπιστίας Λογισμικού.

ΚΕΦΑΛΑΙΟ 8:

Γνώσεις από μαθήματα της σχολής που έχουν εφαρμοστεί κατά τη σχεδίαση και ανάπτυξη της εφαρμογής

Εισαγωγή:

Κατά τη φοίτησή μου στην Σχολή Πληροφορικής του ΑΤΕΙ Θεσσαλονίκης, έχω εφοδιαστεί με αρκετές πρακτικές και θεωρητικές γνώσεις γύρω από την τεχνολογία και τον κόσμο των υπολογιστών. Γνώσεις που έχω προσπαθήσει να εφαρμόσω κατά τον σχεδιασμό και ανάπτυξη της εφαρμογής αυτής.

Μερικές από αυτές ανά μάθημα διδασκαλίας:

ΠΡΟΓ/ΣΜΟΣ ΥΠΟΛΟΓΙΣΤΩΝ Ι / ΠΡΟΓ/ΣΜΟΣ ΥΠΟΛΟΓΙΣΤΩΝ ΙΙ

Η διδασκαλία προγραμματισμού στις διάφορες γλώσσες και κυρίως σε Java, που είναι σε μεγάλο βαθμό όμοια με την C#, με βοήθησε τόσο στη μάθηση προγραμματισμού όσο και στην διαμόρφωση προγραμματιστικής σκέψης, εφόδια που μου ήταν απαραίτητα για να φέρω εις πέρας την ολοκλήρωση της πτυχιακής μου εργασίας.

ΜΕΘΟΔΟΛΟΓΙΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΙΙ

Η δημιουργία και ανάλυση των διαφόρων διαγραμμάτων που διδάχτηκαν στο μάθημα αυτό, με βοήθησαν σημαντικά στην ανάλυση και επεξήγηση του Class Diagram που παράχθηκε από την παρούσα εφαρμογή.

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ Ι / ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΙΙ

Οι γνώσεις που απόκτησα γύρω από τις βάσεις δεδομένων, όπως δημιουργία και συσχέτιση πινάκων, σύνδεση και επικοινωνία βάσης με διάφορες γλώσσες προγραμματισμού, εκτέλεση ερωτήσεων προς τη βάση και ανάκτηση δεδομένων με βοήθησαν σημαντικά στην υλοποίηση της εφαρμογής, καθώς ένα μεγάλο μέρος της λειτουργικότητας του προγράμματος, αν όχι το μεγαλύτερο βασίζεται σε μια βάση δεδομένων.

ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΙΙ

Ένα από τα σημαντικότερα κομμάτια του μαθήματος αυτού, είναι η διδασκαλία των διαφόρων μεθοδολογιών ανάπτυξης λογισμικού, η επιλογή της οποίας προκύπτει από τις εκάστοτε ανάγκες της εφαρμογής που πρόκειται να αναπτυχθεί.
Με βάση τις γνώσεις αυτές και αξιολογώντας τις καταστάσεις, πάρθηκε η απόφαση η εφαρμογή να αναπτυχθεί βάσει της μεθοδολογίας «Ανάπτυξη κατά Προτυποποίηση».

ΠΟΙΟΤΗΤΑ ΚΑΙ ΑΞΙΟΠΙΣΤΙΑ ΛΟΓΙΣΜΙΚΟΥ

Το μάθημα αυτό σε συνδυασμό με τα πιο πάνω μαθήματα, με βοήθησαν τόσο στον τρόπο ανάπτυξης του κώδικα της εφαρμογής, όσο και στον τρόπο αξιολόγησης της ποιότητας και αξιοπιστίας του λογισμικού.
Η εφαρμογή χωρίστηκε σε τρία μέρη, όπως αυτά έχουν περιγραφεί σε προηγούμενο κεφάλαιο, με αποτέλεσμα πιο τακτοποιημένο κώδικα, πιο εύκολη ανάπτυξη κώδικα καθώς επίσης και πιο εύκολη αποσφαλμάτωση του.

ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ένας από τους σημαντικότερους τομείς της Πληροφορικής είναι και η Ασφάλεια Πληροφοριακών συστημάτων. Στις μέρες που διανύουμε αλλά και χρόνια πριν, οι κακόβουλες επιθέσεις είναι πολύ συχνό φαινόμενο, είτε αυτές γίνονται με σκοπό

το προσωπικό κέρδος είτε με σκοπό την προσωπική ικανοποίηση του ατόμου που προκάλεσε τη ζημία.
Για αποφυγή τέτοιων καταστάσεων, κατά την εγγραφή νέου χρήστη, το username και το password που δίνει, κωδικοποιούνται προτού να αποθηκευτούν στη βάση.
Ο τρόπος με τον οποίο γίνεται η κωδικοποίηση έχει περιγραφεί σε προηγούμενο κεφάλαιο.

ΑΝΑΠΤΥΞΗ ΔΙΕΠΙΦΑΝΕΙΩΝ ΧΡΗΣΤΗ

Πολύ σημαντικό μέρος κατά την ανάπτυξη μιας εφαρμογής είναι η διεπιφάνεια χρήστη. Πρέπει να είναι σωστά σχεδιασμένη, ώστε να είναι όσο το δυνατό περισσότερο φιλική προς το χρήστη και να μην δημιουργούνται απορίες μόνο με το κοίταγμά της. Να είναι δηλαδή ένα ωραίο και ευχάριστο περιβάλλον, του οποίου οι ενέργειες που μπορούν να εκτελεστούν να είναι απλές και ξεκάθαρες.

Επίλογος:

Συνοψίζοντας, τα εφόδια που μπορεί να πάρεις κανείς από τη σχολή γύρω τους ηλεκτρονικούς υπολογιστές, είναι υπέρ αρκετά ώστε να τον βοηθήσουν τόσο στην αντιμετώπιση τυχόν δυσκολιών που θα συναντήσει στη μετέπειτα εξέλιξη του όσο και στην επαγγελματική του αποκατάσταση .

ΕΠΙΛΟΓΟΣ

Συνοψίζοντας, όπως διαφαίνεται από την πιο πάνω λεπτομερή παρουσίαση της δομής και της λειτουργίας του λογισμικού συστήματος που αναπτύχθηκε, η εφαρμογή βήμα προς βήμα της ορθής σειράς, με την οποία αναλύεται και αναπτύσσεται ένα σύστημα λογισμικού, δεν μπορεί παρά να οδηγήσει σε ένα καλό αποτέλεσμα, όσον αφορά την επίτευξη των στόχων λογισμικού, οι οποίοι στην προκειμένη περίπτωση ήταν, τόσο η δημιουργία ενός εύχρηστου και λειτουργικού λογισμικού συστήματος, το οποίο να ανταποκρίνεται στις απαιτήσεις της επιχείρησης, όσο και ο εμπλουτισμός των γνώσεων μου στον τομέα αυτό.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1) Παναγιωτοπούλου, Ι.Χ. (2003), *Από την Java στη C#*, Πειραιάς 2003.

2) Halvorson, M. (2008), *Microsoft Visual Basic 2008 Βήμα Βήμα*.

3) Ramakrishnan R. , Gehrke J. (2002), *Συστήματα Διαχείρισης Βάσεων Δεδομένων*, Τόμος Α, 2η έκδοση.

4) Ramakrishnan R. , Gehrke J. (2002), *Συστήματα Διαχείρισης Βάσεων Δεδομένων*, Τόμος Β, 2η έκδοση.

5) Sharp, J. (2008), *Microsoft Visual C# 2008 Βήμα Βήμα.*

Διαδικτυακοί τόποι και Ιστιοσελίδες

1) http://msdn.microsoft.com/
2) http://www.csharp-station.com/Tutorial.aspx
3) http://www.csharphelp.com/2006/12/c-tutorial-for-beginners
4) http://www.java2s.com/Tutorial/CSharp/CatalogCSharp.htm
5) http://www.functionx.com/sqlserver/index.htm
6) http://www.infiniteskills.com/training/crystal-reports-2008-fundamentals.html
7) http://www.wikipedia.org
8) http://www.youtube.com

ΠΑΡΑΡΤΗΜΑ: Ανάλυση τμημάτων κώδικα

Μέθοδοι και διαδικασίες που εκτελούνται κατά την προσπάθεια σύνδεσης ενός χρήστη με το σύστημα

Διαδικασία 1: «Μέθοδος btnCancelLogin_Click()»

```
private void btnCancelLogin_Click(object sender, EventArgs e)
{
      Application.Exit();
}
```

Με το πάτημα του κουμπιού **«Cancel»** καλείται η μέθοδος **btnCancelLogin_Click()** και εκτελείται η εντολή **Application.Exit**(), με την οποία η εφαρμογή κλείνει.

