ΔΡ. ΑΘΑΝΑΣΙΟΣ ΛΟΥΚΟΠΟΥΛΟΣ

ΔΡ. ΕΥΑΓΓΕΛΟΣ ΘΕΟΔΩΡΙΔΗΣ

Εισαγωγή στην SQL Εργαστηριακές Ασκήσεις σε MySQL5.7

Εργαστηριακός Οδηγός

Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγράμματα και Βοηθήματα
www.kallipos.gr

HEALLINK
Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

ΔΡ. ΑΘΑΝΑΣΙΟΣ ΛΟΥΚΟΠΟΥΛΟΣ

ΔΡ. ΕΥΑΓΓΕΛΟΣ ΘΕΟΔΩΡΙΔΗΣ

Εισαγωγή στην SQL Εργαστηριακές Ασκήσεις σε MySQL5.7

Εργαστηριακός Οδηγός

Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγράμματα και Βοηθήματα
www.kallipos.gr

Τίτλος Ηλεκτρονικού Συγγράμματος

Συγγραφή

Δρ. Αθανάσιος Λουκόπουλος

Δρ. Ευάγγελος Θεοδωρίδης

Κριτικός αναγνώστης

Δρ. Μιχαήλ Βασιλακόπουλος

Συντελεστές έκδοσης

Τεχνική Επεξεργασία: Αλεξία Τσουμάνη

ISBN: 978-960-603-473-2

ΣΥΝΔΕΣΜΟΣ ΕΛΛΗΝΙΚΩΝ ΑΚΑΔΗΜΑΪΚΩΝ ΒΙΒΛΙΟΘΗΚΩΝ

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

www.kallipos.gr

Πίνακας περιεχομένων

Πίνακας συντομεύσεων-ακρωνύμια

DBMS	Database Management System
ΣΔΒΔ	Σύστημα Διαχείρισης Βάσεων Δεδομένων
ER	Entiry Relationship Model
ΟΣ	Μοντέλο Οντοτήτων Συσχετίσεων
SQL	Structured Query Language
DB	Database
ΒΔ	Βάση Δεδομένων
QL	Query Language
PK	Primary Key
ΠΚ	Πρωτεύον Κλειδί
DDL	Data Definition Language
DML	Data Manipulation Language

Πρόλογος

Τα Συστήματα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) είναι από τα πλέον σημαντικά αντικείμενα στον κλάδο της Πληροφορικής και ενσωματώνονται στα εκπαιδευτικά προγράμματα όλων των αντίστοιχων τριτοβάθμιων τμημάτων. Αν και υπάρχει πληθώρα εξαιρετικών συγγραμμάτων πάνω στο αντικείμενο που αναλύουν σε βάθος ζητήματα που εκτείνονται από τη σχεδίαση Βάσεων Δεδομένων (ΒΔ) στο εννοιολογικό επίπεδο μέχρι την οργάνωση στο φυσικό επίπεδο, εντούτοις απουσιάζουν εν πολλοίς εγχειρίδια που θα μπορούσαν να αποτελέσουν οδηγό ενός εργαστηρίου που στόχο θα έχει την εξοικείωση των συμμετεχόντων με τα Σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων (ΣΣΔΒΔ) και τη γλώσσα SQL. Σκοπός του παρόντος συγγράμματος είναι να καλύψει κατά το δυνατόν το προαναφερθέν κενό.

Η παρουσίαση βασικών εννοιών όσον αφορά στο σχεδιασμό ΒΔ αλλά και των εντολών SQL γίνεται χρησιμοποιώντας την τρέχουσα έκδοση του ΣΣΔΒΔ MySQL 5.7. Ο λόγος που επιλέχθηκε το συγκεκριμένο DBMS είναι εξαιτίας της δημοτικότητάς του σε Web εφαρμογές. Σημειώνουμε ότι το μεγαλύτερο μέρος των εντολών που περιγράφονται και των παραδειγμάτων ισχύουν για τα κυριότερα ΣΣΔΒΔ.

Το αναγνωστικό κοινό στο οποίο απευθύνεται το σύγγραμμα είναι αυτό των αρχαρίων που δεν έχει πρότερη επαφή με το αντικείμενο των ΒΔ και επιθυμεί να αποκτήσει βασικές γνώσεις όσων αφορά στη σχεδίαση ΒΔ και τη σύνταξη ερωτημάτων σε SQL. Με εξαίρεση το τελευταίο κεφάλαιο (διασύνδεση με Java) η ύλη του συγγράμματος δεν προϋποθέτει κάποια ιδιαίτερη προηγούμενη γνώση για να γίνει κατανοητή αν και είναι βέβαιο πως αυτό θα είναι πιο εύκολο για τους φοιτητές τμημάτων σχετικών με την Πληροφορική.

Κλείνοντας τον πρόλογο θα θέλαμε να σημειώσουμε ότι σε κάθε θεματική ενότητα του συγγράμματος παρουσιάζεται σύντομη επισκόπηση των κύριων εννοιών. Ως εκ τούτου το σύγγραμμα είναι σε μεγάλο βαθμό αυτοτελές, χωρίς αυτό να σημαίνει ότι μπορεί να υποκαταστήσει τα κλασικά συγγράμματα του χώρου ως κύριο εγχειρίδιο ενός πρώτου μαθήματος ΒΔ καθώς η ύλη που καλύπτεται είναι υποσύνολο της ύλης ενός τυπικού μαθήματος. Αντίθετα θεωρούμε ότι έχει επικουρική αξία ως αυτοτελές εγχειρίδιο εργαστηριακού μαθήματος

Εισαγωγή

Κύρια φιλοδοξία του συγγράμματος είναι να αποτελέσει εγχειρίδιο για το εργαστηριακό κομμάτι ενός πρώτου μαθήματος πάνω σε σχεσιακές ΒΔ που αποσκοπεί στην απόκτηση βασικών γνώσεων προγραμματισμού σε SQL. Ως ΣΣΔΒΔ αναφοράς χρησιμοποιείται η MySQL στην τελευταία έκδοσή της (5.7 καθώς γράφονταν αυτές οι γραμμές).Το σύγγραμμα είναι οργανωμένο σε 14 κεφάλαια /ενότητες που αντιστοιχούν στις τυπικές εβδομάδες ενός εξαμήνου στην τριτοβάθμια εκπαίδευση. Η θεματολογία κάθε κεφαλαίου προσπαθεί να ακολουθήσει κατά το δυνατόν τη σειρά με την οποία αναμένεται να παρουσιαστούν τα αντίστοιχα αντικείμενα στο θεωρητικό μέρος ενός πρώτου μαθήματος ΒΔ. Κάθε κεφάλαιο περιέχει μία σύνοψη του καλυπτόμενου θέματος, παράδειγμα λυμένης εργαστηριακής άσκησης και άλυτα ενδεικτικά θέματα. Επίσης συνοδεύεται από επεξηγηματικό video. Ακολουθεί η οργάνωση των κεφαλαίων.

Στο Κεφ. 1 παρουσιάζονται ιστορικά στοιχεία για την εξέλιξη των ΒΔ και ακολουθούν οδηγίες εγκατάστασης της MySQL 5.7.

Στο Κεφ. 2 παρουσιάζονται στοιχεία που αφορούν στο σχεδιασμό ΒΔ. Ιδιαίτερο βάρος δίνεται στο μοντέλο οντοτήτων – συσχετίσεων (ΟΣ) και τη συνακόλουθη εξαγωγή πινάκων από το διάγραμμα ΟΣ.

Στο Κεφ. 3 παρουσιάζονται οι βασικές εντολές που αφορούν στη δημιουργία ΒΔ και πινάκων. Επίσης συζητούνται οι βασικοί τύποι που υποστηρίζονται από τη MySQL 5.7 καθώς επίσης και οι περιορισμοί πρωτεύοντος κλειδιού, μοναδικού κλειδιού και ξένου κλειδιού.

Στο Κεφ. 4 παρουσιάζονται οι βασικές εντολές για εισαγωγή εγγραφών τόσο από αρχείο όσο και από τη γραμμή εντολών. Επιπλέον συζητούνται εντολές που αφορούν στην επισκόπηση και τροποποίηση πινάκων.

Στο Κεφ. 5 συζητιέται η βασική δομή ενός SQL ερωτήματος SELECT-FROM-WHERE. Επίσης γίνεται αναφορά σε κάποιες από τις ειδικές συναρτήσεις που προσφέρονται από τη MySQL 5.7 και αφορούν στη διαμόρφωση αποτελεσμάτων. Τα παραδείγματα εμπλέκουν ερωτήματα σε έναν πίνακα.

Το Κεφ. 6 αναφέρεται στην υλοποίηση των πράξεων ένωσης, τομής, διαφοράς και καρτεσιανού γινομένου στην SQL και στη συνακόλουθη χρήση τους για την απάντηση ερωτημάτων που εμπλέκουν περισσότερους του ενός πίνακες. Στο παράδειγμα εργαστηριακής άσκησης δεν περιλαμβάνονται οι πράξεις της τομής και της διαφοράς καθώς η MySQL 5.7 δεν υλοποιεί τις πράξεις INTERSECT και EXCEPT.

Στο Κεφ. 7 παρουσιάζονται οι συνενώσεις πινάκων. Συζητούνται οι διαφορετικοί τρόποι γραφής εσωτερικών συνενώσεων και ακολουθεί η παρουσίαση των εξωτερικών συνενώσεων. Τέλος δείχνεται ο τρόπος υλοποίησης τομής και διαφοράς μέσω εσωτερικών και εξωτερικών συνενώσεων αντιστοίχως. Επιπλέον παρουσιάζεται και η υλοποίηση της πλήρους εξωτερικής συνένωσης που δεν υποστηρίζεται από την MySQL 5.7 απευθείας, ως ένωση αριστερής και δεξιάς εξωτερικής συνένωσης.

Το Κεφ. 8 αναφέρεται σε εμφωλευμένα ερωτήματα. Παρουσιάζεται η χρήση τους τόσο ως προσωρινούς πίνακες, στο FROM ενός εξωτερικού ερωτήματος, όσο και ως δεύτερο σκέλος συγκρίσεων, στο WHERE. Επιπλέον δείχνεται η υλοποίηση των πράξεων τομής και διαφοράς μέσω εμφωλευμένων ερωτημάτων. Τέλος συζητούνται ζητήματα βελτιστοποίησης κατά τη χρήση εμφωλευμένων στην MySQL 5.7.

Στο Κεφ. 9 παρουσιάζονται ερωτήματα συνάθροισης (aggregation queries) ενώ το Κεφ. 10 περιέχει την περιγραφή, χρήση και υλοποίηση της πράξης της διαίρεσης.

Το Κεφ. 11 συνοψίζει τις εντολές τις σχετικές με ενημέρωση και διαγραφή εγγραφών από έναν ή περισσότερους πίνακες.

Στο Κεφ. 12 περιλαμβάνεται μία μικρή εισαγωγή πάνω σε αποθηκευμένες διαδικασίες (stored procedures) και τη χρήση ευρετηρίων, ενώ το Κεφ. 13 εισάγει στην έννοια της δοσοληψίας και παρουσιάζει βασικά στοιχεία που αφορούν στην υποστήριξη δοσοληψιών από τη MySQL 5.7.

Τέλος στο Κεφ. 14 παρουσιάζονται βασικά στοιχεία που αφορούν στη διασύνδεση Java και MySQL.

Κεφάλαιο 1 Εισαγωγή στα Συστήματα Διαχείρισης Σχεσιακών ΒΔ / MySQL 5.7

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν ιστορικά στοιχεία που αφορούν στην ανάπτυξη των Συστημάτων Διαχείρισης Σχεσιακών ΒΔ (Relational Database Management Systems RDBMSs). Στη συνέχεια θα γίνει επισκόπηση του RDBMS MySQL της έκδοσης 5.7.

Προαπαιτούμενη γνώση
Δεν προαπαιτείται κάποια ιδιαίτερη γνώση από τον αναγνώστη. Καλό θα ήταν όμως ο αναγνώστης να έχει:

- *Βασικές γνώσεις κάποιας γλώσσας προγραμματισμού.*
- *Βασικές γνώσεις σε δομές δεδομένων.*

1.1 Εισαγωγικές Έννοιες

Καθημερινά, έστω και εν αγνοία μας, χρησιμοποιούμε συστήματα διαχείρισης βάσεων δεδομένων (database management system DBMS, εφεξής ΣΔΒΔ). Από την πλοήγηση σε ιστοτόπους, μέχρι τις δοσοληψίες μας με τα ATM των τραπεζών, στην πραγματικότητα δίνουμε εντολές και αλληλεπιδρούμε με κάποιο ΣΔΒΔ που σώζει τη σχετική πληροφορία, π.χ. την τιμή και τα στοιχεία ενός προϊόντος που επιθυμούμε να αγοράσουμε από ένα e-shop ή τα στοιχεία του τραπεζικού μας λογαριασμού. Τι είναι όμως μία Βάση Δεδομένων (ΒΔ εφεξής) και τι είναι ένα ΣΔΒΔ;

- ΒΔ: Μία συλλογή δεδομένων οργανωμένη με τρόπο που να διευκολύνει πράξεις πάνω σε αυτά. Τέτοιες πράξεις για παράδειγμα σε μία ΒΔ που σώζει στοιχεία φοιτητών μπορεί να περιλαμβάνουν: εύρεση του τηλεφώνου ενός φοιτητή, διαγραφή ενός φοιτητή, υπολογισμό του μέσου όρου της βαθμολογίας του, αλλαγή της διεύθυνσης ενός φοιτητή κλπ.
- ΣΔΒΔ: Μία εφαρμογή που αλληλεπιδρά με τους τελικούς χρήστες, πιθανώς με άλλες εφαρμογές και με τη ΒΔ εκτελώντας τις πράξεις που επιθυμεί ο χρήστης. Το κυριότερο χαρακτηριστικό των ΣΔΒΔ είναι ότι προσφέρουν μία *αφαίρεση (abstraction)* του *φυσικού* τρόπου με τον οποίο οργανώνεται μία ΒΔ στο μέσο μόνιμης αποθήκευσης, π.χ. σκληρό δίσκο. Επιπλέον, προσφέρουν έναν αφαιρετικό τρόπο διαχείρισης των δεδομένων μέσω γλωσσών ειδικού σκοπού, τις οποίες ονομάζουμε γλώσσες ερωτημάτων (query languages QL).

Είναι σύνηθες στην καθομιλουμένη να συγχέονται οι έννοιες της Βάσης Δεδομένων (ΒΔ εφεξής) και του ΣΔΒΔ. Έτσι πολλές φορές ακούμε να λέγεται πχ. «..η εφαρμογή χρησιμοποιεί τη ΒΔ MS Access..» αντί του ορθού «..ΣΔΒΔ MS Access..».

1.2 Ιστορικά Στοιχεία

Από την πρώτη κιόλας εμφάνιση των Η/Υ έγιναν αντιληπτές οι δυνατότητες που προσέφεραν στην αποθήκευση, τροποποίηση και επεξεργασία δεδομένων. Με τα σειριακά μέσα μόνιμης αποθήκευσης των δεκαετιών του ’40 και ’50 (π.χ. μαγνητικές ταινίες) οι δυνατότητες αυτές ήταν αδύνατο να αναπτυχθούν στο βαθμό που γνωρίζουμε σήμερα. Για την ακρίβεια τις προαναφερθείσες δεκαετίες ο ρόλος της αποθήκευσης σε μια υπολογιστική πλατφόρμα έμοιαζε περισσότερο σε ότι σήμερα γνωρίζουμε ως αντίγραφα ασφαλείας (backup). Παρόλα αυτά οι ερευνητές και εταιρείες του χώρου αναγνωρίζοντας τις δυνατότητες εξέλιξης στον τομέα της αποθήκευσης και επεξεργασίας δεδομένων, ίδρυσαν το 1959 την ομάδα CODASYL (Conference/Committee on Data Systems Languages), με στόχο την παραγωγή μιας πρότυπης γλώσσας χειρισμού δεδομένων.

Τη δεκαετία του ’60 έκαναν την εμφάνισή τους τα πρώτα μέσα αποθήκευσης άμεσης προσπέλασης, πρώτα με τη μορφή τυμπάνων (drums) και έπειτα με τη μορφή των σκληρών δίσκων (hard disk drives) που γνωρίζουμε σήμερα. Η άμεση προσπέλαση που προσέφεραν τα νέα συστήματα αποθήκευσης έγινε αμέσως κατανοητό ότι μπορούσε να κάνει τα υπολογιστικά συστήματα της εποχής να αντικαταστήσουν σε μεγάλο βαθμό τις ανθρώπινες διαδικασίες που βασίζονταν σε αποθήκευση και διαχείριση δεδομένων σε χαρτί. Έτσι άρχισε να διατυπώνεται σαφώς η ανάγκη για πρότυπα οργάνωσης, αποθήκευσης και χειρισμού δεδομένων δηλ. πράξεων σε αυτά. Δεν είναι τυχαίο ότι ο όρος ΒΔ διατυπώθηκε για πρώτη φορά το 1962 σύμφωνα με το λεξικό της Οξφόρδης. Επιπλέον, οι καινούργιες δυνατότητες που προσέφερε το υλικό έκαναν ορατή την περίπτωση της ταυτόχρονης αλληλεπίδρασης πολλών χρηστών με τα ίδια δεδομένα, γεγονός που ήταν αποφασιστικής σημασίας για την ανάπτυξη της αγοράς των ΒΔ στον τραπεζικό τομέα.

Τα συστήματα διαχείρισης δεδομένων της δεκαετίας του ’60 δύσκολα μπορεί να ειπωθεί ότι επρόκειτο για ΣΔΒΔ τουλάχιστον με τη σημερινή έννοια του όρου. Συνήθως αναφερόμαστε σε αυτά χρησιμοποιώντας τον όρο *Συστήματα Διαχείρισης Αρχείων*. Παράδειγμα τέτοιου συστήματος ήταν το IDS (Integrated Data Store). Με διάφορες εταιρείες να προσφέρει κάθε μια τα δικά της εργαλεία για αποθήκευση και χειρισμό δεδομένων η ανάγκη για ύπαρξη κάποιου στάνταρ οδήγησε στη δημιουργία εντός του CODASYL της πρώτης ομάδας ΒΔ από τον Charles Bachman δημιουργό του IDS. Η ομάδα αυτή οδήγησε στη διατύπωση του επονομαζόμενου CODASYL προτύπου σύμφωνα με το οποίο:

- Τα δεδομένα αναπαρίστανται ως συνδεδεμένη λίστα (στη γενική περίπτωση γράφος) εγγραφών.
- Η εύρεση μιας εγγραφής μπορεί να γίνει είτε βάσει κάποιας τιμής κλειδιού, ή κάνοντας πλοήγηση. Στην πρώτη περίπτωση χρησιμοποιείται ένας πίνακας κατακερματισμού για άμεση εύρεση της εγγραφής μέσω της τιμής κλειδιού. Στη δεύτερη περίπτωση ξεκινώντας από μία αρχική εγγραφή γίνεται πλοήγηση ακολουθώντας δείκτες που σώζουν οι εγγραφές. Στο πρότυπο υποστηριζόταν και μία επιπλέον πράξη: η διαπέραση όλων των εγγραφών.

Το CODASYL πρότυπο αναφέρεται επίσης σαν *μοντέλο δικτύου* ή *πλοήγησης*. Υποπερίπτωση αυτού του προτύπου ήταν το *ιεραρχικό μοντέλο* στο οποίο οι εγγραφές οργανώνονταν σε ιεραρχική/δεντρική μορφή (αντί γενικού γράφου). Τέλος η ομάδα ΒΔ του CODASYL δημιούργησε το πρότυπο μιας γλώσσας χειρισμού ΒΔ την COBOL.

Σχεδόν ταυτόχρονα με την οριστικοποίηση του CODASYL προτύπου, ο Edgar Codd που δούλευε για την IBM την περίοδο εκείνη πρότεινε το 1970 το *σχεσιακό μοντέλο* ΒΔ (relational database model). Η πρόταση βασίστηκε στην παρατήρηση ότι ο προγραμματισμός με το CODASYL πρότυπο ήταν δυσχερής. Ως εκ τούτου χρειαζόταν ένας απλούστερος τρόπος να μοντελοποιούνται τα δεδομένα των ΒΔ στο λογικό επίπεδο. Ο Codd πρότεινε για πρώτη φορά να μοντελοποιούνται τα δεδομένα σε μορφή πινάκων όπου οι στήλες αντιστοιχούν σε πεδία και οι γραμμές σε εγγραφές. Για αποφυγή δέσμευσης πλεονάζοντος χώρου διείδε και τον τρόπο βέλτιστου σχεδιασμού πινάκων. Πλέον αναφερόμαστε σε αυτή τη διαδικασία με τον όρο *κανονικοποίηση πινάκων*.

Για να κατανοήσουμε το μέγεθος της συμβολής ας θεωρήσουμε το ακόλουθο παράδειγμα. Έστω ότι θέλουμε η ΒΔ να σώζει στοιχεία φοιτητών και τις βαθμολογίες τους σε κάποια μαθήματα. Στο CODASYL πρότυπο μία εγγραφή φοιτητή θα μπορούσε να μοιάζει κάπως έτσι:

<ΑΜ, όνομα, επμο, δείκτηςΦπριν, δείκτηςΦμετά, δείκτηςΒαθμών>

<μάθημα, βαθμός, δείκτηςΒπριν, δείκτηςΒμετά>

Με άλλα λόγια μια πιθανή οργάνωση θα ενείχε:

- Τους φοιτητές σε μια διπλά συνδεδεμένη λίστα.
- Κάθε εγγραφή στην παραπάνω λίστα θα είχε δείκτη σε διπλά συνδεδεμένη λίστα με τους αναλογούντες βαθμούς.

Όπως μπορεί να εξαχθεί, ακόμα και απλές πράξεις όπως: Βρες τη βαθμολογία του φοιτητή με επμο Χ στο μάθημα Υ ενέχουν αρκετά πολύπλοκη διατύπωση ακόμα και στο λογικό επίπεδο (χωρίς να λάβουμε υπόψη τον τρόπο αποθήκευσης στο φυσικό μέσο).

Αντίθετα χρησιμοποιώντας το σχεσιακό μοντέλο η παραπάνω ΒΔ θα μπορούσε να υλοποιηθεί ως ένας πίνακας:
(ΑΜ, όνομα, επμο, μάθημα, βαθμός)

Το πρόβλημα με αυτή τη σχεδίαση είναι ότι αν ένας φοιτητής δίνει 40 μαθήματα τότε θα πρέπει να έχω 40 εγγραφές ανά φοιτητή στις οποίες θα επαναλαμβάνεται η πληροφορία του ον/μου (σπατάλη χώρου).

Ο Codd έδειξε πως μπορεί με αυτόματο τρόπο να παραχθεί μία βέλτιστη λύση σχεδιασμού που θα ελαχιστοποιεί τη σπατάλη χώρου. Στο παραπάνω παράδειγμα η βέλτιστη σχεδίαση θα γινόταν με δύο πίνακες ως εξής:
(ΑΜ, όνομα, επμο)
(ΑΜ, μάθημα, βαθμός)

Να σημειωθεί ότι σύμφωνα με την παραπάνω σχεδίαση: (i) στον πρώτο πίνακα κάθε φοιτητής εμφανίζεται μία φορά, (ii) δεύτερο πίνακα για κάθε φοιτητή υπάρχουν 40 εγγραφές, μία για κάθε μάθημα, (iii) το αρχικό ερώτημα εύρεσης βαθμολογίας εμπλέκει και τους δύο πίνακες. Ο Codd πρότεινε τη σχεσιακή άλγεβρα που έχει ως τελεστέους πίνακες (σχέσεις) και ως τελεστές πράξεις πάνω σε πίνακες και έδειξε ότι η άλγεβρα αυτή μπορούσε να απαντήσει στα περισσότερα των ερωτημάτων που τίθενται σε ένα ΣΔΒΔ. Στο παράδειγμά μας και σε ελεύθερη μετάφραση το αρχικό ερώτημα εύρεσης διατυπώνεται ως εξής: Συνένωσε τους δύο πίνακες στο πεδίο ΑΜ, επέλεξε τον/τις εγγραφές που έχουν επμο=Χ και μάθημα=Υ και εμφάνισε το βαθμό.

Η απλότητα και δύναμη της σχεσιακής άλγεβρας ώθησε τη βιομηχανία τη δεκαετία του ’70 στη σταδιακή υιοθέτηση του σχεσιακού μοντέλου όσον αφορά στη μοντελοποίηση των δεδομένων και στην εγκατάλειψη του CODASYL προτύπου. Παραδείγματα αρχικών *σχεσιακών ΣΔΒΔ* (ΣΣΔΒΔ) περιλαμβάνουν την Ingres από το Berkeley και το System R της IBM. Παράλληλα άρχισε να δημιουργείται το πρότυπο της SQL (Structured Query Language) πού έγινε τελικά αποδεκτό ως στάνταρ τη δεκαετία του ’80. Η SQL είναι μία γλώσσα σύνταξης ερωτημάτων που βασίζεται στη σχεσιακή άλγεβρα και χρησιμοποιείται ευρέως όπου χρησιμοποιείται ΣΣΔΒΔ και όχι μόνο.

Τις δεκαετίες που ακολούθησαν προτάθηκαν και άλλα μοντέλα δεδομένων. Ξεχωρίζουμε τη δεκαετία του ’80 το αντικειμενοστραφές μοντέλο, ενώ από το 2000 και μετά ξεχωρίζουν οι XML ΒΔ. Παρά τις εξελίξεις που μεσολάβησαν τα χρόνια από τη δεκαετία του ’80 και μετά τα ΣΣΔΒΔ εξακολουθούν να παίζουν πρωτεύοντα ρόλο σε πληθώρα εφαρμογών και για αυτόν το λόγο είναι σύνηθες να απαρτίζουν τον κύριο κορμό ενός μαθήματος Βάσεων Δεδομένων.

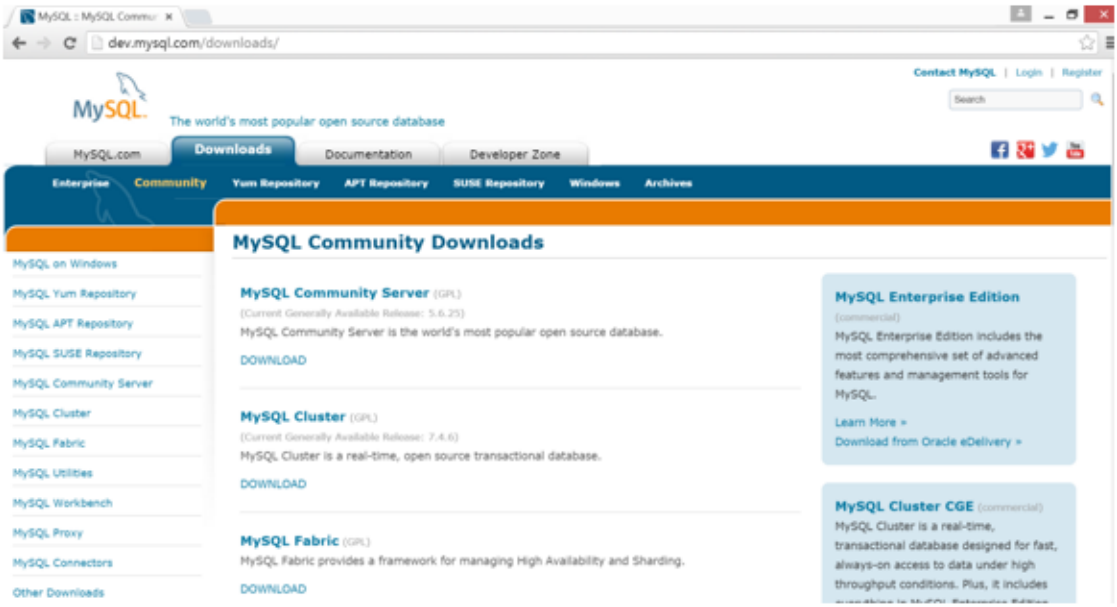
1.3 Οργάνωση Εργαστηρίου και ΣΔΒΔ

Στόχος του συνολικού εργαστηριακού μαθήματος είναι η εκμάθηση βασικών στοιχείων όσων αφορά στην κατασκευή μιας σχεσιακής ΒΔ. Κάτι τέτοιο είναι σύμφυτο τόσο με την εκμάθηση της SQL ως γλώσσας σύνταξης ερωτημάτων, όσο και του κομματιού του ορισμού και διαχείρισης δεδομένων (DDL data definition language, DML data manipulation language). Για τους σκοπούς αυτούς επιλέχθηκε το ΣΣΔΒΔ MySQL στην τρέχουσα έκδοση 5.7.

Ο λόγος επιλογής της MySQL ως εργαλείου εκμάθησης έγκειται στο γεγονός ότι είναι ιδιαίτερα διαδεδομένη όσον αφορά στην υλοποίηση Web services. Για την ακρίβεια η διάδοσή της είναι τέτοια που υπάρχει και το σχετικό ακρώνυμο LAMP stack για να αναφερθεί κανείς στην υλοποίηση ενός service χρησιμοποιώντας Linux, Apache Web Server, MySQL και PHP.

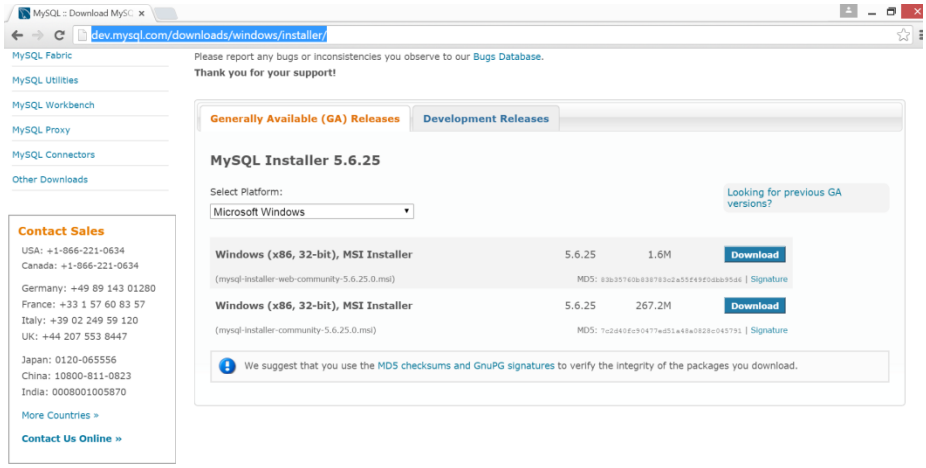
1.4 Εγκατάσταση MySQL Server σε Περιβάλλον Windows

Για την εγκατάσταση του συστήματος διαχείρισης βάσεων δεδομένων θα χρειαστεί να ανακτηθούν από τον ιστότοπο: http://dev.mysql.com/downloads/ δύο βασικά πακέτα λογισμικού: το **MySQL Community Server** και **MySQL Workbench** για την εγκατάσταση του εξυπηρετητή και του γραφικού περιβάλλοντος για την εκτέλεση λειτουργιών σε αυτόν αντίστοιχα.



Εικόνα 1.1: *Download MySQL.*

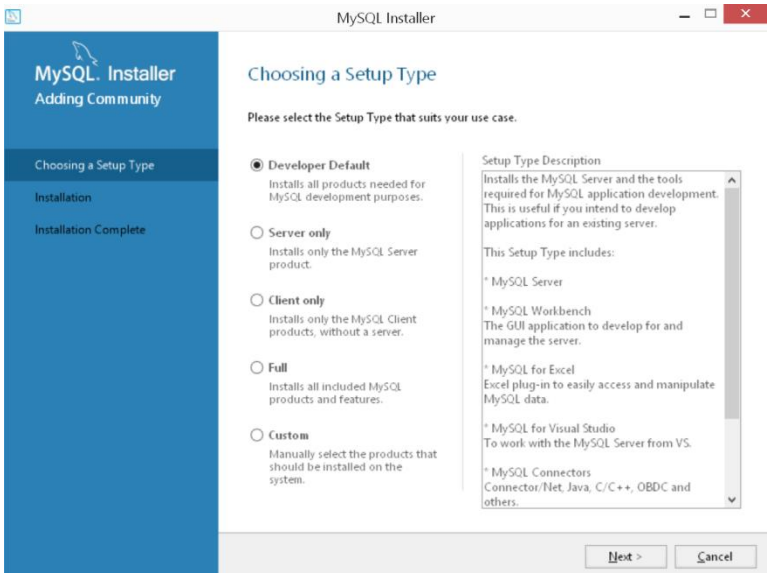
Για την εγκατάσταση όλων των δυνατών εφαρμογών της MySQL μπορεί να χρησιμοποιηθεί το ενοποιημένο πακέτο εγκατάστασης από τον σύνδεσμο: http://dev.mysql.com/downloads/windows/installer/



Εικόνα 1.2: *Download MySQL.*

Οι βασικές επιλογές εγκατάστασης είναι

- Developer Default: Εγκατάσταση εξυπηρετητή (server), clients, βιβλιοθηκών διασύνδεσης με τον server.
- Server Only: Εγκατάσταση μόνο του server.
- Client Only: Εγκατάσταση μόνο των προγραμμάτων σύνδεσης με τον server.
- Full: Εγκατάσταση όλων των στοιχείων
- Custom: Αναλυτική επιλογή των στοιχείων προς εγκατάσταση

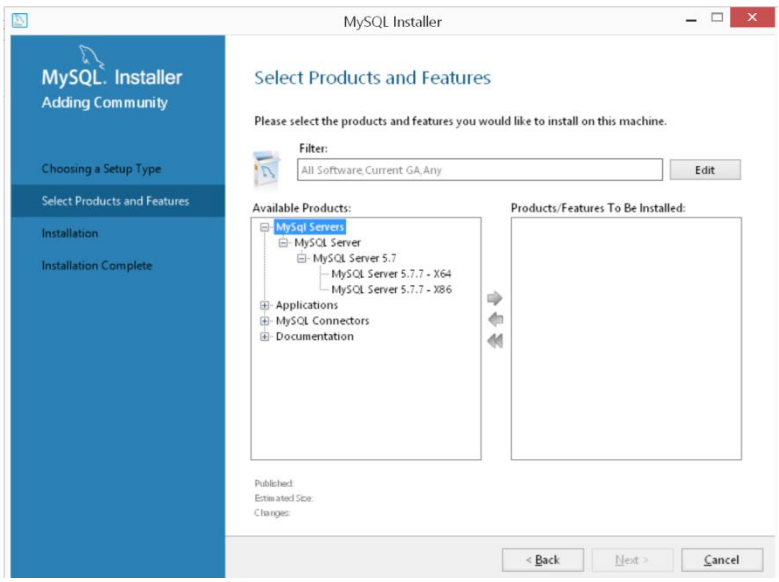


Εικόνα 1.3: *Εγκατάσταση MySQL – Επιλογή τύπου setup.*

Επιλέγοντας Custom εγκατάσταση οι βασικές επιλογές για εγκατάσταση που έχουμε είναι οι ακόλουθες:

1. MySQL Server
2. Applications
3. MySQL Connectors
4. Documentation

Στην πρώτη επιλογή επιλέγουμε την έκδοση του MySQL Server 32 ή 64 bit.

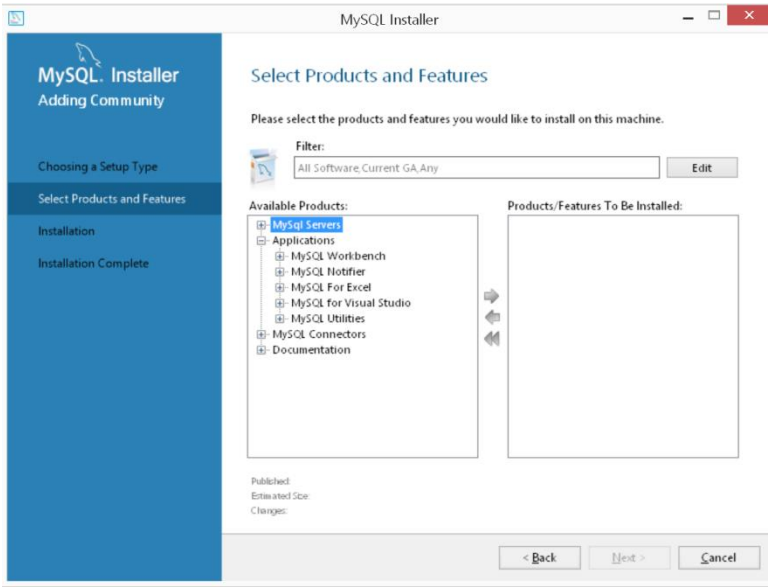


Εικόνα 1.4: *Εγκατάσταση MySQL – Επιλογή στοιχείων προς εγκατάσταση.*

Στις επιλογές Application οι δυνατές επιλογές είναι:

1. MySQL Workbench: Γραφικό Περιβάλλον διαχείρισης και προγραμματισμού του MySQL Server.
2. MySQL Notifier: Εφαρμογή που εγκαθίσταται στα Windows και στο system tray κάθε χρήστη και δείχνει σε αυτόν την κατάσταση του MySQL Server, αν είναι σε λειτουργία κλπ.

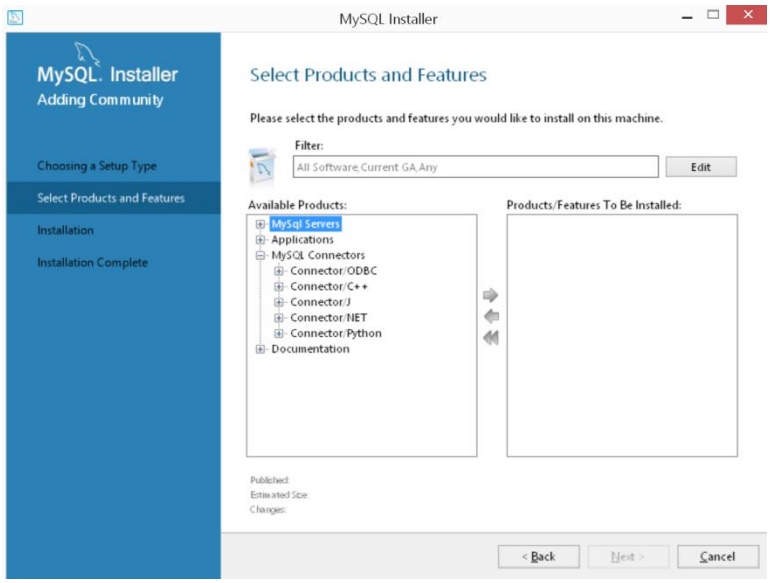
3. MySQL for Excel: Εφαρμογή διασύνδεσης της MySQL με την εφαρμογή MS Office Excel.
4. MySQL for Visual Studio: Εφαρμογή διασύνδεσης της MySQL με την εφαρμογή ανάπτυξης λογισμικού Visual Studio.



