ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΠΤΥΞΗ ΕΚΠΑΙΔΕΥΤΙΚΟΥ ΕΡΓΑΛΕΙΟΥ ΒΑΣΙΣΜΕΝΟΥ ΣΤΟΝ ΑΛΓΟΡΙΘΜΟ ΤΗΣ ΚΑΝΟΝΙΚΟΠΟΙΗΣΗΣ ΓΙΑ ΤΗ ΔΗΜΙΟΥΡΓΙΑ ΣΧΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Διπλωματική Εργασία

του

Χρήστου Στεφανίδη

Θεσσαλονίκη, 5/2016

ΑΝΑΠΤΥΞΗ ΕΚΠΑΙΔΕΥΤΙΚΟΥ ΕΡΓΑΛΕΙΟΥ
ΒΑΣΙΣΜΕΝΟΥ ΣΤΟΝ ΑΛΓΟΡΙΘΜΟ ΤΗΣ ΚΑΝΟΝΙΚΟΠΟΙΗΣΗΣ
ΓΙΑ ΤΗ ΔΗΜΙΟΥΡΓΙΑ ΣΧΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Χρήστος Στεφανίδης

Πτυχίο Εμπορίας & Διαφήμισης, ΑΤΕΙ Θεσσαλονίκης, 2004

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπουσα Καθηγήτρια
Γεωργία Κολωνιάρη

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21/06/2016

Γεώργιος Ευαγγελίδης	Γεωργία Κολωνιάρη	Στυλιανός Ξυνόγαλος
.....	.....	.....

Χρήστος Στεφανίδης

.....

Περίληψη

Η παρούσα διπλωματική εργασία έχει ως αντικείμενο την ανάπτυξη λογισμικού εκπαιδευτικού εργαλείου (στο εξής πρόγραμμα) το οποίο θα αυτοματοποιεί τη διαδικασία της κανονικοποίησης πινάκων σε BCNF μορφή. Ο χρήστης του προγράμματος αρχικά δημιουργεί ένα σχήμα βάσης δεδομένων, στο οποίο προσθέτει γνωρίσματα και δημιουργεί συναρτησιακές εξαρτήσεις με αυτά. Εν συνεχεία, με τους κατάλληλους αλγόριθμους οι οποίοι περιγράφονται στην εργασία, το πρόγραμμα εντοπίζει τα υποψήφια κλειδιά του σχήματος της βάσης δεδομένων και πραγματοποιεί διάσπαση πινάκων, κανονικοποιώντας το αρχικό σχήμα σε μικρότερους, κανονικούς BCNF πίνακες. Με αυτό τον τρόπο απλουστεύουμε τη δημιουργία του σχήματος μιας βάσης δεδομένων και περιορίζουμε σημαντικά τα λάθη από ανωμαλίες στην εισαγωγή, επεξεργασία και διαγραφή των εγγραφών της.

Η ανάπτυξη του προγράμματος βασίστηκε σε αλγόριθμους που εφαρμόζουν την υπάρχουσα θεωρία των συναρτησιακών εξαρτήσεων και της κανονικοποίησης, η οποία παρουσιάζεται αναλυτικά στην παρούσα εργασία.

Το πρόγραμμα δημιουργήθηκε στην αντικειμενοστραφή γλώσσα C#, στο περιβάλλον προγραμματισμού Visual Studio 2015 Express της Microsoft. Στο τελευταίο κεφάλαιο της εργασίας αναλύονται οι σημαντικότερες κλάσεις και μέθοδοί του. Στο παράρτημα παρατίθεται αυτούσιος ο πηγαίος κώδικας του προγράμματος, έτσι ώστε να είναι στη διάθεση κάθε ενδιαφερόμενου.

Λέξεις Κλειδιά: Πρόγραμμα, βάσεις δεδομένων, συναρτησιακή εξάρτηση, κλειδί, κανονικοποίηση, BCNF

Abstract

This thesis has as its object the development of an educational tool (program) which automates the normalization process in BCNF form. The program user initially creates a database scheme, adding features and functional dependencies therewith. Subsequently, with appropriate algorithms, which are described in this paper, the program identifies the candidate keys of the database scheme tables and performs normalization of the original scheme into smaller, BCNF normal tables. In this way we can simplify the creation of a database scheme and significantly limit the anomalies and mistakes of irregularities in the import, editing and deletion of records.

The development of the program was based on algorithms that implement the existing theory of functional dependencies and normalization, analyzed in this paper.

The program was created in the object-oriented language C#, using Microsoft Visual Studio Express 2015. In the last chapter the major classes and their methods are analysed. Finally, in the annex the source code is presented, available to everyone.

Keywords: Program, database, functional dependency, key, normalization, BCNF

Πρόλογος – Ευχαριστίες

Στην ως τώρα επαγγελματική μου καριέρα δεν ήταν λίγες οι φορές εκείνες που χρειάστηκε να "σκαρφιστώ" τη δημιουργία ενός σχήματος βάσης δεδομένων, κάτι που ελλείψει γνώσεων το έκανα διαισθητικά. Ώσπου επιτέλους πήρα τις απαραίτητες κατευθύνσεις που χρειαζόμουν, καθώς και τα ερεθίσματα για περαιτέρω μελέτη, στο μάθημα "Βάσεις Δεδομένων" του 1ου εξαμήνου του μεταπτυχιακού, από τους καθηγητές κ.κ. Γεώργιο Ευαγγελίδη και Γεωργία Κολωνιάρη, με χρήσιμες γνώσεις για τη δημιουργία μοντέλου οντοτήτων - συσχετίσεων, για τη μετατροπή του σε λογικό σχήμα, για τα σχεσιακά μοντέλα και την κανονικοποίηση.

Θέλω να ευχαριστήσω πρωτίστως την επιβλέπουσα καθηγήτρια της διπλωματικής εργασίας μου, κα Γεωργία Κολωνιάρη, που μοιράστηκε μαζί μου τις εξαιρετικές γνώσεις της στο αντικείμενο των βάσεων δεδομένων, για τη σπουδαία συνεργασία της στην ανάπτυξη του προγράμματος και για την υπομονή της στις ατελείωτες απορίες μου.

Ευχαριστώ, επίσης, και τους αγαπητούς καθηγητές του τμήματος Πληροφορικής, κ. Αλέξανδρο Χατζηγεωργίου και κ. Στέλιο Ξυνόγαλο για την πρόθυμη ανταπόκρισή τους σε κάθε είδους απορία μου. Κυρίως, όμως, για τον τρόπο με τον οποίο προσέγγισαν την παρουσίαση των μαθημάτων "Αντικειμενοστρεφής Ανάπτυξη Λογισμικού (Java)" και "Προγραμματισμός Παιχνιδιών Σοβαρού Σκοπού (C#)", κάτι το οποίο κέντρισε το ενδιαφέρον μου για περαιτέρω εμβάθυνση στον υπέροχο κόσμο του προγραμματισμού.

Ευχαριστώ τους φίλους μου, που όλο αυτό τον καιρό που ετοίμαζα την εργασία, μου συμπαραστάθηκαν και μου έδωσαν ιδέες για το πρόγραμμα. Ιδιαιτέρως ευχαριστώ τον καλό φίλο και συνάδελφο στο μεταπτυχιακό, Δημήτρη Σουκούλη, που είχε την καλοσύνη να τεστάρει το πρόγραμμα στα διάφορα στάδια της ανάπτυξής του, πραγματοποιώντας την μία "πυρηνική" διάσπαση μετά την άλλη.

Τέλος, ευχαριστώ μέσα από την καρδιά μου τους γονείς μου, Γιώργο και Μάρθα, για όλη τους την υποστήριξη καθ' όλη τη διάρκεια των μεταπτυχιακών σπουδών μου.

Περιεχόμενα

1 Εισαγωγή 1

1.1 Πρόβλημα - Σημαντικότητα του θέματος 1

1.2 Σκοπός - Στόχοι 2

1.3 Διάρθρωση της εργασίας 3

2 Βάσεις δεδομένων 4

2.1 Βασικές έννοιες 4

2.2 Ιδεατό σχήμα 5

2.3 Λογικό σχήμα 7

2.4 Παράδειγμα δημιουργίας σχήματος με το μοντέλο Οντοτήτων -Συσχετίσεων 9

2.4.1 Ανάλυση απαιτήσεων - περιγραφή βάσης δεδομένων....9

2.4.2 Σενάρια χρήσης της βάσης δεδομένων....10

2.4.3 Το ιδεατό σχήμα....12

2.4.4 Το λογικό σχήμα....17

2.4.5 Σύνοψη....19

3 Κανονικοποίηση 21

3.1 Εισαγωγή 21

3.2 Συναρτησιακές εξαρτήσεις 21

3.2.1 Γενικά για τις συναρτησιακές εξαρτήσεις....21

3.2.2 Τετριμμένες συναρτησιακές εξαρτήσεις....23

3.2.3 Αξιώματα του Armstrong....23

3.2.4 Προσδιορισμός κλειδιών....24

3.2.5 Συνδυαστική για τον εντοπισμό κλειδιών....26

3.2.6 Πρώτες κανονικές μορφές....28

3.2.7 Κανονική μορφή Boyce - Codd (BCNF)....30

3.3 Παραδείγματα δημιουργίας σχήματος με κανονικοποίηση 34

3.3.1 Παράδειγμα 1ο: Βάση δεδομένων για χρηματιστηριακή ενημέρωση....34

3.3.2 Παράδειγμα 2ο: Βάση δεδομένων για εργαστήριο ζαχαροπλαστικής....41

4 Ανάπτυξη του προγράμματος 44

4.1 Εισαγωγή 44

4.2 Λειτουργίες 44

4.3 Τεχνικές προδιαγραφές 45

4.4 Έλεγχος 46

4.5 Εγκατάσταση του προγράμματος 46
4.6 Εκκίνηση προγράμματος 47
4.7 Νέο σχήμα - άνοιγμα σχήματος 47
4.8 Δημιουργία - επεξεργασία γνωρίσματος 50
4.9 Δημιουργία – επεξεργασία συναρτησιακής εξάρτησης 52
4.10 Προσδιορισμός συναρτησιακής εξάρτησης ως τετριμμένης 55
4.11 Εγκλεισμός γνωρισμάτων 56
4.12 Εύρεση κλειδιών 58
4.13 Η κλάση Key 61
4.14 Διάσπαση πινάκων και η κλάση Relation 62
4.15 Ρυθμίσεις προγράμματος 65
5 Επίλογος 66
5.1 Μελλοντικές επεκτάσεις 66

Κατάλογος Εικόνων

Εικόνα 1: Το ιδεατό σχήμα της βάσης δεδομένων.....14
Εικόνα 2: Το σχήμα της χρηματιστηριακής ενημέρωσης.....38
Εικόνα 3: Η κεντρική οθόνη του προγράμματος.....48
Εικόνα 4: Δημιουργία νέου γνωρίσματος.....50
Εικόνα 5: Δημιουργία συναρτησιακής εξάρτησης.....53
Εικόνα 6: Ένα σύνολο τετριμμένων συναρτησιακών εξαρτήσεων.....55
Εικόνα 7: Επιλογή γνωρισμάτων για υπολογισμό εγκλεισμού.....56
Εικόνα 8: Προβολή διαδικασίας υπολογισμού εγκλεισμού.....56
Εικόνα 9: Προβολή αποτελεσμάτων εύρεσης κλειδιών.....58
Εικόνα 10: Φόρμα σταδιακής διάσπασης σε BCNF.....62

Κατάλογος Αποσπασμάτων Κώδικα

Απόσπασμα Κώδικα 1: Η μέθοδος Main()....47
Απόσπασμα Κώδικα 2: Η μέθοδος scLoad()....49
Απόσπασμα Κώδικα 3: Η μέθοδος Create()....51
Απόσπασμα Κώδικα 4: Η μέθοδος AttrCreate()....52
Απόσπασμα Κώδικα 5: Η μέθοδος FDCreate()....54
Απόσπασμα Κώδικα 6: Η μέθοδος ToString() της κλάσης FD....54
Απόσπασμα Κώδικα 7: Προσδιορισμός τετριμμένης συναρτησιακής εξάρτησης....55
Απόσπασμα Κώδικα 8: Η μέθοδος attrClosure()....57
Απόσπασμα Κώδικα 9: Η μέθοδος AttrBinarySelection()....59
Απόσπασμα Κώδικα 10: Η μέθοδος KeysGets()....61
Απόσπασμα Κώδικα 11: Η μέθοδος AddToKey()....61
Απόσπασμα Κώδικα 12: Η μέθοδος ToString() της κλάσης Key....61
Απόσπασμα Κώδικα 13: Η μέθοδος ToString() της κλάσης Relation....63
Απόσπασμα Κώδικα 14: Η μέθοδος DecomposeInSteps()....65

Συμβολισμοί

♠: Χαρακτηρίζει ένα γνώρισμα ως κλειδί (στην εργασία και το πρόγραμμα)

1NF, 2NF, 3NF, 4NF: 1st, 2nd, 3rd, 4th Normal Forms (4 πρώτες κανονικές μορφές)

BCNF: Boyce - Codd Normal Form (κανονική μορφή Boyce - Codd)

CLR: Common Language Runtime

DBMS: Database Management System (Σύστημα Διαχείρισης Βάσεων Δεδομένων)

DDL: Data Definition Language (υποσύνολο εντολών SQL)

FCL: Framework Class Library (βιβλιοθήκη κλάσεων)

FD: Functional Dependency (συναρτησιακή εξάρτηση)

FTSE: Σταθμισμένος χρηματιστηριακός δείκτης επιλεγμένων μετοχών

GUI: Graphics User Interface (περιβάλλον διεπαφής χρήστη)

I/O: Input - Output (είσοδος - έξοδος σε hardware και software)

LINQ: Language - Integrated Query (γλώσσα ερωτημάτων για βάσεις δεδομένων)

NF: Normal Form (κανονική μορφή)

PDF: Portable Document Format (τύπος αρχείου απεικόνισης εγγράφων)

SQL: Structured Query Language (δομημένη γλώσσα ερωτημάτων για βάσεις δεδομένων)

ΒΔ: Βάση Δεδομένων

ΓΔ: Γενικός Δείκτης του Χρηματιστηρίου Αθηνών

ΝΠΔΔ: Νομικό Πρόσωπο Δημοσίου Δικαίου

ΟΣ: Μοντέλο Οντοτήτων - Συσχετίσεων

ΧΑ: Χρηματιστήριο Αθηνών

1 Εισαγωγή

1.1 Πρόβλημα - Σημαντικότητα του θέματος

Η διαδικασία της κανονικοποίησης είναι η εναλλακτική προσέγγιση στη δημιουργία ενός σχήματος βάσης δεδομένων. Η πρώτη προσέγγιση είναι η "εκ των άνω προς τα κάτω" όπου, με τη βοήθεια του μοντέλου Οντοτήτων - Συσχετίσεων του Chen (1976), μπορούμε να δημιουργήσουμε αρχικά ένα ιδεατό σχήμα αφού έχουμε αναλύσει τις απαιτήσεις του όλου συστήματος της βάσης δεδομένων. Στη συνέχεια, το ιδεατό σχήμα "μεταφράζεται" στο λογικό σχήμα. Σε αυτό το επίπεδο, χρησιμοποιώντας το σχεσιακό μοντέλο του Codd (1970), οι οντότητες και οι συσχετίσεις, εφαρμόζοντας κι εδώ έναν συγκεκριμένο αλγόριθμο, μετατρέπονται σε σχεσιακούς πίνακες με γνωρίσματα και κλειδιά. Έτσι, μπορούμε μετά να προχωρήσουμε στο τελικό, φυσικό σχήμα, με τη βοήθεια της γλώσσας SQL, σε κάποιο από τα γνωστά συστήματα διαχείρισης βάσεων δεδομένων που θα επιλέξουμε. Η προσέγγιση αυτή βασίζεται κυρίως στη διαίσθηση και την εμπειρία, οπότε απαιτούνται καλές γνώσεις στο αντικείμενο του σχεδιασμού του σχήματος μιας βάσης δεδομένων.

Ωστόσο, η όλη σχεδίαση του ιδεατού σχήματος με το μοντέλο Οντοτήτων - Συσχετίσεων μπορεί να είναι μια χρονοβόρα και απαιτητική διαδικασία όπου πρέπει να αναλυθούν σε βάθος όλες οι απαιτήσεις της βάσης δεδομένων. Μια κακή σχεδίαση του ιδεατού μοντέλου είναι σχεδόν σίγουρο ότι θα οδηγήσει αργότερα, κατά τη δημιουργία του φυσικού σχήματος της βάσης δεδομένων, σε μη επιθυμητή επανάληψη πληροφοριών, καθώς και σε αδυναμία αντιπροσώπευσης κάποιων άλλων. Επιπλέον, δεν είναι εφικτή η αναπαράσταση ορισμένων περιορισμών στο μοντέλο των Οντοτήτων - Συσχετίσεων. Ένα κακό σχεδιασμένο σχήμα με μεγάλους πίνακες οδηγεί σε χρονοβόρες και εκτεταμένες αναζητήσεις, ακόμα και για λίγες "γραμμές" πληροφοριών και οι διαγραφές σε ορισμένες εγγραφές μπορεί να προκαλέσουν ανεπιθύμητη απαλοιφή χρήσιμων δεδομένων. Επίσης, αν χρειαστεί μελλοντικά να γίνουν αναβαθμίσεις (updates) με νέα πεδία, αυτές μπορεί να απαιτούν πολύ χρόνο και να είναι δαπανηρές.

Από την άλλη πλευρά, η κανονικοποίηση δημιουργεί το τελικό σχήμα της βάσης δεδομένων προσεγγίζοντας την όλη διαδικασία "από κάτω προς τα πάνω", βασιζόμενη περισσότερο σε συγκεκριμένο αλγόριθμο, παρά στη διαίσθηση και την εμπειρία που απαιτεί η δημιουργία του ιδεατού σχήματος με το μοντέλο των Οντοτήτων - Συσχετίσεων. Το βασικό σκεπτικό της είναι η διάσπαση ενός universal πίνακα γνωρισμάτων σε μικρότερους και πιο λειτουργικούς, απαλείφοντας έτσι την

επαναληψιμότητα και τα ενδεχόμενα λάθη εισαγωγής, επεξεργασίας και διαγραφής δεδομένων. Εντούτοις, η δυσκολία των απαιτούμενων βημάτων κατά τη διαδικασία της κανονικοποίησης αποθαρρύνει συχνά τους ενασχολούμενους. Επιπλέον, η δημιουργία ενός σχήματος βάσης δεδομένων με μεγάλο πλήθος γνωρισμάτων μπορεί να καταστεί χρονοβόρα κατά την εφαρμογή των σχετικών αλγορίθμων της κανονικοποίησης.

Στα πλαίσια λοιπόν αυτά δημιουργήθηκε πρόγραμμα εκπαιδευτικής φύσης και συνάμα καταγράφηκε, με απλά λόγια, η διαδικασία της κανονικοποίησης. Με αυτό τον τρόπο, η παρούσα διπλωματική εργασία στο σύνολό της δύναται να αποτελέσει ένα βοήθημα για φοιτητές και στελέχη επιχειρήσεων που εργάζονται στο αντικείμενο βάσεων δεδομένων. Πέραν όμως της εκπαιδευτικής φύσης του προγράμματος, υπάρχει η δυνατότητα και για αυτοματοποιημένη διάσπαση πινάκων, όπου απλώς ο χρήστης του προγράμματος παίρνει τα τελικά αποτελέσματα των διασπάσεων, χωρίς να τον ενδιαφέρει να εκπαιδευτεί στην όλη διαδικασία, καθώς αυτή "τρέχει" στο παρασκήνιο.

1.2 Σκοπός - Στόχοι

Ο κύριος σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη ενός εύχρηστου και πρακτικού προγράμματος - εκπαιδευτικού εργαλείου, το οποίο θα μπορεί να χρησιμοποιείται μελλοντικά από καθηγητές και φοιτητές. Επιπλέον, το πρόγραμμα θα μπορεί να βοηθήσει και άλλους ενδιαφερόμενους, κι έξω από τον πανεπιστημιακό χώρο, που ενδεχομένως θα θέλουν να δημιουργήσουν ένα σχεσιακό σχήμα βάσης δεδομένων, ακολουθώντας συγκεκριμένο αυτοματοποιημένο, αλγοριθμικό τρόπο (κανονικοποίηση) κι όχι διαισθητικά όπως γίνεται στις περισσότερες περιπτώσεις. Καθώς οι διασπάσεις των πινάκων στο πρόγραμμα γίνονται τόσο καθετοποιημένα, όσο και σταδιακά, εξυπηρετούνται οι σκοποί τόσο της χρήσης του προγράμματος καθαρά και μόνο για δημιουργία σχήματος βάσης δεδομένων, όσο και της εκπαιδευτικής φύσης του με την παρουσίαση σταδιακά των βημάτων και του σκεπτικού που βρίσκεται πίσω από κάθε μια διάσπαση πίνακα.

Στόχος επίσης της εργασίας είναι να συγκεντρωθούν και να παρουσιαστούν με τρόπο απλό και σαφή οι βασικές αρχές της κανονικοποίησης, το σκεπτικό με το οποίο δημιουργούμε συναρτησιακές εξαρτήσεις και πως με τη χρήση τους προχωράμε στη διάσπαση των πινάκων σε κανονική μορφή. Η εργασία και το πρόγραμμα εστιάζουν περισσότερο στην κανονική μορφή που είναι γνωστή ως BCNF, ενώ γίνεται μια σύντομη αναφορά και στις 3 πρώτες κανονικές μορφές. Επιπλέον, παρουσιάζεται και ένα

εκτεταμένο, αναλυτικό παράδειγμα δημιουργίας σχήματος βάσης δεδομένων με τις δύο προσεγγίσεις: με τη χρήση του μοντέλου Οντοτήτων - Συσχετίσεων του Chen και με τη μέθοδο της κανονικοποίησης.

Εκτός από το τελικό, ολοκληρωμένο πρόγραμμα, θα διατεθεί, για ελεύθερη χρήση, ο πηγαίος κώδικάς του, έτσι ώστε να δοθεί η ευκαιρία, σε κάθε ενδιαφερόμενο να επεκτείνει περαιτέρω την εργασία αυτή και ενδεχομένως και το πρόγραμμα.

1.3 Διάρθρωση της εργασίας

Μετά το παρόν κεφάλαιο 1 της εισαγωγής, στο κεφάλαιο 2 παρουσιάζονται συνοπτικά οι βασικές έννοιες των βάσεων δεδομένων και των τεσσάρων επιπέδων αφαίρεσης. Καθότι όμως το αντικείμενο μελέτης είναι το ιδεατό και το λογικό επίπεδο, η διπλωματική εργασία εστιάζεται αποκλειστικά σε αυτά. Ειδικότερα, περιγράφεται σε γενικές γραμμές η δημιουργία ενός σχήματος βάσης δεδομένων με το μοντέλο Οντοτήτων - Συσχετίσεων και εν συνεχεία η μετατροπή του σε σχεσιακούς πίνακες του λογικού επιπέδου. Επιπλέον, παρατίθεται ένα αναλυτικό παράδειγμα δημιουργίας σχήματος βάσεων δεδομένων με βάση αυτό το μοντέλο.

Στο κεφάλαιο 3 αναλύεται η διαδικασία της κανονικοποίησης σε κανονική μορφή πινάκων BCNF, αφού πρώτα επεξηγούνται έννοιες όπως οι συναρτησιακές εξαρτήσεις, τα κλειδιά, ο εγκλεισμός των γνωρισμάτων και οι κανονικές μορφές. Επίσης, παρατίθεται ένα παράδειγμα κανονικοποίησης ενός σχήματος βάσης δεδομένων με τη βοήθεια του προγράμματος που αναπτύχθηκε.

Στο κεφάλαιο 4 παρουσιάζονται οι βασικές προδιαγραφές και απαιτήσεις του προγράμματος - εκπαιδευτικού εργαλείου. Επίσης, παρουσιάζεται και σχολιάζεται σημαντικό μέρος του πηγαίου κώδικα και των κυριότερων κλάσεων και μεθόδων του.

Τέλος, στο παράρτημα παρατίθενται αυτούσια οι σύντομες οδηγίες χρήσης (manual), καθώς επίσης ο κώδικας του προγράμματος όπως είχε διαμορφωθεί κατά την ολοκλήρωση της συγγραφής της διπλωματικής εργασίας.

2 Βάσεις δεδομένων

2.1 Βασικές έννοιες

Με τον όρο δεδομένα (data) εννοούμε τα δεδομένα που γνωρίζουμε και μπορούμε να καταγράψουμε. Για να είναι τα δεδομένα χρήσιμα θα πρέπει να έχουν ορισμένα χαρακτηριστικά και συγκεκριμένα να είναι ακριβή, πλήρη, σχετικά και έγκαιρα (Κεχρής, 2005). Αντιθέτως, μια κακοσχεδιασμένη δομή δεδομένων είναι πολύ πιθανό να αποτύχει να παρέχει τις απαιτούμενες πληροφορίες, ή ακόμα χειρότερα, μπορεί να δώσει ξεπερασμένες, προβληματικές ή αντικρουόμενες πληροφορίες (Ricardo, 2012).

Βάση δεδομένων (database) είναι μια συλλογή δεδομένων τα οποία είναι αποθηκευμένα σε αρχεία με μια συστηματικά οργανωμένη δομή (Pelland, 2006). Σύμφωνα με την Andersen (2001), μια βάση δεδομένων είναι μια οργανωμένη συλλογή σχετικών πληροφοριών που χρησιμοποιούνται για συγκεκριμένη χρήση, π.χ. την υποστήριξη μιας βιβλιοθήκης. Δεδομένα τα οποία δεν σχετίζονται μεταξύ τους, δεν μπορούν να χαρακτηριστούν ως βάση δεδομένων (Κεχρής, 2005).

Το γνωστότερο μοντέλο διαχείρισης βάσεων δεδομένων είναι το σχεσιακό (relational). Το σημαντικότερο στοιχείο μιας σχεσιακής βάσης δεδομένων είναι ο πίνακας, ο οποίος είναι μια δομή που αποτελείται από γραμμές (εγγραφές) και στήλες (γνωρίσματα). Μια σχεσιακή βάση δεδομένων μπορεί να περιέχει πολλούς πίνακες που σχετίζονται μεταξύ τους. Η διαχείριση των δεδομένων με τη βοήθεια πινάκων προσφέρει σπουδαία πλεονεκτήματα, με κυριότερο τη μείωση της επαναλαμβανόμενης πληροφορίας. Έτσι, εξοικονομείται χώρος αποθήκευσης και παράλληλα επιταχύνεται η διαδικασία της επεξεργασίας των δεδομένων. Επίσης, η διαχείριση των δεδομένων, ο εντοπισμός και η προσπέλαση των πληροφοριών καθίστανται ευκολότεροι.

Γενικά όταν αναφερόμαστε σε βάσεις δεδομένων, υπάρχουν 4 επίπεδα αφαίρεσης. Το πρώτο επίπεδο είναι η πραγματικότητα και συνήθως αφορά τον οργανισμό για τον οποίο σχεδιάζεται η βάση δεδομένων, π.χ. μια τράπεζα, μια σχολή, μια επιχείρηση, ή πιο απλά η προσωπική μας συλλογή από βιβλία, ταινίες DVD και μουσικά CD. Σε αυτό το επίπεδο αναλύουμε τις απαιτήσεις και καταγράφουμε όλες τις απαραίτητες πληροφορίες που χρειάζεται να παρακολουθούμε προκειμένου να λάβουμε αποφάσεις για τον οργανισμό ή την προσωπική συλλογή μας. Επιπλέον, λαμβάνουμε υπόψη τις ανάγκες των χρηστών της βάσης δεδομένων οι οποίοι περιγράφουν τα είδη των λειτουργιών που θα εκτελούν τα ίδια τα δεδομένα. Στο δεύτερο επίπεδο

δημιουργούμε το ιδεατό σχήμα, χρησιμοποιώντας το μοντέλο Οντοτήτων - Συσχετίσεων. Εν συνεχεία, το ιδεατό σχήμα το "μεταφράζουμε" σε λογικό σχήμα (σχεσιακό μοντέλο) στο οποίο, όπως θα δούμε, μπορούμε να φτάσουμε, εναλλακτικά, απευθείας μέσω της κανονικοποίησης με διάσπαση πινάκων. Τέλος, στο τέταρτο επίπεδο σχεδιάζουμε το φυσικό σχήμα σε κάποιο σύστημα διαχείρισης βάσεων δεδομένων (DBMS) το οποίο έχουμε διαλέξει, όπως τα MySQL, PostgreSQL, Microsoft SQL Server, Oracle κλπ.

Από τα παραπάνω 4 επίπεδα, στην παρούσα εργασία παρουσιάζονται το 2ο (ιδεατό) και το 3ο (λογικό). Τα άλλα δύο επίπεδα δεν αποτελούν αντικείμενο μελέτης της εργασίας, αν και σε ορισμένα σημεία θα γίνουν κάποιες σύντομες αναφορές σε αυτά. Το κύριο αντικείμενο της εργασίας είναι το λογικό επίπεδο, αφού με την κανονικοποίηση επιχειρούμε να φτάσουμε στο τελικό αποτέλεσμα προσεγγίζοντας την όλη διαδικασία "από κάτω προς τα πάνω". Εντούτοις, θα παρουσιαστεί συνοπτικά και το ιδεατό μοντέλο, καθώς και ένα εκτεταμένο, αναλυτικό παράδειγμα δημιουργίας σχήματος βάσεων δεδομένων με τη βοήθεια του μοντέλου Οντοτήτων - Συσχετίσεων.

2.2 Ιδεατό σχήμα

Ένα από τα πιο διαδεδομένα μοντέλα που χρησιμοποιούνται για την αναπαράσταση ιδεατών σχημάτων είναι το μοντέλο Οντοτήτων - Συσχετίσεων (Entity - Relationship Model), το οποίο ανέπτυξε το 1976 ο Chen και βασίζεται στην "εκ των άνω προς τα κάτω" προσέγγιση.

Το μοντέλο Οντοτήτων - Συσχετίσεων περιλαμβάνει τις έννοιες οντότητα, γνώρισμα και συσχέτιση. Αναπαριστά εννοιολογικά τα δεδομένα, αφού οι παραπάνω έννοιες συμβολίζονται αντίστοιχα με τα γεωμετρικά σχήματα της έλλειψης, του ορθογωνίου και του ρόμβου και συνδέονται μεταξύ τους με γραμμές οι οποίες αναπαριστούν τις συσχετίσεις.

Μια οντότητα είναι ένα "αντικείμενο" του κόσμου που ξεχωρίζει από άλλα αντικείμενα. Για παράδειγμα, ο "Φοιτητής" είναι μια οντότητα. Κάθε οντότητα (entity) έχει τα δικά της χαρακτηριστικά γνωρίσματα ή ιδιότητες (attributes) τα οποία συνδέονται μαζί της με γραμμές. Τα γνωρίσματα αυτά παίρνουν τιμές από ένα συγκεκριμένο πεδίο ορισμού, για παράδειγμα το έτος γέννησης είναι ένας ακέραιος αριθμός.

Ένα γνώρισμα είναι πρωτεύον (κλειδί) αν προσδιορίζει την οντότητα μοναδικά, π.χ. ο αριθμός μητρώου ενός φοιτητή. Ουσιαστικά, ένα κλειδί μας επιτρέπει να προσδιορίζουμε ένα σύνολο ιδιοτήτων που είναι αρκετό για να ξεχωρίζουν δύο

αντικείμενα οντοτήτων μεταξύ τους. Το κλειδί θα πρέπει να επιλεγεί με τρόπο τέτοιο έτσι ώστε οι ιδιότητές του να μην αλλάζουν ποτέ ή να αλλάζουν σπάνια.

Σύνθετο κλειδί (composite key) είναι εκείνο που αποτελείται από δύο ή περισσότερα γνωρίσματα. Όταν ένα κλειδί μπορεί, μόνο με τα τελείως απαραίτητα γνωρίσματα, να προσδιορίζει μοναδικά το σύνολο μιας οντότητας, να είναι δηλαδή το "ελάχιστο" κλειδί, τότε θεωρείται υποψήφιο κλειδί (candidate key). Αν σε αυτό, προστεθούν επιπλέον γνωρίσματα, τα οποία όμως δεν είναι απαραίτητα για τον προσδιορισμό της μοναδικότητας, τότε δημιουργείται ένα υπερσύνολο του υποψήφιου κλειδιού, το οποίο ονομάζεται υπερκλειδί.

Οι τιμές των κλειδιών είναι μοναδικές σε μια βάση δεδομένων. Δεν πρέπει να επιτρέπεται σε καμία περίπτωση η χρήση της τιμής null, καθώς δύο ή περισσότερες εγγραφές με τιμή κλειδιού ίση με null δεν θα μπορούν να ξεχωρίσουν μεταξύ τους.

Οι οντότητες συνδέονται μεταξύ τους με συσχετίσεις, οι οποίες στο μοντέλο Οντοτήτων - Συσχετίσεων συμβολίζονται με ρόμβους και γραμμές ή βέλη. Μια συσχέτιση δηλώνεται με ένα ρήμα, για παράδειγμα "ο φοιτητής *είναι* γραμμένος στο τάδε μάθημα". Μια συσχέτιση μπορεί, επίσης, να έχει δικά της γνωρίσματα. Επιπλέον, μπορεί να είναι ολική ή μερική και χαρακτηρίζεται από τον λόγο πληθυκότητας. Η πληθυκότητα σε μια σχέση ορίζει τον αριθμό των αντικειμένων που συμμετέχουν στη συσχέτιση από κάθε οντότητα.

Υπάρχουν 4 κατηγορίες πληθυκότητας. Ειδικότερα:

- ✓ <u>Ένα προς ένα (1:1)</u>

  Μια συσχέτιση μεταξύ δύο οντοτήτων Α και Β είναι 1:1 όταν κάθε αντικείμενο της Α σχετίζεται το πολύ με ένα αντικείμενο της Β και αντίστροφα.
- ✓ <u>Ένα προς πολλά (1:N)</u>

  Όταν κάθε αντικείμενο της οντότητας Α μπορεί να σχετίζεται με πολλά αντικείμενα της Β, αλλά αντίστροφα, κάθε αντικείμενο της Β σχετίζεται με το πολύ ένα αντικείμενο της Α.
- ✓ <u>Πολλά προς ένα (N:1)</u>

  Ακριβώς ίδιο με το προηγούμενο, μόνο που αλλάζει η φορά της συσχέτισης.
- ✓ <u>Πολλά προς πολλά (M:N)</u>

  Όταν κάθε αντικείμενο της οντότητας Α μπορεί να σχετίζεται με πολλά αντικείμενα της Β και αντίστροφα.

Για παράδειγμα, η συσχέτιση μεταξύ των οντοτήτων "Φοιτητής" και "Μάθημα" μπορεί να χαρακτηριστεί ότι έχει λόγο πληθυκότητας M:N, όταν ένας φοιτητής μπορεί να εγγραφεί σε πολλά μαθήματα και αντίστροφα σε κάθε μάθημα μπορούν να είναι εγγεγραμμένοι πολλοί φοιτητές. Αντίστοιχα, η συσχέτιση μεταξύ των οντοτήτων "Μάθημα" και "Καθηγητής" μπορεί να χαρακτηριστεί ως 1:N, όταν ένα μάθημα μπορεί να διδάσκεται μόνο από έναν καθηγητή, αλλά αντίστροφα κάθε καθηγητής μπορεί να διδάσκει πολλά μαθήματα.

Αν όλα τα αντικείμενα μιας οντότητας πρέπει να συμμετέχουν σε μια συσχέτιση, τότε έχουμε ολική (υποχρεωτική) συμμετοχή της οντότητας στη συσχέτιση, κι αυτή συμβολίζεται με τη χάραξη διπλής ή έντονης γραμμής. Αντίθετα, μια απλή γραμμή υποδηλώνει τη μερική συμμετοχή, όπου δεν είναι υποχρεωτικό κάθε αντικείμενο της οντότητας να συμμετέχει στη συσχέτιση.

Αν τα αντικείμενα μιας οντότητας Β εξαρτώνται από τα αντικείμενα μιας οντότητας Α, τότε η Β λέγεται ασθενής (ή αδύναμη) οντότητα που εξαρτάται από την ισχυρή οντότητα της Α. Η ασθενής οντότητα Β μπορεί να προσδιοριστεί μονοσήμαντα με τη βοήθεια του κλειδιού της ισχυρής οντότητας Α. Χαρακτηριστικό παράδειγμα ασθενούς οντότητας είναι ένα αρχείο του υπολογιστή το οποίο ανήκει και ταυτοποιείται από το όνομα και τον φάκελο στον οποίο ανήκει. Για να αναπαραστήσουμε μια ασθενή οντότητα και τη σχέση της, χρησιμοποιούμε γραμμές με έντονο περίγραμμα και το κλειδί της ασθενούς οντότητας το υπογραμμίζουμε με διακεκομμένη γραμμή.

Ένα κακοσχεδιασμένο ιδεατό σχήμα θα είναι δύσκολο να υλοποιηθεί μετά ως βάση δεδομένων, ιδιαίτερα αν τα γνωρίσματα και οι συσχετίσεις μεταξύ των οντοτήτων δεν είναι καλά προσδιορισμένα. Μάλιστα, θα συνεχίσουν να ανακύπτουν προβλήματα καθ' όλη τη διάρκεια της ζωής της βάσης δεδομένων, καθώς θα πρέπει να ανασκευάζεται και να επιδιορθώνεται η δομή της κάθε φορά που θα προκύπτουν νέες ανάγκες για διαφορετικές πληροφορίες. Επομένως, αξίζει και πρέπει να επενδυθεί προσπάθεια και χρόνος ικανός για τη δημιουργία του καλύτερου δυνατού ιδεατού σχήματος.

2.3 Λογικό σχήμα

Στο λογικό επίπεδο χρησιμοποιούμε το σχεσιακό μοντέλο που πρότεινε και ανέπτυξε ο Codd το 1970 και είναι το πιο ευρέως χρησιμοποιούμενο μοντέλο, καθώς ένας μεγάλος αριθμός συστημάτων βάσεων δεδομένων βασίζεται σε αυτό. Το σχεσιακό μοντέλο βασίζεται στην ιδέα της σχέσης, η οποία αναπαρίσταται με πίνακα δύο

διαστάσεων, όπου οι γραμμές αντιστοιχούν σε επιμέρους, ξεχωριστές εγγραφές και οι στήλες αντιστοιχούν στα πεδία. Μια σχεσιακή βάση δεδομένων αποτελείται από ένα σύνολο πινάκων, καθένας εκ των οποίων έχει ένα μοναδικό όνομα. Η ισχύς του σχεσιακού μοντέλου βασίζεται στο ότι αυτό χαρακτηρίζεται από την απλότητα και την ελκυστικότητά του στη χρήση.

Επομένως, οι οντότητες που δημιουργήθηκαν στο ιδεατό επίπεδο μετατρέπονται πλέον σε πίνακες. Δηλαδή, οι σχεδιαστές σχεδιάζουν πρώτα τη διάταξη της βάσης δεδομένων μοντελοποιώντας τα δεδομένα στο υψηλό επίπεδο (ιδεατό) με τη βοήθεια του μοντέλου Οντοτήτων - Συσχετίσεων και εν συνεχεία τα μετατρέπουν στο σχεσιακό σχήμα. Έτσι, μια οντότητα Α μετατρέπεται στον πίνακα Α ο οποίος φέρει όλα τα γνωρίσματά του ως μεταβλητές ή γνωρίσματα διαφορετικού τύπου. Κάθε πεδίο έχει μοναδική ονομασία που το ξεχωρίζει από τα υπόλοιπα πεδία. Επιπλέον, δημιουργούμε πίνακες και για τις επιμέρους συσχετίσεις με λόγο πληθυκότητας M:N.

Ο πίνακας αποτελεί μια κεντρική έννοια του λογικού σχήματος. Πρόκειται για μια δομή που αποτελείται από γραμμές (εγγραφές) και στήλες (πεδία ή γνωρίσματα). Για να γίνει πιο εύκολη η ανάγνωση ενός πίνακα, προσδίδεται σε αυτόν ένα όνομα και αντίστοιχα δίνεται ένα όνομα σε κάθε μια από τις στήλες του πίνακα. Επισημαίνεται ότι η σειρά προβολής των στηλών και των γραμμών δεν έχει καμία σημασία στη συνολική δομή και λειτουργία του πίνακα.

Κάθε πίνακας περιέχει ένα πεδίο που παίζει το ρόλο του κύριου κλειδιού (primary key), ενώ επιπλέον μπορεί να περιέχει και ξένα (foreign) κλειδιά ώστε να συσχετίζεται με άλλους πίνακες. Ουσιαστικά, το κλειδί είναι ένα δεδομένο με το οποίο μπορούμε να ξεχωρίσουμε μεταξύ τους τις εγγραφές. Ως κλειδί, φυσικά, επιλέγεται κάποιο από τα γνωρίσματα της αντίστοιχης οντότητας του ιδεατού σχήματος.

Το κύριο κλειδί είναι μια τιμή η οποία χρησιμοποιείται για να χαρακτηρίσει με μονοσήμαντο τρόπο μια συγκεκριμένη εγγραφή του πίνακα. Μπορεί να αποτελείται από ένα ή περισσότερα πεδία οπότε και ονομάζεται σύνθετο κλειδί (composite key). Από την άλλη, το ξένο κλειδί είναι ένα πεδίο του πίνακα το οποίο είναι συσχετισμένο με ένα πεδίο ενός άλλου πίνακα. Το ξένο κλειδί είναι πάντα κύριο κλειδί στον άλλο πίνακα και χρησιμοποιείται για να ενισχύσει την ακεραιότητα δεδομένων.

Για την απεικόνιση των πινάκων στο λογικό επίπεδο, εμφανίζουμε το όνομα του πίνακα και μέσα σε παρένθεση τις ονομασίες των γνωρισμάτων του. Τα γνωρίσματα μπορεί να είναι διαφόρων γνωστών τύπων, όπως ακέραιοι, δεκαδικοί, κείμενο text κλπ. Για να συνδέσουμε στο λογικό σχήμα δύο πίνακες μεταξύ τους, χρησιμοποιούμε τα

κλειδιά τους μεταφέροντάς τα από τον έναν πίνακα στον άλλο ως ξένα κλειδιά, αν πρόκειται για σχέση 1:1 ή 1:N. Στην περίπτωση του 1:N, το πεδίο που μεταφέρεται στον πρώτο πίνακα (στην πλευρά του 1) παίζει το ρόλο του ξένου κλειδιού. Αν έχουμε λόγο πληθυκότητας M:N, τότε δημιουργούμε έναν τρίτο πίνακα και σε αυτόν μεταφέρουμε ως ξένο, σύνθετο κλειδί τα κύρια κλειδιά των δύο συσχετιζόμενων πινάκων, δηλαδή έναν συνδυασμό κοινών πεδίων από τους δύο πίνακες. Τα ξένα κλειδιά δεν χρειάζεται να έχουν μοναδικές τιμές, ωστόσο αν λαμβάνονται υπόψη ως σύνθετα κλειδιά, τότε θα πρέπει ο συνδυασμός που παράγουν να είναι μοναδικός σε όλες τις εγγραφές του πίνακα.

2.4 Παράδειγμα δημιουργίας σχήματος με το μοντέλο Οντοτήτων -Συσχετίσεων

2.4.1 Ανάλυση απαιτήσεων - περιγραφή βάσης δεδομένων

Έστω ότι θέλουμε να δημιουργήσουμε βάση δεδομένων η οποία θα περιέχει στοιχεία που αφορούν εταιρείες εισηγμένες στο Χρηματιστήριο (στο εξής ΧΑ). Στη βάση δεδομένων θα έχουν πρόσβαση, μέσω σχετικών εφαρμογών, χρηματιστηριακές εταιρείες, επενδυτικοί οίκοι, συντάκτες επιχειρηματικού ρεπορτάζ, μικροεπενδυτές κλπ.

Σε γενικές γραμμές η βάση δεδομένων θα περιλαμβάνει όλες τις εταιρείες των οποίων οι μετοχές βρίσκονται στην αγορά του ΧΑ, ανεξαρτήτως αν διαπραγματεύονται ακόμα ή όχι. Αυτό γίνεται καθώς υπάρχει σημαντικός αριθμός εταιρειών που έχουν περιέλθει σε καθεστώς χρηματιστηριακής επιτήρησης, αναστολής ή και διαγραφής, παρόλα αυτά μας ενδιαφέρει η παρακολούθησή τους γιατί ενδέχεται ακόμα οι μέτοχοί τους να αναζητούν σχετική ειδησεογραφία ή ιστορικό τιμών τους.

Τα περιεχόμενα της βάσης δεδομένων θα είναι τα εξής:

- ✓ Οι εταιρείες και γενικά στοιχεία τους, όπως ο κλάδος στον οποίο ανήκουν, η κατηγορία τους, το σύμβολο με το οποίο διαπραγματεύονται στο ΧΑ, το πλήθος των μετοχών τους, συμμετοχή σε δείκτες όπως ο Γενικός Δείκτης ή οι δείκτες FTSE κλπ.
- ✓ Καταγραφή ιστορικού τιμών των μετοχών ανά συνεδρίαση (όπως τιμές ανοίγματος - κλεισίματος, χαμηλό - υψηλό ημέρας, όγκος και αξία συναλλαγών, αριθμός πράξεων). Δεν μας ενδιαφέρουν ωστόσο λειτουργίες όπως επεξεργασία στατιστικών ή αποθήκευση δεικτών (π.χ.

Κινητοί Μέσοι Όροι) τα οποία απλά θα υπολογίζονται από τις εφαρμογές πρόσβασης που θα υποβάλλουν σχετικά ερωτήματα στη βάση δεδομένων.

- ✓ Καταγραφή των ημερήσιων τιμών κλεισίματος του Γενικού Δείκτη.
- ✓ Ειδησεογραφία η οποία θα συγκεντρώνεται και θα εισάγεται στη βάση δεδομένων από συντάκτες γνωστών ηλεκτρονικών μέσων του χρηματιστηριακού χώρου (Euro2Day, Capital, Liberal, Naftemporiki).
- ✓ Παρουσίαση της μετοχικής εταιρικής σύνθεσης και συγκεκριμένα ποιοι είναι οι μέτοχοι με ποσοστό άνω του 5% σε κάθε εταιρεία βάσει των ανακοινώσεων του Χρηματιστηρίου.
- ✓ Προβολή εταιρικών οικονομικών αποτελεσμάτων (ενεργητικό, ίδια κεφάλαια, κύκλος εργασιών, κέρδη), καθώς και των ετήσιων ισολογισμών. Οι ισολογισμοί αυτοί θα βρίσκονται σε ηλεκτρονική μορφή αρχείου PFD σε ενιαίο φάκελο, αποθηκευμένου στον ίδιο server όπου θα βρίσκεται η βάση δεδομένων.

2.4.2 Σενάρια χρήσης της βάσης δεδομένων

Περιγράφονται συνοπτικά οι λειτουργίες της βάσης δεδομένων, καθώς επίσης και περιπτώσεις δυνητικών χρηστών της και ποια είναι τα δικαιώματα πρόσβασής τους.

2.4.2.1 Χρήστης #1

Αφού δημιουργηθεί το τελικό, φυσικό σχήμα της βάσης δεδομένων, θα οριστεί ένας υπεύθυνος (στο εξής Διαχειριστής) ο οποίος θα έχει τις εξής αρμοδιότητες:

- ✓ Θα καταχωρήσει στη βάση δεδομένων όλες τις εταιρείες που διαπραγματεύονται στο ΧΑ, καθώς και τα γενικά στοιχεία τους. Θα φροντίζει για την επικαιροποίηση των στοιχείων μετά από κάθε αλλαγή που πραγματοποιείται (π.χ. αν μετονομαστεί μια εταιρεία ή αλλάξει κλάδο διαπραγμάτευσης κλπ).
- ✓ Μετά από κάθε χρηματιστηριακή συνεδρίαση και αφού λάβει το ημερήσιο δελτίο τιμών του ΧΑ, θα εισάγει τα σχετικά δεδομένα της συνεδρίασης (ημερομηνία, κλείσιμο Γενικού Δείκτη) και των επιμέρους μετοχών (όπως τιμές ανοίγματος - κλεισίματος, ελάχιστη - μέγιστη τιμή, αξία συναλλαγών, πλήθος πράξεων κλπ).

- ✓ Θα ενημερώνει τη σύνθεση των επιμέρους δεικτών FTSE και συγκεκριμένα ποιες μετοχές απαρτίζουν κάθε έναν από αυτούς, με ποια στάθμιση και την ημερομηνία τελευταίας αναθεώρησης του δείκτη FTSE.
- ✓ Θα αναζητεί τους ετήσιους ισολογισμούς των εταιρειών σε ηλεκτρονική μορφή αρχείου PDF (συνήθως δημοσιεύονται από τον μήνα Μάρτιο και μετά). Αφού ενημερώσει τη βάση δεδομένων με τα μεγέθη "ενεργητικό", "ίδια κεφάλαια", "κύκλος εργασιών" και "καθαρά κέρδη", θα ανεβάζει (upload) το αρχείο του ισολογισμού PDF στον ειδικό φάκελο που θα βρίσκεται στο server όπου φιλοξενείται η βάση δεδομένων. Στο φάκελο δεν μπορεί εκ των πραγμάτων να υπάρχουν περισσότερα από ένα αρχεία PDF με το ίδιο όνομα. Επομένως, το όνομα αρχείου θα λειτουργεί ως κλειδί και ο Διαχειριστής θα φροντίζει αμέσως μετά το upload να ενημερώνει τη βάση δεδομένων με το όνομα του νέου αρχείου, ώστε αυτό να μπορεί να εντοπιστεί κατά την αναζήτηση από τους τελικούς χρήστες.
- ✓ Θα ενημερώνει τη βάση δεδομένων για κάθε αλλαγή στη μετοχική σύνθεση των εταιρειών, κατόπιν σχετικών ανακοινώσεων του ΧΑ.

2.4.2.2 <u>Χρήστης #2</u>

- ✓ Σε συνεργασία με τα ηλεκτρονικά μέσα χρηματιστηριακής ενημέρωσης Euro2Day, Capital, Liberal και Naftemporiki, κάθε ένα από αυτά θα αναθέσει σε έναν δικό του δημοσιογράφο (στο εξής Συντάκτης) ως χρήστη της βάσης δεδομένων. Ο Συντάκτης θα έχει τη δυνατότητα να μπαίνει στη βάση δεδομένων και να την ενημερώνει, προσθέτοντας ειδησεογραφία του μέσου στο οποίο εργάζεται και η οποία θα αφορά τις εταιρείες του ΧΑ και μόνον αυτές.
- ✓ Η βάση δεδομένων δεν θα περιλαμβάνει αυτούσιες τις ειδήσεις, αλλά μια αναφορά σε αυτές. Ο Συντάκτης θα προσθέτει σε αυτή στοιχεία όπως ημερομηνία της δημοσίευσης, την πρώτη παράγραφο της είδησης, εταιρεία ή εταιρείες που αφορά και τον ηλεκτρονικό σύνδεσμο (link) της σελίδας όπου δημοσιεύτηκε.
- ✓ Ο Συντάκτης θα έχει τη δυνατότητα να ανανεώνει τα στοιχεία των δημοσιεύσεων (σε περίπτωση που υπήρξε κάποια λάθος εγγραφή ή άλλαξε κάποιος σύνδεσμος κλπ) ή να διαγράφει τελείως κάποιες εγγραφές

από τη βάση δεδομένων. Δεν θα έχει ωστόσο τα πλήρη δικαιώματα πρόσβασης που θα έχει ο Διαχειριστής.