Διαδικασία 2: «Μέθοδος AttemptLogin()»

```
private void AttemptLogin()
{
    Classes.clsUser user = new Classes.clsUser();
    if (user.Load(txtUsername.Text, txtPassword.Text) == true)
    {
        {
            Classes.clsCommon.LoggedInUserID = user.UserID;
            frmWelcomeScreen welcome = new frmWelcomeScreen();
            frmMain main = new frmMain();
            if (user.IsAdministrator)
               main.χρήστεςToolStripMenuItem.Enabled = true;
             else
                main.χρήστεςToolStripMenuItem.Enabled = false;

            welcome.Show();
            this.Hide();
        }
    }
    else
             MessageBox.Show("Το όνομα χρήστη ή ο κωδικός που πληκτρολογήσατε
             είναι  λανθασμένος.\nΠαρακαλώ ξαναδοκιμάστε.");
}
```

Όταν πατηθεί το κουμπί **«Log In»** καλείται η μέθοδος **AttemptLogin()** για επιβεβαίωση των στοιχείων του χρήστη. Αρχικά δημιουργείται ένα Instance της κλάσης clsUser και μέσω αυτού καλείται η μέθοδος **Load()** της κλάσης αυτής (βλ. Διαδικασία 3). Η μέθοδος Load() είναι τύπου «**bool**», δέχεται σαν παράμετρο το Username και το Password του χρήστη και εκτελώντας ένα Query προς τη βάση,

ελέγχει αν όντως ο χρήστης αυτός υπάρχει στη βάση. Αν τα στοιχεία του χρήστη είναι λανθασμένα (δηλαδή δεν υπάρχει στη βάση), εμφανίζεται κατάλληλο μήνυμα και του δίνεται η δυνατότητα να ξαναδοκιμάσει. Αντίθετα αν υπάρχει στη βάση, φορτώνονται όλα τα στοιχεία του στις μεταβλητές της κλάσης για περαιτέρω διαχείριση. Στη συνέχεια, γίνεται «set» το «UserID» του χρήστη στην κλάση clsCommon για λόγους που επεξηγήθηκαν σε προηγούμενο κεφάλαιο.

Ακολούθως δημιουργείται ένα Instance της frmMain, γίνεται έλεγχος και αν ο χρήστης είναι Administraton, όλα τα εργαλεία της εφαρμογής γίνονται διαθέσιμα, αλλιώς κάποια από αυτά γίνονται μη διαθέσιμα (Enabled False).

Τέλος, εμφανίζεται η φόρμα «υποδοχής»(frmWelcomeScreen) και γίνεται απόκρυψη της φόρμας «frmlogin».

Διαδικασία 3: «Μέθοδος Load() - Κλάση clsUser»

```
public bool Load(string username, string password)
{
   try
   {
      cn = new SqlConnection(clsCommon.MyConnectionString());
      command = new SqlCommand();
      command.Connection = cn;
      command.CommandText =
    "Open Symmetric Key symLock Decryption by Asymmetric Key asymLock with
password='794613'; " +
        "SELECT
        usr_UserID,usr_Password,usr_Username,usr_LastName,usr_FirstName,usr_Active,usr_Ad
        ministrator FROM Users " +
        "WHERE usr_Username='" + username + "' AND
        Convert(varchar(max),DecryptByKey(usr_Password))='" + password + "' AND
        usr_Active='True' AND usr_Deleted='False'; "+
       "Close symmetric key symLock";

       dtUser = new DataTable("User");
       da = new SqlDataAdapter();
       da.SelectCommand = command;
       cn.Open();
       da.Fill(dtUser);
       if (dtUser.Rows.Count == 1)
       {
          this.UserID = Convert.ToInt32(dtUser.Rows[0]["usr_UserID"]);
          this.FirstName = dtUser.Rows[0]["usr_FirstName"].ToString();
          this.LastName = dtUser.Rows[0]["usr_LastName"].ToString();
          this.Username = dtUser.Rows[0]["usr_Username"].ToString();
          this.Password = dtUser.Rows[0]["usr_Password"].ToString();
          this.Active = Convert.ToBoolean(dtUser.Rows[0]["usr_Active"]);
          this.IsAdministrator = Convert.ToBoolean(dtUser.Rows[0]["usr_Administrator"]);
          cn.Close(); command.Dispose(); da.Dispose();
          return true;
        }
        else
        {
           cn.Close(); command.Dispose(); da.Dispose();
           return false;
        }
            }
   catch(Exception ex)
   {
      throw ex;
   }
}
```

Όπως φαίνεται και από τον κώδικα του «Πίνακα 3», η μέθοδος **Load()** δέχεται δύο παραμέτρους, το Username και το Password του χρήστη.
Σε μια μεταβλητή τύπου **SqlConnection** αποθηκεύεται το **ConnectionString,** το οποίο είναι απαραίτητο για τη σύνδεση με τη βάση. Στη συνέχεια αρχικοποιείται ένα αντικείμενο τύπου **SqlCommand** με όνομα **command** και γίνεται **set** σε αυτό το **ConnectionString.** Ακολούθως, γίνεται **set** στον **command** και το **Query** που θα εκτελεστεί στη βάση για ταυτοποίηση των στοιχείων του χρήστη.

Ενέργειες που θα εκτελεστούν με το Query:

Αρχικά, ανοίγει η σύνδεση με τον αλγόριθμο κρυπτογράφησης και στη συνέχεια αφού αποκρυπτογραφηθούν οι κωδικοί που βρίσκονται στη βάση, γίνεται η σύγκριση και η ταύτιση με τον κωδικό που έδωσε ο χρήστης. Τέλος, εμφανίζονται όλα τα στοιχεία του χρήστη που έχει για Username και Password αυτά που δέχτηκε σαν παράμετρο η μέθοδος και η σύνδεση με τον αλγόριθμο κλείνει.

Εν συνεχεία, γίνεται αρχικοποίηση ενός **DataTable** με όνομα «**User»** και ακολουθεί η αρχικοποίηση του **SqlDataAdapter.** Στο SqlDataAdapter που μόλις αρχικοποιήθηκε γίνεται set το αντικείμενο command.
Στη συνέχεια ανοίγει η σύνδεση με τη βάση και με τη βοήθεια του **αντικειμένου da** (τύπου SqlDataAdapter) εκτελείτε το Query, καλείται η **μέθοδος Fill()** και γεμίζει το DataTable με ό,τι αυτό επιστρέψει.
Αφού γεμίσει το DataTable με ένα **«if statement»,** γίνεται έλεγχος του αριθμού των γραμμών που περιέχει. Αν ο αριθμός αυτός ισούται με ένα, δηλαδή τα στοιχεία που έδωσε ο χρήστης είναι σωστά (συνεπώς υπάρχει στη βάση), φορτώνονται στις μεταβλητές της κλάσης όλα τα στοιχεία του χρήστη μέσω του DataTable και η μέθοδος επιστρέφει True, αλλιώς η μέθοδος επιστρέφει False.

Διαδικασία 4: «Μέθοδος Save() - Κλάση clsUser»

```
public void Save()
{
   try
   {
      cn = new SqlConnection(clsCommon.MyConnectionString());
      string strCommand = "";
      if (this.blNewEntry)
      {
         strCommand =
       "Open Symmetric Key symLock Decryption by Asymmetric Key asymLock with
password = '794613'; " +
       "insert into Users
(usr_FirstName,usr_LastName,usr_UserName,usr_Password,usr_Active,usr_Administrator) "
+
        "values ('" + this.FirstName + "','" + this.LastName + "','" + this.Username +
        "',(EncryptByKey(Key_GUID('symLock'),CONVERT(varchar(max),'" + this.Password +
        "'))),'" + this.Active + "','" + this.IsAdministrator + "'); " +
         "close symmetric key symLock";
      }
      else
      {
         strCommand =
         "Open Symmetric Key symLock Decryption by Asymmetric Key asymLock with
password = '794613'; " +
        "UPDATE Users SET usr_FirstName=N'" + this.FirstName + "',usr_LastName= N'" +
        this.LastName +  "',usr_Username='"+this.Username+"',usr_Active='" +
        this.Active + "',usr_Administrator='" + this.IsAdministrator +
        "',usr_Password=(EncryptByKey(Key_GUID('symLock'),CONVERT(varchar(max),'" +
        this.Password + "'))) WHERE usr_UserID=" + this.UserID +";"+
                    "close symmetric key symLock";
    }
    command = new SqlCommand(strCommand, cn);

    cn.Open();
    command.ExecuteNonQuery();
    cn.Close();
    command.Dispose();
    cn.Dispose();
  }
  catch(Exception ex)
  {
     throw ex;
  }
```

Η μέθοδος Save() της κλάσης clsUser καλείται στις εξής περιπτώσεις:

- κατά την αποθήκευση ενός νέου χρήστη
- κατά την ενημέρωση των στοιχείων ενός χρήστη που ήδη υπάρχει

Επεξήγηση κώδικα:

Γίνεται έλεγχος και αν η μέθοδος κλήθηκε για την αποθήκευση ενός νέου χρήστη (blNewEntry=true), τότε, αποθηκεύεται σε μια μεταβλητή τύπου string το Query το οποίο θα εκτελεστεί για την αποθήκευση του στη βάση. Με το Query αυτό, θα γίνει εισαγωγή όλων των στοιχείων του χρήστη στη βάση, και μάλιστα το Password του θα κωδικοποιηθεί προτού αποθηκευτεί για λόγους ασφάλειας.

Σε περίπτωση που η μέθοδος κλήθηκε για την αποθήκευση ενός χρήστη που ήδη υπάρχει στη βάση (blNewEntry=false), τότε στην string μεταβλητή

αποθηκεύεται το Query το οποίο θα εκτελεστεί για την ανανέωση των στοιχείων του χρήστη.