Εικόνα 1.5: *Εγκατάσταση MySQL – Επιλογή στοιχείων προς εγκατάσταση.*

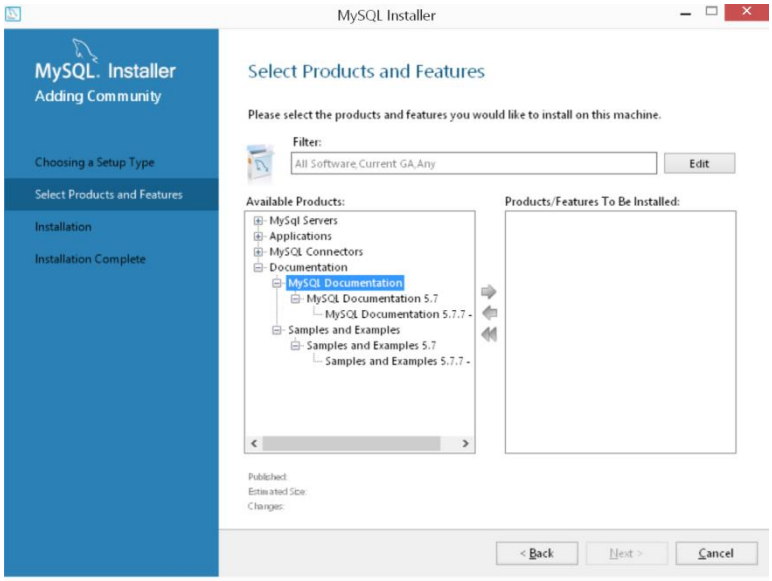
Στις επιλογές MySQL Connectors υπάρχουν οι απαραίτητες βιβλιοθήκες/οδηγοί για να διασυνδέσουμε τον MySQL Server με εφαρμογές/εκτελέσιμα προγράμματα υλοποιημένα σε διάφορες γλώσσες προγραμματισμού. Οι διαθέσιμες βιβλιοθήκες είναι οι:

1. ODBC
2. C++
3. J – Java
4. NET - .NET
5. Python



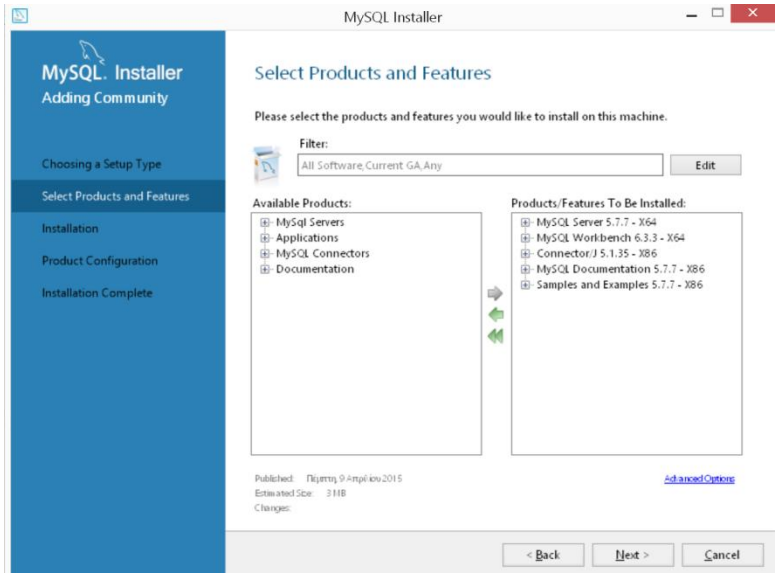
Εικόνα 1.6: *Εγκατάσταση MySQL – Επιλογή στοιχείων προς εγκατάσταση.*

Στην εγκατάσταση της τεκμηρίωσης (Documentation) υπάρχουν η τεκμηρίωση των διαφόρων χαρακτηριστικών του MySQL Server και ένα σύνολο δοκιμαστικών (sample) βάσεων δεδομένων.



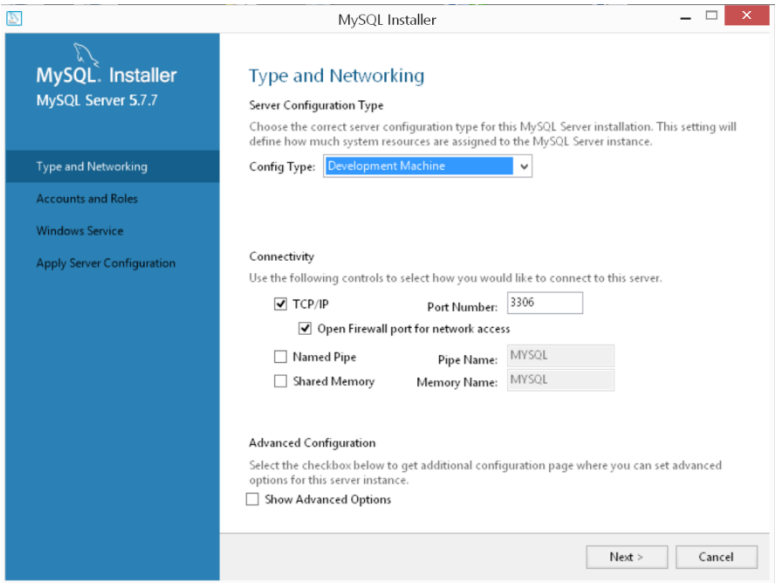
Εικόνα 1.7: *Εγκατάσταση MySQL – Επιλογή στοιχείων προς εγκατάσταση.*

Για τις ανάγκες του εργαστηρίου κατ' ελάχιστον πρέπει να επιλεγούν ο MySQL Server, το MySQL Workbench, η τεκμηρίωση (Documentation) και ο Connector/J (για την Java).



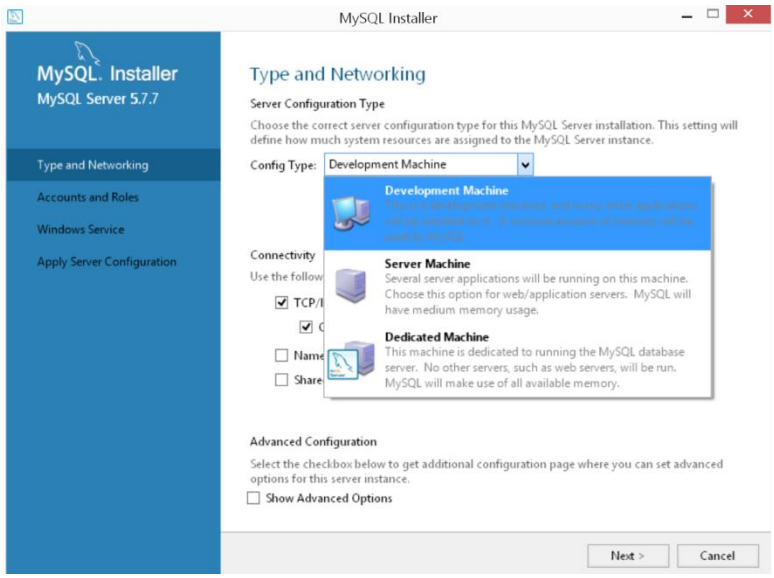
Εικόνα 1.8: *Εγκατάσταση MySQL – Επιλογή στοιχείων προς εγκατάσταση.*

Κατά την εγκατάσταση του MySQL Server το πακέτο εγκατάστασης ζητάει από τον χρήστη τις ακόλουθες παραμέτρους εγκατάστασης και εκτέλεσης.



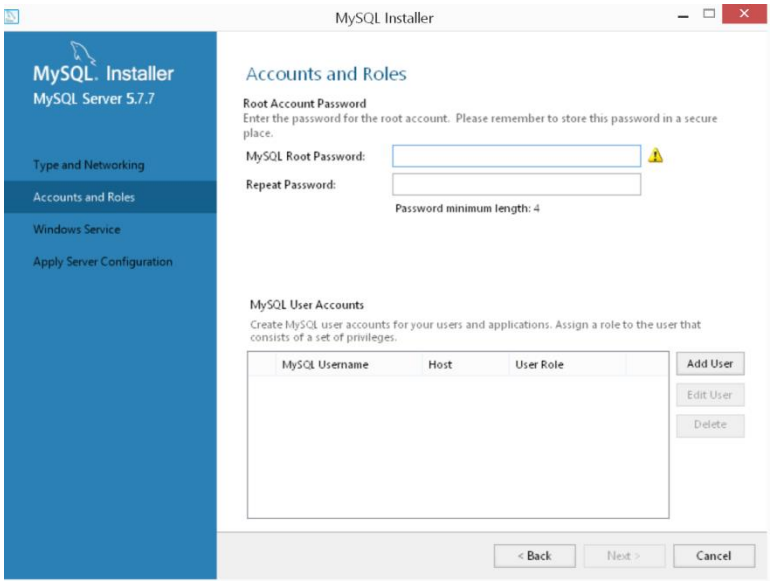
Εικόνα 1.9: *Εγκατάσταση MySQL – Επιλογή παραμέτρων λειτουργίας.*

Η βασική επιλογή είναι το Θύρα (Port Number) στην οποία θα δέχεται ο εξυπηρετητής τα αιτήματα. Η τυπική θύρα του MySQL Server είναι η 3306. Επίσης ο χρήστης καλείται να επιλέξει σε τι τρόπο λειτουργίας θα εγκατασταθεί ο εξυπηρετητής (Developer, Server, Dedicated). Κάθε τρόπος λειτουργίας δίνει/χρησιμοποιεί με διαφορετικό τρόπο τους πόρους του υλικού (CPU, Memory).



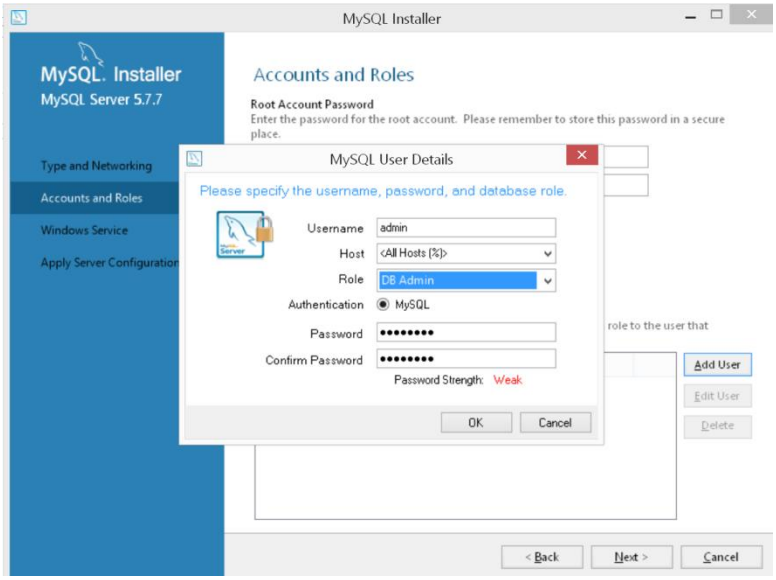
Εικόνα 1.10: *Εγκατάσταση MySQL – Επιλογή παραμέτρων λειτουργίας.*

Στη συνέχεια καλείται ο χρήστης να δώσει τον κωδικό του διαχειριστή του συστήματος (root). Ο λογαριασμός αυτός έχει πλήρη δικαιώματα στις λειτουργίες του εξυπηρετητή (εγκατάσταση/απεγκατάσταση, έναρξη/παύση, διαχείριση ΒΔ, διαχείριση χρηστών κλπ.).

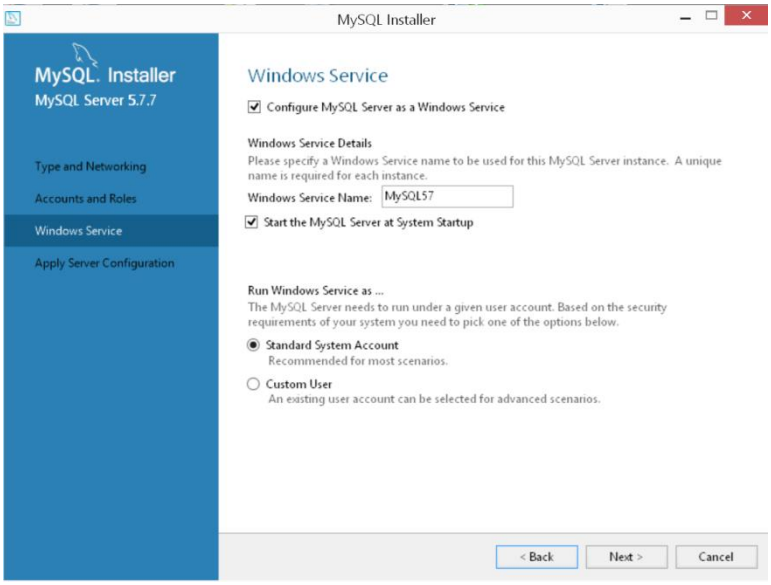


Εικόνα 1.11: *Εγκατάσταση MySQL – Λογαριασμός χρήστη.*

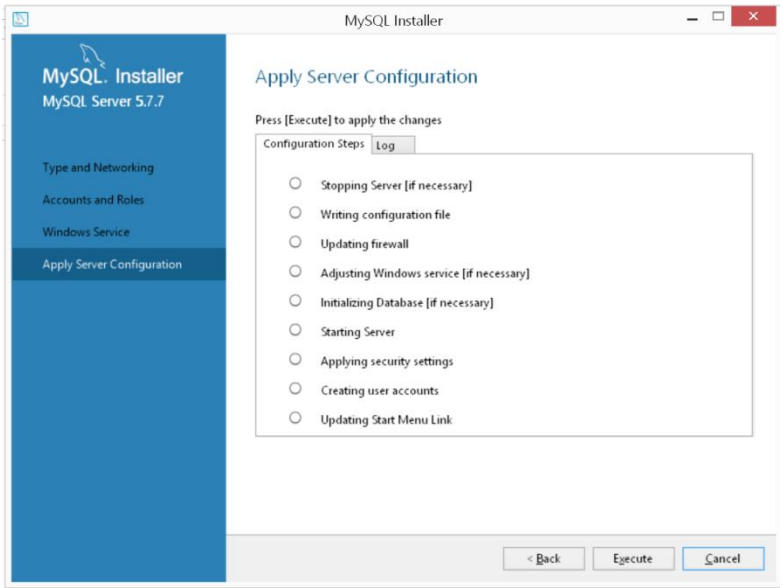
Υπάρχει και η δυνατότητα να δημιουργηθούν και άλλοι λογαριασμοί χρηστών με κάποιον συγκεκριμένο ρόλο (οι ρόλοι θα καλυφθούν σε επόμενο εργαστήριο).



Εικόνα 1.12: *Εγκατάσταση MySQL – Σύνδεση με MySQL Server.*

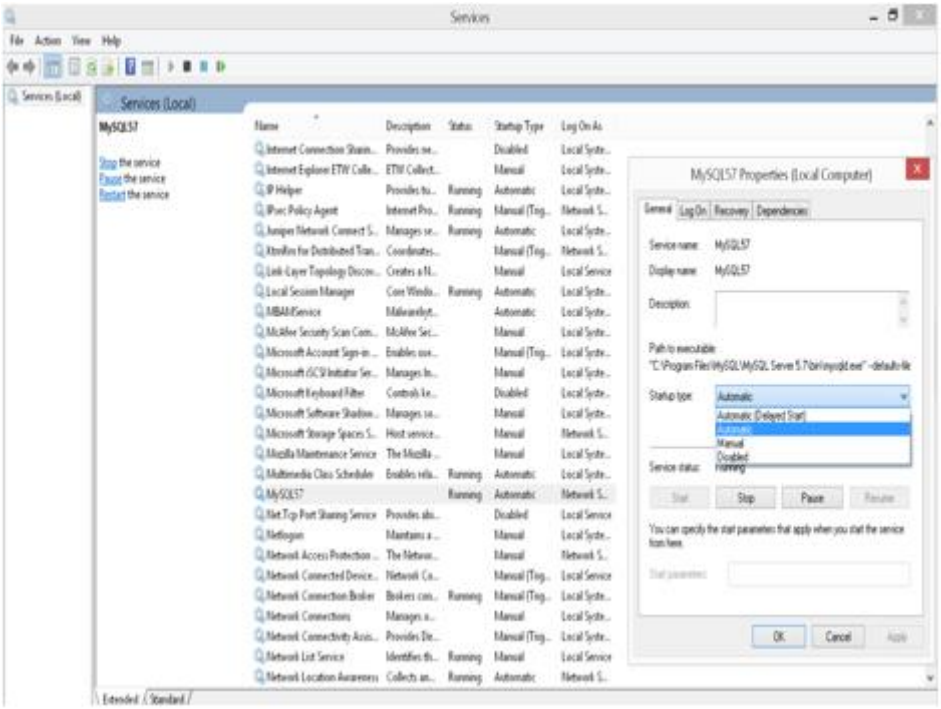


Εικόνα 1.13: *Εγκατάσταση MySQL – Windows Services.*



Εικόνα 1.14: *Εγκατάσταση MySQL – Server Configuration.*

Με το τέλος της εγκατάστασης ο εξυπηρετητής εκτελείται. Επίσης έχουν παραμετροποιηθεί οι υπηρεσίες των Windows ώστε να ξεκινούν τον MySQL Server με την εκκίνηση του λειτουργικού συστήματος. Προς επιβεβαίωση, ελέγξτε τις υπηρεσίες (services) των Windows. Δώστε στο μενού έναρξης το υπηρεσίες (services).

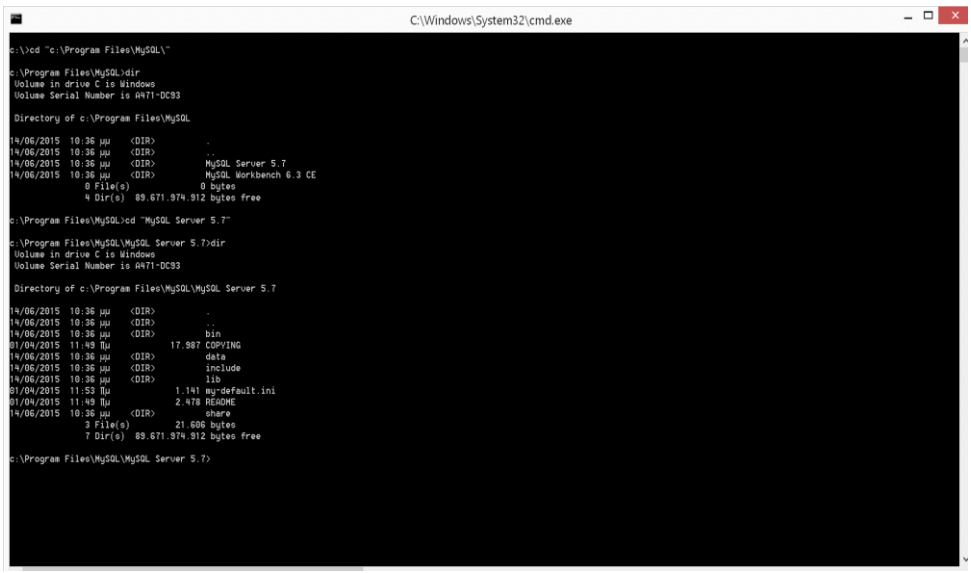


Εικόνα 1.15: *Εγκατάσταση MySQL – Windows Services.*

1.5 Command Line Client

Για τη σύνδεση και την εκτέλεση διαφόρων λειτουργιών στον MySQL Server μπορεί να χρησιμοποιηθεί είτε η εφαρμογή γραμμής εντολών (command line) ή το γραφικό περιβάλλον (Workbench).

Στη γραμμή εντολών των Windows (έναρξη, cmd) μεταβείτε στον φάκελο εγκατάστασης της MySQL. Τυπικά αυτός είναι ο φάκελος C:\Program Files\MySQL.



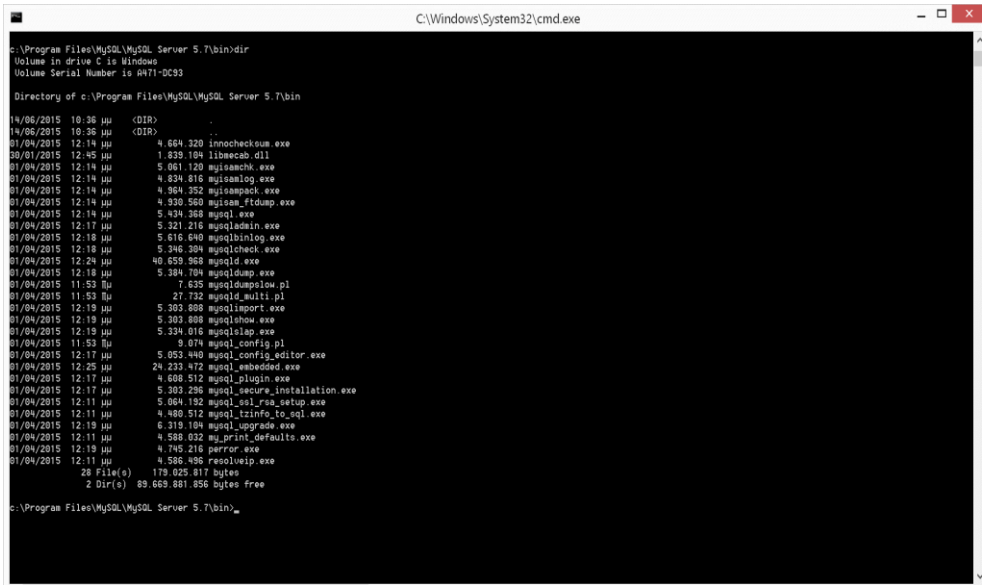
Εικόνα 1.16: *Μετάβαση στο φάκελο MySQL.*

Υπάρχουν 2 βασικοί φάκελοι: Ο MySQL Server και ο MySQL Workbench. Μεταβείτε στον πρώτο φάκελο. Ο φάκελος αυτός αποτελείται από τους εξής υποφακέλους:

1. bin: φάκελος εκτελέσιμων αρχείων.

2. data: φάκελος δεδομένων.
3. include: φάκελος βιβλιοθηκών πηγαίου κώδικα.
4. lib: φάκελος εκτελέσιμων βιβλιοθηκών.
5. my-default.ini: αρχείο ορισμού παραμέτρων.

Μεταβείτε στο φάκελο bin με τις εκτελέσιμες εφαρμογές.



Εικόνα 1.17: *Μετάβαση στο φάκελο bin.*

Τα βασικά εκτελέσιμα αρχεία είναι το mysqld.exe, mysqldump.exe, mysql.exe που είναι τα εκτελέσιμα του εξυπηρετητή, της εφαρμογής εξαγωγής ΒΔ και της γραμμής εντολών MySQL αντίστοιχα.

Για την σύνδεση με τον MySQL Server θα γίνει χρήση του εκτελέσιμου mysql.exe το οποίο λαμβάνει μία σειρά από παραμέτρους. Οι πιο βασικές είναι οι ακόλουθες:
mysql -u USERNAME -pPASSWORD -h HOSTNAMEORIP -P PORTNUM DATABASENAME

- -u USERNAME: το όνομα χρήστη μετά το -u
- -pPASSWORD: ο κωδικός χρήστη (χωρίς κενό) μετά το -p. Αν παραληφθεί το PASSWORD ζητείται αργότερα κατά την σύνδεση. Αυτός είναι και ο πιο ενδεδειγμένος τρόπος εισαγωγής καθώς δε φαίνεται έτσι ο κωδικός στην γραμμή εντολών.
- -h HOSTNAMEORIP: το όνομα του εξυπηρετητή ή η IP διεύθυνση (πχ. localhost ή 127.0.0.1). Αν παραληφθεί χρησιμοποιείται localhost.
- -P PORTNUM: η θύρα σύνδεσης. Αν παραληφθεί χρησιμοποιείται η τυπική 3306.
- DATABASENAME: το όνομα της ΒΔ που θέλουμε να επεξεργαστούμε. Αν το παραλείψουμε θα πρέπει να την επιλέξουμε στην συνέχεια.

Συνεπώς αν θέλουμε να συνδεθούμε από έναν υπολογιστή σε έναν MySQL Server εγκατεστημένο στο ίδιο μηχάνημα θα δώσουμε την εντολή:

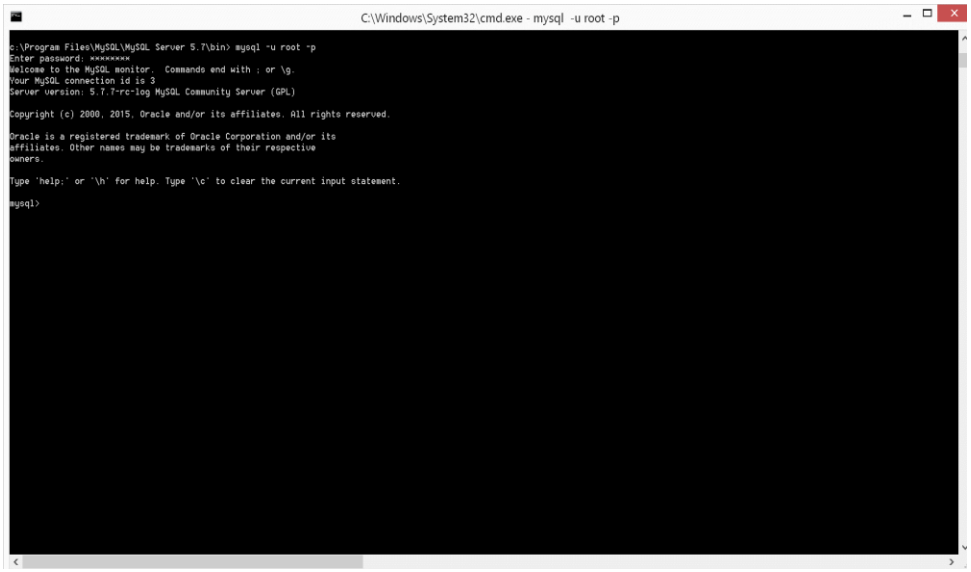
```
mysql -u root -p
```

θα μας ζητήσει τον κωδικό για τον λογαριασμό root και στη συνέχεια θα συνδεθούμε στην γραμμή εντολών της mysql.

Δύο συχνά θέματα που μπορεί να αντιμετωπισθούν είναι τα ακόλουθα:

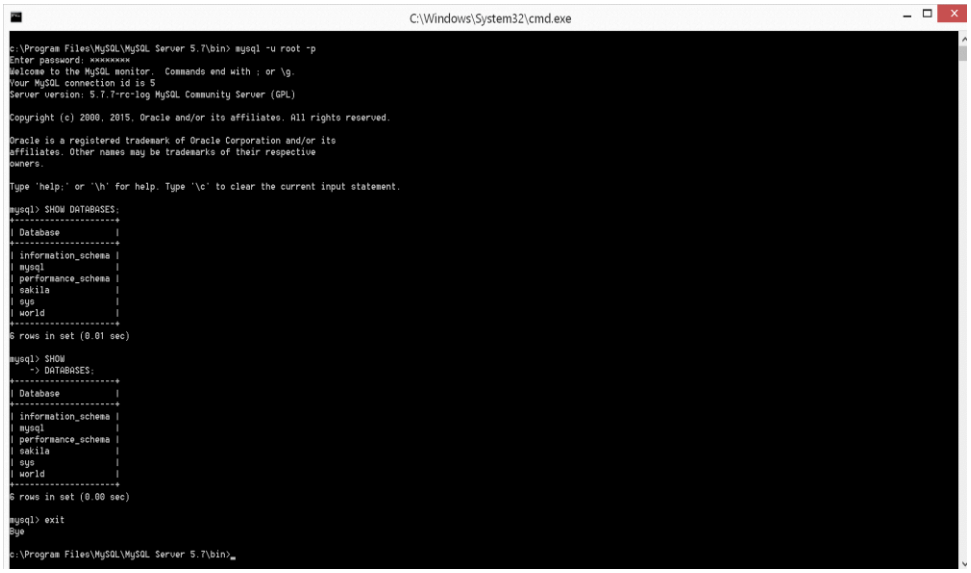
- Να μην εκτελείται ο εξυπηρετητής οπότε θα πάρουμε το μήνυμα λάθους: ERROR 2003 (HY000): Can’t connect to MySQL server on ' localhost ' (10061)

- Να μη δώσουμε σωστό κωδικό για τον root, οπότε θα πάρουμε το μήνυμα λάθους: ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)



Εικόνα 1.18: *Σύνδεση με MySQL Server.*

Με την επιτυχημένη σύνδεση μας δίνεται η γραμμή εντολών mysql>. Από το σημείο αυτό και μετά μπορούμε να γράψουμε SQL εντολές καθώς και «ειδικότερες» εντολές του MySQL Server όπως θα δούμε στη διάρκεια του εργαστηρίου. Για την εκτέλεση εντολών, πληκτρολογούμε την εντολή, σε μία η περισσότερες γραμμές, τερματιζόμενη με το ερωτηματικό ‘;’ και δίνουμε ENTER.



Εικόνα 1.19: *Παραδείγματα εντολών.*

Συνεπώς όπως φαίνεται στην Εικόνα 1.19, είτε δώσουμε την εντολή:

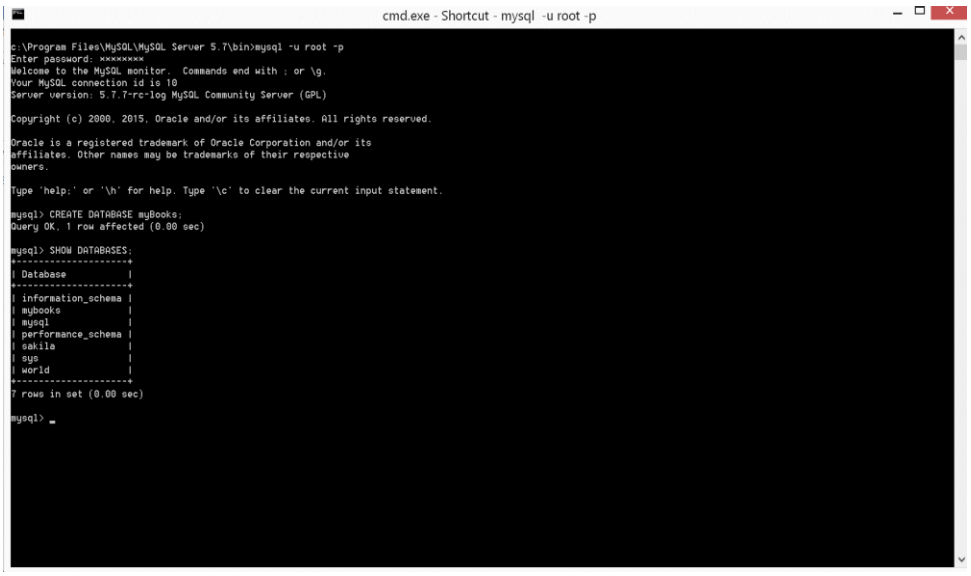
```
SHOW DATABASES;
```

ή γράψουμε:

```
SHOW
DATABASES;
```

το αποτέλεσμα είναι το ίδιο. Δηλαδή οι εντολές του κώδικα μπορεί να καταλαμβάνουν περισσότερες από μία γραμμές.

Για να επιστρέψουμε στη γραμμή εντολών των Windows δίνουμε την εντολή:
Exit;
Για να δημιουργήσουμε μία βάση δεδομένων χρησιμοποιούμε την εντολή CREATE DATABASE και το όνομα της ΒΔ. Π.χ.:
CREATE DATABASE myBooks;
Ενώ για να δούμε όλες τις διαχειριζόμενες ΒΔ χρησιμοποιούμε την εντολή SHOW DATABASES;

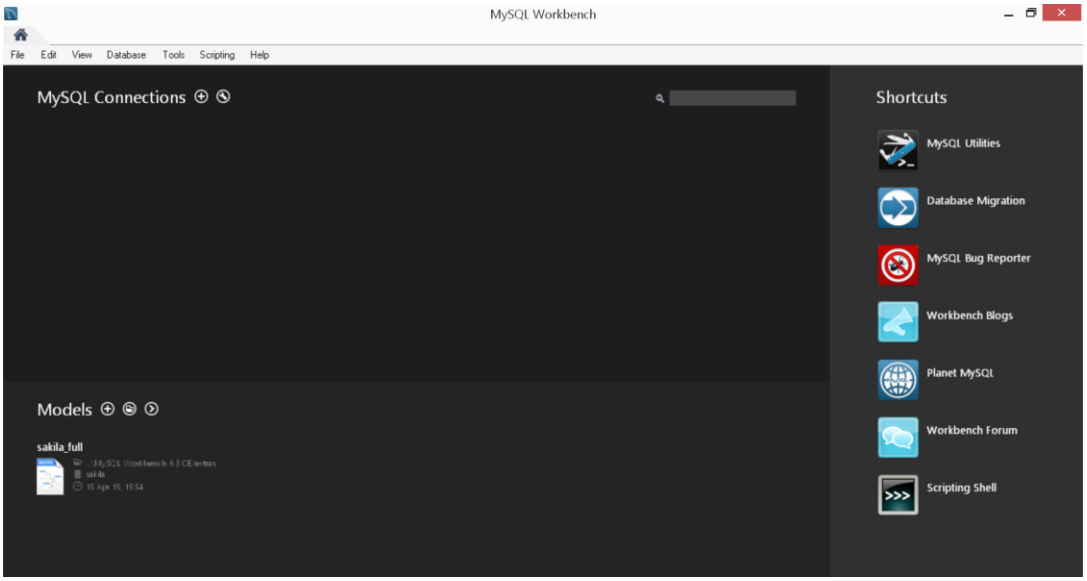


Εικόνα 1.20: *Η εντολή SHOW DATABASES.*

Για την διαγραφή μίας ΒΔ χρησιμοποιούμε την εντολή DROP DATABASE και το όνομα της βάσης. Π.χ.
DROP DATABASE myBooks;
Περισσότερα για τις παραπάνω εντολές χειρισμού ΒΔ θα αναλυθούν στο Κεφ. 3.

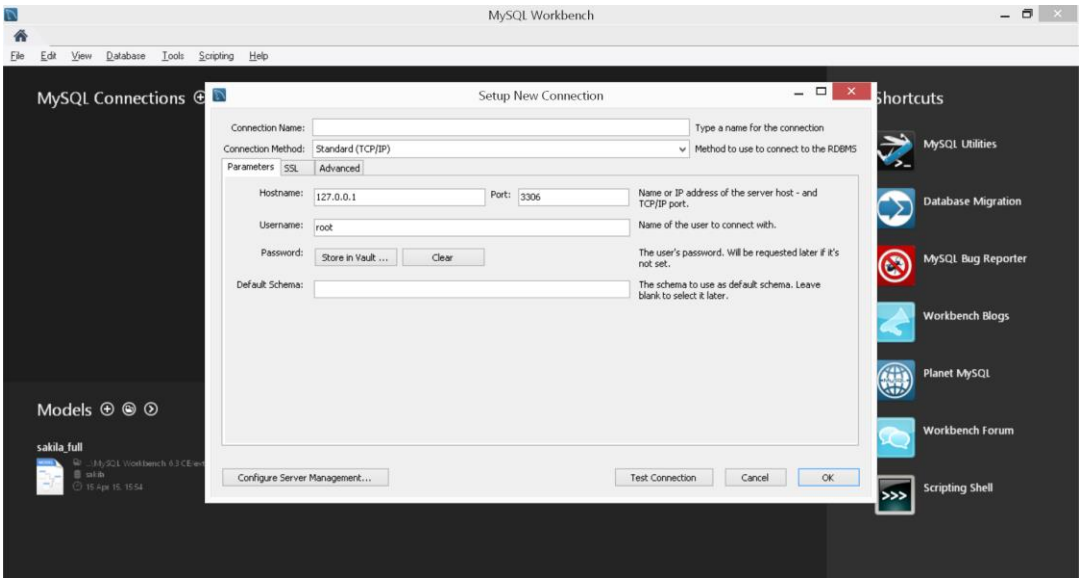
1.6 MySQL Workbench

Όταν εκκινήσουμε το MySQL Workbench, η γραφική διεπαφή μας ζητάει να ορίσουμε τις παραμέτρους σύνδεσης με τον SQL Server εξυπηρετητή (MySQL Connections). Οι παράμετροι είναι αντίστοιχοι με αυτούς που δίνουμε στον command line client.

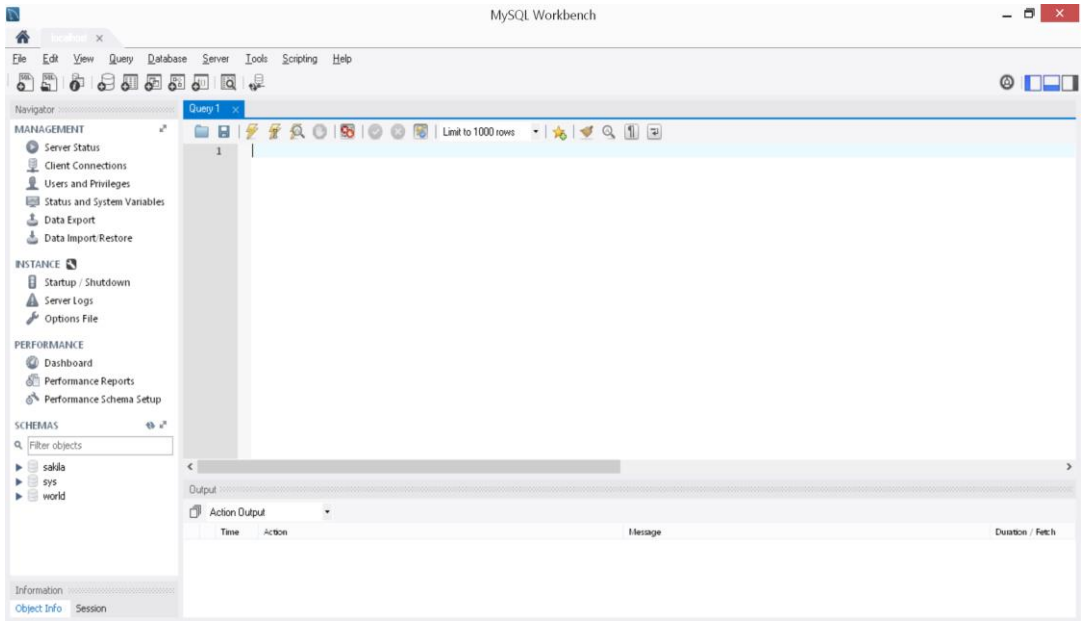


Εικόνα 1.21: *Παραμετροποίηση Workbench.*

Οι βασικές παράμετροι που πρέπει να περάσουν είναι το hostname, port, username, database και connection name.