2.4.2.3 Χρήστης #3

Στην κατηγορία αυτή βρίσκονται οι τελικοί χρήστες της βάσης δεδομένων, όπως οι συντάκτες χρηματιστηριακού ρεπορτάζ που αναζητούν ειδήσεις, οι οικονομικοί αναλυτές, οι χρηματιστηριακές εταιρείες, απλοί μικροεπενδυτές κλπ.

Η πρόσβαση στα δεδομένα θα γίνεται μέσω εφαρμογών οι οποίες θα δημιουργηθούν από τρίτους και στους οποίους θα δώσουμε τη δυνατότητα να έχουν πρόσβαση στα δεδομένα της βάσης δεδομένων. Έτσι, με το κατάλληλο GUI (περιβάλλον διεπαφής χρήστη) θα παρουσιάζονται, με ικανοποιητικό τρόπο, επιλεγμένα στοιχεία της βάσης δεδομένων και ανάλογα φυσικά με τις απαιτήσεις των τελικών χρηστών.

Για παράδειγμα, ένας χρηματιστηριακός αναλυτής ενδιαφέρεται, εκτός από μια απλή παράθεση ιστορικών τιμών μιας μετοχής, να δει επίσης στατιστικές αναλύσεις και τεχνικούς δείκτες, όπως Κινητοί Μέσοι Όροι, MACD κλπ οι οποίοι παράγονται από σύνθετες συναρτήσεις που δέχονται ως ορίσματα τιμές από τη βάση δεδομένων.

Ένας μικροεπενδυτής θέλει να βλέπει πιο απλά στοιχεία, όπως έναν πίνακα ή ένα γράφημα των τιμών κλεισίματος μετοχών, πάντα μέσω μιας εφαρμογής - διεπαφής ή ενός χρηματιστηριακού site το οποίο θα λαμβάνει δεδομένα από τη βάση δεδομένων.

Τέλος, ένας συντάκτης χρηματιστηριακού ρεπορτάζ ενδιαφέρεται πιο πολύ για την ειδησεογραφία ανά εταιρεία, οπότε θα έχει γρήγορη πρόσβαση στους συνδέσμους των μέσων τα οποία ενημερώνουν τη βάση δεδομένων με δημοσιεύσεις τους.

2.4.3 Το ιδεατό σχήμα

Το ιδεατό σχήμα της βάσης δεδομένων (Εικόνα 1) δημιουργήθηκε με τη βοήθεια του μοντέλου Οντοτήτων - Συσχετίσεων στο πρόγραμμα σχεδίασης Dia (διαθέσιμο: http://dia.en.softonic.com/). Οι οντότητες που δημιουργήθηκαν περιγράφονται παρακάτω. Υπογραμμίζονται τα επιμέρους γνωρίσματα της κάθε οντότητας. Επίσης, τα κλειδιά κάθε οντότητας επισημαίνονται με το σύμβολο ♠ (μπαστούνι).

2.4.3.1 Εταιρεία

Είναι η πιο σημαντική οντότητα του σχήματος. Τα γνωρίσματά της είναι τα εξής:

♠Σύμβολο: περιέχει μέχρι 8 χαρακτήρες. Είναι το βασικό διακριτικό γνώρισμα μιας μετοχής στο ΧΑ και είναι μοναδικό για κάθε εταιρεία (π.χ. ΕΤΕ για την Εθνική Τράπεζα, ΕΛΠΕ για τα Ελληνικά Πετρέλαια κλπ) και για αυτό επιλέγεται ως κύριο κλειδί της οντότητας.

Ονομασία: η διακριτική ονομασία της εταιρείας, π.χ. Εθνική Τράπεζα. Δεν μπορεί να αποτελεί κλειδί, αφού ενδέχεται δύο εταιρείες να έχουν την ίδια ονομασία.

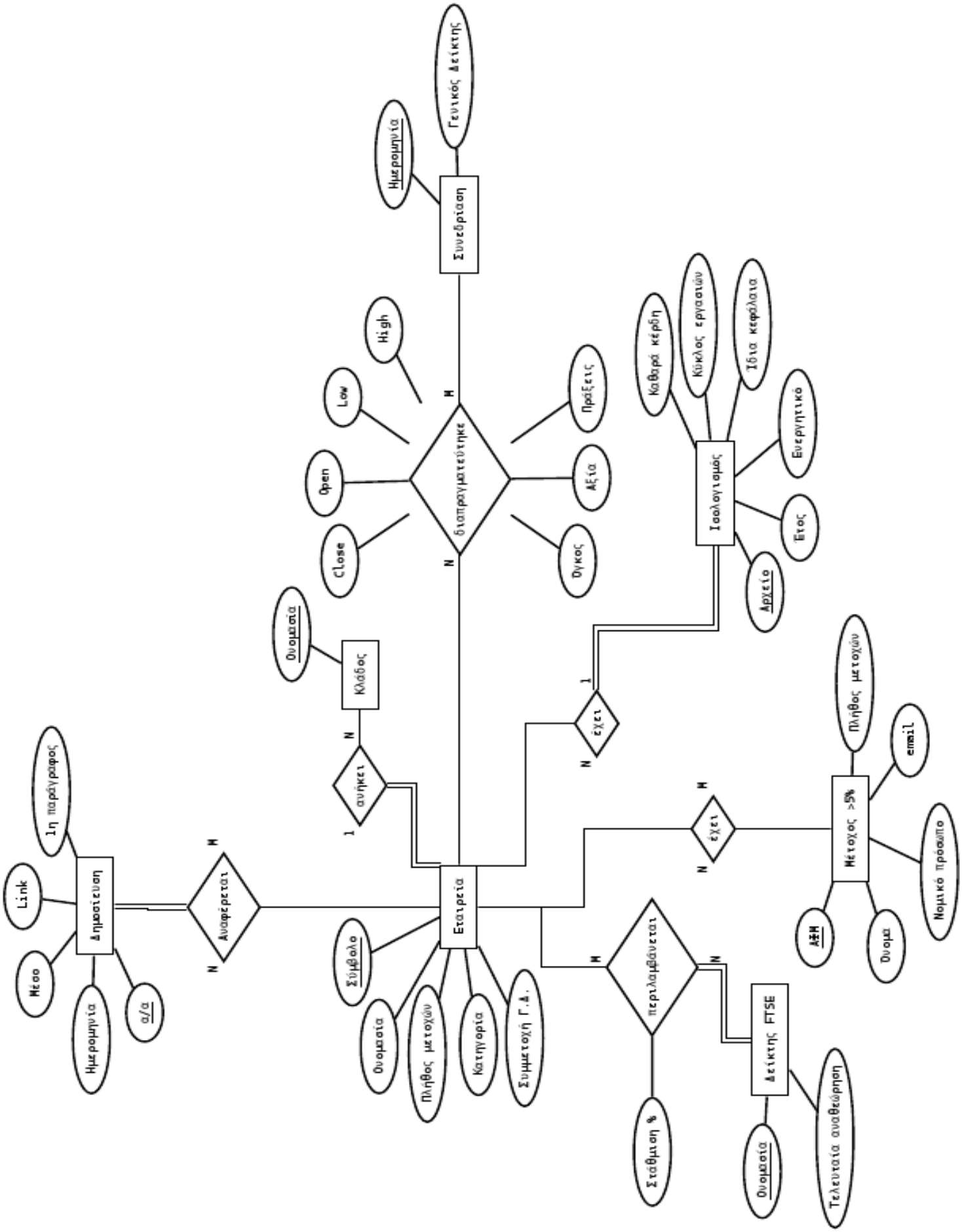
Πλήθος μετοχών: ακέραιος αριθμός που υποδηλώνει το πλήθος των μετοχών στις οποίες είναι διαιρεμένο το μετοχικό κεφάλαιο της εταιρείας.

Κατηγορία: κάθε εταιρεία διαπραγματεύεται σε μια συγκεκριμένη κατηγορία του ΧΑ. Οι κατηγορίες αυτές είναι η Α (κύρια αγορά), Β (χαμηλής διασποράς), Ε (επιτήρησης), Υ (υπό αναστολή) και Χ (διαγράφηκε).

Συμμετοχή Γ.Δ.: υποδηλώνει αν μια εταιρεία συμμετέχει ή όχι στη σύνθεση του Γενικού Δείκτη.

2.4.3.2 Κλάδος

Υπάρχουν τουλάχιστον 20 κλάδοι στο Χρηματιστήριο με μοναδικές ♠ονομασίες όπως τράπεζες, ασφάλειες, τηλεπικοινωνίες, real estate, πληροφορική κλπ. και σε κάθε κλάδο μπορεί να περιλαμβάνονται εταιρείες. Κάθε εταιρεία πρέπει να ανήκει σε έναν και μόνο κλάδο (ολική συμμετοχή). Από την άλλη πλευρά, ωστόσο, δεν υπάρχει υποχρεωτική συμμετοχή αφού ενδέχεται ένας κλάδος να έχει μείνει χωρίς καμία εταιρεία, λόγω μετακινήσεων εταιρειών από τον έναν κλάδο στον άλλο σε περιπτώσεις αλλαγής αντικειμένου επιχειρηματικής δραστηριότητας.



Εικόνα 1: Το ιδεατό σχήμα της βάσης δεδομένων

2.4.3.3 Δείκτης FTSE

Οι δείκτες FTSE λειτουργούν σαν "καλάθια" μετοχών με σκοπό την απεικόνιση μιας συνισταμένης πορείας των τιμών τους. Υπάρχουν αρκετοί δείκτες με μοναδικές ♠ονομασίες όπως FTSE Large Cap, FTSE Τραπεζών, FTSE Εταιρειών Πληροφορικής κλπ. Μια εταιρεία μπορεί να περιλαμβάνεται (μερική συμμετοχή) σε πολλούς (0 ή περισσότερους) δείκτες FTSE και σε κάθε έναν από αυτόν μετέχει με συγκεκριμένο ποσοστό στάθμισης. Σε κάθε FTSE πρέπει να περιλαμβάνεται τουλάχιστον μια εταιρεία (ολική συμμετοχή). Οι δείκτες αυτοί αναθεωρούνται περιοδικά, οπότε μας ενδιαφέρει να διατηρούμε στη βάση δεδομένων την ημερομηνία της τελευταίας αναθεώρησης.

2.4.3.4 Μέτοχος >5%

Όποτε ένας μέτοχος, που διατηρεί ποσοστό άνω του 5% στην εταιρεία, αυξάνει ή μειώνει το ποσοστό του, η εταιρεία προβαίνει σε σχετική ανακοίνωση, οπότε ο Διαχειριστής θα ενημερώσει τη βάση δεδομένων. Μια εταιρεία μπορεί να έχει πολλούς (0 ή περισσότερους) μετόχους με ποσοστό άνω του 5% και φυσικά ένας μέτοχος μπορεί να έχει μερίδιο και σε άλλες, ενδεχομένως και πολλές, εταιρείες. Δεν υπάρχει επομένως ολική συμμετοχή σε καμία περίπτωση, αφού κάλλιστα ένας μέτοχος μπορεί να μην έχει μερίδιο άνω του 5% σε καμία εταιρεία.

Τα γνωρίσματα της οντότητας είναι τα εξής:

♠ΑΦΜ: είναι το κλειδί της οντότητας, ως μοναδικό διακριτικό γνώρισμα.

Όνομα: περιλαμβάνει το ονοματεπώνυμο του μετόχου (ή την επωνυμία αν πρόκειται για νομικό πρόσωπο).

Νομικό πρόσωπο: υποδηλώνει αν ο μέτοχος είναι νομικό πρόσωπο ή όχι, χωρίς να μας ενδιαφέρουν περαιτέρω πληροφορίες, όπως αν είναι ΝΠΔΔ, εταιρεία κλπ.

email: η διεύθυνση email του μετόχου.

Πλήθος μετοχών: το πλήθος των μετοχών που έχει ο μέτοχος στην εταιρεία.

2.4.3.5 Δημοσίευση

Ο Συντάκτης μπορεί να εισάγει στη βάση δεδομένων αναφορές σε δημοσιεύσεις. Μια εταιρεία μπορεί να αναφέρεται σε πολλές δημοσιεύσεις (μπορεί και σε καμία, μερική συμμετοχή) και αντίστοιχα μια δημοσίευση πρέπει να αφορά μία ή και περισσότερες εταιρείες (π.χ. στην περίπτωση μιας είδησης που αφορά σε μια συμφωνία

μεταξύ 2 εταιρειών), άρα έχουμε υποχρεωτική (ολική) συμμετοχή. Κάθε δημοσίευση (εγγραφή) είναι μοναδική και θα της δίνεται αυτόματα από τη βάση δεδομένων το τεχνητό κλειδί ♠α/α (με auto increment).

Τα υπόλοιπα γνωρίσματα της οντότητας είναι:

Ημερομηνία: η ημερομηνία της δημοσίευσης.

Μέσο: η ονομασία του μέσου η οποία θα λαμβάνεται από μια λίστα τύπου enum με τα ορίσματα Euro2Day, Capital, Liberal, Naftemporiki.

Link: ο ηλεκτρονικός σύνδεσμος όπου βρίσκεται καταχωρημένη η σχετική δημοσίευση.

1η παράγραφος: το εισαγωγικό κείμενο της πρώτης παραγράφου της δημοσίευσης.

2.4.3.6 Ισολογισμός

Κάθε χρονιά που μια εταιρεία δημοσιεύει τον ετήσιο ισολογισμό της, ο Διαχειριστής θα τον "ανεβάζει" (upload) στο φάκελο του server σε ηλεκτρονική μορφή PDF, δίνοντάς του μια τυχαία, μοναδική ♠ονομασία αρχείου (π.χ. ELPE_isologismos2014.pdf), η οποία θα λειτουργεί και ως κλειδί της οντότητας. Μια εταιρεία μπορεί να έχει πολλούς (ή κανέναν) ισολογισμό (μερική συμμετοχή), ενώ κάθε ισολογισμός μπορεί και πρέπει (ολική συμμετοχή) να αναφέρεται σε μία εταιρεία.

Τα υπόλοιπα γνωρίσματα της οντότητας είναι τα εξής:

Έτος: είναι η χρονιά χρήσης στην οποία αναφέρεται ο ισολογισμός.

Ενεργητικό, ίδια κεφάλαια, κύκλος εργασιών, καθαρά κέρδη: αναφέρονται στα αντίστοιχα μεγέθη του ισολογισμού.

2.4.3.7 Συνεδρίαση

Καθημερινά, πλην αργιών, διεξάγεται μια συνεδρίαση στο ΧΑ με αγοραπωλησίες μετοχών. Κάθε συνεδρίαση έχει ως μοναδικό κλειδί την ♠ημερομηνία διεξαγωγής. Ο Διαχειριστής, μετά το κλείσιμο της συνεδρίασης, θα εισάγει στη βάση δεδομένων μια νέα εγγραφή συνεδρίασης με την τρέχουσα ημερομηνία και την τιμή κλεισίματος του Γενικού Δείκτη που αποτελεί το δεύτερο γνώρισμα της οντότητας αυτής.

Η συνεδρίαση ως οντότητα συσχετίζεται με την εταιρεία. Πολλές εταιρείες μπορούν να διαπραγματεύονται σε μια συνεδρίαση (χωρίς να είναι υποχρεωτικό να

εμφανίζονται πάντα στον πίνακα συναλλαγών μιας συνεδρίασης, άρα υπάρχει μερική συμμετοχή) και σε μια συνεδρίαση μπορούν να εμφανίζονται πολλές εταιρείες (μερική συμμετοχή). Καθώς μια εταιρεία συσχετίζεται με μια ημερήσια διαπραγμάτευση, θεωρούμε ως γνωρίσματα της συγκεκριμένης συσχέτισης τις τελικές τιμές που αφορούν το κλείσιμο Close, το άνοιγμα Open, την ελάχιστη Low και τη μέγιστη τιμή High, καθώς επίσης και τον όγκο (σε τεμάχια), την αξία (σε ευρώ) και τον αριθμό των πράξεων.

2.4.4 Το λογικό σχήμα

Αφού ολοκληρώθηκε η δημιουργία του ιδεατού σχήματος με το μοντέλο Οντοτήτων - Συσχετίσεων, προχωράμε στη μεταφορά του σε λογικό σχήμα. Συγκεκριμένα, μετατρέπουμε τις οντότητες και τα γνωρίσματά τους σε πίνακες με πεδία. Το κύριο κλειδί κάθε πίνακα επισημαίνεται και εδώ με το σύμβολο ♠ μπροστά από το γνώρισμα (πεδίο). Σε περίπτωση που υπάρχει σύνθετο κλειδί, τότε εμφανίζεται σε όλα τα γνωρίσματα που μετέχουν σε αυτό το ίδιο σύμβολο.

2.4.4.1 Εταιρεία

COMPANY(♠*com_symbol, com_name, com_sharesnum, com_category, com_gdpart*)

2.4.4.2 Κλάδος

BRANCH(♠*bra_name*)

Επειδή η οντότητα του κλάδου συσχετίζεται 1:N με την εταιρεία, στον πίνακα της εταιρείας θα προσθέσουμε ως ξένο κλειδί το κύριο κλειδί του κλάδου (αφού η εταιρεία βρίσκεται στην πλευρά του 1) και η τελική μορφή του πίνακα της εταιρείας θα έχει ως εξής:

COMPANY(♠*com_symbol, bra_name, com_sharesnum, com_category, com_gdpart*)

2.4.4.3 <u>Δείκτης FTSE</u>

FTSE(♠*ftse_name, ftse_revision*)

Ο δείκτης FTSE συσχετίζεται με την εταιρεία με σχέση M:N, επομένως δημιουργούμε έναν νέο πίνακα, τον FTSE_INCLUDED, ο οποίος έχει ως γνώρισμα την στάθμιση included_weigh και επιπλέον έχει δύο ξένα κλειδιά: το σύμβολο της εταιρείας και την ονομασία του FTSE. Το σύνθετο κλειδί του πίνακα αυτού θα είναι ο συνδυασμός των δύο ξένων κλειδιών. Επομένως:

FTSE_INCLUDED(♠*com_symbol,* ♠*ftse_name, included_weigh*)

Ξένα κλειδιά: com_symbol, ftse_name

Σύνθετο κλειδί: (com_symbol, ftse_name)

2.4.4.4 <u>Μέτοχος >5%</u>

SHAREHOLDER(♠*sha_afm, sha_name, sha_entity, sha_email, sha_sharesnum*)

Ο μέτοχος συσχετίζεται με την εταιρεία με σχέση M:N, επομένως δημιουργούμε έναν νέο πίνακα, τον HAS_SHAREHOLDER, ο οποίος έχει δύο ξένα κλειδιά: το σύμβολο της εταιρείας και το ΑΦΜ του μετόχου. Το σύνθετο κλειδί του πίνακα αυτού θα είναι ο συνδυασμός των δύο ξένων κλειδιών. Επομένως:

HAS_SHAREHOLDER(♠*com_symbol,* ♠*sha_afm*)

Ξένα κλειδιά: com_symbol, sha_afm

Σύνθετο κλειδί: (com_symbol, sha_afm)

2.4.4.5 <u>Δημοσίευση</u>

PUBLISH(♠*pub_id, pub_date, pub_mme, pub_link, pub_intro*)

Η δημοσίευση συσχετίζεται με την εταιρεία στην οποία αναφέρεται με σχέση M:N. Επομένως δημιουργούμε έναν νέο πίνακα, τον HAS_PUBLISH, ο οποίος έχει δύο ξένα κλειδιά: το σύμβολο της εταιρείας και τον α/α της δημοσίευσης. Το σύνθετο κλειδί του πίνακα αυτού θα είναι ο συνδυασμός των δύο ξένων κλειδιών. Επομένως:

HAS_PUBLISH(♠*com_symbol,* ♠*pub_id*)

Ξένα κλειδιά: com_symbol, pub_id

Σύνθετο κλειδί: (com_symbol, pub_id)

2.4.4.6 Ισολογισμός

SHEET(♠*sheet_name, com_symbol, sheet_year, sheet_equity, sheet_bookvalue, sheet_turnover, sheet_profits*)

Ο ισολογισμός συσχετίζεται με την εταιρεία σε σχέση N:1, επομένως θα τοποθετήσουμε ως ξένο κλειδί στον πίνακα του ισολογισμού το κύριο κλειδί της εταιρείας com_symbol (αφού ο ισολογισμός βρίσκεται από την πλευρά του 1).

2.4.4.7 Συνεδρίαση

SESSION(♠*ses_date, ses_gd*)

Η συνεδρίαση συσχετίζεται M:N με την εταιρεία, επομένως δημιουργούμε έναν νέο πίνακα για τη συσχέτιση "διαπραγματεύτηκε". Ο πίνακας θα πάρει ως ξένα κλειδιά το κύριο κλειδί της εταιρείας com_symbol και της συνεδρίασης ses_date και θα φέρει και τα υπόλοιπα γνωρίσματα της συσχέτισης ως εξής:

PART(♠*com_symbol,* ♠*ses_date, part_close, part_open, part_low, part_high, part_volume, part_value, part_acts*)

Ξένα κλειδιά: com_symbol, ses_date

Σύνθετο κλειδί: (com_symbol, ses_date)

2.4.5 Σύνοψη

Συνοψίζοντας, οι πίνακες που δημιουργήθηκαν στο λογικό επίπεδο, με τα γνωρίσματα και τα κλειδιά τους, είναι σε αλφαβητική σειρά οι εξής:

- BRANCH(♠*bra_name*)
- COMPANY(♠*com_symbol, bra_name, com_sharesnum, com_category, com_gdpart*)
- FTSE(♠*ftse_name, ftse_revision*)
- FTSE_INCLUDED(♠*com_symbol,* ♠*ftse_name, included_weigh*)
- HAS_PUBLISH(♠*com_symbol,* ♠*pub_id*)
- HAS_SHAREHOLDER(♠*com_symbol,* ♠*sha_afm*)
- PART(♠*com_symbol,* ♠*ses_date, part_close, part_open, part_low, part_high, part_volume, part_value, part_acts*)
- PUBLISH(♠*pub_id, pub_date, pub_mme, pub_link, pub_intro*)
- SESSION(♠*ses_date, ses_gd*)

- SHAREHOLDER(♠*sha_afm, sha_name, sha_entity, sha_email, sha_sharesnum*)
- SHEET(♠*sheet_name, com_symbol, sheet_year, sheet_equity, sheet_bookvalue, sheet_turnover, sheet_profits*)

Στη συνέχεια, στο φυσικό επίπεδο, ο προγραμματιστής της βάσης δεδομένων θα δημιουργήσει στο επιλεγμένο σύστημα διαχείρισης βάσεων δεδομένων τους παραπάνω πίνακες, χρησιμοποιώντας εντολές DDL της γλώσσας SQL. Ωστόσο, η διαδικασία αυτή δεν αποτελεί αντικείμενο μελέτης της διπλωματικής εργασίας.

3 Κανονικοποίηση

3.1 Εισαγωγή

Η κανονικοποίηση (normalization) βασίζεται σε ένα καλά θεμελιωμένο, θεωρητικό υπόβαθρο το οποίο συστηματοποιεί τη διαδικασία προσδιορισμού του σχήματος μιας βάσης δεδομένων, προσεγγίζοντάς την "από κάτω προς τα πάνω". Πρόκειται για μια τυπική διαδικασία που επιτρέπει την αναγνώριση κακώς σχεδιασμένων πινάκων και υποδεικνύει έναν αλγοριθμικό τρόπο βάσει του οποίου θα γίνει η διάσπασή τους. Ο τελικός σκοπός της κανονικοποίησης είναι να μετατρέψει τα δεδομένα μιας βάσης δεδομένων σε απλουστευμένη μορφή ώστε να εξαλειφθούν τα προβλήματα, να αποφεύγεται η άσκοπη επανάληψη πληροφοριών και να περιορίζεται ο βαθμός πλεονασμού των δεδομένων για να διατηρείται η ακεραιότητά τους.

Ο σχεδιαστής της βάσης δεδομένων ξεκινά τη διαδικασία δημιουργώντας έναν universal πίνακα όλων των γνωρισμάτων και με βάση τις συναρτησιακές εξαρτήσεις και έναν συγκεκριμένο αλγόριθμο, δημιουργεί τους πίνακες του λογικού σχήματος.

3.2 Συναρτησιακές εξαρτήσεις

3.2.1 Γενικά για τις συναρτησιακές εξαρτήσεις

Μια θεμελιώδης έννοια στην κανονικοποίηση είναι η συναρτησιακή εξάρτηση (functional dependency). Αυτή καθορίζει έναν περιορισμό ανάμεσα σε στήλες (πεδία) ενός πίνακα. Για δύο γνωρίσματα Α και Β που ανήκουν σε έναν πίνακα R ισχύει η συναρτησιακή εξάρτηση Α → Β αν κάθε τιμή του Α συνδέεται με μια μόνο τιμή του Β. Ο συμβολισμός Α → Β διαβάζεται "το Β είναι συναρτησιακά εξαρτώμενο από το Α" ή πιο απλά "το Α δίνει το Β". Πρόκειται δηλαδή για έναν τύπο σχέσης ανάμεσα στα γνωρίσματα ή αλλιώς, έναν τύπο περιορισμού, δηλαδή μια γενίκευση της έννοιας του κλειδιού. Επισημαίνεται ότι τα Α και Β είναι δυνατόν να είναι σύνολα χαρακτηριστικών. Επιπλέον, σε έναν πίνακα μπορεί να ισχύουν περισσότερες της μιας συναρτησιακές εξαρτήσεις ανάμεσα στα πεδία του.

Οι συναρτησιακές εξαρτήσεις μας επιτρέπουν να εκφράζουμε γεγονότα για τον οργανισμό που μοντελοποιούμε με τη βάση δεδομένων μας (Silberschatz, 2002). Ένας ορισμός που δίνεται από την Ricardo (2012) για τη συναρτησιακή εξάρτηση αναφέρει ότι εάν το R είναι ένα σχεσιακό σχήμα και τα Α, Β είναι μη κενά γνωρίσματα του R,

δεχόμαστε ότι το Β είναι συναρτησιακά εξαρτώμενο (functionally dependent) από το Α, αν και μόνο αν κάθε τιμή του Α σχετίζεται με μια μόνο ακριβώς τιμή του Β.

Έτσι, για κάθε ζεύγος τιμών x_i και x_j, σε κάθε αντικείμενο του R, ισχύει ο κανόνας "*αν* $x_iA = x_jA$ *τότε* $x_iB = x_jB$". Με απλά λόγια, αν δύο εγγραφές του πίνακα R έχουν την ίδια τιμή στη στήλη Α, τότε θα πρέπει οι δύο αντίστοιχες τιμές της στήλης Β να έχουν την ίδια τιμή μεταξύ τους, σε διαφορετική περίπτωση λέμε ότι "η συναρτησιακή εξάρτηση παραβιάζεται". Ωστόσο, δεν είναι υποχρεωτικό να ισχύει το αντίστροφο. Δηλαδή, για τις ίδιες τιμές της στήλης Β μπορούν να αντιστοιχούν 2 διαφορετικές μεταξύ τους τιμές στην στήλη Α. Επομένως, στην περίπτωση της συναρτησιακής εξάρτησης "Α → Β", το Α αποτελεί ένα υποψήφιο κλειδί του πίνακα.

Ουσιαστικά, μια συναρτησιακή εξάρτηση είναι μια σχέση 1:N ("1 προς πολλά") από το γνώρισμα Α προς το γνώρισμα Β. Είναι ένας περιορισμός ακεραιότητας στον οποίο κάθε αντικείμενο της βάσης δεδομένων πρέπει να "υπακούει". Έτσι, στη συναρτησιακή εξάρτηση Α → Β, το Α που βρίσκεται στα αριστερά της σχέσης το ονομάζουμε ορίζουσα και λέμε ότι προσδιορίζει (determinant) και το Β στα δεξιά ότι εξαρτάται (dependent) ή προσδιορίζεται.

Για να ορίσουμε μια συναρτησιακή εξάρτηση θα πρέπει να θυμόμαστε ότι το αριστερό σκέλος δίνει πάντα τις ίδιες τιμές στο δεξί σκέλος. Για παράδειγμα, μια σωστή συναρτησιακή εξάρτηση είναι η "ΑΦΜ → Όνομα", αφού 2 ή περισσότερες εγγραφές με την ίδια τιμή στη στήλη "ΑΦΜ" έχουν εκ των πραγμάτων την ίδια μεταξύ τους τιμή και στη στήλη "Όνομα" (καθώς ο αριθμός ΑΦΜ είναι μοναδικός για κάθε φυσικό πρόσωπο, λειτουργεί δηλαδή ως κλειδί). Αν προεκτείνουμε το παράδειγμα αυτό, ένα ΑΦΜ μπορεί να δώσει και το έτος γέννησης, καθότι είναι μοναδικό, οπότε γράφουμε "ΑΦΜ → Έτος γέννησης", άρα μπορούμε να συνενώσουμε τα δεξιά σκέλη των δύο συναρτησιακών εξαρτήσεων και να γράψουμε "ΑΦΜ → Όνομα, Έτος γέννησης". Επίσης, ισχύει ότι ένα γνώρισμα είναι συναρτησιακά εξαρτώμενο από τον εαυτό του, δηλαδή Α → Α. Τέλος, επισημαίνεται ότι το αριστερό σκέλος μιας συναρτησιακής εξάρτησης αντιμετωπίζεται ως ένα ενιαίο σύνολο γνωρισμάτων.

3.2.2 Τετριμμένες συναρτησιακές εξαρτήσεις

Ορισμένες συναρτησιακές εξαρτήσεις χαρακτηρίζονται ως τετριμμένες (trivial) επειδή ικανοποιούνται από όλες τις σχέσεις. Μια συναρτησιακή εξάρτηση είναι τετριμμένη αν το δεξί σκέλος είναι υποσύνολο του αριστερού (RIGHT $\subseteq$ LEFT). Για παράδειγμα, οι συναρτησιακές εξαρτήσεις Α,Β → Α, Α,Β → Α,Β, Α,Β → Β είναι όλες τους τετριμμένες, αφού και στις 3 περιπτώσεις όλα τα γνωρίσματα του δεξιού σκέλους περιλαμβάνονται στο αριστερό.

Οι μη-τετριμμένες συναρτησιακές εξαρτήσεις μπορούν να διακριθούν σε πλήρως μη-τετριμμένες και σε μερικώς μη-τετριμμένες. Μια συναρτησιακή εξάρτηση είναι πλήρως μη-τετριμμένη όταν κανένα γνώρισμα από το δεξί σκέλος δεν περιλαμβάνεται στα αριστερά, π.χ. Α,Β → Γ. Στην άλλη περίπτωση, μια συναρτησιακή εξάρτηση είναι μερικώς μη-τετριμμένη όταν ένα τουλάχιστον από τα γνωρίσματα από το δεξί σκέλος δεν είναι μέρος των γνωρισμάτων του αριστερού σκέλους, π.χ. Α,Β → Β,Γ.

3.2.3 Αξιώματα του Armstrong

Όταν δοθεί ένα σύνολο από συναρτησιακές εξαρτήσεις, μπορούμε να αποδείξουμε ότι ισχύουν και κάποιες άλλες εξαρτήσεις λαμβάνοντας υπόψη τα αξιώματα του Armstrong (1974). Οι κανόνες αυτοί ουσιαστικά κληρονομούν γνωστές ιδιότητες από τη θεωρία των συνόλων. Ειδικότερα, ισχύουν τα εξής:

- Αντανακλαστικότητα: Αν Β $\subseteq$Α, τότε Α → Β.
- Επαύξηση: Αν Α → Β , τότε ΑΓ → ΒΓ.
- Μεταβατικότητα: Αν Α → Β και Β → Γ, τότε Α → Γ.

Επιπλέον, ισχύουν και οι κανόνες της ένωσης και της διάσπασης, αν και αυτοί δεν είναι ουσιώδους σημασίας και η ισχύς τους μπορεί να αποδειχθεί δια μέσου των αξιωμάτων του Armstrong.

- Ένωση: Αν Α → Β και Α → Γ, τότε Α → ΒΓ.
- Διάσπαση: Αν Α → ΒΓ , τότε Α → Β και Α → Γ.

3.2.4 Προσδιορισμός κλειδιών

Με τη βοήθεια των συναρτησιακών εξαρτήσεων και λαμβάνοντας υπόψη τα αξιώματα του Armstrong, μπορούμε να προσδιορίσουμε όλα τα υποψήφια κλειδιά ενός πίνακα R. Για να ελέγξουμε αν ένα γνώρισμα X του πίνακα αποτελεί υποψήφιο κλειδί του R, πρέπει αρχικά να υπολογίσουμε τον εγκλεισμό γνωρισμάτων X^+.

Ο εγκλεισμός X^+ (closure) ενός συνόλου X από γνωρίσματα, από το F (όπου F το σύνολο των συναρτησιακών εξαρτήσεων) είναι το σύνολο όλων των γνωρισμάτων που εξαρτώνται συναρτησιακά από το X μέσω του F. Ο αλγόριθμός του μπορεί να περιγραφεί συνοπτικά με τον εξής ψευδοκώδικα:

```
Result = X
Εκτέλεση βρόχου (για κάθε συναρτησιακή εξάρτηση τύπου A → B, με B € F)
            Αν A € Result, τότε Result = Result ∪ B
```

Αφού υπολογιστεί ο εγκλεισμός X^+, ελέγχεται αν περιλαμβάνει όλο το πλήθος των γνωρισμάτων του πίνακα R κι αν ναι, τότε το X είναι υποψήφιο κλειδί του R.

Επαναλαμβάνουμε την ίδια διαδικασία για όλα τα γνωρίσματα του R μέχρι να φτάσουμε στο τελικό σημείο να έχουμε υπολογίσει όλα τα υποψήφια κλειδιά του. Κατά τη διαδικασία αυτή, δεν λαμβάνουμε υπόψη τις τετριμμένες συναρτησιακές εξαρτήσεις.

Για παράδειγμα, έστω ο πίνακας R = {A, B, Γ, Δ, E, Z} και οι παρακάτω συναρτησιακές εξαρτήσεις, κι εμείς θέλουμε να ελέγξουμε αν το γνώρισμα A και ο συνδυασμός γνωρισμάτων {A, Δ} αποτελούν υποψήφια κλειδιά.

$A \rightarrow B$

$A \rightarrow \Gamma$

$\Gamma,\Delta \rightarrow E$

$\Gamma,\Delta \rightarrow Z$

$B \rightarrow E$

Για την περίπτωση του A^+: Αρχικά προσθέτουμε στον εγκλεισμό του A το ίδιο ο γνώρισμα A αφού προσδιορίζει τον εαυτό του. Επομένως, ισχύει:

$\{A\}^+ = \{A\}$

Με βάση τις συναρτησιακές εξαρτήσεις $A \rightarrow B$ και $A \rightarrow \Gamma$, προσθέτουμε στον εγκλεισμό τα B και Γ, οπότε έχουμε:

$\{A\}^+ = \{A, B, \Gamma\}$

Με βάση τη συναρτησιακή εξάρτηση $B \rightarrow E$, προσθέτουμε και το E στον εγκλεισμό, αφού με βάση τον αλγόριθμο του εγκλεισμού, το αριστερό σκέλος της

συναρτησιακής εξάρτησης είναι υποσύνολο του ως τώρα αποτελέσματος του εγκλεισμού. Οπότε, ο εγκλεισμός διαμορφώνεται ως εξής:

$\{A\}^+ = \{A, B, Γ, E\}$

Οι συναρτησιακές εξαρτήσεις Γ,Δ → Ε και Γ,Δ → Ζ δεν προσθέτουν κάτι παραπάνω στον εγκλεισμό, αφού το γνώρισμα Δ δεν είναι υποσύνολο του ως τώρα αποτελέσματος. Οπότε η διαδικασία υπολογισμού του εγκλεισμού ολοκληρώνεται. Ελέγχοντας το αποτέλεσμα του εγκλεισμού $\{A\}^+$ διαπιστώνουμε ότι δεν περιλαμβάνει το σύνολο όλων των γνωρισμάτων του R, οπότε το Α δεν αποτελεί υποψήφιο κλειδί του R.

Για την περίπτωση του $\{A, Δ\}^+$: Αρχικά προσθέτουμε στον εγκλεισμό τα γνωρίσματα Α και Δ αφού προσδιορίζουν τον εαυτό τους.

(1) $\{A, Δ\}^+ = \{A, Δ\}$

Με βάση την συναρτησιακή εξάρτηση Α → Β, προσθέτουμε στη σχέση 1 το Β, οπότε έχουμε:

(2) $\{A, Δ\}^+ = \{A, B, Δ\}$

Με βάση την συναρτησιακή εξάρτηση Α → Γ, προσθέτουμε στη σχέση 2 το Γ, οπότε έχουμε:

(3) $\{A, Δ\}^+ = \{A, B, Γ, Δ\}$

Με βάση τις συναρτησιακές εξαρτήσεις Γ,Δ → Ε και Γ,Δ → Ζ, προσθέτουμε στη σχέση 3 τα Ε και Ζ, οπότε έχουμε:

(4) $\{A, Δ\}^+ = \{A, B, Γ, Δ, E, Z\}$

Δεν χρειάζεται να χρησιμοποιήσουμε την τελευταία συναρτησιακή εξάρτηση Β → Ε, αφού όπως βλέπουμε στην τελευταία σχέση (4) ο εγκλεισμός των $\{A, Δ\}^+$ ήδη πια περιλαμβάνει το σύνολο των γνωρισμάτων του πίνακα R. Επομένως, ο συνδυασμός των γνωρισμάτων {Α, Δ} αποτελεί υποψήφιο κλειδί του R.

Η αναζήτηση των υποψήφιων κλειδιών ξεκινά πάντα παίρνοντας τα γνωρίσματα ανά ένα ξεχωριστά κι αργότερα σε συνδυασμούς ανά *κ* γνωρίσματα ($1 \leq κ \leq ν$, όπου *ν* το πλήθος των γνωρισμάτων). Στη συνέχεια, ελέγχουμε συνδυασμούς γνωρισμάτων, παίρνοντας συνδυασμούς τους ανά δύο, εν συνεχεία ανά τρία, τέσσερα κλπ, λαμβάνοντας όμως υπόψη ότι σε αυτούς τους συνδυασμούς δεν θα πρέπει να εμπεριέχονται κλειδιά που ήδη βρήκαμε στο προηγούμενο επίπεδο. Ο αλγόριθμος της διαδικασίας αυτής μπορεί να περιγραφεί συνοπτικά με τον εξής ψευδοκώδικα:

```
Έστω R: αρχικός πίνακας, ν: πλήθος γνωρισμάτων του R, κ: πλήθος
γνωρισμάτων συνδυασμού, Χ: συνδυασμός γνωρισμάτων, keyList: τα υποψήφια
κλειδιά.
```

```
new keyList
κ = 0
Βρόχος εκτέλεσης (εκτελείται όσο κ ≤ ν)
      κ = κ + 1
      Βρόχος ελέγχου όλων των δυνητικών συνδυασμών Χ με κ γνωρίσματα
      πλην αυτών που αποτελούν υπερσύνολο κάποιου κλειδιού της keyList
            Υπολογισμός εγκλεισμού X+
            Αν X+ = R τότε προσθήκη στη keyList του Χ ως κλειδί
```

Για παράδειγμα, αν στο πρώτο επίπεδο αναζήτησης (ανά ένα γνώρισμα ξεχωριστά) εντοπίσουμε ως υποψήφιο κλειδί το Α, στο δεύτερο επίπεδο αναζήτησης (συνδυασμός ανά δύο), δεν θα ελέγξουμε τους συνδυασμούς που περιλαμβάνουν το Α, και συγκεκριμένα τους {Α, Β}, {Α, Γ}, {Α, Δ}, {Α, Ε}, {Α, Ζ}, καθώς σε αυτούς εμπεριέχεται το κλειδί Α, άρα οι συνδυασμοί αυτοί θεωρούνται ως υπερσύνολα κλειδιού (υπερκλειδιά) κι επομένως δεν χρειάζεται να τους λάβουμε υπόψη. Αντίθετα, θα πρέπει π.χ. να ελέγξουμε ως υποψήφιο κλειδί το συνδυασμό {Β, Γ}. Το ίδιο θα γίνει και στο τρίτο επίπεδο (συνδυασμοί γνωρισμάτων) όπου αν για παράδειγμα ήδη έχουμε εντοπίσει ως υποψήφιο κλειδί το {Β, Γ}, τότε δεν θα λάβουμε υπόψη ως υποψήφιο κλειδί έναν συνδυασμό τριών γνωρισμάτων που να εμπεριέχει τα Β και Γ ταυτόχρονα, π.χ. το {Β, Γ, Ε}, ωστόσο εξετάζουμε συνδυασμούς που τα περιέχουν ξεχωριστά, π.χ. τους {Β, Δ, Ε} και {Γ, Δ, Ε}.

Θα πρέπει επίσης να επισημανθεί ότι ελέγχουμε τα γνωρίσματα σε συνδυασμούς κι όχι σε διατάξεις (π.χ. διάταξη είναι η {Α, Β} και η {Β, Α} όπου απλά αλλάζει η σειρά των γνωρισμάτων), καθώς δεν παίζει κανένα ρόλο η σειρά με την οποία εμφανίζονται τα γνωρίσματα σε έναν συνδυασμό κλειδιών.

3.2.5 Συνδυαστική για τον εντοπισμό κλειδιών

Κατά τη διαδικασία της αναζήτησης των κλειδιών, προκειμένου να υπολογίσουμε το πλήθος των δυνητικών συνδυασμών των γνωρισμάτων μεταξύ τους, ανατρέχουμε στη θεωρία της Συνδυαστικής. Έστω, λοιπόν, ότι έχουμε ένα πεπερασμένο σύνολο με ν (διαφορετικά μεταξύ τους) στοιχεία και κ έναν θετικό ακέραιο (με $\kappa \leq \nu$). Για να υπολογίσουμε το πλήθος των συνδυασμών των ν στοιχείων ανά κ υπολογίζουμε τη σχέση $\frac{\nu!}{\kappa!\,(\nu-\kappa)!}$ (όπου $1 \leq \kappa \leq \nu$).

Έστω ότι στο παραπάνω παράδειγμα θέλουμε να πάρουμε όλους τους δυνητικούς συνδυασμούς όλων των γνωρισμάτων του πίνακα R, ανά 5 γνωρίσματα (δηλαδή ν=6 και κ=5). Με βάση την παραπάνω σχέση που δόθηκε, υπολογίζουμε ότι

$$\frac{6!}{5!\,(6-5)!} = \frac{720}{120} = 6$$

, δηλαδή έχουμε 6 συνδυασμούς από τον αρχικό πίνακα R.

Ένας καλός αλγόριθμος για τον προσδιορισμό των 6 αυτών συνδυασμών είναι με τη βοήθεια του δυαδικού συστήματος, όπου κάθε συνδυασμός bits είναι μοναδικός. Πιο συγκεκριμένα, δημιουργούμε όλους τους *ν*-bit αριθμούς (όπου *ν* το πλήθος των γνωρισμάτων) και αντιστοιχούμε κάθε στήλη σε ένα γνώρισμα, ξεκινώντας από τα δεξιά προς τα αριστερά. Από τους αριθμούς που προκύπτουν, κρατάμε μόνο αυτούς που σε κάθε γραμμή έχουν ακριβώς *κ*-bits με τιμή 1 (δηλαδή συνδυασμούς ανά *κ* γνωρίσματα, εν προκειμένω ανά 5). Απορρίπτουμε τους υπόλοιπους δυαδικούς αριθμούς, αφού δεν θέλουμε ούτε μεμονωμένα γνωρίσματα (1bit), ούτε συνδυασμούς ανά 2 ή 3 ή 4 γνωρίσματα. Επομένως, έχουμε τους εξής συνδυασμούς που μας ενδιαφέρουν:

Z	**E**	**Δ**	**Γ**	**B**	**A**
0	1	1	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	0

Σε κάθε γραμμή, το bit 1 ισοδυναμεί με συμμετοχή του γνωρίσματος στο συνδυασμό, οπότε οι τελικοί 6 συνδυασμοί των γνωρισμάτων είναι οι εξής:

{A, B, Γ, Δ, E}

{A, B, Γ, Δ, Z}

{A, B, Γ, E, Z}

{A, B, Δ, E, Z}

{A, Γ, Δ, E, Z}

{B, Γ, Δ, E, Z}

Σε περίπτωση μεγάλου αριθμού *κ* γνωρισμάτων ανά συνδυασμό, προκειμένου να πετύχουμε βελτιστοποίηση του παραπάνω αλγόριθμου, ξεκινάμε να δημιουργούμε τη σειρά των δυαδικών αριθμών από το τη σειρά $2^{κ}$-1, στην οποία και εντοπίζεται ο πρώτος πλήρης συνδυασμός *κ* ψηφίων με bit ίσον με 1. Έτσι, για το παραπάνω παράδειγμα όπου

δημιουργούμε συνδυασμούς ανά 5 γνωρίσματα, η πρώτη τιμή που θα αναζητήσουμε βρίσκεται στη σειρά 2^5-1 = 31_{10} = 011111_2 και πράγματι, αυτή η τιμή μας δίνει τον πρώτο πλήρη συνδυασμό 5 γνωρισμάτων, τον {Α, Β, Γ, Δ, Ε}.

3.2.6 Πρώτες κανονικές μορφές

Για να εκτελεστεί η κανονικοποίηση ακολουθείται μια συγκεκριμένη σειρά βημάτων. Σε κάθε ένα από αυτά τα βήματα αναλύεται και πιθανώς επανασχεδιάζεται κάθε πίνακας της βάσης δεδομένων με βάση ένα συγκεκριμένο κριτήριο - κανόνα που ονομάζεται κανονική μορφή (normal form - NF). Στην ουσία, με την κανονικοποίηση διαμορφώνουμε μια σχέση σε μια υψηλότερη κανονική μορφή. Οι πρώτες κανονικές μορφές ονομάζονται και συμβολίζονται αντίστοιχα ως εξής:

- ✓ Πρώτη κανονική μορφή - 1NF,
- ✓ Δεύτερη κανονική μορφή - 2NF,
- ✓ Τρίτη κανονική μορφή - 3NF,
- ✓ Boyce - Codd κανονική μορφή - BCNF.

Οι συγκεκριμένες κανονικές μορφές συνιστούν περιοριστικές προδιαγραφές, οι οποίες αυξάνουν ανάλογα με την τάξη τους: κάθε BCNF πίνακας είναι επίσης και 3NF, κάθε πίνακας 3NF είναι επίσης και 2NF, κάθε 2NF πίνακας είναι επίσης και 1NF. Οι σημαντικότερες πάντως κανονικές μορφές, από πλευράς σχεδιασμού βάσεων δεδομένων, αποτελούν η 3NF και η BCNF.

Πριν ένας πίνακας προσδιοριστεί ότι είναι σε πρώτη κανονική μορφή, θεωρείται ότι βρίσκεται σε μη κανονική μορφή. Αυτή η μορφή είναι απλά ένα σύνολο δεδομένων, χωρίς κλειδιά και χωρίς διαμοιρασμό των δεδομένων σε επιμέρους πίνακες. Τα στοιχεία ενός τέτοιου πίνακα δεν μπορούν να χρησιμοποιηθούν σε βάση δεδομένων.

Οι 3 πρώτες κανονικές μορφές διατυπώθηκαν από τον Edgar Codd, το 1970 και το 1972. Η πρώτη κανονική μορφή (1NF) ουσιαστικά εξαλείφει τα διπλότυπα στοιχεία. Ένας πίνακας είναι σε πρώτη κανονική μορφή, αν και μόνο αν, κάθε γνώρισμα έχει μια ατομική τιμή για κάθε εγγραφή. Για παράδειγμα, ένας πίνακας ο οποίος για το γνώρισμα "Μαθήματα που Παρακολουθεί" έχει 3 τιμές σε μια εγγραφή του φοιτητή Α προφανώς και δεν είναι σε πρώτη κανονική μορφή, ή όπως αλλιώς λέμε, την παραβιάζει. Πρακτικά αυτό σημαίνει ότι σε ένα "κελί" μιας εγγραφής δεν επιτρέπονται πολλαπλές τιμές, λίστες, επιμέρους υποσύνολα πεδίων κλπ. Για να μετατρέψουμε αυτό τον πίνακα σε πρώτη κανονική μορφή πρέπει να δημιουργήσουμε 3 ξεχωριστές εγγραφές, μια για κάθε

μάθημα που παρακολουθεί ο φοιτητής. Εναλλακτικά, μπορούμε να προσθέσουμε επιπλέον στήλες, μια για κάθε μάθημα, εφόσον βέβαια ξέρουμε τον μέγιστο αριθμό μαθημάτων που ο φοιτητής μπορεί να παρακολουθήσει.

Η πρώτη κανονική μορφή είναι ουσιαστικά η μόνη υποχρεωτική για όλους τους πίνακες μιας σχεσιακής βάσης δεδομένων και όλες οι άλλες κανονικές μορφές είναι προαιρετικές. Ναι μεν ένας τέτοιος πίνακας μπορεί να μην είναι καλά σχεδιασμένος, εντούτοις μπορεί άνετα να υλοποιηθεί σε ένα σύστημα διαχείρισης βάσεων δεδομένων. Επιπλέον, η πρώτη κανονική μορφή, σε αντίθεση με τις άλλες, δεν απαιτεί επιπλέον πληροφορίες, όπως συναρτησιακές εξαρτήσεις (Ramakrishnan και Gehrke, 2000).

Σε δεύτερη κανονική μορφή (2NF) είναι ένας πίνακας που ήδη βρίσκεται σε πρώτη κανονική μορφή και όλα τα γνωρίσματά του που δεν αποτελούν κλειδιά είναι συναρτησιακά πλήρως εξαρτώμενα από το κύριο κλειδί. Το πλήρως εξαρτώμενο έχει την εξής έννοια: σε έναν πίνακα R, ένα γνώρισμα A είναι πλήρως συναρτησιακά εξαρτώμενο από ένα γνώρισμα ή συνδυασμό γνωρισμάτων X του R, εάν το A είναι συναρτησιακά εξαρτώμενο από το X αλλά όχι από κάποιο υποσύνολο του X.

Η δεύτερη κανονική μορφή απαιτεί όλα τα δεδομένα του πίνακα να σχετίζονται κατευθείαν με το θέμα του πίνακα, που συνήθως υποδεικνύεται από το πεδίο που παίζει το ρόλο του κύριου κλειδιού. Για να μετατρέψουμε έναν πίνακα από 1NF σε 2NF πρέπει να προσδιορίσουμε τις συναρτησιακές εξαρτήσεις και να απομονώσουμε τα γνωρίσματα που εξαρτώνται από τις ορίζουσες αυτών. Οι ορίζουσες αυτές, μαζί με τα γνωρίσματα που ορίζουν, μεταφέρονται σε νέο, ξεχωριστό πίνακα, ενώ ο αρχικός πίνακας εξακολουθεί να διατηρεί τα γνωρίσματά του, πλην όμως των εξαρτώμενων γνωρισμάτων της ορίζουσας. Η διαδικασία αυτή είναι γνωστή ως διάσπαση (ή αποσύνθεση) πινάκων και είναι παρόμοια για την 3NF και την BCNF.