Στη συνέχεια, αρχικοποιείται το command με το ConnectionString και το string που περιέχει το Query, ανοίγει η σύνδεση με τη βάση, εκτελείται το Query και η σύνδεση κλείνει.

Διαδικασία φόρτωσης των δεδομένων σε ένα DataGridView από τη στιγμή που επιλέγεται ένα Item

Έστω ότι επιλέχθηκε το Item Πελάτες:

Διαδικασία 5: «customersToolStripMenuItem_Click()»

```
public void customersToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (isLocked)
        btnReturnToPersons_Click(sender, e);

    try
    {
        tbcMain.SelectedTab = tbcMain.TabPages["tpgPersons"];
        txtPersonType.Text =
        Convert.ToString(Convert.ToInt32(Enums.clsEnums.PersonType.Customer));
        LoadGridPersons(Convert.ToInt32(Enums.clsEnums.PersonType.Customer));
    }
    catch (Exception ex)
    {
        MessageBox.Show("Σφάλμα: " + ex.Message);
    }
}
```

Αρχικά ελέγχεται η τιμή της μεταβλητής «isLocked», αν είναι ίση με true, δηλαδή αν έχει προηγηθεί η επιλογή της «Προβολής Ιστορικού» από τον χρήστη, τότε επιστρέφει κατευθείαν στο tab Persons, χωρίς να εκτελεστούν οι υπόλοιπες ενέργειες (βλ. Διαδικασία 16).

Σε αντίθετη περίπτωση, κατευθείαν τίθεται ως επιλεγμένο tab, το tab Person.

Στη συνέχεια, στο κρυφό textbox του tab Persons, τίθεται ως περιεχόμενο του, η τιμή του «enum Customer» και καλείται η μέθοδος **LoadGridPersons()** βάσει της τιμής του enum αυτού (βλ. Διαδικασία 6).

Διαδικασία 6: «Μέθοδος LoadGridPerson()»

```
private void LoadGridPersons(int PersonType)
{
      Classes.clsPerson person = new Classes.clsPerson(PersonType);
      person.Load(-2);
      dgvPersons.DataSource = person.dt;
      dgvPersons.Columns["PersonID"].Visible = false;
      dgvPersons.Columns["per_PersonTypeCode"].Visible = false;
      dgvPersons.Columns["ΣΗΜΕΙΩΣΕΙΣ"].Visible = false;
      dgvPersons.Columns["ΟΝΟΜΑ"].Width = 200;
      dgvPersons.Columns["ΔΙΕΥΘΥΝΣΗ"].Width = 300;
      dgvPersons.Columns["ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ"].Width = 200;
      dgvPersons.Columns["ΙΣΤΟΣΕΛΙΔΑ"].Width = 200;
      person = null;

      if (dgvPersons.Rows.Count > 0)
      {
         if (dgvPersons.SelectedRows.Count == 1)
         {
            dgvPersons.SelectedRows[0].Selected = false;
            ClearFieldsPerson();
         }
      }
}
```

Αφού κληθεί η μέθοδος αυτή, δημιουργείται ένα αντικείμενο της κλάσης **«clsPerson»,** αρχικοποιώντας την μεταβλητή **«intPersonType»** της κλάσης αυτής (βλ. Διαδικασία 7), που ουσιαστικά είναι ο τύπος του ατόμου (στην προκειμένη περίπτωση Customer).

Στη συνέχεια με τη βοήθεια του αντικειμένου που δημιουργήθηκε, καλείται η μέθοδος **«Load()»** της κλάσης clsPerson, για την άντληση όλων των δεδομένων του πίνακα «Person» και την αποθήκευση τους σε ένα αντικείμενο τύπου DataTable (βλ. Διαδικασία 7).

Ακολούθως, με τη βοήθεια του αντικειμένου αυτού, τίθενται στο DataSource του DataGridView (dgvPersons) τα δεδομένα που αντλήθηκαν, κρύβονται οι μη επιθυμητές στήλες και ορίζεται το μέγεθος κάποιων από αυτές που θα εμφανιστούν στο DataGridView.

Μετά από την πιο πάνω διαδικασία, το αντικείμενο person αδειάζει για να μην σπαταλείται μνήμη και από-επιλέγεται η by default επιλεγμένη εγγραφή του DataGridView.

Διαδικασία 7: «Μέθοδος Load()-Κλάση clsPerson»

```
public clsPerson(int PersonType)
{
   this.intPersonType = PersonType;
}

public void Load(int PersonID)
{
   this.perPersonID = PersonID;

    try
    {
       cn = new SqlConnection(clsCommon.MyConnectionString());
       command = new SqlCommand();
       command.Connection = cn;

       if (this.perPersonID == -2)
 command.CommandText =
"SELECT SUM(tra_AmountRemaining) as ΙΣΟΖΥΓΙΟ, per_personID as PersonID, per_Name as
ONOMA,    per_Address as ΔΙΕΥΘΥΝΣΗ, per_Telephone as 'ΑΡ. ΤΗΛΕΦΩΝΟΥ', per_Fax as 'ΑΡ.
ΦΑΞ', " +
"per_Email as 'ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ', per_Website as ΙΣΤΟΣΕΛΙΔΑ, per_PersonTypeCode,
per_Notes as  'ΣΗΜΕΙΩΣΕΙΣ' FROM Person LEFT JOIN [Transaction] on per_PersonID =
tra_PersonCode " +
"WHERE per_Deleted ='False' AND per_PersonTypeCode=" + this.intPersonType +
" GROUP BY
per_PersonID,per_Name,per_Address,per_Telephone,per_Fax,per_Email,per_Website,per_Pers
onTypeCode,per_Notes";

       else
command.CommandText =
"SELECT SUM(tra_AmountRemaining) as ΙΣΟΖΥΓΙΟ, per_personID as PersonID, per_Name as
ONOMA,            per_Address as ΔΙΕΥΘΥΝΣΗ, per_Telephone as 'ΑΡ. ΤΗΛΕΦΩΝΟΥ', per_Fax
as 'ΑΡ. ΦΑΞ', " +
"per_Email as 'ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ', per_Website as ΙΣΤΟΣΕΛΙΔΑ, per_PersonTypeCode FROM
Person INNER JOIN [Transaction] on per_PersonID=tra_PersonCode " +
"WHERE per_PersonID=" + this.perPersonID + "GROUP BY
per_personID,per_Name,per_Address,per_Telephone,per_Fax,per_Email,per_Website,per_Pers
onTypeCode ";
          dt = new DataTable("Person");
          da = new SqlDataAdapter();
          da.SelectCommand = command;
          cn.Open();
          da.Fill(dt);
          cn.Close();
          command.Dispose();
          da.Dispose();

       if (PersonID != -2)
       {
           this.perPersonID = Convert.ToInt32(dt.Rows[0]["PersonID"]);
           this.perName = Convert.ToString(dt.Rows[0]["ONOMA"]);
           this.perAddress = Convert.ToString(dt.Rows[0]["ΔΙΕΥΘΥΝΣΗ"]);
           this.perTelephone = Convert.ToString(dt.Rows[0]["ΑΡ. ΤΗΛΕΦΩΝΟΥ"]);
           this.perFax = Convert.ToString(dt.Rows[0]["ΑΡ. ΦΑΞ"]);
           this.perEmail = Convert.ToString(dt.Rows[0]["ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ"]);
           this.perWebsite = Convert.ToString(dt.Rows[0]["ΙΣΤΟΣΕΛΙΔΑ"]);
           this.perNotes = Convert.ToString(dt.Rows[0]["ΣΗΜΕΙΩΣΕΙΣ"]);
           this.intPersonType = Convert.ToInt32(dt.Rows[0]["per_PersonTypeCode"]);
        }

    }
    catch (Exception ex)
    {
       throw ex;
    }
}
```

Καταρχήν, η μέθοδος Load() παρουσιάζει δύο συμπεριφορές ανάλογα με την παράμετρο που δέχεται. Αν η παράμετρος αυτή είναι ίση με «-2», τότε εκτελεί ένα Query και αντλεί από τη βάση όλες τις εγγραφές του πίνακα που δεν είναι διαγεγραμμένες (per_Deleted=False) και σε σχέση πάντοτε με τον τύπο του ατόμου (PersonType), αντιθέτως, αν η παράμετρος είναι ίση με το ID κάποιου συγκεκριμένου ατόμου, φορτώνει τα στοιχεία του ατόμου αυτού και δίνει τις τιμές τους στις μεταβλητές της κλάσης για περαιτέρω διαχείριση της συγκεκριμένης εγγραφής .
Και στις δύο περιπτώσεις καλείται η πιο κάτω μέθοδος για να αδειάσει το περιεχόμενο των textbox του tab Persons.