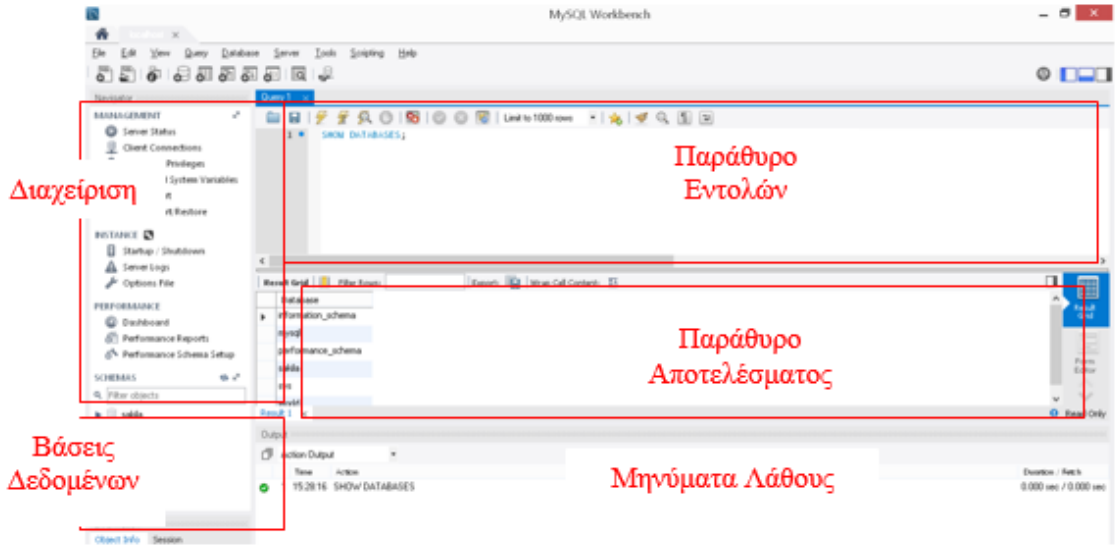


Εικόνα 1.22: *Παραμετροποίηση Workbench.*



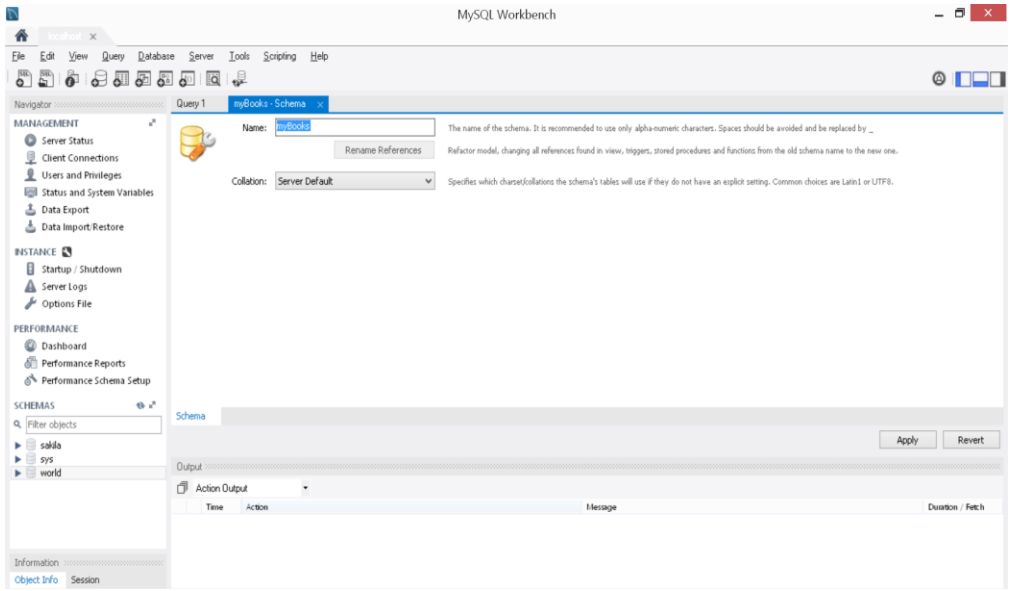
Εικόνα 1.23: *Γραφικό περιβάλλον Workbench.*

Τα βασικά μέρη στο γραφικό Περιβάλλον είναι: (i) οι επιλογές για την διαχείριση του MySQL Server (Management, Instance, Performance), (ii) οι ΒΔ που διαχειρίζεται ο εξυπηρετητής, (iii) ο παράθυρο σύνταξης εντολών και (iv) το παράθυρο προβολής των αποτελεσμάτων.



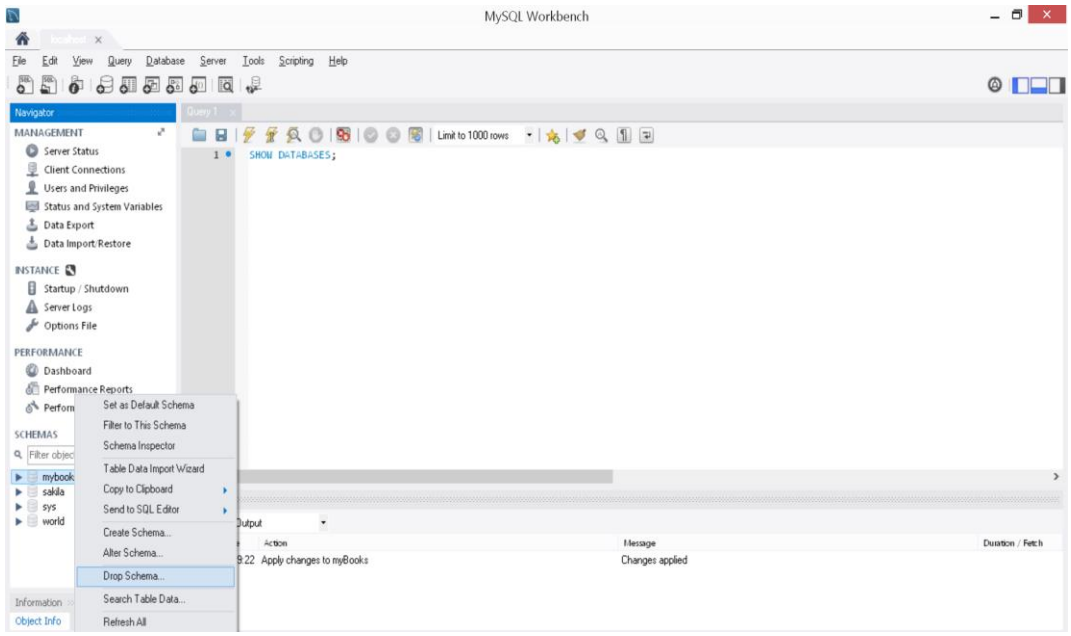
Εικόνα 1.24: *Βασικά μέρη γραφικού περιβάλλοντος Workbench.*

Για τη δημιουργία μίας βάσης δεδομένων είτε θα εκτελέσουμε την εντολή CREATE DATABASE στο παράθυρο εντολών ή θα επιλέξουμε γραφικά το create new schema, θα δώσουμε το όνομα της ΒΔ και apply από το γραφικό περιβάλλον.



Εικόνα 1.25: *Δημιουργία Βάσης Δεδομένων.*

Για τη διαγραφή μίας ΒΔ είτε θα εκτελεστεί η εντολή DROP DATABASE στο παράθυρο εντολών ή θα επιλεγεί η ΒΔ από τον τομέα των βάσεων δεδομένων και στη συνέχεια θα επιλεγεί το DROP Schema όπως φαίνεται και στο παρακάτω σχήμα.



Εικόνα 1.26: *Διαγραφή Βάσης Δεδομένων.*

1.7 Εργαστηριακή Άσκηση

Στην παρούσα φάση δε θα χρειαστεί να υλοποιήσετε κάποια εργαστηριακή άσκηση. Είναι επιθυμητό όμως να: (i) διαβάσετε την αναφερόμενη βιβλιογραφία και (ii) κάνετε εγκατάσταση του MySQL server καθώς και του Workbench σε κάποιο μηχάνημα.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαια 1,2.
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαια 1,2.
MySQL Client – Σύνδεση με MySQL server
http://dev.mysql.com/doc/refman/5.7/en/connecting-disconnecting.html
http://dev.mysql.com/doc/refman/5.7/en/entering-queries.html
MySQL Workbench.
http://dev.mysql.com/doc/workbench/en/wb-getting-started-tutorial.html
https://en.wikipedia.org/wiki/CODASYL
https://en.wikipedia.org/wiki/Integrated_Data_Store
https://en.wikipedia.org/wiki/COBOL
https://en.wikipedia.org/wiki/Integrated_Data_Store
https://en.wikipedia.org/wiki/Charles_Bachman
https://en.wikipedia.org/wiki/Edgar_F._Codd
https://en.wikipedia.org/wiki/IBM_Information_Management_System
https://en.wikipedia.org/?title=SQL

Κεφάλαιο 2 Στοιχεία Θεωρίας Σχεδιασμού Σχεσιακών Βάσεων Δεδομένων

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν βασικά στοιχεία όσον αφορά στο σχεδιασμό Βάσεων Δεδομένων. Πιο συγκεκριμένα θα γίνει επισκόπηση του σχεσιακού μοντέλου καθώς και του διαγράμματος Οντοτήτων-Συσχετίσεων ΟΣ (Entity Relationship Model – ER) για τη μοντελοποίηση των δεδομένων στο εννοιολογικό επίπεδο. Θα συζητηθεί το επεκτεταμένο μοντέλο ΟΣ (Extended Entity Relationship Model EER) και οι κανόνες μετατροπής από EER σε πίνακες. Τέλος θα παρουσιαστούν παραδείγματα και θα ακολουθήσουν ασκήσεις προς επίλυση.

Προαπαιτούμενη γνώση
Δεν προαπαιτείται κάποια ιδιαίτερη γνώση.

2.1 Επισκόπηση Σχεσιακού Μοντέλου

Στο σχεσιακό μοντέλο τα δεδομένα αναπαρίστανται ως *σχέσεις* (*relations*) δηλ. ως πίνακες. Κάθε σχέση έχει μία ονομασία και ορισμένα *γνωρίσματα* (*attributes*) που δεν είναι τίποτα άλλο παρά τα πεδία ή αλλιώς οι στήλες του πίνακα. Για παράδειγμα σε μία σχέση Πελάτης τα γνωρίσματα μπορεί να είναι: ΑΦΜ, όνομα, επώνυμο, διεύθυνση. Οι επιτρεπτές τιμές κάθε γνωρίσματος απαρτίζουν το *πεδίο ορισμού* του γνωρίσματος. Για παράδειγμα στη σχέση Πελάτης το γνώρισμα όνομα είναι λογικό να υποθέσουμε ότι θα είναι αλφαριθμητικό (συγκεκριμένοι τύποι που υποστηρίζονται από την SQL θα συζητηθούν σε επόμενα κεφάλαια). Αντίθετα το ΑΦΜ ίσως να θελήσουμε να το αναπαραστήσουμε ως ακέραιο και όχι ως αλφαριθμητικό.

Το *σχήμα μιας σχέσης* είναι το όνομα της σχέσης μαζί με τα ονόματα των γνωρισμάτων και τα πεδία ορισμού τους. Για παράδειγμα το σχήμα της σχέσης Πελάτης θα μπορούσε να είναι (χρησιμοποιώντας πιο γενικούς τύπους):

Πελάτης (ΑΦΜ: int, όνομα: string, επώνυμο string, διεύθυνση string)

και χρησιμοποιώντας τους τύπους της SQL:

Πελάτης (ΑΦΜ: INT, όνομα: CHAR(20), επώνυμο CHAR(20), διεύθυνση VARCHAR(60))

με το CHAR(20) να υποδηλώνει στατικό πίνακα χαρακτήρων (20 θέσεων στην περίπτωση των γνωρισμάτων όνομα και επώνυμο) και το VARCHAR να δηλώνει δυναμικό πίνακα χαρακτήρων (60 θέσεων για το γνώρισμα διεύθυνση).

Εφεξής στο σχήμα των σχέσεων που θα παρουσιάζουμε θα παραλείπουμε χάριν συντομίας τα πεδία ορισμού και θα αναφέρουμε μόνο τα ονόματα των γνωρισμάτων.

Πλειάδα (*tuple*) σε μία σχέση είναι στην ουσία μία εγγραφή (γραμμή) του πίνακα, στην οποία έχουν εκχωρηθεί τιμές στα γνωρίσματα της σχέσης. Για παράδειγμα μία πλειάδα της σχέσης Πελάτης (ΑΦΜ, όνομα, επώνυμο, διεύθυνση) μπορεί να είναι η:
<088673454, “Χρήστος”, “Παπαχρήστος”, “Κολοκοτρώνη 37, 35100, Λαμία”>
ενώ δεν μπορεί να είναι οι:
<“ab4562jd”, “Χρήστος”, “Παπαχρήστος”, “Κολοκοτρώνη 37, 35100, Λαμία”> (το ΑΦΜ δε θα έπρεπε να είναι αλφαριθμητικό)
<“ab4562jd”, “Παπαχρήστος”, “Κολοκοτρώνη 37, 35100, Λαμία”> (υπάρχουν λιγότερα πεδία)
<“ab4562jd”, “Χρήστος”, “Παπαχρήστος”, “Κολοκοτρώνη 37, 35100, Λαμία”, 30/01/15> (υπάρχουν περισσότερα πεδία).

Ως *στιγμιότυπο* ή *περίσταση* (*instance*) μιας σχέσης ορίζουμε οποιοδήποτε υποσύνολο του καρτεσιανού γινομένου των πεδίων ορισμού των γνωρισμάτων ή πιο απλά οποιοδήποτε σύνολο έγκυρων πλειάδων.

Μία *σχεσιακή ΒΔ* είναι ένα σύνολο από σχέσεις που πληρούν τους κανόνες του σχεσιακού μοντέλου (αναφερόμαστε σε αυτούς στη συνέχεια). Ορίζουμε σαν *σχήμα μιας σχεσιακής ΒΔ* το σύνολο των σχημάτων των σχέσεων που την απαρτίζουν.

2.1.1 Κανόνες σχεσιακού μοντέλου και περιορισμοί

Στο σχεσιακό μοντέλο γίνονται οι ακόλουθες βασικές υποθέσεις:

Κάθε σχέση πρέπει να έχει ξεχωριστό όνομα.

- Τα γνωρίσματα μιας σχέσης πρέπει να φέρουν ξεχωριστό όνομα.
- Δεν παίζουν ρόλο: (α) η σειρά αναγραφής των γνωρισμάτων μίας σχέσης, (β) η σειρά αναγραφής των σχέσεων/σχημάτων των σχέσεων σε μία ΒΔ/σχήμα ΒΔ και (γ) δεν παίζει ρόλο η σειρά των πλειάδων μέσα σε μία σχέση.
- Απαγορεύεται στην περίσταση μιας σχέσης να υπάρχουν δύο ίδιες πλειάδες.
- Κάθε γνώρισμα μιας πλειάδας μπορεί να λαμβάνει μία το πολύ τιμή (ή απουσία τιμής δηλ. NULL) η οποία θα πρέπει να ανήκει στο πεδίο ορισμού του γνωρίσματος. Με άλλα λόγια απαγορεύονται γνωρίσματα πολλαπλών τιμών. Όταν πληρείται αυτή η συνθήκη θα λέμε ότι η σχέση ανήκει στην πρώτη κανονική μορφή (1NF).

Το σχεσιακό μοντέλο υπαγορεύει τους ακόλουθους δύο βασικούς *περιορισμούς* (*constraints*): (α) *ακεραιότητας οντοτήτων* και (β) *αναφορικής ακεραιότητας*. Πριν εξηγήσουμε τους περιορισμούς θα πρέπει να δοθούν κάποιοι επιπλέον ορισμοί. Ως παράδειγμα χρησιμοποιούμε τη σχέση Πελάτης (ΑΦΜ, όνομα, επώνυμο, διεύθυνση).

- *Κλειδί (key)* μιας σχέσης είναι ένα οποιοδήποτε σύνολο γνωρισμάτων της. Για παράδειγμα (επώνυμο, διεύθυνση) ή (όνομα) ή (ΑΦΜ, όνομα, διεύθυνση) κλπ.
- *Υπερκλειδί (superkey)* είναι ένα κλειδί που διαχωρίζει με μοναδικό τρόπο τις πλειάδες της σχέσης. Με άλλα λόγια δεν υπάρχουν δύο πλειάδες που με ίδιες τιμές στα γνωρίσματα του υπερκλειδιού. Κατά συνέπεια ξέροντας τις τιμές στα γνωρίσματα του υπερκλειδιού μπορούμε να γνωρίζουμε σε ποια πλειάδα αναφερόμαστε. Για παράδειγμα το (ΑΦΜ, όνομα, διεύθυνση) είναι ένα υπερκλειδί, ενώ το (όνομα, επώνυμο) δεν είναι καθώς μπορεί να υπάρχουν συνωνυμίες.
- *Υποψήφιο κλειδί (candidate key)* είναι ένα υπερκλειδί που έχει την ιδιότητα του αμείωτου. Η ιδιότητα αυτή λέει ότι αφαιρώντας οποιοδήποτε γνώρισμα από το υπερκλειδί το υπερκλειδί παύει να είναι υπερκλειδί. Για παράδειγμα το (ΑΦΜ, όνομα διεύθυνση) δεν είναι υποψήφιο κλειδί καθώς δεν έχει την ιδιότητα του αμείωτου, αφού μπορώ να αφαιρέσω πχ. το όνομα και αυτό που απομένει δηλ. το (ΑΦΜ, διεύθυνση) εξακολουθεί να είναι υπερκλειδί. Αντίθετα το (ΑΦΜ) είναι υποψήφιο καθώς έχει ένα μόνο πεδίο που διαχωρίζει με μοναδικό τρόπο τις πλειάδες (αφαιρώντας το προκύπτει το κενό σύνολο που δεν μπορεί να παίξει το ρόλο του διαχωριστή). Αν κάναμε την κάπως «εξεζητημένη» υπόθεση ότι δεν υπάρχει περίπτωση να συμπίπτουν δύο πελάτες και στο ον/μο και στη διεύθυνση, τότε ένα δεύτερο υποψήφιο κλειδί θα ήταν το: (όνομα, επώνυμο, διεύθυνση).
- *Πρωτεύον κλειδί (primary key)* είναι το υποψήφιο κλειδί που τελικά επιλέγουμε για να παίξει το ρόλο του διαχωριστή. Στο παράδειγμα μπορεί ή να επιλεγεί το (ΑΦΜ) ή το (όνομα, επώνυμο, διεύθυνση). Για λόγους που θα γίνουν καλύτερα αντιληπτοί στη συνέχεια του εργαστηρίου είναι προτιμότερο το πρωτεύον κλειδί να απαρτίζεται από ένα γνώρισμα κατά προτίμηση απλού τύπου πχ. ένας ακέραιος. Έτσι στο παράδειγμά μας, η επιλογή του (ΑΦΜ) είναι προτιμητέα έναντι του (όνομα, επώνυμο, διεύθυνση).
- *Εναλλακτικό κλειδί* είναι ένα υποψήφιο κλειδί που δεν επιλέχθηκε ως πρωτεύον. Στο παράδειγμά μας, μετά την επιλογή του (ΑΦΜ) ως πρωτεύον το (όνομα, επώνυμο, διεύθυνση) γίνεται εναλλακτικό κλειδί.

Σημείωση: Εξαιτίας του γεγονότος ότι επαναλαμβανόμενες πλειάδες απαγορεύονται στο σχεσιακό μοντέλο, σε κάθε σχέση μπορεί να σχηματιστεί πρωτεύον κλειδί (στην ακραία περίπτωση θα απαρτίζεται από όλα τα γνωρίσματα της σχέσης).

Θεωρείστε ότι το σχήμα μιας ΒΔ περιλαμβάνει τις εξής σχέσεις:

Πελάτης (ΑΦΜ, όνομα, επώνυμο, διεύθυνση)

Τιμολόγιο (id, ποσό, ημ/νια, *ΑΦΜ*)

Η σχέση τιμολόγιο έχει ένα μοναδικό κωδικό τιμολογίου (το id), και την πληροφορία τι ποσό κόπηκε, πότε και σε ποιόν πελάτη.

Ορίζουμε ως *ξένο κλειδί* (*foreign key*) σε μία σχέση, ένα κλειδί (σύνολο γνωρισμάτων) που είναι πρωτεύον κλειδί σε κάποια άλλη σχέση. Στο παραπάνω σχήμα εύκολα γίνεται αντιληπτό ότι το ΑΦΜ στη σχέση Τιμολόγιο είναι ξένο κλειδί καθώς είναι το πρωτεύον κλειδί της σχέσης Πελάτης. Παρατηρήστε ότι υπογραμμισμένα στο σχήμα είναι τα πρωτεύοντα κλειδιά των δύο σχέσεων και με italics το ξένο κλειδί. Εφεξής όπου είναι δυνατό θα ακολουθούμε αυτή τη σύμβαση ώστε να είναι εύκολα αντιληπτός από το σχήμα ο ρόλος κάποιων γνωρισμάτων.

Είμαστε τώρα σε θέση να εξηγήσουμε τους περιορισμούς ακεραιότητας οντοτήτων και αναφορικής ακεραιότητας που προαναφέρθηκαν.

- *Ακεραιότητα οντοτήτων* (*entity integrity*). Τα γνωρίσματα που απαρτίζουν το πρωτεύον κλειδί θα πρέπει να λαμβάνουν τιμές (δηλ. να μην είναι NULL) σε όλες τις πλειάδες της σχέσης. Να σημειώσουμε ότι η NULL τιμή δηλώνει απουσία τιμής. Με την ακεραιότητα οντοτήτων αποφεύγουμε την αδυναμία στο διαχωρισμό δύο πλειάδων με NULL τιμές στο πρωτεύον κλειδί.
- *Αναφορική ακεραιότητα (referential integrity).* Δεν μπορεί να έχουμε τιμή σε ξένο κλειδί η οποία δεν υπάρχει στη σχέση στην οποία αναφέρεται δηλ. είναι πρωτεύον κλειδί. Στο παράδειγμα μας η αναφορική ακεραιότητα επιβάλει να μην υπάρχει πλειάδα στη σχέση Τιμολόγιο με τιμή στο ΑΦΜ που δεν υπάρχει σε καμία πλειάδα της σχέσης Πελάτης. Αυτός ο περιορισμός είναι εν πολλοίς αναμενόμενος και «φυσιολογικός», καθώς δε θα επιθυμούσαμε στην πράξη να κόψουμε τιμολόγιο σε ΑΦΜ για το οποίο δεν υπάρχουν επιπλέον στοιχεία.

2.1.2 Σχεσιακή Άλγεβρα

Στο σχεσιακό μοντέλο είναι ορισμένη η *σχεσιακή άλγεβρα*, που ενέχει πράξεις των οποίων οι τελεστές και τα αποτελέσματα είναι σχέσεις. Ο ρόλος της άλγεβρας αυτής είναι να προσφέρει ένα μαθηματικό τρόπο (βασιζόμενο στη θεωρία συνόλων) σύνταξης *ερωτημάτων* (*queries*) σε μία σχεσιακή ΒΔ. Η αναλυτική παρουσίαση των πράξεων της σχεσιακής άλγεβρας ξεφεύγει από το σκοπό του παρόντος συγγράμματος. Θα πρέπει όμως να πούμε ότι η γλώσσα SQL που είναι και το κύριο αντικείμενο του συγγράμματος στην ουσία υλοποιεί τη σχεσιακή άλγεβρα.

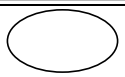
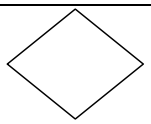
2.2 Επισκόπηση Διαγράμματος Οντοτήτων-Συσχετίσεων

Το διάγραμμα οντοτήτων-συσχετίσεων (εφεξής ER) χρησιμοποιείται στο σχεδιασμό του σχήματος της ΒΔ δοθέντος κάποιων αρχικών απαιτήσεων.

2.2.1 Βασικά στοιχεία του ER

Στο ER η πληροφορία που θέλουμε να περιλαμβάνει η ΒΔ αναπαρίσταται ως: *οντότητες, γνωρίσματα* και *συσχετίσεις*. Ως οντότητα εκλαμβάνεται ένα αντικείμενο το οποίο έχει επιπλέον ιδιότητες (γνωρίσματα) τα οποία επιθυμούμε να αποθηκεύονται στη ΒΔ. Συνήθως σε μία περιγραφή απαιτήσεων οντότητες είναι ουσιαστικά τα οποία έχουν επιπλέον χαρακτηριστικά. Ομοειδείς οντότητες απαρτίζουν ένα *σύνολο οντοτήτων*. Ως συσχέτιση αναπαρίσταται κάποια αλληλεπίδραση μεταξύ δύο (ή περισσοτέρων) οντοτήτων. Συνήθως σε μια περιγραφή απαιτήσεων οι συσχετίσεις βρίσκονται στο κείμενο ως ρήματα δηλωτικά της επίδρασης μίας οντότητας σε κάποια άλλη. Ομοειδείς συσχετίσεις απαρτίζουν ένα *σύνολο συσχετίσεων*. Οι βασικές αναπαραστάσεις που χρησιμοποιούνται στο ER διάγραμμα εμφανίζονται στον Πίνακα 2.1.

Πίνακας 2.1: Δομικά στοιχεία του ER.

Σύνολο οντοτήτων	
Γνώρισμα	
Σύνολο συσχετίσεων	

Στη συνέχεια παρουσιάζουμε ένα παράδειγμα σχεδιασμού ER διαγράμματος από προδιαγραφές απαιτήσεων. Κατά τη διάρκεια της βήμα προς βήμα σχεδίασης αναφέρουμε και επιπλέον στοιχεία του ER.

2.2.2 Παράδειγμα χρήσης

Έστω ότι μας δίνεται η ακόλουθη περιγραφή απαιτήσεων:

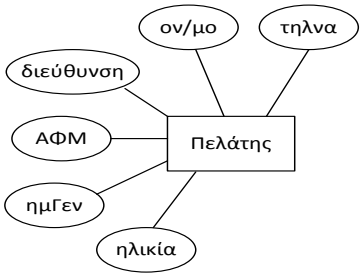
Σχεδιάστε μια ΒΔ για μία τράπεζα. Συγκεκριμένα θέλουμε να κρατάμε στοιχεία για τους πελάτες της: ΑΦΜ, ον/μο, διεύθυνση, ένα ή περισσότερα τηλέφωνα, ημ/νια γέννησης και ηλικία. Κάθε πελάτης μπορεί να έχει έναν ή περισσοτέρους λογαριασμούς. Σε κάθε λογαριασμό μας ενδιαφέρει να κρατάμε την ημ/νια δημιουργίας του και το υπόλοιπο του. Η τράπεζα επίσης χορηγεί δάνεια για τα οποία θέλουμε να κρατάμε στοιχεία όπως: σε ποιον χορηγήθηκε το δάνειο, το ποσό του δανείου, τη διάρκεια αποπληρωμής και το επιτόκιο.

Πριν ξεκινήσουμε τη σχεδίαση θα πρέπει να αναφέρουμε ότι οι περιγραφές απαιτήσεων από τους δυνητικούς αγοραστές μιας ΒΔ, σπανίως είναι πλήρεις και ο σαφής καθορισμός απαιτήσεων συχνά είναι επίπονος. Το ανωτέρω παράδειγμα ενέχει (εσκεμμένα) ασάφειες όπως θα φανεί και στην πορεία.

Ξαναδιαβάζοντας τις απαιτήσεις πρώτα προσπαθούμε να αντιληφθούμε ποια σύνολα οντοτήτων υπάρχουν. Σύμφωνα με τους ορισμούς του Κεφ. 2.2.1 ως οντότητες μοντελοποιούμε ουσιαστικά που έχουν επιπλέον χαρακτηριστικά που τα προσδιορίζουν. Κατά συνέπεια ο πελάτης είναι μία οντότητα και θα πρέπει να μοντελοποιήσουμε στο ER διάγραμμα όλες τις ομοειδείς οντότητες ως ένα σύνολο οντοτήτων πχ., με το όνομα Πελάτες. Χάριν ευκολίας, εφεξής θα χρησιμοποιούμε κάποιες φορές τον όρο οντότητα και όταν θέλουμε να υποδηλώσουμε σύνολο οντοτήτων οδηγώντας σε ονόματα που είναι στον ενικό (Πελάτης αντί για Πελάτες). Ομοίως και για τις συσχετίσεις. Έτσι έχουμε την αναπαράσταση της οντότητας Πελάτης που εμφανίζεται στην Εικόνα 2.1.

Παρατηρήστε ότι τα γνωρίσματα συνδέονται μέσω απλών γραμμών με την οντότητα στην οποία ανήκουν (δηλ. την οποία χαρακτηρίζουν). Όπως και στο σχεσιακό μοντέλο έτσι και στο ER διάγραμμα κάθε οντότητα έχει ένα *πρωτεύον κλειδί* που παίζει το ρόλο του διαχωριστή (πχ. στο Πελάτης το ΑΦΜ). Τα γνωρίσματα που απαρτίζουν το πρωτεύον κλειδί τα υπογραμμίζουμε για να είναι εύκολα αναγνωρίσιμα.

Ξεχωρίζουμε τα γνωρίσματα σε: απλά, παραγόμενα, σύνθετα και πολλαπλών τιμών. Ως *απλά* θεωρούμε τα *γνωρίσματα* τα οποία δεν εμπίπτουν σε κάποια από τις άλλες κατηγορίες. Για τα απλά γνωρίσματα η σχεδίαση είναι όπως παραπάνω.



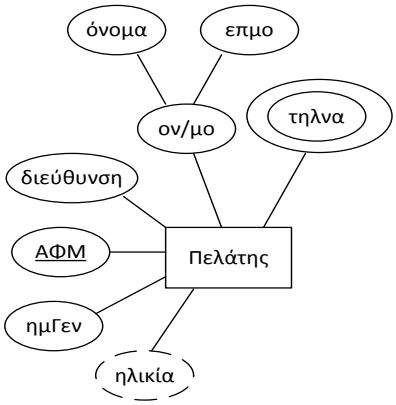
Εικόνα 2.1: *Αρχική σχεδίαση της οντότητας Πελάτης.*

Παραγόμενα γνωρίσματα είναι αυτά των οποίων η τιμή μπορεί να καθοριστεί από τις τιμές άλλων γνωρισμάτων. Στο παράδειγμα η ηλικία είναι παραγόμενο γνώρισμα καθώς γνωρίζοντας την ημερομηνία γέννησης (ημΓεν) μπορούμε χρησιμοποιώντας την τρέχουσα ημ/νια να την υπολογίσουμε. Τα γνωρίσματα αυτά τα συμβολίζουμε με διακεκομμένη έλλειψη.

Σύνθετα είναι τα *γνωρίσματα* τα οποία μπορούν να αναλυθούν περαιτέρω σε υπογνωρίσματα. Τα αναπαριστούμε βάζοντας τα υπογνωρίσματα στα οποία αναλύονται (με ελλείψεις) να συνδέονται με το σύνθετο γνώρισμα. Στο παράδειγμα τόσο το ον/μο όσο και η διεύθυνση και η ημΓεν μπορεί να ειδωθούν σαν σύνθετα. Το ον/μο μπορεί να αναλυθεί σε όνομα και επμο, η διεύθυνση σε οδό, αριθμό, ΤΚ, πόλη και η ημΓεν σε ημέρα, μήνα, έτος. Αντίθετα τα: ΑΦΜ, τηλνα και ηλικία δε διαιρούνται περαιτέρω και οι τιμές τους είναι ατομικές (αδιαίρετες), πχ. ηλικία=20. Για λόγους απλότητας στο σχεδιασμό δεν επιλέγουμε πάντα όσα γνωρίσματα δυνητικά μπορεί να είναι σύνθετα να είναι όντως. Αντίθετα αναπαριστούμε ως σύνθετα, γνωρίσματα για τα οποία περιμένουμε να έχουμε ερωτήματα πάνω στα υπογνωρίσματα στα οποία αναλύονται. Για παράδειγμα αν πρέπει η ΒΔ να μπορεί να απαντά στο ακόλουθο ερώτημα: «βρες τους πελάτες που το όνομά τους είναι Δημήτρης ή Δήμητρα», πιθανώς για να σταλεί ευχετήρια κάρτα στις 26/10, τότε είναι προτιμητέο το ον/μο να αναλυθεί σε όνομα και επμο. Αντίθετα αν αναμένουμε το γνώρισμα να χρησιμοποιείται ως έχει, πχ. έχουμε ερωτήματα μόνο της μορφής: «Εμφάνισε τη διεύθυνση του πελάτη Χ» και όχι «Εμφάνισε την οδό στην οποία μένει ο πελάτης Χ», τότε δεν προσφέρει στην πράξη τίποτα η μοντελοποίηση του γνωρίσματος ως σύνθετου. Μία άλλη περίπτωση καταδεικνύεται με το γνώρισμα ημΓεν. Ίσως είναι αναμενόμενο να έχω ερώτημα της μορφής: «Βρες το έτος γέννησης του πελάτη Χ» κάτι που οδηγεί στη σύνθετη αναπαράσταση του ημΓεν. Θεωρητικά ο σχεδιασμός του ER είναι ανεξάρτητος από συγκεκριμένο RDBMS στο οποίο θα υλοποιηθεί η ΒΔ. Παρόλα αυτά τόσο το πρότυπο της SQL όσο και οι περισσότερες υλοποιήσεις του (τα περισσότερα RDBMS) προσφέρουν πληθώρα τύπων και συναρτήσεων για επεξεργασία ημερομηνιών και χρονοσφραγίδων (timestamps). Ως αποτέλεσμα θα ήταν πρακτικά προτιμότερη η αναπαράσταση του ημΓεν ως απλού γνωρίσματος.

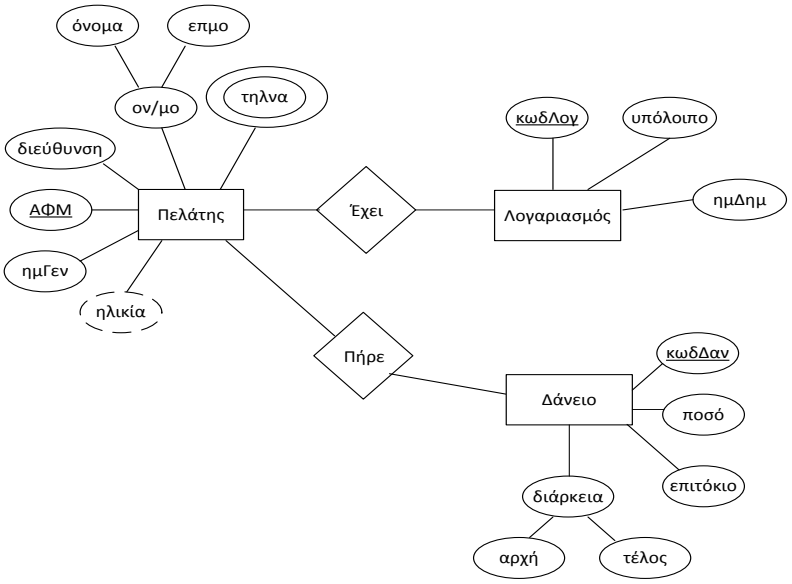
Γνωρίσματα πολλαπλών τιμών είναι όσα μπορούν να πάρουν περισσότερες από μία τιμές. Τα διαχωρίζουμε βάζοντας διπλή αντί για απλή έλλειψη. Στο παράδειγμα το τηλνα είναι ένα τέτοιο γνώρισμα καθώς από τις απαιτήσεις προκύπτει ότι για ένα συγκεκριμένο πελάτη θα καταχωρήσω ίσως περισσότερα του ενός τηλέφωνα, πχ., σταθερό οικίας, κινητό, σταθερό εργασίας, κινητό άλλου οικογενειακού μέλους.

Συνοψίζοντας τα παραπάνω, το κομμάτι του ER της Εικόνας 2.1 ξαναγράφεται όπως φαίνεται στην Εικόνα 2.2.



Εικόνα 2.2: *Τελική σχεδίαση της οντότητας Πελάτης.*

Συνεχίζοντας την ανάλυση των απαιτήσεων και το σχεδιασμό του ER, διαπιστώνουμε ότι τόσο οι τραπεζικοί λογαριασμοί όσο και τα δάνεια έχουν δικά τους χαρακτηριστικά άρα μάλλον πρέπει να αναπαρασταθούν ως οντότητες, πολύ περισσότερο αφού εύκολα διαπιστώνεται η συσχετίσεις τους με τους πελάτες της τράπεζας. Οδηγούμαστε έτσι στο παρακάτω ER διάγραμμα της Εικόνας 2.3.



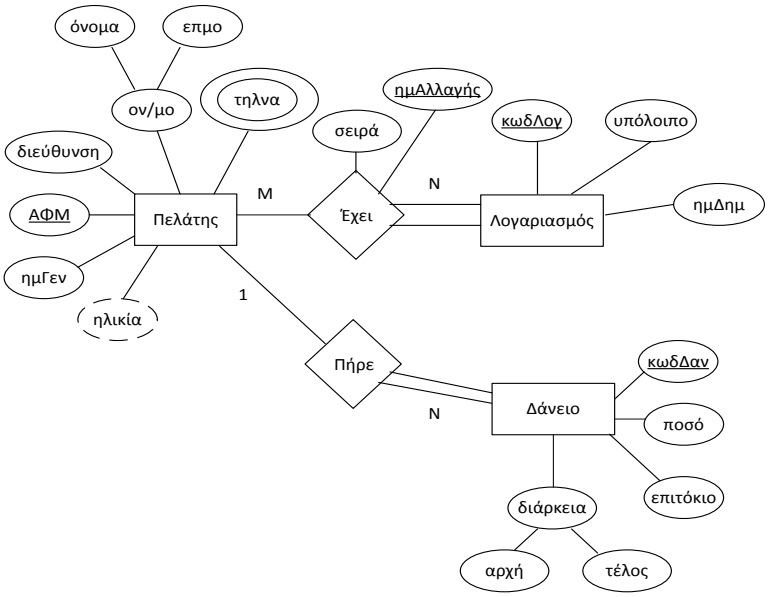
Εικόνα 2.3: *Συσχετίσεις μεταξύ των οντοτήτων Πελάτης, Λογαριασμός και Δάνειο.*

Παρατηρήστε ότι σύμφωνα με την αρχική περιγραφή τα γνωρίσματα ενός λογαριασμού είναι το υπόλοιπο και η ημερομηνία δημιουργίας (η περιγραφή απαιτήσεων δεν είναι πλήρης). Μόνο όμως με τα παραπάνω δύο γνωρίσματα δεν μπορεί να σχηματιστεί πρωτεύον κλειδί καθώς είναι πιθανό δύο λογαριασμοί να έχουν δημιουργηθεί την ίδια ημ/νια και να έχουν το ίδιο υπόλοιπο. Για αυτό το λόγο στο ER προσθέσαμε ένα επιπλέον γνώρισμα, το κωδΛογ, για να παίξει το ρόλο του πρωτεύοντος κλειδιού και το ίδιο συνέβη στο Δάνειο. Παρατηρήστε επίσης ότι Λογαριασμός και Δάνειο συσχετίζονται με τον Πελάτη μέσω ξεχωριστών συσχετίσεων (Έχει και Πήρε) που αναπαριστούν σε ποιόν πελάτη ανήκει ποιος λογαριασμός και ποιος πελάτης πήρε ποιο δάνειο.

Οι συσχετίσεις χαρακτηρίζονται περαιτέρω στο ER διάγραμμα βάσει: *συμμετοχής* και *πληθικότητας* (*cardinality*). Όσον αφορά στη *συμμετοχή* ελέγχουμε αν όλες οι οντότητες ενός συνόλου οντοτήτων μετέχουν σε μια συγκεκριμένη συσχέτιση. Αν ναι τότε υπάρχει υποχρεωτική συμμετοχή του συνόλου οντοτήτων στη συσχέτιση και το συμβολίζουμε με διπλή γραμμή. Στο παράδειγμα όσον αφορά στη συσχέτιση Έχει ο Λογαριασμός έχει υποχρεωτική συμμετοχή αφού δεν μπορεί να υπάρξει λογαριασμός χωρίς ιδιοκτήτη, ενώ ο Πελάτης δεν έχει

υποχρεωτική συμμετοχή καθώς ένας πελάτης μπορεί να έχει πάρει δάνειο αντί να έχει λογαριασμό (στην πράξη θα χρειάζονταν διευκρινήσεις ως προς τις απαιτήσεις). Αντίστοιχα ισχύουν και για τη συσχέτιση Πήρε.

Όσον αφορά στην *πληθικότητα* χαρακτηρίζουμε τη συμμετοχή ενός συνόλου οντοτήτων σε μια συσχέτιση, διερωτώμενοι αν μία οντότητα του συνόλου μπορεί να μετάσχει μία ή και περισσότερες φορές στη συσχέτιση. Στο παράδειγμα όσον αφορά στη συσχέτιση Έχει, ένας πελάτης μπορεί να έχει πολλούς λογαριασμούς και ομοίως ένας λογαριασμός μπορεί να ανήκει σε πολλούς πελάτες. Σε αυτήν την περίπτωση λέμε ότι η συσχέτιση είναι *πολλά προς πολλά* και αναγράφουμε στις ακμές της Μ και Ν για να το δηλώσουμε. Θεωρήστε (δεν είναι ξεκάθαρο από τις απαιτήσεις) ότι ένα δάνειο μπορεί να έχει έναν και μόνο ιδιοκτήτη. Σε αυτήν την περίπτωση και αφού είναι λογικό να υποθέσουμε ότι ένας πελάτης μπορεί να έχει πολλά δάνεια η συσχέτιση Πήρε είναι συσχέτιση *ένα προς πολλά* και τη συμβολίζουμε αναγράφοντας 1 στην ακμή του Πελάτη και Ν στην ακμή που ανήκει στο Δάνειο. Κατ' αναλογία με τις συσχετίσεις Μ-Ν και 1-Ν, υπάρχουν και συσχετίσεις που μπορεί να είναι 1-1, δηλώνοντας ότι κάθε οντότητα στα δύο σύνολα που συσχετίζονται, συσχετίζεται με το πολύ μία ακριβώς οντότητα από το άλλο σύνολο. Στο παράδειγμα δεν υπάρχει τέτοια συσχέτιση.