Στην τρίτη κανονική μορφή (3NF) εξαλείφουμε τις μεταβατικές συναρτησιακές εξαρτήσεις με τη βοήθεια του αξιώματος της μεταβατικότητας, καθώς αυτές προκαλούν ανωμαλίες εισαγωγής, διαγραφής και ενημέρωσης. Ένας πίνακας είναι σε 3NF αν, όταν ισχύει μια μη-τετριμμένη συναρτησιακή εξάρτηση $X \rightarrow A$, τότε το X είναι υπερκλειδί ή το A είναι μέλος κάποιου υποψήφιου κλειδιού.

Πρακτικά, η τρίτη κανονική μορφή εξαλείφει τα πεδία του πίνακα που μπορεί να παράγονται από άλλα πεδία. Η ουσία της τρίτης κανονικής μορφής είναι ότι κάθε γνώρισμα που δεν είναι κλειδί, είναι συναρτησιακά εξαρτώμενο από το κλειδί του πίνακα κι από κανένα άλλο γνώρισμα. Η 3NF είναι επαρκής για πίνακες που έχουν ένα μονό υποψήφιο κλειδί, αλλά αποδείχτηκε ανεπαρκής στις περιπτώσεις όπου υπάρχουν

πολλαπλά, σύνθετα υποψήφια κλειδιά. Για να αντιμετωπιστεί αυτό το θέμα, οι Boyce και Codd (1974) δημιούργησαν την BCNF μορφή.

3.2.7 Κανονική μορφή Boyce - Codd (BCNF)

Το 1974, ο Edgar Codd, μαζί με τον Raymond Boyce, επαναδιατύπωσαν την τρίτη κανονική μορφή, αναβαθμίζοντάς την σε "3.5". Η μορφή αυτή είναι γνωστή ως BCNF. Για να βρίσκεται ένας πίνακας σε BCNF μορφή θα πρέπει να έχουν απομακρυνθεί όλοι οι πλεονασμοί με τη βοήθεια των συναρτησιακών εξαρτήσεων και εάν, για κάθε μια μη-τετριμμένη συναρτησιακή εξάρτηση A → B, το A να είναι υπερκλειδί.

Επομένως, για να ελέγξουμε αν ένας πίνακας είναι σε BCNF μορφή, προσδιορίζουμε όλες τις ορίζουσες των συναρτησιακών εξαρτήσεων που σχετίζονται με αυτόν και επιβεβαιώνουμε ότι είναι υπερκλειδιά. Εάν δεν είναι, τότε η συναρτησιακή εξάρτηση παραβιάζει την BCNF μορφή, οπότε διασπάμε περαιτέρω τον πίνακα μέχρι να έχουμε μια ομάδα πινάκων που όλοι τους να είναι σε BCNF μορφή. Η συνολική σχεδίαση μιας βάσης δεδομένων είναι σε BCNF μορφή αν κάθε μέλος του συνόλου των πινάκων της είναι σε BCNF μορφή. Χρησιμοποιώντας τη διαδικασία διάσπασης σε BCNF μπορούμε να πάρουμε τους επιθυμητούς, τελικούς BCNF πίνακες χωρίς να χρειάζεται πρώτα να κανονικοποιήσουμε σε 1NF, 2NF και 3NF.

Οι BCNF πίνακες συνήθως δεν παρουσιάζουν ανωμαλίες εισαγωγής, ενημέρωσης (δηλαδή τα δεδομένα να έχουν αντιφατικές - αντικρουόμενες πληροφορίες όποτε τα ενημερώνουμε ή προσπαθούμε να τα ανακτήσουμε) και διαγραφής (όπου μπορεί άθελά μας να διαγραφούν στοιχεία χρήσιμα για μας).

Η συνολική διαδικασία της διάσπασης έχει ως εξής: Αρχικά αναλύουμε τις απαιτήσεις που θα έχει η βάση δεδομένων και συγκεντρώνουμε εκείνα τα γνωρίσματα που μας ενδιαφέρουν σε έναν ενιαίο, universal πίνακα τον οποίο ονομάζουμε R. Στη συνέχεια, προσδιορίζουμε τις συναρτησιακές εξαρτήσεις μεταξύ των γνωρισμάτων και με τη βοήθειά τους κάνουμε σταδιακά τις διασπάσεις έως ότου φτάσουμε σε μικρότερους, BCNF πίνακες. Δεν χρησιμοποιούμε για τις διασπάσεις τις συναρτησιακές εξαρτήσεις που η ορίζουσά τους αποτελεί υποσύνολο του υποψήφιου κλειδιού του πίνακα R. Ο αλγόριθμος της διάσπασης μπορεί να περιγραφεί ως ένας βρόχος που εξετάζει με τη σειρά τους πίνακες που προκύπτουν μετά από κάθε διάσπαση, ξεκινώντας πρώτα από τον universal πίνακα. Ο βρόχος αυτός "τρέχει" ενόσω υπάρχει έστω κι ένας

πίνακας που δεν είναι σε μορφή BCNF. Όσο λοιπόν ικανοποιείται αυτή η συνθήκη, συμβαίνουν δύο τινά:

1) Εντοπίζεται μια συναρτησιακή εξάρτηση X → Y (όπου X, Y σύνολα γνωρισμάτων) στον πίνακα R_i η οποία παραβιάζει την BCNF μορφή. Αν συμβαίνει αυτό, τότε εφαρμόζεται το επόμενο βήμα.

2) Διασπάται ο πίνακας R_i σε δύο νέους πίνακες. Ο ένας πίνακας, με την ονομασία R_{i+1} περιλαμβάνει τα γνωρίσματα X και Y και ο άλλος (R_{i+2}) περιλαμβάνει όλα τα γνωρίσματα του πίνακα R πλην μόνο των γνωρισμάτων Y.

Περιέχει δηλαδή κι ο δεύτερος πίνακας την ορίζουσα X, η οποία έτσι θα λειτουργεί ως συνδετικός κρίκος με τον πρώτο πίνακα στον οποίο θα λειτουργεί ως κλειδί. Οι δύο αυτοί πίνακες, την επόμενη φορά που θα εκτελεστεί ο βρόχος, θα εξεταστούν κατά πόσο είναι σε μορφή BCNF, κι αν δεν είναι, θα διασπαστούν περαιτέρω με τη βοήθεια και των υπολοίπων συναρτησιακών εξαρτήσεων.

Η διαδικασία διάσπασης ενός πίνακα R_i σε δύο νέους μπορεί να περιγραφεί με τον εξής ψευδοκώδικα:

```
Εκτέλεση βρόχου που εκτελεί κατά σειρά τις συναρτησιακές εξαρτήσεις F
      Αν η Fx παραβιάζει την BCNF του Ri τότε
            new Ri+1 = όλα τα γνωρίσματα της Fx
            new Ri+2 = Ri χωρίς τα εξαρτώμενα γνωρίσματα της Fx
(σημείωση: ο δείκτης i στην πρόσθεση λειτουργεί ως χαρακτήρας string)
```

Ο στόχος είναι όλοι οι τελικοί πίνακες που θα προκύψουν να είναι BCNF. Επίσης, είναι επιθυμητό να διατηρηθούν τα γνωρίσματα και οι συναρτησιακές εξαρτήσεις. Η διατήρηση των γνωρισμάτων σημαίνει πως κάθε γνώρισμα του αρχικού πίνακα θα εμφανίζεται τουλάχιστον σε έναν από τους τελικούς πίνακες. Η διατήρηση των συναρτησιακών εξαρτήσεων σημαίνει ότι θέλουμε κατά τη διαδικασία των διασπάσεων να εξασφαλιστεί πως τα γνωρίσματα της κάθε συναρτησιακής εξάρτησης ξεχωριστά βρίσκονται, αν είναι δυνατό, στον ίδιο τελικό πίνακα.

Παρατίθεται τώρα ένα παράδειγμα διάσπασης, με βάση τα όσα προαναφέρθηκαν[1]. Έστω ο πίνακας R(Α, Β, Γ, Δ, Ε, Ζ, Η, Θ, Ι, Κ) και οι κάτωθι συναρτησιακές εξαρτήσεις. Ζητείται να διασπαστεί ο R κατά BCNF.

(1) Α, Β → Γ

(2) Α → Δ, Ε

(3) Β → Ζ

[1] Η άσκηση έχει ληφθεί από το μάθημα "Βάσεις Δεδομένων" του μεταπτυχιακού (α' εξάμηνο 2015), υπό τους καθηγητές κ. Γεώργιο Ευαγγελίδη και κα. Γεωργία Κολωνιάρη

(4) Z → H, Θ

(5) Δ → I, K

Υπολογίζοντας τον εγκλεισμό των γνωρισμάτων (ξεχωριστά και ανά συνδυασμούς) καταλήγουμε στο συμπέρασμα ότι μόνο ο εγκλεισμός του συνδυασμού A, B περιλαμβάνει το σύνολο των γνωρισμάτων του πίνακα R, επομένως το {A, B} είναι υποψήφιο κλειδί του R.

Πριν ξεκινήσουμε τη διάσπαση, παραβλέπουμε τις συναρτησιακές εξαρτήσεις στων οποίων τις ορίζουσες είναι υποσύνολο το κλειδί {A, B}. Προφανώς είναι μόνο η 1η, οπότε δεν θα τη λάβουμε υπόψη κατά τις διασπάσεις. Επιπλέον, αγνοούμε όσες συναρτησιακές εξαρτήσεις είναι τετριμμένες.

Στη συνέχεια, ελέγχουμε σε ποιες από τις συναρτησιακές εξαρτήσεις μπορούμε να εφαρμόσουμε τον κανόνα της μεταβατικότητας του Armstrong. Αρχικά, εντοπίζουμε τη 2η συναρτησιακή εξάρτηση, αφού το γνώρισμα {Δ} από το δεξί σκέλος της είναι υποσύνολο της ορίζουσας της 5ης συναρτησιακής εξάρτησης. Οπότε, τροποποιούμε την 2η συναρτησιακή εξάρτηση, προσθέτοντας στα δεξιά της τον εγκλεισμό του $\{A\}^+$, δηλαδή A → Δ, E, I, K. Ομοίως και για την 3η συναρτησιακή εξάρτηση, αφού με τον κανόνα της επαγωγής, αφού B → Z και Z → H, Θ, ισχύει ότι B → Z, H, Θ.

Επομένως, οι συναρτησιακές εξαρτήσεις, τις οποίες τελικά θα λάβουμε υπόψη για τις διασπάσεις, διαμορφώνονται ως εξής:

(1) A → Δ, E, I, K

(2) B → Z, H, Θ

(3) Z → H, Θ

(4) Δ → I, K

Ξεκινάμε τη διάσπαση ελέγχοντας μία προς μία τις 4 διαθέσιμες συναρτησιακές εξαρτήσεις. Αν κάποια από αυτή εμπεριέχεται μέσα στον πίνακα που θέλουμε να διασπάσουμε κι επιπλέον υπάρχει έστω και ένα επιπλέον γνώρισμα, τότε λέμε ότι η συναρτησιακή εξάρτηση παραβιάζει την BCNF μορφή, οπότε κάνουμε τη διάσπαση.

Έτσι, η 1η συναρτησιακή εξάρτηση A → Δ, E, I, K περιλαμβάνεται στον πίνακα R και πλέον αυτής υπάρχουν και άλλα γνωρίσματα. Οπότε, ο πίνακας R διασπάται σε δύο νέους πίνακες: ο πρώτος θα περιέχει τα γνωρίσματα της συναρτησιακής εξάρτησης και θα έχει ως κλειδί την ορίζουσα, δηλαδή R_1{♠A, Δ, E, I, K} και ο δεύτερος θα περιέχει την ορίζουσα {A} συν τα υπόλοιπα γνωρίσματα του πίνακα R (πλην δηλαδή των {Δ, E, I, K}), δηλαδή R_2{♠A, ♠B, Γ, Z, H, Θ) με κλειδιά τον συνδυασμό {A, B}. Σημειώνεται ότι αμέσως μετά από κάθε διάσπαση, επαναπροσδιορίζονται τα κλειδιά των

νέων πινάκων, με βάση τις συναρτησιακές εξαρτήσεις που τους αφορούν. Τα κλειδιά αυτά θα πρέπει να προσδιορίζουν το σύνολο των γνωρισμάτων των νέων πινάκων.

Έχουμε πια δύο νέους πίνακες, τους R_1 και R_2. Ελέγχουμε αν σε αυτούς τους πίνακες παραβιάζεται η BCNF από τις 3 συναρτησιακές εξαρτήσεις που απέμειναν.

Στην περίπτωση του R_1 παραβιάζεται η BCNF με τη 4η συναρτησιακή εξάρτηση Δ → I, K, αφού τα {Δ, I, K} περιλαμβάνονται σε αυτόν και επιπλέον υπάρχουν κι άλλα γνωρίσματα. Οπότε, διασπάμε τον πίνακα σε R_{11}{♠Δ, I, K} με κλειδί το Δ και R_{12}{♠A, Δ, E} με κλειδί το A.

Στην περίπτωση του R_2 βλέπουμε ότι παραβιάζεται η BCNF μορφή από την 2η συναρτησιακή εξάρτηση αφού περιλαμβάνονται όλα της τα γνωρίσματα στον πίνακα και επιπλέον υπάρχουν κι άλλα γνωρίσματα σε αυτόν. Τον διασπάμε επομένως με τη βοήθεια της B → Z, H Θ και προκύπτουν δύο νέοι πίνακες, ο R_{21}{♠B, Z, H, Θ}με κλειδί το B και ο R_{22}{♠A, ♠B, Γ} με κλειδί το συνδυασμό {A, B}, λόγω της συναρτησιακής εξάρτησης A, B → Γ που καθιστά κλειδί του πίνακα τον συγκεκριμένο συνδυασμό.

Μία επισήμανση για την αρίθμηση των πινάκων: αφού διασπάμε τον πίνακα R_v, οι πίνακες που θα προκύψουν μπορούν να έχουν είτε την αρίθμηση της μορφής R_{v1}, R_{v2}, κλπ ή διαδοχική αρίθμηση για το σύνολο των πινάκων, της μορφής R_1, R_2, ..., R_v. Με την πρώτη προσέγγιση μπορούμε πιο εύκολα να εντοπίζουμε, από ποιου πίνακα τη διάσπαση προήλθαν οι δύο νέοι πίνακες. Επιπλέον, όπως αναφέρθηκε και σε προηγούμενα κεφάλαια, δεν παίζει κανένα ρόλο η σειρά με την οποία εμφανίζονται τα γνωρίσματα στις συναρτησιακές εξαρτήσεις και τους πίνακες.

Έτσι, έχουμε πλέον 4 πίνακες: τους R_{11}, R_{12}, R_{21} και R_{22}, καθώς και μία συναρτησιακή εξάρτηση που απέμεινε να ελέγξουμε, την Z → H, Θ. Βλέπουμε ότι αυτή παραβιάζει την BCNF μορφή του πίνακα R_{21}, οπότε αυτός διασπάται σε R_{221}{♠Z, H, Θ} και R_{222}{♠B, Z}. Καθώς πλέον δεν υπάρχουν άλλες διαθέσιμες συναρτησιακές εξαρτήσεις, οι δύο τελευταίοι πίνακες είναι BCNF και η διαδικασία ολοκληρώνεται.

Επομένως, με τη βοήθεια του αλγορίθμου της κανονικοποίησης και των συναρτησιακών εξαρτήσεων, διασπάσαμε τον αρχικό πίνακα R(A, B, Γ, Δ, E, Z, H, Θ, I, K) στους εξής BCNF πίνακες:

R_{11}{♠Δ, I, K}

R_{12}{♠A, Δ, E}

R_{22}{♠A, ♠B, Γ}

R_{221}{♠Z, H, Θ}

R_{222}{♠B, Z}

3.3 Παραδείγματα δημιουργίας σχήματος με κανονικοποίηση

3.3.1 Παράδειγμα 1ο: Βάση δεδομένων για χρηματιστηριακή ενημέρωση

Περιγράφεται στη συνέχεια η διαδικασία δημιουργίας σχήματος βάσης δεδομένων χρηματιστηριακής ενημέρωσης με κανονικοποίηση. Η διαδικασία υλοποιήθηκε με τη βοήθεια του προγράμματος - εκπαιδευτικού εργαλείου.

Αρχικά, αναλύουμε τις απαιτήσεις και τις λειτουργίες της βάσης δεδομένων. Συγκεντρώνουμε όλα τα χαρακτηριστικά - γνωρίσματα των οποίων οι πληροφορίες μας ενδιαφέρουν, κι αυτά παρουσιάζονται παρακάτω. Σημειωτέον ότι εμφανίζονται σε αλφαβητική σειρά καθαρά για πρακτικούς λόγους, αφού στην πραγματικότητα η σειρά καταχώρησης των γνωρισμάτων σε έναν πίνακα δεν έχει καμία απολύτως σημασία. Επιπλέον, για να περιοριστεί το πλήθος των γνωρισμάτων που καταχωρούνται στον universal πίνακα, ομαδοποιούμε τα κοινά γνωρίσματα με ένα κοινό όνομα, εξαιρώντας όμως από την ομαδοποίηση το γνώρισμα εκείνο που προσδιορίζει κάποια άλλα, π.χ. το com_symbol προσδιορίζει όλη την ομάδα COM.

- ✓ bra_name: η ονομασία του κλάδου
- ✓ com_symbol: το σύμβολο της μετοχής

COM
- ✓ com_category: η κατηγορία όπου διαπραγματεύεται η μετοχή
- ✓ com_gdpart: υποδηλώνει αν η μετοχή περιλαμβάνεται στον Γενικό Δείκτη
- ✓ com_name: η ονομασία της εταιρείας
- ✓ com_sharesnum: το πλήθος μετοχών της εταιρείας

- ✓ ftse_name: η ονομασία του δείκτη FTSE
- ✓ ftse_revision: η ημερομηνία τελευταίας αναθεώρησης του δείκτη FTSE
- ✓ included_weigh: το ποσοστό στάθμισης μιας μετοχής στον δείκτη FTSE

PART
- ✓ part_acts: το πλήθος των πράξεων της μετοχής σε μια συνεδρίαση
- ✓ part_close: η τιμή κλεισίματος της μετοχής σε μια συνεδρίαση
- ✓ part_high: το υψηλό ημέρας της μετοχής σε μια συνεδρίαση
- ✓ part_low: το χαμηλό ημέρας της μετοχής σε μια συνεδρίαση
- ✓ part_open: η τιμή ανοίγματος της μετοχής σε μια συνεδρίαση
- ✓ part_value: η αξία συναλλαγών της μετοχής σε μια συνεδρίαση
- ✓ part_volume: ο όγκος συναλλαγών της μετοχής σε μια συνεδρίαση

- ✓ pub_com_id: ο αύξων αριθμός της δημοσίευσης, ξεχωριστός ανά εταιρεία

PUB
- ✓ pub_date: η ημερομηνία καταχώρησης της δημοσίευσης
- ✓ pub_intro: το κείμενο της πρώτης παραγράφου της δημοσίευσης
- ✓ pub_link: ο ηλεκτρονικός σύνδεσμος της δημοσίευσης
- ✓ pub_mme: η ονομασία του μέσου όπου αναρτήθηκε η δημοσίευση

- ✓ ses_date: η ημερομηνία της συνεδρίασης
- ✓ ses_gd: η τιμή κλεισίματος του Γενικού Δείκτη της συνεδρίασης
- ✓ sha_afm: το ΑΦΜ του μετόχου

SHA
- ✓ sha_email: η διεύθυνση email του μετόχου
- ✓ sha_entity: υποδηλώνει αν ο μέτοχος είναι ΝΠΔΔ
- ✓ sha_name: το όνομα του μετόχου

- ✓ sha_com_sharesnum: το πλήθος μετοχών του μετόχου σε μια εταιρεία
- ✓ sheet_id: ο α/α καταχώρησης του ισολογισμού

SHEET
- ✓ sheet_bookvalue: το ενεργητικό του ισολογισμού
- ✓ sheet_equity: τα ίδια κεφάλαια του ισολογισμού
- ✓ sheet_profits: τα καθαρά κέρδη του ισολογισμού
- ✓ sheet_year: η χρονιά χρήσης που αφορά ο ισολογισμός
- ✓ sheet_turnover: ο κύκλος εργασιών του ισολογισμού

Τοποθετούμε τα παραπάνω γνωρίσματα στον universal πίνακα, ονόματι R:

R(*bra_name, com_symbol, COM, ftse_name, ftse_revision, included_weigh, PART, pub_com_id, PUB, ses_date, ses_gd, sha_afm, sha_com_sharesnum, SHA, sheet_id, SHEET*)

Στη συνέχεια, ορίζουμε το σύνολο των συναρτησιακών εξαρτήσεων F, ανάλογα με τις απαιτήσεις που αναλύθηκαν κατά τον προσδιορισμό των γνωρισμάτων. Στην ουσία, οι συναρτησιακές εξαρτήσεις λειτουργούν ως περιορισμοί ακεραιότητας των δεδομένων που εισάγονται στο φυσικό σχήμα και οι οποίοι δεν μπορούν να παραβιάζονται. Για παράδειγμα, καθώς κάθε σύμβολο μετοχής αντιστοιχεί σε μια εταιρεία, μια καταχώρηση δεδομένων των παρακάτω δύο εγγραφών, παραβιάζει τον περιορισμό ακεραιότητας δεδομένων που έχει οριστεί, αφού στο σύμβολο μετοχής ΑΙΟΛΚ δεν δύναται να αντιστοιχούν δύο διαφορετικές εταιρείες κι επιπλέον στην ίδια συνεδρίαση της 11/03/2016 δεν μπορούν να αντιστοιχούν δύο διαφορετικές τιμές.

- ✓ Εγγραφή: 1
  Ημερομηνία συνεδρίασης: 11/03/2016
  Σύμβολο μετοχής: ΑΙΟΛΚ
  Ονομασία εταιρεία: ΑΙΟΛΙΚΗ ΑΕΕΧ
  Τιμή κλεισίματος μετοχής: 0,50

- ✓ Εγγραφή: 2
  Ημερομηνία συνεδρίασης: 11/03/2016
  Σύμβολο μετοχής: ΑΙΟΛΚ
  Ονομασία εταιρεία: Τράπεζα Πειραιώς
  Τιμή κλεισίματος μετοχής: 0,54

Με το σκεπτικό αυτό, προσδιορίζουμε τις πρώτες συναρτησιακές εξαρτήσεις. Έτσι, το σύμβολο της μετοχής προσδιορίζει το όνομα της εταιρείας, το πλήθος μετοχών της, την κατηγορία στην οποία διαπραγματεύεται και το αν συμμετέχει ή όχι στον Γενικό Δείκτη, δηλαδή την ομάδα COM. Επίσης, κάθε εταιρεία ανήκει σε έναν κλάδο εταιρειών, επομένως το σύμβολο μιας μετοχής προσδιορίζει το όνομα του κλάδου στον οποίο ανήκει. Επομένως έχουμε τις εξής δύο συναρτησιακές εξαρτήσεις:

- com_symbol → COM
- com_symbol → bra_name

Χρησιμοποιώντας τον κανόνα της ένωσης (αν Α→Β και Α→Γ, τότε Α→ Β,Γ) συνενώνουμε τα δεξιά σκέλη των συναρτησιακών εξαρτήσεων, οπότε τελικά έχουμε την πρώτη συναρτησιακή εξάρτηση:

(1) com_symbol → COM, bra_name

Το όνομα του δείκτη FTSE δίνει την ημερομηνία που αυτός αναθεωρήθηκε για τελευταία φορά, καθώς κάθε δείκτης FTSE (π.χ. των τραπεζών, των εταιρειών πληροφορικής κλπ) έχει τη δική του ημερομηνία αναθεώρησης. Επομένως, ισχύει:

(2) ftse_name → ftse_revision

Το ποσοστό στάθμισης (συμμετοχής) που έχει κάθε μετοχή σε έναν δείκτη FTSE δίνεται από το γνώρισμα included_weigh. Επομένως, το γνώρισμα αυτό δίνεται από τον συνδυασμό των γνωρισμάτων του συμβόλου της μετοχής και του ονόματος του δείκτη:

(3) com_symbol, ftse_name → included_weigh

Για να δούμε ποιες δημοσιεύσεις αφορούν κάθε εταιρεία, πρέπει να συνδυάσουμε το γνώρισμα com_symbol με τον κωδικό της δημοσίευσης pub_com_id, κι αυτό γιατί κάθε εταιρεία έχει τους δικούς της αύξοντες αριθμούς δημοσιεύσεων. Έτσι π.χ. η δημοσίευση με α/α 4 για την εταιρεία ΑΙΟΛΙΚΗ μπορεί να είναι τελείως διαφορετική από αυτή που έχει τον ίδιο α/α, αλλά αφορά την εταιρεία ΟΤΕ. Αυτό γίνεται γιατί κάθε δημοσίευση μπορεί να αφορά περισσότερες από μία εταιρείες ταυτόχρονα. Σε διαφορετική περίπτωση, αν δηλαδή κάθε δημοσίευση αφορούσε μόνο μια εταιρεία, τότε

πολύ απλά θα δίναμε έναν ενιαίο α/α για όλες τις δημοσιεύσεις, κι αυτός θα προσδιόριζε το σύμβολο της μετοχής.

Δημιουργούμε επομένως την κάτωθι συναρτησιακή εξάρτηση, η οποία έχει ως εξαρτημένη μεταβλητή την PUB, δηλαδή την ομάδα γνωρισμάτων, όπως το ηλεκτρονικό μέσο όπου καταχωρήθηκε και φιλοξενείται η δημοσίευση, τον ηλεκτρονικό σύνδεσμο όπου αυτή αναρτήθηκε κλπ.

(4) com_symbol, pub_com_id → PUB

Για να δούμε τις πληροφορίες ενός μετόχου χρειαζόμαστε το ΑΦΜ του. Με αυτό μπορούμε να δούμε για παράδειγμα το όνομά του, το email του κλπ. Έχουμε λοιπόν:

(5) sha_afm → SHA

Θα πρέπει εδώ να επισημανθεί ότι επίτηδες δεν συμπεριλάβαμε στην ομάδα γνωρισμάτων SHA το γνώρισμα sha_com_sharesnum. Αυτό έγινε γιατί το sha_com_sharesnum αφορά το πλήθος μετοχών του μετόχου σε μια εταιρεία. Ένας μέτοχος μπορεί να έχει για παράδειγμα 50.000 μετοχές στην εταιρεία ΑΙΟΛΙΚΗ και άλλες 5.000 μετοχές στην εταιρεία ΟΤΕ. Επομένως, δεν μπορούμε να δεχτούμε ότι έχοντας μόνο το ΑΦΜ ενός μετόχου μπορούμε να προσδιορίσουμε το πλήθος των μετοχών που κατέχει. Χρειάζεται να συνδυαστεί το ΑΦΜ με το σύμβολο της μετοχής στην οποία συμμετέχει ο μέτοχος. Επομένως, δημιουργούμε τη συναρτησιακή εξάρτηση:

(6) sha_afm, com_symbol → sha_sharesnum

Κάθε ισολογισμός προσδιορίζεται μοναδικά από ένα όνομα αρχείου στο server όπου θα φιλοξενείται. Αυτό το όνομα μπορεί απλά να είναι ένας αύξοντας αριθμός (sheet_id), λειτουργώντας έτσι ως κλειδί. Οπότε, έχοντας το sheet_id μπορούμε να πάρουμε πληροφορίες για το σε ποια εταιρεία ανήκει και τα επιμέρους οικονομικά στοιχεία της ομάδας SHEET που περιλαμβάνει ο ισολογισμός, δηλαδή τον κύκλο εργασιών του, τα καθαρά κέρδη κλπ. Επομένως, έχουμε:

(7) sheet_id → com_symbol, SHEET

Η ημερομηνία μιας συνεδρίασης μας δίνει την τιμή κλεισίματος του Γενικού Δείκτη, αφού κάθε συνεδρίαση μπορεί να έχει μόνο μία τιμή. Αν δηλαδή δύο εγγραφές με ίδια ημερομηνία 12/03/2016, έχουν διαφορετικές τιμές κλεισίματος, π.χ. 600 και 650 μονάδες, τότε παραβιάζεται η ακεραιότητα των δεδομένων. Επομένως, έχουμε:

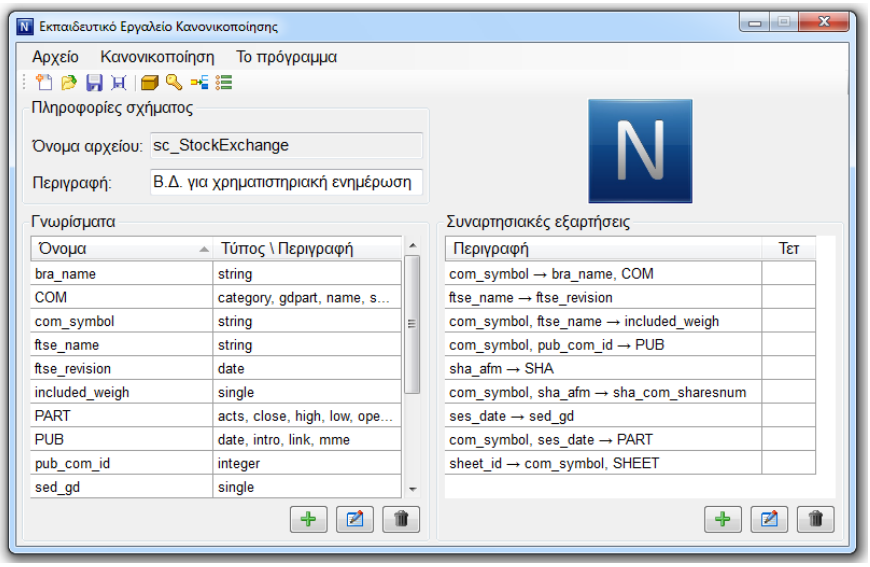
(8) ses_date → ses_gd

Σε κάθε συνεδρίαση του ΧΑ αγοράζονται και πωλούνται μετοχές και διαμορφώνονται οι τιμές και τα στατιστικά της ημέρας. Έτσι, στο τέλος της συνεδρίασης, καταχωρούνται στοιχεία όπως τιμή κλεισίματος της μετοχής, αξία

συναλλαγών κλπ. Επομένως, για να προσδιορίσουμε ένα από αυτά τα γνωρίσματα, π.χ. την τιμή κλεισίματος της μετοχής ΑΙΟΛΚ για τις 12/03/2016, θα πρέπει να συνδυάσουμε την ημερομηνία της συνεδρίασης και το σύμβολο της μετοχής. Έτσι, λόγω του περιορισμού ακεραιότητας δεδομένων, αποκλείεται σε μια πλειάδα εγγραφών, μια μετοχή στην ίδια συνεδρίαση να εμφανίζει δύο διαφορετικές τιμές κλεισίματος. Δημιουργούμε, επομένως, την εξής συναρτησιακή εξάρτηση:

(9) ses_date, com_symbol → PART

Συνοψίζοντας τα παραπάνω, έχουμε 17 γνωρίσματα και 9 συναρτησιακές εξαρτήσεις. Στο πρόγραμμα Εκπαιδευτικό Εργαλείο Κανονικοποίησης δημιουργούμε ένα νέο σχήμα. Καταχωρούμε τα γνωρίσματα, εισάγοντας το όνομα του καθενός και προαιρετικά τον τύπο ή την περιγραφή του γνωρίσματος. Επίσης, εισάγουμε τις συναρτησιακές εξαρτήσεις που περιγράφηκαν παραπάνω (Εικόνα 2).



Εικόνα 2: Το σχήμα της χρηματιστηριακής ενημέρωσης

Αποθηκεύουμε το σχήμα με την ονομασία sc_StockExchange και συνεχίζουμε με τη διαδικασία της κανονικοποίησης. Αρχικά, ελέγχουμε το υποψήφιο κλειδί του πίνακα και προκύπτει ότι αυτό είναι ο συνδυασμός των γνωρισμάτων {com_symbol, ftse_name, pub_com_id, ses_date, sha_afm, sheet_year}. Στη συνέχεια, με την εντολή "Κανονικοποίηση > Διάσπαση" το πρόγραμμα κανονικοποιεί τον αρχικό πίνακα με βάση τις συναρτησιακές εξαρτήσεις που ορίσαμε και προκύπτει η εξής ακολουθία διασπάσεων (αντιγραμμένη από το παράθυρο αποτελεσμάτων του προγράμματος):

Με την "com_symbol → bra_name, COM" ο R διασπάται σε:

R1{♠com_symbol, bra_name, COM} (BCNF)

R2{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, ftse_revision, included_weigh, PART, PUB, sed_gd, SHA, sha_com_sharesnum, SHEET}

Με την "ftse_name → ftse_revision" ο R2 διασπάται σε:

R3{♠ftse_name, ftse_revision} (BCNF)

R4{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, included_weigh, PART, PUB, sed_gd, SHA, sha_com_sharesnum, SHEET}

Με την "com_symbol, ftse_name → included_weigh" ο R4 διασπάται σε:

R5{♠com_symbol, ♠ftse_name, included_weigh} (BCNF)

R6{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, PART, PUB, sed_gd, SHA, sha_com_sharesnum, SHEET}

Με την "com_symbol, pub_com_id → PUB" ο R6 διασπάται σε:

R7{♠com_symbol, ♠pub_com_id, PUB} (BCNF)

R8{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, PART, sed_gd, SHA, sha_com_sharesnum, SHEET}

Με την "sha_afm → SHA" ο R8 διασπάται σε:

R9{♠sha_afm, SHA} (BCNF)

R10{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, PART, sed_gd, sha_com_sharesnum, SHEET}

Με την "com_symbol, sha_afm → sha_com_sharesnum" ο R10 διασπάται σε:

R11{♠com_symbol, ♠sha_afm, sha_com_sharesnum} (BCNF)

R12{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, PART, sed_gd, SHEET}

Με την "ses_date → sed_gd" ο R12 διασπάται σε:

R13{♠ses_date, sed_gd} (BCNF)

R14{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, PART, SHEET}

Με την "com_symbol, ses_date → PART" ο R14 διασπάται σε:

R15{♠com_symbol, ♠ses_date, PART} (BCNF)

R16{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id, com_symbol, SHEET}

Με την "sheet_id → com_symbol, SHEET" ο R16 διασπάται σε:

R17{♠sheet_id, com_symbol, SHEET} (BCNF)

R18{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id} (BCNF)

Οι πίνακες BCNF είναι οι εξής:

R1{♠com_symbol, bra_name, COM}

R3{♠ftse_name, ftse_revision}

R5{♠com_symbol, ♠ftse_name, included_weigh}

R7{♠com_symbol, ♠pub_com_id, PUB}

R9{♠sha_afm, SHA}

R11{♠com_symbol, ♠sha_afm, sha_com_sharesnum}

R13{♠ses_date, sed_gd}

R15{♠com_symbol, ♠ses_date, PART}

R17{♠sheet_id, com_symbol, SHEET}

R18{♠ftse_name, ♠pub_com_id, ♠ses_date, ♠sha_afm, ♠sheet_id}

Στην περίπτωση του πίνακα R18, είναι μεν πίνακας BCNF, ωστόσο σαν ιδέα δεν είναι καλή αφού σαν δομή πίνακα δεν έχει κάποιο νόημα και επιπλέον θα δέχεται πολλές, επαναλαμβανόμενες πληροφορίες. Οι κανονικές μορφές που βασίζονται σε συναρτησιακές εξαρτήσεις δεν είναι ικανοποιητικές για την αντιμετώπιση καταστάσεων όπως αυτή. Για αυτό το λόγο, έχουν οριστεί άλλες εξαρτήσεις και κανονικές μορφές.

Για να αντιμετωπιστεί αυτό το πρόβλημα, μπορούμε να ορίσουμε μια νέα μορφή περιορισμού, που ονομάζεται εξάρτηση πολλαπλών τιμών ή πλειότιμη εξάρτηση (Teorey, 2006). Με ένα σύνολο πλειότιμων εξαρτήσεων μπορούμε να δημιουργήσουμε την 4η κανονική μορφή (4NF), την οποία ανέπτυξε ο Ronald Fagin (1977), ωστόσο καθώς δεν αποτελεί αντικείμενο της παρούσας εργασίας και του προγράμματος, αφήνεται ως μια δυνητική μελλοντική επέκταση της εργασίας.

3.3.2 Παράδειγμα 2ο: Βάση δεδομένων για εργαστήριο ζαχαροπλαστικής

Έστω ένα εργαστήριο ζαχαροπλαστικής που θέλει να δημιουργήσει μια απλή βάση δεδομένων η οποία να περιέχει πληροφορίες με τα υλικά παραγωγής που διαθέτει ανά πάσα στιγμή στην αποθήκη του, τα προϊόντα που δημιουργεί από αυτά, καθώς και τις παραγγελίες που πρέπει να παραδώσει. Επιπλέον, σκοπός της βάσης δεδομένων είναι να έχει μια εικόνα του κόστους παραγωγής ανά προϊόν, ώστε να προσδιορίζει την τιμή πώλησης σε επίπεδο τέτοιο που να αποφέρει κέρδος. Επίσης, να γνωρίζει τις καταληκτικές ημερομηνίες ανά παραγγελία που πρέπει να παραδώσει. Κάθε παραγγελία μπορεί να συνδυάζει διαφορετικές ποσότητες από τα προϊόντα που παράγει.

Μελετώντας τις παραπάνω απαιτήσεις του εργαστηρίου ζαχαροπλαστικής, προκύπτουν τα εξής γνωρίσματα:

- ✓ ingr_name: ονομασία υλικού, μοναδική για κάθε ένα από αυτά
- ✓ ingr_volume: η διαθέσιμη ποσότητα του υλικού στην αποθήκη
- ✓ ingr_avgcost: το μέσο κόστος αγοράς του διαθέσιμου υλικού
- ✓ prod_name: η ονομασία του παραγόμενου προϊόντος, μοναδική για το καθένα
- ✓ prod_price: η ενδεικτική τιμή πώλησης ανά προϊόν
- ✓ prod_ingr_volume: η ποσότητα του υλικού που χρειάζεται για την παραγωγή μιας μονάδας προϊόντος
- ✓ ord_id: ο αύξων αριθμός της παραγγελίας, μοναδικός για κάθε μία
- ✓ ord_date: η ημερομηνία που πρέπει να παραδοθεί η παραγγελία
- ✓ ord_prod_volume: η ποσότητα του προϊόντος ανά παραγγελία

Συνεχίζουμε ορίζοντας τις συναρτησιακές εξαρτήσεις. Αρχικά, παρατηρούμε ότι το όνομα του υλικού προσδιορίζει τη διαθέσιμη ποσότητά του στην αποθήκη μας, καθώς και το μέσο κόστος αγοράς της. Η εφαρμογή που θα "τρέχει" την βάση δεδομένων, θα φροντίζει κάθε φορά που αγοράζονται νέες ποσότητες του προϊόντος και εισάγονται στην αποθήκη, να επαναϋπολογίζει και να ενημερώνει την νέα μέση τιμή κόστους ανά μονάδα προϊόντος. Επομένως, η πρώτη συναρτησιακή εξάρτηση είναι η εξής:

(1) ingr_name → ingr_avgcost, ingr_volume

Το όνομα του προϊόντος δίνει την ενδεικτική τιμή πώλησής του.

(2) prod_name → prod_price

Κάθε προϊόν για να παραχθεί χρειάζεται πολλά διάφορα υλικά. Για να προσδιοριστεί η επιθυμητή ποσότητα ανά υλικό, συνδυάζουμε το όνομα του προϊόντος και το όνομα του υλικού, δηλαδή:

(3) prod_name, ingr_name → prod_ingr_volume

Κάθε παραγγελία έχει τη δική της ημερομηνία παράδοσης.

(4) order_id → order_date

Κάθε παραγγελία μπορεί να συνδυάζει διαφορετικά προϊόντα, σε διαφορετικές ποσότητες το καθένα από αυτά. Οπότε, για να προσδιορίσουμε την ποσότητα ανά προϊόν της παραγγελίας, συνδυάζουμε τον αύξοντα αριθμό της παραγγελίας με το όνομα του προϊόντος, δηλαδή:

(5) ord_id, prod_name → ord_prod_volume

Αφού εισάγουμε τα παραπάνω γνωρίσματα και συναρτησιακές εξαρτήσεις στο πρόγραμμα, πραγματοποιούμε κανονικοποίηση και προκύπτει ότι το υποψήφιο κλειδί του πίνακα είναι το {ingr_name, ord_id, prod_name}. Η διάσπαση των πινάκων ακολουθεί την κάτωθι διαδικασία.

Με την "ingr_name → ingr_avgcost, ingr_volume" ο R διασπάται σε:

R1{♠ingr_name, ingr_avgcost, ingr_volume} (BCNF)

R2{♠ingr_name, ♠ord_id, ♠prod_name, ord_date, ord_prod_volume, prod_ingr_volume, prod_price}

Με την "prod_name → prod_price" ο R2 διασπάται σε:

R3{♠prod_name, prod_price} (BCNF)

R4{♠ingr_name, ♠ord_id, ♠prod_name, ord_date, ord_prod_volume, prod_ingr_volume}

Με την "ingr_name, prod_name → prod_ingr_volume" ο R4 διασπάται σε:

R5{♠ingr_name, ♠prod_name, prod_ingr_volume} (BCNF)

R6{♠ingr_name, ♠ord_id, ♠prod_name, ord_date, ord_prod_volume}

Με την "ord_id, prod_name → ord_prod_volume" ο R6 διασπάται σε:

R7{♠ord_id, ♠prod_name, ord_prod_volume} (BCNF)

R8{♠ingr_name, ♠ord_id, ♠prod_name, ord_date}

Με την "ord_id → ord_date" ο R8 διασπάται σε:

R9{♠ord_id, ord_date} (BCNF)

R10{♠ingr_name, ♠ord_id, ♠prod_name} (BCNF)

Οι πίνακες BCNF είναι οι εξής:

R1{♠ingr_name, ingr_avgcost, ingr_volume}

R3{♠prod_name, prod_price}

R5{♠ingr_name, ♠prod_name, prod_ingr_volume}

R7{♠ord_id, ♠prod_name, ord_prod_volume}

R9{♠ord_id, ord_date}

R10{♠ingr_name, ♠ord_id, ♠prod_name}

Ο πίνακας R1 είναι στην ουσία ένας κατάλογος των διαθέσιμων υλικών μας, όπου με κύριο κλειδί την ονομασία του υλικού μπορούμε να έχουμε πληροφορίες για το μέσο κόστος αγοράς του καθενός και την διαθέσιμη ποσότητά του στην αποθήκη.

Ο R3 είναι ένας τιμοκατάλογος πώλησης των προϊόντων του εργαστηρίου.

Ο R5 είναι ένας πίνακας συνταγών. Αν καταχωρηθούν τα στοιχεία στη βάση δεδομένων, η εφαρμογή με ένα ερώτημα SQL θα μπορεί να επιστρέφει ανά προϊόν τα υλικά και τις ποσότητες ανά υλικό που απαιτούνται για τη δημιουργία του.

Ο R7 είναι πίνακας παραγγελιών. Μπορούμε να δούμε τι προϊόντα και σε τι ποσότητες περιλαμβάνονται σε κάθε παραγγελία. Επίσης, με τα κατάλληλα ερωτήματα SQL μπορούν να προκύψουν πληροφορίες όπως το σύνολο των προϊόντων που πρέπει να παραχθούν (άρα και κατ' επέκταση των υλικών που πρέπει να αγοραστούν εκ των προτέρων) ώστε να ικανοποιηθούν όλες οι παραγγελίες εγκαίρως. Μάλιστα, αν στο ερώτημα αυτό συνδυαστεί και ο πίνακας R9, το εργαστήρι γνωρίζει και τις καταληκτικές ημερομηνίες μέχρι τις οποίες θα πρέπει να έχει ετοιμάσει τα προϊόντα.

4 Ανάπτυξη του προγράμματος

4.1 Εισαγωγή

Το πρόγραμμα το οποίο αναπτύχθηκε ως εκπαιδευτικό εργαλείο για την κανονικοποίηση, δημιουργήθηκε σε γλώσσα προγραμματισμού C#, στο περιβάλλον "Microsoft Visual Studio Express 2015" και βασίζεται στην τεχνολογία WinForms.

Η C# (προφέρεται *σι σαρπ)* είναι η κυριότερη γλώσσα προγραμματισμού της Microsoft για την πλατφόρμα .NET Framework. Κατάγεται απευθείας από τις δύο πιο επιτυχημένες γλώσσες προγραμματισμού, την C και την C++ και σχετίζεται πάρα πολύ με την Java. Η C# δημιουργήθηκε στα τέλη της δεκαετίας του '90 και ο κύριος αρχιτέκτονάς της ήταν ο Anders Hejlsberg. Έκτοτε, η C# αναπτύχθηκε με ταχείς ρυθμούς καθώς η Microsoft προσέθετε διαρκώς νέα χαρακτηριστικά, όπως τη γλώσσα ερωτημάτων για βάσεις δεδομένων LINQ (Language-Integrated Query). Η τελευταία έκδοση της C# είναι η 6.0 και εμφανίστηκε τον Ιούλιο του 2015.

Στη συνέχεια περιγράφονται οι προδιαγραφές και οι σημαντικότερες λειτουργίες του προγράμματος, οι αλγόριθμοι που ακολουθήθηκαν και ορισμένα στιγμιότυπα (screenshots). Κάποια από αυτά ενδέχεται να διαφέρουν με την τελική μορφή του προγράμματος κι αυτό γιατί μετά το πέρας της παρούσας διπλωματικής εργασίας μπορεί να γίνουν αλλαγές στο πρόγραμμα με νεότερες αναβαθμίσεις. Τέλος, παρατίθενται και μερικά επιλεγμένα αποσπάσματα από τον πηγαίο κώδικα με σύντομες επεξηγήσεις.

4.2 Λειτουργίες

Η βασική λειτουργικότητα του προγράμματος έγκειται στη διάσπαση ενός universal πίνακα σε μικρότερους, κανονικούς πίνακες που ικανοποιούν τη συνθήκη BCNF, με τη βοήθεια του αλγόριθμου της κανονικοποίησης. Όλο το περιεχόμενο, με τα γνωρίσματα του universal πίνακα και τις συναρτησιακές εξαρτήσεις, περιλαμβάνονται στην έννοια "σχήμα", το οποίο διαμορφώνεται κατάλληλα, ανάλογα με τις κατά BCNF διασπάσεις των πινάκων που γίνονται κατά τη διάρκεια του προγράμματος.

Οι σημαντικότερες λειτουργίες που επιτελεί το πρόγραμμα είναι οι εξής:

- ✓ Ο χρήστης δημιουργεί και αποθηκεύει ένα αρχικό σχήμα universal πίνακα, στον οποίο εισάγει τα γνωρίσματά του, καθώς και τις συναρτησιακές εξαρτήσεις που σχετίζονται με αυτόν.

- ✓ Επιλέγοντας ένα γνώρισμα (ή συνδυασμό δύο και περισσότερων), ο χρήστης μπορεί να δει τον εγκλεισμό τους. Το πρόγραμμα παρουσιάζει αναλυτικά τη διαδικασία υπολογισμού του εγκλεισμού.
- ✓ Το πρόγραμμα μπορεί να προσδιορίζει τα υποψήφια κλειδιά του πίνακα.
- ✓ Η διάσπαση των πινάκων γίνεται με δύο τρόπους: είτε "αυτοματοποιημένα", όπου παρουσιάζεται αναλυτικά η διαδικασία διάσπασης με τη χρήση των συναρτησιακών εξαρτήσεων, είτε σταδιακά, όπου ο χρήστης επιλέγει βήμα προς βήμα τον πίνακα που θέλει να διασπάσει και την συναρτησιακή εξάρτηση που θα χρησιμοποιήσει.

4.3 Τεχνικές προδιαγραφές

Καταγράφηκαν αναλυτικές τεχνικές προδιαγραφές για την επιθυμητή τελική εικόνα του προγράμματος. Ειδικότερα:

- ✓ Το πρόγραμμα πρέπει να είναι απλό και εύχρηστο.
- ✓ Το περιβάλλον διεπαφής χρήστη (GUI) να μην κουράζει με πολλές λεπτομέρειες.
- ✓ Να έχει την τυπική δομή μιας συνηθισμένης εφαρμογής Windows, με παράθυρα, μενού και κουμπιά εντολών, ενημερωτικά μηνύματα κλπ. Επίσης, η διαδικασία αποθήκευσης και φόρτωσης αρχείου να είναι παρόμοια με αυτή γνωστών Windows εφαρμογών, π.χ. του Office, ώστε ο χρήστης να εξοικειωθεί εύκολα.
- ✓ Να υπάρχουν μηχανισμοί που αποτρέπουν λανθασμένες εισαγωγές από πλευράς χρήστη, οι οποίες μπορεί να οδηγήσουν σε ανεπιθύμητα σφάλματα και κατάρρευση του προγράμματος. Τέτοιοι μηχανισμοί για παράδειγμα είναι να μην επιτρέπεται ο χρήστης να εισάγει γνώρισμα με ονομασία που ήδη χρησιμοποιείται για άλλο γνώρισμα, το κουμπί καταχώρησης "OK" να ενεργοποιείται μόνο αφού έχουν πληκτρολογηθεί σωστά τα απαραίτητα στοιχεία στη φόρμα κλπ.
- ✓ Όπου εμφανίζονται αποτελέσματα μιας διαδικασίας, π.χ. κατά τον υπολογισμό του εγκλεισμού ενός γνωρίσματος ή την αναζήτηση των κλειδιών, κρίθηκε σκόπιμο στα πλαίσια της εκπαιδευτικής φύσης του προγράμματος, να επεξηγούνται τα στάδιά της.

- ✓ Να γίνει προσπάθεια βελτιστοποίησης των αλγόριθμων του προγράμματος έτσι ώστε να είναι μειωμένος ο χρόνος απόκρισής του, ιδίως σε περιπτώσεις χρονοβόρων διαδικασιών, π.χ. της κανονικοποίησης ενός μεγάλου σχήματος με πολλά γνωρίσματα.
- ✓ Να διατηρεί τον εκπαιδευτικό του προσανατολισμό, βοηθώντας τον χρήστη με κατανοητά μηνύματα. Στα πλαίσια αυτά, αποφασίστηκε να δημιουργηθούν απλές οδηγίες χρήσης (manual) του προγράμματος, οι οποίες και συμπεριλαμβάνονται στο παράρτημα της εργασίας.

4.4 Έλεγχος

Αφού ολοκληρώθηκε η ανάπτυξη του προγράμματος, έγινε εξαντλητικός έλεγχος σε όλες τις λειτουργίες του για ενδεχόμενα σφάλματα (bugs), έτσι ώστε να περιοριστεί σημαντικά η πιθανότητα ανεπιθύμητου τερματισμού του. Η διαδικασία των διασπάσεων ελέγχθηκε με παράλληλη επαλήθευση πολλών ασκήσεων κανονικοποίησης "στο χέρι". Επίσης, δοκιμάστηκε η γενικότερη λειτουργικότητά του από τρίτους χρήστες, υπό μορφή beta testing, των οποίων το feedback αξιοποιήθηκε για διόρθωση του κώδικα και βελτίωση του προγράμματος. Τέλος, το πρόγραμμα εκτελέστηκε σε διαφορετικά λειτουργικά συστήματα Windows ώστε να εξακριβωθεί η συμβατότητά του με αυτά.