Διαδικασία 8: «Μέθοδος ClearFieldsPerson()»

```
private void ClearFieldsPerson()
{
    txtName.Clear();
    txtEmail.Clear();
    txtFax.Clear();
    txtBalance.Clear();
    txtAddress.Clear();
    txtWebsite.Clear();
    txtPhone.Clear();
    txtNotesPerson.Clear()
}
```

Διαδικασία Προσθήκης νέας εγγραφής

Έστω ότι ο χρήστης βρίσκεται στο tab Persons. Όταν πατήσει το κουμπί «Προσθήκη» εκτελείται ο πιο κάτω κώδικας:

Διαδικασία 9: «Μέθοδος btnAddPerson_Click()»

```
private void btnAddPerson_Click(object sender, EventArgs e)
{
    grpPersonDetails.Enabled = true;
    btnSavePerson.Enabled = true;
    btnCancelPerson.Enabled = true;

    btnAddPerson.Enabled = false;
    btnUpdatePerson.Enabled = false;
    btnDeletePerson.Enabled = false;
    dgvPersons.Enabled = false;
    ClearFieldsPerson();
    if (txtPersonType.Text != "")
    {
       try
       {
          Person = new clsPerson(Convert.ToInt32(txtPersonType.Text));
          Person.blNewEntry = true;
       }
       catch (Exception ex)
       {
          throw ex;
       }
    }
}
```

Με την κλήση της μεθόδου αυτής, αρχικά, γίνεται διαθέσιμο το grpPersonDetails, τα κουμπιά «Αποθήκευση» και «Ακύρωση», ενώ γίνονται μη διαθέσιμα τα κουμπιά «Προσθήκη», «Ενημέρωση», «Διαγραφή» καθώς και το DataGridView. Έτσι ο χρήστης έχει τη δυνατότητα, είτε να αποθηκεύσει ένα νέο άτομο στη βάση, είτε να ακυρώσει την ενέργεια που μόλις επέλεξε.

Στη συνέχεια δημιουργείται ένα Instance της κλάσης clsPerson, δίνοντας για τιμή στη μεταβλητή «intPersonType» της κλάσης αυτής, τον τύπο του ατόμου που πρόκειται να αποθηκευτεί και στην «boolean» μεταβλητή (blNewEntry) την τιμή «True», δηλαδή ότι πρόκειται για νέα εισαγωγή.

Αφού ο χρήστης δώσει τα στοιχεία του νέου ατόμου και πατήσει το κουμπί «Αποθήκευση» εκτελείται ο πιο κάτω κώδικας:

Διαδικασία 10: «Μέθοδος btnSavePerson_Click()»

```
private void btnSavePerson_Click(object sender, EventArgs e)
{
   try
   {
      dgvPersons.Enabled = true;
      grpPersonDetails.Enabled = false;
      Person.perAddress = txtAddress.Text;
      Person.perEmail = txtEmail.Text;
      Person.perFax = txtFax.Text;
      Person.perName = txtName.Text;
      Person.perWebsite = txtWebsite.Text;
      Person.perTelephone = txtPhone.Text;
      Person.perNotes = txtNotesPerson.Text;
      Person.Save();
      MessageBox.Show("Επιτυχής Αποθήκευση!");
   }
   catch (Exception ex)
   {
      MessageBox.Show("Σφάλμα: " + ex.Message);
   }
      Person = null;
      LoadGridPersons(Convert.ToInt32(txtPersonType.Text));
      btnSavePerson.Enabled = false;
      btnCancelPerson.Enabled = false;
      grpPersonDetails.Enabled = false;
      btnAddPerson.Enabled = true;
      btnUpdatePerson.Enabled = true;
      btnDeletePerson.Enabled = true;
}
```

Στη μέθοδο αυτή, με τη βοήθεια του αντικειμένου Person δίνονται στις μεταβλητές της κλάσης clsPerson οι τιμές που εισήγαγε ο χρήστης στα αντίστοιχα textbox. Ακολούθως, καλείται η μέθοδος **«Save()»** της κλάσης αυτής για την αποθήκευση του ατόμου (βλ. Διαδικασία 11). Ο χρήστης ενημερώνεται με κατάλληλο μήνυμα για την επιτυχία της αποθήκευσης και το αντικείμενο «Person» αδειάζει.

Τέλος, καλείται η μέθοδος **«LoadGridPersons()»** (βλ. Διαδικασία 6), για να εμφανιστούν τα δεδομένα του πίνακα στο DataGridView, συμπεριλαμβανομένου και της νέας εγγραφής. Το tab επαναφέρεται στην αρχική του κατάσταση.

Αν ο χρήστης πατούσε **«Ακύρωση»** αντί για **«Αποθήκευση»** ο κώδικας που θα εκτελείτο φαίνεται στην Διαδιακασία 12.

Διαδικασία 11: «Μέθοδος Save()-Κλάση clsPerson»

```
public void Save()
{
   try
   {
        cn = new SqlConnection(clsCommon.MyConnectionString());
        SqlDataAdapter daPerson =
        new SqlDataAdapter("SELECT per_personID as PersonID, per_Name as ONOMA,
        per_Address as ΔΙΕΥΘΥΝΣΗ,      per_Telephone as 'ΑΡ. ΤΗΛΕΦΩΝΟΥ', per_Fax as 'ΑΡ.
        ΦΑΞ', " +
       "per_Email as 'ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ', per_Website as ΙΣΤΟΣΕΛΙΔΑ, per_PersonTypeCode,
       per_Notes as  ΣΗΜΕΙΩΣΕΙΣ FROM Person WHERE per_PersonID=" + this.perPersonID, cn);

       DataTable dtPerson = new DataTable("Person");
       daPerson.Fill(dtPerson);

       if (blNewEntry == true)
       {
           DataRow dr = dtPerson.NewRow();
           dr["ONOMA"] = this.perName;
           dr["ΔΙΕΥΘΥΝΣΗ"] = this.perAddress;
           dr["ΑΡ. ΤΗΛΕΦΩΝΟΥ"] = this.perTelephone;
           dr["ΑΡ. ΦΑΞ"] = this.perFax;
           dr["ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ"] = this.perEmail;
           dr["ΙΣΤΟΣΕΛΙΔΑ"] = this.perWebsite;
           dr["ΣΗΜΕΙΩΣΕΙΣ"] = this.perNotes;
           dr["per_PersonTypeCode"] = this.intPersonType;
           dtPerson.Rows.Add(dr);
        }
        else
        {
           DataRow dr = dtPerson.Rows[0];
           dr["ONOMA"] = this.perName;
           dr["ΔΙΕΥΘΥΝΣΗ"] = this.perAddress;
           dr["ΑΡ. ΤΗΛΕΦΩΝΟΥ"] = this.perTelephone;
           dr["ΑΡ. ΦΑΞ"] = this.perFax;
           dr["ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ"] = this.perEmail;
           dr["ΙΣΤΟΣΕΛΙΔΑ"] = this.perWebsite;
           dr["ΣΗΜΕΙΩΣΕΙΣ"] = this.perNotes;
         }

         SqlCommandBuilder cmdB = new SqlCommandBuilder(daPerson);
         daPerson.Update(dtPerson);

         cmdB.Dispose();
         dtPerson.Dispose();
         daPerson.Dispose();
         cn.Close(); cn.Dispose();
   }
   catch (Exception ex)
   {
       throw ex;
   }
```

Η μέθοδος «Save()» παρουσιάζει δύο συμπεριφορές:

- Αν αναφερόμαστε σε μια υπάρχουσα εγγραφή, εκτελείται ένα Query προς τη βάση αντλώντας όλα τα δεδομένα της εγγραφής αυτής, τα οποία αποθηκεύονται σε ένα DataTable. Στη συνέχεια, με βάση τη δομή και το περιεχόμενο του DataTable δημιουργείται ένα **«DataRow».** Με τη χρήση

του, εισάγονται σε όλες τις στήλες της εγγραφής του DataTable τα νέα στοιχεία που είναι αποθηκευμένα στις μεταβλητές της κλάσης.

Ακολούθως αρχικοποιείται ο **«SqlCommandBuilder»** με βάση τη δομή του πίνακα Person (δηλαδή με βάση τα πεδία και τους τύπους των πεδίων που επέστρεψε το Query που εκτελέστηκε), ώστε να καταστεί δυνατή η ανανέωση της εγγραφής αυτής με τα καινούργια στοιχεία που υπάρχουν στο DataTable.

- Αν αναφερόμαστε σε καινούργια εγγραφή, το Query το οποίο εκτελείται, επιστρέφει μόνο τα πεδία και τους τύπους των πεδίων του πίνακα και αποθηκεύονται στο DataTable. Στη συνέχεια δημιουργείται ένα νέο DataRow με βάση το DataTable και με τη χρήση του εισάγονται τα στοιχεία της νέας εγγραφής στο *DataTable(dtPerson.Rows.Add(dr)).* Ακολούθως, με τη βοήθεια του **SqlCommandBuilder,** εισάγεται η νέα εγγραφή στη βάση.

Διαδικασία 12: «Μέθοδος btnCancelPerson_Click()»

```
private void btnCancelPerson_Click(object sender, EventArgs e)
{

   if (Person != null)
       Person = null;

    ClearFieldsPerson();

    dgvPersons.Enabled = true;
    grpPersonDetails.Enabled = false;
    btnSavePerson.Enabled = false;
    btnCancelPerson.Enabled = false;

    btnAddPerson.Enabled = true;
    btnUpdatePerson.Enabled = true;
    btnDeletePerson.Enabled = true;

}
```

Η μέθοδος αυτή αρχικά ελέγχει αν το αντικείμενο Person δεν είναι null, και αν ναι, το αδειάζει. Στη συνέχεια καλεί τη μέθοδο **«ClearFieldsPerson()»** (βλ. Διαδικασία 8) και επαναφέρει το tab στην αρχική του κατάσταση.