Εικόνα 2.4: *Περιορισμοί πληθικότητας και συμμετοχής στις συσχετίσεις μεταξύ των οντοτήτων Πελάτης, Λογαριασμός και Δάνειο.*

Εδώ θα θέλαμε να τονίσουμε ότι όπως οι οντότητες έχουν γνωρίσματα, έτσι μπορεί να έχουν και οι συσχετίσεις. Ας υποθέσουμε ότι επεκτείνουμε την περιγραφή των απαιτήσεων ως εξής: *σε κάθε λογαριασμό με πολλούς ιδιοκτήτες θέλουμε να καταγράφουμε τη σειρά των ιδιοκτητών*. Με μία πρώτη σκέψη θα αρκούσε να προσθέταμε ένα γνώρισμα (σειρά). Το ερώτημα είναι που; Αν προστεθεί στο Λογαριασμό θα σήμαινε ότι για έναν συγκεκριμένο λογαριασμό όλοι οι ιδιοκτήτες του θα είχαν την ίδια σειρά. Από την άλλη αν προστεθεί στον Πελάτη θα σήμαινε ότι σε όλους τους λογαριασμούς στους οποίους συμμετέχει ένας συγκεκριμένος πελάτης συμμετέχει πάντα με την ίδια σειρά. Αν το σκεφτούμε καλά, η σειρά αφορά ζευγάρι πελάτη, λογαριασμού, κατά συνέπεια είναι γνώρισμα της συσχέτισης Έχει. Επιπλέον, θα θέλαμε να σημειώσουμε ότι είναι πιθανό το ίδιο ζευγάρι οντοτήτων να συσχετίζεται πολλές φορές μέσω της ίδιας συσχέτισης στην οποία περίπτωση κάποιο/α από τα γνωρίσματα της συσχέτισης θα πρέπει να παίξουν το ρόλο του διαχωριστή και τα υπογραμμίζουμε. Για παράδειγμα αν υποθέσουμε ότι επιτρέπονταν να αλλάξει η σειρά των ιδιοκτητών ενός λογαριασμού, τότε ένας πελάτης μπορεί να συσχετιστεί με τον ίδιο λογαριασμό πολλές φορές (όσες φορές άλλαξε η σειρά του). Για να αναπαρασταθεί αυτή η πληροφορία θα μπορούσαμε στη συσχέτιση Έχει να προσθέσουμε ένα ακόμα πεδίο ημΑλλαγής που θα δηλώνει πότε ξεκίνησε να ισχύει η συγκεκριμένη σειρά. Στην Εικόνα 2.4 φαίνεται το νέο ER διάγραμμα.

2.2.3 Αδύναμες οντότητες

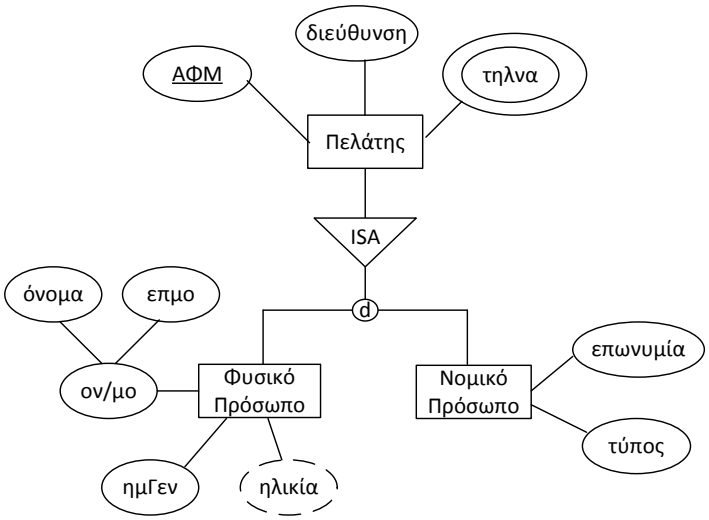
Κάποιες φορές μπορεί να έχουμε οντότητες οι οποίες δεν μπορούν βάσει των γνωρισμάτων τους να σχηματίσουν πρωτεύον κλειδί αλλά μπορούν δανειζόμενες το πρωτεύον κλειδί από κάποια άλλη. Ας υποθέσουμε ότι θέλαμε να επεκτείνουμε τις απαιτήσεις μας ως εξής: θέλουμε για κάθε λογαριασμό να καταχωρούμε τις πράξεις που έχουν γίνει. Κάθε πράξη ενός λογαριασμού έχει έναν αύξον αριθμό (σειρά) με την οποία συνέβη, συγκεκριμένο τύπο (πχ. ανάληψη, κατάθεση κλπ.), ημ/νια που έγινε και το ποσό που αφορούσε. Το ερώτημα είναι πως θα πρέπει να αναπαρασταθούν οι πράξεις. Μία πρώτη ανάγνωση είναι ότι εφόσον αφορούν λογαριασμούς θα πρέπει να είναι γνώρισμα της Λογαριασμός. Για την ακρίβεια να είναι γνώρισμα πολλαπλών τιμών και σύνθετο. Μία άλλη εναλλακτική είναι να αναπαρασταθεί ως οντότητα.

Το παραπάνω δίλλημα δηλ. οντότητα ή γνώρισμα είναι γενικότερο και δεν αφορά μόνο τη συγκεκριμένη περίπτωση. Συμμετρικά θα μπορούσε κανείς να μοντελοποιήσει το ον/μο αντί για γνώρισμα του Πελάτης σαν ξεχωριστή οντότητα που θα συσχετίζεται με το Πελάτης. Δεν υπάρχει αυστηρός κανόνας για το τι είναι προτιμότερο. Ο ευρηστικός κανόνας είναι ότι αν μας ενδιαφέρει κάτι από μόνο του, προτιμάμε να δημιουργήσουμε οντότητα, αλλιώς αν μας ενδιαφέρει μόνο ως χαρακτηριστικό κάποιας άλλης οντότητας προτιμάμε την αναπαράσταση ως γνώρισμα. Σημειώνουμε πάντως ότι η αναπαράσταση ως οντότητα προσφέρει τη δυνατότητα συμμετοχής σε συσχετίσεις ενώ ένα γνώρισμα δεν μπορεί να συμμετάσχει.

Επανερχόμενοι στο παράδειγμα είναι λογικό να υποθέσουμε ότι οι πράξεις ενός λογαριασμού έχουν από μόνες τους ενδιαφέρον στην οποία περίπτωση και μοντελοποιούνται ως ξεχωριστή οντότητα συσχετιζόμενη με τη Λογαριασμός και με γνωρίσματα, βάσει της περιγραφής: Α/Α, τύπο, ημ/νια και ποσό. Το ερώτημα είναι σχηματίζεται πρωτεύον κλειδί; Ακόμα και με όλα τα γνωρίσματα δεν μπορεί να σχηματιστεί, καθώς μπορεί να συμβεί να έχω τον ίδιο τύπο πράξης σε δύο διαφορετικούς λογαριασμούς την ίδια ημ/νια, να αφορά το ίδιο ποσό και να έχει τον ίδιο αύξον αριθμό στους αντίστοιχους λογαριασμούς. Είναι φανερό ότι για να προσδιοριστεί με μοναδικό τρόπο μία πράξη πρέπει να περιλαμβάνει μαζί με τον αύξον αριθμό της, και τον κωδικό του λογαριασμού στον οποίο έγινε. Με άλλα λόγια η οντότητα Πράξη δεν είναι αυθύπαρκτη αλλά εξαρτάται από την Λογαριασμός. Θα λέμε λοιπόν ότι η Πράξη είναι αδύναμη οντότητα και εξαρτάται από την Λογαριασμός που είναι η βασική ή κύρια οντότητα. Συμβολίζουμε τις αδύναμες οντότητες με διπλό παραλληλόγραμμο και για να δείξουμε ποια είναι η κύρια οντότητα βάζουμε διπλό ρόμβο στην αντίστοιχη συσχέτιση μεταξύ αδύναμης και κύριας.

2.2.4 Εξειδικεύσεις-Γενικεύσεις

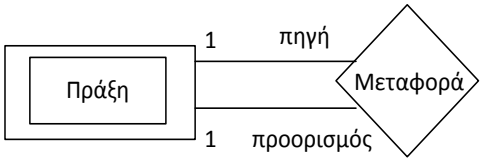
Θεωρείστε ότι σύμφωνα με τις απαιτήσεις θα πρέπει να διαχωρίσουμε τους πελάτες σε φυσικά πρόσωπα με γνωρίσματα (ΑΦΜ, ον/μο, ημΓεν, ηλικία, τηλνα, διεύθυνση) και σε νομικά πρόσωπα για τα οποία θέλουμε να σώζουμε ΑΦΜ, επωνυμία, τηλνα, διεύθυνση, και τον τύπο τους (πχ. ΑΕ, ΟΕ, δημοσίου δικαίου). Ένας τρόπος είναι να σπάσουμε το Πελάτης σε δύο σύνολα ένα για τα φυσικά και ένα για τα νομικά πρόσωπα. Η σχεδίαση αυτή έχει προβλήματα καθώς εκτός των άλλων θα διπλασιαστούν και οι συσχετίσεις Έχει και Πήρε αυξάνοντας την πολυπλοκότητα στη σύνταξη ερωτημάτων. Για το λόγο αυτό στο *επεκτεταμένο μοντέλο οντοτήτων συσχετίσεων* (*extended entity relationship model EER*) υπάρχει η έννοια της εξειδίκευσης μιας οντότητας σε κατηγορίες (αντίστοιχα της γενίκευσης κατηγοριών σε μία οντότητα). Οι εξειδικεύσεις μοιάζουν με την κληρονομικότητα των κλάσεων στην αντικειμενοστρέφεια και αναπαρίστανται με ένα τρίγωνο προς τη φορά της εξειδίκευσης στο οποίο αναγράφεται ISA δηλ. ΕΙΝΑΙ. Όταν μία οντότητα μπορεί να εξειδικευτεί σε μία μόνο κατηγορία, όπως στο παράδειγμα όπου δεν μπορεί κάποιος να είναι και φυσικό και νομικό πρόσωπο, τότε αναγράφουμε σε κύκλο ένα d (από το αγγλικό disjoint). Σε διαφορετική περίπτωση αφήνουμε τον κύκλο κενό. Ακολουθεί το κομμάτι του EER που αφορά στην εξειδίκευση στην Εικόνα 2.5.



Εικόνα 2.5: *Εξειδίκευση της οντότητας Πελάτης.*

2.2.5 Άλλα ζητήματα στις συσχετίσεις

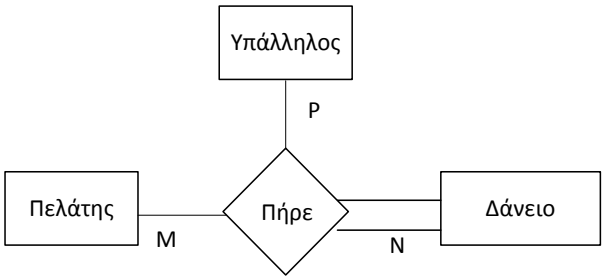
Αυτοσυσχετίσεις. Η μεταφορά ποσού από έναν λογαριασμό Α σε έναν Β ενέχει δύο πράξεις μία για τον Α (τύπος: μεταφορά ή ανάληψη) και μία για τον Β (τύπος: κατάθεση). Στον υπάρχον σχεδιασμό αυτές οι πράξεις μπορούν να σωθούν αλλά ο συσχετισμός μεταξύ τους όχι. Αυτό κάνει πολύ δύσκολη την απάντηση στο ερώτημα: «βρες σε ποιο λογαριασμό μεταφέρθηκαν τα χρήματα που έφυγαν με την πράξη Χ από το λογαριασμό Υ». Με άλλα λόγια για να αναπαρασταθεί πλήρως η έννοια της μεταφοράς χρημάτων θα πρέπει εκτός των δύο πράξεων στους ξεχωριστούς λογαριασμούς να υπάρχει και συσχέτισή τους. Η συσχέτιση αυτή καλείται αυτοσυσχέτιση εφόσον οι οντότητες που μετέχουν ανήκουν στο ίδιο σύνολο οντοτήτων. Ακολουθεί η σχηματική αναπαράσταση στην Εικόνα 2.6.



Εικόνα 2.6: *Παράδειγμα αυτοσυσχέτισης.*

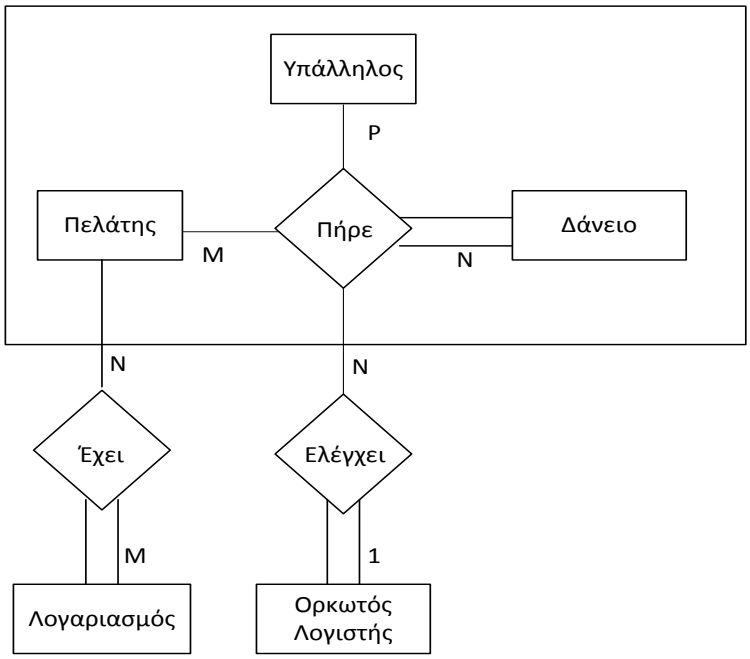
Στις αυτοσυσχετίσεις οι ακμές της συσχέτισης φέρουν περιγραφές που ονομάζονται ρόλοι. Αυτό είναι απαραίτητο για να μπορούμε να χαρακτηρίσουμε κατάλληλα τους περιορισμούς πληθικότητας και συμμετοχής (αναλόγως του ρόλου). Στην περίπτωσή μας δεν υπάρχει υποχρεωτική συμμετοχή και η πληθικότητα είναι 1 τόσο από τη μεριά της αρχικής πράξης (πηγή) όσο και της τελικής (προορισμός), αφού μοντελοποιούμε μόνο μεταφορά από έναν λογαριασμό σε έναν άλλο. Αν θέλαμε να μοντελοποιήσουμε την ταυτόχρονη μεταφορά από ένα λογαριασμό σε πολλούς άλλους τότε η πληθικότητα θα ήταν Ν από τη μεριά του ρόλου προορισμός.

Ν-αδικές συσχετίσεις. Μέχρι τώρα έχουμε μιλήσει μόνο για συσχετίσεις μεταξύ δύο συνόλων οντοτήτων. Στη γενικότερη περίπτωση μία συσχέτιση μπορεί να συσχετίζει Ν οντότητες στην οποία περίπτωση λέμε ότι η συσχέτιση είναι Ν-αδικού βαθμού. Για παράδειγμα έστω ότι χρειάζεται να μοντελοποιηθεί το εξής. *Θέλουμε να σώζουμε πληροφορίες για τους υπαλλήλους. Ένα δάνειο μπορεί να έχει πολλούς κατόχους και για να εγκριθεί θα πρέπει να ελεγχθεί η πιστοληπτική ικανότητα των πελατών που θα το πάρουν. Ο έλεγχος περιλαμβάνει έναν τουλάχιστον υπάλληλο ανά πελάτη*. Με άλλα λόγια πρέπει να μπορούμε να ξέρουμε ποιος/οι υπάλληλος/οι ήλεγξαν, ποιον πελάτη, σε ποιο δάνειο. Η μοντελοποίηση γίνεται με τριαδική συσχέτιση πολλά προς πολλά προς πολλά όπως φαίνεται στην Εικόνα 2.7.



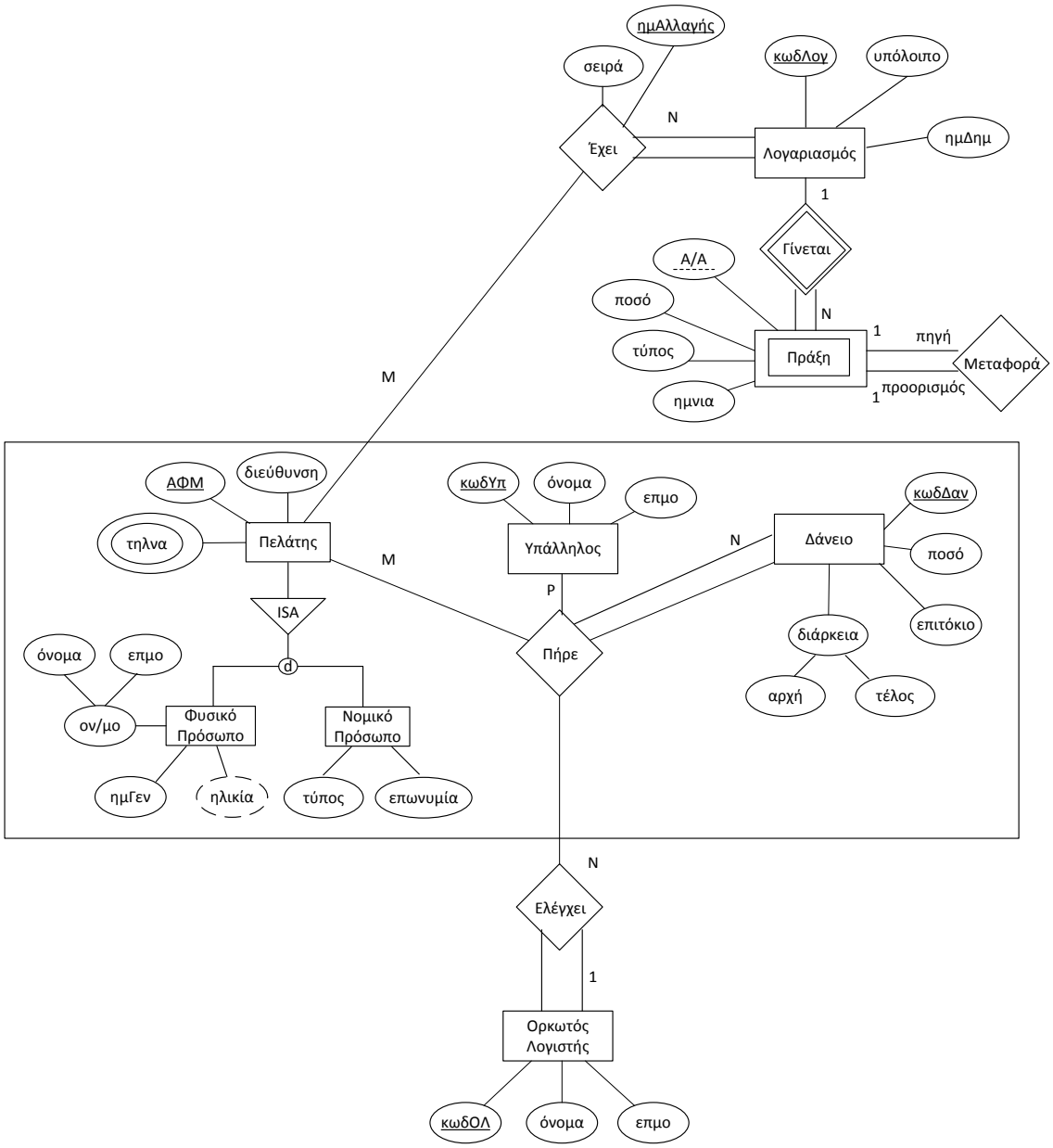
Εικόνα 2.7: *Παράδειγμα τριαδικής συσχέτισης.*

Γενικά η Ν-αδικές συσχετίσεις έχουν δυσκολίες στην αναπαράστασή τους αν δεν είναι της μορφής 1-1-1 ή M-N-P. Για παράδειγμα αν έχω μια τριαδική συσχέτιση μεταξύ των R, S και T η οποία είναι 1-M μεταξύ R και S, 1-N μεταξύ R και T και 1-P μεταξύ S και T αν γράψω ότι η συσχέτιση είναι 1-M-N ίσως υπονοείται M-N μεταξύ S-T ενώ δεν είναι. Αν αντίθετα θεωρήσω ότι είναι 1-1-N τότε υπονοείται 1-1 μεταξύ R-S που ούτε αυτό ισχύει. Γενικά καλό είναι να αποφεύγεται η άσκοπη χρήση τριαδικών και άνω συσχετίσεων. Παράδειγμα άσκοπης χρήσης έχω αν στο παραπάνω παράδειγμα υπέθετα ότι ένα δάνειο παίρνεται από έναν μόνο πελάτη. Σε αυτήν την περίπτωση ο ρόλος του ελέγχου μπορεί να αναπαρασταθεί με δυαδική συσχέτιση μεταξύ των: Υπάλληλος και Δάνειο και δε χρειαζόμαστε τριαδική συσχέτιση.



Εικόνα 2.8: *Παράδειγμα άθροισης.*

Αθροίσεις (Aggregations). Υπάρχουν περιπτώσεις που πρέπει να συσχετιστεί ένα σύνολο οντοτήτων με ένα σύνολο συσχετίσεων. Αυτό απαγορεύεται κανονικά. Για το λόγο αυτό περικλείουμε σε ορθογώνιο παραλληλόγραμμο το σύνολο συσχετίσεων που μας ενδιαφέρει μαζί με τα σύνολα οντοτήτων που συσχετίζει, φτιάχνοντας έτσι μία «υπεροντότητα» που κατόπιν μπορεί να μετέχει συνολικά (ως άθροισμα επιμέρους στοιχείων) σε συσχετίσεις Η συσχέτιση ενός συνόλου οντοτήτων με ένα άθροισμα αναπαρίσταται ενώνοντας την με τη συσχέτιση του αθροίσματος. Στο προηγούμενο παράδειγμα έστω ότι είχαμε ορκωτούς λογιστές να ασκούν έλεγχο πάνω στις ελεγκτικές δράσεις των υπαλλήλων όσον αφορά στα δάνεια. Ο τρόπος αναπαράστασης θα ήταν αυτός που φαίνεται στην Εικόνα 2.8.
Παρατηρήστε ότι η συσχέτιση Ελέγχει εμπλέκει τα Ορκωτός Λογιστής και Πήρε δηλ. το άθροισμα. Όλες οι υπόλοιπες συσχετίσεις που υπήρχαν και ενέπλεκαν οντότητες που ανήκουν στο άθροισμα, δεν αλλάζουν πχ. η συσχέτιση Έχει.



Εικόνα 2.9: *Τελικό ER διάγραμμα.*

2.2.6 Τελικό ER διάγραμμα

Υποθέτοντας ότι η μοναδική πληροφορία που μας ενδιαφέρει (χάριν απλότητας) για τους ορκωτούς λογιστές και τους υπαλλήλους είναι το όνομα και το επώνυμό τους, το τελικό ER διάγραμμα που ικανοποιεί όλες τις απαιτήσεις που παρουσιάστηκαν στα Κεφ. 2.2.1-5 είναι αυτό της Εικόνας 2.9.

Στη συνέχεια παρουσιάζουμε τη μεθοδολογία με την οποία μπορούμε να σχεδιάσουμε το σχήμα της ΒΔ από ER διάγραμμα, χρησιμοποιώντας το ανωτέρω ER ως παράδειγμα.

2.2.7 Μετατροπή σε πίνακες

Οντότητες (όχι αδύναμες). Κάθε οντότητα σχηματίζει ξεχωριστό πίνακα με πεδία τα απλά γνωρίσματα της οντότητας στο ER. Βάσει αυτού έχουμε στο παράδειγμα (υπογραμμισμένα τα πεδία του πρωτεύοντος κλειδιού):

Π1. Πελάτης (ΑΦΜ, διεύθυνση)
Π2. Υπάλληλος (κωδΥπ, όνομα, επμο)
Π3. ΟρκωτόςΛογιστής (κωδΟΛ, όνομα, επμο)
Π4. Δάνειο (κωδΔαν, ποσό, επιτόκιο)
Π5. Λογαριασμός (κωδΛογ, υπόλοιπο, ημΔημ)

Παραγόμενα γνωρίσματα. Μπορούμε να επιλέξουμε να μην περιληφθούν στο σχήμα (πχ. αν ο χρόνος υπολογισμού τους θεωρηθεί αμελητέος).

Σύνθετα γνωρίσματα. Δεν τα περιλαμβάνουμε στο σχήμα και αντ' αυτών περιλαμβάνουμε τα υπογνωρίσματα στα οποία αναλύονται. Βάσει αυτού ο Π4 γίνεται:
Π4.1. Δάνειο (κωδΔαν, ποσό, επιτόκιο, αρχή, τέλος)

Γνωρίσματα πολλαπλών τιμών. Δεν περιλαμβάνονται στο σχήμα της σχέσης που προκύπτει από την οντότητα που χαρακτηρίζουν αλλά σχηματίζουν δικό τους ξεχωριστό πίνακα. Σε αυτόν μετέχει το πρωτεύον κλειδί της οντότητας και το γνώρισμα αλλά όχι ως πολλαπλών τιμών αλλά ως μίας τιμής. Τόσο το πρωτεύον κλειδί όσο και το γνώρισμα μίας τιμής απαρτίζουν το πρωτεύον κλειδί του πίνακα που δημιουργήθηκε. Βάσει αυτού για το γνώρισμα τηλνα θα φτιαχτεί ο εξής ξεχωριστός πίνακας (με πλάγια γραμματοσειρά ξένο κλειδί):
Π6. Τηλνα (*ΑΦΜ*, τηλνο)

Δυαδικές συσχετίσεις Μ-Ν. Σχηματίζουν ξεχωριστό πίνακα με πεδία ότι γνωρίσματα υπάρχουν στη συσχέτιση, καθώς και τα πρωτεύοντα κλειδιά των οντοτήτων που μετέχουν. Στο νέο πίνακα πρωτεύον κλειδί είναι τα πρωτεύοντα κλειδιά των δύο οντοτήτων μαζί με τυχόν επιπλέον διαχωριστές συσχετίσεων που υπάρχουν. Βάσει αυτού θα σχηματιστεί ο εξής επιπλέον πίνακας:
Π7. Έχει (*ΑΦΜ*, *κωδΛογ*, σειρά, ημΑλλαγής)

Δυαδικές συσχετίσεις 1-Ν και 1-1. Αν είναι 1-Ν με υποχρεωτική συμμετοχή από τη μεριά του πολλά, τότε δε σχηματίζεται καινούργιος πίνακας αλλά μπαίνει το πρωτεύον κλειδί της οντότητας που έχει πληθικότητα 1 (μαζί με τα γνωρίσματα της συσχέτισης αν υπάρχουν), στον πίνακα που σχηματίζεται από την οντότητα με πληθικότητα Ν. Τέτοιου είδους συσχέτιση είναι η Γίνεται η οποία και δε σχηματίζει ξεχωριστό πίνακα. Επειδή όμως είναι συσχέτιση μεταξύ βασικής και αδύναμης οντότητας θα αντιμετωπιστεί στη συνέχεια. Αν η συσχέτιση είναι 1-1 με υποχρεωτική συμμετοχή τουλάχιστον από το ένα μέρος, πάλι δε σχηματίζεται ξεχωριστός πίνακας, αλλά ο πίνακας που προκύπτει από την οντότητα που έχει υποχρεωτική συμμετοχή επαυξάνεται με το πρωτεύον κλειδί της άλλης οντότητας και τα γνωρίσματα της συσχέτισης. Αν έχω συσχέτιση 1-1 με υποχρεωτική συμμετοχή και από τις δύο μεριές τότε μπορούμε να επαυξήσουμε όποιον πίνακα θέλουμε με το πρωτεύον κλειδί του άλλου πίνακα. Τέλος, αν έχουμε συσχέτιση 1-1 χωρίς υποχρεωτική συμμετοχή και από τις δύο μεριές ή 1-Ν χωρίς υποχρεωτική συμμετοχή από τη μεριά του Ν, τότε φτιάχνεται καινούργιος πίνακας όπως στην περίπτωση της συσχέτισης Μ-Ν. Ο λόγος είναι ότι αν ακολουθούσαμε την προηγούμενη τακτική δηλ. επαύξηση ενός πίνακα με το πρωτεύον κλειδί του άλλου, τότε εφόσον δεν υπάρχει υποχρεωτική συμμετοχή, κάποιες πλειάδες θα μπορούσαν να είχαν NULL τιμές στα πεδία του ξένου κλειδιού. Βάσει των παραπάνω σχηματίζουμε τους εξής πίνακες:
Αυτοσυσχετίσεις. Ακολουθούμε τους κανόνες για τις συσχετίσεις που παρουσιάστηκαν. Το πρωτεύον κλειδί της οντότητας είτε φτιαχτεί ξεχωριστός πίνακας είτε όχι, χρησιμοποιείται μαζί με το ρόλο του. Στο παράδειγμα θα φτιαχτεί ξεχωριστός πίνακας για τη συσχέτιση Μεταφορά ως εξής:

Π8. Μεταφορά (*κωδΛογΠηγή*, *ΑΑΠηγή*, *κωδΛογΠροορισμός*, *ΑΑΠροορισμός*)

Αδύναμες οντότητες. Η αδύναμη οντότητα σχηματίζει ξεχωριστό πίνακα με πεδία τα γνωρίσματά της, το πρωτεύον κλειδί της κύριας οντότητας και αν υπάρχουν τα αντίστοιχα γνωρίσματα της συσχέτισης με αυτήν. Στο πρωτεύον κλειδί του πίνακα συμμετέχουν το πρωτεύον κλειδί της κύριας και όποιος διαχωριστής υπάρχει στην αδύναμη. Έτσι έχω για την αδύναμη οντότητα Πράξη τον εξής πίνακα:
Π9. Πράξη (*κωδΛογ*, ΑΑ, ποσό, τύπος, ημνια)

Εξειδικεύσεις. Σχηματίζεται ξεχωριστός πίνακας για κάθε εξειδίκευση με πεδία τα γνωρίσματα της εξειδίκευσης και επιπλέον το πρωτεύον κλειδί της γενίκευσης. Έτσι έχουμε τους εξής πίνακες για της εξειδικεύσεις του Πελάτης:
Π10. ΦυσικόΠρόσωπο (*ΑΦΜ*, όνομα, επμο, ημΓεν)
Π11. ΝομικόΠρόσωπο (*ΑΦΜ*, τύπος, επωνυμία)

Ν-αδικές συσχετίσεις. Σχηματίζουν ξεχωριστό πίνακα με πρωτεύον κλειδί τα πρωτεύοντα κλειδιά των οντοτήτων που μετέχουν. Επιπλέον μπαίνουν ως πεδία όσα γνωρίσματα υπάρχουν στη συσχέτιση. Κατά συνέπεια για την Πήρε φτιάχνεται ο ακόλουθος πίνακα:
Π12. Πήρε (*ΑΦΜ*, *κωδΔαν*, *κωδΥπ*)

Αθροίσεις. Οι συσχετίσεις μεταξύ οντότητας και άθροισης ακολουθούν τους κανόνες των δυαδικών και Ν-αδικών συσχετίσεων μόνο που θεωρείται ως ο πίνακας που αντιστοιχεί στην άθροιση ο πίνακας της συσχέτισης που δημιουργεί την άθροιση. Στο παράδειγμα η συσχέτιση Ελέγχει είναι μεταξύ της οντότητας Ορκωτός Λογιστής και της άθροισης του σχήματος. Ο πίνακας που θεωρούμε ως «αντιπρόσωπο» της άθροισης είναι ο πίνακας Πήρε. Η συσχέτιση Ελέγχει είναι 1-Ν αλλά χωρίς υποχρεωτική συμμετοχή από τη μεριά του Ν, κατά συνέπεια θα φτιαχτεί νέος πίνακας που θα έχει τα πρωτεύοντα κλειδιά της ΟρκωτόςΛογιστής και της άθροισης δηλ. της Πήρε.
Π13. Ελέγχει (*κωδΟΛ*, *ΑΦΜ*, *κωδΔαν*, *κωδΥπ*)

Το τελικό σχήμα της ΒΔ που προκύπτει από το ER του παραδείγματος είναι οι πίνακες Π1-Π3, Π4.1 και Π5-Π13.

2.3 Λυμένες Εργαστηριακές Ασκήσεις

Άσκηση 1

Εκφώνηση:	Να σχεδιαστεί ΒΔ για ένα τμήμα προσωπικού μίας εταιρείας: • Για κάθε εργαζόμενο διατηρούνται το ΑΦΜ, το ΑΜΚΑ, η οικογενειακή κατάσταση, στοιχεία επικοινωνίας (email, τηλέφωνο), ο μισθός, τα παιδία του και τα τμήματα που εργάζεται. • Για τα παιδία του καταγράφονται το όνομα και η ηλικία. Τα παιδιά αναγνωρίζονται μοναδικά με βάση το όνομα όταν γνωρίζουμε τον γονέα του. • Για τα τμήματα της επιχείρησης διατηρούνται τα εξής στοιχεία: κωδικός τμήματος, όνομα και συνολικός προϋπολογισμός του τμήματος. • Κάθε τμήμα το διαχειρίζεται ένας προϊστάμενος.
Ζητούμενα:	1. Εξαγωγή των βασικών οντοτήτων του προβλήματος, των γνωρισμάτων της κάθε οντότητας καθώς και των συσχετίσεων οντοτήτων μεταξύ τους. 2. Σχεδιασμός του διαγράμματος Οντοτήτων-Συσχετίσεων (ER diagram)

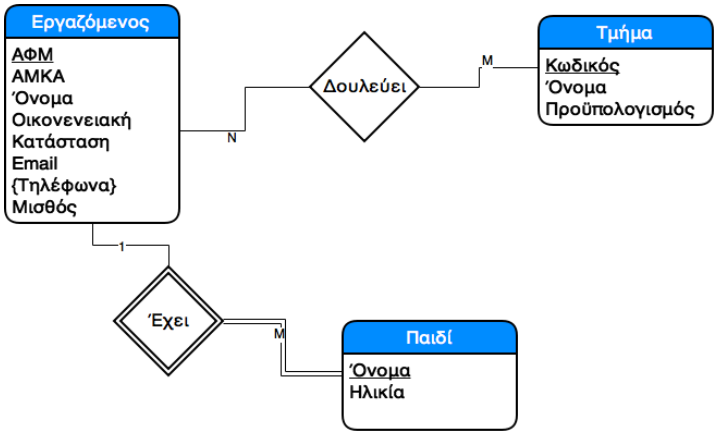
Αρχικά εντοπίζονται από το πρόβλημα οι οντότητες και τα βασικά χαρακτηριστικά τους:

- Εργαζόμενος
  - ΑΦΜ (πρωτεύον κλειδί)
  - ΑΜΚΑ (εναλλακτικό κλειδί)
  - Όνομα
  - Οικογενειακή Κατάσταση
  - Email
  - Τηλέφωνο
  - Μισθός
- Παιδί (αδύναμη οντότητα)
  - Όνομα (μετέχει στο πρωτεύον κλειδί)
  - Ηλικία
- Τμήμα
  - Κωδικός τμήματος (πρωτεύον κλειδί)
  - Όνομα Τμήματος
  - Προϋπολογισμός

Στη συνέχεια εντοπίζονται οι συσχετίσεις μεταξύ των οντοτήτων:

- Εργαζόμενος **δουλεύει** σε Τμήμα
- Εργαζόμενος **έχει** παιδί
- Εργαζόμενος **διαχειρίζεται** Τμήμα (για τον προϊστάμενο)

Το διάγραμμα οντοτήτων-συσχετίσεων που συνοψίζει τα παραπάνω φαίνεται στην Εικόνα 2.10.



Εικόνα 2.10: *ER της πρώτης άσκησης.*

Άσκηση 2

Εκφώνηση:	Να σχεδιαστεί ΒΔ για ένα τμήμα Πληροφορικής σε κάποιο ΑΕΙ. Για τους φοιτητές θέλουμε να διατηρούμε το ονοματεπώνυμο, ΑΜ, ένα ή περισσότερα τηλέφωνα επικοινωνίας, το έτος που βρίσκονται, τα μαθήματα που έχουν δώσει και το βαθμό που έγραψαν σε αυτά. Κάθε μάθημα έχει κωδικό και τίτλο και για κάποια από αυτά υπάρχουν προαπαιτούμενα μαθήματα. Χρειάζεται επίσης να διατηρούμε πληροφορία σχετικά με τους καθηγητές του τμήματος. Τα στοιχεία των καθηγητών είναι το ονοματεπώνυμό τους και το ποια μαθήματα διδάσκουν (αν διδάσκουν). Υποθέτουμε πως κάθε μάθημα διδάσκεται από ένα μόνο καθηγητή.
Ζητούμενα:	1. Εξαγωγή των βασικών οντοτήτων του προβλήματος, των χαρακτηριστικών της κάθε οντότητας καθώς και των συσχετίσεων οντοτήτων μεταξύ τους. 2. Σχεδιασμός του διαγράμματος Οντοτήτων-Συσχετίσεων (ER diagram) 3. Εξαγωγή του σχήματος της ΒΔ.

Ζητούμενο 1: Εξαγωγή των βασικών οντοτήτων του προβλήματος, των χαρακτηριστικών της κάθε οντότητας καθώς και των συσχετίσεων οντοτήτων μεταξύ τους.