4.5 Εγκατάσταση του προγράμματος

Το πρόγραμμα είναι standalone application και "τρέχει" σε Windows XP (με Service Pack 3), Windows Vista, Windows 7, Windows 8 και Windows 10. Εγκαθίσταται με την τεχνολογία ClickOnce της Microsoft, η οποία δίνει τη δυνατότητα να "κατεβαίνουν" αυτομάτως στον υπολογιστή του χρήστη οι μελλοντικές αναβαθμίσεις (upgrades) του. Η εγκατάσταση του προγράμματος μπορεί να γίνει από τον ηλεκτρονικό σύνδεσμο http://www.myprograms.eu/apps/normalization/gr/normalizationsetup.htm.

Στη σελίδα αυτή, ο χρήστης πατάει Install και εν συνεχεία "Εκτέλεση" (ή "Run"). Επιπλέον, έχει δημιουργηθεί σελίδα αφιερωμένη στο πρόγραμμα, στη διεύθυνση http://www.myprograms.eu/normalization με οδηγίες για το πρόγραμμα και την εγκατάστασή του.

4.6 Εκκίνηση προγράμματος

Αφού ο χρήστης εκκινήσει το πρόγραμμα με διπλό κλικ στο εκτελέσιμο αρχείο ή στη συντόμευσή του, καλείται η αρχική μέθοδος Main() που στη γλώσσα C# αποτελεί το σημείο εισόδου κάθε προγράμματος. Η μέθοδος Main() ξεκινά το πρόγραμμα καλώντας την κλάση - φόρμα FRMMAIN (Απόσπασμα Κώδικα 1).

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new FRMMAIN());
}
```

Απόσπασμα Κώδικα 1: Η μέθοδος Main()

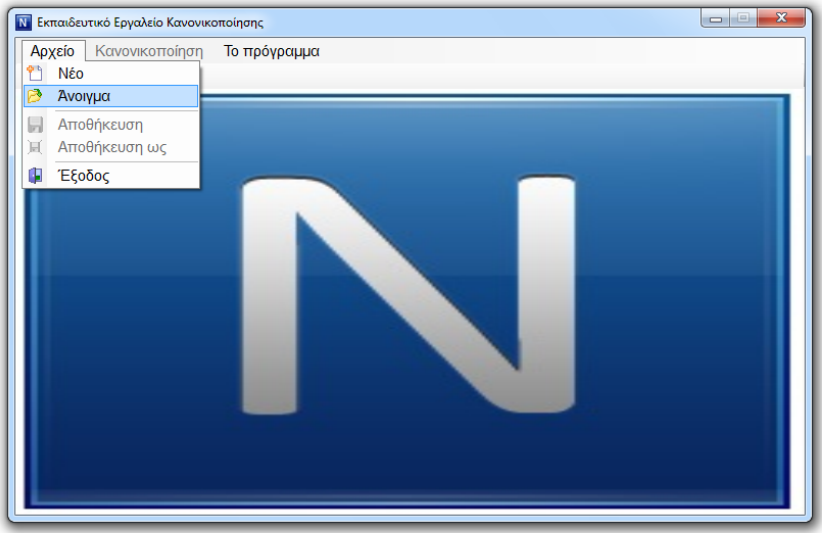
Η FRMMAIN είναι η κεντρική φόρμα του προγράμματος η οποία εκκινεί τις περισσότερες διαδικασίες. Το κλείσιμό της συνεπάγεται ουσιαστικά και τον τερματισμό του προγράμματος. Σε αυτήν περιλαμβάνονται μεταξύ άλλων, οι πίνακες των γνωρισμάτων και των συναρτησιακών εξαρτήσεων και διάφορα μενού με εντολές, όπως άνοιγμα σχήματος, δημιουργία γνωρίσματος, προσδιορισμός κλειδιών κλπ.

4.7 Νέο σχήμα - άνοιγμα σχήματος

Αφού ξεκινήσει το πρόγραμμα εμφανίζεται η κενή, αρχική οθόνη (Εικόνα 3). Εδώ ο χρήστης πρέπει είτε να δημιουργήσει ένα νέο σχήμα με την εντολή "Αρχείο > Νέο", ή να φορτώσει ένα προϋπάρχον αρχείο με την εντολή "Αρχείο > Άνοιγμα". Τα αρχεία, όπου αποθηκεύονται οι πληροφορίες του σχήματος, είναι αρχεία κειμένου με κατάληξη .nor και περιλαμβάνουν τα γνωρίσματα του σχήματος και τις συναρτησιακές εξαρτήσεις. Τα αποτελέσματα των επόμενων βημάτων της κανονικοποίησης, με την αυτοματοποιημένη εύρεση των κλειδιών και τη διάσπαση των πινάκων, δεν αποθηκεύονται στο αρχείο. Έτσι ο χρήστης θα εκτελεί κάθε φορά ο ίδιος "χειροκίνητα" τη διαδικασία της κανονικοποίησης (επιλέγοντας τις κατάλληλες εντολές) ώστε να βλέπει, στα πλαίσια της εκπαιδευτικής φύσης του προγράμματος, την εξέλιξή της.

Από την κεντρική φόρμα του προγράμματος ο χρήστης μπορεί, εκτός από τη δημιουργία - αποθήκευση σχήματος, να εισάγει, να επεξεργαστεί και να διαγράψει γνωρίσματα και συναρτησιακές εξαρτήσεις, να υπολογίσει τον εγκλεισμό επιλεγμένων γνωρισμάτων, να προσδιορίσει το κλειδί του σχήματος, να κανονικοποιήσει το σχήμα σε BCNF μορφή, είτε αυτοματοποιημένα, είτε σε στάδια. Επίσης, μπορεί από το μενού

εντολών "Το πρόγραμμα" να προσαρμόσει ορισμένες από τις ρυθμίσεις του προγράμματος και να δει τις σύντομες οδηγίες χρήσης του.



Εικόνα 3: Η κεντρική οθόνη του προγράμματος

Η δημιουργία και το άνοιγμα του σχήματος υλοποιούνται μέσα από την δημόσια μέθοδο scLoad() η οποία δέχεται ως παραμέτρους τη διαδρομή όπου βρίσκεται αποθηκευμένο το αρχείο και τη bool μεταβλητή isNew, η οποία αν είναι true, σημαίνει ότι ο χρήστης επέλεξε να δημιουργήσει ένα νέο σχήμα. Στο Απόσπασμα Κώδικα 2 παρατίθεται ένα μέρος της μεθόδου αυτής.

```
public void scLoad(string fullpath, bool isNew)
{
    //αν η τιμή isNew είναι true, τότε δημιουργείται ένα νέο, κενό σχήμα,
αλλιώς φορτώνεται ένα αποθηκευμένο
    if (isNew)
    {
        scFilename = fullpath;
        path = "";
        scDescription = "";
        resetList();
    }
    else
    {
        //από τη διαδρομή εξάγεται το όνομα του αρχείου και αποθηκεύεται στη
μεταβλητή filename
        scFilename = Path.GetFileNameWithoutExtension(fullpath);
        path = fullpath;

        //φορτώνεται το αρχείο path
        StreamReader FileR = new StreamReader(path,
System.Text.Encoding.UTF8);

        //διαβάζεται η περιγραφή του σχήματος
        scDescription = FileR.ReadLine();
```

```
        //διαβάζεται το πλήθος των γνωρισμάτων
        int iSuma;
        iSuma = Int32.Parse(FileR.ReadLine());

        //διαβάζονται ένα προς ένα τα γνωρίσματα (όνομα και τύπος)
        //για κάθε ένα από αυτά, δημιουργείται ένα αντικείμενο τύπου Attr
        resetList();
        string sName, sType;
        for (int i = 0; i < iSuma; i++)
        {
            sName = FileR.ReadLine();
            sType = FileR.ReadLine();
            AttrCreate(sName, sType);
        }

        //διαβάζεται το πλήθος των συναρτησιακών εξαρτήσεων και για κάθε ένα
από αυτά δημιουργείται ένα αντικείμενο FD
        iSuma = Int32.Parse(FileR.ReadLine());

        for (int x = 0; x < iSuma; x++)
        {
            FD fd = new FD();

            //διαβάζεται η ένδειξη αν η συναρτησιακή εξάρτηση είναι τετριμμένη
            fd.IsTrivial = bool.Parse(FileR.ReadLine());

            //προσδιορίζεται το πλήθος των γνωρισμάτων που υπάρχουν στα δύο
σκέλη της συναρτησιακής εξάρτησης
            int left;
            int right;
            left = Int32.Parse(FileR.ReadLine());
            right = Int32.Parse(FileR.ReadLine());

            //προσδιορίζεται το αντικείμενο attr που αντιστοιχεί στη μεταβλητή
str και προστίθεται στο αριστερό ή το δεξί σκέλος της συναρτησιακής εξάρτησης
            Attr attrread;
            string str;
            int y;
            for (y = 0; y < left; y++)
            {
                str = FileR.ReadLine();
                attrread = AttrGetObject(str);
                fd.AddLeft(attrread);
            }
            for (y = 0; y < right; y++)
            {
                str = FileR.ReadLine();
                attrread = AttrGetObject(str);
                fd.AddRight(attrread);
            }
            FDCreate(fd);
        }

        FileR.Close();
    }
}
```

Απόσπασμα Κώδικα 2: Η μέθοδος scLoad()

4.8 Δημιουργία - επεξεργασία γνωρίσματος

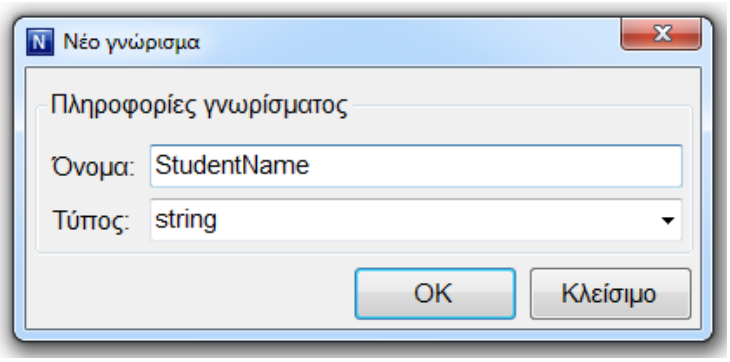
Τη δημιουργία, όπως και την επεξεργασία των γνωρισμάτων, τη χειρίζεται η κλάση - φόρμα frmAttr (Εικόνα 4), η οποία δημιουργεί αντικείμενα τύπου Attr.

Κάθε αντικείμενο της κλάσης Attr περιέχει ένα και μόνο γνώρισμα, με πεδία όπως το όνομα του γνωρίσματος και ο τύπος του. Η frmAttr αποτρέπει την εισαγωγή στο πρόγραμμα γνωρισμάτων των οποίων η ονομασία ήδη χρησιμοποιείται.

Μια χρήσιμη δυνατότητα που παρέχει το πρόγραμμα είναι η εισαγωγή ακολουθίας γνωρισμάτων με διαδοχικούς αλφαβητικούς χαρακτήρες. Αυτό επιτυγχάνεται με την πληκτρολόγηση του συμβόλου ">" ανάμεσα σε δύο γράμματα του αλφαβήτου, με την προϋπόθεση ότι αυτά είναι κεφαλαία και στην ίδια γλώσσα. Έτσι, για παράδειγμα, αν ο χρήστης πληκτρολογήσει "A > D", το πρόγραμμα θα δημιουργήσει 4 αντικείμενα - γνωρίσματα με τις ονομασίες A, B, C και D.

Αφού δημιουργηθεί ένα νέο γνώρισμα, αυτό εμφανίζεται στον πίνακα γνωρισμάτων της κεντρικής φόρμας και πλέον μπορεί να χρησιμοποιηθεί για δημιουργία συναρτησιακών εξαρτήσεων. Ο χρήστης μπορεί να επεξεργαστεί ένα γνώρισμα, για να αλλάξει το όνομά του ή τον τύπο του, κάνοντας διπλό κλικ στο αντίστοιχο κελί ή και να το διαγράψει. Δεν επιτρέπεται ωστόσο διαγραφή γνωρίσματος που συμμετέχει σε συναρτησιακή εξάρτηση.

Στα Απόσπασμα Κώδικα 3 και Απόσπασμα Κώδικα 4 παρατίθεται ένα μέρος του κώδικα που αφορά τη δημιουργία ενός νέου γνωρίσματος.



Εικόνα 4: Δημιουργία νέου γνωρίσματος

```
private void Create()
{
    string sName = txtName.Text.Trim();
    string sType = comType.Text.Trim();
    int x = 0; //το πλήθος των νέων γνωρισμάτων που καταχωρήθηκαν

    //αρχικά ελέγχεται η περίπτωση ο χρήστης να έχει ζητήσει τη δημιουργία
πολλαπλών γνωρισμάτων
```

```
    //αυτό γίνεται εισάγοντας το σύμβολο ">" ανάμεσα σε δύο χαρακτήρες, π.χ.
"A>D"
    if (sName.Contains(">"))
    {
        //ελέγχεται πρώτα αν η χρήση του ειδικού συμβόλου ">" έγινε σωστά
        //να είναι δηλαδή στη μέση το ">" και τα γράμματα να είναι κεφαλαία
και να ανήκουν στην ίδια γλώσσα (ελληνικά ή αγγλικά)
        string s = sName;
        if (s.Length == 3 && s[1].ToString() == ">" && CharsCheck(s[0], s[2]))
        {
            char c1 = sName[0];
            char c2 = sName[2];

            for (char cc = c1; cc <= c2; cc++)
                if (Char.IsLetter(cc))
                    //στέλνονται στην FRMMAIN τα στοιχεία του γνωρίσματος ώστε
να δημιουργηθεί αντικείμενο με το όνομα cc
                    if (frmmain.AttrCreate(cc.ToString(), sType)) x++;
        }
    }

    //αν έγινε εισαγωγή μόνο ενός γνωρίσματος, στέλνονται στην FRMMAIN τα
στοιχεία του γνωρίσματος ώστε να δημιουργηθεί το αντικείμενο με το όνομα sName
    else if (frmmain.AttrCreate(sName, sType)) x++;
    else
    {
        //αν επιστραφεί η ένδειξη false σημαίνει ότι το όνομα του γνωρίσματος
χρησιμοποιείται ήδη για άλλο γνώρισμα
        MessageBox.Show("Το όνομα \" " + sName + "\" χρησιμοποιείται ήδη για
άλλο γνώρισμα.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    //ανάλογα με το πλήθος x των γνωρισμάτων που καταχωρήθηκαν, εμφανίζονται
τα σχετικά μηνύματα
    if (x == 0)
    {
        MessageBox.Show("Δεν καταχωρήθηκε κανένα νέο γνώρισμα. Είτε τα ονόματα
χρησιμοποιούνται ήδη για άλλα γνωρίσματα, είτε δεν χρησιμοποιήσατε σωστά την
αυτόματη καταχώρηση.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    if (x == 1)
        MessageBox.Show("Το νέο γνώρισμα καταχωρήθηκε.", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Information);
    else
        MessageBox.Show("Καταχωρήθηκαν " + x.ToString() + " νέα γνωρίσματα.",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);
    txtName.Text = "";
    comType.Text = "";
}
```

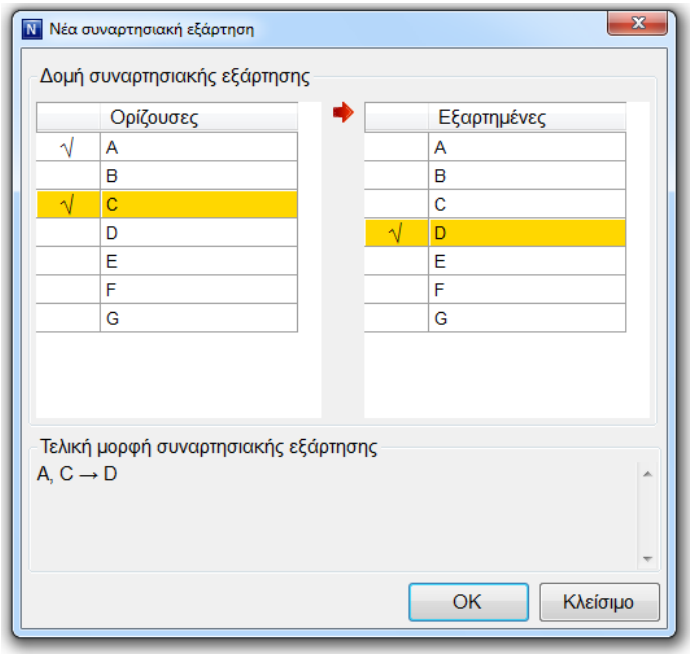
Απόσπασμα Κώδικα 3: Η μέθοδος Create()

```
public bool AttrCreate(string name, string type)
{
    //ελέγχεται αν το όνομα του νέου γνωρίσματος χρησιμοποιείται ήδη, κι αν
ναι, επιστρέφεται η ένδειξη false
    if (AttrExists(name, null)) return false;

    //δημιουργείται αντικείμενο τύπου Attr και προστίθεται στην attrList
    Attr attr = new Attr(name, type);
    attrList.Add(attr);

    //τα στοιχεία του νέου γνωρίσματος προστίθενται στο GridAttr
    DataGridView g = GridAttr;
    g.Rows.Add();
    g.Rows[g.RowCount - 1].Cells["colName"].Value = name; //όνομα γνωρίσματος
    g.Rows[g.RowCount - 1].Cells["colType"].Value = type; //τύπος γνωρίσματος

    //επιστρέφεται η ένδειξη true
    return true;
}
```

Απόσπασμα Κώδικα 4: Η μέθοδος AttrCreate()

4.9 Δημιουργία – επεξεργασία συναρτησιακής εξάρτησης

Η δημιουργία, όπως και η επεξεργασία, των συναρτησιακών εξαρτήσεων χειρίζεται από την κλάση - φόρμα frmFD (Εικόνα 5), η οποία δημιουργεί αντικείμενα τύπου FD (Απόσπασμα Κώδικα 5). Αφού δημιουργηθεί μια συναρτησιακή εξάρτηση, αυτή εμφανίζεται στον σχετικό πίνακα της κεντρικής φόρμας. Με διπλό κλικ στο αντίστοιχο κελί, ο χρήστης έχει τη δυνατότητα να επεξεργαστεί μια συναρτησιακή εξάρτηση, αλλάζοντας τα στοιχεία των γνωρισμάτων που την προσδιορίζουν. Επίσης, με το πλήκτρο Delete ή το κουμπί της διαγραφής, ο χρήστης μπορεί να αφαιρέσει τη συναρτησιακή εξάρτηση από το σχήμα.



Εικόνα 5: Δημιουργία συναρτησιακής εξάρτησης

Κάθε αντικείμενο της κλάσης FD αποτελεί μια συναρτησιακή εξάρτηση, η οποία περιλαμβάνει γνωρίσματα, δηλαδή αντικείμενα τύπου Attr. Για τον προσδιορισμό των δύο σκελών της συναρτησιακής εξάρτησης, το αντικείμενο FD εμπεριέχει δύο λίστες γνωρισμάτων, την Left και την Right. Επιπλέον, η κλάση FD περιλαμβάνει διάφορες δημόσιες μεθόδους πρόσβασης ή μετάλλαξης, όπως την AddLeft() και την AddRight() που προσθέτουν γνωρίσματα ή λίστα γνωρισμάτων στο αριστερό και το δεξί σκέλος αντίστοιχα, την GetAll() που επιστρέφει λίστα με όλα τα γνωρίσματα που εμπεριέχονται στη συναρτησιακή εξάρτηση, καθώς και μια override μέθοδο ToString() η οποία επιστρέφει την τελική απεικόνιση της συναρτησιακής εξάρτησης. Στο Απόσπασμα Κώδικα 6 παρατίθεται ο σχετικός κώδικας της μεθόδου ToString().

```
public bool FDCreate(FD fd)
{
    //ελέγχεται αν η νέα συναρτησιακή εξάρτηση είναι παρόμοια με μια άλλη, κι
αν ναι, επιστρέφεται η ένδειξη false
    if (FDExists(fd, -1)) return false;

    //το νέο αντικείμενο προστίθεται στην FDList
    FDList.Add(fd);

    //η νέα συναρτησιακή εξάρτηση προστίθεται στο GridFD
    DataGridView g = GridFD;
    g.Rows.Add();
    g.Rows[g.RowCount - 1].Cells["colFD"].Value = fd.ToString(); //περιγραφή
συναρτησιακής εξάρτησης
    g.Rows[g.RowCount - 1].Cells["colID"].Value = fd.GetAA(); //κωδικός
αριθμός της συναρτησιακής εξάρτησης
```

```
    if (fd.IsTrivial) g.Rows[g.RowCount - 1].Cells["colTet"].Value = "\u221A";
//tick για τετριμμένη συναρτησιακή εξάρτηση

    //επιστρέφεται η ένδειξη true
    return true;
}
```

Απόσπασμα Κώδικα 5: Η μέθοδος FDCreate()

```
public override string ToString()
{
    //δημιουργούνται οι πίνακες LeftT και RightT
    List<string> LeftT = new List<string>();
    List<string> RightT = new List<string>();

    //σε αυτούς τους πίνακες προστίθενται οι ονομασίες των γνωρισμάτων των
πινάκων Left και Right
    foreach (Attr attr in Left)
        LeftT.Add(attr.Name);
    foreach (Attr attr in Right)
        RightT.Add(attr.Name);

    //οι πίνακες LeftT και RightT ταξινομούνται αλφαβητικά
    LeftT.Sort();
    RightT.Sort();

    //διαβάζονται τα περιεχόμενα των πινάκων LeftT και RightT και
δημιουργείται η τελική μορφή της συναρτησιακής εξάρτησης η οποία και
επιστρέφεται ως string
    string str = "";
    string left = "";
    string right = "";

    //πρώτα του πίνακα LeftT
    for (int i = 0; i < LeftT.Count; i++)
    {
        if (left != "") left = left + ", ";
        left = left + LeftT[i];
    }

    //μετά του πίνακα RightT
    for (int i = 0; i < RightT.Count; i++)
    {
        if (right != "") right = right + ", ";
        right = right + RightT[i];
    }

    //σχηματίζεται η τελική μορφή της συναρτησιακής εξάρτησης
    //το "\u2192" χρησιμοποιείται για την απεικόνιση του δεξιού βέλους
    str = left + " " + "\u2192" + " " + right;

    return str;
}
```

Απόσπασμα Κώδικα 6: Η μέθοδος ToString() της κλάσης FD

4.10 Προσδιορισμός συναρτησιακής εξάρτησης ως τετριμμένης

Για τον προσδιορισμό μιας συναρτησιακής εξάρτησης ως τετριμμένης ή όχι, εξετάζεται αν τα γνωρίσματα του δεξιού σκέλους είναι υποσύνολα του αριστερού. Αυτό γίνεται με τη βοήθεια ενός βρόχου ο οποίος επαναλαμβάνεται τόσες φορές, όσες και το πλήθος των γνωρισμάτων του δεξιού σκέλους. Αν μια συναρτησιακή εξάρτηση προκύπτει ότι είναι τετριμμένη, τότε εμφανίζεται η ένδειξη "√" (tick) στο σχετικό κελί του πίνακα (Εικόνα 6) των συναρτησιακών εξαρτήσεων.

Συναρτησιακές εξαρτήσεις

Περιγραφή	Τετ
A, B → A	√
A → A	√
A, C → C	√

Εικόνα 6: Ένα σύνολο τετριμμένων συναρτησιακών εξαρτήσεων

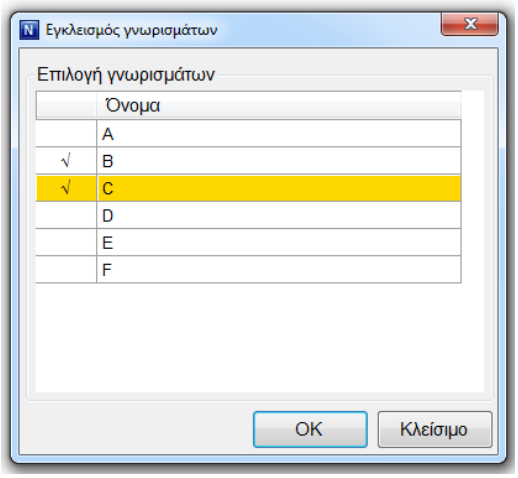
Σε κάθε μια εκτέλεση του βρόχου, εξετάζεται αν το κάθε ένα γνώρισμα ξεχωριστά του δεξιού σκέλους περιλαμβάνεται και στο αριστερό σκέλος. Αν αυτό δεν ισχύει έστω και για ένα γνώρισμα του δεξιού σκέλους, τότε η συναρτησιακή εξάρτηση θεωρείται μη τετριμμένη. Σε διαφορετική περίπτωση, η συναρτησιακή εξάρτηση προσδιορίζεται ως τετριμμένη και ενημερώνεται η σχετική μεταβλητή ως true. Ο σχετικός βρόχος παρατίθεται στο Απόσπασμα Κώδικα 7.

```
IsTrivial = true;
foreach (Attr attr in Right)
{
    if (!Left.Contains(attr))
    {
        IsTrivial = false;
        return;
    }
}
```

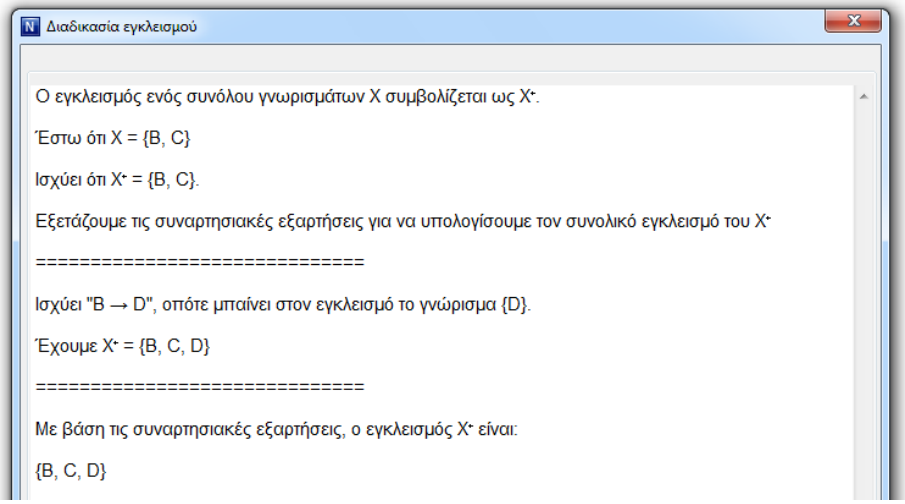
Απόσπασμα Κώδικα 7: Προσδιορισμός τετριμμένης συναρτησιακής εξάρτησης

4.11 Εγκλεισμός γνωρισμάτων

Με την εντολή "Κανονικοποίηση > Εγκλεισμός" (Εικόνα 7) το πρόγραμμα υπολογίζει και εμφανίζει βήμα προς βήμα (Εικόνα 8) τον εγκλεισμό ενός γνωρίσματος ή συνδυασμού δύο και περισσότερων γνωρισμάτων.



Εικόνα 7: Επιλογή γνωρισμάτων για υπολογισμό εγκλεισμού



Εικόνα 8: Προβολή διαδικασίας υπολογισμού εγκλεισμού

Ο υπολογισμός του εγκλεισμού πραγματοποιείται με τη μέθοδο attrClosure() της κλάσης - φόρμας frmClosure. Η μέθοδος αυτή δέχεται ως παράμετρο συνδυασμό ενός ή περισσοτέρων γνωρισμάτων, των οποίων θέλουμε να υπολογίσουμε τον εγκλεισμό.

Η μέθοδος ξεκινά με τη δήλωση πίνακα, ονόματι closure, ο οποίος θα περιέχει τον εγκλεισμό του ορίσματος. Ως πρώτες τιμές του πίνακα ορίζονται αυτομάτως τα ίδια τα γνωρίσματα του ορίσματος, αφού εξ ορισμού προσδιορίζουν τον εαυτό τους.

Εν συνεχεία, εκτελείται βρόχος (Απόσπασμα Κώδικα 8) ο οποίος εξετάζει, μία προς μία, τις συναρτησιακές εξαρτήσεις. Αν το πλήθος της τομής του αριστερού σκέλους της συναρτησιακής εξάρτησης και του ως τώρα περιεχομένου του πίνακα closure είναι μικρότερο ή ίσο του πλήθους των γνωρισμάτων του αριστερού σκέλους, τότε

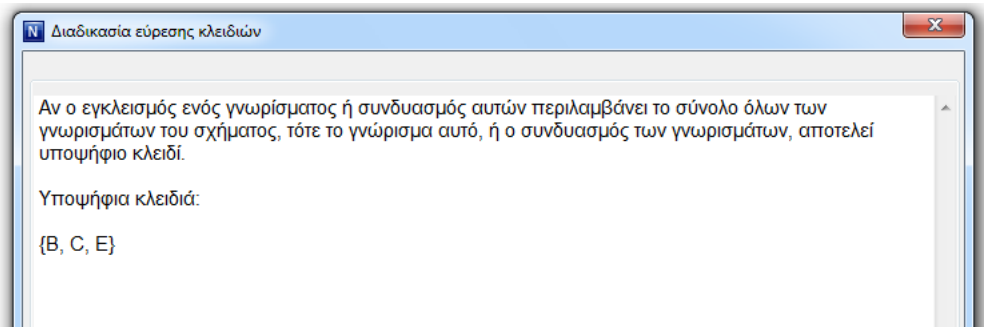
προστίθενται στον εγκλεισμό όλα τα γνωρίσματα του δεξιού σκέλους (με την προϋπόθεση να μην είχαν προστεθεί ήδη νωρίτερα). Ο βρόχος τότε ξεκινά από την αρχή, έτσι ώστε να ελεγχθούν ξανά όλες οι συναρτησιακές εξαρτήσεις. Αφού ολοκληρωθεί πλήρως η σάρωσή τους, το περιεχόμενο του πίνακα closure επιστρέφεται ως εγκλεισμός. Αν μάλιστα το πλήθος των γνωρισμάτων του εγκλεισμού ισούται με το πλήθος όλων των γνωρισμάτων του σχήματος, τότε ο συνδυασμός των γνωρισμάτων που εξετάσαμε στην αρχή αποτελεί υποψήφιο κλειδί ή υπερκλειδί αν υπερκαλύπτει το υποψήφιο κλειδί.

```
public List<Attr> attrClosure(List<Attr> attrS, bool showOut)
{
    //ο πίνακας closure περιλαμβάνει τον εγκλεισμό των γνωρισμάτων attrS
    List<Attr> closure = new List<Attr>();
    closure.AddRange(attrS);

//η RepeatLoop χρησιμοποιείται ως σημείο επανεκκίνησης του βρόχου σε περίπτωση
που πρέπει να ελεγχθούν από την αρχή οι συναρτησιακές εξαρτήσεις
RepeatLoop:

    //ελέγχω μία προς μία τις συναρτησιακές εξαρτήσεις
    foreach (FD fd in FDList)
    {
        //ελέγχεται με την τομή αν τα γνωρίσματα του αριστερού σκέλους της
συναρτησιακής εξάρτησης περιλαμβάνονται στον ως τώρα εγκλεισμό
        if (fd.GetLeft().Intersect(closure).Count() >= fd.GetLeft().Count)
        {
            //αν ναι, τότε προστίθενται τα γνωρίσματα του δεξιού σκέλους στην
συναρτησιακή εξάρτηση, με την προϋπόθεση να μην έχουν ήδη προστεθεί, κι αν
αυτό γίνει τότε η διαδικασία αρχίζει ξανά από την RepeatLoop
            List<Attr> toadd = new List<Attr>();
            foreach (Attr attrR in fd.GetRight())
            {
                if (!closure.Contains(attrR))
                {
                    anyFDUsed = true;
                    toadd.Add(attrR);
                }
            }
            if (toadd.Count > 0)
            {
                //closure.Add(attrR);
                closure.AddRange(toadd);
                goto RepeatLoop;
            }
        }
    }
    return closure;
}
```

Απόσπασμα Κώδικα 8: Η μέθοδος attrClosure()

4.12 Εύρεση κλειδιών

Με την εντολή "Κανονικοποίηση > Κλειδιά" το πρόγραμμα εμφανίζει τα υποψήφια κλειδιά του σχήματος (Εικόνα 9).



Εικόνα 9: Προβολή αποτελεσμάτων εύρεσης κλειδιών

Προκειμένου να εξεταστούν όλοι οι πιθανοί συνδυασμοί (από 1 ως *ν*-1, όπου *ν* το πλήθος των γνωρισμάτων), χρησιμοποιείται η μέθοδος AttrBinarySelection(), όπου εντοπίζονται όλοι οι δυνατοί συνδυασμοί των γνωρισμάτων του σχήματος, ανά *κ* γνωρίσματα (Απόσπασμα Κώδικα 9). Οι συνδυασμοί αυτοί ελέγχονται αν μπορούν να αποτελέσουν υποψήφια κλειδιά με τη μέθοδο attrClosure(), για τον υπολογισμό του εγκλεισμού κι αν ναι, δημιουργείται ένα υποψήφιο κλειδί το οποίο προστίθεται στη λίστα με το όνομα keyList.

```
private void AttrBinarySelection(List<FD> FDList, List<Attr> newAttrList, ref
List<Key> keyList, int k, frmOut frmout, bool showOut)
{
    string sBin = "";
    int i, x;

    //δημιουργείται μια καθαρή λίστα γνωρισμάτων, χωρίς αυτά που είναι
excluded
    List<Attr> cleanList = new List<Attr>();
    foreach (Attr attr in newAttrList)
        if (!attr.Exclude) cleanList.Add(attr);

    //αν όλα τα γνωρίσματα είναι excluded ή το πλήθος των συνδυασμών k είναι
μεγαλύτερο του μεγέθους της λίστας, επιστρέφεται κενή λίστα
    if (cleanList.Count == 0 | k > cleanList.Count) return;

    //δημιουργείται μια καινούρια fdlist η οποία θα φιλοξενεί μόνο τις
συναρτησιακές εξαρτήσεις που αφορούν τα γνωρίσματα της newAttrList
    //πριν γίνει αυτό, διασπώνται οι συναρτησιακές εξαρτήσεις που έχουν
πολλαπλά γνωρίσματα στο δεξί σκέλος τους
    List<FD> tempofdlist = new List<FD>();
    foreach (FD fd in FDList)
    {
        foreach (Attr attr in fd.GetRight())
        {
            FD newfd = new FD();
            newfd.AddLeft(fd.GetLeft());
```

```
            newfd.AddRight(attr);
            if (newfd.GetAll().Intersect(newAttrList).Count() ==
newfd.GetAll().Count) tempofdlist.Add(newfd);
        }
    }

    //προσδιορίζεται το πλήθος των δυνητικών συνδυασμών
    int maxComb;
    maxComb = Factorial((ulong)cleanList.Count) / (Factorial((ulong)k) *
Factorial((ulong)(cleanList.Count - k)));

    //η αναζήτηση των συνδυασμών ξεκινά από τον πρώτο δυαδικό αριθμό που δίνει
πλήρη συνδυασμό k γνωρισμάτων, κι αυτός δίνεται από τον τύπο (2^k)-1
    i = (int)Math.Pow(2, k) - 1;
    x = 0;
    while (x < maxComb)
    {
        //ελέγχεται αν ο δυαδικός αριθμός έχει k ψηφία 1
        if (bin[i] == k)
        {
            x++;
            //δημιουργείται αντικείμενο τύπου Key
            Key key = new Key();

            //ο αριθμός i μετατρέπεται σε δυαδικό
            sBin = Convert.ToString(i, 2);

            //το sBin αντιστρέφεται
            char[] ch = sBin.ToCharArray();
            Array.Reverse(ch);
            sBin = new string(ch);

            //ελέγχεται η θέση των ψηφίων 1 στον δυαδικό αριθμό και
προστίθενται στο κλειδί key τα αντίστοιχα γνωρίσματα
            for (int j = 0; j < sBin.Length; j++)
                if (sBin[j] == '1')
                {
                    key.AddToKey(cleanList[j]);
                }
                //ελέγχεται αν ο εγκλεισμός του νέου κλειδιού περιλαμβάνει όλα
τα γνωρίσματα του σχήματος, κι αν ναι, τότε προστίθεται στη λίστα των
υποψήφιων κλειδιών
            if (key != null)
            {
                frmClosure frm = new frmClosure(key.GetAttrs(), tempofdlist,
this);
                if (frm.attrClosure(key.GetAttrs(),
false).Intersect(newAttrList).Count() == newAttrList.Count) keyList.Add(key);
            }
        }
        i++;
    }
}
```

Απόσπασμα Κώδικα 9: Η μέθοδος AttrBinarySelection()

Η λίστα keyList επιστρέφεται στη μέθοδο KeysGet() (Απόσπασμα Κώδικα 10), η οποία ουσιαστικά την είχε καλέσει πιο πριν, έχοντας πλέον όλους τους δυνατούς συνδυασμούς των γνωρισμάτων του σχήματος, των οποίων υπολογίζεται ο εγκλεισμός και αν αυτός περιλαμβάνει το σύνολο των γνωρισμάτων του σχήματος, τότε ο συνδυασμός καταχωρείται στην keyList ως υποψήφιο κλειδί.

```
public List<Key> KeysGet(List<FD> FDList, List<Attr> newAttrList, bool
showOut)
{
    //δημιουργείται πίνακας με τα υποψήφια κλειδιά του σχήματος
    List<Key> keyList = new List<Key>();

    //ελέγχονται ένα προς ένα όλα τα γνωρίσματα για το κατά πόσον μπορεί να
συμμετέχουν σε κλειδί
    //αν ένα γνώρισμα δεν βρίσκεται σε κανένα αριστερό σκέλος κάποιας FD τότε
αποκλείεται από τον έλεγχο
    foreach (Attr attr in newAttrList)
    {
        attr.Exclude = true;
        foreach (FD fd in FDList)
            if (fd.GetLeft().Contains(attr))
                attr.Exclude = false;

        //για να οριστικοποιηθεί ο αποκλεισμός του γνωρίσματος, πρέπει να μην
συμμετέχει όμως και σε κανένα δεξί
        //κι αυτό γιατί τότε απλά προσδιορίζει τον εαυτό του, οπότε μπορεί να
βρεθεί σε σύνθετο κλειδί
        //έτσι δημιουργείται μια τοπική λίστα με τα δεξιά γνωρίσματα όλων των
FD
        if (attr.Exclude)
        {
            List<Attr> attrall = new List<Attr>();
            foreach (FD fd in FDList)
                attrall.AddRange(fd.GetRight());

            //αν το γνώρισμα attr υπάρχει στην τοπική λίστα με τα γνωρίσματα
των δεξιών σκελών, ο αποκλεισμός του γνωρίσματος αίρεται
            if (attrall.Contains(attr)) attr.Exclude = true;
            else attr.Exclude = false;
            attrall = null;
        }
    }

    //ελέγχεται ο εγκλεισμός όλων των συνδυασμών των γνωρισμάτων του σχήματος
κι αν αυτός περιλαμβάνει όλα τα γνωρίσματα, τότε έχουμε ένα υποψήφιο κλειδί
    //το πλήθος των γνωρισμάτων είναι n και τα εξετάζουμε σε συνδυασμούς ανά
k, με τη βοήθεια της μεθόδου AttrBinarySelection
    int n, k;
    n = newAttrList.Count;
    k = 1;
    for (k = 1; k < n; k++)
        AttrBinarySelection(FDList, newAttrList, ref keyList, k, frmout,
showOut);
```

```
    //αφού ολοκληρώθηκε η διαδικασία, ελέγχεται αν βρέθηκε έστω και ένα
κλειδί, κι αν όχι, επιστρέφεται ως κλειδί του σχήματος ένα νέο κλειδί με όλα
τα γνωρίσματα, αλλιώς μόνο τα κλειδιά που καταχωρήθηκαν στην keyList
    if (keyList.Count == 0)
    {
        Key key = new Key();
        for (int i = 0; i < newAttrList.Count; i++)
            key.AddToKey(newAttrList[i]);
        keyList.Add(key);
    }
    return keyList;
}
```

Απόσπασμα Κώδικα 10: Η μέθοδος KeysGets()

4.13 Η κλάση Key

Κάθε φορά που εντοπίζεται ένα υποψήφιο κλειδί, δημιουργείται ένα αντίστοιχο αντικείμενο τύπου Key. Η κλάση Key εμπεριέχει τη λίστα keyAttrs, η οποία δέχεται γνωρίσματα, δηλαδή αντικείμενα τύπου Attr.

Η βασικότερη λειτουργία της κλάσης είναι η προσθήκη γνωρισμάτων στο κλειδί με τη μέθοδο μετάλλαξης AddToKey() (Απόσπασμα Κώδικα 11). Επίσης, έχει γίνει override η μέθοδος ToString() έτσι ώστε να επιστρέφει το κλειδί, με κατάλληλη μορφοποίηση για προβολή στην οθόνη (Απόσπασμα Κώδικα 12).

```
public void AddToKey(List<Attr> list)
{
    //ελέγχεται αν τα κλειδιά υπάρχουν ήδη στον πίνακα, κι αν όχι,
προστίθενται κανονικά
    foreach (Attr attr in list)
        if (!keyAttrs.Contains(attr))
            AddToKey(attr);
}
```

Απόσπασμα Κώδικα 11: Η μέθοδος AddToKey()

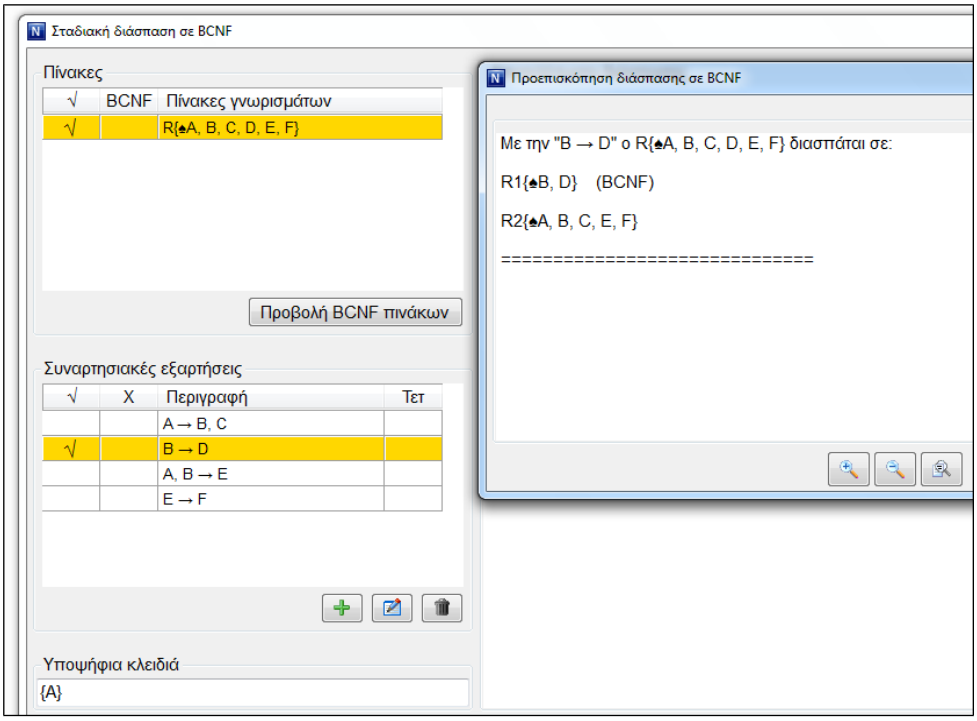
```
public override string ToString()
{
    //στον τοπικό πίνακα names προστίθενται τα ονόματα των γνωρισμάτων που
μετέχουν στο κλειδί, ταξινομούνται αλφαβητικά και επιστρέφονται ως ενιαίο,
μορφοποιημένο string
    List<string> names = new List<string>();
    foreach (Attr attr in keyAttrs)
        names.Add(attr.Name);
    names.Sort();
    string str = string.Join(", ", names);
    str = "{" + str + "}"; return str;
}
```

Απόσπασμα Κώδικα 12: Η μέθοδος ToString() της κλάσης Key

4.14 Διάσπαση πινάκων και η κλάση Relation

Η διάσπαση των πινάκων κατά BCNF στο πρόγραμμα γίνεται με δύο τρόπους:

- "αυτοματοποιημένα", με την εντολή "Κανονικοποίηση > Διάσπαση" όπου εμφανίζονται σε μια φόρμα όλα τα στάδια της διάσπασης και
- σταδιακά (Εικόνα 10), με την εντολή "Κανονικοποίηση > Σταδιακή διάσπαση" όπου ο χρήστης επιλέγει σταδιακά τον πίνακα που θέλει να διασπάσει και τη συναρτησιακή εξάρτηση που θα χρησιμοποιηθεί. Σε αυτή την περίπτωση, μπορεί κατά τη διάρκεια των διασπάσεων να δημιουργήσει ή να επεξεργαστεί τις συναρτησιακές εξαρτήσεις.



Εικόνα 10: Φόρμα σταδιακής διάσπασης σε BCNF

Για τις ανάγκες της διαδικασίας της διάσπασης σε BCNF δημιουργήθηκε η κλάση Relation, η οποία δημιουργεί αντικείμενα που αντιπροσωπεύουν πίνακες που προέκυψαν από διασπάσεις. Ωστόσο, κι ο universal πίνακας, που δημιουργεί αρχικά ο χρήστης, καταχωρείται ως αντικείμενο τύπου Relation.

Κάθε φορά που διασπάται ένας πίνακας Relation, δημιουργούνται δύο νέοι πίνακες, ο Relation1 και ο Relation2. Σε αυτούς τους δύο πίνακες διαμοιράζονται τα γνωρίσματα του διασπασμένου πίνακα με βάση τον αλγόριθμο της διάσπασης σε BCNF.

Οι πιο σημαντικές λειτουργίες της κλάσης Relation είναι η μέθοδος SetKey() που ορίζει το κλειδί του πίνακα και η μέθοδος GetList() που επιστρέφει τη λίστα με τα

γνωρίσματα του πίνακα. Επίσης, έγινε override η μέθοδος ToString() η οποία επιστρέφει την τελική μορφή του πίνακα, για προβολή στην οθόνη (Απόσπασμα Κώδικα 13).

```
public override string ToString()
{
    //στον τοπικό πίνακα names προστίθενται το όνομα του πίνακα, τα ονόματα
των γνωρισμάτων που μετέχουν στον πίνακα, τα κλειδιά τους, ταξινομούνται
αλφαβητικά και επιστρέφονται ως ενιαίο, μορφοποιημένο string
    //το "\u2660" χρησιμοποιείται για την απεικόνιση του κλειδιού
(μπαστουνιού)
    List<string> names = new List<string>();
    foreach (Attr attr in attrList)
    {
        if (key.GetAttrs().Contains(attr))
            names.Add("\u2660" + attr.Name);
        else
            names.Add(attr.Name);
    }

    names.Sort();
    string str = string.Join(", ", names);
    str = name + "{" + str + "}";
    return str;
}
```

Απόσπασμα Κώδικα 13: Η μέθοδος ToString() της κλάσης Relation

Η διαδικασία της διάσπασης εκκινείται και ελέγχεται από τη δημόσια μέθοδο Decompose() της κλάσης - φόρμας FRMMAIN. Η διαδικασία αυτή είναι εκτενής και ξεκινά τις διασπάσεις από τον universal πίνακα έως ότου φτάσει στο σημείο όλοι οι πίνακες να είναι σε BCNF μορφή. Οι διασπάσεις γίνονται με τον αλγόριθμο BCNF. Όσο για τη διαδικασία της σταδιακής διάσπασης, αυτή γίνεται με τη μέθοδο DecomposeInSteps(). Στο Απόσπασμα Κώδικα 14 παρατίθεται ένα μέρος του κώδικα αυτής της μεθόδου. Οι δύο μέθοδοι είναι σε γενικές γραμμές παρόμοιες, η διαφορά είναι ότι στη $2^{η}$ δεν ελέγχονται όλες οι συναρτησιακές εξαρτήσεις προκειμένου να γίνει μια διάσπαση πίνακα, αλλά μόνο η συναρτησιακή εξάρτηση που επιλέγει ο χρήστης.