Διαδικασία 13: «Μέθοδος dgvPersons_SelectionChanged()»

```
private void dgvPersons_SelectionChanged(object sender, EventArgs e)
{
  if (dgvPersons.SelectedRows.Count == 1)
  {
    try
    {
      txtName.Text
=dgvPersons.SelectedRows[0].Cells["ΟΝΟΜΑ"].Value.ToString();
      txtAddress.Text =
dgvPersons.SelectedRows[0].Cells["ΔΙΕΥΘΥΝΣΗ"].Value.ToString();
      txtEmail.Text = dgvPersons.SelectedRows[0].Cells["ΗΛ.
ΤΑΧΥΔΡΟΜΕΙΟ"].Value.ToString();
      txtFax.Text = dgvPersons.SelectedRows[0].Cells["ΑΡ.
ΦΑΞ"].Value.ToString();
      txtWebsite.Text =
dgvPersons.SelectedRows[0].Cells["ΙΣΤΟΣΕΛΙΔΑ"].Value.ToString();
      txtPhone.Text = dgvPersons.SelectedRows[0].Cells["ΑΡ.
ΤΗΛΕΦΩΝΟΥ"].Value.ToString();
      txtNotesPerson.Text =
dgvPersons.SelectedRows[0].Cells["ΣΗΜΕΙΩΣΕΙΣ"].Value.ToString();
      txtBalance.Text =
this.dgvPersons.SelectedRows[0].Cells["ΙΣΟΖΥΓΙΟ"].Value.ToString();
    }
    catch (Exception ex)
    {
      MessageBox.Show("Σφάλμα: " + ex.Message);
    }
}
```

Ας θεωρηθεί ότι ο χρήστης βρίσκεται στο tab Person, όταν επιλέξει κάποια από τις εγγραφές του DataGridView, αμέσως καλείται η πιο πάνω μέθοδος .

Γίνεται έλεγχος και αν η επιλεγμένη εγγραφή είναι μόνο μία, τότε τα textbox γεμίζουν με τις τιμές της εγγραφής αυτής.

Διαδικασία 14 : «Μέθοδος btnDeletePerson_Click()»

```
private void btnDeletePerson_Click(object sender, EventArgs e)
{
  if (MessageBox.Show("Είστε σίγουρος ότι θέλετε να διαγράψετε την εγγραφή
αυτή;",
  "Διαγραφή    Ατόμου", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.Yes)
  {
     try
     {
     Person = new clsPerson(Convert.ToInt32(txtPersonType.Text));
     Person.perPersonID =
Convert.ToInt32(dgvPersons.SelectedRows[0].Cells["PersonID"].Value);
     Person.Delete();
     MessageBox.Show("Η εγγραφή έχει διαγραφεί επιτυχώς!");
     LoadGridPersons(Convert.ToInt32(txtPersonType.Text));
     }
     catch (Exception ex)
     {
       MessageBox.Show("Σφάλμα: " + ex.Message);
     }
  }
}
```

Όταν ο χρήστης πατήσει το κουμπί **«Διαγραφή»,** εμφανίζεται ένα «Messagebox» με κατάλληλο μήνυμα για να επιβεβαιώσει την ενέργεια του. Αν από το Messagebox επιλέξει **«Yes»,** τότε δημιουργείται ένα Instance της κλάσης «clsPerson», ορίζοντας τον τύπο του ατόμου σε μια μεταβλητή της κλάσης. Στη συνέχεια ορίζεται και το ID του ατόμου στη μεταβλητή **perPersonID** και καλείται η μέθοδος **«Delete()»** (βλ. Διαδικασία 15). Αφού γίνει η διαγραφή, εμφανίζεται μήνυμα για ενημέρωση του χρήστη και καλείται η μέθοδος **«LoadGridPersons()»** (βλ. Διαδικασία 6) για να φορτώσει στο DataGridView τα νέα δεδομένα που προέκυψαν μετά την διαγραφή.

Διαδικασία 15: «Μέθοδος Delete() - Κλάση clsPerson»

```
public void Delete()
{
    try
    {
        cn = new SqlConnection(clsCommon.MyConnectionString());
        SqlDataAdapter daPerson =
        new SqlDataAdapter("SELECT per_personID as PersonID, per_Name as ΟΝΟΜΑ,
        per_Address as           ΔΙΕΥΘΥΝΣΗ, per_Telephone as 'ΑΡ. ΤΗΛΕΦΩΝΟΥ',
        per_Fax as 'ΑΡ. ΦΑΞ', " +
        "per_Email as 'ΗΛ. ΤΑΧΥΔΡΟΜΕΙΟ', per_Website as ΙΣΤΟΣΕΛΙΔΑ,
per_PersonTypeCode, per_Notes as ΣΗΜΕΙΩΣΕΙΣ, per_Deleted FROM Person WHERE
per_PersonID=" + this.perPersonID, cn);
        DataTable dtPerson = new DataTable("Person");
        daPerson.Fill(dtPerson);
        DataRow dr = dtPerson.Rows[0];
        dr["per_Deleted"] = true;

        SqlCommandBuilder cmdB = new SqlCommandBuilder(daPerson);
        daPerson.Update(dtPerson);

        cmdB.Dispose();
        dtPerson.Dispose();
        daPerson.Dispose();
        cn.Close(); cn.Dispose();
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Με τη μέθοδο **«Delete()»** εκτελείται ένα Query προς τη βάση, και με βάση το ID του ατόμου, αντλούνται/φορτώνονται όλα τα δεδομένα της εγγραφής που πρόκειται να διαγραφεί στις μεταβλητές της κλάσης. Ουσιαστικά η εγγραφή αυτή δεν διαγράφεται από την βάση, αλλά γίνεται μια ανανέωση των πεδίων της. Συγκεκριμένα ορίζεται στο πεδίο **«per_Deleted»** η τιμή **«True»,** έτσι την επόμενη φορά που θα φορτωθούν τα περιεχόμενα του πίνακα Person στο DataGridView η εγγραφή αυτή δεν θα εμφανιστεί (βλ. Διαδικασία 6).

Διαδικασία 16: "Μέθοδος btnViewHistory_Click()"

```
private void btnViewHistory_Click(object sender, EventArgs e)
{
   try
   {
      if (dgvPersons.SelectedRows.Count == 1)
      {
         tbcMain.SelectedTab = tbcMain.TabPages["tpgTransactions"];
         LoadGridTransactionsForPerson(Convert.ToInt32(dgvPersons.SelectedRows[0].Cells[
         "PersonID"].Value));
         grpTransactionDetails.Enabled = false;
         btnAddTransaction.Enabled = false;
         btnUpdateTransaction.Enabled = false;
         btnDeleteTransaction.Enabled = false;
         btnReturnToPersons.Visible = true;
         isLocked = true;
       }
       else
MessageBox.Show("Παρακαλώ επιλέξτε μία εγγραφή.","Επιλογή
εγγραφής",MessageBoxButtons.OK,MessageBoxIcon.Error);
   }
   catch (Exception ex)
   {
         MessageBox.Show("Σφάλμα: " + ex.Message);
   }
}
```

Όταν ο χρήστης πατήσει το κουμπί **«Προβολή Ιστορικού»** καλείται η πιο πάνω μέθοδος. Γίνεται έλεγχος και αν ο χρήστης δεν επέλεξε κάποια εγγραφή από το DataGridView εμφανίζεται κατάλληλο μήνυμα, αλλιώς μεταφέρεται στο **«tab Transactions»**.

Στη συνέχεια, καλείται η μέθοδος **«LoadGridTransactionsForPerson()»** (βλ. Διαδικασία 17), για να γίνει η εμφάνιση των συναλλαγών του συγκεκριμένου ατόμου. Το κουμπί **«Κλείσιμο Προβολής Ιστορικού»** ή αλλιώς «btnReturnToPersons» γίνεται από κρυφό, ορατό (Visible= true) και η μεταβλητή «isLocked» γίνεται true. Η χρήση της μεταβλητής αυτής φαίνεται στον πίνακα 5.

Διαδικασία 17: «Μέθοδος LoadGridTransactionsForPerson()»