Οι κύριες οντότητες που εντοπίζουμε στο παραπάνω πρόβλημα είναι οι οντότητες: **φοιτητής (student)**, **καθηγητής (professor)** και **μάθημα (course)**. Δεν είναι απαραίτητο να χρησιμοποιήσουμε και μία οντότητα τμήμα ή σχολή καθώς το πρόβλημα αναφέρεται σε ένα και μόνο τμήμα ΑΕΙ. Τα γνωρίσματα που χρειάζεται κατ' ελάχιστο να διατηρήσουμε στην βάση δεδομένων του προβλήματος για το κάθε στιγμιότυπο των παραπάνω οντοτήτων είναι:

- **Φοιτητής:** Όνομα**,** Επώνυμο**,** Αριθμό Μητρώου, ένα σύνολο τηλεφώνων, το έτος
- **Καθηγητής:** Όνομα**,** Επώνυμο, Κωδικός Καθηγητή (ΚΚ)
- **Μάθημα:** Τίτλος Μαθήματος, Κωδικός Μαθήματος (ΚΜ)

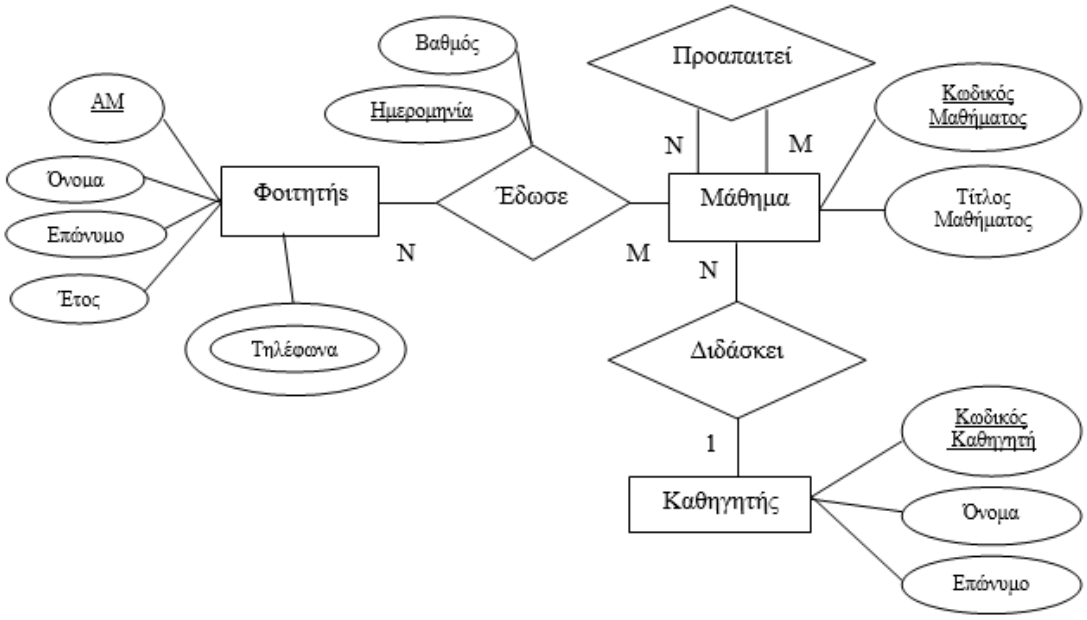
Είναι απαραίτητο για τις οντότητες καθηγητής και μάθημα να χρησιμοποιήσουμε κατάλληλους κωδικούς. Στην περίπτωση του καθηγητή ο Κωδικός Καθηγητή (ΚΚ) θα μας βοηθήσει σε περίπτωση συνωνυμίας (όνομα και επώνυμο) να αναπαραστήσουμε τους δύο διδάσκοντες με τα ίδια στοιχεία. Κατ' αντιστοιχία το ίδιο χρειάζεται για τα μαθήματα. Με χρήση κωδικού μαθήματος θα είναι δυνατό να αποθηκευτούν δύο μαθήματα με τον ίδιο τίτλο (κάτι τέτοιο είναι δυνατό να συμβεί π.χ. αν αλλάξει το πρόγραμμα σπουδών κλπ.).

Οι συσχετίσεις μεταξύ των οντοτήτων είναι η ακόλουθες:

- **Φοιτητής –ΕΔΩΣΕ (Exam)- Μάθημα:** Κάθε φοιτητής έχει δώσει εξετάσεις σε κάποια μαθήματα, συνεπώς μέσω αυτής της σχέσης θα πρέπει να καταγράφεται ο βαθμός που έλαβε στην εξέταση. Επειδή είναι δυνατό να δώσει ένας φοιτητής εξετάσεις παραπάνω από μία φορά μέχρι να επιτύχει σε κάποιο μάθημα, χρειάζεται να διατηρείται η ημερομηνία της κάθε αποτυχημένης εξέτασης. Συνεπώς τα δύο χαρακτηριστικά που περιγράφουν την σχέση ενός φοιτητή με ένα μάθημα είναι ο βαθμός εξέτασης καθώς και η ημερομηνία εξέτασης του μαθήματος. Κάθε μαθητής μπορεί να συσχετίζεται με πολλά μαθήματα συνεπώς ο τύπος συσχέτισης είναι πολλά προς πολλά.
- **Μάθημα – ΠΡΟΑΠΑΙΤΕΙ (Require) – Μάθημα:** Κάθε μάθημα μπορεί να έχει ως προαπαιτούμενα (αν έχει) ένα ή περισσότερα μαθήματα. Το ζευγάρι κωδικών μαθημάτων είναι αρκετό να αναπαραστήσει την συσχέτιση αυτή η οποία είναι τύπου M-N.
- **Καθηγητής – ΔΙΔΑΣΚΕΙ (Teach)– Μάθημα:** Σύμφωνα με το πρόβλημα κάθε μάθημα διδάσκεται από έναν μόνο καθηγητή συνεπώς η συσχέτιση είναι τύπου 1-N καθώς είναι δυνατό ένας καθηγητής να διδάσκει ένα ή περισσότερα μαθήματα.

Ζητούμενο 2: Σχεδιασμός του διαγράμματος Οντοτήτων-Συσχετίσεων (ER diagram)

Το πλήρες διάγραμμα οντοτήτων συσχετίσεων φαίνεται στην Εικόνα 2.11. Παρατηρήστε ότι το γνώρισμα Τηλέφωνα είναι γνώρισμα πολλαπλών τιμών και ότι η συσχέτιση Προαπαιτεί είναι αυτοσυσχέτιση. Οι ρόλοι (δεν αναγράφονται στο ER για απλότητα) είναι: (i) το μάθημα που προαπαιτεί (ΚΜ1) και (ii) το μάθημα που προαπαιτείται (ΚΜ2).



Εικόνα 2.11: *ER της δεύτερης άσκησης.*

Ζητούμενο 3: Εξαγωγή του σχήματος της ΒΔ.

Εφαρμόζοντας τους κανόνες μετατροπής ER σε πίνακες, το σχήμα της ΒΔ έχει ως εξής:

Φοιτητής (ΑΜ, Όνομα, Επώνυμο, Έτος)
Τηλέφωνα (ΑΜ, τηλνο)
Μάθημα (ΚΜ, τίτλοςΜαθήματος)
Καθηγητής (ΚΚ, Όνομα, Επώνυμο)
Έδωσε (ΑΜ, ΚΜ, βαθμός, Ημερομηνία)
Διδάσκει (ΚΜ, ΚΚ)
Προαπαιτεί (ΚΜ1, ΚΜ2)

2.4 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Θέλουμε να σχεδιάσουμε μια ΒΔ για το εσωτερικό πρόγραμμα εκπαίδευσης μιας μεγάλης εταιρείας. Η ΒΔ θέλουμε να κρατά τα στοιχεία κάθε εργαζομένου. Κάθε εργαζόμενος έχει ον/μο και ένα ID. Τα μαθήματα διδάσκονται και παρακολουθούνται από εργαζομένους της εταιρείας. Κάθε μάθημα έχει όνομα και ID και μπορεί να προσφέρεται περισσότερες από μία φορές. Για κάθε φορά που το μάθημα είναι διαθέσιμο θέλουμε να κρατάμε την ώρα και τη μέρα διδασκαλίας. Επίσης κάθε προσφορά μαθήματος έχει μοναδικό (μέσα στο ίδιο μάθημα) ID και μοναδικό διδάσκοντα (που μπορεί να διαφέρει από προσφορά σε προσφορά). Τέλος στη ΒΔ θέλουμε να κρατάμε τους βαθμούς των εργαζομένων στα μαθήματα που πήραν.

1. Σχεδιάστε το ER διάγραμμα της ΒΔ.
2. Δώστε το σχήμα της ΒΔ.

Άσκηση 2

Θέλουμε να σχεδιάσουμε μια ΒΔ για ένα φροντιστήριο. Για κάθε μαθητή θέλουμε να κρατάμε το ον/μο, ποια μαθήματα και σε τι τάξη παρακολουθεί. Επίσης θέλουμε να κρατάμε πληροφορίες για τα τμήματα που λειτουργούν (ποιοι μαθητές τα απαρτίζουν, σε τι μάθημα αναφέρονται, ποιος είναι ο καθηγητής), καθώς και για τους καθηγητές (ον/μο, συνολικές ώρες διδασκαλίας) Τέλος θέλουμε να κρατάμε πληροφορίες για το ωρολόγιο πρόγραμμα της εβδομάδος.

.

1. Σχεδιάστε το ER διάγραμμα της ΒΔ.
2. Δώστε το σχήμα της ΒΔ.

Άσκηση 3

Θέλουμε να φτιάξουμε μια βάση δεδομένων που πρέπει να κρατά πληροφορίες για τις ομάδες και τα παιχνίδια ενός πρωταθλήματος. Μια ομάδα έχει έναν αριθμό από παίκτες που δεν συμμετέχουν όλοι σε κάθε παιχνίδι. Θέλουμε να κρατάμε τα παιχνίδια στα οποία έπαιξαν οι παίκτες, το αποτέλεσμα του παιχνιδιού και σε ποια θέση έπαιξαν.

1. Σχεδιάστε το ER διάγραμμα της ΒΔ. Πιθανώς θα πρέπει να προσθέσετε γνωρίσματα που δεν αναφέρονται ρητώς στην περιγραφή. Αναφέρετε τυχόν υποθέσεις που κάνατε.
2. Δώστε το σχήμα της ΒΔ.

Άσκηση 4

Θέλουμε να φτιάξουμε μια βάση για την ΤΡΑΙΝΟΣΕ όπου θα κρατάμε πληροφορίες για τις αμαξοστοιχίες, τα δρομολόγια και τις κρατήσεις θέσεων. Κάθε αμαξοστοιχία έχει ξεχωριστό κωδικό και δρομολόγιο. Στο δρομολόγιο μιας αμαξοστοιχίας θέλουμε να κρατάμε την πόλη αναχώρησης, την πόλη άφιξης, ενδιάμεσους σταθμούς, καθώς και τις αντίστοιχες ώρες. Για παράδειγμα θα πρέπει να είμαστε σε θέση να κρατάμε την ακόλουθη πληροφορία: "Η αμαξοστοιχία Ι53 εκτελεί το ακόλουθο δρομολόγιο: αναχώρηση από Αθήνα στις 11:00, άφιξη Λιανοκλάδι στις 13:00, άφιξη Λάρισα στις 15:00, τελικός προορισμός Θες/νίκη στις 17:00". Η κράτηση θέσεων γίνεται ως εξής. Κάθε φορά που εκτελείται ένα δρομολόγιο αποφασίζεται ποια βαγόνια θα αποτελούν την αμαξοστοιχία και η σειρά με την οποία θα μπουν. Κάθε βαγόνι έχει το δικό του κωδικό και τη δική του αρίθμηση θέσεων. Για παράδειγμα: «Στο επόμενο

δρομολόγιο της αμαξοστοιχίας Ι53 μετέχουν τα βαγόνια Ζ85 σαν πρώτο και Χ87 σα δεύτερο. Η θέση 23 του δεύτερου βαγονιού είναι κενή».

1. Σχεδιάστε το ER διάγραμμα της ΒΔ.
2. Δώστε το σχήμα της ΒΔ.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαια 3,4,5,7.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαια 3,4,6.
MySQL Workbench Modeling.
https://dev.mysql.com/doc/workbench/en/wb-basic-modeling.html
https://dev.mysql.com/doc/workbench/en/wb-data-modeling.html
SHOW DATABASES, CREATE DATABASE και DROP DATABASE.
http://dev.mysql.com/doc/refman/5.7/en/database-use.html

Κεφάλαιο 3 Δημιουργία Πινάκων – Data Definition Language (DDL)

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν βασικά στοιχεία του Data Definition Language (DDL). Πιο συγκεκριμένα θα παρουσιασθεί ο τρόπος δημιουργίας πινάκων, διάφοροι βασικοί τύποι δεδομένων καθώς και οι περιορισμοί ακεραιότητας δεδομένων. Εν συνεχεία θα παρουσιασθεί ο σχεδιασμός και υλοποίηση μίας Βάσης Δεδομένων για την μηχανοργάνωση των φοιτητών/μαθημάτων/βαθμών ενός τμήματος ΑΕΙ.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- Σύντομη επισκόπηση MySQL και οδηγίες εγκατάστασης (Κεφ. 1.4 – 1.6).
- Βασική κατανόηση του σχεσιακού μοντέλου και του ER διαγράμματος (Κεφ. 2)

3.1 Εισαγωγικές Έννοιες

Σε ένα σχεσιακό σύστημα διαχείρισης βάσης δεδομένων (relational database management system RDBMS) ο ορισμός των δεδομένων γίνεται χρησιμοποιώντας τη γλώσσα ορισμού δεδομένων (data definition language DDL). Με άλλα λόγια η DDL μπορούμε να πούμε ότι είναι το κομμάτι της SQL που χρησιμοποιείται για τον ορισμό και επανακαθορισμό πινάκων. Στη συνέχεια παρουσιάζουμε τον τρόπο δημιουργίας πινάκων στην SQL. Ως βάση για την παρουσίαση τόσο στο παρόν κεφάλαιο όσο και στα επόμενα, θα χρησιμοποιήσουμε ένα μέρος του σχήματος της ΒΔ που παρουσιάστηκε στα Κεφ. 2.2.6 και 2.2.7. Πιο συγκεκριμένα, από το ER διάγραμμα της Εικόνας 2.9 κρατάμε μόνο τα σύνολα οντοτήτων Πελάτης, Λογαριασμός και Πράξη. Επιπλέον υποθέτουμε ότι ο Πελάτης μπορεί να είναι μόνο φυσικό πρόσωπο όπως στο ER διάγραμμα της Εικόνας 2.2 και στους λογαριασμούς δεν κρατιέται η σειρά αναγραφής των ιδιοκτητών. Τέλος, χάριν παραδείγματος, θα προσθέσουμε ένα ακόμα πεδίο (κωδΠ) στον πίνακα Πελάτης που θα παίζει το ρόλο πρωτεύοντος κλειδιού αντί του ΑΦΜ. Το σχήμα της ΒΔ που θα χρησιμοποιήσουμε ως παράδειγμα έχει ως εξής:

Πελάτης (κωδΠ, ΑΦΜ, διεύθυνση, όνομα, επμο, ημΓεν)
Τηλνα (*κωδΠ*, τηλνο)
Λογαριασμός (κωδΛογ, υπόλοιπο, ημΔημ)
Έχει (*κωδΠ*, *κωδΛογ*)
Πράξη (*κωδΛογ*, ΑΑ, ποσό, τύπος, ημνια)
Μεταφορά (*κωδΛογΠηγή*, *ΑΑΠηγή*, *κωδΛογΠροορισμός*, *ΑΑΠροορισμός*)

Για την ακρίβεια οι πίνακες στους οποίους θα βασίσουμε μέρος των παραδειγμάτων του εργαστηρίου είναι οι ακόλουθοι (στα Αγγλικά):

Customer (cid, afm, address, name, sname, dateOfBirth)
Phones (*cid*, pnum)
Account (accid, balance, dateOfCreation)
Owns (*cid*, *accid*)
Action (*accid*, actid, amount, type, dateOfAction)
Transfer (*accidSource*, *actidSource*, *accidDest*, *actidDest*)

3.1.1 Δημιουργία ΒΔ

Πριν προχωρήσουμε με τη δημιουργία των πινάκων που θα απαρτίζουν τη ΒΔ πρέπει πρώτα να φτιάξουμε τη ΒΔ μέσα στην οποία θα ορίσουμε τους πίνακες. Παρακάτω συνοψίζεται το συντακτικό και η λειτουργία κάποιων βασικών εντολών που αφορούν στη δημιουργία ΒΔ. Για επιπλέον πληροφορίες μπορεί κανείς να ανατρέξει στο εγχειρίδιο της MySQL 5.7 (Κεφ. 3.3 και Κεφ. 13.1)

CREATE DATABASE *όνομα_ΒΔ* : Δημιουργεί μία καινούργια ΒΔ με όνομα το *όνομα_ΒΔ*.
CREATE SCHEMA *όνομα_ΒΔ* : Ίδια με την CREATE DATABASE.
SHOW DATABASES : Εμφανίζει τις ΒΔ που υπάρχουν.
SHOW SCHEMAS : Ίδια με την SHOW DATABASES.
DROP DATABASE *όνομα_ΒΔ* : Διαγράφει τη ΒΔ με όνομα το *όνομα_ΒΔ*.
DROP SCHEMA *όνομα_ΒΔ* : Ίδια με την DROP DATABASE.
USE *όνομα_ΒΔ* : Επιλέγει τη ΒΔ με όνομα το *όνομα_ΒΔ* ως ενεργή. Όλες οι εντολές χειρισμού πινάκων, ερωτημάτων κλπ. που θα ακολουθήσουν θα αφορούν τη συγκεκριμένη ΒΔ.

Στο ANSI πρότυπο της SQL οι εντολές CREATE DATABASE και CREATE SCHEMA διέφεραν. Το σκεπτικό ήταν να επιτρέπεται η ύπαρξη πολλών σχημάτων μέσα σε μία ΒΔ με σκοπό να δίνονται διαφορετικά επίπεδα προσπέλασης στους χρήστες. Επειδή στην πράξη ο έλεγχος πρόσβασης χρηστών επιτυγχάνεται διαφορετικά, στη MySQL και σε πολλά άλλα RDBMS οι έννοιες του σχήματος και ΒΔ συμπίπτουν.

Ας πάμε να φτιάξουμε την πρώτη μας ΒΔ. Πρώτα θα δούμε ποιες άλλες ΒΔ υπάρχουν.
SHOW DATABASES;
Στη συνέχεια δημιουργούμε μια ΒΔ για την τραπεζική μας εφαρμογή.
CREATE DATABASE test;
Ας σβήσουμε τη ΒΔ που μόλις δημιουργήσαμε και ας την ξαναφτιάξουμε με πιο περιγραφικό όνομα. Επίσης ας την επιλέξουμε για τη συνέχεια.
DROP DATABASE test;
CREATE DATABASE bank_schema;
USE bank_schema;

Το ερωτηματικό ‘;’ στο τέλος κάθε εντολής δεν είναι μέρος της σύνταξης της εντολής, αλλά είναι ο προκαθορισμένος χαρακτήρας (delimiter) της MySQL, που δηλώνει το τέλος μιας εντολής και καλεί για την εκτέλεσή της. Τα επιτρεπτά ονόματα πινάκων και πεδίων (χωρίς να επεκταθούμε περισσότερο) ακολουθούν τους συνήθεις κανόνες που γνωρίζουμε από τις γλώσσες προγραμματισμού. Επιπλέον αν και η SQL δεν είναι case sensitive όσον αφορά τις εντολές καλή πρακτική είναι να γράφουμε εντολές, τύπους κλπ. με κεφαλαία για λόγους αναγνωσιμότητας.

Αν για οποιοδήποτε λόγο επιθυμούμε να δούμε τη ΒΔ που έχουμε επιλέξει, μπορούμε με την ακόλουθη εντολή:
SELECT DATABASE();
Στο παράδειγμα θα δούμε να επιστρέφεται ένας πίνακας με πεδίο το DATABASE() και μία εγγραφή το bank_schema.

3.1.2 Η εντολή CREATE TABLE

Η εντολή με την οποία ορίζουμε πίνακα σε ΒΔ είναι η εντολή CREATE TABLE, της οποίας η βασική σύνταξη έχει ως εξής:

CREATE TABLE *όνομα_πίνακα* (
    όνομα_πεδίου τύπος_πεδίου [*επιπλέον_επιλογές*],
    ….,

[*περιορισμοί*],
….
)

όπου οι *επιπλέον_επιλογές* και οι *περιορισμοί* είναι προαιρετικά, ενώ μπορεί να έχουμε πολλαπλές δηλώσεις πεδίων και περιορισμών. Σχετικά με το πλήρες συντακτικό της εντολής CREATE TABLE μπορεί κανείς να ανατρέξει στο εγχειρίδιο της MySQL 5.7 (Κεφ. 13.1.14).

Για παράδειγμα για να δημιουργήσω τον πίνακα Customer μια πρώτη προσπάθεια θα ήταν να γράψουμε το εξής:

```
CREATE TABLE Customer (
        cid INT,
        afm CHAR(10),
        address VARCHAR(50),
        name CHAR(20),
        sname CHAR(20),
        dateOfBirth DATE
);
```

Μια καλή πρακτική είναι τουλάχιστον το πρώτο γράμμα στο όνομα ενός πίνακα να είναι κεφαλαίο και στο όνομα πεδίου να είναι πεζό. Επίσης, αν και στην SQL δεν παίζουν ρόλο οι γραμμές καλό είναι να γράφουμε για λόγους αναγνωσιμότητας με τον παραπάνω τρόπο (κάθε πεδίο σε νέα γραμμή). Για να γίνει κατανοητό συγκρίνεται την παραπάνω δήλωση με την ακόλουθη ισοδύναμη:

```
CREATE TABLE Customer (cid INT, address VARCHAR(50), name CHAR(20), sname CHAR(20), dateOfBirth DATE);
```

Η SQL υποστηρίζει πληθώρα τύπων για αριθμούς, αλφαριθμητικά, ημ/νίες, χρόνο κλπ. Μια πλήρης καταγραφή μπορεί να βρεθεί στο εγχειρίδιο της MySQL 5.7 (Κεφ. 11). Οι βασικοί τύποι με τους οποίους θα ασχοληθούμε στα παραδείγματά μας είναι οι εξής:
CHAR(X): Αλφαριθμητικό σταθερού μεγέθους X.
VARCHAR(X): Αλφαριθμητικό μεταβλητού μεγέθους X.
TINYINT: Ακέραιος μεγέθους 1 byte.
SMALLINT: Ακέραιος μεγέθους 2 bytes.
MEDIUMINT: Ακέραιος μεγέθους 3 bytes.
INT: Ακέραιος μεγέθους 4 bytes.
BIGINT: Ακέραιος μεγέθους 8 bytes.
FLOAT: Δεκαδικός μεγέθους 4 bytes.
DOUBLE: Δεκαδικός μεγέθους 8 bytes.
DATE: Ημ/νια στη μορφή ‘YYYY-MM-DD’, πχ., ‘2014-01-31’.
TIME: Ώρα στη μορφή ‘HH:MM:SS’, π.χ., ‘12:30:00’ αντιστοιχεί στις 12 και μισή το μέσημέρι.
TIMESTAMP: Ημ/νια και ώρα μαζί πχ., ‘2014-01-31 12:30:00’.

Η επιλογή ανάμεσα στο πλήθος των διαφορετικών ακεραίων ή ανάμεσα σε απλής ή διπλής ακρίβειας αριθμό κινητής υποδιαστολής πρέπει να γίνεται αναλόγως του εύρους των τιμών που αναμένονται για το πεδίο. Για παράδειγμα επιλέξαμε να αναπαραστήσουμε το afm ως αλφαριθμητικό αντί για ακέραιο, για να είναι ευκολότερος ο χειρισμός των ΑΦΜ που ξεκινούν με ένα ή περισσότερα μηδενικά.

Αξίζει λίγο να σταθούμε στη διαφορά μεταξύ CHAR και VARCHAR. Οι δύο αυτοί τύποι ορίζουν αλφαριθμητικά, ο πρώτος σταθερού μεγέθους, ο δεύτερος μεταβλητού. Αν κατά την εισαγωγή τιμής ξεπεραστεί το μέγεθος που ορίζεται μέσα στις παρενθέσεις, οι υπόλοιποι χαρακτήρες που εισάγονται αγνοούνται και θα εμφανιστεί αντίστοιχο μήνυμα (περισσότερα στο επόμενο κεφάλαιο). Αν το μέγεθος του αλφαριθμητικού που εισάγεται είναι μικρότερο από τα

όρια που καθορίζονται, στον τύπο CHAR η δέσμευση εξακολουθεί να παραμένει όσο είναι το όριο που αναγράφεται στην παρένθεση, ενώ στον VARCHAR δεσμεύεται τόσος χώρος όσος αρκεί για να σωθεί το αλφαριθμητικό. Για παράδειγμα αν στο name εισάγουμε: 'Thomas' (6 χαρακτήρες) επειδή ο τύπος του πεδίου είναι CHAR(20) η δέσμευση θα γίνει για 20 χαρακτήρες, ενώ αν ήταν VARCHAR(20) θα γίνει για τους 6 χαρακτήρες που εισάγαμε. Αυτό δε σημαίνει ότι πρέπει πάντα να χρησιμοποιείται ο τύπος VARCHAR έναντι του CHAR. Χωρίς να μπούμε σε λεπτομέρειες, αρκεί να πούμε ότι το να διατηρείται σταθερό το μήκος εγγραφών «βοηθάει» το RDBMS ενώ το μεταβλητό μήκος εγγραφών οδηγεί σε πιο πολύπλοκη εσωτερική δομή. Ως απλός κανόνας μπορεί να χρησιμοποιηθεί το εξής: αν περιμένουμε μεγάλες αποκλίσεις στο μέγεθος των εισαγόμενων αλφαριθμητικών καλό είναι να χρησιμοποιούμε VARCHAR αλλιώς όχι.

Επανερχόμενοι στο παράδειγμά μας, παρατηρούμε ότι στον πίνακα δεν έχουμε ορίσει πρωτεύον κλειδί. Σε αντίθεση με το σχεσιακό μοντέλο όπως περιγράφηκε στο Κεφ. 2, στην SQL μπορώ να έχω διπλές εγγραφές τόσο σε πίνακα της ΒΔ (εφ' όσον δεν έχει οριστεί πρωτεύον κλειδί), όσο και στα αποτελέσματα ερωτημάτων (περισσότερα για αυτό σε επόμενο κεφάλαιο). Ορισμένες φορές το πρωτεύον κλειδί είναι ένα πεδίο στο οποίο δε μας «ενδιαφέρουν» οι συγκεκριμένες τιμές που παίρνει, αρκεί να διαφέρουν από εγγραφή σε εγγραφή. Σε αυτήν την περίπτωση μπορούμε να επιλέξουμε οι τιμές που θα παίρνει να δίνονται αυτόματα. Επιπλέον παρατηρούμε ότι κάποια πεδία πρέπει να παίρνουν υποχρεωτικά τιμές πχ., τα afm, name, surname, και κάποια όχι, πχ., dateOfBirth.

Για να μπορέσουμε να επιτύχουμε τα παραπάνω, στον καινούργιο σχεδιασμό μας θα πρέπει να χρησιμοποιήσουμε τις *επιπλέον_επιλογές* που υπάρχουν στη σύνταξη της CREATE TABLE, επιγραμματικά οι κυριότερες των οποίων είναι οι εξής:
NOT NULL: Δηλώνει ότι το πεδίο πρέπει να έχει υποχρεωτικά τιμή.
DEFAULT: Δηλώνει την προκαθορισμένη τιμή ενός πεδίου αν δεν έχει περαστεί τιμή.
UNIQUE [KEY]: Δηλώνει εναλλακτικό κλειδί (το KEY προαιρετικό).
[PRIMARY] KEY: Δηλώνει πρωτεύον κλειδί (το PRIMARY είναι προαιρετικό αλλά καλό είναι να γράφεται για λόγους αναγνωσιμότητας).
AUTO_INCREMENT: Δηλώνει ότι το πεδίο θα παίρνει τιμές αυτόματα. Στην MySQL 5.7 μπορεί να υπάρχει μόνο ένα πεδίο σε κάθε πίνακα με αυτή την επιλογή. Η αυτόματη εκχώρηση τιμής γίνεται προσθέτοντας +1 στην προηγούμενη που εκχωρήθηκε.

Λαμβάνοντας υπόψη τα παραπάνω επανερχόμαστε με το νέο ορισμό πίνακα ως εξής:

```
CREATE TABLE Customer (
        cid INT AUTO_INCREMENT PRIMARY KEY,
        afm CHAR(10) UNIQUE KEY,
        address VARCHAR(50) DEFAULT 'Unknown',
        name CHAR(20) NOT NULL DEFAULT 'Unknown',
        sname CHAR(20) NOT NULL DEFAULT 'Unknown',
        dateOfBirth DATE DEFAULT NULL
);
```

Tip 1: Επιχειρώντας να κάνετε αντιγραφή και επικόλληση της παραπάνω εντολής στον MySQL client πιθανότατα θα λάβετε σφάλμα που έχει να κάνει με τη χρήση των '' στο 'Unknown', δηλ. με το σετ χαρακτήρων. Γράφοντας την εντολή στον client το αποτέλεσμα θα μοιάζει κάπως έτσι: 'Unknown' και θα εκτελείται χωρίς πρόβλημα.
Tip 2: Σε περίπτωση που θέλετε να σβήσετε τον πίνακα Customer για να τον ξαναδηλώσετε χρησιμοποιήστε την εντολή: DROP TABLE Customer; (περισσότερα για την εντολή στο επόμενο κεφάλαιο).

Εναλλακτικά τη δήλωση του PRIMARY KEY και του UNIQUE KEY μπορούμε να την έχουμε μετά τις δηλώσεις πεδίων ως περιορισμό. Για την ακρίβεια ο προηγούμενος τρόπος χρησιμοποιείται μόνο αν ως PRIMARY ή UNIQUE KEY είναι ένα πεδίο. Η δήλωση του πίνακα Customer έχει ως ακολούθως:

```
CREATE TABLE Customer (
        cid INT AUTO_INCREMENT,
        afm CHAR(10),
        address VARCHAR(50) DEFAULT 'Unknown',
        name CHAR(20) NOT NULL DEFAULT 'Unknown',
        sname CHAR(20) NOT NULL DEFAULT 'Unknown',
        dateOfBirth DATE DEFAULT NULL,
        PRIMARY KEY (cid),
        UNIQUE KEY (afm)
);
```

Πεδία στα οποία δεν αναγράφεται καμία επιλογή θεωρείται ότι μπορούν να πάρουν NULL τιμές (που δηλώνουν απουσία τιμής). Πεδία που είναι μέρος PRIMARY ή UNIQUE KEY δεν επιτρέπεται να έχουν NULL τιμές. Η δήλωση PRIMARY και UNIQUE KEY υπονοεί NOT NULL το οποίο δε χρειάζεται να γραφεί. Αντίστοιχα ισχύουν για το AUTO_INCREMENT (και αυτό υπονοεί NOT NULL) αλλά και ότι είναι τουλάχιστον UNIQUE KEY και πρέπει να φτιαχτεί ευρετήριο υποχρεωτικά (περισσότερα στο Κεφ. 12). Συνήθως το AUTO_INCREMENT χρησιμοποιείται σε PRIMARY KEY κατά το παράδειγμα.

3.1.3 Ξένα κλειδιά

Ας προχωρήσουμε με τη δήλωση και των υπολοίπων πινάκων ξεκινώντας με τον πίνακα Phones(*cid*, pnum). Στον πίνακα αυτόν αναπαρίστανται τα τηλέφωνα των πελατών. Για παράδειγμα αν ο πελάτης με cid=1 έχει δύο τηλέφωνα 6911222333 και 2101122233 στον πίνακα Phones θα υπάρχουν οι εξής δύο εγγραφές: <1, 6911222333> και <1, 2101122233>. Για το λόγο αυτό το πρωτεύον κλειδί του πίνακα αποτελείται από δύο πεδία και το cid και το pnum. Η δήλωση του PRIMARY KEY (σαν περιορισμό) θα έχει ως εξής:

```
PRIMARY KEY (cid, pnum)
```

Επιπλέον στον πίνακα Phones πρέπει να οριστεί το cid ως ξένο κλειδί. Σχετικά με το ρόλο των ξένων κλειδιών και τους περιορισμούς αναφορικής ακεραιότητας αναφερθήκαμε εκτενώς στο Κεφ. 2. Στο παράδειγμά μας θα πρέπει να απαγορεύσουμε στον πίνακα Phones να εισαχθεί εγγραφή πχ. <1, 6911222333> αν στον πίνακα Customer δεν υπάρχει εγγραφή με cid 1.

Ξένα κλειδιά ορίζονται είτε ως *επιπλέον_επιλογές* στον ορισμό του πεδίου, είτε σαν *περιορισμοί*. Παρατήρηση: Για λόγους αναγνωσιμότητας καλό είναι να προτιμάται η εισαγωγή ξένων κλειδιών μετά τις δηλώσεις των πεδίων. Η σύνταξη της δήλωσης ενός ξένου κλειδιού ως περιορισμό έχει ως εξής:

```
FOREIGN KEY (πεδία) REFERENCES όνομα_πίνακα (πεδία αναφοράς)
```

Στο παράδειγμά μας γράφω:

```
FOREIGN KEY (cid) REFERENCES Customer (cid)
```

Αξίζει να σημειωθεί ότι το όνομα του ξένου κλειδιού και του κλειδιού στο οποίο αναφέρεται δε χρειάζεται να συμπίπτουν. Υποχρεωτικά όμως πρέπει να είναι ιδίου τύπου. Επιτρέπεται η δήλωση ξένου κλειδιού που απαρτίζεται από δύο ή περισσότερα πεδία, αρκεί τα αντίστοιχα πεδία αναφοράς να είναι πρωτεύοντα κλειδιά στον πίνακα που αναφερόμαστε. Για παράδειγμα η παρακάτω δήλωση:

```
FOREIGN KEY (x, y) REFERENCES T (a, b)
```

είναι καθόλα επιτρεπτή αρκεί τα x και y να έχουν ίδιους τύπους με τα a και b αντίστοιχα, και τα a και b από κοινού να απαρτίζουν το πρωτεύον κλειδί του πίνακα T.

Μετά τον ορισμό ενός ξένου κλειδιού δημιουργούνται τα εξής ερωτήματα. Πρώτον τί θα πρέπει να γίνει σε περίπτωση που κάποια εγγραφή στον πίνακα αναφοράς αλλάξει τιμή στο πρωτεύον κλειδί και δεύτερον τί θα πρέπει να συμβεί αν μια εγγραφή στον πίνακα αναφοράς διαγραφεί. Για παράδειγμα, έστω ότι ο πίνακας Phones έχει την εγγραφή <1, 6911222333>. Το ερώτημα είναι τι θα πρέπει να συμβεί αν στον πίνακα Customer αλλάξει η εγγραφή με cid 1 από 1 πχ., σε 30. Η εγγραφή <1, 6911222333> στο Phones παραβιάζει τώρα τους περιορισμούς αναφορικής ακεραιότητας καθώς πλέον δεν υπάρχει εγγραφή στο Customer με cid 1. Για να καθορίσουμε τι επιθυμούμε να συμβεί σε περίπτωση αλλαγής ή διαγραφής μιας εγγραφής χρησιμοποιούμε στη δήλωση ξένου κλειδιού τις δηλώσεις ON UPDATE και ON DELETE. Η σύνταξη της δήλωσης ξένου κλειδιού είναι τώρα η εξής:

FOREIGN KEY (*πεδία*) REFERENCES *όνομα_πίνακα* (*πεδία αναφοράς*)
ON DELETE *ενέργεια*
ON UPDATE *ενέργεια*

Οι πιθανές ενέργειες που υποστηρίζονται είναι οι εξής:

RESTRICT: Απαγορεύει τη διαγραφή ή ενημέρωση της εγγραφής.
CASCADE: Ενημερώνει ή διαγράφει τις αντίστοιχες εγγραφές.
SET NULL: Διατηρεί τις εγγραφές με NULL στα αντίστοιχα πεδία.
NO ACTION: Απαγορεύει τη διαγραφή ή ενημέρωση της εγγραφής.