```
private void DecomposeInSteps(bool isPreview)
{
    int i;
    int iRel = 0;
    int iFD = 0;

    //προσδιορίζεται ο πίνακας που έχει επιλεγεί
    g = GridRel;
    Relation rel = null;
    for (i = 0; i < g.RowCount; i++)
        if (g.Rows[i].Cells["colRelSel"].Value.ToString() == "\u221A")
```

```
        {
            rel = relList[i];
            iRel = i;
        }

    //προσδιορίζεται η συναρτησιακή εξάρτηση που έχει επιλεγεί
    g = GridFD;
    FD fd = null;
    for (i = 0; i < g.RowCount; i++)
        if (g.Rows[i].Cells["colFDSel"].Value.ToString() == "\u221A")
        {
            fd = FDList[i];
            iFD = i;
        }

    //αν η τομή x του συνόλου των γνωρισμάτων της συναρτησιακής εξάρτησης και
των γνωρισμάτων του πίνακα είναι μικρότερη σε αριθμό από το πλήθος των
γνωρισμάτων του πίνακα και ίση με το πλήθος των γνωρισμάτων της συναρτησιακής
εξάρτησης, τότε παραβιάζεται η BCNF μορφή και ο πίνακας μπορεί να διασπαστεί
    int x = fd.GetAll().Intersect(rel.GetList()).Count();
    if (x < rel.GetList().Count && x == fd.GetAll().Count)
    {
        //παρακάτω δημιουργούνται δύο νέοι πίνακες, ο rel1 και ο rel2

        //ο rel1 πίνακας παίρνει τα γνωρίσματα της συναρτησιακής εξάρτησης
        Relation rel1 = new Relation(fd.GetAll());

        //ο rel2 πίνακας παίρνει τα γνωρίσματα από το αριστερό σκέλος της
συναρτησιακής εξάρτησης, συν τα γνωρίσματα του πίνακα που διασπάστηκε, πλην
αυτών που βρίσκονται στο δεξί σκέλος της συναρτησιακής εξάρτησης
        List<Attr> tempo = new List<Attr>();
        tempo.AddRange(fd.GetLeft());
        tempo.AddRange(rel.GetList().Except(fd.GetRight()));
        Relation rel2 = new Relation(tempo);

        //δημιουργούνται δύο κλειδιά, ένα για τον καθένα πίνακα
        Key key1 = new Key();
        Key key2 = new Key();

        //το κλειδί του πρώτου πίνακα είναι η ορίζουσα της συναρτησιακής
εξάρτησης που προκάλεσε την διάσπαση
        key1.AddToKey(fd.GetLeft());
        rel1.SetKey(key1);

        //προσδιορίζουμε το κλειδί του δεύτερου πίνακα (αυτό που δίνει όλα τα
γνωρίσματά του)
        //δημιουργούμε μια τοπική λίστα κλειδιών και ως κλειδί του δεύτερου
πίνακα ορίζεται το πρώτο κλειδί της λίστας
        List<Key> tempoKeyList = new List<Key>();
        tempoKeyList = frmmain.KeysGet(FDList, rel2.GetList(), false);
        key2.AddToKey(tempoKeyList[0].GetAttrs());
        rel2.SetKey(key2);

        //ορίζονται τα ονόματα των δύο νέων πινάκων
        rel1.Name = rel.Name + "1";
        rel2.Name = rel.Name + "2";

        //οι δύο νέοι πίνακες προστίθενται στη λίστα
        relList.Add(rel1);
```

```
        relList.Add(rel2);

        MessageBox.Show("Έγινε διάσπαση σε δύο νέους πίνακες, τον " +
rel1.Name + " και τον " + rel2.Name + ".", this.Text, MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
    else //σε διαφορετική περίπτωση η BCNF δεν παραβιάζεται και εμφανίζεται
σχετικό μήνυμα
    {
        frmOut frm = new frmOut(this.Text, true, frmmain);
        frm.AddOut("Η συναρτησιακή εξάρτηση\n\n\"" + fd.ToString() +
"\"\n\nδεν σχετίζεται με τον πίνακα\n\n" + rel.ToString() + "\n\nκαι επομένως
δεν γίνεται διάσπαση.");
        frm.ShowDialog(this);
        frm.Dispose();
        return;
    }
}
```

Απόσπασμα Κώδικα 14: Η μέθοδος DecomposeInSteps()

4.15 Ρυθμίσεις προγράμματος

Κρίθηκε σκόπιμο στο μενού "Το πρόγραμμα" να προστεθεί η εντολή "Ρυθμίσεις", ώστε ο χρήστης να μπορεί να αλλάζει ορισμένες από τις παραμέτρους του προγράμματος. Αυτές είναι το μέγεθος της γραμματοσειράς της φόρμας των αποτελεσμάτων, η χρήση ή όχι αύξοντα αριθμού στις ονομασίες των διασπασμένων πινάκων και οι τύποι γνωρισμάτων. Από προεπιλογή, στο πρόγραμμα συμπεριλήφθηκαν οι εξής τύποι: blob, boolean, char, date, double, enum, float, integer, long, number, single, text, time, varchar. Ο χρήστης έχει τη δυνατότητα να προσθέτει επιπλέον τύπους ως προεπιλογή ή να τροποποιεί - διαγράφει τους υπάρχοντες.

5 Επίλογος

5.1 Μελλοντικές επεκτάσεις

Αρχικός σκοπός της διπλωματικής εργασίας ήταν η δημιουργία του εκπαιδευτικού εργαλείου - προγράμματος, το οποίο θα μπορεί να χρησιμοποιείται μελλοντικά από τους φοιτητές στο μάθημα των Βάσεων Δεδομένων και της κανονικοποίησης. Κάθε feedback από τους χρήστες του προγράμματος, με παρατηρήσεις και ιδέες για μελλοντική, περαιτέρω ανάπτυξή του είναι ευπρόσδεκτο.

Πλέον αυτού όμως, ο κώδικας του προγράμματος είναι προς ελεύθερη χρήση από τον καθένα, ώστε αν κάποιος ενδιαφερθεί να αναπτύξει κι άλλο το πρόγραμμα ή να προσεγγίσει την διαδικασία της κανονικοποίησης από άλλη οπτική, να μπορεί να το πράξει έχοντάς το σαν οδηγό.

Το πρόγραμμα που δημιουργήθηκε μπορεί να αναπτυχθεί περαιτέρω σε διάφορα επίπεδα. Μεταξύ άλλων, μερικές ιδέες για προσθήκες στο πρόγραμμα είναι οι εξής:

- ✓ Δυνατότητα αναίρεσης ενεργειών, π.χ. μιας κατά λάθος διαγραφή ενός γνωρίσματος από τον χρήστη.
- ✓ Δυνατότητα εκτύπωσης ή εξαγωγής σε άλλη μορφή αρχείου (π.χ. PDF) των αποτελεσμάτων των διαδικασιών του προγράμματος.
- ✓ Μετάφραση του περιβάλλοντος διεπαφής με το χρήστη (GUI) στα αγγλικά και σε άλλες γλώσσες.
- ✓ Επέκταση του προγράμματος με σκοπό την κάλυψη και των επόμενων κανονικών μορφών (4NF, 5NF κλπ).

Βιβλιογραφία - Αρθρογραφία (πηγές)

Βρέντζου, Τ., *Σύνολα*, Διαθέσιμο: http://ma8eno.gr/attachments/File/sinolalocked.pdf, (21 Φεβρουαρίου 2016)

Κεχρής, Ε. (2005). *Σχεσιακές Βάσεις Δεδομένων*. Αθήνα: Εκδόσεις Κριτική Α.Ε.

Κούτρας, Μ. (2006). *Εισαγωγή στη Συνδυαστική*. Αθήνα: Εκδόσεις Σταμούλη Α.Ε.

Andersen, V. (2001). *Access 2002: The Complete Reference*. USA: McGraw-Hill

Armstrong, W. (1974). *Dependency Structures of Data Base Relationships*, IFIP Congress: 580 - 583

Chen, P. (1976). *The Entity-Relationship Model - Toward a Unified View of Data,* ACM Transactions on Database Systems, Volume 1, 1: 9-36

Codd, E. (1970). *A relational Model of Data for Large Shared Data Banks*, Communications of the ACM, Volume 13, 6: 377-387

Codd, E. (1972). *Further Normalization of the Data Base Relational Model*, Rustin: 33 – 64

Codd, E. (1974). *Recent Investigations into Relational Data Base Systems*, IBM Research Report RJ1385

EduGrabs, *Trivial and Non Trivial Functional Dependency examples*, 29 Ιουνίου 2015, Διαθέσιμο: http://www.edugrabs.com/trivial-and-non-trivial-functional-dependency-examples/, (21 Φεβρρουαρίου 2016)

Fagin, R. (1977). *Multivalued Dependencies and a New Normal Form for Relational Databases,* ACM Transactions on Database Systems, Volume 2, 3: 262-278

Forouzan, B., Gilberg, R. (2002). *Foundations of Computer Science: From Data Manipulation to Theory of Computation*. USA: Brooks / Cole Publishing Co. Pacific Grove

Microsoft, *Πλεονεκτήματα του Microsoft .NET Framework*, 12 Μαρτίου 2007. Διαθέσιμο: https://support.microsoft.com/el-gr/kb/936704, (29 Οκτωβρίου 2015)

Pelland, P. (2006). *Build a program now! Microsoft Visual C# 2005 Express Edition*. Washington: Microsoft Press

Ramakrishnan, P., Gehrke, J. (2000). *Database Management Systems*. New York: The McGraw-Hill Companies

Ricardo, C. (2012). *Databases Illuminated*. Sudbury: Jones & Bartlett Learning

Schildt, H. (2009). *C# The Complete Reference*. New York: The McGraw-Hill Companies

Silberschatz, A., Korth, H., Sudarshan, S. (2002). *Database System Concepts.* New York: The McGraw-Hill Companies

Teorey, T., Lightstone, S., Nadean, T. (2006). *Database Modeling and Design.* San Fransisco: Elsevier Inc.

w3schools, *SQL General Data Types*, Διαθέσιμο: http://www.w3schools.com/sql/sql_datatypes_general.asp, (4 Φεβρουαρίου 2016)

Wikipedia, *Database*, 19 Φεβρουαρίου 2016, Διαθέσιμο: https://en.wikipedia.org/wiki/Database, (22 Φεβρουαρίου 2016)

Wikipedia, *Database normalization,* 1 Φεβρουαρίου 2016, Διαθέσιμο: https://en.wikipedia.org/wiki/Database_normalization, (22 Φεβρουαρίου 2016)

Wikipedia, *Edgar F. Codd*, 10 Φεβρουαρίου 2016, Διαθέσιμο: https://en.wikipedia.org/wiki/Edgar_F._Codd, (29 Φεβρουαρίου 2016)

Wikipedia, *.NET Framework*, 19 Φεβρουαρίου 2016, Wikipedia, Διαθέσιμο: https://en.m.wikipedia.org/wiki/.NET_Framework, (22 Φεβρουαρίου 2016)

Παράρτημα Α

Στις επόμενες σελίδες παρατίθεται αυτούσιο το εγχειρίδιο (manual) με τις σύντομες οδηγίες χρήσης του προγράμματος. Το εγχειρίδιο αυτό είναι επίσης διαθέσιμο μέσα στο πρόγραμμα, με την εντολή "Το πρόγραμμα > Οδηγίες χρήσης", καθώς επίσης και online, στη διεύθυνση:

http://www.myprograms.eu/files/normalization_app_manual.pdf

Καθώς το πρόγραμμα θα συνεχίσει να αναπτύσσεται και να εξελίσσεται και μετά το πέρας της διπλωματικής εργασίας, εξυπακούεται ότι το περιεχόμενο των επόμενων σελίδων μπορεί να διαφέρει από τις μελλοντικές εκδόσεις του εγχειριδίου.

Εκπαιδευτικό Εργαλείο Κανονικοποίησης

Σύντομες οδηγίες χρήσης

Εισαγωγή

Το πρόγραμμα **Εκπαιδευτικό Εργαλείο Κανονικοποίησης** αυτοματοποιεί τη διαδικασία της κανονικοποίησης πινάκων σε **BCNF μορφή**.

Ο χρήστης με το πρόγραμμα δημιουργεί ένα **σχήμα βάσης δεδομένων**, στο οποίο προσθέτει γνωρίσματα και δημιουργεί συναρτησιακές εξαρτήσεις με αυτά. Εν συνεχεία, το πρόγραμμα εντοπίζει τα **υποψήφια κλειδιά** του σχήματος και πραγματοποιεί διάσπαση πινάκων, κανονικοποιώντας το αρχικό σχήμα σε μικρότερους, κανονικούς **BCNF πίνακες**. Με αυτό τον τρόπο απλουστεύεται η δημιουργία του σχήματος μιας βάσης δεδομένων και περιορίζονται σημαντικά τα λάθη από ανωμαλίες στην εισαγωγή και επεξεργασία των εγγραφών της.

Το πρόγραμμα αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας μου, στο Πρόγραμμα Μεταπτυχιακών Σπουδών Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας. Ευχαριστώ θερμά την επιβλέπουσα καθηγήτρια της εργασίας, κυρία Γεωργία Κολωνιάρη, για την πολύτιμη βοήθειά της, σε όλα τα στάδια της ανάπτυξης του προγράμματος.

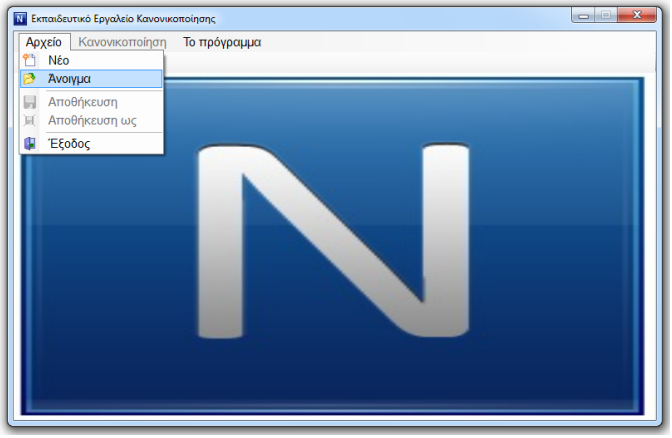
Μπορείτε να δείτε την διπλωματική εργασία με την εντολή "Το πρόγραμμα > Η εργασία".

Οι απορίες και οι προτάσεις σας για περαιτέρω ανάπτυξη του προγράμματος είναι φυσικά ευπρόσδεκτες, μέσω email στην ηλεκτρονική διεύθυνση info@myprograms.eu.

Κεντρική φόρμα

Ξεκινώντας το πρόγραμμα, έχετε τη δυνατότητα είτε να δημιουργήσετε ένα **νέο σχήμα** με την εντολή "Αρχείο > Νέο", είτε να φορτώσετε ένα παλιότερο επιλέγοντας "Άνοιγμα".

Με την εντολή "Το πρόγραμμα > Ρυθμίσεις" μπορείτε να παραμετροποιήσετε ορισμένες από τις λειτουργίες του προγράμματος.



Νέο σχήμα

Αφού δημιουργήσετε ένα νέο σχήμα και προσδιορίσετε το όνομά του, ξεκινήστε εισάγοντας τα γνωρίσματα και τις συναρτησιακές εξαρτήσεις.

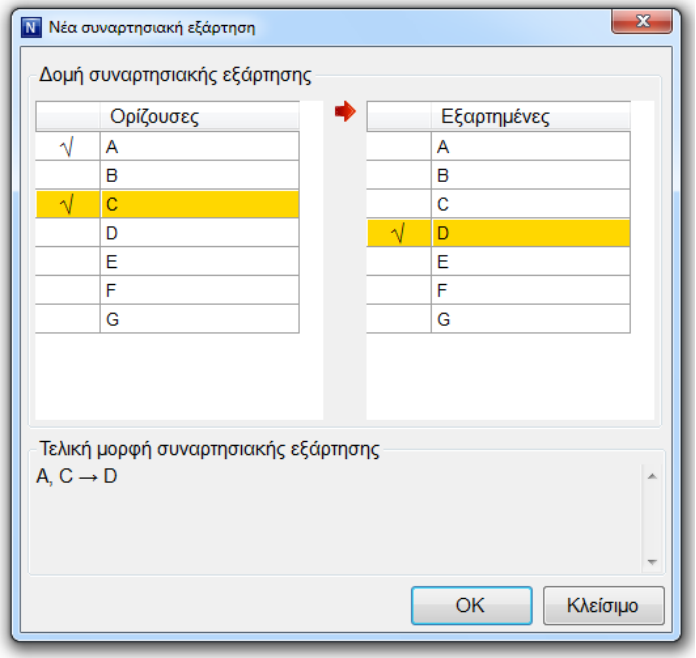
Για να εισάγετε ένα **νέο γνώρισμα**, πατήστε το κουμπί με το πράσινο "+" κάτω από τον αριστερό πίνακα "Γνωρίσματα". Θα σας ζητηθεί να πληκτρολογήσετε το όνομα του γνωρίσματος και τον τύπο του (προαιρετικά), π.χ. string, boolean κλπ. Κάθε γνώρισμα έχει δικό του, μοναδικό όνομα. Μπορείτε να εισάγετε ταυτόχρονα μια **ακολουθία γνωρισμάτων**, χρησιμοποιώντας το σύμβολο ">", με την προϋπόθεση το όνομα να είναι μόνο ένα γράμμα του αλφαβήτου, στα κεφαλαία και στην ίδια γλώσσα. Επίσης, η σειρά πρέπει να είναι αύξουσα. Δηλαδή, πληκτρολογώντας "A>E" και πατώντας "OK", θα δημιουργηθούν ταυτόχρονα 4 γνωρίσματα, τα Α, Β, Γ, Δ και Ε.

Κάθε φορά που κάνετε μια αλλαγή στο πρόγραμμα, όπως δημιουργία γνωρίσματος, οι αλλαγές πρέπει να **αποθηκεύονται** στον Η/Υ σας. Πατήστε το κουμπί "Αποθήκευση" (ή "Αποθήκευση ως") προκειμένου να αποθηκεύσετε το σχήμα που δουλεύετε.

Τα γνωρίσματα που δημιουργείτε εμφανίζονται στον αριστερό πίνακα. Από εκεί, μπορείτε να τα επεξεργαστείτε, αλλάζοντας τα στοιχεία τους, κάνοντας είτε διπλό κλικ πάνω τους, είτε πατώντας το κουμπί "Επεξεργασία γνωρίσματος" κάτω από τον πίνακα. Πατώντας το κουμπί "Διαγραφή γνωρίσματος" ή το πλήκτρο Delete του πληκτρολογίου σας, το γνώρισμα θα

διαγραφεί. Δεν θα επιτραπεί η διαγραφή του γνωρίσματος αν αυτό συμμετέχει σε μια συναρτησιακή εξάρτηση.

Μπορείτε να χρησιμοποιήσετε τα γνωρίσματα που καταχωρήσατε για να δημιουργήσετε **συναρτησιακές εξαρτήσεις**. Για να το κάνετε αυτό, πατήστε το κουμπί με το πράσινο "συν" κάτω από τον δεξί πίνακα "Συναρτησιακές εξαρτήσεις". Στο νέο παράθυρο που θα ανοίξει, επιλέξτε από τον αριστερό πίνακα τις **ορίζουσες** κι από τον δεξί πίνακα τις **εξαρτημένες** ιδιότητες της συναρτησιακής εξάρτησης. Για να επιλέξετε γνωρίσματα, κάνετε κλικ στο αντίστοιχο κελί της πρώτης στήλης. Στο κάτω μέρος του παραθύρου θα εμφανιστεί η τελική μορφή της συναρτησιακής εξάρτησης, έτσι όπως θα καταχωρηθεί αφού πατήσετε "OK". Μην παραλείψετε, εν συνεχεία, να αποθηκεύσετε τις αλλαγές που κάνατε στο σχήμα.



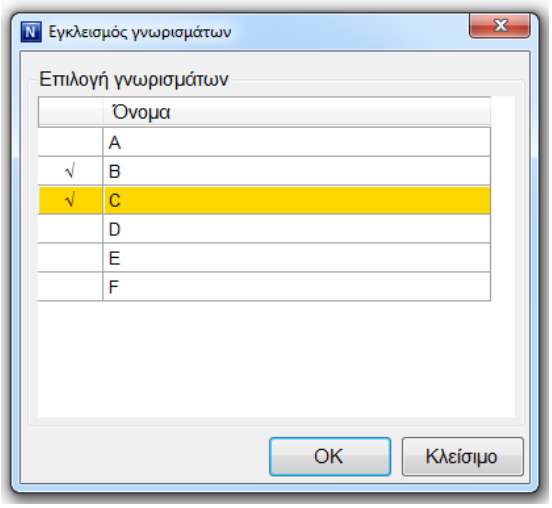
Η νέα συναρτησιακή εξάρτηση προστίθεται και εμφανίζεται πια στον πίνακα "Συναρτησιακές εξαρτήσεις". Στην τελευταία στήλη "Τετ" εμφανίζεται ένα σύμβολο "ν" σε περίπτωση που η συναρτησιακή εξάρτηση είναι **τετριμμένη**. Μπορείτε να επεξεργαστείτε μια συναρτησιακή εξάρτηση, κάνοντας είτε διπλό κλικ πάνω της, είτε πατώντας το κουμπί "Επεξεργασία συναρτησιακής εξάρτησης" κάτω από τον πίνακα. Πατώντας το κουμπί "Διαγραφή συναρτησιακής εξάρτησης" ή το πλήκτρο Delete του πληκτρολογίου σας, η συναρτησιακή εξάρτηση θα διαγραφεί. Ωστόσο, οι διαδικασίες αυτές δεν είναι οριστικές αν δεν αποθηκεύσετε το σχήμα. Δηλαδή, αν π.χ. διαγράψετε κατά λάθος μια συναρτησιακή εξάρτηση, μπορείτε να δοκιμάσετε να επαναφέρετε το σχήμα σε μια πρότερη μορφή του φορτώνοντάς το ξανά με την εντολή "Αρχείο > Άνοιγμα".

Κανονικοποίηση

Έχοντας δημιουργήσει ένα σχήμα με γνωρίσματα και συναρτησιακές εξαρτήσεις, μπορείτε να προχωρήσετε στις λειτουργίες της κανονικοποίησης από το αντίστοιχο μενού εντολών της.

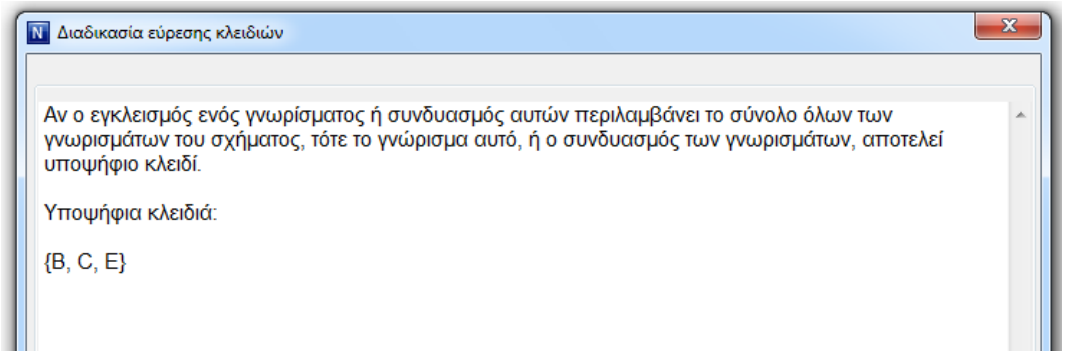
> Εγκλεισμός

Στο παράθυρο που θα ανοίξει, επιλέξτε με κλικ στην πρώτη στήλη τα γνωρίσματα εκείνα των οποίων θέλετε να υπολογίσετε τον εγκλεισμό. Αν επιλέξετε δύο ή περισσότερα γνωρίσματα θα προσδιοριστεί ο εγκλεισμός τους ως συνδυασμός, κι όχι ξεχωριστά. Αν δηλαδή, π.χ. καταχωρήσετε τη συναρτησιακή εξάρτηση "Α, Β → Γ" και θέλετε να υπολογίσετε τον εγκλεισμό του Α, η τιμή που θα επιστραφεί θα είναι μόνο το Α, αφού το Γ το προσδιορίζει μόνο ο συνδυασμός των Α και Β μαζί.



> Κλειδιά

Εμφανίζονται τα υποψήφια κλειδιά (ένα ή και περισσότερα) του σχήματος, ανάλογα με τις συναρτησιακές εξαρτήσεις που δημιουργήσατε.



> Διάσπαση

Πραγματοποιεί **αυτοματοποιημένη διάσπαση** κατά BCNF του αρχικού πίνακα, όλων των γνωρισμάτων που δημιουργήσατε στην κεντρική φόρμα του προγράμματος και στο τέλος εμφανίζει όλους τους BCNF πίνακες που προέκυψαν από αυτή.

Πριν ξεκινήσει η διαδικασία της διάσπασης, το πρόγραμμα προσδιορίζει, με τη βοήθεια των συναρτησιακών εξαρτήσεων, τα υποψήφια κλειδιά του σχήματος. Αμέσως μετά, αφαιρεί αυτομάτως από τις συναρτησιακές εξαρτήσεις όσες είναι τετριμμένες και όσες έχουν υποσύνολο του υποψήφιου κλειδιού στο αριστερό σκέλος τους. Επιπλέον, το πρόγραμμα χρησιμοποιεί το αξίωμα της μεταβατικότητας του Armstrong, συνενώνοντας τις αντίστοιχες συναρτησιακές εξαρτήσεις. Για παράδειγμα, αν έχουμε "Α → Β" και "Β → Γ", η πρώτη συναρτησιακή εξάρτηση θα προσθέσει στο δεξί της σκέλος την εξαρτημένη της Β, οπότε θα γίνει "Α → Β, Γ".

Στη συνέχεια πραγματοποιούνται οι αυτόματες διασπάσεις, με τη σειρά που καταχωρήσατε τις συναρτησιακές εξαρτήσεις, και τα αποτελέσματα, δηλαδή οι BCNF πίνακες, καθώς και οι συναρτησιακές εξαρτήσεις που δε χρησιμοποιήθηκαν, εμφανίζονται στο παράθυρο αποτελεσμάτων του προγράμματος. Αν κάποια συναρτησιακή εξάρτηση δε χρησιμοποιήθηκε, τότε το πρόγραμμα προτείνει μια **εναλλακτική λύση**, κάνοντας τη διάσπαση με την αντίστροφη σειρά των συναρτησιακών εξαρτήσεων. Για να δείτε την εναλλακτική λύση, πατήστε το κουμπί "Εναλλακτική", κάτω αριστερά στο παράθυρο αποτελεσμάτων.

Σε κάθε πίνακα που προκύπτει από τη διάσπαση, το πρόγραμμα προσδιορίζει ποιο είναι το κλειδί του και προσθέτει το σύμβολο ♠ (μπαστούνι) πριν το όνομα του γνωρίσματος. Αν το κλειδί είναι σύνθετο, τότε εμφανίζονται πολλαπλά ♠, ένα για κάθε γνώρισμα του κλειδιού. Αν τέλος ένας πίνακας έχει περισσότερα από ένα υποψήφια κλειδιά, το πρόγραμμα επιλέγει το πρώτο από αυτά.

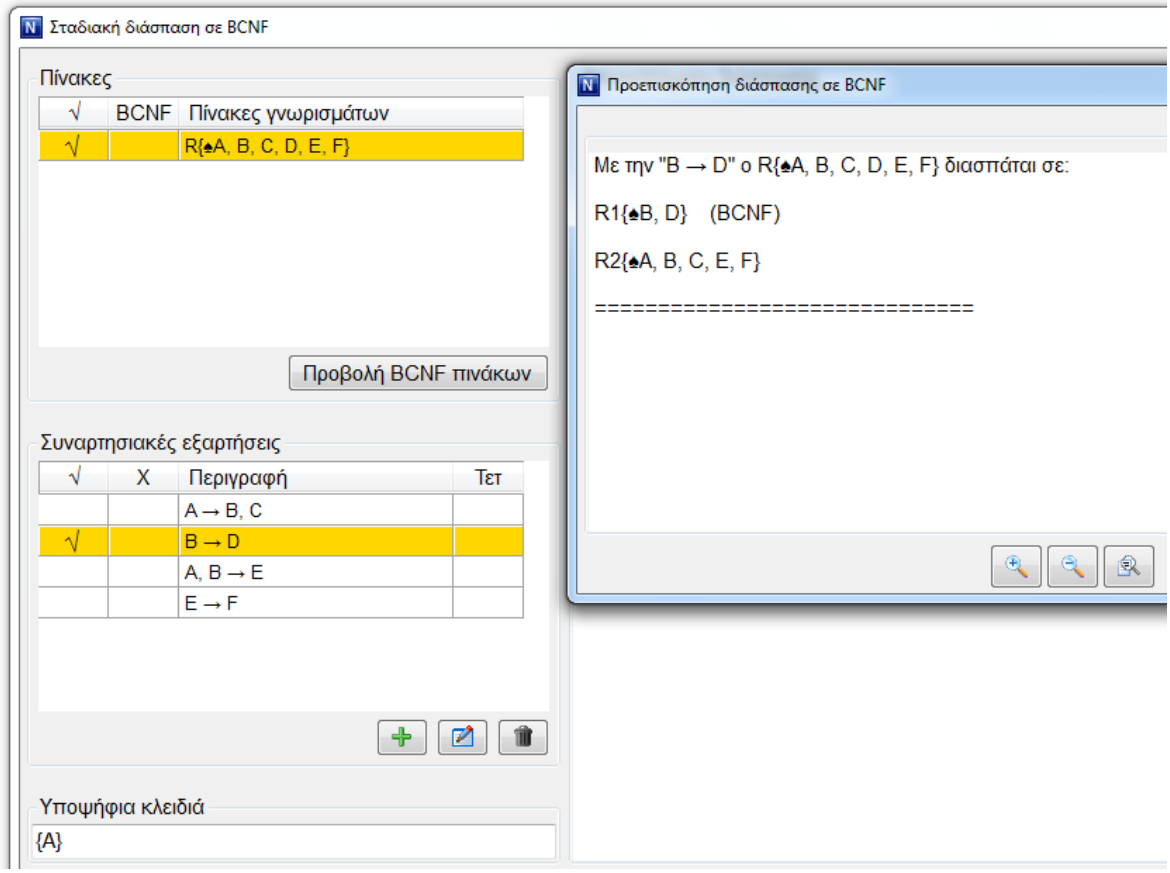
> Σταδιακή διάσπαση

Με τη σταδιακή διάσπαση μπορείτε να διασπάσετε σταδιακά τους πίνακες, επιλέγοντας κάθε φορά ποιον πίνακα θέλετε να διασπάσετε και με τη βοήθεια ποιας συναρτησιακής εξάρτησης. Για να επιλέξετε τα στοιχεία που θέλετε, κάνετε κλικ στο αντίστοιχο κελί της πρώτης στήλης ώστε να εμφανιστεί το σύμβολο "ν". Αφού το κάνετε αυτό, μπορείτε μετά να πατήσετε κάτω αριστερά είτε "**Προεπισκόπηση**" αν θέλετε να δείτε απλώς ποιο θα είναι το αποτέλεσμα της διάσπασης, είτε "**Διάσπαση**" για να προχωρήσει κανονικά η διαδικασία και να καταχωρηθούν οι νέοι πίνακες.

Αν η συναρτησιακή εξάρτηση που επιλέξατε παραβιάζει την BCNF μορφή του πίνακα που επίσης διαλέξατε, τότε η διάσπαση θα γίνει και θα δημιουργηθούν δύο νέοι πίνακες οι οποίοι θα προστεθούν στους πίνακες γνωρισμάτων και θα εμφανιστούν στα αποτελέσματα της διάσπασης. Αν μια συναρτησιακή εξάρτηση που επιλέξατε είναι τετριμμένη ή περιέχει υποσύνολο του υποψήφιου κλειδιού του αρχικού πίνακα στο αριστερό σκέλος της ή τα γνωρίσματά της δεν περιλαμβάνονται όλα στον προς διάσπαση πίνακα, τότε η διάσπαση δεν πραγματοποιείται.

Μπορείτε να προσθέσετε νέες συναρτησιακές εξαρτήσεις, να τις τροποποιήσετε ή να τις διαγράψετε. Εντούτοις, οι αλλαγές που θα κάνετε δεν επηρεάζουν τις συναρτησιακές εξαρτήσεις του αρχικού σχήματος. Επιπλέον, αν στο μεταξύ έχετε προχωρήσει ήδη σε διασπάσεις πινάκων, οι ενδεχόμενες τροποποιήσεις που θα κάνετε στις συναρτησιακές εξαρτήσεις δεν επηρεάζουν την κατάσταση των διασπασμένων πινάκων, ούτε τα υποψήφια κλειδιά τους που έχουν ήδη οριστεί. Με απλό κλικ στην στήλη "X" των συναρτησιακών εξαρτήσεων μπορείτε να επισημάνετε ότι τη συγκεκριμένη την έχετε εξετάσει. Η επισήμανση "X" εμφανίζεται επίσης αυτόματα στους πίνακες και τις συναρτησιακές εξαρτήσεις που έχουν ήδη χρησιμοποιηθεί.

Με διπλό κλικ πάνω σε έναν πίνακα μπορείτε να δείτε όλα τα υποψήφια κλειδιά του. Τέλος, με το κουμπί **"Προβολή BCNF πινάκων"** εμφανίζονται όλοι BCNF πίνακες που έχουν προκύψει.



Παράθυρο αποτελεσμάτων

Στο κάτω μέρος του παραθύρου αποτελεσμάτων υπάρχουν τα κουμπιά ρύθμισης του μεγέθους της **γραμματοσειράς**. Στις Ρυθμίσεις του προγράμματος μπορείτε να ορίσετε το προεπιλεγμένο μέγεθος που επιθυμείτε.

Με την εντολή "**Εξαγωγή**" το περιεχόμενο του παραθύρου εξάγεται ως αρχείο html και ανοίγει στον επιλεγμένο browser σας, όπου μπορείτε να το εκτυπώσετε, αποθηκεύσετε κλπ.

Μπορείτε ελεύθερα να γράψετε ή να αφαιρέσετε κείμενο από το παράθυρο αποτελεσμάτων. Ότι αλλαγές κάνετε, θα εμφανιστούν και στο html σε περίπτωση που κάνετε εξαγωγή του περιεχομένου. Πατώντας τη συντόμευση Ctrl + A επιλέγεται όλο το κείμενο του παραθύρου.

Εκπαιδευτικό Εργαλείο Κανονικοποίησης

Προγραμματισμός: Χρήστος Στεφανίδης

email: info@myprograms.eu

http://www.myprograms.eu/

Σύμβουλος προγραμματισμού: Γεωργία Κολωνιάρη

email: gkoloniari@uom.edu.gr

http://users.uom.gr/~gkoloniari/

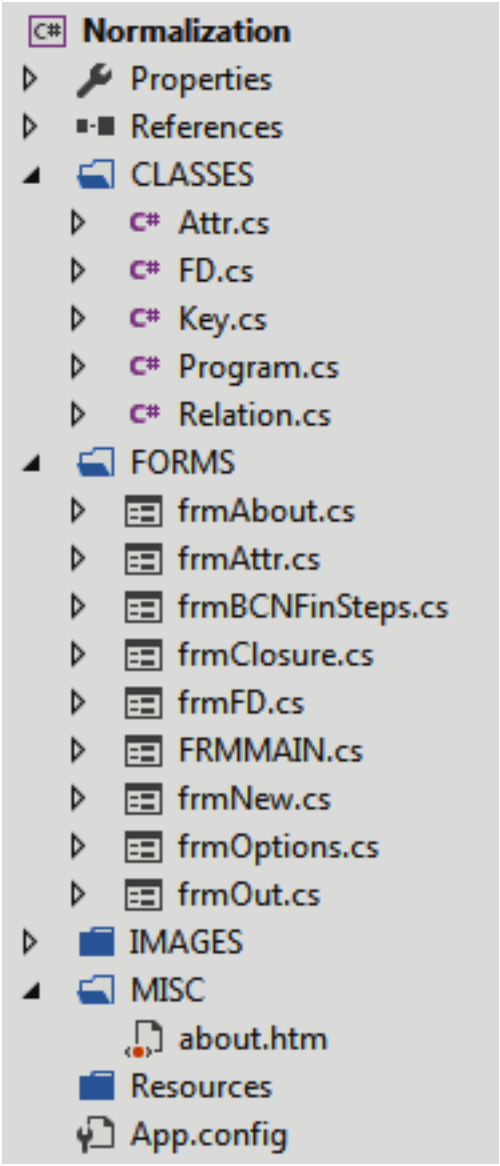
Παράρτημα Β

Στις επόμενες σελίδες παρατίθεται αυτούσιος ο πηγαίος κώδικας του προγράμματος, στη μορφή κατά την οποία ήταν κατά την ημέρα που αυτό ολοκληρώθηκε. Καθώς το πρόγραμμα θα συνεχίσει να αναπτύσσεται και να εξελίσσεται και μετά το πέρας της διπλωματικής εργασίας, εξυπακούεται ότι το περιεχόμενο των επόμενων σελίδων μπορεί να διαφέρει σημαντικά από τον πηγαίο κώδικα των μελλοντικών αναβαθμίσεων του προγράμματος.

Αρχικά, παρουσιάζεται η διάρθρωση του project και εν συνεχεία ο κώδικας των επιμέρους κλάσεών του. Στις περιπτώσεις των κλάσεων που έχουν και οπτική μορφή παρατίθεται και ένα απλό screenshot σε φάση σχεδίασης της φόρμας.

Συνολικά, το project απαρτίζεται από 14 κλάσεις, εκ των οποίων οι 9 έχουν και οπτική μορφή (WinForms). Η διαδικασία της ανάπτυξής του ολοκληρώθηκε στις 14/03/2016 και διήρκεσε περίπου 105 ημέρες. Αποτελείται από 3.500 γραμμές κώδικα, συμπεριλαμβανομένων και των σχολίων (comments). Το συνολικό μέγεθος του φακέλου που το φιλοξενεί είναι 13 MB.

Διάρθρωση project



Attr.cs

```
public class Attr //η κλάση Attr δημιουργεί γνωρίσματα του σχήματος
{
    //μεταβλητές με τα χαρακτηριστικά του γνωρίσματος
    string name; //όνομα
    string type; //τύπος
    bool exclude; //αποκλείει τη συμμετοχή σε κλειδί

    /// <summary>
    /// Κατασκευαστής του γνωρίσματος Attr
    /// </summary>
    /// <param name="name">Ονομασία γνωρίσματος</param>
    /// <param name="type">Τύπος γνωρίσματος</param>
    public Attr(string name, string type)
    {
        //κατά τη δημιουργία του γνωρίσματος, περνιούνται στις αντίστοιχες
μεταβλητές η ονομασία και ο τύπος του
        this.name = name;
        this.type = type;
    }

    /// <summary>
    /// Ονομασία του γνωρίσματος
    /// </summary>
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    /// <summary>
    /// Τύπος του γνωρίσματος
    /// </summary>
    public string Type
    {
        get { return type; }
        set { type = value; }
    }

    /// <summary>
    /// Αποκλείει τη συμμετοχή σε κλειδί
    /// </summary>
    public bool Exclude
    {
        get { return exclude; }
        set { exclude = value; }
    }
}
```

FD.cs

```
using System.Collections.Generic;
using System.Linq;

public class FD //η κλάση FD δημιουργεί συναρτησιακές εξαρτήσεις (functional
dependency)
{
    //οι πίνακες Left και Right περιέχουν τα γνωρίσματα που βρίσκονται στο
αριστερό και το δεξί σκέλος αντίστοιχα της συναρτησιακής εξάρτησης
    //αντίστοιχα, οι BLeft και BRight είναι τα backup τους
```

```
    private List<Attr> Left = new List<Attr>();
    private List<Attr> BLeft = new List<Attr>();
    private List<Attr> Right = new List<Attr>();
    private List<Attr> BRight = new List<Attr>();
    private int aa; //υποδηλώνει τον αύξοντα αριθμό της συναρτησιακής
εξάρτησης
    private bool isTrivial; //υποδηλώνει αν η συναρτησιακή εξάρτηση είναι
τετριμμένη
    private bool excluded; //υποδηλώνει αν η συναρτησιακή εξάρτηση αποκλείεται
    public static int count; //υποδηλώνει το πλήθος των συναρτησιακών
εξαρτήσεων

    /// <summary>
    /// Κατασκευαστής συναρτησιακής εξάρτησης
    /// </summary>
    public FD()
    {
        count++;
        aa = count;
        Excluded = false;
    }

    /// <summary>
    /// Αντιγράφει τα περιεχόμενα της συναρτησιακής εξάρτησης fd
    /// </summary>
    /// <param name="fd">Η συναρτησιακή εξάρτηση την οποία
αντιγράφουμε</param>
    public void Clone(FD fd)
    {
        IsTrivial = fd.isTrivial;
        Left.AddRange(fd.GetLeft());
        Right.AddRange(fd.GetRight());
    }

    /// <summary>
    /// Επιστρέφει τον αύξοντα αριθμό της συναρτησιακής εξάρτησης
    /// </summary>
    public int GetAA()
    {
        return aa;
    }

    /// <summary>
    /// Δημιουργεί backup της συναρτησιακής εξάρτησης
    /// </summary>
    public void Backup()
    {
        BLeft.Clear();
        foreach (Attr attr in Left)
            BLeft.Add(attr);
        BRight.Clear();
        foreach (Attr attr in Right)
            BRight.Add(attr);
    }

    /// <summary>
    /// Επαναφέρει τη συναρτησιακή εξάρτηση από το backup
    /// </summary>
    public void Restore()
```

```
    {
        Reset();
        foreach (Attr attr in BLeft)
            Left.Add(attr);
        foreach (Attr attr in BRight)
            Right.Add(attr);
        Trivial();
    }

    /// <summary>
    /// Επιστρέφει true αν το γνώρισμα περιλαμβάνεται στην συναρτησιακή
εξάρτηση FD
    /// </summary>
    /// <param name="attrcheck">Το αντικείμενο - γνώρισμα που
ελέγχουμε</param>
    public bool CheckAttr(Attr attrcheck)
    {
        foreach (Attr attr in Left)
            if (attrcheck == attr) return true;
        foreach (Attr attr in Right)
            if (attrcheck == attr) return true;
        return false;
    }

    /// <summary>
    /// Επαναφέρει το αντικείμενο στην αρχική του μορφή
    /// </summary>
    public void Reset()
    {
        Left.Clear();
        Right.Clear();
        isTrivial = false;
    }

    /// <summary>
    /// Προσθέτει ένα γνώρισμα τύπου Attr στο αριστερό σκέλος της FD
    /// </summary>
    public void AddLeft(Attr attr)
    {
        //ελέγχεται αν το γνώρισμα υπάρχει ήδη στα αριστερά, κι αν όχι, τότε
προστίθεται
        if (!GetLeft().Contains(attr))
        {
            Left.Add(attr);
            Trivial();
        }
    }

    /// <summary>
    /// Προσθέτει μια λίστα τύπου Attr στο αριστερό σκέλος της FD
    /// </summary>
    public void AddLeft(List<Attr> attrList)
    {
        //προστίθενται ένα ένα τα γνωρίσματα της attrList στα αριστερά
        foreach (Attr attr in attrList)
            AddLeft(attr);
    }

    /// <summary>
```

```
    /// Προσθέτει ένα γνώρισμα τύπου Attr στο δεξί σκέλος της συναρτησιακής
εξάρτησης
    /// </summary>
    public void AddRight(Attr attr)
    {
        //ελέγχεται αν το γνώρισμα υπάρχει ήδη στα δεξιά, κι αν όχι, τότε
προστίθεται
        if (!GetRight().Contains(attr))
        {
            Right.Add(attr);
            Trivial();
        }
    }

    /// <summary>
    /// Προσθέτει μια λίστα τύπου Attr στο δεξί σκέλος της συναρτησιακής
εξάρτησης
    /// </summary>
    public void AddRight(List<Attr> attrList)
    {
        //προστίθενται ένα ένα τα γνωρίσματα της attrList στα δεξιά
        foreach (Attr attr in attrList)
            AddRight(attr);
    }

    /// <summary>
    /// Αφαιρείται η λίστα τύπου Attr από το δεξί σκέλος της συναρτησιακής
εξάρτησης
    /// </summary>
    public void RemoveRight(List<Attr> attrList)
    {
        Right = Right.Except(attrList).ToList();
    }

    /// <summary>
    /// Επιστρέφει την τελική μορφή της συναρτησιακής εξάρτησης.
    /// </summary>
    public override string ToString()
    {
        //δημιουργούνται οι πίνακες LeftT και RightT
        List<string> LeftT = new List<string>();
        List<string> RightT = new List<string>();

        //σε αυτούς τους πίνακες προστίθενται οι ονομασίες των γνωρισμάτων των
πινάκων Left και Right
        foreach (Attr attr in Left)
            LeftT.Add(attr.Name);
        foreach (Attr attr in Right)
            RightT.Add(attr.Name);

        //οι πίνακες LeftT και RightT ταξινομούνται αλφαβητικά
        LeftT.Sort();
        RightT.Sort();

        //διαβάζονται τα περιεχόμενα των πινάκων LeftT και RightT και
δημιουργείται η τελική μορφή της συναρτησιακής εξάρτησης η οποία και
επιστρέφεται ως string
        string str = "";
        string left = "";
```

```
            string right = "";

            //πρώτα του πίνακα LeftT
            for (int i = 0; i < LeftT.Count; i++)
            {
                if (left != "") left = left + ", ";
                left = left + LeftT[i];
            }

            //μετά του πίνακα RightT
            for (int i = 0; i < RightT.Count; i++)
            {
                if (right != "") right = right + ", ";
                right = right + RightT[i];
            }

            //σχηματίζεται η τελική μορφή της συναρτησιακής εξάρτησης
            //το "\u2192" χρησιμοποιείται για την απεικόνιση του δεξιού βέλους
            str = left + " " + "\u2192" + " " + right;

            return str;
        }

        /// <summary>
        /// Επιστρέφει τη λίστα με τα αντικείμενα - γνωρίσματα στο αριστερό σκέλος
της συναρτησιακής εξάρτησης
        /// </summary>
        public List<Attr> GetLeft()
        {
            return Left;
        }

        /// <summary>
        /// Επιστρέφει τη λίστα με τα αντικείμενα - γνωρίσματα στο δεξί σκέλος της
συναρτησιακής εξάρτησης
        /// </summary>
        public List<Attr> GetRight()
        {
            return Right;
        }

        /// <summary>
        /// Επιστρέφει τη λίστα με όλα τα γνωρίσματα της συναρτησιακής εξάρτησης
        /// </summary>
        public List<Attr> GetAll()
        {
            return Left.Concat(Right).Except(Right.Intersect(Left)).ToList();
        }

        /// <summary>
        /// Ελέγχει αν η συναρτησιακή εξάρτηση είναι τετριμμένη ή όχι και
ενημερώνει την σχετική μεταβλητή
        /// </summary>
        private void Trivial()
        {
            //Πρέπει και τα δύο σκέλη να είναι μη κενά
            if (Right.Count * Left.Count == 0)
            {
                IsTrivial = false;
```

```
            return;
        }

        //ο παρακάτω βρόχος εξετάζει κατά σειρά όλα τα γνωρίσματα του δεξιού
σκέλους, κι αν κάποιο από αυτά δεν περιλαμβάνεται στο αριστερό σκέλος, τότε η
συναρτησιακή εξάρτηση θεωρείται μη τετριμμένη και η μέθοδος διακόπτεται
        IsTrivial = true;
        foreach (Attr attr in Right)
        {
            if (!Left.Contains(attr))
            {
                IsTrivial = false;
                return;
            }
        }
    }

    /// <summary>
    /// Προσδιορίζει αν η συναρτησιακή εξάρτηση είναι τετριμμένη
    /// </summary>
    public bool IsTrivial
    {
        get { return isTrivial; }
        set { isTrivial = value; }
    }

    /// <summary>
    /// Προσδιορίζει αν η συναρτησιακή εξάρτηση αποκλείεται
    /// </summary>
    public bool Excluded
    {
        get { return excluded; }
        set { excluded = value; }
    }
}
```

Key.cs

```
using System.Collections.Generic;
using System.Linq;

public class Key //η κλάση Key δημιουργεί κλειδιά που περιλαμβάνουν γνωρίσματα
(μονά ή και συνδυασμούς)
{
    //ο πίνακας KeyAttrs περιέχει τα γνωρίσματα που περιλαμβάνει το κλειδί
    private List<Attr> keyAttrs = new List<Attr>();

    /// <summary>
    /// Προσθέτει ένα γνώρισμα τύπου Attr στο κλειδί
    /// </summary>
    public void AddToKey(Attr attr)
    {
        keyAttrs.Add(attr);
    }

    /// <summary>
    /// Προσθέτει μια λίστα από γνωρίσματα στο κλειδί
    /// </summary>
    public void AddToKey(List<Attr> list)
    {
```

```
        //ελέγχεται αν τα κλειδιά υπάρχουν ήδη στον πίνακα, κι αν όχι,
προστίθενται κανονικά
        foreach (Attr attr in list)
            if (!keyAttrs.Contains(attr))
                AddToKey(attr);
    }

    /// <summary>
    /// Επιστρέφει πίνακα με τα γνωρίσματα που περιλαμβάνονται στο κλειδί Key
    /// </summary>
    public List<Attr> GetAttrs()
    {
        return keyAttrs;
    }

    /// <summary>
    /// Ελέγχει αν το κλειδί περιλαμβάνεται ήδη σαν υποσύνολο στον πίνακα
KeyList
    /// </summary>
    /// <param name="keyList">Ο πίνακας με τα κλειδιά που ήδη έχουν
καταχωρηθεί</param>
    public bool KeyExists(List<Key> keyList)
    {
        //ελέγχεται με την τομή αν τα γνωρίσματα ενός ήδη καταχωρημένου
κλειδιού της keyList περιλαμβάνονται στο νέο κλειδί
        foreach (Key keysearch in keyList)
            if (keysearch.GetAttrs().Intersect(keyAttrs).Count() >=
keysearch.GetAttrs().Count) return true;
        return false;
    }

    /// <summary>
    /// Επιστρέφει τα γνωρίσματα του κλειδιού
    /// </summary>
    public override string ToString()
    {
        //στον τοπικό πίνακα names προστίθενται τα ονόματα των γνωρισμάτων που
μετέχουν στο κλειδί, ταξινομούνται αλφαβητικά και επιστρέφονται ως ενιαίο,
μορφοποιημένο string
        List<string> names = new List<string>();
        foreach (Attr attr in keyAttrs)
            names.Add(attr.Name);
        names.Sort();
        string str = string.Join(", ", names);
        str = "{" + str + "}";
        return str;
    }
}
```

Program.cs

```
using System;
using System.Windows.Forms;

static class Program
{
    /// <summary>
    /// Το σημείο εκκίνησης του προγράμματος
    /// </summary>
    [STAThread]
```

```
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new FRMMAIN());
    }
}
```

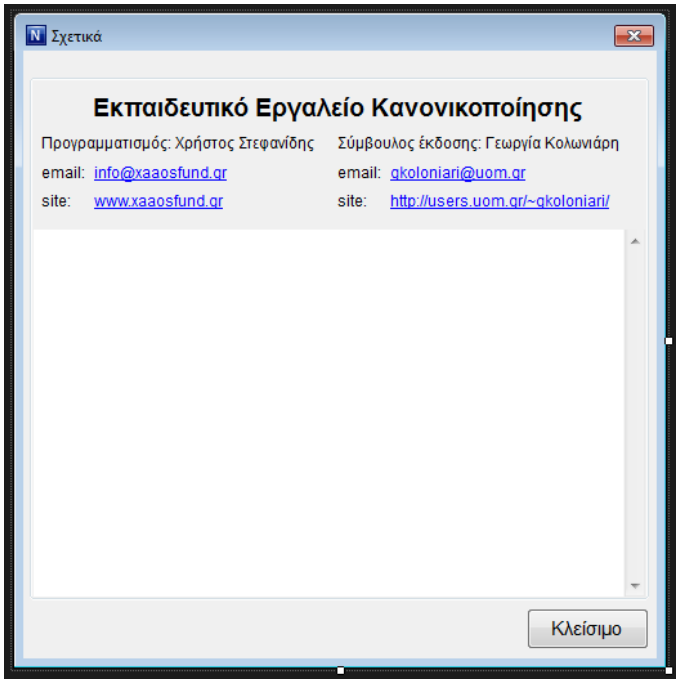
Relation.cs

```
using System.Collections.Generic;

class Relation //η κλάση Relation δημιουργεί πίνακες γνωρισμάτων, που είτε είναι μορφής BCNF, είτε όχι
{
    public static bool isAA; //προσδιορίζει αν η αρίθμηση των πινάκων θα γίνεται με α/α ή με το σύστημα 1-2
    public static int aa; //προσδιορίζει τον αύξοντα αριθμό του πίνακα
    private bool BCNF; //προσδιορίζει αν ο πίνακας είναι BCNF
    private bool excluded; //προσδιορίζει αν ο πίνακας αποκλείεται από διασπάσεις
    private string name; //το όνομα του πίνακα
    private Key key = new Key(); //το κλειδί του πίνακα
    private List<Attr> attrList = new List<Attr>(); //λίστα με τα γνωρίσματα που περιέχει ο πίνακας

    /// <summary>
    /// Κατασκευαστής του πίνακα χωρίς παραμέτρους
    /// </summary>
    public Relation()
    {

    }

    /// <summary>
    /// Ορίζει το κλειδί του πίνακα
    /// </summary>
    public void SetKey(Key key)
    {
        this.key = key;
    }

    /// <summary>
    /// Επιστρέφει το κλειδί του πίνακα
    /// </summary>
    public Key GetKey()
    {
        return key;
    }

    /// <summary>
    /// Κατασκευαστής του πίνακα με αρχικοποίηση των γνωρισμάτων
    /// </summary>
    /// <param name="attrList">Λίστα με όλα τα γνωρίσματα</param>
    public Relation(List<Attr> attrList)
    {
        //προστίθεται το περιεχόμενο της παραμέτρου στη λίστα attrList του πίνακα BCNF
        foreach (Attr attr in attrList) AttrAdd(attr);
    }
```

```
    /// <summary>
    /// Προστίθεται ένα γνώρισμα στη λίστα attrList του πίνακα
    /// </summary>
    /// <param name="attr">Το γνώρισμα που εισάγεται</param>
    public void AttrAdd(Attr attr)
    {
        //ελέγχεται αν το γνώρισμα υπάρχει ήδη στη λίστα, κι αν όχι, τότε
καταχωρείται
        if (!attrList.Contains(attr)) attrList.Add(attr);
    }