```
private void LoadGridTransactionsForPerson(int PersonCode)
{
   try
   {
        Transaction = new clsTransaction();
        Transaction.LoadByPerson(PersonCode);
        dgvTransactions.DataSource = Transaction.dt;

        dgvTransactions.Columns["tra_TransactionID"].Visible = false;
        dgvTransactions.Columns["tra_PersonCode"].Visible = false;
        dgvTransactions.Columns["tra_TransactionTypeCode"].Visible = false;
        dgvTransactions.Columns["tra_TransactionStatusCode"].Visible = false;
        dgvTransactions.Columns["trt_Description"].Visible = false;
        dgvTransactions.Columns["trs_Description"].Visible = false;
        dgvTransactions.Columns["tra_PaymentMethodCode"].Visible = false;
        dgvTransactions.Columns["tra_PaymentTermsCode"].Visible = false;
        dgvTransactions.Columns["tra_ExternalNotes"].Visible = false;
        dgvTransactions.Columns["tra_InternalNotes"].Visible = false;
        dgvTransactions.Columns["tra_Discount"].Visible = false;
        dgvTransactions.Columns["tra_SubTotal"].Visible = false;
        dgvTransactions.Columns["tra_TaxAmount"].Visible = false;
        dgvTransactions.Columns["tra_PaymentInDays"].Visible = false;

        dgvTransactions.Columns["ONOMA"].Width = 300;
        Transaction = null;
   }
   catch (Exception ex)
   {
        MessageBox.Show("Σφάλμα: " + ex.Message);
   }
}
```

Όπως φαίνεται και από τον πιο πάνω κώδικα, αρχικά δημιουργείται ένα Instance της κλάσης clsTransaction και με τη βοήθεια του καλείται η μέθοδος **«LoadByPerson()»** (βλ. Διαδικασία 18) για την εμφάνιση όλων των συναλλαγών του συγκεκριμένου χρήστη στο DataGridView. Στη συνέχεια κρύβονται μερικές μη επιθυμητές στήλες, ορίζεται επιθυμητό μέγεθος σε κάποιες άλλες και το αντικείμενο Transaction αδειάζει.

Διαδικασία 18: «Μέθοδος LoadByPerson() - Κλάση clsTransaction»

```
public void LoadByPerson(int PersonCode)
{
  try
  {
     cn = new SqlConnection(clsCommon.MyConnectionString());
     command = new SqlCommand();
     command.Connection = cn;
     command.CommandText =
"SELECT [Transaction].tra_TransactionID, Person.per_Name AS ONOMA,
[Transaction].tra_PersonCode,  [Transaction].tra_DateCreated AS [HMEP.
ΣΥΝΑΛΛΑΓΗΣ], " +
"dbo.[Transaction].tra_AmountPayable AS [ΟΛΙΚΟ ΠΟΣΟ],
dbo.[Transaction].tra_AmountRemaining AS ΥΠΟΛΟΙΠΟ,
dbo.[Transaction].tra_TransactionTypeCode, " +
"dbo.[Transaction].tra_TransactionStatusCode,
dbo.TransactionType.trt_Description, dbo.TransactionStatus.trs_Description, "
+
"dbo.[Transaction].tra_PaymentTermsCode,dbo.[Transaction].tra_PaymentMethodCo
de,tra_PaymentInDays, " +
"dbo.[Transaction].tra_Discount,dbo.[Transaction].tra_SubTotal,dbo.[Transacti
on].tra_TaxAmount, " +
"[Transaction].tra_InternalNotes, [Transaction].tra_ExternalNotes,
tra_DocumentNumber as 'AP.    ΤΙΜΟΛΟΓΙΟΥ',usr_UserName as ΠΩΛΗΤΗΣ,tra_EntryBy
" +
"FROM dbo.Person INNER JOIN " +
"dbo.[Transaction] ON dbo.Person.per_PersonID =
dbo.[Transaction].tra_PersonCode INNER JOIN Users on usr_UserID =
[Transaction].tra_EntryBy INNER JOIN " +
"dbo.TransactionType ON dbo.[Transaction].tra_TransactionTypeCode =
dbo.TransactionType.trt_TransactionTypeID INNER JOIN " +
"dbo.TransactionStatus ON dbo.[Transaction].tra_TransactionStatusCode =
dbo.TransactionStatus.trs_TransactionStatusID " +
"WHERE tra_PersonCode =" + PersonCode;

     dt = new DataTable("Transaction");
     da = new SqlDataAdapter();
     da.SelectCommand = command;
     cn.Open();
     da.Fill(dt);
     cn.Close();
     command.Dispose();
     da.Dispose();
  }
  catch (Exception ex)
  {
     throw ex;
  }
}
```

Η μέθοδος αυτή δέχεται σαν παράμετρο το ID ενός συγκεκριμένου ατόμου και με βάση αυτό αντλεί από την βάση όλες τις συναλλαγές (τιμολόγια, προσφορές) που αφορούν το συγκεκριμένο άτομο.

Διαδικασία 19: «Μέθοδος btnFindInvoice_Click()»

```
private void btnFindInvoice_Click(object sender, EventArgs e)
{
  try
  {
     frmMain main = new frmMain();
     main.btnTraSelect.Visible = true;
     switch (Convert.ToInt32(txtPaymentType.Text))
     {
        case (int)Enums.clsEnums.PaymentType.Customers:
              main.πωλήσειςToolStripMenuItem_Click_1(sender, e);
              break;
        case (int)Enums.clsEnums.PaymentType.Suppliers:
              main.αγορέςToolStripMenuItem_Click(sender, e);
              break;
      }
      main.ShowDialog();
      if (main.TransactionCode != -1)
      {
          txtTransactionCode.Text = main.TransactionCode.ToString();
          clsTransaction tra = new clsTransaction();
          tra.Load(main.TransactionCode);
          txtPaymentInvoiceDocumentNumber.Text = tra.DocumentNumber;
          txtPaymentPersonName.Text = Convert.ToString(main.PaymentPersonName);
          txtPayAmountRemaining.Text =
Convert.ToString(main.payTraAmountRemaining);
          txtPayAmount.Text = Convert.ToString(main.payTraAmountRemaining);
       }
       main.btnTraSelect.Visible = false;
       main.Dispose();
   }
   catch (Exception ex)
   {
      MessageBox.Show("Σφάλμα: " + ex.Message);
   }
}
```

Αρχικά, δημιουργείται ένα Instance της «frmMain». Το κουμπί **«Επιλογή»** (btnTraSelect) στο «tab Transactions» γίνεται ορατό (Visible=true) και με μια «switch statement» ελέγχεται σε ποιο τύπο πληρωμής βρίσκεται ο χρήστης, ώστε να εμφανιστούν στο «tab Transactions» τα κατάλληλα δεδομένα. Αφού επιλέξει ο χρήστης κάποιο τιμολόγιο και πατήσει το κουμπί **«Επιλογή»** (βλ. Διαδικασία 20) κάποια από τα στοιχεία του συγκεκριμένου τιμολογίου αποθηκεύονται σε μεταβλητές της frmMain και ο χρήστης μεταφέρεται και πάλι στο «tab Payment» . Στη συνέχεια δημιουργείται ένα αντικείμενο της κλάσης clsTransaction και καλείται η μέθοδος **Load()** με παράμετρο το ID του τιμολογίου που επιλέχθηκε, έτσι τα στοιχεία του τιμολογίου φορτώνονται στις μεταβλητές της κλάσης. Ακολούθως, κάποια από αυτά τα στοιχεία, αλλά και κάποια από τα στοιχεία που αποθηκεύτηκαν σε μεταβλητές της frmMain και είναι αναγκαία για την πληρωμή,

γίνονται set στα ανάλογα textbox του tab Payment και το κουμπί «Επιλογή» γίνεται και πάλι κρυφό.

Διαδικασία 20: «Μέθοδος btnTraSelect_Click()»

```
private void btnTraSelect_Click(object sender, EventArgs e)
{
   if (dgvTransactions.SelectedRows.Count == 1)
   {
       this.TransactionCode =
       Convert.ToInt32(dgvTransactions.SelectedRows[0].Cells["tra_Transaction
       ID"].Value);
       this.PaymentPersonName =
       Convert.ToString(dgvTransactions.SelectedRows[0].Cells["ΟΝΟΜΑ"].Value)
       ;
       this.payTraAmountRemaining =
       Convert.ToDecimal(dgvTransactions.SelectedRows[0].Cells["ΥΠΟΛΟΙΠΟ"].Va
       lue);

       flag = true;
       this.Close();
   }
   else
   {
      MessageBox.Show("Πρέπει πρώτα να επιλέξετε μια εγγραφή!");
   }
}
```

Πως παίρνουν τιμές τα combobox μέσα στις φόρμες

Πιο κάτω παρουσιάζεται ο κώδικας, ο οποίος εκτελείται για να εμφανιστούν οι διάφορες τιμές στο combobox με ένδειξη **«Μέθοδος Πληρωμής»** (tab Payments)

Διαδικασία 21: «Μέθοδος LoadCombosPayment()»

```
private void LoadCombosPayment()
{
   try
   {
       clsPaymentMethods pme = new clsPaymentMethods();
       pme.Load(-2);
       cmbPaymentMethod.DataSource = pme.dt;
       cmbPaymentMethod.DisplayMember = "ΜΕΘΟΔΟΣ ΠΛΗΡΩΜΗΣ";
       cmbPaymentMethod.ValueMember = "pme_PaymentMethodID";
       pme = null;
   }
   catch (Exception ex)
   {
       MessageBox.Show("Σφάλμα: " + ex.Message);
   }
}
```

Αρχικά δημιουργείται ένα αντικείμενο της κλάσης clsPaymentMethods, και με τη βοήθεια του, καλείται η μέθοδος Load() της κλάσης αυτής με παράμετρο «-2». Η σημασία της παραμέτρου «-2», είναι η ίδια όπως παρουσιάστηκε στη μέθοδο Load() της κλάσης clsperson (βλ. Διαδικασία 7). Τα δεδομένα που αντλούνται από τη βάση αποθηκεύονται σε ένα DataTable και στη συνέχεια στο DataSource του εν λόγω combobox. Ακολούθως, τίθενται ως περιεχόμενο του combobox, οι τιμές του πεδίου «ΜΕΘΟΔΟΣ ΠΛΗΡΩΜΗΣ» απ' όλες τις εγγραφές του πίνακα.

Διαδικασία 22: «Τμήμα κώδικα από το κουμπί "Αποθήκευση" – Tab Payments»