Οι ενέργειες RESTRICT και NO ACTION στην MySQL 5.7 είναι ακριβώς ισοδύναμες δηλαδή και οι δύο απαγορεύουν άμεσα την ενημέρωση ή τη διαγραφή. Ο λόγος που υπάρχουν και οι δύο είναι για λόγους συμβατότητας με άλλα RDBMS όπου το NO ACTION ενέχει μελλοντικό και όχι άμεσο έλεγχο. Μία άλλη ενέργεια που υπάρχει στο πρότυπο της SQL αλλά δεν υποστηρίζεται επαρκώς από τη MySQL 5.7 είναι η SET DEFAULT που επαναφέρει το ξένο κλειδί στην προκαθορισμένη τιμή. Αν δεν προσδιοριστεί καμία ενέργεια θεωρείται το RESTRICT επιλεγμένο. Η *ενέργεια* SET NULL δεν μπορεί να χρησιμοποιηθεί σε πεδία που είναι ορισμένα σαν NOT NULL ή αποτελούν μέρος PRIMARY ή UNIQUE KEY. Στο παράδειγμα μας, αν έχω στο Phones την εγγραφή <1, 6911222333> και στο Customer αλλάξω την εγγραφή με cid 1 από cid 1 σε 30 τότε θα έχω αναλόγως της ενέργειας:

RESTRICT: Η αλλαγή από 1 σε 30 στο Customer δε θα πραγματοποιηθεί. Για να γίνει θα πρέπει πρώτα να διαγραφεί η εγγραφή <1, 6911222333> στο Phones.
CASCADE: Η αλλαγή από 1 σε 30 στο Customer θα πραγματοποιηθεί. Η εγγραφή <1, 6911222333> του Phones θα αλλάξει σε <30, 6911222333>.
SET NULL: Δεν είναι δυνατόν να οριστεί αυτή η ενέργεια καθώς το cid είναι μέρος του PRIMARY KEY στον πίνακα Phones και κατ' επέκταση δεν μπορεί να πάρει NULL τιμές. Αν επιχειρούσαμε να εισάγουμε FOREIGN KEY με αυτή την ενέργεια θα παίρναμε μήνυμα λάθους. Συνοψίζοντας στο παράδειγμά μας, έχουμε:

Σχήμα
Customer (<u>cid</u>, afm, address, name, sname, dateOfBirth)
Phones (<u>*cid*</u>, <u>pnum</u>)
Account (<u>accid</u>, balance, dateOfCreation)
Owns (<u>*cid*</u>, <u>*accid*</u>)
Action (<u>*accid*</u>, <u>actid</u>, amount, type, dateOfAction)
Transfer (<u>*accidSource*</u>, <u>*actidSource*</u>, <u>*accidDest*</u>, <u>*actidDest*</u>)

Δηλώσεις
```
CREATE TABLE Customer (
        cid INT AUTO_INCREMENT,
```

```
        afm CHAR(10),
        address VARCHAR(50) DEFAULT ‘Unknown’,
        name CHAR(20) NOT NULL DEFAULT ‘Unknown’,
        sname CHAR(20) NOT NULL DEFAULT ‘Unknown’,
        dateOfBirth DATE DEFAULT NULL,
        PRIMARY KEY (cid),
        UNIQUE KEY (afm)
);

CREATE TABLE Phones (
        cid INT,
        pnum CHAR(10),
        PRIMARY KEY (cid, pnum),
        FOREIGN KEY (cid) REFERENCES Customer (cid)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);

CREATE TABLE Account (
        accid    BIGINT COMMENT 'This is the way to add comment: BIGINT for scalability',
        balance FLOAT NOT NULL DEFAULT 0,
        dateOfCreation  DATE NOT NULL,
        PRIMARY KEY (accid)
);

CREATE TABLE Owns (
        cid INT,
        accid BIGINT,
        PRIMARY KEY (cid, accid),
        FOREIGN KEY (cid) REFERENCES Customer (cid)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
        FOREIGN KEY (accid) REFERENCES Account (accid)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);

CREATE TABLE Action (
        accid BIGINT,
        actid INT,
        amount FLOAT NOT NULL DEFAULT 0,
        type     TINYINT NOT NULL DEFAULT 0,
        dateOfAction DATE NOT NULL,
        PRIMARY KEY (accid, actid),
        FOREIGN KEY (accid) REFERENCES Account (accid)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);

CREATE TABLE Transfer (
        accidSource BIGINT,
        actidSource INT,
        accidDest BIGINT,
        actidDest INT,
        PRIMARY KEY (accidSource, actidSource, accidDest, actidDest),
```

```
    FOREIGN KEY (accidSource, actidSource) REFERENCES Action (accid, actid)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (accidDest, actidDest) REFERENCES Action (accid, actid)
    ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Σημείωση 1: Στον πίνακα Action θέσαμε τον τύπο μιας πράξης σε λογαριασμό σε TINYINT καθώς αναμένουμε το πλήθος των δυνατών πράξεων να είναι μικρό. Κατ' αντιστοιχία το accid τέθηκε σε BIGINT καθώς το πλήθος των λογαριασμών που υπάρχουν μπορεί να γίνει εξαιρετικά μεγάλο.

Σημείωση 2: Στον πίνακα Transfer έχω δύο ξένα κλειδιά το λογαριασμό πηγή και το λογαριασμό προορισμού. Κάθε λογαριασμός προσδιορίζεται από ζευγάρι πεδίων για αυτό έχουμε και την παραπάνω δήλωση του πίνακα Transfer.

3.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Να σχεδιαστεί Βάση Δεδομένων για ένα τμήμα Πληροφορικής σε κάποιο ΑΕΙ. Για τους φοιτητές θέλουμε να διατηρούμε το ονοματεπώνυμο, Α.Μ, ένα ή περισσότερα τηλέφωνα επικοινωνίας, το έτος που βρίσκονται, τα μαθήματα που έχουν δώσει και το βαθμό που έγραψαν σε αυτά. Κάθε μάθημα έχει κωδικό και τίτλο και για κάποια από αυτά υπάρχουν προαπαιτούμενα μαθήματα. Χρειάζεται επίσης να διατηρούμε πληροφορία σχετικά με τους καθηγητές του τμήματος. Τα στοιχεία των καθηγητών είναι το ονοματεπώνυμο τους και το ποια μαθήματα διδάσκουν (αν διδάσκουν). Υποθέτουμε πως κάθε μάθημα διδάσκεται από ένα μόνο καθηγητή.
Ζητούμενα:	3. Εξαγωγή των βασικών οντοτήτων του προβλήματος, των χαρακτηριστικών της κάθε οντότητας καθώς και των συσχετίσεων οντοτήτων μεταξύ τους. 4. Σχεδιασμός του διαγράμματος Οντοτήτων-Συσχετίσεων (ER diagram) 5. Σχεδιασμός πινάκων σχεσιακού μοντέλου. 6. Εφαρμογή Περιορισμών ακεραιότητας δεδομένων (πρωτεύοντα και δευτερεύοντα κλειδιά –(primary/secondary keys)). 7. Υλοποίηση του σχεδιαστικού μέρους (ER diagram) με χρήση του εργαλείου MySQL Model του MySQL Workbench. 8. Υλοποίηση της ΒΔ και των αντίστοιχων πινάκων με χρήση SQL-DDL εντολών με χρήση του εργαλείου MySQL Workbench.

Ζητούμενο 1: Εξαγωγή των βασικών οντοτήτων του προβλήματος, των χαρακτηριστικών της κάθε οντότητας καθώς και των συσχετίσεων οντοτήτων μεταξύ τους.

Οι κύριες οντότητες που εντοπίζουμε στο παραπάνω πρόβλημα είναι η οντότητα **φοιτητής (student)**, **καθηγητής (professor)** και **μάθημα (Course)**. Δεν είναι απαραίτητο να χρησιμοποιήσουμε και μία οντότητα τμήμα ή σχολή καθώς το πρόβλημα αναφέρεται σε ένα και μόνο τμήμα ΑΕΙ. Τα γνωρίσματα που χρειάζεται κατ' ελάχιστο να διατηρήσουμε στην βάση δεδομένων του προβλήματος για το κάθε στιγμιότυπο των παραπάνω οντοτήτων είναι:

- **Φοιτητής:** Όνομα**,** Επώνυμο**,** Αριθμό Μητρώου, ένα σύνολο τηλεφώνων, το έτος
- **Καθηγητής:** Όνομα**,** Επώνυμο, Κωδικός Καθηγητή (ΚΚ)
- **Μάθημα:** Τίτλος Μαθήματος, Κωδικός Μαθήματος (ΚΜ)

Είναι απαραίτητο για τις οντότητες καθηγητής και μάθημα να χρησιμοποιήσουμε κατάλληλους κωδικούς. Στην περίπτωση του καθηγητή ο Κωδικός Καθηγητή (ΚΚ) θα μας βοηθήσει σε περίπτωση συνωνυμίας (όνομα και επώνυμο) να αναπαραστήσουμε τους δύο διδάσκοντες με τα ίδια στοιχεία. Κατά αντιστοιχία το ίδιο χρειάζεται για τα μαθήματα. Με χρήση κωδικού μαθήματος θα είναι δυνατό να αποθηκευτούν δύο μαθήματα με τον ίδιο τίτλο (κάτι τέτοιο είναι δυνατό να συμβεί π.χ. αν αλλάξει το πρόγραμμα σπουδών κλπ.).

Οι συσχετίσεις μεταξύ των οντοτήτων είναι η ακόλουθες:

- **Φοιτητής –ΕΔΩΣΕ (Exam)- Μάθημα:** Κάθε φοιτητής έχει δώσει εξετάσεις σε κάποια μαθήματα, συνεπώς μέσω αυτής της σχέσης θα πρέπει να καταγράφεται ο βαθμός που έλαβε στην εξέταση. Επειδή είναι δυνατό να δώσει ένας φοιτητής εξετάσεις παραπάνω από μία φορά μέχρι να λάβει προβιβάσημο βαθμό χρειάζεται να διατηρείται η ημερομηνία της κάθε αποτυχημένης εξέτασης. Συνεπώς τα δύο

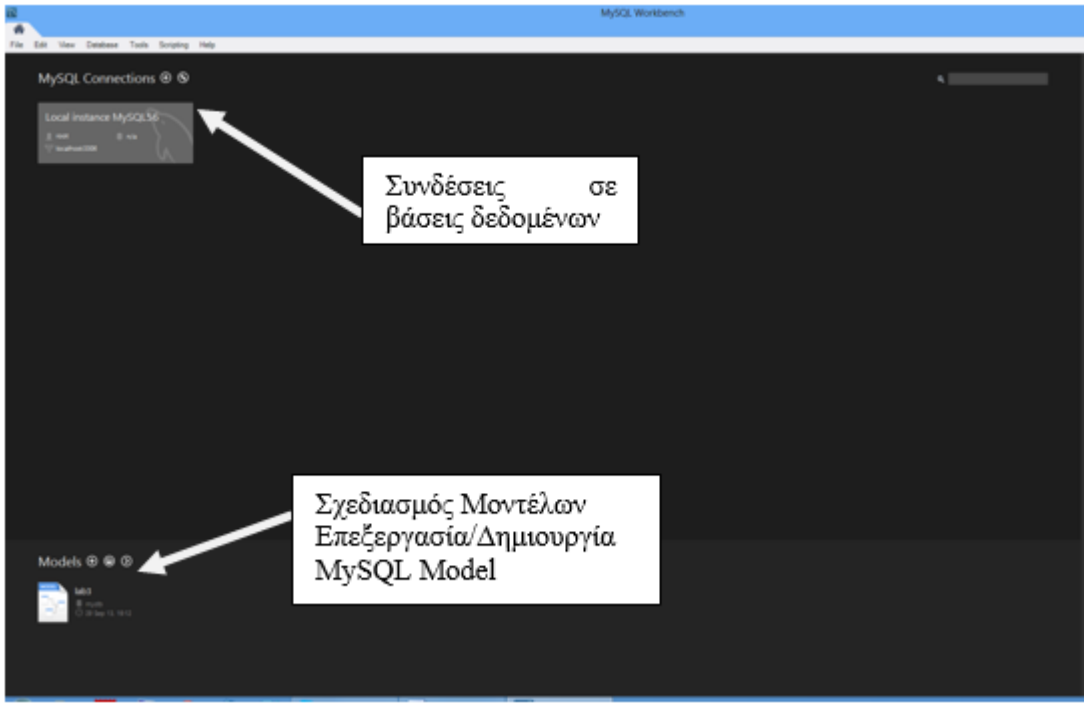
χαρακτηριστικά που περιγράφουν την σχέση ενός φοιτητή με ένα μάθημα είναι ο βαθμός εξέτασης καθώς και η ημερομηνία εξέτασης του μαθήματος. Κάθε μαθητής μπορεί να συσχετίζεται με πολλά μαθήματα συνεπώς ο τύπος συσχέτισης είναι M:N.

- **Μάθημα – ΠΡΟΑΠΑΙΤΕΙ (Requires) – Μάθημα:** Κάθε μάθημα μπορεί να έχει ως προαπαιτούμενα (αν έχει) ένα ή περισσότερα μαθήματα. Το ζευγάρι κωδικών μαθημάτων είναι αρκετό να αναπαραστήσει την συσχέτιση αυτή η οποία είναι τύπου M:N.
- **Καθηγητής – ΔΙΔΑΣΚΕΙ (Teach)– Μάθημα:** Σύμφωνα με το πρόβλημα κάθε μάθημα διδάσκεται από έναν μόνο καθηγητή συνεπώς η συσχέτιση είναι τύπου 1:N καθώς είναι δυνατό ένας καθηγητής να διδάσκει ένα ή περισσότερα μαθήματα.

Ζητούμενο 2: Σχεδιασμός του διαγράμματος Οντοτήτων-Συσχετίσεων (ER diagram)

Για τον σχεδιασμό του διαγράμματος Οντοτήτων θα χρησιμοποιηθεί το εργαλείο MySQL Model του MySQL Workbench[1]. Το MySQL Model είναι ένα γραφικό περιβάλλον που παρέχεται από το σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων MySQL. Μέσω του εργαλείου αυτού είναι δυνατό να σχεδιαστούν τα διαγράμματα Οντοτήτων-Συσχετίσεων (ER diagrams), να περιγραφούν οι πίνακες του σχεσιακού μοντέλου (οι πίνακες, τα χαρακτηριστικά, οι τύποι των χαρακτηριστικών κλπ.) καθώς και οι διάφοροι περιορισμοί ακεραιότητας όπως τα πρωτεύοντα και δευτερεύοντα κλειδιά. Συνεπώς το εργαλείο αυτό είναι κατάλληλο για την καταγραφή και οπτικοποίησης των ζητούμενων 1-5 της παρούσας άσκησης. Εν συνεχεία ο σχεδιασμός είναι δυνατό να δημιουργήσει την βάση δεδομένων αυτόματα στο εξυπηρετητή της MySQL (Ζητούμενο 6). Στην περίπτωση μας το ζητούμενο 6 θα το εκτελέσουμε με χρήση εντολών SQL-DDL (SQL- data definition language).

Η έναρξη του MySQL Model είναι δυνατή από την αρχική σελίδα του MySQL Workbench (βλέπε Εικόνα 3.1).

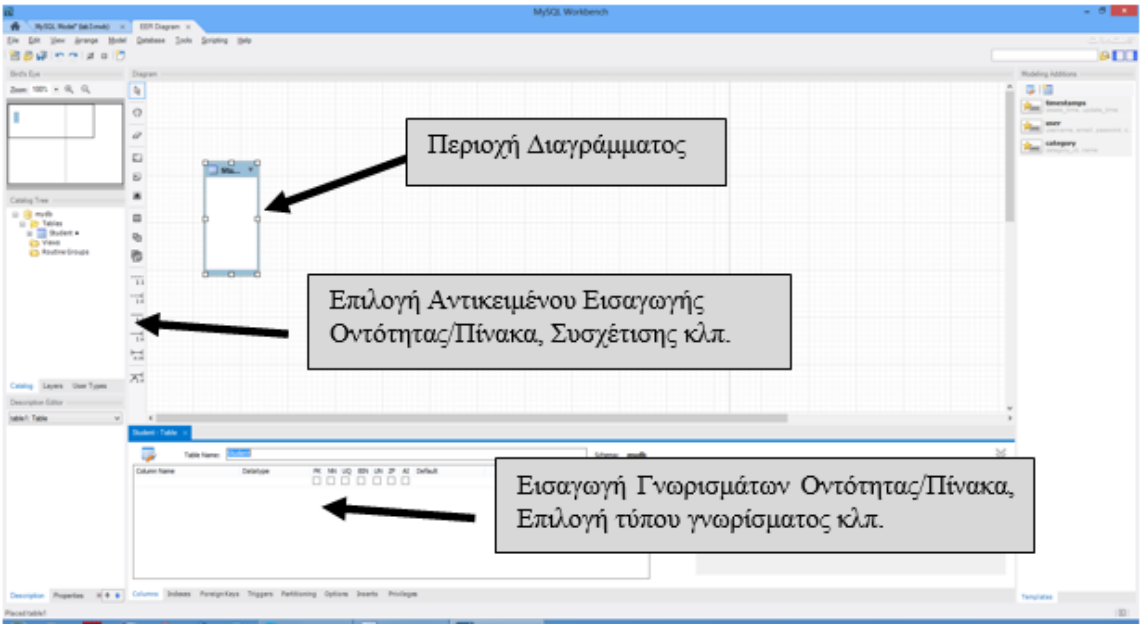


Εικόνα 3.1: *Αρχική οθόνη MySQL Workbench.*

[1] Για εξοικείωση με το περιβάλλον MySQL Workbench ανατρέξτε στο εργαστήριο 1.

Οι βασικές λειτουργίες του περιβάλλοντος MySQL Model είναι οι ακόλουθες ενώ τα βασικά τμήματα της εφαρμογής παρουσιάζονται στην Εικόνα 3.2:

- Σχεδιασμός Οντοτήτων-Συσχετίσεων
- Επιλογή γνωρισμάτων οντοτήτων και επιλογή του τύπου τους
- Επιλογή περιορισμών ακεραιότητας
- Επιλογή τύπων συσχέτισης

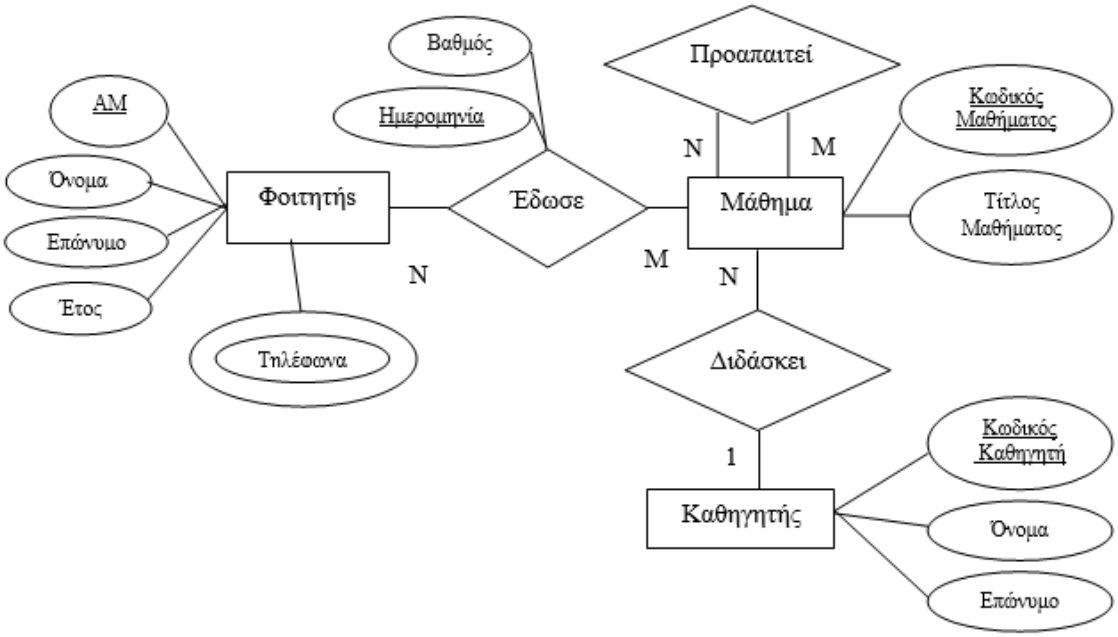


Εικόνα 3.2: *MySQL Model - βασικές επιλογές.*

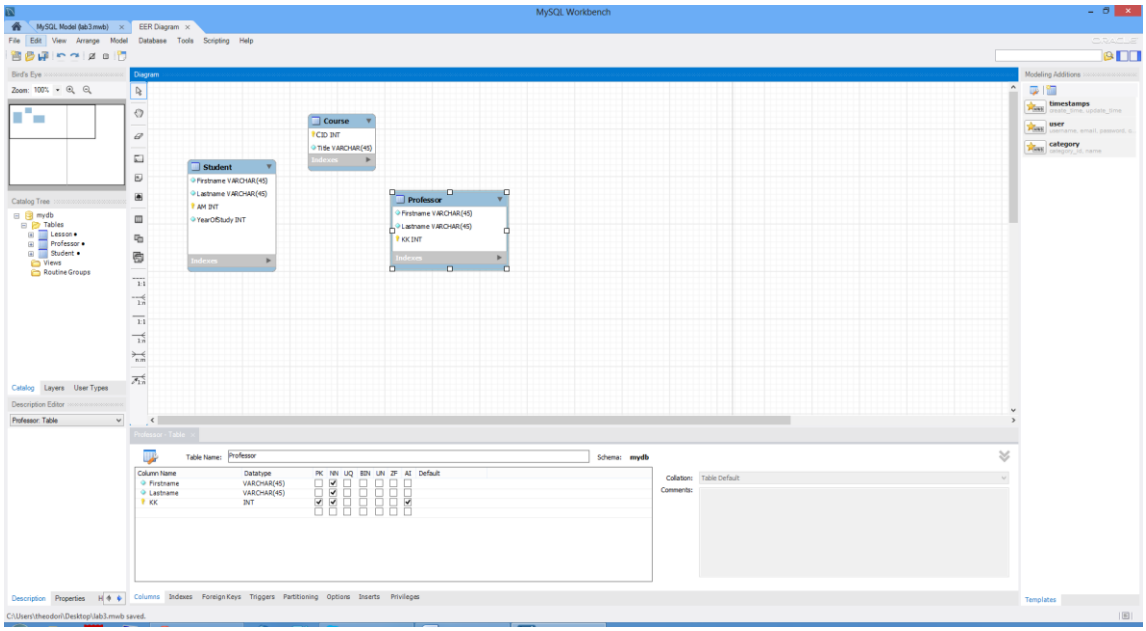
Αρχικά σχεδιάζουμε τις τρείς οντότητες του προβλήματος **φοιτητής (student)**, **καθηγητής (professor)** και **μάθημα (course)**. Για κάθε οντότητα εισάγουμε τα γνωρίσματα τους καθώς και τον τύπο των γνωρισμάτων.

- **Φοιτητής:**
  - Όνομα - συμβολοσειρά
  - Επώνυμο - συμβολοσειρά
  - Αριθμό Μητρώου (ΑΜ) - ακέραιος
  - Έτος - ακέραιος
  - Σύνολο τηλεφώνων – σύνολο από συμβολοσειρές
- **Καθηγητής:**
  - Όνομα - συμβολοσειρά
  - Επώνυμο - συμβολοσειρά
  - Κωδικός Καθηγητή (ΚΚ) - ακέραιος
- **Μάθημα:**
  - Τίτλος Μαθήματος- συμβολοσειρά
  - Κωδικός Μαθήματος (ΚΜ) - ακέραιος

Το πλήρες διάγραμμα οντοτήτων συσχετίσεων φαίνεται στην Εικόνα 3.3. Παρατηρήστε ότι το γνώρισμα Τηλέφωνα είναι γνώρισμα πολλαπλών τιμών και ότι η συσχέτιση Προαπαιτεί είναι αυτοσυσχέτιση. Οι ρόλοι (δεν αναγράφονται στο ER για απλότητα) είναι: (i) το μάθημα που προαπαιτεί (ΚΜ1) και (ii) το μάθημα που προαπαιτείται (ΚΜ2).



Εικόνα 3.3: *Διάγραμμα Οντοτήτων-Συσχετίσεων.*



Εικόνα 3.4: *Οντότητες και γνωρίσματα τους στο MySQL Model.*

Ζητούμενο 3: Σχεδιασμός πινάκων σχεσιακού μοντέλου.

Για την αναπαράσταση της παραπάνω μοντελοποίησης (ER diagram - Εικόνα 3.3, Εικόνα 3.4) θα χρειαστούν αρχικά τρείς πίνακες μία για κάθε μία οντότητα.

- Φοιτητής (ΑΜ, Όνομα, Επώνυμο, Έτος)
- Μάθημα (Κωδικός Μαθήματος, Τίτλος Μαθήματος, Κωδικός Καθηγητή)
- Καθηγητής (Κωδικός Καθηγητή, Όνομα, Επώνυμο)

Για την αναπαράσταση και αποθήκευση των συσχετίσεων μεταξύ των οντοτήτων θα χρειαστούν οι ακόλουθοι πίνακες:

- Έδωσε (ΑΜ Φοιτητή, Κωδικός Μαθήματος, Ημερομηνία, Βαθμός). Παρατηρήστε ότι μοναδικό γνώρισμα είναι ο συνδυασμός του ΑΜ, του κωδικού μαθήματος και της ημερομηνίες καθώς αυτά εξασφαλίζουν την μοναδική ταυτοποίηση ενός βαθμού που πήρε ένας φοιτητής σε μία εξέταση.

- Προαπαιτεί (Κωδικός Μαθήματος1, Κωδικός Μαθήματος2). Παρατηρήστε ότι μοναδικό γνώρισμα είναι ο συνδυασμός των δύο κωδικών.

Επίσης παρατηρείστε ότι η συσχέτιση διδάσκει υλοποιείται μέσω του πεδίου Κωδικός καθηγητή στον πίνακα μάθημα και αυτό γιατί κάθε μάθημα διδάσκεται από έναν και μόνο καθηγητή. Για την αναπαράσταση ενός γνωρίσματος πολλαπλών τιμών όπως τα τηλέφωνα στην οντότητα Φοιτητής θα χρησιμοποιηθεί ο πίνακας:

- Τηλέφωνα (ΑΜ Φοιτητή, Αριθμός Τηλεφώνου)

Παρατίθενται οι ορισμοί των παραπάνω πινάκων/γνωρισμάτων με λατινικά ονόματα μίας και στη συνέχεια αυτά θα χρησιμοποιηθούν στον προγραμματισμό των εντολών SQL για την δημιουργία τους (Βλέπε και Εικόνα 3.5).

- Student (AM, Firstname, Lastname, YearOfStudy)

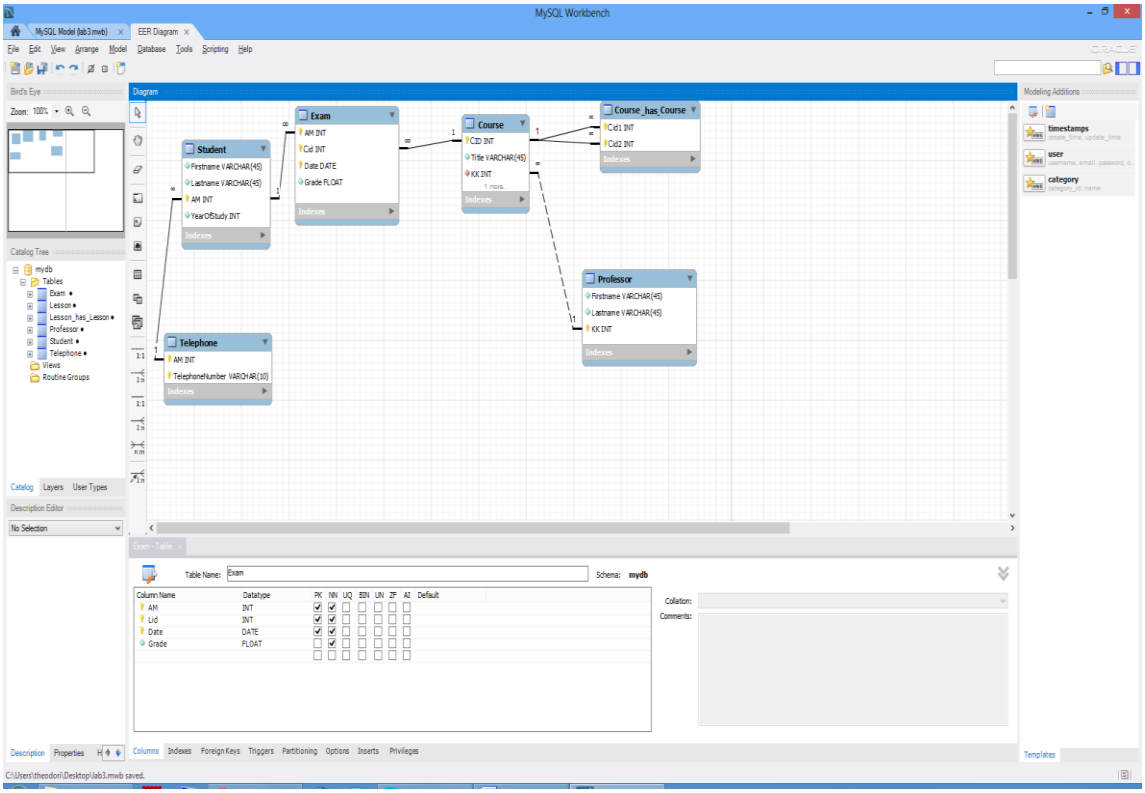
- Course (Cid, Title, Pid)

- Professor (KK, Firstname, Lastname)

- Exam (AM, Cid, Date, Grade).

- Requires (Cid1, Cid2)

- Telephone (AM, TelphoneNumber)



Εικόνα 3.5: *Επαυξημένο Διάγραμμα Οντοτήτων-Συσχετίσεων (EER diagram MySQL Model).*

Ζητούμενο 5: Εφαρμογή Περιορισμών ακεραιότητας δεδομένων

Οι περιορισμοί ακεραιότητας των δεδομένων στο παραπάνω σχεσιακό μοντέλο είναι οι ακόλουθοι:

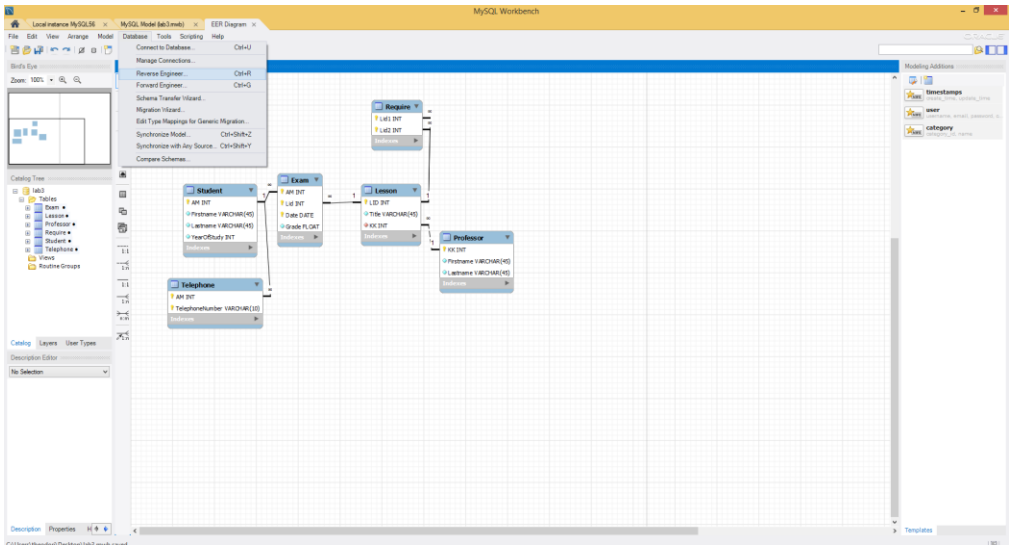
- Πρωτεύοντα Κλειδιά (Primary Keys):
    - Φοιτητής (Student): το AM είναι πρωτεύον κλειδί – ακέραιος. Αν η εφαρμογή μας είναι η βάση που ανατίθενται τα AM στους φοιτητές θα μπορούσε να γίνει και auto_increment. Δίνεται νέο AM προσθέτοντας μία τιμή (συνήθως +1) στην τελευταία τιμή που έχει χρησιμοποιηθεί.
    - Τηλέφωνα – Έχουμε σύνθετο κλειδί καθώς κάθε ζευγάρι από AM και έναν αριθμό θα θέλαμε να αποθηκευτούν μόνο μία φορά στον πίνακα.
    - Μάθημα (Course) : το Cid (κωδικός μαθήματος) είναι το πρωτεύον κλειδί– ακέραιος. Αν η εφαρμογή μας είναι η βάση που ανατίθενται τα Cid στα μαθήματα θα μπορούσε να γίνει auto_increment.
    - Προαπαιτεί (Requires): Έχουμε σύνθετο κλειδί με το ζευγάρι των κωδικών μαθημάτων καθώς θέλουμε η προαπαίτηση να αποθηκεύεται μία φορά στον πίνακα.
    - Καθηγητής (Professor): Ο KK είναι πρωτεύον κλειδί και μπορεί να ορισθεί και ως auto_increment.
    - Έδωσε (Exam): Το πρωτεύον κλειδί σε αυτή την περίπτωση αποτελείται από τα τρία γνωρίσματα AM, Cid, Date για να ορισθεί μοναδικά κάθε εξέταση ενός φοιτητή σε ένα μάθημα.
- Ξένα Κλειδιά (Foreign Keys):
    - Τηλέφωνα – Το πεδίο AM στο πίνακα είναι ξένο κλειδί στον πίνακα φοιτητής.
    - Μάθημα (Course) : το πεδίο KK είναι ξένο κλειδί στον πίνακα καθηγητής.
    - Προαπαιτεί (Requires): Και τα δύο κλειδιά είναι ξένα κλειδιά στον πίνακα Μάθημα.
    - Έδωσε (Exam): Τα κλειδιά AM και Cid είναι ξένα κλειδιά στον πίνακα Φοιτητή και Μάθημα αντίστοιχα.

Ζητούμενο 6: Υλοποίηση της ΒΔ και των αντίστοιχων πινάκων με χρήση SQL-DDL εντολών με χρήση του εργαλείου MySQL Workbench.

```
CREATE SCHEMA `university`;
USE `university`;
-- -----------------------------------------------------
-- Table `university`.`Student`
-- -----------------------------------------------------
CREATE TABLE `university`.`Student` (
  `AM` INT NOT NULL AUTO_INCREMENT,
  `Firstname` VARCHAR(45) NOT NULL,
  `Lastname` VARCHAR(45) NOT NULL,
  `YearOfStudy` INT NOT NULL,
  PRIMARY KEY (`AM`)
);


-- -----------------------------------------------------
-- Table `university`.`Professor`
-- -----------------------------------------------------
CREATE TABLE `university`.`Professor` (
  `KK` INT NOT NULL AUTO_INCREMENT,
  `Firstname` VARCHAR(45) NOT NULL,
  `Lastname` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`KK`)
);


-- -----------------------------------------------------
-- Table `university`.`Course`
-- -----------------------------------------------------
CREATE TABLE `university`.`Course` (
  `CID` INT NOT NULL AUTO_INCREMENT,
  `Title` VARCHAR(45) NOT NULL,
  `KK` INT NOT NULL,
  PRIMARY KEY (`CID`),
  CONSTRAINT `fk_Course_Professor`
    FOREIGN KEY (`KK`)
    REFERENCES `university`.`Professor` (`KK`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);


-- -----------------------------------------------------
-- Table `university`.`Exam`
-- -----------------------------------------------------
CREATE TABLE `university`.`Exam` (
  `AM` INT NOT NULL,
  `Cid` INT NOT NULL,
  `Date` DATE NOT NULL,
  `Grade` FLOAT NOT NULL,
  PRIMARY KEY (`AM`, `Cid`, `Date`),
  CONSTRAINT `fk_Student_Exam`
    FOREIGN KEY (`AM`)
    REFERENCES `university`.`Student` (`AM`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Exam_Course`
    FOREIGN KEY (`Cid`)
    REFERENCES `university`.`Course` (`CID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
```

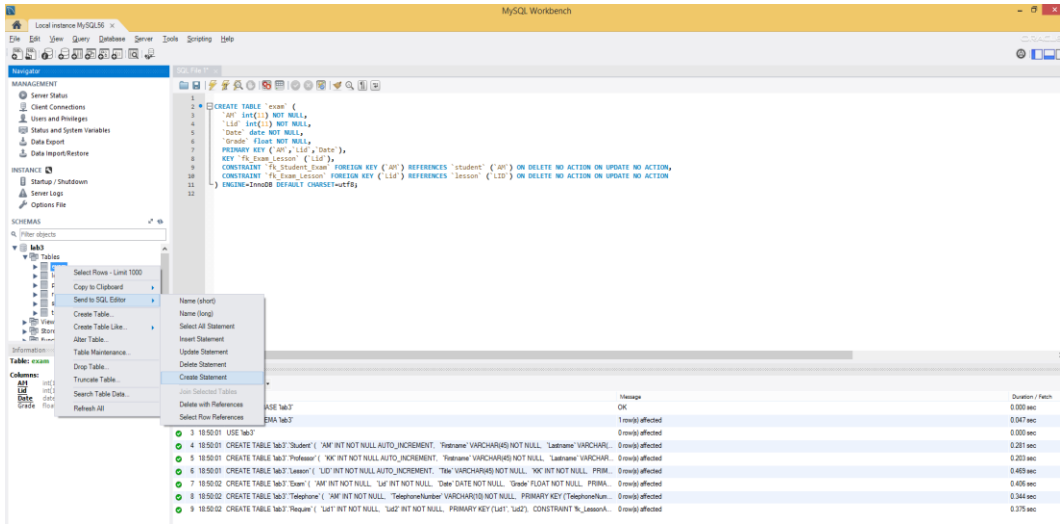
```
-- -----------------------------------------------------
-- Table `university`.`Telephone`
-- -----------------------------------------------------
CREATE TABLE `university`.`Telephone` (
  `AM` INT NOT NULL,
  `TelephoneNumber` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`TelephoneNumber`, `AM`),
  CONSTRAINT `fk_Telephone_Student`
    FOREIGN KEY (`AM`)
    REFERENCES `university`.`Student` (`AM`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);


-- -----------------------------------------------------
-- Table `university`.`Requires`
-- -----------------------------------------------------
CREATE TABLE `university`.`Requires` (
  `Cid1` INT NOT NULL,
  `Cid2` INT NOT NULL,
  PRIMARY KEY (`Cid1`, `Cid2`),
  CONSTRAINT `fk_CourseA_Requires`
    FOREIGN KEY (`Cid1`)
    REFERENCES `university`.`Course` (`CID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Requires_CourseB`
    FOREIGN KEY (`Cid2`)
    REFERENCES `university`.`Course` (`CID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
```

Το MySQL Workbench είναι δυνατόν να μας παράγει αυτόματα τον πηγαίο κώδικα SQL ορισμού και δημιουργίας των πινάκων από το μοντέλο EER που έχουμε δημιουργήσει. Για να το εκτελέσουμε αυτό επιλέγουμε την ενέργεια Forward Engineer (βλέπε Εικόνα 3.6).



Εικόνα 3.6: *Ενέργεια Forward Engineer για δημιουργία/εκτέλεση του κώδικα SQL-DDL με αποτέλεσμα την δημιουργία της ΒΔ στον εξυπηρετητή της MySQL.*

Επιπρόσθετα από τη ΒΔ που έχουμε δημιουργήσει στο παρελθόν είναι δυνατόν να εξάγουμε τον κώδικα SQL-DDL που ορίζει/δημιουργεί τους πίνακες της βάσεις καθώς και τους διάφορους περιορισμούς ακεραιότητας ή τους καταλόγους κλπ. Όπως φαίνεται και στην Εικόνα 3.7, επιλέγουμε «δεξί κλικ» πάνω στον πίνακα της βάσης που επιθυμούμε και στην συνέχεια «Send to SQL editor» → «Create Statement».



Εικόνα 3.7: *Εξαγωγή του κώδικα SQL-DDL από τον MySQL εξυπηρετητή.*

3.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Δημιουργήστε μία βάση δεδομένων για μία δανειστική βιβλιοθήκη. Οι οντότητες που υπάρχουν στην βάση αυτή είναι: οι **Χρήστες** που περιγράφονται από ένα μοναδικό αναγνωριστικό, όνομα, επώνυμο διεύθυνση, τκ, δήμος, νομός τηλέφωνο, email, επάγγελμα, τα **Βιβλία** που περιγράφονται από ένα μοναδικό αναγνωριστικό, τίτλο, συγγραφείς, και **Συγγραφείς** που περιγράφονται από ένα μοναδικό αναγνωριστικό, όνομα, επώνυμο και χώρα. Τέλος για κάθε βιβλίο διατηρείται ο αριθμός των αντιτύπων που διαθέτει η βιβλιοθήκη ενώ για το δανεισμό ενός βιβλίο διατηρείται η ημερομηνία λήψης και η ημερομηνία επιστροφής του βιβλίου.

Ζητούμενα:

1. Δημιουργήστε το διάγραμμα ER.
2. Δημιουργήστε το διάγραμμα EER model στο MySQL Workbench.
3. Δημιουργήστε την ΒΔ library.
4. Δημιουργήστε τους πίνακες του παραπάνω προβλήματος.

Άσκηση 2

Δημιουργήστε μία βάση δεδομένων για ένα μαγαζί ηλεκτρικών ειδών. Οι οντότητες που υπάρχουν στην βάση αυτή είναι: τα **Προϊόντα** που περιγράφονται από ένα μοναδικό αναγνωριστικό, όνομα, περιγραφή, εταιρεία παραγωγής, τύπος προϊόντος(tv,ac,ψυγείο κλπ.), τιμή, προμηθευτής και **Προμηθευτές** που περιγράφονται από το ΑΦΜ, επωνυμία, διεύθυνση. Τέλος για κάθε προϊόν διατηρείτε η ποσότητα του αποθέματος και η τιμή που αγοράστηκε από τον προμηθευτή καθώς και η τελική τιμή πώλησης.

Ζητούμενα:

1. Δημιουργήστε το διάγραμμα ER.
2. Δημιουργήστε το διάγραμμα EER model στο MySQL Workbench.
3. Δημιουργήστε την ΒΔ library.
4. Δημιουργήστε τους πίνακες του παραπάνω προβλήματος.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαιο 8.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
CREATE TABLE.
http://dev.mysql.com/doc/refman/5.7/en/create-table.html
http://www.w3schools.com/sql/sql_create_table.asp
MySQL Datatypes. http://dev.mysql.com/doc/refman/5.7/en/data-types.html
Data Definition. http://dev.mysql.com/doc/refman/5.7/en/sql-syntax-data-definition.html
NOT NULL. http://www.w3schools.com/sql/sql_notnull.asp
DEFAULT. http://www.w3schools.com/sql/sql_default.asp
PRIMARY KEY. http://www.w3schools.com/sql/sql_primarykey.asp
FOREIGN KEY. http://www.w3schools.com/sql/sql_foreignkey.asp
AUTO_INCREMENT. http://www.w3schools.com/sql/sql_autoincrement.asp

Κεφάλαιο 4 Εισαγωγή Εγγραφών και Τροποποίηση Πινάκων

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν επιπλέον εντολές του DML που έχουν να κάνουν με εισαγωγή εγγραφών, επισκόπηση και τροποποίηση πινάκων.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι οι ακόλουθες ενότητες:

- **Εργαστήριο 3:** Δημιουργία ΒΔ και Πινάκων.

4.1 Εισαγωγικές Έννοιες

Έχοντας δημιουργήσει τους πίνακες Customer, Phones, Account, Owns, Action, Transfer μπορούμε να αρχίσουμε να «γεμίζουμε» τη ΒΔ εισάγοντας εγγραφές. Δύο τρόποι υπάρχουν για την εισαγωγή εγγραφών. Ο πρώτος είναι χρησιμοποιώντας την εντολή INSERT..VALUES.. με την οποία εισάγουμε μία ή περισσότερες εγγραφές των οποίων το περιεχόμενο δίνεται μαζί με την εντολή. Ο δεύτερος είναι φορτώνοντας εγγραφές από αρχείο με την εντολή LOAD DATA.. INFILE.

4.1.1 Εισάγοντας εγγραφές με την INSERT

Για την πλήρη σύνταξη της εντολής ο αναγνώστης μπορεί να ανατρέξει στο Κεφ. 13.2.5 του εγχειριδίου της MySQL5.7. Ο βασικός σκελετός της εντολής έχει ως εξής:

INSERT *επιλογές* [INTO] *όνομα_πίνακα* VALUES (*εγγραφή*), (*εγγραφή*),…

Για παράδειγμα με την ακόλουθη εντολή εισάγουμε δύο καινούργιες εγγραφές στον πίνακα Customer (cid, afm, address, name, sname, dateOfBirth):