    /// <summary>
    /// Ορίζει αν ο πίνακας είναι BCNF
    /// </summary>
    public bool IsBCNF
    {
        get { return BCNF; }
        set { BCNF = value; }
    }

    /// <summary>
    /// Ορίζει αν ο πίνακας αποκλείεται από διασπάσεις
    /// </summary>
    public bool Excluded
    {
        get { return excluded; }
        set { excluded = value; }
    }

    /// <summary>
    /// Το όνομα του πίνακα
    /// </summary>
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    /// <summary>
    /// Επιστρέφει λίστα με τα γνωρίσματα του πίνακα
    /// </summary>
    public List<Attr> GetList()
    {
        return attrList;
    }

    public override string ToString()
    {
        //στον τοπικό πίνακα names προστίθενται το όνομα του πίνακα, τα
ονόματα των γνωρισμάτων που μετέχουν στον πίνακα, τα κλειδιά τους,
ταξινομούνται αλφαβητικά και επιστρέφονται ως ενιαίο, μορφοποιημένο string
        //το "\u2660" χρησιμοποιείται για την απεικόνιση του κλειδιού
(μπαστουνιού)
        List<string> names = new List<string>();
        foreach (Attr attr in attrList)
        {
            if (key.GetAttrs().Contains(attr))
                names.Add("\u2660" + attr.Name);
```

```
            //names.Add("#" + attr.Name);
            else
                names.Add(attr.Name);
        }

        names.Sort();
        string str = string.Join(", ", names);
        str = name + "{" + str + "}";
        return str;
    }
}
```

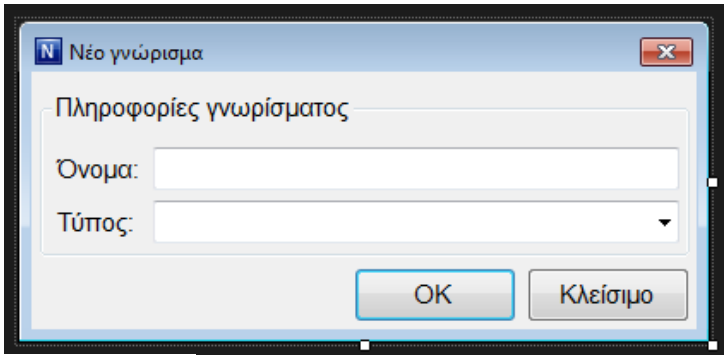
frmAbout.cs




```
using System;
using System.Diagnostics;
using System.Windows.Forms;

public partial class frmAbout : Form
{
    public frmAbout()
    {
        InitializeComponent();
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void tClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void lblSendEmail_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
    {
```

```
        //διαδικασία αποστολής email
        try
        {
            Process.Start("mailto:" + "info@myprograms.eu");
        }
        catch
        {
            MessageBox.Show("Παρουσιάστηκε κάποιο πρόβλημα και δεν μπορεί να
σταλεί email.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void lblURL_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
    {
        //ανοίγει η σελίδα της κανονικοποίησης
        Process.Start("http://www.myprograms.eu/normalization");
    }

    private void frmAbout_Load(object sender, EventArgs e)
    {
        htm.Navigate(AppDomain.CurrentDomain.BaseDirectory +
"MISC\\about.htm");
        htm.Select();
    }

    private void lblSendEmailGeorgia_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
    {
        //διαδικασία αποστολής email
        try
        {
            Process.Start("mailto:" + "gkoloniari@uom.gr");
        }
        catch
        {
            MessageBox.Show("Παρουσιάστηκε κάποιο πρόβλημα και δεν μπορεί να
σταλεί email.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void lblURLGeorgia_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
    {
        Process.Start("http://users.uom.gr/~gkoloniari/");
    }
}
```

frmAttr.cs




```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

public partial class frmAttr : Form
{
    Attr attrEdit; //μεταβλητή που αναφέρεται στο υπό επεξεργασία γνώρισμα
    FRMMAIN frmmain; //μεταβλητή που αναφέρεται στην FRMMAIN

    /// <summary>
    /// Κατασκευαστής της φόρμας νέου γνωρίσματος
    /// </summary>
    public frmAttr(FRMMAIN frmmain)
    {
        InitializeComponent();
        this.frmmain = frmmain;
        this.Text = "Νέο γνώρισμα";
    }

    /// <summary>
    /// Κατασκευαστής της φόρμας επεξεργασίας ενός γνωρίσματος
    /// </summary>
    /// <param name="attrEdit">Το αντικείμενο του υπό επεξεργασία
γνωρίσματος</param>
    public frmAttr(FRMMAIN frmmain, Attr attrEdit)
    {
        InitializeComponent();
        this.frmmain = frmmain;
        this.attrEdit = attrEdit;
        this.Text = "Επεξεργασία γνωρίσματος";
    }

    private void mnuClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void txtName_TextChanged(object sender, EventArgs e)
    {
        //το όνομα δεν μπορεί να είναι κενό
        if (txtName.Text.Trim().Equals(""))
            btnOK.Enabled = false;
```

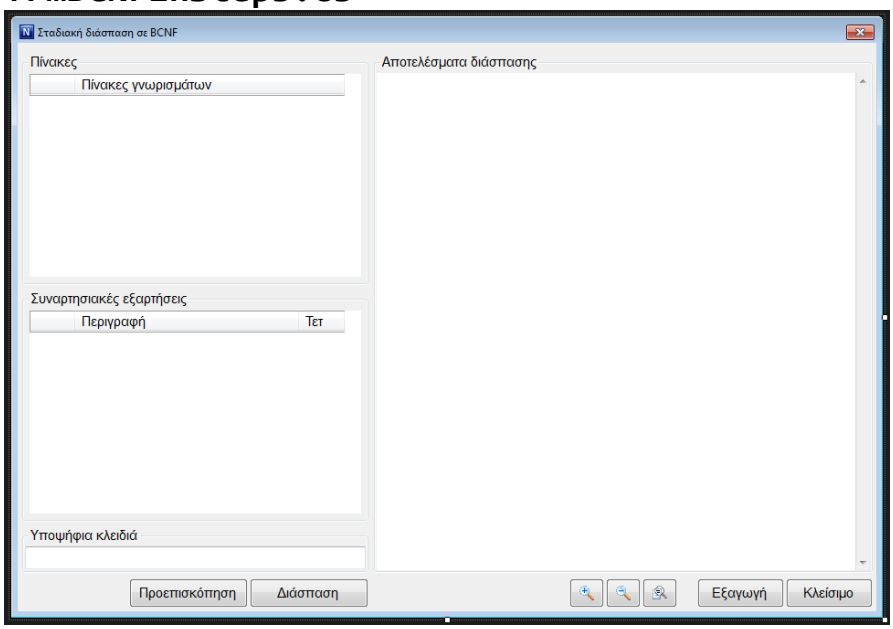
```
        else
            btnOK.Enabled = true;
    }

    private void btnOK_Click(object sender, EventArgs e)
    {
        //απαγορεύεται η χρήση κόμματος στο όνομα
        if (txtName.Text.Contains(","))
        {
            MessageBox.Show("Δεν επιτρέπεται η χρήση κόμματος στο όνομα του
γνωρίσματος.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtName.SelectAll();
            txtName.Select();
            return;
        }

        //ανάλογα με το αν έχουμε να κάνουμε με νέο γνώρισμα ή επεξεργασία
γνωρίσματος, καλείται η σχετική μέθοδος
        if (attrEdit == null) Create();
        else Edit();
    }

    /// <summary>
    /// Επιστρέφει true αν τα c1 και c2 ανήκουν στην ίδια γλώσσα (ελληνικά ή
αγγλικά)
    /// </summary>
    private bool CharsCheck(char c1, char c2)
    {
        //ελληνικά
        if ((c1 >= 'Α' & c1 <= 'Ω') && (c2 >= 'Α' & c2 <= 'Ω')) return true;
        //αγγλικά
        if ((c1 >= 'A' & c1 <= 'Z') && (c2 >= 'A' & c2 <= 'Z')) return true;

        //σε οποιαδήποτε άλλη περίπτωση, επιστρέφεται false
        return false;
    }

    /// <summary>
    /// Δημιουργείται το γνώρισμα
    /// </summary>
    private void Create()
    {
        string sName = txtName.Text.Trim();
        string sType = comType.Text.Trim();
        int x = 0; //το πλήθος των νέων γνωρισμάτων που καταχωρήθηκαν

        //αρχικά ελέγχεται η περίπτωση ο χρήστης να έχει ζητήσει τη δημιουργία
πολλαπλών γνωρισμάτων
        //αυτό γίνεται εισάγοντας το σύμβολο ">" ανάμεσα σε δύο χαρακτήρες,
π.χ. "A>D"
        if (sName.Contains(">"))
        {
            //ελέγχεται πρώτα αν η χρήση του ειδικού συμβόλου ">" έγινε σωστά
            //να είναι δηλαδή στη μέση το ">" και τα γράμματα να είναι
κεφαλαία και να ανήκουν στην ίδια γλώσσα (ελληνικά ή αγγλικά)
            string s = sName;
            if (s.Length == 3 && s[1].ToString() == ">" && CharsCheck(s[0],
s[2]))
            {
```

```
                char c1 = sName[0];
                char c2 = sName[2];

                //ελέγχεται αν το πλήθος των νέων γνωρισμάτων θα αυξήσουν το
σύνολο των γνωρισμάτων πάνω από το μέγιστο επιτρεπόμενο, κάτι που δεν
επιτρέπεται
                int count = 0;
                for (char ccc = c1; ccc <= c2; ccc++)
                    count++;
                if (frmmain.GetAttrList().Count + count > FRMMAIN.MaxAttrNum)
                {
                    MessageBox.Show("Το συνολικό πλήθος των καταχωρημένων
γνωρισμάτων δεν μπορεί να υπερβαίνει τα " + FRMMAIN.MaxAttrNum + ".",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }

                for (char cc = c1; cc <= c2; cc++)
                    if (Char.IsLetter(cc))
                        //στέλνονται στην FRMMAIN τα στοιχεία του γνωρίσματος
ώστε να δημιουργηθεί αντικείμενο με το όνομα cc
                        if (frmmain.AttrCreate(cc.ToString(), sType)) x++;
            }
        }
        //ελέγχεται επίσης ο αριθμός των καταχωρημένων γνωρισμάτων σε
περίπτωση που ο χρήστης θέλει να εισάγει ένα μόνο γνώρισμα
        else if (frmmain.GetAttrList().Count == FRMMAIN.MaxAttrNum)
        {
            MessageBox.Show("Το συνολικό πλήθος των καταχωρημένων γνωρισμάτων
δεν μπορεί να υπερβαίνει τα " + FRMMAIN.MaxAttrNum + ".", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        //αν έγινε εισαγωγή μόνο ενός γνωρίσματος, στέλνονται στην FRMMAIN τα
στοιχεία του γνωρίσματος ώστε να δημιουργηθεί το αντικείμενο με το όνομα sName
        else if (frmmain.AttrCreate(sName, sType)) x++;
        else
        {
            //αν επιστραφεί η ένδειξη false σημαίνει ότι το όνομα του
γνωρίσματος χρησιμοποιείται ήδη για άλλο γνώρισμα
            MessageBox.Show("Το όνομα \" " + sName + "\" χρησιμοποιείται ήδη
για άλλο γνώρισμα.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtName.SelectAll();
            txtName.Select();
            return;
        }

        //ανάλογα με το πλήθος x των γνωρισμάτων που καταχωρήθηκαν,
εμφανίζονται τα σχετικά μηνύματα
        if (x == 0)
        {
            MessageBox.Show("Δεν καταχωρήθηκε κανένα νέο γνώρισμα. Είτε τα
ονόματα χρησιμοποιούνται ήδη για άλλα γνωρίσματα, είτε δεν χρησιμοποιήσατε
σωστά την αυτόματη καταχώρηση.", this.Text, MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }
        if (x == 1)
```

```
            MessageBox.Show("Το νέο γνώρισμα καταχωρήθηκε.", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Information);
        else
            MessageBox.Show("Καταχωρήθηκαν " + x.ToString() + " νέα
γνωρίσματα.", this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);
        txtName.Text = "";
        comType.Text = "";
        txtName.Select();
        frmmain.SaveOn(); //ενεργοποιούνται τα μενού αποθήκευσης της FRMMAIN
    }

    /// <summary>
    /// Επεξεργάζεται το γνώρισμα
    /// </summary>
    private void Edit()
    {
        //το νέο όνομα του γνωρίσματος δεν πρέπει να χρησιμοποιείται για άλλο
γνώρισμα
        if (frmmain.AttrExists(txtName.Text.Trim(), attrEdit))
        {
            MessageBox.Show("Το όνομα \" " + txtName.Text.Trim() + "\"
χρησιμοποιείται ήδη για άλλο γνώρισμα.", this.Text, MessageBoxButtons.OK,
MessageBoxIcon.Error);
            txtName.SelectAll();
            txtName.Select();
        }
        else
        {
            //αποθηκεύονται το όνομα και ο τύπος του γνωρίσματος και η φόρμα
κλείνει
            attrEdit.Name = txtName.Text.Trim();
            attrEdit.Type = comType.Text.Trim();
            this.Close();
        }
    }

    private void tClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void frmAttrNew_Load(object sender, EventArgs e)
    {
        //στο comType φορτώνονται οι διαθέσιμοι τύποι γνωρισμάτων
        List<string> list = new List<string>();

        frmOptions frm = new frmOptions(frmmain);
        list = frm.LoadTypes();

        for (int i = 0; i < list.Count; i++)
            comType.Items.Add(list[i]);
        frm.Dispose();

        //αν το γνώρισμα είναι υπό επεξεργασία, τότε εμφανίζονται τα στοιχεία
του στη φόρμα
        if (attrEdit != null)
        {
            txtName.Text = attrEdit.Name;
            comType.Text = attrEdit.Type;
```

```
            //αν ο τύπος του γνωρίσματος δεν περιλαμβάνεται στους
αποθηκευμένους τύπους, τότε προστίθεται στο comType
            if (!list.Contains(comType.Text)) comType.Items.Add(comType.Text);
        }
    }
}
```

frmBCNFinSteps.cs




```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;

public partial class frmBCNFinSteps : Form
{
    private List<Attr> attrList; //η μεταβλητή attrList αναφέρεται στον
αντίστοιχο πίνακα γνωρισμάτων της FRMMAIN
    private List<FD> FDList; //η μεταβλητή FDList αναφέρεται στο αντίγραφο των
συναρτησιακών εξαρτήσεων της FRMMAIN
    private List<Key> keyList; //τα κλειδιά του πίνακα
    private List<Relation> relList = new List<Relation>(); //λίστα με τους
πίνακες
    private FRMMAIN frmmain; //μεταβλητή αναφοράς στην FRMMAIN
    private string sOut; //το κείμενο που εμφανίζεται στο txtOut
    private frmOut frmout; //μεταβλητή που αναφέρεται στην frmOut

    public frmBCNFinSteps(FRMMAIN frmmain, List<Attr> attrList, List<FD>
FDList)
    {
        InitializeComponent();
        this.attrList = attrList;
        this.FDList = FDList;
        this.frmmain = frmmain;

        frmout = new frmOut("Προεπισκόπηση διάσπασης σε BCNF", true, frmmain);
```

```
    //προσδιορίζονται τα κλειδιά του πίνακα και εμφανίζονται στο txtKeys
    keyList = frmmain.KeysGet(FDList, attrList, false);

    List<string> names = new List<string>();
    foreach (Key key in keyList)
        names.Add(key.ToString());
    names.Sort();
    string str = string.Join("   ", names);
    txtKeys.Text = str;

    //μηδενίζεται η αρίθμηση της Relation
    Relation.aa = 0;
}

private void btnExit_Click(object sender, System.EventArgs e)
{
    this.Close();
}

/// <summary>
/// Αυξάνει το zoom στο txtOut
/// </summary>
private void btnZoomIn_Click(object sender, System.EventArgs e)
{
    //αυξάνεται το μέγεθος της γραμματοσειράς του txtOut
    float i = txtOut.Font.Size;
    i++;
    txtOut.Font = new Font(txtOut.Font.Name, i);
}

/// <summary>
/// Μειώνει το zoom στο txtOut
/// </summary>
private void btnZoomOut_Click(object sender, System.EventArgs e)
{
    //μειώνεται το μέγεθος της γραμματοσειράς του txtOut
    float i = txtOut.Font.Size;
    i--;
    if (i >= 1)
        txtOut.Font = new Font(txtOut.Font.Name, i);
}

/// <summary>
/// Επαναφέρει το zoom του txtOut στο 100%
/// </summary>
private void btnZoomReset_Click(object sender, System.EventArgs e)
{
    txtOut.Font = new Font(txtOut.Font.Name, 11);
}

private void frmBCNFinSteps_KeyDown(object sender, KeyEventArgs e)
{
    //ελέγχεται αν πατήθηκε το Ctrl+A ώστε να επιλεγεί όλο το txtOut
    if (e.Control)
        if (e.KeyCode == Keys.A) txtOut.SelectAll();
}

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

```
        //μορφοποιείται το GridRel
        DataGridView g;
        g = GridRel;

        //περιορίζεται το flickering
        g.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        g.Columns["colRelSel"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        g.Columns["colRelBCNF"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        g.Columns["colRel"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;

        //εμφανίζονται οι επικεφαλίδες
        g.Columns["colRelSel"].HeaderCell.Value = "\u221A";
        g.Columns["colRelBCNF"].HeaderCell.Value = "BCNF";

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        g.Columns["colRelSel"].Width = g.Width * 50 / 363;
        g.Columns["colRelBCNF"].Width = g.Width * 50 / 363;
        g.Columns["colRel"].Width = g.Width * 245 / 363;

        //δημιουργείται ο πρώτος πίνακας Rel που έχει όλα τα γνωρίσματα
        Relation rel = new Relation(attrList);
        rel.Name = "R";
        rel.SetKey(keyList[0]);
        relList.Add(rel);

        //ο νέος πίνακας εμφανίζεται στο GridRel
        g.Rows.Add();
        g.Rows[g.RowCount - 1].Cells["colRelSel"].Value = "";
        g.Rows[g.RowCount - 1].Cells["colRel"].Value = rel.ToString();
//πίνακας γνωρισμάτων
        g.Rows[g.RowCount - 1].Cells["colRelBCNF"].Value = "";
        g.Rows[0].Cells["colRelSel"].Value = "\u221A";
        g.CurrentCell = g[1, g.RowCount - 1];
        g.CurrentCell.Selected = false;

        //μορφοποιείται το GridFD
        g = GridFD;

        //περιορίζεται το flickering
        g.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        g.Columns["colFD"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;
        g.Columns["colFDSel"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        g.Columns["colFDX"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        g.Columns["colTet"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;

        //εμφανίζονται οι επικεφαλίδες
        g.Columns["colFDSel"].HeaderCell.Value = "\u221A";
```

```
        g.Columns["colFDX"].HeaderCell.Value = "X";

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        g.Columns["colFDX"].Width = g.Width * 50 / 363;
        g.Columns["colFDSel"].Width = g.Width * 50 / 363;
        g.Columns["colFD"].Width = g.Width * 195 / 363;
        g.Columns["colTet"].Width = g.Width * 50 / 363;

        //εμφανίζονται οι συναρτησιακές εξαρτήσεις στο GridFD
        g = GridFD;
        foreach (FD fd in FDList)
        {
            g.Rows.Add();
            g.Rows[g.RowCount - 1].Cells["colFDSel"].Value = "";
            g.Rows[g.RowCount - 1].Cells["colFDX"].Value = "";
            g.Rows[g.RowCount - 1].Cells["colFD"].Value = fd.ToString();
//περιγραφή συναρτησιακής εξάρτησης
            if (fd.IsTrivial) g.Rows[g.RowCount - 1].Cells["colTet"].Value =
"\u221A"; //tick για τετριμμένη συναρτησιακή εξάρτηση
        }
        if (g.RowCount == 1) g.Rows[0].Cells["colFDSel"].Value = "\u221A";
        if (g.RowCount > 0)
        {
            g.CurrentCell = g[1, 0];
            g.CurrentCell.Selected = false;
        }
        CheckBCNF();
    }

    private void GridRel_CellDoubleClick(object sender,
DataGridViewCellEventArgs e)
    {
        //εμφανίζεται στην frmOut ο επιλεγμένος πίνακας, καθώς και τα υποψήφια
κλειδιά του αρκεί να έχει επιλεγεί η 1η στήλη και όχι η γραμμή επικεφαλίδων
        if (e.RowIndex == -1 | e.ColumnIndex != 2) return;

        DataGridView g = GridRel;
        frmOut frm = new frmOut("Πίνακας γνωρισμάτων", true, frmmain);
        frm.AddOut(relList[e.RowIndex].ToString() + "\n\n");

        //προσδιορίζονται τα κλειδιά του πίνακα και εμφανίζονται στο frmOut
        frm.AddOut("Υποψήφια κλειδιά:\n\n");
        List<Key> tempokeylist = new List<Key>();
        tempokeylist = frmmain.KeysGet(FDList, relList[e.RowIndex].GetList(),
false);
        foreach (Key key in tempokeylist)
            frm.AddOut(key.ToString() + "\n\n");

        frm.ShowDialog(this);
        frm.Dispose();
    }

    private void GridFD_CellDoubleClick(object sender,
DataGridViewCellEventArgs e)
    {
        //αν έχει επιλεγεί η στήλη των συναρτησιακών εξαρτήσεων, καλείται το
συμβάν click στο κουμπί επεξεργασίας τους, εκτός κι αν έχει επιλεγεί η γραμμή
επικεφαλίδων
```

```
        if (e.RowIndex != -1 && e.ColumnIndex == 2) btnFDEdit_Click(this, e);
    }

    private void txtKeys_DoubleClick(object sender, EventArgs e)
    {
        //ανοίγει στην frmOut το περιεχόμενο του txtKeys
        frmOut frm = new frmOut("Υποψήφια κλειδιά", true, frmmain);
        frm.AddOut(txtKeys.Text);
        frm.ShowDialog(this);
        frm.Dispose();
    }

    /// <summary>
    /// Τοπική μέθοδος που επιλέγει ή αποεπιλέγει το κελί της πρώτης στήλης
του γνωρίσματος, ανάλογα με το κλικ του χρήστη
    /// </summary>
    /// <param name="g">Το επιλεγμένο Grid</param>
    private void GridSel(DataGridView g, DataGridViewCellEventArgs e)
    {
        if (e.RowIndex == -1) return;

        if (e.ColumnIndex == 1 && g == GridFD)
        {
            if (g.Rows[e.RowIndex].Cells[1].Value.ToString() == "X")
                g.Rows[e.RowIndex].Cells[1].Value = "";
            else
                g.Rows[e.RowIndex].Cells[1].Value = "X";
        }

        if (e.ColumnIndex == 0)
        {
            //αποεπιλέγονται όλα τα υπόλοιπα κελιά πλην της e.RowIndex
            for (int i = 0; i < g.RowCount; i++)
                if (i != e.RowIndex)
                    g.Rows[i].Cells[0].Value = "";

            if (g.Rows[e.RowIndex].Cells[0].Value.ToString() == "\u221A")
                g.Rows[e.RowIndex].Cells[0].Value = "";
            else
                g.Rows[e.RowIndex].Cells[0].Value = "\u221A";
        }
    }

    private void GridRel_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        GridSel(GridRel, e);
    }

    private void GridFD_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        GridSel(GridFD, e);
    }

    private void btnPreview_Click(object sender, EventArgs e)
    {
        if (CheckTick()) DecomposeInSteps(true);
    }

    private void btnDecompose_Click(object sender, EventArgs e)
```

```
    {
        if (CheckTick()) DecomposeInSteps(false);
    }

    /// <summary>
    /// Ελέγχει αν έχουν επιλεγεί ένας πίνακας και μια συναρτησιακή εξάρτηση
    /// </summary>
    private bool CheckTick()
    {
        int x = 0;
        for (int i = 0; i < GridRel.RowCount; i++)
            if (GridRel.Rows[i].Cells["colRelSel"].Value.ToString() != "")
            {
                x++;
                break;
            }
        for (int i = 0; i < GridFD.RowCount; i++)
            if (GridFD.Rows[i].Cells["colFDSel"].Value.ToString() != "")
            {
                x++;
                break;
            }

        if (x != 2)
        {
            MessageBox.Show("Πρέπει να επιλέξετε έναν πίνακα και μια
συναρτησιακή εξάρτηση.", this.Text, MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return false;
        }
        else return true;
    }

    /// <summary>
    /// Μέθοδος σταδιακής διάσπασης
    /// </summary>
    /// <param name="isPreview">Προσδιορίζει αν θα είναι προεπισκόπηση ή
κανονική διάσπαση</param>
    private void DecomposeInSteps(bool isPreview)
    {
        int i;
        int iRel = 0;
        int iFD = 0;
        DataGridView g;

        frmout.ResetOut();

        UpdateExcludedFD();

        //προσδιορίζεται ο πίνακας που έχει επιλεγεί
        g = GridRel;
        Relation rel = null;
        for (i = 0; i < g.RowCount; i++)
            if (g.Rows[i].Cells["colRelSel"].Value.ToString() == "\u221A")
            {
                rel = relList[i];
                iRel = i;
            }
```

```
        //αν ο πίνακας που έχει επιλεγεί δεν μπορεί να διασπαστεί περαιτέρω,
βγαίνει σχετικό μήνυμα
        if (!isPreview && rel.Excluded)
        {
            frmOut frm = new frmOut(this.Text, true, frmmain);
            frm.AddOut("Ο πίνακας\n\n" + rel.ToString() + "\n\nέχει
αποκλειστεί από περαιτέρω διασπάσεις.");
            frm.ShowDialog(this);
            frm.Dispose();
            return;
        }

        //προσδιορίζεται η συναρτησιακή εξάρτηση που έχει επιλεγεί
        g = GridFD;
        FD fd = null;
        for (i = 0; i < g.RowCount; i++)
            if (g.Rows[i].Cells["colFDSel"].Value.ToString() == "\u221A")
            {
                fd = FDList[i];
                iFD = i;
            }

        //εξετάζεται αν η συναρτησιακή εξάρτηση είναι τετριμμένη
        if (fd.IsTrivial)
        {
            frmOut frm = new frmOut(this.Text, true, frmmain);
            frm.AddOut("Η συναρτησιακή εξάρτηση\n\n\"" + fd.ToString() +
"\"\n\nείναι τετριμμένη, επομένως δεν χρησιμοποιείται για διάσπαση πινάκων.");
            g.Rows[g.CurrentCell.RowIndex].Cells["colFDX"].Value = "X";
            frm.ShowDialog(this);
            frm.Dispose();
            return;
        }

        //αν το αριστερό σκέλος της συναρτησιακής εξάρτησης περιλαμβάνει
κλειδί, τότε δεν χρησιμοποιείται
        //ο έλεγχος γίνεται με τη βοήθεια της τομής
        foreach (Key key in keyList)
            if (fd.GetLeft().Intersect(key.GetAttrs()).Count() >=
key.GetAttrs().Count)
            {
                frmOut frm = new frmOut(this.Text, true, frmmain);
                frm.AddOut("Η συναρτησιακή εξάρτηση\n\n\"" + fd.ToString() +
"\"\n\nπεριλαμβάνει υποψήφιο κλειδί στο αριστερό σκέλος της, επομένως δεν
παραβιάζει την BCNF μορφή και δεν χρησιμοποιείται για διάσπαση πινάκων.");
                g.Rows[g.CurrentCell.RowIndex].Cells["colFDX"].Value = "X";
                frm.ShowDialog(this);
                frm.Dispose();
                return;
            }

        //επίσης ελέγχεται αν η συναρτησιακή εξάρτηση έχει ως ορίζουσα
υποψήφιο κλειδί του προς διάσπαση πίνακα
        List<Key> tempoRelKey = new List<Key>();
        tempoRelKey = frmmain.KeysGet(FDList, rel.GetList(), false);
        foreach (Key key in tempoRelKey)
            if (fd.GetLeft().Intersect(key.GetAttrs()).Count() >=
key.GetAttrs().Count)
            {
```

```
                frmOut frm = new frmOut(this.Text, true, frmmain);
                frm.AddOut("Η συναρτησιακή εξάρτηση\n\n\"" + fd.ToString() +
"\"\n\nπεριλαμβάνει υποψήφιο κλειδί του πίνακα " + rel.Name + ", επομένως δεν
μπορεί να τον διασπάσει.");
                g.Rows[g.CurrentCell.RowIndex].Cells["colFDX"].Value = "X";
                frm.ShowDialog(this);
                frm.Dispose();
                return;
            }

        //αν η τομή x του συνόλου των γνωρισμάτων της συναρτησιακής εξάρτησης
και των γνωρισμάτων του πίνακα είναι μικρότερη σε αριθμό από το πλήθος των
γνωρισμάτων του πίνακα και ίση με το πλήθος των γνωρισμάτων της συναρτησιακής
εξάρτησης, τότε παραβιάζεται η BCNF μορφή και ο πίνακας μπορεί να διασπαστεί
        int x = fd.GetAll().Intersect(rel.GetList()).Count();
        if (x < rel.GetList().Count && x == fd.GetAll().Count)
        {
            //παρακάτω δημιουργούνται δύο νέοι πίνακες, ο rel1 και ο rel2

            //ο rel1 πίνακας παίρνει τα γνωρίσματα της συναρτησιακής εξάρτησης
            Relation rel1 = new Relation(fd.GetAll());

            //ο rel2 πίνακας παίρνει τα γνωρίσματα από το αριστερό σκέλος της
συναρτησιακής εξάρτησης, συν τα γνωρίσματα του πίνακα που διασπάστηκε, πλην
αυτών που βρίσκονται στο δεξί σκέλος της συναρτησιακής εξάρτησης
            List<Attr> tempo = new List<Attr>();
            tempo.AddRange(fd.GetLeft());
            tempo.AddRange(rel.GetList().Except(fd.GetRight()));
            Relation rel2 = new Relation(tempo);

            //δημιουργούνται δύο κλειδιά, ένα για τον καθένα πίνακα
            Key key1 = new Key();
            Key key2 = new Key();

            //το κλειδί του πρώτου πίνακα είναι η ορίζουσα της συναρτησιακής
εξάρτησης που προκάλεσε την διάσπαση
            key1.AddToKey(fd.GetLeft());
            rel1.SetKey(key1);

            //προσδιορίζουμε το κλειδί του δεύτερου πίνακα (αυτό που δίνει όλα
τα γνωρίσματά του)
            //δημιουργούμε μια τοπική λίστα κλειδιών και ως κλειδί του
δεύτερου πίνακα ορίζεται το πρώτο κλειδί της λίστας
            List<Key> tempoKeyList = new List<Key>();
            tempoKeyList = frmmain.KeysGet(FDList, rel2.GetList(), false);
            key2.AddToKey(tempoKeyList[0].GetAttrs());
            rel2.SetKey(key2);

            //ορίζονται τα ονόματα των δύο νέων πινάκων
            if (Relation.isAA)
            {
                rel1.Name = "R" + ++Relation.aa;
                rel2.Name = "R" + ++Relation.aa;
                if (isPreview) Relation.aa = Relation.aa - 2;
            }
            else
            {
                rel1.Name = rel.Name + "1";
                rel2.Name = rel.Name + "2";
```

```
            }

            //οι δύο νέοι πίνακες προστίθενται στη λίστα
            if (!isPreview)
            {
                relList.Add(rel1);
                relList.Add(rel2);

                //εμφανίζονται τα σχετικά μηνύματα στο txtOut
                AddOut("Με την \"" + fd.ToString() + "\" ο " + rel.Name + "
διασπάται σε:\n\n" + rel1.ToString() + RelBCNF(rel1) + "\n\n" +
rel2.ToString() + RelBCNF(rel2) + "\n\n");
                AddOut("==============================\n\n");

                //μπαίνει Χ στα Grid για τον πίνακα rel και τη συναρτησιακή
εξάρτηση fd
                GridRel.Rows[iRel].Cells["colRelBCNF"].Value = "X";
                GridFD.Rows[iFD].Cells["colFDX"].Value = "X";
                rel.IsBCNF = false;
                rel.Excluded = true;

                //προστίθενται οι δύο νέοι πίνακας στο GridRel
                g = GridRel;
                g.Rows.Add(2);
                g.Rows[g.RowCount - 2].Cells["colRelSel"].Value = "";
                g.Rows[g.RowCount - 2].Cells["colRel"].Value =
rel1.ToString();
                g.Rows[g.RowCount - 1].Cells["colRelSel"].Value = "";
                g.Rows[g.RowCount - 1].Cells["colRel"].Value =
rel2.ToString();

                g.CurrentCell = g[1, g.RowCount - 1];
                g.CurrentCell.Selected = false;
                CheckBCNF();
                MessageBox.Show("Έγινε διάσπαση σε δύο νέους πίνακες, τον " +
rel1.Name + " και τον " + rel2.Name + ".", this.Text, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            else
            {
                frmout.AddOut("Με την \"" + fd.ToString() + "\" ο " +
rel.ToString() + " διασπάται σε:\n\n" + rel1.ToString() + RelBCNF(rel1) +
"\n\n" + rel2.ToString() + RelBCNF(rel2) + "\n\n");
                frmout.AddOut("==============================\n\n");
                frmout.ShowDialog(frmmain);
                frmout.ResetOut();
            }
        }
        else //σε διαφορετική περίπτωση η BCNF δεν παραβιάζεται και
εμφανίζεται σχετικό μήνυμα
        {
            frmOut frm = new frmOut(this.Text, true, frmmain);
            frm.AddOut("Η συναρτησιακή εξάρτηση\n\n\"" + fd.ToString() +
"\"\n\nδεν σχετίζεται με τον πίνακα\n\n" + rel.ToString() + "\n\nκαι επομένως
δεν γίνεται διάσπαση.");
            frm.ShowDialog(this);
            frm.Dispose();
            return;
        }
```

```
    }

    /// <summary>
    /// Προστίθεται κείμενο που θα εμφανιστεί στο txtOut
    /// </summary>
    private void AddOut(string sOut)
    {
        this.sOut = this.sOut + sOut;
        txtOut.Text = this.sOut.Replace("\n", Environment.NewLine);

        //ο κέρσορας πηγαίνει στο τέλος του txtOut
        txtOut.SelectionStart = txtOut.Text.Length - 1;
        txtOut.SelectionLength = 0;
        txtOut.ScrollToCaret();
    }

    private void frmBCNFinSteps_FormClosing(object sender,
FormClosingEventArgs e)
    {
        //αν τυχόν υπάρχει αρχείο που δημιουργήθηκε κατά την εξαγωγή σε HTML,
διαγράφεται
        if (File.Exists("output.htm")) File.Delete("output.htm");
    }

    private void btnExport_Click(object sender, EventArgs e)
    {
        //τα περιεχόμενα του txtOut εξάγονται σε μορφή HTML
        string str = txtOut.Text.Replace(Environment.NewLine, "<br>");

        StreamWriter FileW = new StreamWriter("output.htm", false,
System.Text.Encoding.UTF8);
        FileW.WriteLine(str);
        FileW.Close();

        Process.Start("output.htm");
    }

    private void btnClear_Click(object sender, EventArgs e)
    {
        sOut = "";
        txtOut.Clear();
        txtOut.Focus();
    }

    private void btnFDNew_Click(object sender, EventArgs e)
    {
        //ανοίγει η φόρμα δημιουργίας νέας συναρτησιακής εξάρτησης
        frmFD frm = new frmFD(attrList, this, frmmain); //στέλνουμε στη νέα
φόρμα τη λίστα με τα γνωρίσματα
        frm.ShowDialog(this);
        frm.Dispose();
        GridFD.Select();
    }

    /// <summary>
    /// Μέθοδος ελέγχου και προσθήκης νέας συναρτησιακής εξάρτησης στην FDList
    /// </summary>
    public void FDCreate(FD fd)
    {
```

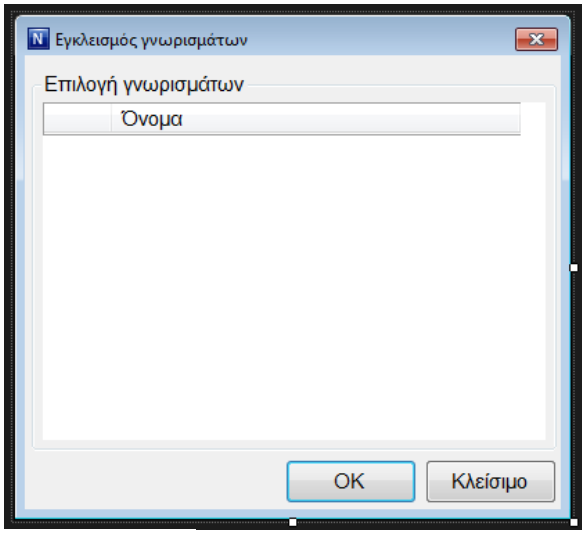
```
        //το νέο αντικείμενο προστίθεται στην FDList
        FDList.Add(fd);

        //η νέα συναρτησιακή εξάρτηση προστίθεται στο GridFD και η εστίαση
μεταφέρεται στη νέα γραμμή
        DataGridView g = GridFD;
        g.Rows.Add();
        g.Rows[g.RowCount - 1].Cells["colFDSel"].Value = "";
        g.Rows[g.RowCount - 1].Cells["colFDX"].Value = "";
        g.Rows[g.RowCount - 1].Cells["colFD"].Value = fd.ToString();
//περιγραφή συναρτησιακής εξάρτησης
        if (fd.IsTrivial)
            g.Rows[g.RowCount - 1].Cells["colTet"].Value = "\u221A"; //tick
για τετριμμένη συναρτησιακή εξάρτηση
        else
            g.Rows[g.RowCount - 1].Cells["colTet"].Value = "";
        g.CurrentCell = g[0, g.RowCount - 1];
        g.CurrentCell.Selected = true;
    }

    private void btnFDEdit_Click(object sender, EventArgs e)
    {
        //ελέγχεται ποια γραμμή είναι επιλεγμένη στο GridFD και ανοίγει η
φόρμα με τα στοιχεία της συναρτησιακής εξάρτησης για επεξεργασία
        DataGridView g = GridFD;

        if (g.RowCount > 0) //απαραίτητη προϋπόθεση είναι να υπάρχει έστω και
μια εγγραφή στο GridFD
        {
            //αναζητείται το αντικείμενο που αντιστοιχεί στην επιλεγμένη
συναρτησιακή εξάρτηση και καλείται η μέθοδος επεξεργασίας της
            FD fd;
            fd = frmmain.FDGetObject(g.CurrentCell.Value.ToString(), FDList);
            frmFD frm = new frmFD(attrList, this, frmmain, fd);
            frm.ShowDialog(this); //ανοίγει η frmFD

            //ανανεώνεται το περιεχόμενο του επιλεγμένου κελιού του GridFD (σε
περίπτωση που ενδεχομένως άλλαξαν τα στοιχεία της συναρτησιακής εξάρτησης)
            g.CurrentCell.Value = fd.ToString();
            if (fd.IsTrivial)
g.Rows[g.CurrentCell.RowIndex].Cells["colTet"].Value = "\u221A";
            else g.Rows[g.CurrentCell.RowIndex].Cells["colTet"].Value = "";

            g.CurrentCell.Selected = true;
            frm.Dispose();
        }
    }

    private void GridFD_KeyDown(object sender, KeyEventArgs e)
    {
        //αν μέσα στο GridFD πατήθηκε το πλήκτρο Delete τότε καλείται το
συμβάν click στο κουμπί διαγραφής της συναρτησιακής εξάρτησης
        if (e.KeyCode == Keys.Delete) btnFDDelete_Click(this, e);
    }

    private void btnFDDelete_Click(object sender, EventArgs e)
    {
        //ελέγχεται ποια γραμμή είναι επιλεγμένη στο GridFD και καλείται η
μέθοδος διαγραφής της συναρτησιακής εξάρτησης
```

```
        if (GridFD.RowCount > 0) //απαραίτητη προϋπόθεση είναι να υπάρχει έστω και μια εγγραφή στο GridFD
        {
            int row = GridFD.CurrentCell.RowIndex;
            //αναζητείται το αντικείμενο που αντιστοιχεί στην επιλεγμένη συναρτησιακή εξάρτηση
            FD fd = frmmain.FDGetObject(GridFD.Rows[row].Cells["colFD"].Value.ToString(), FDList);

            //ερώτημα διαγραφής
            if (MessageBox.Show("Να διαγραφεί η συναρτησιακή εξάρτηση \"" + fd.ToString() + "\";", "Διαγραφή συναρτησιακής εξάρτησης", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No) return;

            //αφαιρείται η επιλεγμένη συναρτησιακή εξάρτηση από την FDList κι από το GridFD
            FDList.Remove(fd);
            GridFD.Rows.RemoveAt(row);
        }
    }

    private void btnShowBCNF_Click(object sender, EventArgs e)
    {
        //εμφανίζονται στο txtOut οι πίνακες που είναι σε BCNF μορφή
        bool oneBCNF = false;
        foreach (Relation rel in relList)
            if (rel.IsBCNF)
                oneBCNF = true;

        frmOut frm = new frmOut(btnShowBCNF.Text, true, frmmain);

        AddOut("==============================\n\n");
        if (oneBCNF)
        {
            AddOut("Οι πίνακες BCNF είναι οι εξής:\n\n");
            frm.AddOut("Οι πίνακες BCNF είναι οι εξής:\n\n");
            foreach (Relation rel in relList)
                if (rel.IsBCNF)
                {
                    AddOut(rel.ToString() + "\n\n");
                    frm.AddOut(rel.ToString() + "\n\n");
                }
        }
        else
        {
            AddOut("Δεν υπάρχουν πίνακες BCNF\n\n");
            frm.AddOut("Δεν υπάρχουν πίνακες BCNF\n\n");
        }
        AddOut("==============================\n\n");

        frm.ShowDialog(this);
        frm.Dispose();
    }

    /// <summary>
    /// Ελέγχεται ποιοι πίνακες είναι BCNF
    /// </summary>
    private void CheckBCNF()
    {
```

```
        for (int i = 0; i < relList.Count; i++)
        {
            if (relList[i].IsBCNF | relList[i].Excluded) continue;
            foreach (FD fd in FDList)
            {
                if (fd.Excluded) continue;
                //αν η τομή x του συνόλου των γνωρισμάτων της συναρτησιακής εξάρτησης και των γνωρισμάτων του πίνακα είναι μικρότερη σε αριθμό από το πλήθος των γνωρισμάτων του πίνακα και ίση με το πλήθος των γνωρισμάτων της συναρτησιακής εξάρτησης, τότε παραβιάζεται η BCNF μορφή και ο πίνακας μπορεί να διασπαστεί
                //σε διαφορετική περίπτωση ο πίνακας είναι BCNF
                int x = fd.GetAll().Intersect(relList[i].GetList()).Count();
                if (x < relList[i].GetList().Count && x == fd.GetAll().Count)
                {
                    relList[i].IsBCNF = false;
                    GridRel.Rows[i].Cells["colRelBCNF"].Value = "";
                    break;
                }
                else
                {
                    relList[i].IsBCNF = true;
                    GridRel.Rows[i].Cells["colRelBCNF"].Value = "BCNF";
                }
            }
        }
    }

    /// <summary>
    /// Ελέγχει αν ο πίνακας Rel είναι BCNF
    /// </summary>
    private string RelBCNF(Relation rel)
    {
        foreach (FD fd in FDList)
        {
            if (fd.Excluded) continue;
            //αν η τομή x του συνόλου των γνωρισμάτων της συναρτησιακής εξάρτησης και των γνωρισμάτων του πίνακα είναι μικρότερη σε αριθμό από το πλήθος των γνωρισμάτων του πίνακα και ίση με το πλήθος των γνωρισμάτων της συναρτησιακής εξάρτησης, τότε παραβιάζεται η BCNF μορφή και ο πίνακας μπορεί να διασπαστεί
            //σε διαφορετική περίπτωση ο πίνακας είναι BCNF
            int x = fd.GetAll().Intersect(rel.GetList()).Count();
            if (x < rel.GetList().Count && x == fd.GetAll().Count)
                return "";
        }
        return "    (BCNF)";
    }

    /// <summary>
    /// Ανανεώνεται η κατάσταση excluded των συναρτησιακών εξαρτήσεων
    /// </summary>
    private void UpdateExcludedFD()
    {
        //αποκλείονται οι τετριμμένες συναρτησιακές εξαρτήσεις
        foreach (FD fd in FDList)
            if (fd.IsTrivial) fd.Excluded = true;
```

```
        //εξαιρούνται οι συναρτησιακές εξαρτήσεις που το αριστερό σκέλος τους
περιλαμβάνει κλειδί
        //ο έλεγχος γίνεται με τη βοήθεια της τομής
        foreach (FD fd in FDList)
            foreach (Key key in keyList)
                if (!fd.Excluded &&
fd.GetLeft().Intersect(key.GetAttrs()).Count() >= key.GetAttrs().Count)
                {
                    fd.Excluded = true;
                    break;
                }
    }
}
```

frmClosure.cs




```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Linq;

public partial class frmClosure : Form
{
    private List<Attr> attrList; //πίνακας attrList όπου αποθηκεύονται τα
γνωρίσματα - αντικείμενα τύπου Attr
    private List<FD> FDList; //πίνακας FDList όπου αποθηκεύονται οι
συναρτησιακές εξαρτήσεις - αντικείμενα τύπου FD
    private FRMMAIN frmmain; //μεταβλητή που αναφέρεται στην FRMMAIN
    private frmOut frmout; //μεταβλητή που αναφέρεται στην frmOut
    private bool anyFDUsed; //προσδιορίζει αν χρησιμοποιήθηκε έστω και μια
συναρτησιακή εξάρτηση κατά τη διαδικασία του εγκλεισμού

    /// <summary>
    /// Κατασκευαστής της frmClosure που υπολογίζει τον εγκλεισμό των
γνωρισμάτων
    /// </summary>
    /// <param name="attrList">Πίνακας των γνωρισμάτων</param>
    /// <param name="FDList">Πίνακας των συναρτησιακών εξαρτήσεων</param>
    /// <param name="isOpen">Προσδιορίζει αν η frmClosure είναι ορατή</param>
    public frmClosure(List<Attr> attrList, List<FD> FDList, FRMMAIN frmmain)
    {
        InitializeComponent();
        this.attrList = attrList;
```