```
clsTransaction transaction = new clsTransaction();
transaction.Load(Payment.payTransactionCode);
transaction.AmountRemaining = transaction.AmountRemaining - Payment.payAmount;
if (transaction.AmountRemaining <= 0)
   transaction.TransactionStatusCode =
(int)Enums.clsEnums.TransactionStatus.Εξοφλημένο;
else
   transaction.TransactionStatusCode =
(int)Enums.clsEnums.TransactionStatus.Ανοιχτό;
 transaction.Save();
```

Αφού γίνει αποθήκευση μιας νέας ή υπάρχουσας πληρωμής, δημιουργείται ένα αντικείμενο της κλάσης «clsTransaction». Στη συνέχεια, καλείται η μέθοδος Load() της κλάσης αυτής με βάση τον αριθμό τιμολογίου για το οποίο έγινε η πληρωμή .

Αποτέλεσμα της διαδικασίας αυτής είναι να δοθούν στις μεταβλητές της κλάσης οι τιμές των στοιχείων του συγκεκριμένου τιμολογίου για περαιτέρω διαχείριση. Αφού γίνουν τα πιο πάνω, αφαιρείται από την τιμή της μεταβλητής AmountRemaining (Υπόλοιπο) το νέο ποσό πληρωμής και γίνεται έλεγχος:

Αν προέκυψε μηδενικό ή αρνητικό υπόλοιπο, τότε η μεταβλητή TransactionStatusCode (Κατάσταση τιμολογίου) παίρνει την τιμή **«Εξοφλημένο»,** αλλιώς την τιμή **«Ανοιχτό».**

Τέλος, καλείται η μέθοδος **«Save()»** της κλάσης clsTransaction για αποθήκευση των νέων δεδομένων.

Διαδικασία 23: «Μέθοδος LoadTreeViewStockCategories()»

```
private void LoadTreeViewStockCategories()
{
    clsStockCategory stockCategory = new clsStockCategory();
    stockCategory.Load(-2);

    trvStock.Nodes.Add("Ολα", "Ολα");
    for (int i = 1; i < stockCategory.dt.Rows.Count; i++)
    {
trvStock.Nodes[0].Nodes.Add(stockCategory.dt.Rows[i]["cat_CategoryID"].ToString
(), stockCategory.dt.Rows[i]["ΠΕΡΙΓΡΑΦΗ"].ToString());
    }
    trvStock.Nodes[0].ExpandAll();
}
```

Όταν ο χρήστης μεταφερθεί στο «Tab Stock», καλείται η πιο πάνω μέθοδος για εμφάνιση των κατηγοριών αποθεμάτων μέσα στο TreeView. Αρχικά, δημιουργείται ένα αντικείμενο της κλάσης «clsStockCategory» και καλείται η μέθοδος Load() της κλάσης αυτής. Τα δεδομένα που επιστρέφονται από την εκτέλεση του Query της μεθόδου Load() αποθηκεύονται σε ένα DataTable(dt).
Στη συνέχεια γίνεται «add» στο TreeView ένας κόμβος με το όνομα «Όλα», και με ένα «for statement» αντλούνται από το DataTable όλες οι κατηγορίες αποθεμάτων καθώς και το ID τους, τα οποία γίνονται με τη σειρά τους add στο TreeView.
Τέλος, με τη μέθοδο **ExpandAll()** το **TreeView** «ανοίγει» για να φαίνονται όλοι οι κόμβοι.

Διαδικασία 24: «Μέθοδος btnTransactionSearch_Click()»

```
private void btnTransactionSearch_Click(object sender, EventArgs e)
{
   try
   {
      clsTransaction searchTransaction = new clsTransaction();
      searchTransaction.TransactionTypeCode =
Convert.ToInt32(txtTransactionTypeCode.Text);

      if (btnTransactionSearch.Text == "Προβολή Όλων")
      {
          searchTransaction.Load(-2);
          dgvTransactions.DataSource= searchTransaction.dt;
          btnTransactionSearch.Text = "Εύρεση";
          txtSearchTransaction.Enabled = true;
      }
      else
      {
          searchTransaction.LoadByDocCode(txtSearchTransaction.Text);
          if (searchTransaction.dt.Rows.Count > 0)
          {
              dgvTransactions.DataSource = searchTransaction.dt;
              btnTransactionSearch.Text = "Προβολή Όλων";
              txtSearchTransaction.Enabled = false;
          }
           else
               MessageBox.Show("Δεν βρέθηκαν αποτελέσματα");
               txtSearchTransaction.Font = new Font(this.Font,
FontStyle.Italic);
               txtSearchTransaction.Text = "Αρ. Συναλλαγής";
       }
                searchTransaction = null;
     }
     catch
     {
         MessageBox.Show("Δε βρέθηκαν αποτελέσματα");
     }
}
```

Αφού πατηθεί το κουμπί δίπλα από το textbox με ένδειξη «Αρ. Συναλλαγής», καλείται η πιο πάνω μέθοδος. Αρχικά, δημιουργείται ένα αντικείμενο της κλάσης «clsTransaction» και με τη βοήθεια του ορίζεται στη μεταβλητή «TransactionTypeCode» ο τύπος της συναλλαγής. Στη συνέχεια, γίνεται έλεγχος του τι αναγράφεται στο κουμπί, ώστε να εκτελεστεί η κατάλληλη ενέργεια:

- Αν στο κουμπί αναγράφεται **«Εύρεση»**, τότε καλείται η μέθοδος LoadByDocCode() της κλάσης αυτής, η οποία δέχεται σαν παράμετρο τον αριθμό συναλλαγής που έγραψε ο χρήστης στο textbox. Με βάση τον αριθμό αυτό και τον τύπο της συναλλαγής εκτελείται ένα Query προς τη βάση για την εύρεση του συγκεκριμένου εγγράφου. Αν το έγγραφο υπάρχει, εμφανίζεται μόνο αυτό στο DataGridView, στο κουμπί από **«Εύρεση»** αναγράφεται **«Προβολή Όλων»** και το textbox γίνεται μη διαθέσιμο, αλλιώς το tab επαναφέρεται στην αρχική κατάσταση.

- Αν στο κουμπί αναγράφεται **«Προβολή όλων»** τότε καλείται η μέθοδος Load() της κλάσης «clsTransaction» για επανεμφάνιση όλων των συναλλαγών. Στο κουμπί από **«Προβολή Όλων»** αναγράφεται **«Εύρεση»** και το textbox με ένδειξη «Αρ. Συναλλαγής» γίνεται διαθέσιμο.

Διαδικασία 25: "Μέθοδος pnlTraPaid_Click()"

```
private void pnlTraPending_Click(object sender, EventArgs e)
{
      pnlTraPointer.Location = new Point(935, 112);
      bsTransaction.Filter = "tra_TransactionStatusCode=" +
      (int)Enums.clsEnums.TransactionStatus.Ανοιχτό;

      dgvTransactions.AlternatingRowsDefaultCellStyle.BackColor =
      Color.LightGreen;
      dgvTransactions.ClearSelection();
      ClearFieldsTransactions();
}
```

Αρχικά, μόλις επιλέξει ο χρήστης το συγκεκριμένο **Panel** (χρώματος πράσινο), το Panel που τέθηκε ως **pointer** μεταφέρεται δίπλα από αυτό, για να ξέρει ο χρήστης ποιο Panel είναι επιλεγμένο.

Στη συνέχεια, με τη βοήθεια ενός αντικείμενου **τύπου BindingSource** φιλτράρονται τα δεδομένα του DatagridView, και εμφανίζονται μόνο αυτά που έχουν **Status «Ανοιχτό»,** δηλαδή μόνο οι συναλλαγές οι οποίες εκκρεμούν.

Ακολούθως τίθεται στο χρώμα των γραμμών του DataGridView, το ίδιο χρώμα με αυτό του Panel που επιλέχθηκε.

Διαδικασία 26: «Μέθοδος GenerateDocumentNumber() - Κλάση clsTransaction»

```
private string GenerateDocumentNumber()
{
  string LastDocNum = "";
  string DocumentCode = "";

   switch (this.TransactionTypeCode)
   {
    case (Int32)(Enums.clsEnums.TransactionType.Sale):
        DocumentCode += "RTL-";
        break;
    case (Int32)(Enums.clsEnums.TransactionType.Quotation):
        DocumentCode += "QTN-";
        break;

   }
   string qry = "SELECT top 1 tra_DocumentNumber" +
   " FROM [TRANSACTION]" +
   " WHERE tra_TransactionTypeCode = " + this.TransactionTypeCode +
   " ORDER BY tra_TransactionID desc";