```
INSERT INTO Customer VALUES
(1, '077783234', '56 Baltetsiou st.', 'Kostas', 'Kostantinou', '1990-10-30'),
(2, '175783239', '107 Diakou st.', 'Eleni', 'Kostantinou', '1985-11-02');
```

Στη θέση του VALUES μπορούμε να γράψουμε VALUE χωρίς αυτό να επηρεάζει σε τίποτα ή να υπονοεί το πλήθος των εγγραφών που πρέπει να περαστούν. Για λόγους αναγνωσιμότητας είναι καλό όταν εισάγουμε πολλές εγγραφές με μία εντολή INSERT κάθε εγγραφή να καταλαμβάνει ξεχωριστή γραμμή. Παρατηρήστε ότι για αλφαριθμητικές τιμές και ημ/νιες, οι τιμές περικλείονται μέσα σε ‘’, ενώ οι αριθμητικές όχι.

Προσοχή: Αν και στο σχεσιακό μοντέλο η σειρά των πεδίων σε έναν πίνακα δεν παίζει ρόλο στην SQL παίζει. Η καταχώρηση εγγραφής γίνεται βάζοντας τιμές στα πεδία του πίνακα με τη σειρά με την οποία ορίστηκαν.

Μία χρήσιμη επιλογή στην εντολή INSERT είναι η IGNORE. Το INSERT IGNORE λειτουργεί όπως και το απλό INSERT μόνο που σε περίπτωση σφάλματος στην εισαγωγή μιας εγγραφής, αντί η εκτέλεση να διακοπεί, συνεχίζεται με αγνόηση του σφάλματος. Για παράδειγμα έστω ότι δοκιμάζουμε να προσθέσουμε τις ακόλουθες δύο εγγραφές (αφού βάλαμε τις προηγούμενες δύο):

```
INSERT INTO Customer VALUES
(3, '095111139', '12 Rodon st.', 'Maria', 'Papantoniou', '1967-3-20'),
```

(2, '175744444', '107 Diakou st.', 'Eleni', 'Kostantinou', '1985-11-02');

Θα λάβουμε μήνυμα σφάλματος εξαιτίας της δεύτερης εγγραφής, καθώς υπάρχει ήδη καταχωρημένη εγγραφή με τιμή 2 στο primary key. Ως εκ τούτου δε θα καταχωρηθεί ούτε η πρώτη εγγραφή αν και από μόνη της δεν ενέχει κάποιο σφάλμα. Χρησιμοποιώντας την επιλογή IGNORE ως εξής:

INSERT IGNORE INTO Customer VALUES
(3, '095111139', '12 Rodon st.', 'Maria', 'Papantoniou', '1967-3-20'),
(2, '175744444', '107 Diakou st.', 'Eleni', 'Kostantinou', '1985-11-02');

η πρώτη εγγραφή θα καταχωρηθεί και για τη δεύτερη θα υπάρξει warning αντί για σφάλμα. Μπορούμε να δούμε το warning με την εντολή:

SHOW WARNINGS;

Στη συγκεκριμένη περίπτωση θα δούμε σε μορφή πίνακα το εξής:
1062 Duplicate entry '2' for key 'PRIMARY'

Θα πρέπει να σημειωθεί ότι πιθανά σφάλματα κατά την εισαγωγή εγγραφών συνήθως περιλαμβάνουν: (i) αναντιστοιχία τύπων πχ. αλφαριθμητικό σε DATE τιμή, (ii) αναντιστοιχία πλήθους πεδίων πχ. όταν επιχειρούμε να βάλουμε εγγραφή με λιγότερα ή περισσότερα πεδία από αυτά που υπάρχουν στον πίνακα, (iii) όταν προσπαθούμε να θέσουμε ένα NOT NULL πεδίο σε NULL (iv) παραβίαση PRIMARY KEY, UNIQUE KEY ή FOREIGN KEY.

Προσοχή: Χωρίς IGNORE και στις 4 παραπάνω περιπτώσεις θα έχουμε σφάλμα. Με IGNORE θα έχουμε σφάλμα στην περίπτωση (ii), warnings στις άλλες τρεις. Στις περιπτώσεις (i) και (iii) θα μπει καινούργια εγγραφή ενώ στην (iv) όχι.

Συνοψίζοντας, η χρήση της IGNORE είναι αρκετά συνηθισμένη όταν θέλουμε να εισάγουμε πολλές εγγραφές με μία εντολή αγνοώντας τις εισαγωγές που παραβιάζουν PRIMARY ή UNIQUE KEY. Αξίζει να σημειωθεί ότι η εισαγωγή πολλών εγγραφών (batch insertion) μπορεί να γίνει και με πολλαπλές εντολές INSERT αντί για μία, αλλά θα είναι αισθητά πιο αργή για μικρό και μεσαίο πλήθος εγγραφών, πχ. στην κλίμακα των 1000 εγγραφών.

Προσοχή: Υπάρχει ένα φυσικό όριο στο μέγεθος κάθε SQL εντολής που μπορεί να σταλεί από έναν client στον MySQL server. Το όριο φαίνεται στην παράμετρο αρχικοποίησης max_allowed_packet. Η default τιμή είναι 4M τόσο για τον client mysql όσο και για τον server mysqld. Μπορεί να αλλαχθεί ως argument κατά την κλήση των server/client πχ.:
mysql --max_allowed_packet=32M
Το επιτρεπτό μέγεθος δεν μπορεί να ξεπεράσει το 1G.

Μπορούμε να δούμε τις καταχωρημένες εγγραφές στον πίνακα Customer δίνοντας την ακόλουθη εντολή:
SELECT * FROM Customer;
που αποτελεί ίσως το πιο απλό ερώτημα (query) που μπορούμε να συντάξουμε. Περισσότερα για ερωτήματα και τη χρήση της SELECT FROM WHERE σε επόμενο κεφάλαιο.

Αν επιθυμούμε να καταχωρήσουμε μια εγγραφή χρησιμοποιώντας τις default τιμές σε κάποια πεδία αναγράφουμε στο αντίστοιχο πεδίο DEFAULT. Η επιλογή αυτή είναι ιδιαίτερα χρήσιμη για την καταχώρηση τιμών σε πεδία AUTO_INCREMENT. Για παράδειγμα η εντολή:

INSERT INTO Customer VALUES
(DEFAULT, '175744444', DEFAULT, 'Eleni', 'Kostantinou', DEFAULT);

θα καταχωρήσει μία καινούργια εγγραφή με address ‘Unknown’, dateOfBirth NULL και cid την τιμή του AUTO_INCREMENT.

Μπορούμε να επιλέξουμε τα πεδία στα οποία θα καταχωρηθούν τιμές, ενώ στα υπόλοιπα πεδία καταχωρείται η default τιμή. Έτσι η προηγούμενη εγγραφή θα μπορούσε να είχε εισαχθεί και ως εξής:

INSERT INTO Customer (afm, name, sname) VALUES ('175744444', 'Eleni', 'Kostantinou');

4.1.2 Εισάγοντας και ενημερώνοντας

Πολύ συχνά παρουσιάζεται η ανάγκη να εισάγουμε μία εγγραφή και στην περίπτωση που υπάρχει ήδη κάποια με ίδιο PRIMARY ή UNIQUE KEY η προϋπάρχουσα να αντικατασταθεί από την καινούργια. Στη MySQL μπορούμε να το επιτύχουμε είτε με την εντολή REPLACE ή με την INSERT.. ON DUPLICATE KEY UPDATE. Οι δύο αυτές εντολές δεν ανήκουν στο στάνταρ πρότυπο της SQL και ως εκ τούτου ο κώδικας που τις χρησιμοποιεί ενδέχεται να μην είναι μεταφέρσιμος.

Η REPLACE είναι ισοδύναμη με την INSERT όπως παρουσιάστηκε στην προηγούμενη υποενότητα με τη διαφορά ότι σε περίπτωση σύγκρουσης σε PRIMARY ή UNIQUE KEY η υπάρχουσα εγγραφή στον πίνακα διαγράφεται και στη συνέχεια προστίθεται η καινούργια. Για παράδειγμα υποθέτοντας ότι έχουμε την εγγραφή:
<4, 175744444, Unknown, Eleni, Kostantinou, NULL>
και δώσουμε την εντολή:
REPLACE INTO Customer (cid, address) VALUES (4, '30 Papanikoli st.');
στη θέση της προηγούμενης εγγραφής θα έχουμε:
<4, NULL, 30 Papanikoli st., Unknown, Unknown, NULL>

Η INSERT..ON DUPLICATE KEY UPDATE δε σβήνει την υπάρχουσα εγγραφή, αλλά την ενημερώνει. Μετά το UPDATE βάζουμε ένα ή περισσότερα πεδία και σε τι θέλουμε να αλλάξει η τιμή τους. Κάθε ανάθεση χωρίζεται με κόμμα. Για παράδειγμα έχοντας ήδη την εγγραφή:
<4, NULL, 30 Papanikoli st., Unknown, Unknown, NULL>
η εντολή:

```
INSERT INTO Customer (cid, afm, name, sname, dateOfBirth)
VALUES (4, '175744444', 'Eleni', 'Ioannou', '1990-3-2') ON DUPLICATE KEY UPDATE
afm=VALUES(afm),
name=VALUES(name),
sname=VALUES(sname),
dateOfBirth=VALUES(dateOfBirth);
```

θα αλλάξει την εγγραφή σε: <4, 175744444, 30 Papanikoli st., Eleni, Ioannou, 1990-3-2>. Η ανάθεση πχ. afm=VALUES(afm) δηλώνει ότι το πεδίο afm στην εγγραφή που υπάρχει θα πάρει την αντίστοιχη τιμή του afm που υπάρχει στο VALUES. Δε χρειάζεται να αναφέρουμε τις τιμές που θα πάρουν όλα τα πεδία αν αυτό δεν είναι απαραίτητο. Τέλος, στην ανάθεση μπορούμε να έχουμε οποιαδήποτε επιτρεπτή έκφραση.

Πριν προχωρήσουμε στην επόμενη υποενότητα θα θέλαμε να τονίσουμε τα εξής ως προς τη χρήση των REPLACE και ON DUPLICATE KEY UPDATE:

- Όταν χρησιμοποιούμε τη REPLACE θα πρέπει να είμαστε προσεκτικοί στην περίπτωση που υπάρχει FOREIGN KEY. Καθώς πρώτα διαγράφεται η υπάρχουσα εγγραφή ενεργοποιείται το ON DELETE και όχι το ON UPDATE.
- Η χρήση και των δύο εντολών αντενδείκνυται όταν στόχος μας είναι απλά να ενημερώσουμε μία ή περισσότερες εγγραφές, κάτι το οποίο γίνεται πολύ πιο αποδοτικά με τη χρήση της UPDATE..SET..WHERE που περιγράφεται σε επόμενο κεφάλαιο.

Ο αναγνώστης μπορεί να αναφερθεί στα Κεφ. 13.2.8 και Κεφ. 13.2.5.3 του εγχειριδίου της MySQL 5.7 για περισσότερες πληροφορίες σχετικά με τις εντολές REPLACE και INSERT..ON DUPLICATE KEY UPDATE.

4.1.3 Η εντολή LOAD DATA INFILE

Όταν έχουμε πολλές εγγραφές προς εισαγωγή είναι προτιμητέα από άποψη απόδοσης η εισαγωγή των εγγραφών από αρχείο (αν υπάρχει) με χρήση της LOAD DATA INFILE σε σχέση με τη χρήση της INSERT. Λεπτομέρειες για τη LOAD DATA INFILE μπορεί να βρεθούν στο Κεφ. 13.2.6 του εγχειριδίου της MySQL. Εδώ παρουσιάζουμε τα πιο βασικά στοιχεία της σύνταξης που έχουν ως εξής:

```
LOAD DATA [LOCAL] INFILE 'όνομα_αρχείου' [REPLACE | IGNORE]
INTO TABLE όνομα_πίνακα
[{FIELDS | COLUMNS}
        [TERMINATED BY 'αλφαριθμητικό']
        [[OPTIONALLY] ENCLOSED BY 'χαρακτήρα']
        [ESCAPED BY 'χαρακτήρα']
]
[LINES
        [STARTING BY 'αλφαριθμητικό']
        [TERMINATED BY 'αλφαριθμητικό']
]
```

LOCAL: Η επιλογή σημαίνει ότι η αναζήτηση του αρχείου θα γίνει στο μηχάνημα του client, αλλιώς η αναζήτηση του αρχείου θα γίνει στη μεριά του server.

'όνομα_αρχείου': Το αρχείο πρέπει να είναι text. Μπορεί να έχουμε απλά το όνομα του αρχείου, καθώς επίσης να ορίσουμε σχετικό ή απόλυτο μονοπάτι. Αν δεν έχουμε απόλυτο μονοπάτι η αναζήτηση για το αρχείο αν πρόκειται για τον client θα γίνει ξεκινώντας από το directory στο οποίο βρίσκεται ο client, ενώ αν πρόκειται για το server από το data directory. Στα MS Windows το path δίνεται με το χαρακτήρα \ (backslash). Επειδή το \ σε αλφαριθμητικά χρησιμεύει για καθορισμό ειδικών χαρακτήρων θα πρέπει τα directories του path να ξεχωρίζουνε με διπλό backslash δηλ. \\. Σε linux η χρήση του slash / δε δημιουργεί κανένα πρόβλημα.

REPLACE ή IGNORE: Το IGNORE στη LOAD DATA έχει αντίστοιχη λειτουργία με το IGNORE στην INSERT δηλ. λάθη λόγω σύγκρουσης σε κλειδιά αγνοούνται και οδηγούν σε warnings. Το REPLACE στη LOAD DATA έχει ακριβώς την ίδια σημασία με την εντολή REPLACE, δηλ. σε περίπτωση σύγκρουσης σε κάποιο μοναδικό κλειδί, η υπάρχουσα εγγραφή αντικαθίσταται. Η χρήση των REPLACE και IGNORE είναι προαιρετική.

FIELDS ή COLUMNS: Ισοδύναμες δηλώσεις. Ξεκινούν τη δήλωση του τρόπου με τον οποίο το αρχείο εισόδου μορφοποιεί τις στήλες (πεδία). Η δήλωση είναι προαιρετική.

TERMINATED BY: Προσδιορίζεται ο τρόπος με τον οποίο διαχωρίζονται οι στήλες μεταξύ τους. Η default τιμή είναι το '\t' δηλ. ο χαρακτήρας tab.

ENCLOSED BY: Προσδιορίζει αν οι τιμές των πεδίων περικλείονται από κάποιο χαρακτήρα. Ο χαρακτήρας που περικλείει την τιμή αγνοείται κατά την εισαγωγή. Η προαιρετική επιλογή OPTIONALLY δηλώνει ότι κάποιες στήλες μπορεί να μην περικλείονται από τον χαρακτήρα που δηλώνεται. Η default τιμή είναι το '' (δύο φορές single quotation mark) που υποδηλώνει κενό χαρακτήρα.

ESCAPED BY: Δηλώνει τον χαρακτήρα που χρησιμοποιείται ως ειδικός χαρακτήρας (escape) στο αρχείο. Η default τιμή είναι ο χαρακτήρας backslash που δηλώνεται ως: '\\'.

LINES: Με το STARTING BY ορίζουμε το αλφαριθμητικό το οποίο δηλώνει εκκίνηση γραμμής στο αρχείο. Με το TERMINATED BY ορίζουμε το αλφαριθμητικό με το οποίο τερματίζεται κάθε γραμμή. Η default τιμή για την αρχή γραμμής είναι το '' (δύο φορές single quotation mark) και για το τέλος το '\n'.

Παράδειγμα χρήσης: Έστω ότι έχω σε MS Windows ένα CSV (Comma Separated Values) αρχείο Customer.txt στο path C:\Users\luke\Desktop\kallipos\ με τις ακόλουθες εγγραφές:

1,"077783234","56 Baltetsiou st.","Kostas","Kostantinou","1990-10-30"
2,"175783239","107 Diakou st.","Eleni","Kostantinou","1985-11-02"
3,"095111139","12 Rodon st.","Maria","Papantoniou","1967-03-20"
4,"175744444","30 Papanikoli st.","Eleni","Ioannou","1990-03-02"

και επιθυμούμε να φορτώσουμε τις εγγραφές στον πίνακα Customer υποθέτοντας ότι ο πίνακας δεν έχει καμία εγγραφή.

Σημείωση: Μπορείτε να σβήσετε τις εγγραφές του Customer με την εντολή:
DELETE FROM Customer; Περισσότερα για την εντολή DELETE σε επόμενο κεφάλαιο.

Η φόρτωση του αρχείου μπορεί να γίνει με την εντολή:

```
LOAD DATA LOCAL INFILE 'C:\\Users\\luke\\Desktop\\kallipos\\Customer.txt'
INTO TABLE Customer
COLUMNS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\r\n';
```

Σημείωση: Σε Linux κάθε γραμμή ενός text αρχείου τελειώνει με '\n'. Στα MS Windows τελειώνει με '\r\n' εκτός και αν το αρχείο έχει δημιουργηθεί με το Notepad.

Αν η πρώτη εγγραφή του αρχείου ήταν η:
1,"077783234","56, Baltetsiou st.","Kostas","Kostantinou","1990-10-30"
η παραπάνω εντολή θα το φόρτωνε κανονικά δηλ. το κόμμα στη διεύθυνση δε θα δημιουργούσε πρόβλημα. Αντίθετα αν παραλείπονταν τα διπλά quotes η φόρτωση δε θα ήταν σωστή καθώς θα μεταφράζονταν το: 56, Baltetsiou st. σαν τιμές σε δύο πεδία. Για να μπορέσω να φορτώσω την εγγραφή:
1,077783234,56, Baltetsiou st.,Kostas,Kostantinou,1990-10-30
θα πρέπει το ',' στη διεύθυνση να δηλωθεί ότι είναι μέρος της διεύθυνσης και όχι τέλος πεδίου. Αυτό γίνεται βάζοντας τον χαρακτήρα διαφυγής (escape) μπροστά:
1,077783234,56\, Baltetsiou st.,Kostas,Kostantinou,1990-10-30
Αν ο escape χαρακτήρας που χρησιμοποιήθηκε στο αρχείο ήταν άλλος πχ. το '#' τότε η εντολή φόρτωσης θα έπρεπε να είναι η:

```
LOAD DATA LOCAL INFILE 'C:\\Users\\luke\\Desktop\\kallipos\\Customer.txt'
INTO TABLE Customer
COLUMNS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
ESCAPED BY '#'
LINES TERMINATED BY '\r\n';
```

Αντίστοιχη μέριμνα πρέπει να ληφθεί όταν σε πεδίο που περικλείεται από " θέλουμε να περιλάβουμε το χαρακτήρα: ". Τέλος αν στην τιμή ενός πεδίου στο αρχείο εγγραφών θέλουμε να περιλάβουμε τον escape χαρακτήρα, πρέπει να αναγράφεται ο χαρακτήρας δύο φορές πχ. η εγγραφή:
1,077783234,56#, ##Baltetsiou st.##,Kostas,Kostantinou,1990-10-30
θα φορτωθεί σαν:

cid	*afm*	*address*	*name*	*sname*	*dateOfBirth*
1	077783234	56, #Baltetsiou st.#	Kostas	Kostantinou	1990-10-30

4.1.4 Εντολές επισκόπησης πινάκων

Σε αυτήν την κατηγορία περιλαμβάνονται εντολές που ξεκινούν με τη λέξη SHOW. Για την περιγραφή των εντολών θα χρησιμοποιήσουμε ως παράδειγμα τον πίνακα Customer.

SHOW TABLES;
Εμφανίζει τους πίνακες στη ΒΔ.

SHOW TABLE STATUS;
Εμφανίζει τους πίνακες της ΒΔ μαζί με επιπλέον πληροφορίες κατάστασης για κάθε πίνακα που αφορούν πχ. το πλήθος των εγγραφών, το μέγεθος των εγγραφών, χρόνο δημιουργίας, τροποποίησης κλπ.

SHOW TABLE STATUS LIKE 'Customer';
Ίδια όπως η SHOW TABLE STATUS με τη διαφορά ότι επιστρέφει τις πληροφορίες κατάστασης μόνο για τον πίνακα Customer.

SHOW CREATE TABLE Customer;
Εμφανίζει τον τρόπο με τον οποίο έχει δημιουργηθεί ο πίνακας Customer. Αν ο πίνακας έχει φτιαχτεί όπως περιγράφηκε στο Κεφ. 3 τότε το αποτέλεσμα της εντολής θα μοιάζει με:

```
| customer | CREATE TABLE `customer` (
  `cid` int(11) NOT NULL AUTO_INCREMENT,
  `afm` char(10) DEFAULT NULL,
  `address` varchar(50) DEFAULT 'Unknown',
  `name` char(20) NOT NULL DEFAULT 'Unknown',
  `sname` char(20) NOT NULL DEFAULT 'Unknown',
  `dateOfBirth` date DEFAULT NULL,
  PRIMARY KEY (`cid`),
  UNIQUE KEY `afm` (`afm`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 |
```

Παρατηρήστε ότι στην τελευταία γραμμή της CREATE TABLE υπάρχουν 3 ορίσματα που δεν είχαν περιγραφεί στο Κεφ. 3:

ENGINE: δηλώνει τον τρόπο με τον οποίο αποθηκεύεται ο πίνακας. Το default engine που χρησιμοποιείται είναι το InnoDB. Άλλο διαδεδομένο είναι το MyISAM. Περισσότερα για τους μηχανισμούς αποθήκευσης σε επόμενο κεφάλαιο.

AUTO_INCREMENT: η τρέχουσα τιμή του.

DEFAULT CHARSET: το σετ χαρακτήρων που χρησιμοποιείται. Το default είναι το UTF8.

Είναι προφανές ότι στην αντίστοιχη CREATE TABLE δήλωση μπορούμε να επιλέξουμε άλλες τιμές στις παραπάνω 3 επιλογές. Για παράδειγμα για να ορίσουμε τον πίνακα Customer με τιμή εκκίνησης του AUTO_INCREMENT το 1000 και μηχανισμό αποθήκευσης το MyISAM θα δίναμε:

```
CREATE TABLE Customer (
  …
) ENGINE=MyISAM AUTO_INCREMENT=5;
```

SHOW [FULL] COLUMNS FROM Customer;
Εμφανίζει τα πεδία του Customer και πληροφορίες δημιουργίας για αυτά (τύπος, αν επιτρέπει NULL, αν είναι PRIMARY ή UNIQUE KEY, τη default τιμή, αν είναι AUTO_INCREMENT). Με την επιλογή FULL εμφανίζονται επιπλέον πληροφορίες για τα πεδία πχ. δικαιώματα κλπ.

SHOW [FULL] COLUMNS FROM Customer LIKE 'address';
Ίδια με την προηγούμενη εντολή μόνο που θα επιστραφούν στοιχεία μόνο για το πεδίο address.

EXPLAIN Customer;
Ίδια με τη SHOW COLUMNS FROM Customer. Αντί για τη λέξη EXPLAIN μπορεί να χρησιμοποιηθούν ισοδύναμα τα DESCRIBE και DESC. Να σημειωθεί ότι η εντολή EXPLAIN μπορεί να χρησιμοποιηθεί για να πάρουμε πληροφορίες σχετικά με κάποιο ερώτημα πχ. EXPLAIN SELECT * FROM Customer; Η συγκεκριμένη χρήση δε θα μας απασχολήσει στα πλαίσια του εργαστηρίου. Για περισσότερες πληροφορίες σχετικά με τη σύνταξη και χρήση της εντολής ο αναγνώστης μπορεί να ανατρέξει στο Κεφ. 13.8.2 του εγχειριδίου της MySQL5.7.

{EXPLAIN | DESCRIBE | DESC} Customer address;

Ίδια με την παραπάνω μόνο που επιστρέφει πληροφορίες μόνο για το πεδίο address.

4.1.5 Η εντολή ALTER TABLE

Έχοντας ορίσει έναν πίνακα μπορούμε να αλλάξουμε τα χαρακτηριστικά του με την εντολή ALTER TABLE. Η βασική σύνταξη και λειτουργικότητα της εντολής είναι το αντικείμενο αυτής της υποενότητας. Για περισσότερες πληροφορίες ο αναγνώστης μπορεί να ανατρέξει στο Κεφ. 13.1.6 του εγχειριδίου της MySQL5.7.

ALTER TABLE *όνομα_πίνακα είδος_αλλαγής*;

Ως προς το *είδος_αλλαγής* ξεχωρίζουμε τις εξής τρεις:
ADD: Για να προσθέσουμε στον πίνακα. Κύριες επιτρεπτές προσθέσεις: πεδίο, PRIMARY, UNIQUE ή FOREIGN KEY.
DROP: Για να αφαιρέσουμε από τον πίνακα. Κύριες επιτρεπτές αφαιρέσεις: πεδίο, PRIMARY, UNIQUE ή FOREIGN KEY.
CHANGE: Για να αλλάξουμε τον ορισμό πεδίου.

Παραδείγματα χρήσης

- ALTER TABLE Customer DROP name; Προαιρετικά θα μπορούσε να γραφεί DROP COLUMN name. Διαγράφει το πεδίο name από τον πίνακα εφόσον δεν παραβιάζεται ξένο κλειδί.
- ALTER TABLE Phones DROP FOREIGN KEY phones_ibfk_1; Για να διαγράψουμε περιορισμό foreign key, θα πρέπει να διαγραφεί με το όνομα του περιορισμού. Δίνοντας SHOW CREATE TABLE Phones; μπορούμε να δούμε ότι ο περιορισμός ξένου κλειδιού για το cid έχει το όνομα phones_ibfk_1. Γενικά όταν ορίζουμε περιορισμούς ξένου κλειδιού σε πίνακα φτιαγμένο στο ENGINE=InnoDB που είναι το default, η ονομασία των περιορισμών είναι της μορφής: όνομαΠίνακα_ibfk_αύξονΑριθμόςΠεριορισμού. Το ibfk είναι συντομογραφία για το InnoDB foreign key. Έτσι αν είχα δύο FOREIGN KEY στον πίνακα Phones, τα ονόματα των περιορισμών θα ήταν phones_ibfk_1 και phones_ibfk_2.
- ALTER TABLE Phones DROP PRIMARY KEY; Διαγράφει το πρωτεύον κλειδί στον πίνακα Phones. Το πεδίο που ήταν πρωτεύον κλειδί (cid) εξακολουθεί να υπάρχει απλά δεν ορίζεται πλέον σαν πρωτεύον.
- ALTER TABLE Customer DROP KEY afm; Διαγράφει τον περιορισμό με όνομα afm. Στη δημιουργία του πίνακα Customer θέσαμε το afm να είναι UNIQUE KEY. Ο αντίστοιχος περιορισμός είναι το ίδιο το όνομα του πεδίου (μπορούμε να το δούμε δίνοντας SHOW CREATE TABLE).
- ALTER TABLE Customer ADD
  COLUMN name CHAR(20) NOT NULL DEFAULT 'Unknown';
  Υποθέτοντας ότι το name είχε σβηστεί από τον πίνακα Customer, το ξαναδημιουργούμε δίνοντας τη δήλωση του πεδίου όπως θα κάναμε στην CREATE TABLE. Η λέξη COLUMN στην παραπάνω εντολή είναι προαιρετική. Το πεδίο δημιουργείται και η σειρά που καταλαμβάνει είναι η τελευταία. Αν θέλαμε να είναι πρώτο στον πίνακα θα μπορούσαμε να το είχαμε δημιουργήσει με την εντολή:
  ALTER TABLE Customer ADD
  COLUMN name CHAR(20) NOT NULL DEFAULT 'Unknown' FIRST;
  ενώ αν θέλαμε να καταλάμβανε την ίδια σειρά με αυτήν που είχε στην CREATE TABLE της προηγούμενης υποενότητας θα δίναμε:
  ALTER TABLE Customer ADD
  COLUMN name CHAR(20) NOT NULL DEFAULT 'Unknown' AFTER address;
- ALTER TABLE Phones ADD PRIMARY KEY (cid); Δηλώνει στον πίνακα Phones ότι PRIMARY KEY είναι το cid. Το/τα πεδία στη δήλωση του PRIMARY KEY πρέπει να υπάρχουν στον πίνακα και να μην υπάρχει δηλωμένο άλλο PRIMARY KEY.

- ALTER TABLE Customer ADD UNIQUE KEY (afm); Δηλώνει ως UNIQUE KEY το afm στον πίνακα Customer. Το/τα πεδία που δηλώνονται ως UNIQUE KEY πρέπει να υπάρχουν στον πίνακα. Αν είναι ήδη δηλωμένο το afm ως UNIQUE KEY τότε δημιουργείται δεύτερος περιορισμός με όνομα afm_2 και δίνεται warning. Σε μελλοντικές εκδόσεις οι ταυτόσημες επαναδηλώσεις αναμένεται να απαγορευτούν.
- ALTER TABLE Phones ADD FOREIGN KEY (cid) REFERENCES Customer (cid)
  ON UPDATE CASCADE ON DELETE RESTRICT;
  Δηλώνει FOREIGN KEY στον πίνακα phones ακολουθώντας το συντακτικό που παρουσιάστηκε στο Κεφ. 3. Αν είναι ήδη δηλωμένο το cid ως FOREIGN KEY θα δούμε να δημιουργείται δεύτερος ίδιος περιορισμός χωρίς warning.
- ALTER TABLE Customer CHANGE sname surname CHAR(30) NOT NULL DEFAULT 'Unknown';
  Η σύνταξη του CHANGE έχει ως εξής: CHANGE *υπάρχον_πεδίο νέος_ορισμός_πεδίου*.
  Στο *νέο_ορισμό_πεδίου* ακολουθούμε το συντακτικό που παρουσιάστηκε στο Κεφ. 3 για τον ορισμό πεδίων στην CREATE TABLE. Η εντολή του παραδείγματος αλλάζει το πεδίο sname του Customer σε surname, αλλάζοντας ταυτόχρονα και τον τύπο του από CHAR(20) σε CHAR(30).
  Προσοχή: Αν θέλαμε απλά να αλλάξουμε το όνομα του πεδίου sname και όχι τον τύπο του θα ήταν λάθος να γράφαμε: ALTER TABLE Customer CHANGE sname surname; Η CHANGE περιμένει νέα δήλωση πεδίου όχι μόνο το νέο όνομα. Αν θέλουμε να αλλάξουμε τον τύπο χωρίς να αλλάξουμε το όνομα μπορούμε χρησιμοποιώντας το ίδιο όνομα με το υπάρχον στο *νέο_ορισμό_πεδίου*. Για παράδειγμα:
  ALTER TABLE Customer CHANGE sname sname CHAR(30) NOT NULL DEFAULT 'Unknown';

Ακολουθούν κάποια επιπλέον *είδη_αλλαγής* πέραν των ADD, DROP, CHANGE.
Επιλογές Πίνακα: Αναθέτουμε νέα τιμή σε κάποια από τις επιλογές πίνακα όπως ENGINE, AUTO_COMMIT κλπ. Για παράδειγμα η ALTER TABLE Phones ENGINE=MyISAM; αλλάζει τον τρόπο αποθήκευσης του πίνακα Phones, ενώ ή ALTER TABLE Customer AUTO_COMMIT=1000; αλλάζει την τιμή του AUTO_COMMIT στον πίνακα Customer. Για περισσότερες επιλογές πίνακα ο αναγνώστης μπορεί να δει το κομμάτι *table_option* στη δήλωση της CREATE TABLE (Κεφ. 13.1.14 του εγχειριδίου).
RENAME: Μπορούμε να μετονομάσουμε έναν πίνακα, πχ.:
ALTER TABLE Customer RENAME OurCustomers;
ή έναν περιορισμό UNIQUE KEY πχ.:
ALTER TABLE Customer RENAME afm TO theAfm;

Για τη μετονομασία πίνακα υπάρχει και η ξεχωριστή (έξω από την ALTER TABLE) εντολή RENAME TABLE (Κεφ. 13.1.28 του εγχειριδίου). Για παράδειγμα για να αλλάξω το όνομα του Customer σε OurCustomers θα μπορούσα εναλλακτικά να δώσω την εντολή:
RENAME TABLE Customer TO OurCustomers;

Στην εντολή RENAME TABLE μπορώ να έχω πολλές μετονομασίες πινάκων κάθε μια χωρισμένη με κόμμα πχ.: RENAME TABLE Customer TO OurCustomers, Phones TO OurPhones;

4.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 3 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 1. Δείτε τους διαθέσιμους πίνακες με χρήση της SHOW TABLES. 2. Δείτε την δομή του κάθε πίνακα με χρήση της DESC. 3. Δείτε την εντολή CREATE που δημιουργεί τον κάθε πίνακα με χρήση της SHOW CREATE TABLE. 4. Καταχωρήστε 5 φοιτητές, 5 μαθήματα, 5 καθηγητές, 5 βαθμολογίες, 5 τηλέφωνα και ένα προαπαιτούμενο μάθημα. 5. Με χρήση την εντολής LOAD DATA INFILE φορτώστε δεδομένα στο σχήμα σας χρησιμοποιώντας αρχεία που φέρουν το όνομα του πίνακα και την επέκταση txt. 6. Δημιουργήστε έναν ακόμη πίνακα Tesis στον οποίο θα αποθηκεύονται οι διπλωματικές εργασίες κάθε φοιτητή. Ο πίνακας θα έχει κωδικό εργασίας (SID), τίτλο, κωδικό φοιτητή και επιβλέποντα καθηγητή. 7. Μετονομάστε τον πίνακα Tesis σε Thesis. 8. Προσθέστε μία επιπλέον στήλη με την ημερομηνία έναρξης της διπλωματικής εργασίας με χρήση της ALTER TABLE. 9. Προσθέστε στον πίνακα Student τα ακόλουθα πεδία: email, όνομα και κωδικό πρόσβασης στα υπολογιστικά συστήματα, καθώς και το οικογενειακό εισόδημα (πιθανώς για να υπολογιστεί το στεγαστικό επίδομα). 10. Αλλάξτε τον τύπο της ηλικίας από ακέραιο σε αλφαριθμητικό. 11. Αλλάξτε τον τύπο του κωδικού σε char(10). 12. Διαγράψτε την στήλη των κωδικών. 13. Εισάγετε πάλι τη στήλη των κωδικών με τύπο varchar(20). 14. Εξάγετε το σχήμα university στο αρχείο University.sql με χρήση του προγράμματος MYSQLDUMP. 15. Εισαγωγή ενός σχήματος πχ. University.sql με χρήση του προγράμματος MYSQL (MySQL Workbench).
Ζητούμενα:	Εκτέλεση των κατάλληλων εντολών SQL για την επίτευξη των διαφόρων λειτουργιών

Το σχεσιακό σχήμα της εργαστηριακής άσκησης 3 (Κεφ. 3.2) είναι το ακόλουθο:

- Student (AM, Firstname, Lastname, YearOfStudy)
- Course (Cid, Title, KK)
- Professor (KK, Firstname, Lastname)
- Exam (AM, Cid, Date, Grade).
- Requires (Cid1, Cid2)
- Telephone (AM, TelphoneNumber)

Στη συνέχεια δίνεται ο κώδικας SQL δημιουργίας της βάσης δεδομένων και των πινάκων.