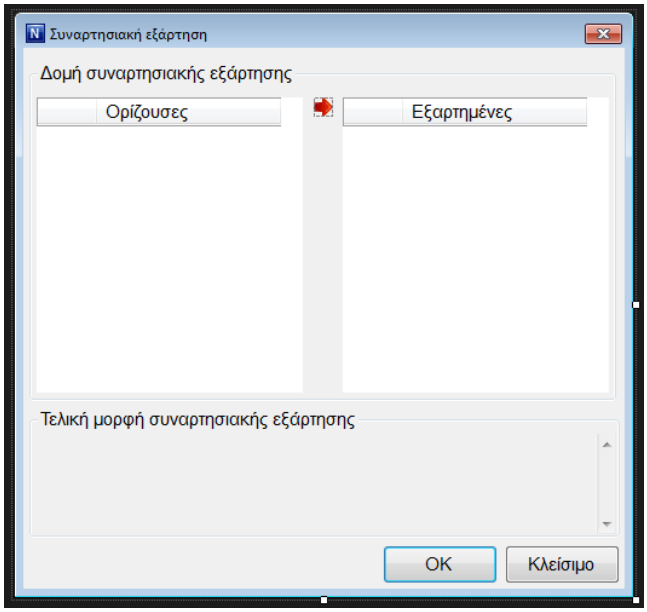
```
        this.FDList = FDList;
        this.frmmain = frmmain;
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void frmClosure_Load(object sender, EventArgs e)
    {
        //μορφοποιείται το GridAttr
        DataGridView g;
        g = GridAttr;

        //περιορίζεται το flickering
        g.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        g.Columns["colSel"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        g.Columns["colName"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        g.Columns["colSel"].Width = g.Width * 50 / 363;
        g.Columns["colName"].Width = g.Width * 295 / 363;

        //εμφανίζονται στο GridAttr όλα τα γνωρίσματα και η εστίαση
μεταφέρεται στην πρώτη γραμμή
        foreach (Attr attr in attrList)
        {
            g.Rows.Add();
            g.Rows[g.RowCount - 1].Cells["colName"].Value = attr.Name;
            g.Rows[g.RowCount - 1].Cells["colSel"].Value = "";

        }
        g.Sort(g.Columns["colName"],
System.ComponentModel.ListSortDirection.Ascending);
        g.CurrentCell = g[1, 0];
        g.CurrentCell.Selected = false;
    }

    /// <summary>
    /// Επιστρέφει τον εγκλεισμό των γνωρισμάτων attrS
    /// </summary>
    /// <param name="attrS">Πίνακας γνωρισμάτων</param>
    public List<Attr> attrClosure(List<Attr> attrS, bool showOut)
    {
        //ο πίνακας closure περιλαμβάνει τον εγκλεισμό των γνωρισμάτων attrS
        List<Attr> closure = new List<Attr>();
        closure.AddRange(attrS);

        //δημιουργείται η frmout που αναφέρεται στην frmOut για να στέλνει
πληροφορίες για τη διαδικασία του εγκλεισμού
        //χρησιμοποιείται το σύμβολο \x207A για την απεικόνιση του + ως εκθέτη
        if (showOut)
        {
```

```
            frmout = new frmOut("Διαδικασία εγκλεισμού", false, null);
            frmout.AddOut("Ο εγκλεισμός ενός συνόλου γνωρισμάτων X
συμβολίζεται ως X\x207A.\n\n");
            Relation rel = new Relation(attrS);
            frmout.AddOut("Έστω ότι X = " + rel.ToString() + "\n\n");
            frmout.AddOut("Ισχύει ότι X\x207A = " + rel.ToString() + ".\n\n");
            frmout.AddOut("Εξετάζουμε τις συναρτησιακές εξαρτήσεις για να
υπολογίσουμε τον συνολικό εγκλεισμό του X\x207A\n\n");
            frmout.AddOut("==============================\n\n");
        }

        //η RepeatLoop χρησιμοποιείται ως σημείο επανεκκίνησης του βρόχου σε
περίπτωση που πρέπει να ελεγχθούν από την αρχή οι συναρτησιακές εξαρτήσεις
        RepeatLoop:

        //ελέγχω μία προς μία τις συναρτησιακές εξαρτήσεις
        foreach (FD fd in FDList)
        {
            //ελέγχεται με την τομή αν τα γνωρίσματα του αριστερού σκέλους της
συναρτησιακής εξάρτησης περιλαμβάνονται στον ως τώρα εγκλεισμό
            if (fd.GetLeft().Intersect(closure).Count() >= fd.GetLeft().Count)
            {
                //αν ναι, τότε προστίθενται τα γνωρίσματα του δεξιού σκέλους
στην συναρτησιακή εξάρτηση, με την προϋπόθεση να μην έχουν ήδη προστεθεί, κι
αν αυτό γίνει τότε η διαδικασία αρχίζει ξανά από την RepeatLoop
                List<Attr> toadd = new List<Attr>();
                foreach (Attr attrR in fd.GetRight())
                {

                    if (!closure.Contains(attrR))
                    {
                        anyFDUsed = true;
                        toadd.Add(attrR);
                    }

                }

                if (toadd.Count > 0)
                {
                    //closure.Add(attrR);
                    closure.AddRange(toadd);

                    if (showOut)
                    {
                        if (toadd.Count == 1)
                            frmout.AddOut("Ισχύει \"" + fd.ToString() + "\",
οπότε μπαίνει στον εγκλεισμό το γνώρισμα {" + toadd[0].Name + "}.\n\n");
                        else
                        {
                            Relation reltoadd = new Relation(toadd);
                            frmout.AddOut("Ισχύει \"" + fd.ToString() + "\",
οπότε μπαίνουν στον εγκλεισμό τα γνωρίσματα " + reltoadd.ToString() +
".\n\n");
                        }

                        Relation rel = new Relation(closure);
                        frmout.AddOut("Έχουμε X\x207A = " + rel.ToString() +
"\n\n");
                        frmout.AddOut("==============================\n\n");
```

```
                    }
                    goto RepeatLoop;
                }
            }
        }
        return closure;
    }

    private void btnOK_Click(object sender, EventArgs e)
    {
        //δημιουργείται πίνακας με τα επιλεγμένα γνωρίσματα
        List<Attr> attrS = new List<Attr>();
        List<Attr> attrS2 = new List<Attr>();
        anyFDUsed = false;

        DataGridView g = GridAttr;
        for (int i = 0; i < g.RowCount; i++)
            if (g.Rows[i].Cells["colSel"].Value.ToString() == "\u221A")
            {

attrS.Add(frmmain.AttrGetObject(g.Rows[i].Cells["colName"].Value.ToString()));

attrS2.Add(frmmain.AttrGetObject(g.Rows[i].Cells["colName"].Value.ToString()))
;
            }

        //αν δεν επιλέχθηκε κανένα γνώρισμα, η διαδικασία ακυρώνεται
        if (attrS.Count == 0)
            MessageBox.Show("Δεν έχετε επιλέξει κανένα γνώρισμα.", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Error);
        //αλλιώς εκτελείται η διαδικασία του εγκλεισμού και εμφανίζονται τα
αποτελέσματά του
        else
        {
            attrS = attrClosure(attrS, true);

            //δημιουργείται τοπικός πίνακας με τα ονόματα των γνωρισμάτων που
περιλαμβάνονται στον εγκλεισμό
            List<string> names = new List<string>();
            foreach (Attr attr in attrS)
                names.Add(attr.Name);
            names.Sort();
            string str = string.Join(", ", names);

            string finalmessage;
            if (anyFDUsed)
                finalmessage = "Με βάση τις συναρτησιακές εξαρτήσεις, ο
εγκλεισμός X\x207A είναι:\n\n{" + str + "}";
            else
            {
                Relation rel = new Relation(attrS2);
                finalmessage = "Καμία συναρτησιακή εξάρτηση δεν έχει ως
ορίζουσα το " + rel.ToString() + ".\n\nΟ εγκλεισμός X\x207A είναι:\n\n{" + str
+ "}";
            }

            //εξετάζεται αν το αποτέλεσμα του εγκλεισμού καθιστά τον
επιλεγμένο συνδυασμό υποψήφιο κλειδί ή υπερκλειδί
            bool superkey = false;
```

```
            if (names.Count == attrList.Count)
            {
                List<Key> tempolist = new List<Key>();
                tempolist = frmmain.KeysGet(FDList, attrList, false);
                Key tempokey = new Key();
                foreach (Key key in tempolist)
                    if (attrS2.Count > key.GetAttrs().Count() &&
key.GetAttrs().Count == key.GetAttrs().Intersect(attrS2).Count())
                    {
                        superkey = true;
                        tempokey = key;
                        break;
                    }
                if (superkey)
                    finalmessage = finalmessage + "\n\nεπομένως το X αποτελεί
υπερκλειδί αφού περιλαμβάνει όλα τα γνωρίσματα και είναι υπερσύνολο του
υποψήφιου κλειδιού " + tempokey.ToString() + " του R.";
                else
                    finalmessage = finalmessage + "\n\nεπομένως το X αποτελεί
υποψήφιο κλειδί αφού περιλαμβάνει όλα τα γνωρίσματα του R.";
            }

            frmout.AddOut(finalmessage);
            frmout.ShowDialog(this);
            frmout.Dispose();
        }
    }

    private void GridAttr_CellClick(object sender, DataGridViewCellEventArgs
e)
    {
        //επιλέγεται ή αποεπιλέγεται το κελί που δέχτηκε το διπλό κλικ, εκτός
κι αν έχει επιλεγεί η γραμμή επικεφαλίδων ή η στήλη 1
        DataGridView g = GridAttr;
        if (e.RowIndex != -1 & e.ColumnIndex != 1)
            if (g.Rows[e.RowIndex].Cells["colSel"].Value.ToString() ==
"\u221A")
                g.Rows[e.RowIndex].Cells["colSel"].Value = "";
            else
                g.Rows[e.RowIndex].Cells["colSel"].Value = "\u221A";

        //αν επιλέχθηκε η 0 γραμμή και η 0 στήλη, επιλέγονται ή αποεπιλέγονται
όλα τα κελιά, ανάλογα με την επιλογή της 1ης γραμμής
        int i;
        if (e.RowIndex == -1 & e.ColumnIndex == 0)
        {
            if (g.Rows[0].Cells["colSel"].Value.ToString() == "\u221A")
                for (i = 0; i < g.RowCount; i++)
                    g.Rows[i].Cells["colSel"].Value = "";
            else
                for (i = 0; i < g.RowCount; i++)
                    g.Rows[i].Cells["colSel"].Value = "\u221A";
        }
    }
}
```

frmFD.cs




```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

public partial class frmFD : Form
{
    private List<Attr> attrList; //πίνακας που αναφέρεται στη λίστα των
γνωρισμάτων
    private FRMMAIN frmmain; //μεταβλητή που αναφέρεται στην FRMMAIN
    private frmBCNFinSteps frmbcnfinsteps; //μεταβλητή που αναφέρεται στην
frmbcnfinsteps
    private FD fdEdit; //μεταβλητή που αναφέρεται στην υπό επεξεργασία
συναρτησιακή εξάρτηση
    private bool toSave; //μεταβλητή που υποδηλώνει αν έγιναν αλλαγές στη
συναρτησιακή εξάρτηση (σε περίπτωση επεξεργασίας) ώστε να ενεργοποιηθούν τα
μενού αποθήκευσης της FRMMAIN

    /// <summary>
    /// Κατασκευαστής νέας συναρτησιακής εξάρτησης για την FRMMAIN
    /// </summary>
    /// <param name="attrList">Πίνακας των γνωρισμάτων</param>
    public frmFD(List<Attr> attrList, FRMMAIN frmmain)
    {
        InitializeComponent();
        this.attrList = attrList;
        this.frmmain = frmmain;
        this.Text = "Νέα συναρτησιακή εξάρτηση";
    }

    /// <summary>
    /// Κατασκευαστής νέας συναρτησιακής εξάρτησης για την frmBCNFinSteps
    /// </summary>
    /// <param name="attrList">Πίνακας των γνωρισμάτων</param>
    public frmFD(List<Attr> attrList, frmBCNFinSteps frmbcnfinsteps, FRMMAIN
frmmain)
    {
        InitializeComponent();
        this.attrList = attrList;
        this.frmbcnfinsteps = frmbcnfinsteps;
```

```
        this.frmmain = frmmain;
        this.Text = "Νέα συναρτησιακή εξάρτηση";
    }

    /// <summary>
    /// Κατασκευαστής επεξεργασίας συναρτησιακής εξάρτησης
    /// </summary>
    /// <param name="attrList">Πίνακας των γνωρισμάτων</param>
    /// <param name="fdEdit">Η υπό επεξεργασία συναρτησιακή εξάρτηση</param>
    public frmFD(List<Attr> attrList, FRMMAIN frmmain, FD fdEdit)
    {
        InitializeComponent();
        this.attrList = attrList;
        this.frmmain = frmmain;
        this.fdEdit = fdEdit;
        this.Text = "Επεξεργασία συναρτησιακής εξάρτησης";
    }

    /// <summary>
    /// Κατασκευαστής επεξεργασίας συναρτησιακής εξάρτησης για την
frmBCNFinSteps
    /// </summary>
    /// <param name="attrList">Πίνακας των γνωρισμάτων</param>
    /// <param name="fdEdit">Η υπό επεξεργασία συναρτησιακή εξάρτηση</param>
    public frmFD(List<Attr> attrList, frmBCNFinSteps frmbcnfinsteps, FRMMAIN
frmmain, FD fdEdit)
    {
        InitializeComponent();
        this.attrList = attrList;
        this.frmmain = frmmain;
        this.frmbcnfinsteps = frmbcnfinsteps;
        this.fdEdit = fdEdit;
        this.Text = "Επεξεργασία συναρτησιακής εξάρτησης";
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    /// <summary>
    /// Ανανεώνει το περιεχόμενο του txtFD
    /// </summary>
    private void Update_txtFD()
    {
        //ελέγχεται αν έχει επιλεγεί έστω και ένα γνώρισμα από το κάθε Grid
        DataGridView gL = GridL;
        DataGridView gR = GridR;
        bool gLok = false;
        bool gRok = false;
        for (int i = 0; i < gL.RowCount; i++)
        {
            if (gL.Rows[i].Cells["colSelL"].Value.ToString() == "\u221A") gLok
= true;
            if (gR.Rows[i].Cells["colSelR"].Value.ToString() == "\u221A") gRok
= true;
        }

        if (gLok & gRok)
```

```
        {
            //δημιουργείται ένα τοπικό αντικείμενο fdtempo
            FD fdtempo = new FD();
            AddAttributes(fdtempo);

            //στο txtFD εμφανίζεται η συναρτησιακή εξάρτηση
            txtFD.Text = fdtempo.ToString();

            //εξετάζεται αν είναι τετριμμένη ώστε να εμφανιστεί σχετικό μήνυμα
στο txtFD
            if (fdtempo.IsTrivial)
                txtFD.Text = txtFD.Text + " (τετριμμένη)";

            btnOK.Enabled = true;
        }
        else //αν δεν υπάρχουν επιλεγμένα γνωρίσματα, τότε απενεργοποιείται το
κουμπί OK
        {
            txtFD.Text = "";
            btnOK.Enabled = false;
        }
    }

    private void btnOK_Click(object sender, EventArgs e)
    {
        //ανάλογα με το αν έχουμε να κάνουμε με νέα συναρτησιακή εξάρτηση ή
επεξεργασία υπάρχουσας, καλείται η σχετική μέθοδος
        if (fdEdit == null)
        {
            if (frmbcnfinsteps == null)
                Create();
            else
                CreateInSteps();
        }
        else
        {
            if (frmbcnfinsteps == null)
                Edit();
            else
                EditInSteps();
        }
    }

    /// <summary>
    /// Δημιουργείται συναρτησιακή εξάρτηση για τη σταδιακή διάσπαση BCNF
    /// </summary>
    private void CreateInSteps()
    {
        //δημιουργείται ένα αντικείμενο τύπου FD το οποίο περιέχει την
συναρτησιακή εξάρτηση
        FD fd = new FD();

        //στο αντικείμενο fd προστίθενται τα γνωρίσματα - αντικείμενα τύπου
Attr
        AddAttributes(fd);

        //το νέο αντικείμενο στέλνεται στην frmbcnfinsteps για να προστεθεί
στην FDList
        frmbcnfinsteps.FDCreate(fd);
```

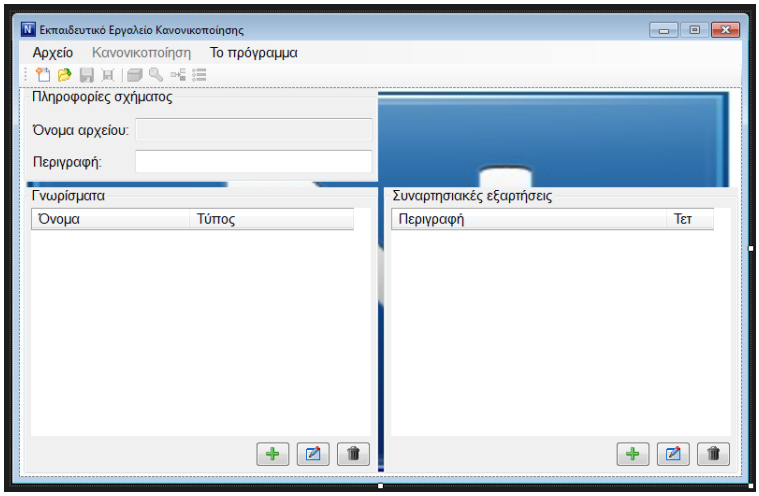
```
        //τα στοιχεία της φόρμας "καθαρίζουν" ώστε ο χρήστης να δημιουργήσει
κι άλλες συναρτησιακές εξαρτήσεις αν θέλει
        MessageBox.Show("Η νέα συναρτησιακή εξάρτηση καταχωρήθηκε.",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);

        //καθαρίζει η στήλη με τις επιλογές των γνωρισμάτων
        DataGridView gL = GridL;
        DataGridView gR = GridR;
        for (int i = 0; i < gL.RowCount; i++)
        {
            gL.Rows[i].Cells["colSelL"].Value = "";
            gR.Rows[i].Cells["colSelR"].Value = "";
        }
        gL.CurrentCell.Selected = false;
        gR.CurrentCell.Selected = false;
        Update_txtFD();
    }

    /// <summary>
    /// Δημιουργείται η συναρτησιακή εξάρτηση
    /// </summary>
    private void Create()
    {
        //δημιουργείται ένα αντικείμενο τύπου FD το οποίο περιέχει την
συναρτησιακή εξάρτηση
        FD fd = new FD();

        //στο αντικείμενο fd προστίθενται τα γνωρίσματα - αντικείμενα τύπου
Attr
        AddAttributes(fd);

        //το νέο αντικείμενο στέλνεται στην frmmain για να προστεθεί στην
FDList
        if (!frmmain.FDCreate(fd)) //αν υπάρχει ήδη αυτή η συναρτησιακή
εξάρτηση, η διαδικασία ακυρώνεται
            MessageBox.Show("Υπάρχει ήδη μια παρόμοια συναρτησιακή εξάρτηση.",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
        else //η προσθήκη της νέας συναρτησιακής εξάρτησης στην FDList
πραγματοποιήθηκε και τα στοιχεία της φόρμας "καθαρίζουν" ώστε ο χρήστης να
δημιουργήσει κι άλλες συναρτησιακές εξαρτήσεις αν θέλει
        {
            MessageBox.Show("Η νέα συναρτησιακή εξάρτηση καταχωρήθηκε.",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);
            frmmain.SaveOn();

            //καθαρίζει η στήλη με τις επιλογές των γνωρισμάτων
            DataGridView gL = GridL;
            DataGridView gR = GridR;
            for (int i = 0; i < gL.RowCount; i++)
            {
                gL.Rows[i].Cells["colSelL"].Value = "";
                gR.Rows[i].Cells["colSelR"].Value = "";
            }
            gL.CurrentCell.Selected = false;
            gR.CurrentCell.Selected = false;
            Update_txtFD();
        }
    }
```

```
    /// <summary>
    /// Επεξεργάζεται η συναρτησιακή εξάρτηση
    /// </summary>
    private void Edit()
    {
        //στο αντικείμενο fdEdit προστίθενται τα γνωρίσματα - αντικείμενα
τύπου Attr
        fdEdit.Reset();
        AddAttributes(fdEdit);

        //ελέγχεται αν υπάρχει παρόμοια συναρτησιακή εξάρτηση
        if (frmmain.FDExists(fdEdit, frmmain.FDGetID(fdEdit)))
        {
            MessageBox.Show("Υπάρχει ήδη μια παρόμοια συναρτησιακή εξάρτηση.",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            if (toSave) frmmain.SaveOn();
            this.Close();
        }
    }

    /// <summary>
    /// Επεξεργάζεται η συναρτησιακή εξάρτηση για τη σταδιακή διάσπαση BCNF
    /// </summary>
    private void EditInSteps()
    {
        //στο αντικείμενο fdEdit προστίθενται τα γνωρίσματα - αντικείμενα
τύπου Attr
        fdEdit.Reset();
        AddAttributes(fdEdit);
        this.Close();
    }

    /// <summary>
    /// Προσθέτει τα επιλεγμένα γνωρίσματα των δύο Grid στο αντικείμενο fd
    /// </summary>
    /// <param name="fd">Αναφορά στο αντικείμενο fd στο οποίο θα προστεθούν τα
γνωρίσματα</param>
    private void AddAttributes(FD fd)
    {
        DataGridView gL = GridL;
        DataGridView gR = GridR;
        //προστίθενται τα επιλεγμένα γνωρίσματα του GridL στο αριστερό σκέλος
και του GridR στο δεξί σκέλος της συναρτησιακής εξάρτησης
        for (int i = 0; i < gL.RowCount; i++)
        {
            if (gL.Rows[i].Cells["colSelL"].Value.ToString() == "\u221A")

fd.AddLeft(frmmain.AttrGetObject(gL.Rows[i].Cells["colL"].Value.ToString()));
            if (gR.Rows[i].Cells["colSelR"].Value.ToString() == "\u221A")

fd.AddRight(frmmain.AttrGetObject(gR.Rows[i].Cells["colR"].Value.ToString()));
        }
    }

    private void frmFD_Load(object sender, EventArgs e)
```

```
    {
        //μορφοποιούνται τα δύο Grid
        DataGridView gL = GridL;
        DataGridView gR = GridR;

        //περιορίζεται το flickering
        gL.DoubleBuffered(true);
        gR.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        gL.Columns["colSelL"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        gL.Columns["colL"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;
        gR.Columns["colSelR"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        gR.Columns["colR"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        gL.Columns["colSelL"].Width = gL.Width * 50 / 220;
        gL.Columns["colL"].Width = gL.Width * 152 / 220;
        gR.Columns["colSelR"].Width = gR.Width * 50 / 220;
        gR.Columns["colR"].Width = gR.Width * 152 / 220;

        //εμφανίζονται στα Grid όλα τα γνωρίσματα
        foreach (Attr attr in attrList)
        {
            gL.Rows.Add();
            gL.Rows[gL.RowCount - 1].Cells["colL"].Value = attr.Name;
            gL.Rows[gL.RowCount - 1].Cells["colSelL"].Value = "";
            if (fdEdit != null && fdEdit.GetLeft().Contains(attr))
gL.Rows[gL.RowCount - 1].Cells["colSelL"].Value = "\u221A";

            gR.Rows.Add();
            gR.Rows[gR.RowCount - 1].Cells["colR"].Value = attr.Name;
            gR.Rows[gR.RowCount - 1].Cells["colSelR"].Value = "";
            if (fdEdit != null && fdEdit.GetRight().Contains(attr))
gR.Rows[gR.RowCount - 1].Cells["colSelR"].Value = "\u221A";
        }
        //τα Grid ταξινομούνται και η εστίαση μεταφέρεται στην πρώτη γραμμή
που έχει επιλεγμένο γνώρισμα
        int i;

        gL.Sort(gL.Columns["colL"],
System.ComponentModel.ListSortDirection.Ascending);
        for (i = 0; i < gL.RowCount; i++)
            if (gL.Rows[i].Cells["colSelL"].Value.ToString() != "")
            {
                gL.FirstDisplayedScrollingRowIndex = i;
                break;
            }
        if (i >= gL.RowCount)
            gL.CurrentCell = gL[1, 0];
        else
            gL.CurrentCell = gL[1, i];
        gL.CurrentCell.Selected = false;
```

```
        gR.Sort(gR.Columns["colR"],
System.ComponentModel.ListSortDirection.Ascending);
        for (i = 0; i < gR.RowCount; i++)
            if (gR.Rows[i].Cells["colSelR"].Value.ToString() != "")
            {
                gR.FirstDisplayedScrollingRowIndex = i;
                break;
            }
        if (i >= gR.RowCount)
            gR.CurrentCell = gR[1, 0];
        else
            gR.CurrentCell = gR[1, i];
        gR.CurrentCell.Selected = false;

        Update_txtFD();
    }

    private void GridL_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        GridSel(GridL, e);
    }

    private void GridR_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        GridSel(GridR, e);
    }

    /// <summary>
    /// Τοπική μέθοδος που επιλέγει ή αποεπιλέγει το κελί της πρώτης στήλης
του γνωρίσματος, ανάλογα με το κλικ του χρήστη
    /// </summary>
    /// <param name="g">Το επιλεγμένο Grid</param>
    private void GridSel(DataGridView g, DataGridViewCellEventArgs e)
    {
        if (e.RowIndex != -1 & e.ColumnIndex != 1)
            if (g.Rows[e.RowIndex].Cells[0].Value.ToString() == "\u221A")
                g.Rows[e.RowIndex].Cells[0].Value = "";
            else
                g.Rows[e.RowIndex].Cells[0].Value = "\u221A";

        //αν επιλέχθηκε η 0 γραμμή και η 0 στήλη, επιλέγονται ή αποεπιλέγονται
όλα τα κελιά, ανάλογα με την επιλογή της 1ης γραμμής
        int i;
        if (e.RowIndex == -1 & e.ColumnIndex == 0)
        {
            if (g.Rows[0].Cells[0].Value.ToString() == "\u221A")
                for (i = 0; i < g.RowCount; i++)
                    g.Rows[i].Cells[0].Value = "";
            else
                for (i = 0; i < g.RowCount; i++)
                    g.Rows[i].Cells[0].Value = "\u221A";
        }
        if (fdEdit != null) toSave = true;
        Update_txtFD();
    }
}
```

FRMMAIN.cs




```
using System;
using System.IO;
using System.Windows.Forms;
using System.Diagnostics;
using System.Reflection;
using System.Collections.Generic;
using System.Linq;

public partial class FRMMAIN : Form
{
    string localApp = ""; //η διαδρομή όπου βρίσκεται ο φάκελος του
προγράμματος με αρχεία όπως οι ρυθμίσεις
    private string scFilename = ""; //η ονομασία του αρχείου όπου αποθηκεύεται
το σχήμα
    private string path = ""; //η πλήρης διαδρομή του αρχείου
    private string scDescription = ""; //η περιγραφή του σχήματος
    private List<Attr> attrList = new List<Attr>(); //δημιουργείται ο πίνακας
attrList όπου αποθηκεύονται τα γνωρίσματα - αντικείμενα τύπου Attr
    private List<FD> FDList = new List<FD>(); //δημιουργείται ο πίνακας FDList
όπου αποθηκεύονται οι συναρτησιακές εξαρτήσεις - αντικείμενα τύπου FD
    private List<byte> bin = new List<byte>(); //βοηθητικός πίνακας για το
δυαδικό σύστημα, μετρά το πλήθος του ψηφίου 1
    private static bool isRunning;
    public static int MaxAttrNum = 20; //το μέγιστο πλήθος των επιτρεπόμενων
γνωρισμάτων

    /// <summary>
    /// Κατασκευαστής της FRMMAIN
    /// </summary>
    public FRMMAIN()
    {
        InitializeComponent();

        if (isRunning) return;

        //ελέγχεται αν υπάρχει ο φάκελος xfdNormalization, κι αν όχι,
δημιουργείται εδώ
        localApp =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicat
ionData), "xfdNormalization");
        if (!Directory.Exists(localApp))
        {
```

```
            Directory.CreateDirectory(localApp);
            SaveOptions();
        }

        if (!File.Exists(localApp + "\\types.txt"))
        {
            frmOptions frm = new frmOptions(this);
            frm.SaveTypes();
            frm.Dispose();
        }

        //φορτώνονται οι ρυθμίσεις του χρήστη
        LoadOptions();

        //καταχωρούνται στον πίνακα bin το πλήθος των ψηφίων 1 στο δυαδικό
σύστημα για κάθε αριθμό
        //η μεταβλητή str αναπαριστά δυαδικό αριθμό, ξεκινώντας από το 0 μέχρι
το μέγιστο δυνητικό συνδυασμό, που είναι το 2 υψωμένο στο μέγιστο επιτρεπόμενο
πλήθος των γνωρισμάτων
        string str;
        int kk = (int)Math.Pow(2, MaxAttrNum);
        for (int i = 0; i <= kk; i++)
        {
            //ο αριθμός i μετατρέπεται σε δυαδικό
            str = Convert.ToString(i, 2);

            //μετριέται το πλήθος των ψηφίων 1 στον δυαδικό αριθμό str και
καταχωρείται στον πίνακα bin
            bin.Add((byte)str.Replace("0", "").Length);
        }
    }

    /// <summary>
    /// Αποθήκευση επιλογών του χρήστη
    /// </summary>
    public void SaveOptions()
    {
        bool isFirstTime;
        //αν το αρχείο δεν υπάρχει, σημαίνει ότι είναι η πρώτη φορά που
γίνεται αποθήκευση
        if (File.Exists(localApp + "\\options.txt"))
            isFirstTime = false;
        else
            isFirstTime = true;

        StreamWriter FileW = new StreamWriter(localApp + "\\options.txt",
false, System.Text.Encoding.UTF8);

        if (isFirstTime) //αν είναι η πρώτη φορά που δημιουργείται το αρχείο,
αποθηκεύονται συγκεκριμένες εγγραφές
        {
            FileW.WriteLine(false);
            FileW.WriteLine(10);
        }
        else //αλλιώς αποθηκεύονται οι επιλογές του χρήστη
        {
            FileW.WriteLine(Relation.isAA);
            FileW.WriteLine(frmOut.iFont);
        }
```

```
            FileW.Close();
        }

        /// <summary>
        /// Φόρτωση επιλογών του χρήστη
        /// </summary>
        private void LoadOptions()
        {
            try
            {
                StreamReader FileR = new StreamReader(localApp + "\\options.txt",
System.Text.Encoding.UTF8);
                Relation.isAA = bool.Parse(FileR.ReadLine());
                frmOut.iFont = int.Parse(FileR.ReadLine());
                FileR.Close();
            }
            catch { }
            if (frmOut.iFont == 0) frmOut.iFont = 11;
        }

        private void mnuExit_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void FRMMAIN_FormClosing(object sender, FormClosingEventArgs e)
        {
            //ερώτημα αν θα κλείσει το πρόγραμμα αφού πρώτα ελεγχθεί αν υπάρχουν
μη αποθηκευμένα δεδομένα
            if (!QSaveOK())
            {
                e.Cancel = true;
                return;
            }
            if (MessageBox.Show("Θέλετε να κλείσει το πρόγραμμα;", this.Text,
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No) e.Cancel
= true;
        }

        private void mnuNew_Click(object sender, EventArgs e)
        {
            //γίνεται έλεγχος αν υπάρχουν μη αποθηκευμένα δεδομένα
            if (QSaveOK())
            {
                //ανοίγει η frmNew για τη δημιουργία του νέου σχήματος
                string newname = "";
                frmNew frm = new frmNew();
                frm.ShowDialog(this);
                newname = frm.GetFilename();
                frm.Dispose();

                //αν ο χρήστης εισήγαγε έγκυρη τιμή στην frmNew, τότε
δημιουργείται ένα νέο σχήμα με αυτή την ονομασία
                if (newname != "")
                {
                    scLoad(newname, true);

                    //ενεργοποιούνται τα μενού αποθήκευσης
                    SaveOn();
```

```
                MessageBox.Show("Το νέο σχήμα δημιουργήθηκε.", "Νέο σχήμα",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }

    private void mnuAbout_Click(object sender, EventArgs e)
    {
        //ανοίγει η φόρμα "Σχετικά"
        frmAbout frm = new frmAbout();
        frm.ShowDialog(this);
        frm.Dispose();
    }

    /// <summary>
    /// Μέθοδος φόρτωσης σχήματος
    /// </summary>
    /// <param name="fullpath">Πλήρης διαδρομή του αρχείου</param>
    /// <param name="isNew">Αν είναι true τότε δημιουργείται ένα νέο, κενό
σχήμα</param>
    public void scLoad(string fullpath, bool isNew)
    {
        //απενεργοποιούνται τα μενού αποθήκευσης
        SaveOff();

        //αν η τιμή isNew είναι true, τότε δημιουργείται ένα νέο, κενό σχήμα,
αλλιώς φορτώνεται ένα αποθηκευμένο
        if (isNew)
        {
            scFilename = fullpath;
            path = "";
            scDescription = "";
            resetList();
        }
        else
        {
            //από τη διαδρομή εξάγεται το όνομα του αρχείου και αποθηκεύεται
στη μεταβλητή filename
            scFilename = Path.GetFileNameWithoutExtension(fullpath);
            path = fullpath;

            //φορτώνεται το αρχείο path
            StreamReader FileR = new StreamReader(path,
System.Text.Encoding.UTF8);

            //η 1η εγγραφή πρέπει να φέρει την ένδειξη NOR
            if (FileR.ReadLine() != "NOR")
            {
                MessageBox.Show("Μη έγκυρη μορφή αρχείου σχήματος.",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Error);
                FileR.Close();
                return;
            }

            //διαβάζεται ο αριθμός έκδοσης του προγράμματος
            //χρησιμοποιείται για να ελέγχουμε συμβατότητα προγράμματος με
αρχεία που δημιουργήθηκαν με παλιότερες εκδόσεις
            FileR.ReadLine();
```

```
            //διαβάζεται η περιγραφή του σχήματος
            scDescription = FileR.ReadLine();

            //διαβάζεται το πλήθος των γνωρισμάτων
            int iSuma;
            iSuma = Int32.Parse(FileR.ReadLine());

            //διαβάζονται ένα προς ένα τα γνωρίσματα (όνομα και τύπος)
            //για κάθε ένα από αυτά, δημιουργείται ένα αντικείμενο τύπου Attr
            resetList();
            string sName, sType;
            for (int i = 0; i < iSuma; i++)
            {
                sName = FileR.ReadLine();
                sType = FileR.ReadLine();
                AttrCreate(sName, sType);
            }

            //διαβάζεται το πλήθος των συναρτησιακών εξαρτήσεων και για κάθε
ένα από αυτά δημιουργείται ένα αντικείμενο FD
            iSuma = Int32.Parse(FileR.ReadLine());

            for (int x = 0; x < iSuma; x++)
            {
                FD fd = new FD();

                //διαβάζεται η ένδειξη αν η συναρτησιακή εξάρτηση είναι
τετριμμένη
                fd.IsTrivial = bool.Parse(FileR.ReadLine());

                //προσδιορίζεται το πλήθος των γνωρισμάτων που υπάρχουν στα
δύο σκέλη της συναρτησιακής εξάρτησης
                int left;
                int right;
                left = Int32.Parse(FileR.ReadLine());
                right = Int32.Parse(FileR.ReadLine());

                //προσδιορίζεται το αντικείμενο attr που αντιστοιχεί στη
μεταβλητή str και προστίθεται στο αριστερό ή το δεξί σκέλος της συναρτησιακής
εξάρτησης
                Attr attrread;
                string str;
                int y;
                for (y = 0; y < left; y++)
                {
                    str = FileR.ReadLine();
                    attrread = AttrGetObject(str);
                    fd.AddLeft(attrread);
                }
                for (y = 0; y < right; y++)
                {
                    str = FileR.ReadLine();
                    attrread = AttrGetObject(str);
                    fd.AddRight(attrread);
                }
                FDCreate(fd);
            }
```

```
            FileR.Close();

            //ταξινομούνται τα GridAttr και GridFD και κρύβεται η εστίαση
            DataGridView g;
            g = GridAttr;
            if (g.RowCount > 0)
            {
                g.Sort(g.Columns["colName"],
System.ComponentModel.ListSortDirection.Ascending);
                g.CurrentCell = g["colName", 0];
                g.CurrentCell.Selected = false;
            }

            g = GridFD;
            if (g.RowCount > 0)
            {
                g.CurrentCell = g["colFD", 0];
                g.CurrentCell.Selected = false;
            }
        }
        //στο GroupBoxInfo εμφανίζονται οι πληροφορίες του σχήματος που άνοιξε
και ενεργοποιούνται συγκεκριμένα στοιχεία της FRMMAIN
        txtFilename.Text = scFilename;
        txtDescription.Text = scDescription;
        picN.Visible = false;
        GroupBoxInfo.Visible = true;
        GroupBoxAttr.Visible = true;
        GroupBoxFD.Visible = true;
        mnuSaveAs.Enabled = true;
        tSaveAs.Enabled = true;
        mnuNormalization.Enabled = true;
        tClosure.Enabled = true;
        tKeys.Enabled = true;
        tBCNF.Enabled = true;
        tBCNFinSteps.Enabled = true;
    }

    /// <summary>
    /// Καθαρίζει την attrList και τον πίνακα με τα γνωρίσματα
    /// </summary>
    private void resetList()
    {
        GridAttr.Rows.Clear();
        attrList.Clear();
        GridFD.Rows.Clear();
        FDList.Clear();
    }

    /// <summary>
    /// Μέθοδος αποθήκευσης σχήματος
    /// </summary>
    public void scSave()
    {
        int i;

        //εξάγεται το όνομα του αρχείου από την επιλεγμένη διαδρομή
αποθήκευσης και εμφανίζεται στο txtFilename
        scFilename = Path.GetFileNameWithoutExtension(path);
        txtFilename.Text = scFilename;
```

```
            scDescription = txtDescription.Text.Trim();

            //δημιουργείται το αρχείο path
            StreamWriter FileW = new StreamWriter(path, false,
System.Text.Encoding.UTF8);

            //1η εγγραφή η ένδειξη NOR
            FileW.WriteLine("NOR");

            //2η εγγραφή ο αριθμός έκδοσης του προγράμματος

FileW.WriteLine(Assembly.GetExecutingAssembly().GetName().Version.Minor);

            //3η εγγραφή η περιγραφή του σχήματος
            FileW.WriteLine(scDescription);

            //εγγράφεται το πλήθος των γνωρισμάτων
            FileW.WriteLine(attrList.Count);

            //εγγράφονται ένα προς ένα τα γνωρίσματα (όνομα και τύπος)
            for (i = 0; i < attrList.Count; i++)
            {
                FileW.WriteLine(attrList[i].Name);
                FileW.WriteLine(attrList[i].Type);
            }

            //εγγράφεται το πλήθος των συναρτησιακών εξαρτήσεων με τα περιεχόμενά
τους
            FileW.WriteLine(FDList.Count);

            for (i = 0; i < FDList.Count; i++)
            {
                FileW.WriteLine(FDList[i].IsTrivial);

                List<Attr> left = FDList[i].GetLeft();
                List<Attr> right = FDList[i].GetRight();

                FileW.WriteLine(left.Count);
                FileW.WriteLine(right.Count);

                int x;
                for (x = 0; x < left.Count; x++)
                {
                    FileW.WriteLine(left[x].Name);
                }
                for (x = 0; x < right.Count; x++)
                {
                    FileW.WriteLine(right[x].Name);
                }
            }

            FileW.Close();

            //απενεργοποιούνται τα μενού αποθήκευσης
            SaveOff();
        }

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

```
        //αν άλλαξε το περιεχόμενο του txtDescription, ενεργοποιούνται τα
μενού αποθήκευσης
        if (txtDescription.Text.Trim() != scDescription) SaveOn();
    }

    private void mnuLoad_Click(object sender, EventArgs e)
    {
        //γίνεται έλεγχος αν υπάρχουν μη αποθηκευμένα δεδομένα
        if (QSaveOK())
        {
            //ανοίγει το παράθυρο επιλογής αρχείου
            if (openFile.ShowDialog(this) == DialogResult.OK &&
openFile.FileName != "") scLoad(openFile.FileName, false);
            openFile.Dispose();
        }
    }

    /// <summary>
    /// Ενεργοποιούνται τα μενού αποθήκευσης
    /// </summary>
    public void SaveOn()
    {
        mnuSave.Enabled = true;
        mnuSaveAs.Enabled = true;
        tSave.Enabled = true;
        tSaveAs.Enabled = true;
    }

    /// <summary>
    /// Απενεργοποιούνται τα μενού αποθήκευσης
    /// </summary>
    public void SaveOff()
    {
        mnuSave.Enabled = false;
        tSave.Enabled = false;
    }

    /// <summary>
    /// Επιστρέφει true αν υπάρχουν μη αποθηκευμένα δεδομένα πριν τη
δημιουργία ή το φόρτωμα σχήματος ή την έξοδο από το πρόγραμμα
    /// Αν επιστραφεί η τιμή true, η διαδικασία που κάλεσε την παρούσα μέθοδο
συνεχίζεται κανονικά, ενώ με false ακυρώνεται
    /// </summary>
    private bool QSaveOK()
    {
        //αν τα μενού αποθήκευσης είναι απενεργοποιημένα, δεν υπάρχει κανένα
πρόβλημα και απλώς επιστρέφεται η ένδειξη true
        if (!mnuSave.Enabled) return true;

        //στη διαφορετική περίπτωση, βγαίνει ερώτημα αποθήκευσης
        DialogResult result = MessageBox.Show("Να αποθηκευθούν τα δεδομένα του
σχήματος \"" + scFilename + "\";", "Αποθήκευση σχήματος",
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button1);

        if (result == DialogResult.No) //αν επιλέχθηκε το No, δεν γίνεται
αποθήκευση, ωστόσο η διαδικασία που κάλεσε την μέθοδο συνεχίζεται κανονικά
            return true;
```

```
        else if (result == DialogResult.Cancel) //αν επιλέχθηκε το Cancel, δεν γίνεται αποθήκευση, κι επιπλέον σταματάει και η διαδικασία που κάλεσε την μέθοδο
            return false;
        else if (result == DialogResult.Yes) //αν επιλέχθηκε το Yes, γίνεται και αποθήκευση και συνεχίζεται κανονικά η διαδικασία που κάλεσε την μέθοδο
        {
            if (path == "") //αν δεν υπάρχει διαδρομή αρχείου σημαίνει ότι δεν έχει αποθηκευτεί το σχήμα ακόμα ούτε μια φορά, οπότε θα κληθεί η SaveAs, αλλιώς θα κληθεί η Save
                mnuSaveAs_Click(this, EventArgs.Empty);
            else
                mnuSave_Click(this, EventArgs.Empty);
            return true;
        }
        else
            return false;
    }

    private void mnuSaveAs_Click(object sender, EventArgs e)
    {
        //ανοίγει το παράθυρο επιλογής αρχείου αποθήκευσης
        //αν δεν υπάρχει διαδρομή αρχείου, τότε εμφανίζεται μόνο το όνομα του αρχείου
        if (path != "")
        {
            saveFile.FileName = path;
        }
        else
        {
            saveFile.FileName = scFilename;
        }

        //αν το αποτέλεσμα του saveFile dialog είναι OK, τότε αποθηκεύεται η επιλεγμένη διαδρομή στη μεταβλητή path και γίνεται η αποθήκευση
        if (saveFile.ShowDialog(this) == DialogResult.OK)
        {
            path = saveFile.FileName;
            scSave();
        }
        saveFile.Dispose();
    }

    private void mnuSave_Click(object sender, EventArgs e)
    {
        //αν δεν υπάρχει διαδρομή αρχείου καλείται η SaveAs, αλλιώς γίνεται κανονικά αποθήκευση
        if (path == "")
            mnuSaveAs_Click(this, EventArgs.Empty);
        else
            scSave();
    }

    private void FRMMAIN_Load(object sender, EventArgs e)
    {
        //μορφοποιείται το GridAttr
        DataGridView g;
        g = GridAttr;
```

```
        //περιορίζεται το flickering
        g.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        g.Columns["colName"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;
        g.Columns["colType"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        g.Columns["colName"].Width = g.Width * 170 / 363;
        g.Columns["colType"].Width = g.Width * 175 / 363;

        //μορφοποιείται το GridFD
        g = GridFD;

        //περιορίζεται το flickering
        g.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        g.Columns["colFD"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;
        g.Columns["colTet"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleCenter;

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        g.Columns["colFD"].Width = g.Width * 295 / 363;
        g.Columns["colTet"].Width = g.Width * 50 / 363;

        isRunning = true;
    }

    private void tNew_Click(object sender, EventArgs e) //ανοίγει νέο σχήμα
    {
        mnuNew_Click(this, e);
    }

    private void tLoad_Click(object sender, EventArgs e) //φορτώνει σχήμα
    {
        mnuLoad_Click(this, e);
    }

    private void tSave_Click(object sender, EventArgs e) //αποθηκεύει σχήμα
    {
        mnuSave_Click(this, e);
    }

    private void tSaveAs_Click(object sender, EventArgs e) //αποθηκεύει ως
σχήμα
    {
        mnuSaveAs_Click(this, e);
    }

    /// <summary>
    /// Επιστρέφει true αν το γνώρισμα attr συμμετέχει σε κάποια από τις
συναρτησιακές εξαρτήσεις
    /// </summary>
```

```
    /// <param name="attr">Το γνώρισμα που ελέγχουμε</param>
    public bool IsAttrInFD(Attr attr)
    {
        foreach (FD fd in FDList)
            if (fd.CheckAttr(attr)) return true;
        return false;
    }

    /// <summary>
    /// Επιστρέφει το αντικείμενο της συναρτησιακής εξάρτησης που αναζητάμε
    /// </summary>
    /// <param name="name">Το όνομα της συναρτησιακής εξάρτησης</param>
    public FD FDGetObject(string name, List<FD> fdlist)
    {
        for (int i = 0; i < fdlist.Count; i++)
            if (fdlist[i].ToString() == name) return fdlist[i];
        //αν δεν βρεθεί, επιστρέφεται η τιμή null
        return null;
    }

    /// <summary>
    /// Επιστρέφει το αντικείμενο της συναρτησιακής εξάρτησης που αναζητάμε
    /// </summary>
    /// <param name="id">Ο αύξοντα αριθμός της συναρτησιακής εξάρτησης</param>
    public FD FDGetObject(int aa)
    {
        for (int i = 0; i < FDList.Count; i++)
            if (FDList[i].GetAA() == aa) return FDList[i];
        //αν δεν βρεθεί, επιστρέφεται η τιμή null
        return null;
    }

    /// <summary>
    /// Επιστρέφει τον αύξοντα αριθμό της συναρτησιακής εξάρτησης. Overload
μέθοδος.
    /// </summary>
    /// <param name="name">Η τελική μορφή της συναρτησιακής εξάρτησης που
αναζητάμε</param>
    public int FDGetID(string name)
    {
        //σαρώνεται η FDList μέχρι να βρεθεί η συναρτησιακή εξάρτηση που
αναζητάμε
        for (int i = 0; i < FDList.Count; i++)
            if (FDList[i].ToString() == name) return i;
        //αν δεν βρεθεί, επιστρέφεται η τιμή -1
        return -1;
    }

    /// <summary>
    /// Επιστρέφει τον αύξοντα αριθμό της συναρτησιακής εξάρτησης. Overload
μέθοδος.
    /// </summary>
    /// <param name="fd">Το αντικείμενο - συναρτησιακή εξάρτηση που
αναζητάμε</param>
    public int FDGetID(FD fd)
    {
        //σαρώνεται η FDList μέχρι να βρεθεί η συναρτησιακή εξάρτηση που
αναζητάμε
        for (int i = 0; i < FDList.Count; i++)
```

```
            if (FDList[i] == fd) return i;
        //αν δεν βρεθεί, επιστρέφεται η τιμή -1
        return -1;
    }

    /// <summary>
    /// Ελέγχει και επιστρέφει true αν υπάρχει παρόμοια συναρτησιακή εξάρτηση
    /// </summary>
    /// <param name="fd">Το αντικείμενο της συναρτησιακής εξάρτησης που
ελέγχουμε</param>
    /// <param name="id">Ο αύξοντας αριθμός της υπό επεξεργασία συναρτησιακής
εξάρτησης</param>
    public bool FDExists(FD fd, int id)
    {
        for (int i = 0; i < FDList.Count; i++)
            if (FDList[i].ToString() == fd.ToString() && i != id) return true;
//βρέθηκε παρόμοια συναρτησιακή εξάρτηση
        return false; //δεν υπάρχει παρόμοια συναρτησιακή εξάρτηση
    }

    /// <summary>
    /// Μέθοδος ελέγχου και προσθήκης νέας συναρτησιακής εξάρτησης στην FDList
    /// </summary>
    /// <param name="fd">Το αντικείμενο συναρτησιακής εξάρτησης fd θα ελεγχθεί
αν υπάρχει κάποιο άλλο παρόμοιο με αυτό πριν καταχωρηθεί.</param>
    public bool FDCreate(FD fd)
    {
        //ελέγχεται αν η νέα συναρτησιακή εξάρτηση είναι παρόμοια με μια άλλη,
κι αν ναι, επιστρέφεται η ένδειξη false
        if (FDExists(fd, -1)) return false;

        //το νέο αντικείμενο προστίθεται στην FDList
        FDList.Add(fd);

        //η νέα συναρτησιακή εξάρτηση προστίθεται στο GridFD και η εστίαση
μεταφέρεται στη νέα γραμμή
        DataGridView g = GridFD;
        g.Rows.Add();
        g.Rows[g.RowCount - 1].Cells["colFD"].Value = fd.ToString();
//περιγραφή συναρτησιακής εξάρτησης
        g.Rows[g.RowCount - 1].Cells["colID"].Value = fd.GetAA(); //κωδικός
αριθμός της συναρτησιακής εξάρτησης
        if (fd.IsTrivial) g.Rows[g.RowCount - 1].Cells["colTet"].Value =
"\u221A"; //tick για τετριμμένη συναρτησιακή εξάρτηση
        g.CurrentCell = g[0, g.RowCount - 1];
        g.CurrentCell.Selected = true;

        //επιστρέφεται η ένδειξη true
        return true;
    }

    /// <summary>
    /// Μέθοδος δημιουργίας νέου γνωρίσματος και προσθήκης του στην attrList.
Επιστρέφει false αν το όνομα χρησιμοποιείται ήδη για άλλο αντικείμενο
    /// </summary>
    /// <param name="name">Ονομασία του γνωρίσματος</param>
    /// <param name="type">Τύπος του γνωρίσματος</param>
    public bool AttrCreate(string name, string type)
    {
```

```
        //ελέγχεται αν το όνομα του νέου γνωρίσματος χρησιμοποιείται ήδη, κι
αν ναι, επιστρέφεται η ένδειξη false
        if (AttrExists(name, null)) return false;

        //δημιουργείται αντικείμενο τύπου Attr και προστίθεται στην attrList
        Attr attr = new Attr(name, type);
        attrList.Add(attr);

        //τα στοιχεία του νέου γνωρίσματος προστίθενται στο GridAttr και η
εστίαση μεταφέρεται στη νέα γραμμή
        DataGridView g = GridAttr;
        g.Rows.Add();
        g.Rows[g.RowCount - 1].Cells["colName"].Value = name; //όνομα
γνωρίσματος
        g.Rows[g.RowCount - 1].Cells["colType"].Value = type; //τύπος
γνωρίσματος
        g.CurrentCell = g[1, g.RowCount - 1];
        g.CurrentCell.Selected = true;

        //επιστρέφεται η ένδειξη true
        return true;
    }

    /// <summary>
    /// Ελέγχει και επιστρέφει true αν το όνομα name χρησιμοποιείται ήδη για
άλλο γνώρισμα
    /// </summary>
    /// <param name="name">Η ονομασία του γνωρίσματος που ελέγχουμε</param>
    /// <param name="attr">Η αναφορά στο αντικείμενο του γνωρίσματος</param>
    public bool AttrExists(string name, Attr attr)
    {
        for (int i = 0; i < attrList.Count; i++)
            if (attrList[i].Name == name && attr != attrList[i]) return true;
//το όνομα χρησιμοποιείται ήδη
        return false; //το όνομα δεν χρησιμοποιείται
    }

    private void btnAttrNew_Click(object sender, EventArgs e)
    {
        //ανοίγει η φόρμα δημιουργίας γνωρισμάτων
        frmAttr frm = new frmAttr(this);
        frm.ShowDialog(this);
        frm.Dispose();
        GridAttr.Select();
    }

    private void btnAttrEdit_Click(object sender, EventArgs e)
    {
        //ελέγχεται ποια γραμμή είναι επιλεγμένη στο GridAttr και ανοίγει η
φόρμα με τα στοιχεία του γνωρίσματος για επεξεργασία
        if (GridAttr.RowCount > 0) //απαραίτητη προϋπόθεση είναι να υπάρχει
έστω και μια εγγραφή στο GridAttr
        {
            int row = GridAttr.CurrentCell.RowIndex;
            //αναζητείται ο αύξοντας αριθμός του επιλεγμένου γνωρίσματος
            int aa =
AttrGetID(GridAttr.Rows[row].Cells["colName"].Value.ToString());
            //καλείται η μέθοδος επεξεργασίας του γνωρίσματος
            AttrEdit(aa, row);
```

```
            GridAttr.CurrentCell.Selected = true;
        }
    }

    private void btnAttrDelete_Click(object sender, EventArgs e)
    {
        //ελέγχεται ποια γραμμή είναι επιλεγμένη στο GridAttr και καλείται η
μέθοδος διαγραφής του γνωρίσματος
        if (GridAttr.RowCount > 0) //απαραίτητη προϋπόθεση είναι να υπάρχει
έστω και μια εγγραφή στο GridAttr
        {
            int row = GridAttr.CurrentCell.RowIndex;
            //αναζητείται ο αύξοντας αριθμός του επιλεγμένου γνωρίσματος
            int aa =
AttrGetID(GridAttr.Rows[row].Cells["colName"].Value.ToString());
            //καλείται η μέθοδος διαγραφής του γνωρίσματος
            AttrDelete(aa, row);
        }
    }

    /// <summary>
    /// Μέθοδος διαγραφής του γνωρίσματος
    /// </summary>
    /// <param name="id">Ο αύξοντας αριθμός του γνωρίσματος</param>
    /// <param name="row">Η γραμμή του GridAttr στην οποία εμφανίζεται το
γνώρισμα</param>
    private void AttrDelete(int id, int row)
    {
        //ελέγχεται αν το γνώρισμα συμμετέχει σε κάποια συναρτησιακή εξάρτηση,
κι αν ναι, η διαδικασία διαγραφής ακυρώνεται
        if (IsAttrInFD(attrList[id]))
        {
            MessageBox.Show("Το γνώρισμα \"" + attrList[id].Name + "\"
συμμετέχει σε συναρτησιακή εξάρτηση και επομένως δεν μπορεί να διαγραφεί.",
"Διαγραφή γνωρίσματος", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        //ερώτημα διαγραφής
        if (MessageBox.Show("Να διαγραφεί το γνώρισμα \"" + attrList[id].Name
+ "\";", "Διαγραφή γνωρίσματος", MessageBoxButtons.YesNo,
MessageBoxIcon.Question) == DialogResult.No) return;

        //αφαιρείται το επιλεγμένο γνώρισμα από την attrList κι από το
GridAttr
        attrList.RemoveAt(id);
        GridAttr.Rows.RemoveAt(row);

        //ενεργοποιούνται τα μενού αποθήκευσης
        SaveOn();
    }

    /// <summary>
    /// Μέθοδος διαγραφής της συναρτησιακής εξάρτησης
    /// </summary>
    /// <param name="id">Ο αύξοντας αριθμός της συναρτησιακής
εξάρτησης</param>
    /// <param name="row">Η γραμμή του GridFD στην οποία εμφανίζεται η
συναρτησιακή εξάρτηση</param>
```

```
    private void FDDelete(int id, int row)
    {
        //ερώτημα διαγραφής
        if (MessageBox.Show("Να διαγραφεί η συναρτησιακή εξάρτηση \"" +
FDList[id].ToString() + "\";", "Διαγραφή συναρτησιακής εξάρτησης",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No) return;

        //αφαιρείται η επιλεγμένη συναρτησιακή εξάρτηση από την FDList κι από
το GridFD
        FDList.RemoveAt(id);
        GridFD.Rows.RemoveAt(row);