   SqlCommand cmd = new SqlCommand(qry, cn);
   cn.Open();
   try
   {
   string str1 = (String)cmd.ExecuteScalar();
   string[] strs = str1.Split('-');
   LastDocNum = strs[1];
   }
   catch
   {
     cn.Close();
     return DocumentCode + "1";
   }
   cn.Close();
   int intLastDocNum = Convert.ToInt32(LastDocNum) + 1;
   DocumentCode += intLastDocNum.ToString();

return DocumentCode;
}
```

Κατά τη διαδικασία αποθήκευσης μιας νέας συναλλαγής (Πώλησης ή Προσφοράς), καλείται η πιο πάνω μέθοδος, η οποία ανάλογα με τον τύπο της συναλλαγής, δημιουργεί ένα μοναδικό αριθμό εγγράφου. Αρχικά δηλώνονται δύο μεταβλητές τύπου string (LastDocNum και DocumentCode). Με ένα **«switch statement»** ελέγχεται ο τύπος της συναλλαγής που δέχεται ως παράμετρο και βασικά το είδος της συναλλαγής για την οποία θα δημιουργηθεί ο αριθμός εγγράφου.

Αν η συναλλαγή αυτή είναι πώληση, στην μεταβλητή **«DocumentCode»** αποθηκεύεται το string **«RTL-»,** αλλιώς το string **«QTN-».** Στην συνέχεια, με ένα Query που εκτελείται προς τη βάση, αποθηκεύεται σε μια μεταβλητή τύπου string(str1) ο τελευταίος αριθμός εγγράφου που αποθηκεύτηκε στη βάση για τον συγκεκριμένο τύπο συναλλαγής. Ακολούθως, το string αυτό γίνεται split

(χωρίζεται), σε 2 μέρη και αποθηκεύεται σε ένα πίνακα. Εν συνεχεία, στη μεταβλητή **LastDocNum** αποθηκεύεται το string που βρισκόταν δεξιά από τον χαρακτήρα **"-"** και αφού μετατραπεί σε integer, προστίθεται σε αυτόν ο αριθμός ένα (+1). Μετά από την πιο πάνω διαδικασία μετατρέπεται και πάλι σε string και προστίθεται στο περιεχόμενο της μεταβλητής **DocumentCode**.

Έτσι επιστρέφεται ο αριθμός εγγράφου που μόλις δημιουργήθηκε. Σε περίπτωση που υπάρξει σφάλμα, σημαίνει ότι είναι η πρώτη καταχώρηση που θα γίνει στον πίνακα γι'αυτόν τον τύπο συναλλαγής, οπότε επιστρέφεται το περιεχόμενο της string μεταβλητής **DocumentCode**

Υπολογισμός του ποσού πληρωμής σε ένα τιμολόγιο

Διαδικασία 27: «Μέθοδος CalculateAmounts()»

```
private void CalculateAmounts()
{
   try
   {
      decimal dclVatAmount = 0.00M;
      decimal dclSubTotal = 0.00M;
      decimal dclDiscount;
      decimal dclAmountPayable = 0.00M;

      if (txtTraDetDiscount.Text == "")
          dclDiscount = 0.00M;
      else
          dclDiscount = Convert.ToDecimal(txtTraDetDiscount.Text);

      for (int i = 0; i < dgvTransactionDetails.Rows.Count; i++)
      {
          dclSubTotal = dclSubTotal +
          Convert.ToDecimal(dgvTransactionDetails.Rows[i].Cells["ΣΥΝΓΡΑΜΜΗΣ"].Value);
      }
      dclVatAmount = (dclSubTotal - dclDiscount) * 0.15m;
      dclAmountPayable = (dclSubTotal - dclDiscount) + dclVatAmount;

      txtTraDetAmountPayable.Text = dclAmountPayable.ToString("F");
      txtTraDetDiscount.Text = dclDiscount.ToString("F");
      txtTraDetSubTotal.Text = dclSubTotal.ToString("F");
      txtTraDetVAT.Text = dclVatAmount.ToString("F");
   }
   catch (Exception ex)
   {
       MessageBox.Show("Σφάλμα: " + ex.Message);
   }
}
```

Η πιο πάνω μέθοδος καλείται για τον υπολογισμό όλων των ποσών μιας συναλλαγής (τιμολόγιο, προσφορά), είτε κατά την εισαγωγή μιας εγγραφής στο DataGridView της φόρμας frmTranastionDetails, είτε κατά την αφαίρεση μιας εγγραφής από αυτό, είτε κατά την αλλαγή του ποσού στο textbox με ένδειξη «Έκπτωση» .

Αρχικά γίνεται έλεγχος του textbox με ένδειξη «Έκπτωση» και αν δεν είναι κενό το περιεχόμενο του μετατρέπεται σε decimal και αποθηκεύεται στη μεταβλητή **«dclDiscount».** Στη συνέχεια, με ένα **«for statement»** προστίθενται όλα τα ποσά που βρίσκονται κάτω από την στήλη με ένδειξη **«ΣΥΝ. ΓΡΑΜΜΗΣ»** και αποθηκεύονται στη μεταβλητή **«dclSubTotal».** Αφού αφαιρεθεί το ποσό της έκπτωσης, από το ποσό αυτό, υπολογίζεται το **«Φ.Π.Α»** και αποθηκεύεται στη μεταβλητή **«dclVatAmount».**

Τέλος, αφού υπολογιστεί το ποσό που πρέπει να πληρωθεί **(dclAmountPayable),** όλες οι τιμές γίνονται set στα αντίστοιχα textbox.

Διαδικασία 28: «Μέθοδος Restock()-Κλάση clsStock»

```
public void Restock(int TransactionID)
{
  if (TransactionID > 0)
  {
    try
    {
       Classes.clsTransaction tra = new clsTransaction();
       tra.Load(TransactionID);
       cn = new SqlConnection(clsCommon.MyConnectionString());
       DataTable dt = new DataTable();
       string qry = "SELECT trd_ProductCode, trd_ProductQuantity FROM
        [TransactionDetails] WHERE  trd_TransactionCode =" + TransactionID;
       SqlDataAdapter da = new SqlDataAdapter(qry, cn);
       da.Fill(dt);

       for (int i = 0; i < dt.Rows.Count; i++)
       {
          decimal NewQty = 0.00M;
          DataTable dt1 = new DataTable();
          int stockID = Convert.ToInt32(dt.Rows[i]["trd_ProductCode"]);
          decimal TransactionDetailsQty =
Convert.ToDecimal(dt.Rows[i]["trd_ProductQuantity"]);
          string qry1 = "SELECT sto_ProductQuantity FROM Stock WHERE sto_StockID =" +
stockID;
          SqlDataAdapter da1 = new SqlDataAdapter(qry1, cn);
          da1.Fill(dt1);
          if (dt1.Rows.Count > 0)
          {
             decimal CurrentQty = Convert.ToDecimal(dt1.Rows[0]["sto_ProductQuantity"]);
             if (tra.TransactionTypeCode ==
Convert.ToInt32(Enums.clsEnums.TransactionType.Sale))
                 NewQty = CurrentQty + TransactionDetailsQty;
             else if (tra.TransactionTypeCode ==
Convert.ToInt32(Enums.clsEnums.TransactionType.Purchase))
                 NewQty = CurrentQty - TransactionDetailsQty;
          }
        clsStock sto = new clsStock();
        sto.Load(stockID);
        sto.stoQuantity = NewQty;
        sto.Save();
        sto = null;
       }
       tra = null;
       cn.Dispose();
       dt.Dispose();
       da.Dispose();
    }
    catch (Exception ex)
    {
```

Κατά την αποθήκευση ενός τιμολογίου καλείται η πιο πάνω μέθοδος. Αρχικά, δημιουργείται ένα αντικείμενο της κλάσης clsTransaction και με τη χρήση του καλείται η μέθοδος Load() της κλάσης αυτής με παράμετρο το ID του συγκεκριμένου τιμολογίου. Με ένα Query, που εκτελείται, αντλούνται από τον πίνακα TransactionDetails οι κωδικοί και οι ποσότητες όλων των αποθεμάτων, που αφορούν το συγκεκριμένο τιμολόγιο και αποθηκεύονται σε ένα DataTable (dt). Στη συνέχεια, με ένα «for statement» αποθηκεύεται ο κωδικός (stockID) και η ποσότητα (TransactionDetailsQty) του αποθέματος σε μεταβλητές, για κάθε εγγραφή του DataTable(dt). Ακολούθως, εκτελείται ένα νέο Query με βάση τον κωδικό αυτό (stockID), για να ανακτηθεί η τρέχουσα ποσότητα του συγκεκριμένου προϊόντος, η οποία είναι αποθηκευμένη στη βάση, και το αποτέλεσμα αποθηκεύεται σε μια μεταβλητή (CurrentQty). Μετά από την πιο πάνω διαδικασία γίνεται έλεγχος και αν το τιμολόγιο προέκυψε από αγορά, τότε στην τρέχουσα ποσότητα του αποθέματος προστίθεται η ποσότητα που βρίσκεται στο τιμολόγιο. Σε αντίθετη περίπτωση, αν το τιμολόγιο προέκυψε από πώληση, τότε από την τρέχουσα ποσότητα του αποθέματος αφαιρείται η ποσότητα που βρίσκεται στο τιμολόγιο.

Τέλος, αφού γίνει Load() το συγκεκριμένο προϊόν, δίνεται στη μεταβλητή **«stoQuantity»** η νέα ποσότητα του προϊόντος και καλείται η μέθοδος Save() για αποθήκευση των αλλαγών.

Ευρετήριο Εικόνων

Ευρετήριο διαδικασιών κώδικα