        //ενεργοποιούνται τα μενού αποθήκευσης
        SaveOn();
    }

    private void GridAttr_CellDoubleClick(object sender,
DataGridViewCellEventArgs e)
    {
        //καλείται το συμβάν click στο κουμπί επεξεργασίας του γνωρίσματος
        //εκτός κι αν έχει επιλεγεί η γραμμή επικεφαλίδων
        if (e.RowIndex != -1)
            btnAttrEdit_Click(this, e);
    }

    /// <summary>
    /// Μέθοδος επεξεργασίας των στοιχείων του γνωρίσματος
    /// </summary>
    /// <param name="id">Ο αύξοντας αριθμός του γνωρίσματος</param>
    /// <param name="row">Η γραμμή του GridAttr στην οποία εμφανίζεται το
γνώρισμα</param>
    private void AttrEdit(int id, int row)
    {
        //πριν γίνει επεξεργασία του γνωρίσματος, κρατιούνται τα παλιά του
στοιχεία για να γίνει μετά έλεγχος αν έγιναν ή όχι αλλαγές
        string name;
        string type;
        name = attrList[id].Name;
        type = attrList[id].Type;

        //ανοίγει η frmAttr ώστε να γίνει η επεξεργασία των στοιχείων
        //και στέλνουμε σε αυτή το αντικείμενο του γνωρίσματος
        frmAttr frm = new frmAttr(this, attrList[id]);
        frm.ShowDialog(this);

        //ελέγχεται αν έγινε κάποια αλλαγή στα στοιχεία του γνωρίσματος
        if (attrList[id].Name != name | attrList[id].Type != type)
        {
            UpdateGridFD();

            //διορθώνονται τα στοιχεία στο GridAttr
            GridAttr.Rows[row].Cells["colName"].Value = attrList[id].Name;
            GridAttr.Rows[row].Cells["colType"].Value = attrList[id].Type;

            //ενεργοποιούνται τα μενού αποθήκευσης
            SaveOn();
        }
        frm.Dispose();
    }
```

```
    /// <summary>
    /// Τροποποιεί τα στοιχεία του GridFD μετά από μια επεξεργασία γνωρίσματος
    /// </summary>
    private void UpdateGridFD()
    {
        DataGridView g = GridFD;
        FD fd;

        if (g.RowCount == 0) return;

        //ελέγχονται μία προς μία οι γραμμές του GridFD και ανανεώνονται οι
συναρτησιακές εξαρτήσεις που περιέχουν με τη βοήθεια του κωδικού αριθμού τους
        int i;
        for (i = 0; i < g.RowCount; i++)
        {
            fd = FDGetObject((int)g.Rows[i].Cells["colID"].Value);
            g.Rows[i].Cells["colFD"].Value = fd.ToString();
        }
    }

    /// <summary>
    /// Επιστρέφει τον αύξοντα αριθμό του γνωρίσματος
    /// </summary>
    public int AttrGetID(string name)
    {
        //σαρώνεται η attrList μέχρι να βρεθεί η ονομασία του γνωρίσματος που
αναζητάμε
        for (int i = 0; i < attrList.Count; i++)
            if (attrList[i].Name == name) return i;
        //αν δεν βρεθεί το όνομα, επιστρέφεται η τιμή -1
        return -1;
    }

    /// <summary>
    /// Επιστρέφει το αντικείμενο Attr στο οποίο αντιστοιχεί το όνομα name.
    /// </summary>
    public Attr AttrGetObject(string name)
    {
        //διασχίζεται η λίστα με τα αντικείμενα ώσπου να βρεθεί αυτό με την
ονομασία name, οπότε κι επιστρέφεται
        for (int i = 0; i < attrList.Count; i++)
            if (attrList[i].Name == name) return attrList[i];
        return null;
    }

    private void GridAttr_KeyDown(object sender, KeyEventArgs e)
    {
        //αν μέσα στο GridAttr πατήθηκε το πλήκτρο Delete τότε καλείται το
συμβάν click στο κουμπί διαγραφής του γνωρίσματος
        if (e.KeyCode == Keys.Delete) btnAttrDelete_Click(this, e);
    }

    private void mnuOptions_Click(object sender, EventArgs e) //ανοίγει η
φόρμα με τις ρυθμίσεις του προγράμματος
    {
        //ανοίγει η φόρμα με τις ρυθμίσεις του προγράμματος
        frmOptions frm = new frmOptions(this);
        frm.ShowDialog(this);
```

```
            frm.Dispose();
        }

        private void btnFDNew_Click(object sender, EventArgs e)
        {
            //ελέγχεται αν υπάρχουν καταχωρημένα 2 τουλάχιστον γνωρίσματα
            if (GridAttr.RowCount < 2)
                MessageBox.Show("Δεν μπορείτε να δημιουργήσετε συναρτησιακή
εξάρτηση έχοντας λιγότερα από 2 γνωρίσματα.", "Νέα συναρτησιακή εξάρτηση",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            else
            {
                //ανοίγει η φόρμα δημιουργίας νέας συναρτησιακής εξάρτησης
                frmFD frm = new frmFD(attrList, this); //στέλνουμε στη νέα φόρμα
τη λίστα με τα γνωρίσματα
                frm.ShowDialog(this);
                frm.Dispose();
                GridFD.Select();
            }
        }

        private void btnFDEdit_Click(object sender, EventArgs e)
        {
            //ελέγχεται ποια γραμμή είναι επιλεγμένη στο GridFD και ανοίγει η
φόρμα με τα στοιχεία της συναρτησιακής εξάρτησης για επεξεργασία
            DataGridView g = GridFD;

            if (g.RowCount > 0) //απαραίτητη προϋπόθεση είναι να υπάρχει έστω και
μια εγγραφή στο GridFD
            {
                //αναζητείται το αντικείμενο που αντιστοιχεί στην επιλεγμένη
συναρτησιακή εξάρτηση και καλείται η μέθοδος επεξεργασίας της
                FD fd;
                fd = FDGetObject(g.CurrentCell.Value.ToString(), FDList);
                frmFD frm = new frmFD(attrList, this, fd);
                frm.ShowDialog(this); //ανοίγει η frmFD

                //ανανεώνεται το περιεχόμενο του επιλεγμένου κελιού του GridFD (σε
περίπτωση που ενδεχομένως άλλαξαν τα στοιχεία της συναρτησιακής εξάρτησης)
                g.CurrentCell.Value = fd.ToString();
                if (fd.IsTrivial)
g.Rows[g.CurrentCell.RowIndex].Cells["colTet"].Value = "\u221A";
                else g.Rows[g.CurrentCell.RowIndex].Cells["colTet"].Value = "";

                g.CurrentCell.Selected = true;
                frm.Dispose();
            }
        }

        private void btnFDDelete_Click(object sender, EventArgs e)
        {
            //ελέγχεται ποια γραμμή είναι επιλεγμένη στο GridFD και καλείται η
μέθοδος διαγραφής της συναρτησιακής εξάρτησης
            if (GridFD.RowCount > 0) //απαραίτητη προϋπόθεση είναι να υπάρχει έστω
και μια εγγραφή στο GridFD
            {
                int row = GridFD.CurrentCell.RowIndex;
                //αναζητείται ο αύξοντας αριθμός της επιλεγμένης συναρτησιακής
εξάρτησης
```

```
            int aa =
FDGetID(GridFD.Rows[row].Cells["colFD"].Value.ToString());
            //καλείται η μέθοδος διαγραφής της συναρτησιακής εξάρτησης
            FDDelete(aa, row);
        }
    }

    private void GridFD_SelectionChanged(object sender, EventArgs e)
    {
        //αποεπιλέγεται η επιλογή στο GridAttr
        GridAttr.CurrentCell.Selected = false;
    }

    private void GridAttr_CurrentCellChanged(object sender, EventArgs e)
    {
        GridChangeSelection(GridAttr, GridFD);
    }

    private void GridFD_CurrentCellChanged(object sender, EventArgs e)
    {
        GridChangeSelection(GridFD, GridAttr);
    }

    private void GridAttr_Enter(object sender, EventArgs e)
    {
        GridChangeSelection(GridAttr, GridFD);
    }

    private void GridFD_Enter(object sender, EventArgs e)
    {
        GridChangeSelection(GridFD, GridAttr);
    }

    /// <summary>
    /// Επιχειρείται να αλλάξει η ένδειξη επιλογής των δύο Grid
    /// </summary>
    /// <param name="g">Αυτό το Grid θα είναι επιλεγμένο</param>
    /// <param name="gnot">Αυτό το Grid δεν θα είναι επιλεγμένο</param>
    private void GridChangeSelection(DataGridView g, DataGridView gnot)
    {
        try
        {
            g.CurrentCell.Selected = true;
        }
        catch { }

        try
        {
            gnot.CurrentCell.Selected = false;
        }
        catch { }
    }

    private void GridFD_KeyDown(object sender, KeyEventArgs e)
    {
        //αν μέσα στο GridFD πατήθηκε το πλήκτρο Delete τότε καλείται το
συμβάν click στο κουμπί διαγραφής της συναρτησιακής εξάρτησης
        if (e.KeyCode == Keys.Delete) btnFDDelete_Click(this, e);
    }
```

```
    private void GridAttr_Leave(object sender, EventArgs e)
    {
        try
        {
            GridAttr.CurrentCell.Selected = false;
        }
        catch { }
    }

    private void GridFD_Leave(object sender, EventArgs e)
    {
        try
        {
            GridFD.CurrentCell.Selected = false;
        }
        catch { }
    }

    private void GridFD_CellDoubleClick(object sender,
DataGridViewCellEventArgs e)
    {
        //καλείται το συμβάν click στο κουμπί επεξεργασίας του γνωρίσματος,
εκτός κι αν έχει επιλεγεί η γραμμή επικεφαλίδων
        if (e.RowIndex != -1) btnFDEdit_Click(this, e);
    }

    private void mnuClosure_Click(object sender, EventArgs e)
    {
        //ανοίγει η φόρμα του εγκλεισμού και στέλνονται σε αυτή οι πίνακες με
τα γνωρίσματα και τις συναρτησιακές εξαρτήσεις, αρκεί όμως να υπάρχει έστω και
μία συναρτησιακή εξάρτηση
        if (FDList.Count == 0)
            MessageBox.Show("Για να προσδιορίσετε τον εγκλεισμό ενός
γνωρίσματος πρέπει να έχετε δημιουργήσει τουλάχιστον μία συναρτησιακή
εξάρτηση.", "Εγκλεισμός", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            frmClosure frm = new frmClosure(attrList, FDList, this);
            frm.ShowDialog(this);
            frm.Dispose();
        }
    }

    private void tClosure_Click(object sender, EventArgs e)
    {
        mnuClosure_Click(this, e);
    }

    private void mnuKeys_Click(object sender, EventArgs e)
    {
        //προσδιορίζονται τα κλειδιά του σχήματος, αρκεί όμως να υπάρχει έστω
και μία συναρτησιακή εξάρτηση
        if (FDList.Count == 0)
            MessageBox.Show("Για να προσδιορίσετε τα κλειδιά πρέπει να έχετε
δημιουργήσει τουλάχιστον μία συναρτησιακή εξάρτηση.", "Κλειδιά",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
            KeysGet(FDList, attrList, true);
```

```
    }

    /// <summary>
    /// Επιστρέφει τα υποψήφια κλειδιά του σχήματος
    /// </summary>
    /// <param name="newAttrList">Η λίστα των γνωρισμάτων μέσα στα οποία θα αναζητηθούν τα κλειδιά</param>
    /// <param name="showOut">Προσδιορίζει αν τα αποτελέσματα θα φανούν στην οθόνη</param>
    public List<Key> KeysGet(List<FD> FDList, List<Attr> newAttrList, bool showOut)
    {
        //δημιουργείται πίνακας με τα υποψήφια κλειδιά του σχήματος
        List<Key> keyList = new List<Key>();

        //δημιουργείται η frmout που αναφέρεται στην frmOut για να στέλνει πληροφορίες για τη διαδικασία της εύρεσης κλειδιών
        frmOut frmout = new frmOut("Διαδικασία εύρεσης κλειδιών", true, this);
        frmout.AddOut("Αν ο εγκλεισμός ενός γνωρίσματος ή συνδυασμός αυτών περιλαμβάνει το σύνολο όλων των γνωρισμάτων του σχήματος, τότε το γνώρισμα αυτό, ή ο συνδυασμός των γνωρισμάτων, αποτελεί υποψήφιο κλειδί.\n\n");
        frmout.AddOut("Υποψήφια κλειδιά:\n\n");

        //ελέγχονται ένα προς ένα όλα τα γνωρίσματα για το κατά πόσον μπορεί να συμμετέχουν σε κλειδί
        //αν ένα γνώρισμα δεν βρίσκεται σε κανένα αριστερό σκέλος κάποιας FD τότε αποκλείεται από τον έλεγχο
        foreach (Attr attr in newAttrList)
        {
            attr.Exclude = true;
            foreach (FD fd in FDList)
                if (fd.GetLeft().Contains(attr))
                    attr.Exclude = false;

            //για να οριστικοποιηθεί ο αποκλεισμός του γνωρίσματος, πρέπει να μην συμμετέχει όμως και σε κανένα δεξί
            //κι αυτό γιατί τότε απλά προσδιορίζει τον εαυτό του, οπότε μπορεί να βρεθεί σε σύνθετο κλειδί
            //έτσι δημιουργείται μια τοπική λίστα με τα δεξιά γνωρίσματα όλων των FD
            if (attr.Exclude)
            {
                List<Attr> attrall = new List<Attr>();
                foreach (FD fd in FDList)
                    attrall.AddRange(fd.GetRight());

                //αν το γνώρισμα attr υπάρχει στην τοπική λίστα με τα γνωρίσματα των δεξιών σκελών, ο αποκλεισμός του γνωρίσματος αίρεται
                if (attrall.Contains(attr)) attr.Exclude = true;
                else attr.Exclude = false;
                attrall = null;
            }
        }

        //ελέγχεται ο εγκλεισμός όλων των συνδυασμών των γνωρισμάτων του σχήματος κι αν αυτός περιλαμβάνει όλα τα γνωρίσματα, τότε έχουμε ένα υποψήφιο κλειδί
        //το πλήθος των γνωρισμάτων είναι n και τα εξετάζουμε σε συνδυασμούς ανά k, με τη βοήθεια της μεθόδου AttrBinarySelection
```

```
        int n, k;
        n = newAttrList.Count;
        k = 1;
        for (k = 1; k < n; k++)
            AttrBinarySelection(FDList, newAttrList, ref keyList, k, frmout,
showOut);

        //αφού ολοκληρώθηκε η διαδικασία, ελέγχεται αν βρέθηκε έστω και ένα
κλειδί, κι αν όχι, επιστρέφεται ως κλειδί του σχήματος ένα νέο κλειδί με όλα
τα γνωρίσματα, αλλιώς μόνο τα κλειδιά που καταχωρήθηκαν στην keyList
        if (keyList.Count == 0)
        {
            Key key = new Key();
            for (int i = 0; i < newAttrList.Count; i++)
                key.AddToKey(newAttrList[i]);
            keyList.Add(key);
            frmout.AddOut(key.ToString() + ".\n\nΕπιλέγεται ως υποψήφιο κλειδί
το σύνολο των γνωρισμάτων του σχήματος, καθώς δεν εντοπίστηκαν ως κλειδιά
μεμονωμένα γνωρίσματα ή συνδυασμοί αυτών.");
        }
        //αν έχει επιλεγεί να ανοίξει η φόρμα με τη διαδικασία εύρεσης των
κλειδιών, τότε ανοίγει η frmOut
        if (showOut)
            frmout.ShowDialog(this);
        frmout.Dispose();
        return keyList;
    }

    /// <summary>
    /// Επιστρέφει πίνακα με όλους τους συνδυασμούς γνωρισμάτων ανά k
    /// </summary>
    /// <param name="k">Το επιθυμητό πλήθος των γνωρισμάτων ανά
συνδυασμό</param>
    private void AttrBinarySelection(List<FD> FDList, List<Attr> newAttrList,
ref List<Key> keyList, int k, frmOut frmout, bool showOut)
    {
        string sBin = "";
        int i;
        int x;

        //δημιουργείται μια καθαρή λίστα γνωρισμάτων, χωρίς αυτά που είναι
excluded
        List<Attr> cleanList = new List<Attr>();
        foreach (Attr attr in newAttrList)
            if (!attr.Exclude) cleanList.Add(attr);

        //αν όλα τα γνωρίσματα είναι excluded ή το πλήθος των συνδυασμών k
είναι μεγαλύτερο του μεγέθους της λίστας, επιστρέφεται κενή λίστα
        if (cleanList.Count == 0 | k > cleanList.Count) return;

        //δημιουργείται μια καινούρια fdlist η οποία θα φιλοξενεί μόνο τις
συναρτησιακές εξαρτήσεις που αφορούν τα γνωρίσματα της newAttrList
        //πριν γίνει αυτό, διασπώνται οι συναρτησιακές εξαρτήσεις που έχουν
πολλαπλά γνωρίσματα στο δεξί σκέλος τους
        List<FD> tempofdlist = new List<FD>();
        foreach (FD fd in FDList)
        {
            foreach (Attr attr in fd.GetRight())
            {
```

```
                FD newfd = new FD();
                newfd.AddLeft(fd.GetLeft());
                newfd.AddRight(attr);
                if (newfd.GetAll().Intersect(newAttrList).Count() ==
newfd.GetAll().Count) tempofdlist.Add(newfd);
            }
        }

        //προσδιορίζεται το πλήθος των δυνητικών συνδυασμών
        int maxComb;
        maxComb = Factorial((ulong)cleanList.Count) / (Factorial((ulong)k) *
Factorial((ulong)(cleanList.Count - k)));

        //η αναζήτηση των συνδυασμών ξεκινά από τον πρώτο δυαδικό αριθμό που
δίνει πλήρη συνδυασμό k γνωρισμάτων, κι αυτός δίνεται από τον τύπο (2^k)-1
        i = (int)Math.Pow(2, k) - 1;
        x = 0;
        while (x < maxComb)
        {
            //ελέγχεται αν ο δυαδικός αριθμός έχει k ψηφία 1
            if (bin[i] == k)
            {
                x++;

                //δημιουργείται αντικείμενο τύπου Key
                Key key = new Key();

                //ο αριθμός i μετατρέπεται σε δυαδικό
                sBin = Convert.ToString(i, 2);

                //το sBin αντιστρέφεται
                char[] ch = sBin.ToCharArray();
                Array.Reverse(ch);
                sBin = new string(ch);

                //ελέγχεται η θέση των ψηφίων 1 στον δυαδικό αριθμό και
προστίθενται στο κλειδί key τα αντίστοιχα γνωρίσματα
                for (int j = 0; j < sBin.Length; j++)
                    if (sBin[j] == '1')
                    {
                        key.AddToKey(cleanList[j]);
                    }
                //ελέγχεται αν το νέο κλειδί δεν περιλαμβάνεται ήδη στην λίστα
με τα κλειδιά
                if (key != null && !key.KeyExists(keyList))
                {
                    //επίσης ελέγχεται αν ο εγκλεισμός του νέου κλειδιού
περιλαμβάνει όλα τα γνωρίσματα του σχήματος, κι αν ναι, τότε προστίθεται στη
λίστα των υποψήφιων κλειδιών
                    frmClosure frm = new frmClosure(key.GetAttrs(),
tempofdlist, this);
                    if (frm.attrClosure(key.GetAttrs(),
false).Intersect(newAttrList).Count() == newAttrList.Count)
                    {
                        keyList.Add(key);

                        //εμφανίζονται στην frmOut τα σχετικά μηνύματα για το
νέο κλειδί
                        if (showOut) frmout.AddOut(key.ToString() + "\n\n");
```

```
                    }
                }
            }
            i++;
        }

    }

    private void tKeys_Click(object sender, EventArgs e)
    {
        mnuKeys_Click(this, e);
    }

    private void MouseOn()
    {
        Cursor.Current = Cursors.WaitCursor;
    }

    private void MouseOff()
    {
        Cursor.Current = Cursors.Default;
    }

    private void mnuHelp_Click(object sender, EventArgs e)
    {
        //ανοίγουν οι σύντομες οδηγίες χρήσης
        Process.Start(AppDomain.CurrentDomain.BaseDirectory +
"MISC\\normalization_application_manual.pdf");
    }

    private void mnuBCNF_Click(object sender, EventArgs e)
    {
        //ελέγχεται αν υπάρχει έστω και μία συναρτησιακή εξάρτηση
        if (FDList.Count == 0)
            MessageBox.Show("Για να κάνετε διάσπαση σε BCNF πρέπει να έχετε
δημιουργήσει τουλάχιστον μία συναρτησιακή εξάρτηση.", "Διάσπαση",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            Decompose(false, this);
        }
    }

    /// <summary>
    /// Μέθοδος διάσπασης σε πίνακες BCNF
    /// </summary>
    /// <param name="alter">True αν πρόκειται για εναλλακτική σειρά
διάσπασης</param>
    /// <param name="frmCall">Μεταβλητή αναφοράς στη φόρμα που κάλεσε τη
μέθοδο</param>
    public void Decompose(bool alter, Form frmCall)
    {
        MouseOn();

        //δημιουργείται η frmout που αναφέρεται στην frmOut για να στέλνει
πληροφορίες για τη διαδικασία της διάσπασης
        frmOut frmout = new frmOut("Διαδικασία διάσπασης σε BCNF", false,
this);
```

```
        //μηδενίζεται η αρίθμηση της Relation
        Relation.aa = 0;

        //ορίζεται λίστα με τους πίνακες Relation
        List<Relation> RelList = new List<Relation>();

        //δημιουργείται νέος πίνακας Relation ο οποίος αρχικά περιλαμβάνει όλα
τα γνωρίσματα του σχήματος
        //και προστίθεται στη λίστα των πινάκων Relation
        Relation relInitial = new Relation(attrList);
        RelList.Add(relInitial);
        relInitial.Name = "R";

        frmout.AddOut("Θέλουμε να διασπάσουμε κατά BCNF τον αρχικό πίνακα " +
relInitial.ToString() + "\n\n");

        //δημιουργείται backup όλων των συναρτησιακών εξαρτήσεων, γιατί
υπάρχει περίπτωση να αλλάξουν στη μέθοδο αυτή
        foreach (FD fd in FDList)
        {
            fd.Backup();
            fd.Excluded = false;
        }

        //αν η λύση είναι η εναλλακτική, αντιστρέφεται η σειρά των
συναρτησιακών εξαρτήσεων
        if (alter) FDList.Reverse();

        //ορίζεται λίστα που παίρνει τα κλειδιά του πίνακα
        List<Key> keyList = new List<Key>();
        keyList = KeysGet(FDList, attrList, false);

        if (keyList.Count > 0)
        {
            frmout.AddOut("Τα υποψήφια κλειδιά του R είναι:\n\n");
            Key newkey = new Key();
            foreach (Key key in keyList)
            {
                frmout.AddOut(key.ToString() + "\n\n");
                newkey.AddToKey(key.GetAttrs());
            }
            relInitial.SetKey(newkey);
        }
        else
        {
            Key key = new Key();
            for (int i = 0; i < attrList.Count; i++)
                key.AddToKey(attrList[i]);
            keyList.Add(key);
            relInitial.SetKey(key);
            frmout.AddOut("Επιλέγεται ως υποψήφιο κλειδί το σύνολο των
γνωρισμάτων του R, καθώς δεν εντοπίστηκαν ως κλειδιά μεμονωμένα γνωρίσματα ή
συνδυασμοί αυτών.");
        }

        //αποκλείονται οι τετριμμένες συναρτησιακές εξαρτήσεις
        foreach (FD fd in FDList)
            if (fd.IsTrivial) fd.Excluded = true;
```

```
        //εξαιρούνται οι συναρτησιακές εξαρτήσεις που το αριστερό σκέλος τους
περιλαμβάνει κλειδί
        //ο έλεγχος γίνεται με τη βοήθεια της τομής
        foreach (FD fd in FDList)
            foreach (Key key in keyList)
                if (!fd.Excluded &&
fd.GetLeft().Intersect(key.GetAttrs()).Count() >= key.GetAttrs().Count)
                {
                    fd.Excluded = true;
                    break;
                }

        //ελέγχεται αν ισχύει ο κανόνας της μεταβατικότητας μεταξύ κάποιων
συναρτησιακών εξαρτήσεων
        //κι αν ναι, τότε το δεξί σκέλος αλλάζει και παίρνει τον εγκλεισμό του
αριστερού σκέλους (πλην τα γνωρίσματα του αριστερού)
        for (int i = 0; i < FDList.Count; i++)
        {
            for (int j = i + 1; j < FDList.Count; j++)
            {
                if (FDList[i] == FDList[j] | FDList[i].Excluded |
FDList[j].Excluded) continue;
                int x = 0;
                foreach (Attr attr in FDList[i].GetRight())
                    if (FDList[j].GetLeft().Contains(attr)) x++;
                if (x >= FDList[j].GetLeft().Count) //αν όλα τα γνωρίσματα του
δεξιού σκέλους βρέθηκαν στο αριστερό, τότε η συναρτησιακή εξάρτηση
τροποποιείται, με τα ίδια γνωρίσματα στα αριστερά και τον εγκλεισμό του
αριστερού σκέλους στα δεξιά
                {
                    frmClosure frm = new frmClosure(attrList, FDList, this);
                    FDList[i].AddRight(frm.attrClosure(FDList[i].GetLeft(),
false));
                    FDList[i].RemoveRight(FDList[i].GetLeft());
                }
            }
        }

        frmout.AddOut("==============================\n\n");

        //εμφανίζονται οι συναρτησιακές εξαρτήσεις
        frmout.AddOut("Οι συναρτησιακές εξαρτήσεις που θα χρησιμοποιηθούν για
τη διάσπαση, μετά:\n(1) την αφαίρεση των τετριμμένων,\n(2) την αφαίρεση αυτών
που δεν παραβιάζουν την BCNF μορφή και\n(3) τη μετατροπή αυτών που ικανοποιούν
τον κανόνα της μεταβατικότητας,\nείναι οι εξής:\n\n");
        bool chk = false;
        foreach (FD fd in FDList)
            if (!fd.Excluded)
            {
                frmout.AddOut(fd.ToString() + "\n\n");
                chk = true;
            }

        if (!chk)
        {
            frmout.AddOut("Καμία!!!\n\nΕπομένως, ο αρχικός πίνακας R είναι σε
BCNF μορφή και η διαδικασία σταματά.\n\n");
            relInitial.IsBCNF = true;
            goto SkipProcess;
```

```
        }

        frmout.AddOut("============================\n\n");

        //σαρώνονται όλοι οι πίνακες για να ελεγχθεί αν υπάρχει έστω και ένας
πίνακας που δεν είναι BCNF ώστε να διασπαστεί
        bool newRel = false;
        for (int i = 0; i < RelList.Count; i++)
        {
            if (!RelList[i].IsBCNF & !RelList[i].Excluded)
            //σαρώνονται όλες οι διαθέσιμες συναρτησιακές συναρτήσεις για να
δούμε αν μπορούμε να τον διασπάσουμε
            {
                foreach (FD fd in FDList)
                {
                    if (fd.Excluded) continue;
                    //αν η τομή x του συνόλου των γνωρισμάτων της
συναρτησιακής εξάρτησης και των γνωρισμάτων του πίνακα είναι μικρότερη σε
αριθμό από το πλήθος των γνωρισμάτων του πίνακα και ίση με το πλήθος των
γνωρισμάτων της συναρτησιακής εξάρτησης, τότε παραβιάζεται η BCNF μορφή και ο
πίνακας μπορεί να διασπαστεί
                    int x =
fd.GetAll().Intersect(RelList[i].GetList()).Count();
                    if (x < RelList[i].GetList().Count && x ==
fd.GetAll().Count)
                    {
                        //παρακάτω δημιουργούνται δύο νέοι πίνακες, ο rel1 και
ο rel2

                        //ο rel1 πίνακας παίρνει τα γνωρίσματα της
συναρτησιακής εξάρτησης
                        Relation rel1 = new Relation(fd.GetAll());

                        //ο rel2 πίνακας παίρνει τα γνωρίσματα από το αριστερό
σκέλος της συναρτησιακής εξάρτησης, συν τα γνωρίσματα του πίνακα που
διασπάστηκε, πλην αυτών που βρίσκονται στο δεξί σκέλος της συναρτησιακής
εξάρτησης
                        List<Attr> tempo = new List<Attr>();
                        tempo.AddRange(fd.GetLeft());

tempo.AddRange(RelList[i].GetList().Except(fd.GetRight()));
                        Relation rel2 = new Relation(tempo);

                        //δημιουργούνται δύο κλειδιά, ένα για τον καθένα
πίνακα
                        Key key1 = new Key();
                        Key key2 = new Key();

                        //το κλειδί του πρώτου πίνακα είναι η ορίζουσα της
συναρτησιακής εξάρτησης που προκάλεσε την διάσπαση
                        key1.AddToKey(fd.GetLeft());
                        rel1.SetKey(key1);

                        //προσδιορίζουμε το κλειδί του δεύτερου πίνακα (αυτό
που δίνει όλα τα γνωρίσματά του)
                        //δημιουργούμε μια τοπική λίστα κλειδιών και ως κλειδί
του δεύτερου πίνακα ορίζεται το πρώτο κλειδί της λίστας
                        List<Key> tempoKeyList = new List<Key>();
                        tempoKeyList = KeysGet(FDList, rel2.GetList(), false);
```

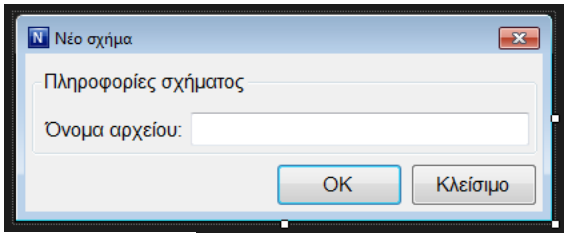
```
                        key2.AddToKey(tempoKeyList[0].GetAttrs());
                        rel2.SetKey(key2);

                        //ορίζονται τα ονόματα των δύο νέων πινάκων
                        if (Relation.isAA)
                        {
                            rel1.Name = "R" + ++Relation.aa;
                            rel2.Name = "R" + ++Relation.aa;
                        }
                        else
                        {
                            rel1.Name = RelList[i].Name + "1";
                            rel2.Name = RelList[i].Name + "2";
                        }

                        //οι δύο νέοι πίνακες προστίθενται στη λίστα
                        RelList.Add(rel1);
                        RelList.Add(rel2);

                        //εμφανίζονται τα σχετικά μηνύματα στην frmOut
                        frmout.AddOut("Με την \"" + fd.ToString() + "\" ο " +
RelList[i].Name + " διασπάται σε:\n\n" + rel1.ToString() + RelBCNF(rel1) +
"\n\n" + rel2.ToString() + RelBCNF(rel2) + "\n\n");

                        frmout.AddOut("==============================\n\n");

                        //ο πίνακας RelList[i] και η συναρτησιακή εξάρτηση fd
αποκλείονται από περαιτέρω διασπάσεις
                        RelList[i].Excluded = true;
                        fd.Excluded = true;

                        newRel = true;
                        break;
                    }
                }

                //ελέγχεται αν δημιουργήθηκαν νέοι πίνακες, κι αν ναι, η
σάρωση αρχίζει από την αρχή
                if (newRel)
                {
                    newRel = false;
                    i = -1;
                }
                else //ο πίνακας θεωρείται ότι είναι σε μορφή BCNF
                {
                    RelList[i].IsBCNF = true;
                    RelList[i].Excluded = true;
                }
            }
        }

        SkipProcess: //αν η διαδικασία της κανονικοποίησης λήξει πρόωρα,
οδηγούμαστε σε αυτό το σημείο του κώδικα

        //επανέρχονται οι συναρτησιακές εξαρτήσεις στην αρχική τους κατάσταση
        foreach (FD fd in FDList)
            fd.Restore();

        //εμφανίζονται στην frmOut ποιοι είναι οι BCNF πίνακες
```

```
        if (RelList.Count > 0)
        {
            frmout.AddOut("==============================\n\n");
            frmout.AddOut("Οι πίνακες BCNF είναι οι εξής:\n\n");
            foreach (Relation relation in RelList)
                if (relation.IsBCNF)
                    frmout.AddOut(relation.ToString() + "\n\n");
            frmout.AddOut("==============================\n\n");
        }

        //αν η διάσπαση ήταν με εναλλακτική σειρά, τότε αντιστρέφεται ξανά η σειρά των συναρτησιακών εξαρτήσεων
        if (alter) FDList.Reverse();

        //αν κάποια από τις συναρτησιακές εξαρτήσεις δεν χρησιμοποιήθηκε, εμφανίζεται το όνομά της και δίνεται η δυνατότητα εναλλακτικής διάσπασης
        foreach (FD fd in FDList)
            if (!fd.Excluded)
            {
                frmout.AddOut("Η συναρτησιακή εξάρτηση \"" + fd.ToString() + "\" δεν χρησιμοποιήθηκε.\n\n");
                if (!alter) frmout.AlterVisible();
            }

        MouseOff();
        frmout.ShowDialog(frmCall);
        frmout.Dispose();
    }

    /// <summary>
    /// Ελέγχει αν ο πίνακας Rel είναι BCNF
    /// </summary>
    private string RelBCNF(Relation rel)
    {
        foreach (FD fd in FDList)
        {
            if (fd.Excluded) continue;
            //αν η τομή x του συνόλου των γνωρισμάτων της συναρτησιακής εξάρτησης και των γνωρισμάτων του πίνακα είναι μικρότερη σε αριθμό από το πλήθος των γνωρισμάτων του πίνακα και ίση με το πλήθος των γνωρισμάτων της συναρτησιακής εξάρτησης, τότε παραβιάζεται η BCNF μορφή και ο πίνακας μπορεί να διασπαστεί
            //σε διαφορετική περίπτωση ο πίνακας είναι BCNF
            int x = fd.GetAll().Intersect(rel.GetList()).Count();
            if (x < rel.GetList().Count && x == fd.GetAll().Count)
                return "";
        }
        return "    (BCNF)";
    }

    private void tBCNF_Click(object sender, EventArgs e)
    {
        mnuBCNF_Click(this, e);
    }

    /// <summary>
    /// Επιστρέφει τη λίστα των καταχωρημένων γνωρισμάτων
    /// </summary>
    public List<Attr> GetAttrList()
```

```
    {
        return attrList;
    }

    /// <summary>
    /// Επιστρέφει την παραγοντική τιμή του x
    /// </summary>
    private static int Factorial(ulong x)
    {
        ulong total = 1;
        for (ulong i = 1; i <= x; i++)
            total = total * i;
        return (int)total;
    }

    private void mnuBCNFinSteps_Click(object sender, EventArgs e)
    {
        //δημιουργούνται αντίγραφα των συναρτησιακών εξαρτήσεων της FDList
        List<FD> fdlist = new List<FD>();
        foreach (FD fd in FDList)
        {
            FD fdnew = new FD();
            fdnew.Clone(fd);
            fdlist.Add(fdnew);
        }

        //ανοίγει η φόρμα της σταδιακής διάσπασης
        frmBCNFinSteps frm = new frmBCNFinSteps(this, attrList, fdlist);
        frm.ShowDialog(this);
        frm.Dispose();
    }

    private void tBCNFinSteps_Click(object sender, EventArgs e)
    {
        //ελέγχεται αν υπάρχουν καταχωρημένα 2 τουλάχιστον γνωρίσματα
        if (GridAttr.RowCount < 2)
            MessageBox.Show("Δεν μπορείτε να προχωρήσετε στη σταδιακή διάσπαση
έχοντας λιγότερα από 2 γνωρίσματα.", "Σταδιακή διάσπαση",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
            mnuBCNFinSteps_Click(this, e);
    }

    private void mnuDiplomatic_Click(object sender, EventArgs e)
    {
        //ανοίγει η διπλωματική εργασία
        Process.Start(AppDomain.CurrentDomain.BaseDirectory +
"MISC\\diplomatiki.pdf");
    }

    private void mnuUpdateNotes_Click(object sender, EventArgs e)
    {
        //ανοίγει η φόρμα με τις τελευταίες προσθήκες
        frmUpdateNotes frm = new frmUpdateNotes();
        frm.ShowDialog(this);
        frm.Dispose();
    }
}
```

```
public static class Extensions //inner class που χρησιμοποιείται για διάφορες
επεκτάσεις
{
    public static void DoubleBuffered(this DataGridView dgv, bool setting)
//μειώνει το flickering στα DataGridView
    {
        Type dgvType = dgv.GetType();
        PropertyInfo pi = dgvType.GetProperty("DoubleBuffered",
BindingFlags.Instance | BindingFlags.NonPublic);
        pi.SetValue(dgv, setting, null);
    }
}
```

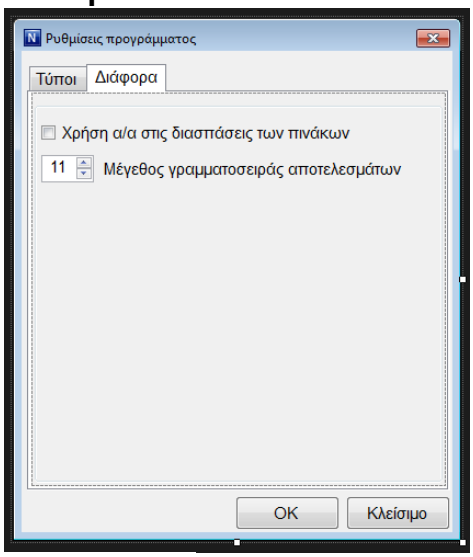
frmNew.cs




```
using System;
using System.Windows.Forms;

public partial class frmNew : Form
{
    //μεταβλητή με την ονομασία του σχήματος που θα χρησιμοποιηθεί και ως
ονομασία του αρχείου
    string filename = "";

    public frmNew()
    {
        InitializeComponent();
    }

    private void mnuClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void txtName_TextChanged(object sender, EventArgs e)
    {
        //το όνομα δεν μπορεί να είναι κενό
        if (txtName.Text.Trim().Equals(""))
            btnOK.Enabled = false;
        else
            btnOK.Enabled = true;
    }

    private void txtFilename_TextChanged(object sender, EventArgs e)
    {
        txtName_TextChanged(this, e);
    }

    public string GetFilename()
    {
        return filename;
    }
```

```
    private void btnOK_Click(object sender, EventArgs e)
    {
        filename = txtName.Text.Trim();
        //πριν δημιουργηθεί το νέο σχήμα, ελέγχεται το όνομά του, το οποίο δεν
πρέπει να περιέχει τους ειδικούς χαρακτήρες < > : "" / \ | ? *
        if (filename.Contains("<") | filename.Contains(">") |
filename.Contains(":") | filename.Contains("\"") | filename.Contains("/") |
filename.Contains("\\") | filename.Contains("? ") | filename.Contains("*"))
        {
            MessageBox.Show("Το όνομα του αρχείου δεν μπορεί να περιλαμβάνει
κανέναν από τους ειδικούς χαρακτήρες < > : \" / \\ | ? *.", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtName.Select();
            txtName.SelectAll();
            filename = "";
            return;
        }
        this.Close();
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void tClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

frmOptions.cs



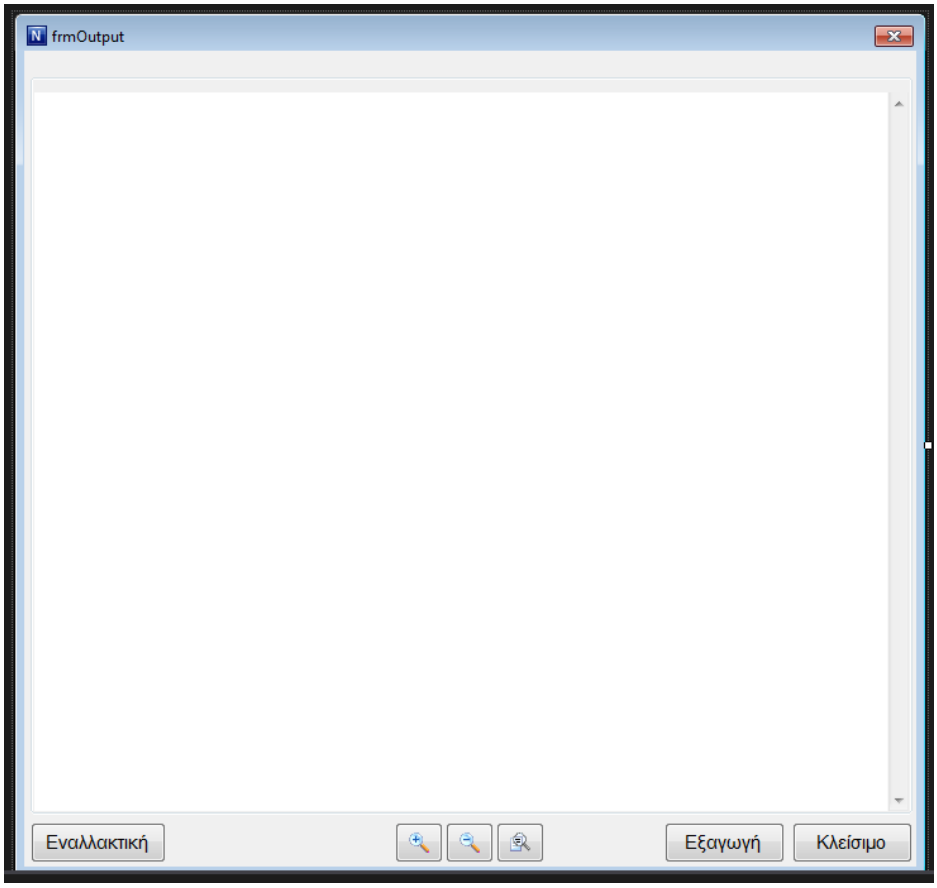

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Windows.Forms;

public partial class frmOptions : Form
{
    private string localApp = ""; //η διαδρομή όπου βρίσκεται ο φάκελος του
προγράμματος με αρχεία όπως οι ρυθμίσεις
```

```
    private FRMMAIN frmmain; //μεταβλητή αναφοράς στην FRMMAIN

    public frmOptions(FRMMAIN frmmain)
    {
        InitializeComponent();
        this.frmmain = frmmain;

        //ελέγχεται αν υπάρχει ο φάκελος xfdNormalization, κι αν όχι,
δημιουργείται εδώ
        localApp =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicat
ionData), "xfdNormalization");
        if (!Directory.Exists(localApp))
        {
            Directory.CreateDirectory(localApp);
            SaveTypes();
        }
    }

    private void tClose_Click(object sender, System.EventArgs e)
    {
        this.Close();
    }

    private void btnExit_Click(object sender, System.EventArgs e)
    {
        this.Close();
    }

    private void frmOptions_Load(object sender, System.EventArgs e)
    {
        //γίνεται η κατάλληλη επιλογή στο chkAA
        chkAA.Checked = Relation.isAA;

        //μορφοποιείται το Grid
        DataGridView g = Grid;
        g.Rows.Add(100);

        //περιορίζεται το flickering
        g.DoubleBuffered(true);

        //στοιχίζονται οι στήλες
        g.Columns["colType"].HeaderCell.Style.Alignment =
DataGridViewContentAlignment.MiddleLeft;

        //προσδιορίζονται τα πλάτη των στηλών ώστε να προσαρμοστούν σε
περίπτωση διαφορετικής ανάλυσης της οθόνης
        g.Columns["colType"].Width = g.Width * 348 / 366;

        //εμφανίζονται στο Grid οι αποθηκευμένοι τύποι γνωρισμάτων
        List<string> list = LoadTypes();

        for (int i = 0; i < list.Count; i++)
        {
            g.Rows[i].Cells["colType"].Value = list[i];
        }

        //η εστίαση του Grid μεταφέρεται στην πρώτη γραμμή και κρύβεται
        g.CurrentCell = g[0, 0];
```

```
        g.CurrentCell.Selected = false;

        numFont.Value = frmOut.iFont;
    }

    public void SaveTypes() //αποθήκευση τύπων γνωρισμάτων
    {
        bool isFirstTime;
        //αν το αρχείο δεν υπάρχει, σημαίνει ότι είναι η πρώτη φορά που
γίνεται αποθήκευση
        if (File.Exists(localApp + "\\types.txt"))
            isFirstTime = false;
        else
            isFirstTime = true;

        //αποθηκεύονται οι τύποι των γνωρισμάτων στο αρχείο types.txt
        StreamWriter FileW = new StreamWriter(localApp + "\\types.txt", false,
System.Text.Encoding.UTF8);

        if (isFirstTime) //αν είναι η πρώτη φορά που δημιουργείται το αρχείο,
αποθηκεύονται συγκεκριμένοι τύποι γνωρισμάτων
        {
            FileW.WriteLine("blob");
            FileW.WriteLine("boolean");
            FileW.WriteLine("char(n)");
            FileW.WriteLine("date");
            FileW.WriteLine("double");
            FileW.WriteLine("enum");
            FileW.WriteLine("float");
            FileW.WriteLine("integer");
            FileW.WriteLine("long");
            FileW.WriteLine("number");
            FileW.WriteLine("single");
            FileW.WriteLine("text");
            FileW.WriteLine("time");
            FileW.WriteLine("varchar(n)");
        }
        else //αλλιώς αποθηκεύονται οι τύποι που εμφανίζονται στο Grid
        {
            for (int i = 0; i < Grid.RowCount; i++)
            {
                string str;
                if (Grid.Rows[i].Cells["colType"].Value != null)
                    str =
Grid.Rows[i].Cells["colType"].Value.ToString().Trim();
                else
                    str = "";
                if (str != "") FileW.WriteLine(str);
            }
        }
        FileW.Close();
    }

    public List<string> LoadTypes()
    {
        //δημιουργείται πίνακας ώστε να επιστραφεί
        List<string> list = new List<string>();

        //φορτώνεται το αρχείο types.txt
```

```
        StreamReader FileR = new StreamReader(localApp + "\\types.txt",
System.Text.Encoding.UTF8);

        string str = "";
        bool isUsed = false;
        while (!FileR.EndOfStream)
        {
            str = FileR.ReadLine();
            //ελέγχεται αν το str χρησιμοποιείται ήδη
            for (int i = 0; i < list.Count; i++)
            {
                if (str == list[i])
                {
                    isUsed = true;
                    break;
                }
            }
            if (!isUsed) list.Add(str);
            isUsed = false;
        }
        FileR.Close();
        list.Sort();
        return list;
    }

    private void btnOK_Click(object sender, EventArgs e)
    {
        Relation.isAA = chkAA.Checked;
        frmOut.iFont = (int)numFont.Value;
        frmmain.SaveOptions();
        SaveTypes();
        this.Close();
    }
}
```

frmOut.cs




```
using System;
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;
using System.Drawing;

public partial class frmOut : Form
{
    public static int iFont; //η επιλογή του χρήστη για το μέγεθος της γραμματοσειράς του txtOut
    private string sOut; //το κείμενο που εμφανίζεται στη φόρμα
    private FRMMAIN frmmain; //μεταβλητή αναφοράς στην φόρμα που καλεί την frmOut

    /// <summary>
    /// Κατασκευαστής της frmOut
    /// </summary>
    /// <param name="sTitle">Το κείμενο που θα φαίνεται στον τίτλο του παραθύρου</param>
    /// <param name="isMini">Προσδιορίζει αν θα εμφανίζεται η μικρή έκδοση της frmOut</param>
    public frmOut(string sTitle, bool isMini, FRMMAIN frmmain)
    {
        InitializeComponent();
        this.Text = sTitle;
        this.frmmain = frmmain;
        txtOut.Font = new Font(txtOut.Font.Name, iFont);

        if (isMini)
        {
```

```
            int x = 300;
            this.Height -= x;
            GroupBox1.Height -= x;
            MoveButton(btnAlter, x);
            MoveButton(btnZoomIn, x);
            MoveButton(btnZoomOut, x);
            MoveButton(btnZoomReset, x);
            MoveButton(btnExport, x);
            MoveButton(btnExit, x);
        }
    }

    /// <summary>
    /// Μετακινεί το επιλεγμένο κουμπί b προς τα πάνω κατά x
    /// </summary>
    private void MoveButton(Button b, int x)
    {
        b.Location = new Point(b.Location.X, b.Location.Y - x);
    }

    /// <summary>
    /// Κάνει ορατό το κουμπί της εναλλακτικής
    /// </summary>
    public void AlterVisible()
    {
        btnAlter.Visible = true;
    }

    /// <summary>
    /// Προστίθεται κείμενο που θα εμφανιστεί στην frmOut
    /// </summary>
    public void AddOut(string sOut)
    {
        this.sOut = this.sOut + sOut;
        txtOut.Text = this.sOut.Replace("\n", Environment.NewLine);
    }

    /// <summary>
    /// Καθαρίζει το κείμενο της frmOut
    /// </summary>
    public void ResetOut()
    {
        sOut = ""; txtOut.Text = "";
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void frmOut_Shown(object sender, EventArgs e)
    {
        txtOut.SelectionStart = 0; txtOut.SelectionLength = 0;txtOut.Select();
    }

    private void btnAlter_Click(object sender, EventArgs e)
    {
        //καλεί την μέθοδο διάσπασης από την FRMMAIN, αλλά με εναλλακτική
σειρά, χρησιμοποιώντας τις συναρτησιακές εξαρτήσεις αντίστροφα
```

```
        frmmain.Decompose(true, this);
    }

    private void frmOut_FormClosing(object sender, FormClosingEventArgs e)
    {
        //αν τυχόν υπάρχει αρχείο που δημιουργήθηκε κατά την εξαγωγή,
διαγράφεται
        if (File.Exists("output.htm")) File.Delete("output.htm");
        //το μέγεθος της γραμματοσειράς επανέρχεται στο αρχικό του
        btnZoomReset_Click(this, e);
    }

    /// <summary>
    /// Αυξάνει το zoom στο txtOut
    /// </summary>
    private void btnZoomIn_Click(object sender, EventArgs e)
    {
        //αυξάνεται το μέγεθος της γραμματοσειράς του txtOut
        float i = txtOut.Font.Size;
        i++; txtOut.Font = new Font(txtOut.Font.Name, i);
    }

    /// <summary>
    /// Επαναφέρει το zoom του txtOut στο 100%
    /// </summary>
    private void btnZoomReset_Click(object sender, EventArgs e)
    {
        txtOut.Font = new Font(txtOut.Font.Name, iFont);
    }

    /// <summary>
    /// Μειώνει το zoom στο txtOut
    /// </summary>
    private void btnZoomOut_Click(object sender, EventArgs e)
    {
        //μειώνεται το μέγεθος της γραμματοσειράς του txtOut
        float i = txtOut.Font.Size;
        i--;
        if (i >= 1) txtOut.Font = new Font(txtOut.Font.Name, i);
    }

    private void frmOut_KeyDown(object sender, KeyEventArgs e)
    {
        //ελέγχεται αν πατήθηκε το Ctrl+A ώστε να επιλεγεί όλο το txtOut
        if (e.Control)
            if (e.KeyCode == Keys.A) txtOut.SelectAll();
    }

    private void btnExport_Click(object sender, EventArgs e)
    {
        //τα περιεχόμενα του txtOut εξάγονται σε μορφή HTML
        string str = txtOut.Text.Replace(Environment.NewLine, "<br>");

        StreamWriter FileW = new StreamWriter("output.htm", false,
System.Text.Encoding.UTF8);
        FileW.WriteLine(str); FileW.Close(); Process.Start("output.htm");
    }
}
```