```
CREATE SCHEMA `university`;
USE `university`;
-- -----------------------------------------------------
-- Table `university`.`Student`
-- -----------------------------------------------------
CREATE TABLE `university`.`Student` (
```

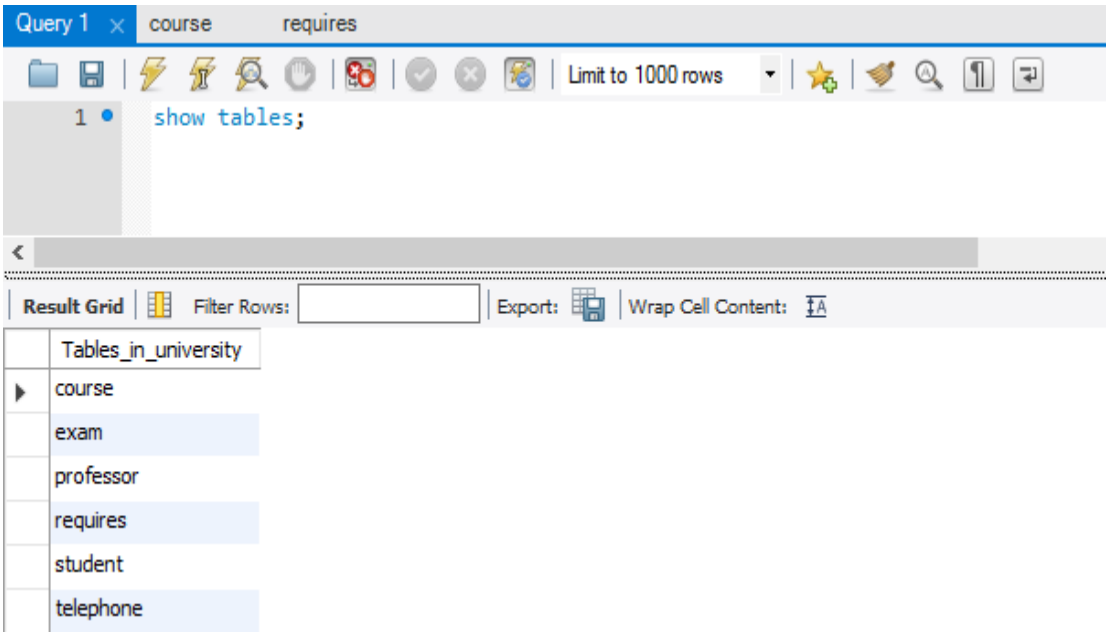
```
  `AM` INT NOT NULL AUTO_INCREMENT,
  `Firstname` VARCHAR(45) NOT NULL,
  `Lastname` VARCHAR(45) NOT NULL,
  `YearOfStudy` INT NOT NULL,
  PRIMARY KEY (`AM`)
);


-- -----------------------------------------------------
-- Table `university`.`Professor`
-- -----------------------------------------------------
CREATE TABLE `university`.`Professor` (
  `KK` INT NOT NULL AUTO_INCREMENT,
  `Firstname` VARCHAR(45) NOT NULL,
  `Lastname` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`KK`)
);


-- -----------------------------------------------------
-- Table `university`.`Course`
-- -----------------------------------------------------
CREATE TABLE `university`.`Course` (
  `CID` INT NOT NULL AUTO_INCREMENT,
  `Title` VARCHAR(45) NOT NULL,
  `KK` INT NOT NULL,
  PRIMARY KEY (`CID`),
  CONSTRAINT `fk_Course_Professor`
    FOREIGN KEY (`KK`)
    REFERENCES `university`.`Professor` (`KK`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);


-- -----------------------------------------------------
-- Table `university`.`Exam`
-- -----------------------------------------------------
CREATE TABLE `university`.`Exam` (
  `AM` INT NOT NULL,
  `Cid` INT NOT NULL,
  `Date` DATE NOT NULL,
  `Grade` FLOAT NOT NULL,
  PRIMARY KEY (`AM`, `Cid`, `Date`),
  CONSTRAINT `fk_Student_Exam`
    FOREIGN KEY (`AM`)
    REFERENCES `university`.`Student` (`AM`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Exam_Course`
    FOREIGN KEY (`Cid`)
    REFERENCES `university`.`Course` (`CID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);

-- -----------------------------------------------------
-- Table `university`.`Telephone`
-- -----------------------------------------------------
CREATE TABLE `university`.`Telephone` (
  `AM` INT NOT NULL,
  `TelephoneNumber` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`TelephoneNumber`, `AM`),
  CONSTRAINT `fk_Telephone_Student`
    FOREIGN KEY (`AM`)
    REFERENCES `university`.`Student` (`AM`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION
);


-- -----------------------------------------------------
-- Table `university`.`Requires`
-- -----------------------------------------------------
CREATE TABLE `university`.`Requires` (
  `Cid1` INT NOT NULL,
  `Cid2` INT NOT NULL,
  PRIMARY KEY (`Cid1`, `Cid2`),
  CONSTRAINT `fk_CourseA_Requires`
    FOREIGN KEY (`Cid1`)
    REFERENCES `university`.`Course` (`CID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Requires_CourseB`
    FOREIGN KEY (`Cid2`)
    REFERENCES `university`.`Course` (`CID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
```

Ζητούμενο 1: Δείτε τους διαθέσιμους πίνακες με χρήση της SHOW TABLES.

Δίνουμε:
SHOW TABLES;

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 4.1.

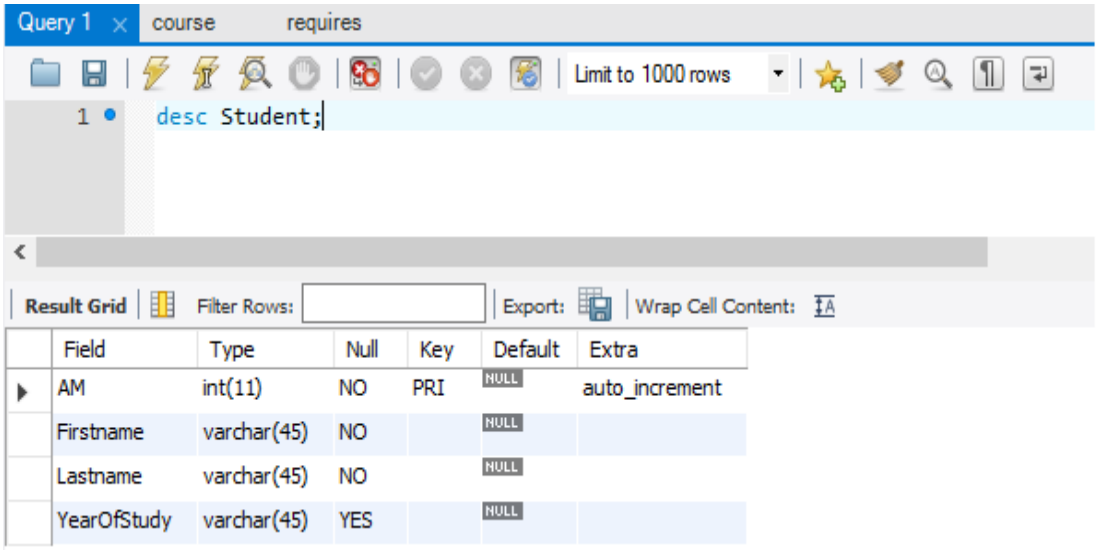


Εικόνα 4.1: *Η εντολή SHOW TABLES.*

Ζητούμενο 2: Δείτε την δομή του κάθε πίνακα με χρήση της DESC.

Δίνουμε:
DESC Student;
DESC Professor;
DESC Course;
DESC Telephone;
DESC Exam;
DESC Requires;

Παράδειγμα αποτελέσματος για τον πίνακα Student φαίνεται στην Εικόνα 4.2.



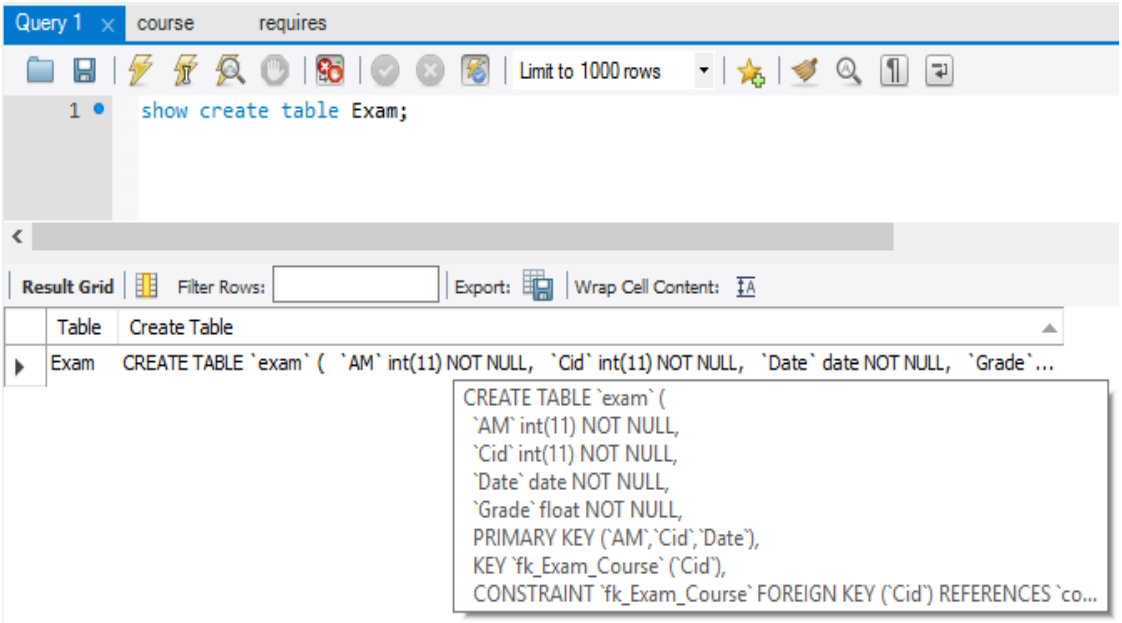
Εικόνα 4.2: *Η εντολή DESC.*

Ζητούμενο 3: Δείτε την εντολή που δημιουργεί τον κάθε πίνακα με χρήση της SHOW CREATE TABLE.

Δίνουμε:

SHOW CREATE TABLE Student;
SHOW CREATE TABLE Professor;
SHOW CREATE TABLE Course;
SHOW CREATE TABLE Telephone;
SHOW CREATE TABLE Exam;
SHOW CREATE TABLE Requires;

Παράδειγμα αποτελέσματος για τον πίνακα Exam φαίνεται στην Εικόνα 4.3.



Εικόνα 4.3: *Η εντολή SHOW CREATE TABLE.*

Ζητούμενο 4: Καταχωρήστε 5 φοιτητές, 5 μαθήματα, 5 καθηγητές, 5 βαθμολογίες, 5 τηλέφωνα και ένα προαπαιτούμενο μάθημα.

INSERT INTO Student(Firstname, Lastname, YearOfStudy)
VALUES (‘Alexandros’, ‘Theocharis’, ‘3’);
INSERT INTO Student(Firstname, Lastname, YearOfStudy)
VALUES (‘Maria’, ‘Karagianni’, ‘2’);
INSERT INTO Student(Firstname, Lastname, YearOfStudy)
VALUES (‘Eleni’, ‘Karaxaliou’, ‘4’);
INSERT INTO Student(Firstname, Lastname, YearOfStudy)
VALUES (‘Kosmas’, ‘Pitas’, ‘4’);
INSERT INTO Student(Firstname, Lastname, YearOfStudy)
VALUES (‘Nikos’, ‘Kouris’, ‘2’);

INSERT INTO Course(Title, KK)
VALUES (‘Baseis Dedomenwn’, 1);
INSERT INTO Course(Title, KK)
VALUES (‘Domes Dedomenwn’, 2);
INSERT INTO Course(Title, KK)
VALUES (‘Programmatismos I’, 3);
INSERT INTO Course(Title, KK)
VALUES (‘Programmatismos II’, 4);
INSERT INTO Course(Title, KK)
VALUES (‘Texnologies Diadiktuou’, 5);

INSERT INTO Professor(Firstname, Lastname)
VALUES (‘Panos’, ‘Tzimas’);
INSERT INTO Professor(Firstname, Lastname)
VALUES (‘Eleni’, ‘Andriopoulou’);
INSERT INTO Professor(Firstname, Lastname)

```
VALUES (‘Nikos’, ‘Theocharopoulos’);
INSERT INTO Professor(Firstname, Lastname)
VALUES (‘Loukas’, ‘Mpakos’);
INSERT INTO Professor(Firstname, Lastname)
VALUES (‘Eleni’, ‘Anagnwstopoulou’);


INSERT INTO Exam(AM, Cid, Date, Grade)
VALUES (1, 3, ‘2015-10-5’, 6);
INSERT INTO Exam(AM, Cid, Date, Grade)
VALUES (2, 4, ‘2015-10-3’, 8);
INSERT INTO Exam(AM, Cid, Date, Grade)
VALUES (3, 5, ‘2014-3-3’, 3);
INSERT INTO Exam(AM, Cid, Date, Grade)
VALUES (4, 4, ‘2014-3-3’, 5);
INSERT INTO Exam(AM, Cid, Date, Grade)
VALUES (5, 4, ‘2015-10-3’, 2);

INSERT INTO Telephone(AM, TelephoneNumber)
VALUES (1, ‘6977009988’);
INSERT INTO Telephone(AM, TelephoneNumber))
VALUES (2, ‘2763103344’);
INSERT INTO Telephone(AM, TelephoneNumber))
VALUES (3, ‘6977777777’);
INSERT INTO Telephone(AM, TelephoneNumber))
VALUES (4, ‘6980000099’);
INSERT INTO Telephone(AM, TelephoneNumber))
VALUES (5, ‘2693032400’);

INSERT INTO Requires(Cid1, Cid2)
VALUES (3, 4);
```

Θεωρούμε ότι το μάθημα Programmatismos II έχει προαπαιτούμενο το μάθημα με τίτλο Programmatismos I.

Ζητούμενο 5: Με χρήση την εντολής LOAD DATA INFILE φορτώστε δεδομένα στο σχήμα σας χρησιμοποιώντας αρχεία που φέρουν το όνομα του πίνακα και την επέκταση txt.

Ένας δεύτερος τρόπος όπως εξηγήθηκε για να εισάγουμε τιμές σε έναν πίνακα είναι χρησιμοποιώντας ένα αρχείο στο οποίο έχουμε γράψει τις τιμές για τα πεδία, που το φορτώνουμε στον πίνακα με την εντολή LOAD DATA INFILE.

Δίνουμε λοιπόν:

```
LOAD DATA INFILE ‘student.txt’ INTO TABLE Student;
LOAD DATA INFILE ‘course.txt’ INTO TABLE Course;
LOAD DATA INFILE ‘professor.txt’ INTO TABLE Professor;
LOAD DATA INFILE ‘exam.txt’ INTO TABLE Exam;
```

Ζητούμενο 6: Δημιουργήστε έναν ακόμη πίνακα Tesis στον οποίο θα αποθηκεύονται οι διπλωματικές εργασίες κάθε φοιτητή. Ο πίνακας θα έχει κωδικό εργασίας (SID), τίτλο, κωδικό φοιτητή και επιβλέποντα καθηγητή.

Στον πίνακα πρωτεύον κλειδί θα είναι το SID ενώ οι κωδικοί φοιτητή και καθηγητή θα είναι ξένα κλειδιά. Ακολουθεί η δημιουργία του πίνακα:

```
-- -----------------------------------------------------
-- Table `university`.`Tesis`
-- -----------------------------------------------------
CREATE TABLE `university`.`Tesis` (
  `SID` INT NOT NULL AUTO_INCREMENT,
  `Title` VARCHAR(45) NOT NULL,
  `AM` INT NOT NULL,
  `KK` INT NOT NULL,
  PRIMARY KEY (`SID`),
  CONSTRAINT `fk_Tesis_Student`
    FOREIGN KEY (`AM`)
    REFERENCES `university`.`Student` (`AM`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Tesis_Professor`
    FOREIGN KEY (`KK`)
    REFERENCES `university`.`Professor` (`KK`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
```

Ζητούμενο 7: Μετονομάστε τον πίνακα Tesis σε Thesis.

Δίνουμε:
RENAME TABLE Tesis TO Thesis;

Ζητούμενο 8: Προσθέστε μία επιπλέον στήλη με την ημερομηνία έναρξης της διπλωματικής εργασίας με χρήση της ALTER TABLE.

Δίνουμε:
ALTER TABLE Thesis ADD Arxiki_Date DATE;

Ζητούμενο 9: Προσθέστε στον πίνακα Student τα ακόλουθα πεδία: email, όνομα και κωδικό πρόσβασης στα υπολογιστικά συστήματα, καθώς και το οικογενειακό εισόδημα (πιθανώς για να υπολογιστεί το στεγαστικό επίδομα).

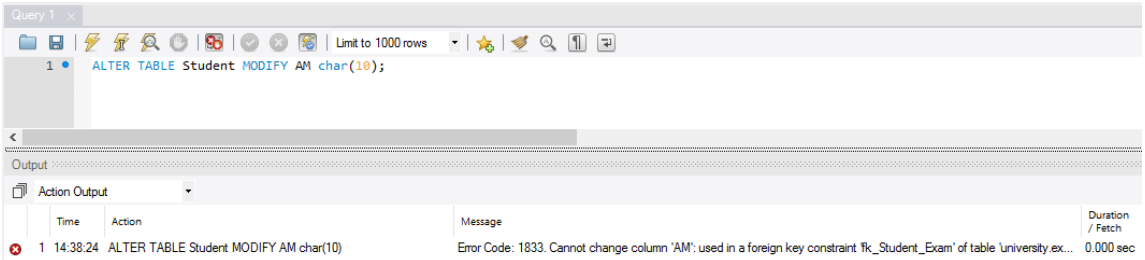
Δίνουμε:
ALTER TABLE Student ADD (email varchar(45), username_system varchar(45), insert_code varchar(20) not null, family_income int(20));

Ζητούμενο 10: Αλλάξτε τον τύπο της ηλικίας από ακέραιο σε αλφαριθμητικό.

Δίνουμε:
ALTER TABLE Student MODIFY YearOfStudy varchar(45);

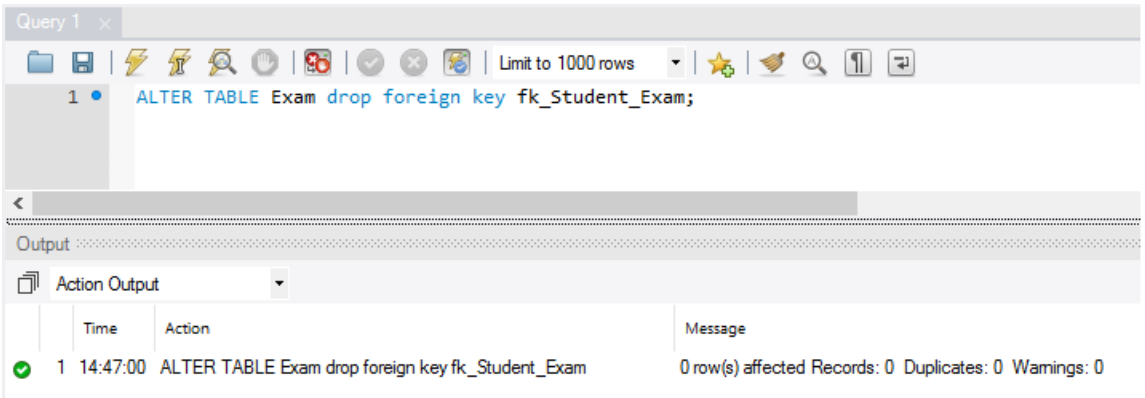
Ζητούμενο 11: Αλλάξτε τον τύπο του κωδικού σε char(10).

Δίνουμε:
ALTER TABLE Student MODIFY AM char(10);

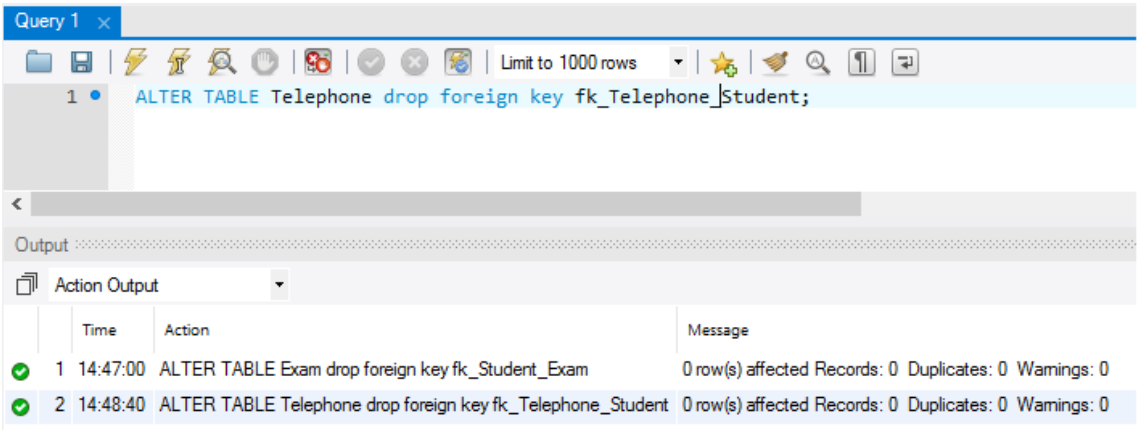


Εικόνα 4.4: *Τρέχοντας την εντολή εμφανίζεται λάθος, καθώς το ΑΜ είναι ξένο κλειδί σε άλλους πίνακες.*

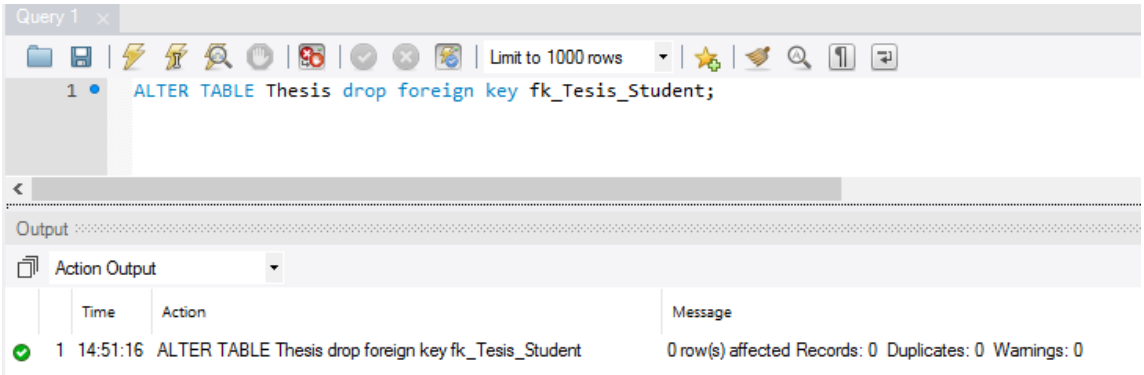
Τρέχοντας την εντολή επιστρέφει λάθος (Error Code: 1833), όπως παρατηρούμε στην Εικόνα 4.4. Αυτό συμβαίνει διότι το ΑΜ χρησιμοποιείται ως ξένο κλειδί στον πίνακα Exam, Telephone και Thesis. Για να τρέξουμε την εντολή πρέπει πρώτα να βγάλουμε το ΑΜ ως ξένο κλειδί (μαζί με τους περιορισμούς) από όλους τους πίνακες που συμμετέχει ως ξένο κλειδί, Εικόνες 4.5-4.7, να εκτελέσουμε την εντολή(Προσοχή: Την εντολή «ALTER TABLE Student MODIFY AM char(10);» την εκτελούμε για όλους τους πίνακες που εμφανίζεται το ΑΜ, όχι μόνο για τον Student, Εικόνες 4.8-4.11) και να ξαναβάλουμε το ΑΜ ως ξένο κλειδί στους πίνακες που υπήρχε, Εικόνες 4.12-4.14.



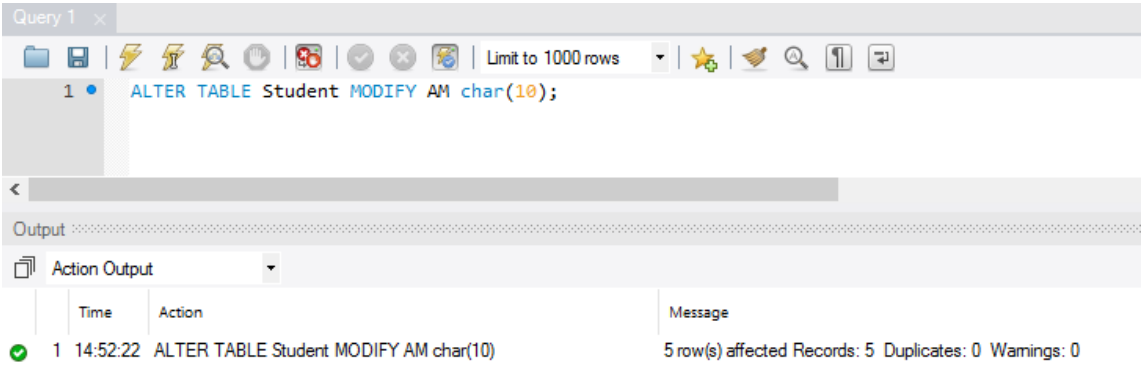
Εικόνα 4.5: *Αφαιρούμε το ξένο κλειδί ΑΜ από τον πίνακα Exam.*



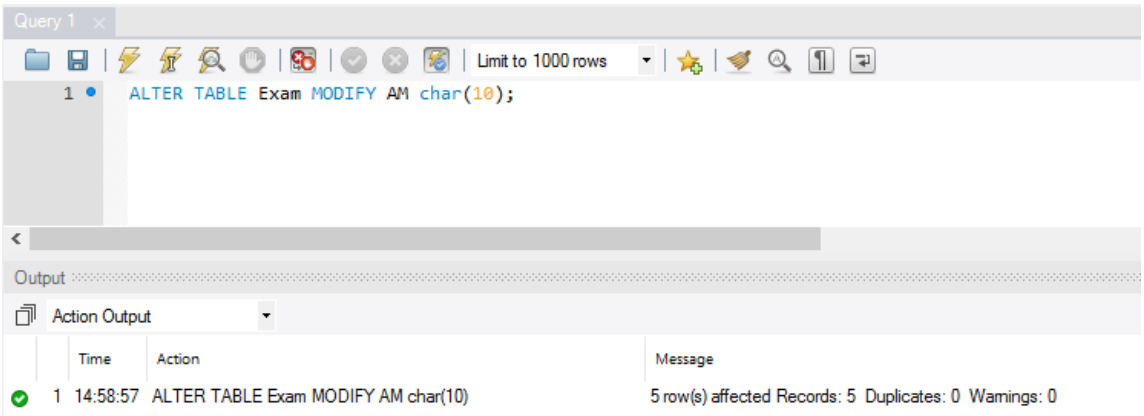
Εικόνα 4.6: *Αφαιρούμε το ξένο κλειδί ΑΜ από τον πίνακα Telephone.*



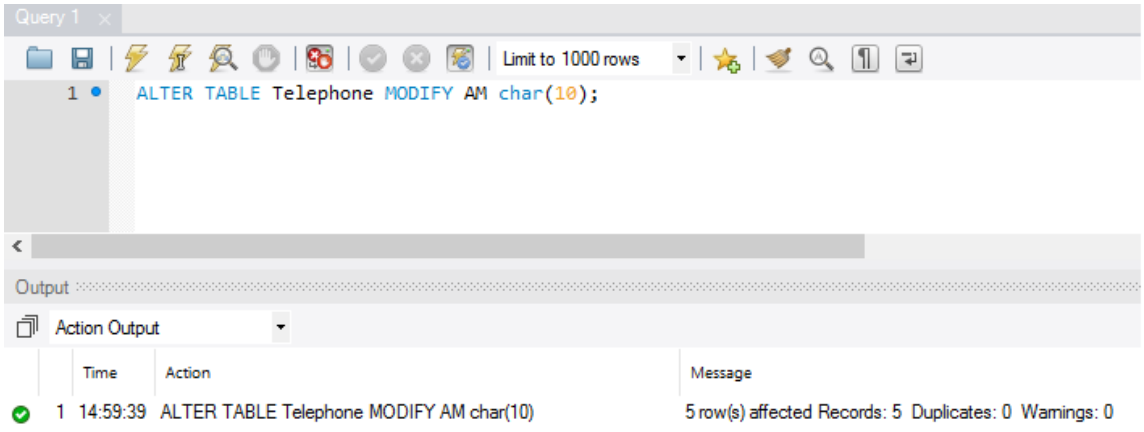
Εικόνα 4.7: *Αφαιρούμε το ξένο κλειδί AM από τον πίνακα Thesis.*



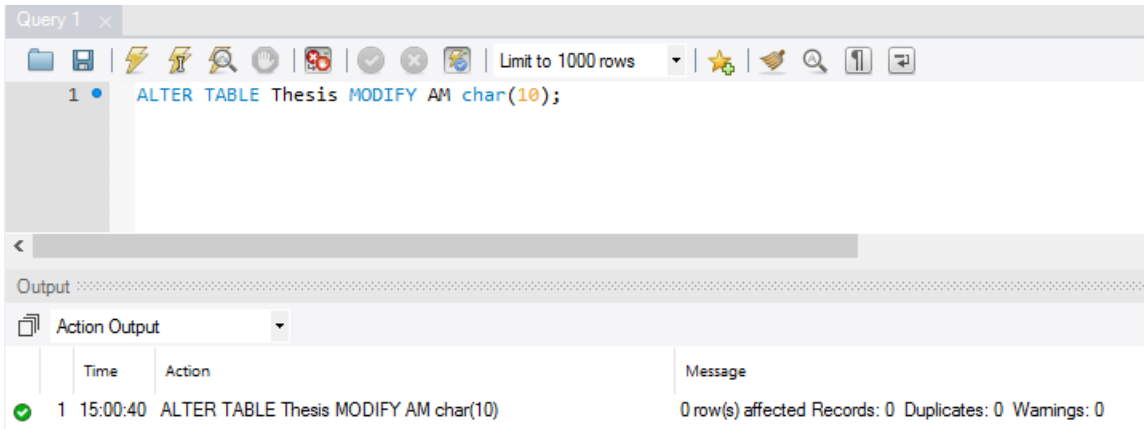
Εικόνα 4.8: *Τρέχουμε την εντολή για τον πίνακα Student.*



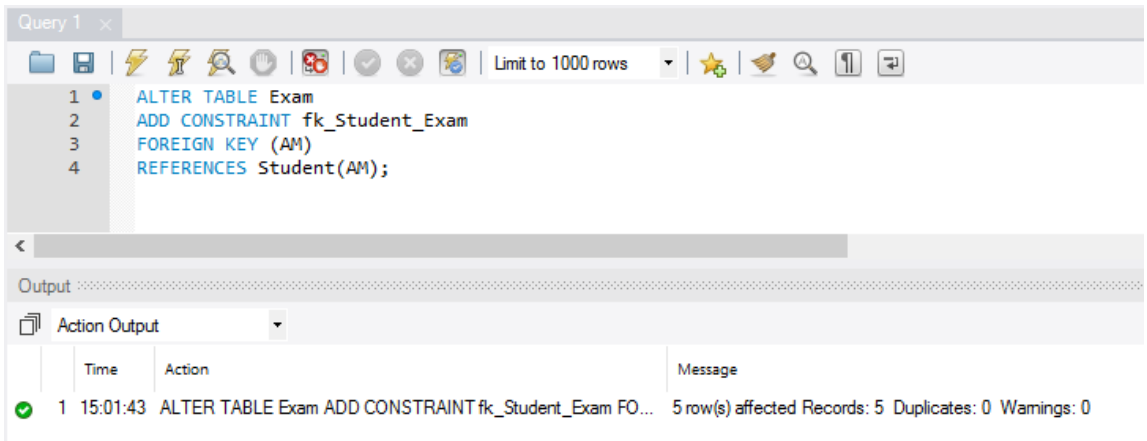
Εικόνα 4.9: *Τρέχουμε την εντολή για τον πίνακα Exam.*



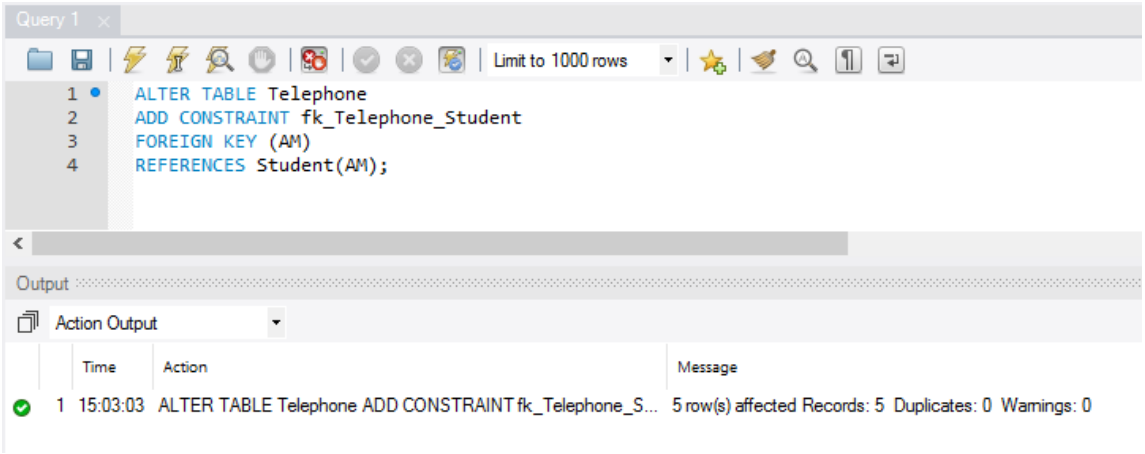
Εικόνα 4.10: *Τρέχουμε την εντολή για τον πίνακα Telephone.*



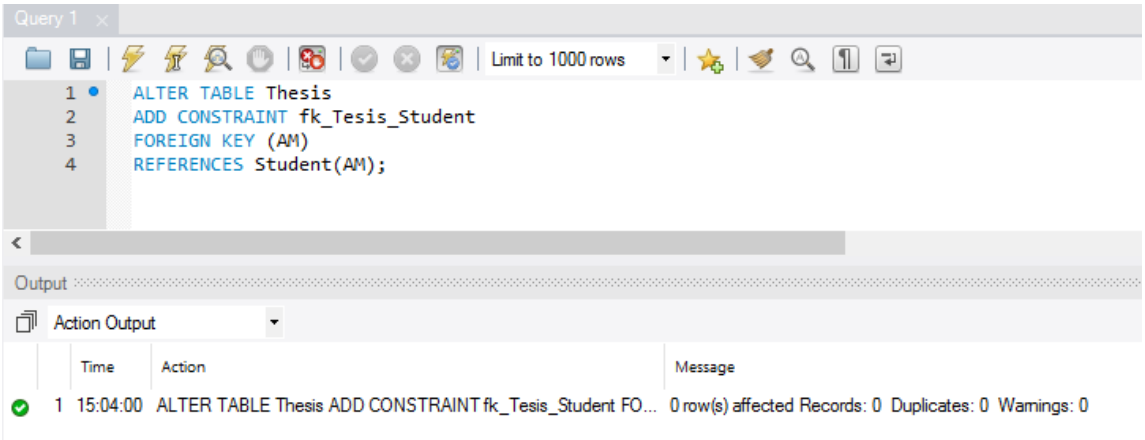
Εικόνα 4.11: *Τρέχουμε την εντολή για τον πίνακα Thesis.*



Εικόνα 4.12: *Επαναφορά του ξένου κλειδιού AM στον πίνακα Exam.*



Εικόνα 4.13: *Επαναφορά του ξένου κλειδιού ΑΜ στον πίνακα Telephone.*



Εικόνα 4.14: *Επαναφορά του ξένου κλειδιού ΑΜ στον πίνακα Thesis.*

Ζητούμενο 12: Διαγράψτε την στήλη των κωδικών.

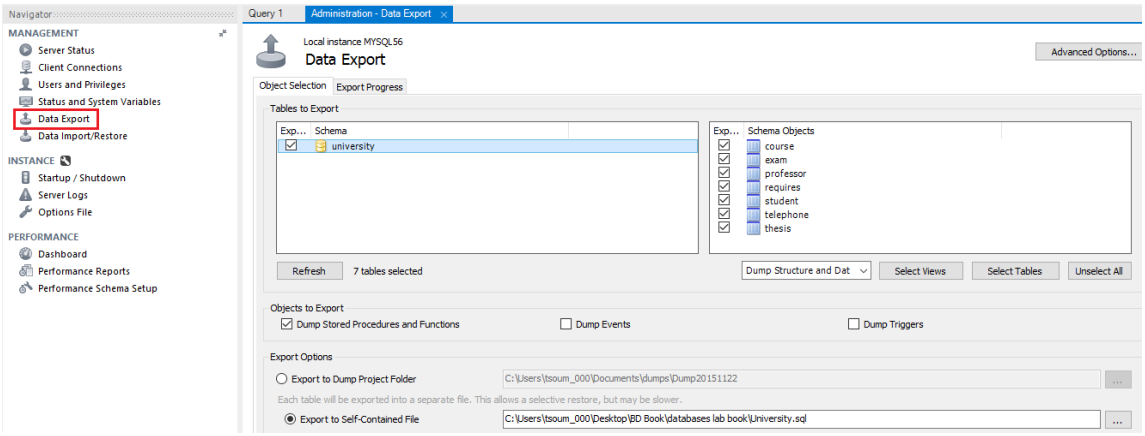
Δίνουμε:
ALTER TABLE Student DROP AM;

Ακολουθείτε την ίδια διαδικασία με το προηγούμενο ερώτημα.

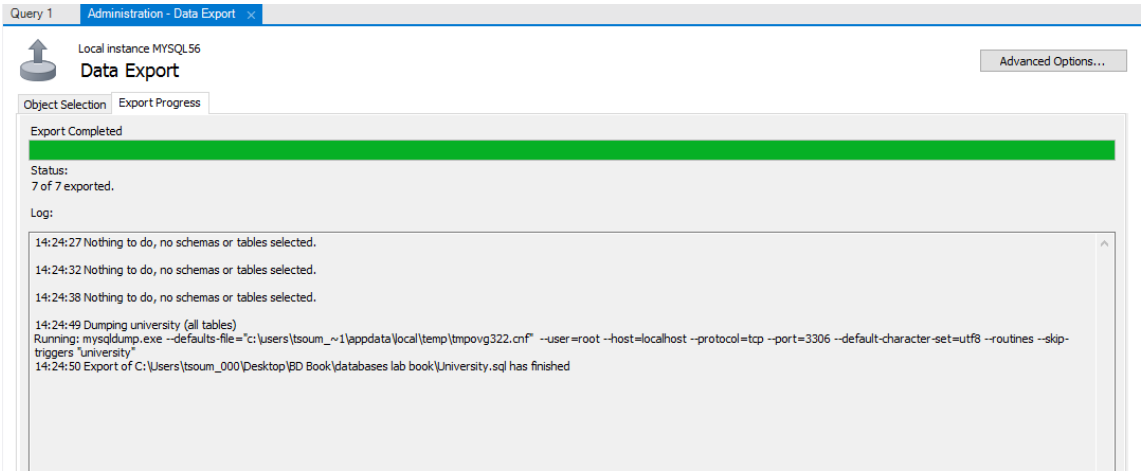
Ζητούμενο 13: Εισάγετε πάλι τη στήλη των κωδικών με τύπο varchar(20).

Δίνουμε:
ALTER TABLE Student ADD AM varchar(20);

Ζητούμενο 14: Εξάγετε το σχήμα university στο αρχείο University.sql με χρήση του προγράμματος MYSQLDUMP.

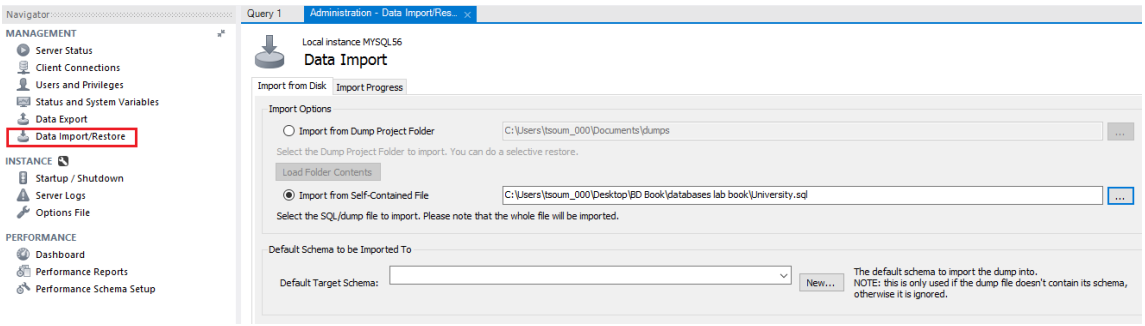


Εικόνα 4.15: *Εξαγωγή σχήματος university στο αρχείο University.sql.*



Εικόνα 4.16: *Επιτυχής εξαγωγή σχήματος.*

Ζητούμενο 15: Εισαγωγή ενός σχήματος πχ. University.sql με χρήση του προγράμματος MYSQL (MySQL Workbench).



Εικόνα 4.17: *Εισαγωγή σχήματος.*

4.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Δημιουργήστε μια Βάση Δεδομένων για μια επιχείρηση:

- Κάθε διευθυντής έχει Όνομα, Επώνυμο, Έτη εμπειρίας και email.
- Η εταιρεία έχει Επωνυμία, Έτος ίδρυσης και Διεύθυνση.
- Ο εργαζόμενος έχει Όνομα, Επώνυμο, Προϋπηρεσία και Ημερομηνία πρόσληψης.
- Μια εταιρεία μπορεί να έχει μόνο έναν διευθυντή και πολλούς εργαζομένους.
- Κάθε εργαζόμενος δουλεύει σε μία και μόνο εταιρεία.

Ζητούμενα:

1. Εξαγωγή των βασικών οντοτήτων του προβλήματος, των χαρακτηριστικών της κάθε οντότητας καθώς και των συσχετίσεων μεταξύ τους.
2. Σχεδιασμός του διαγράμματος Οντοτήτων-Συσχετίσεων (ER diagram).
3. Δημιουργήστε τους πίνακες του παραπάνω προβλήματος.
4. Εισάγετε από 5 εγγραφές σε κάθε πίνακα.
5. Προσθέστε στον πίνακα Εταιρεία το πεδίο Αριθμός υπαλλήλων.
6. Δημιουργήστε τον πίνακα Promitheutis, με χαρακτηριστικά email, Επώνυμο και Προϋπηρεσία.
7. Αλλάξτε το όνομα του πίνακα σε Promitheftis.

Άσκηση 2

Θέλουμε να φτιάξουμε μια βάση δεδομένων που να περιέχει όλες τις ταινίες που παίχτηκαν το 2015, τους σκηνοθέτες τους και τους θεατές που τις παρακολούθησαν. Κάθε ταινία θα έχει τίτλο, πρωταγωνιστή, σκηνοθέτη και αριθμό θεατών που την παρακολούθησε. Κάθε σκηνοθέτης θα έχει ονοματεπώνυμο, ηλικία και αριθμό ταινιών που έχει σκηνοθετήσει. Ο θεατής θα έχει username, βαθμολογία και αριθμό ταινιών που έχει παρακολουθήσει. Υποθέτουμε ότι κάθε ταινία έχει ένα σκηνοθέτη.

Ζητούμενα:

1. Δημιουργήστε το διάγραμμα ER της βάσης δεδομένων.
2. Δημιουργήστε τη βάση δεδομένων.
3. Δημιουργήστε τους πίνακες του προβλήματος.
4. Εισάγετε από 5 εγγραφές σε κάθε πίνακα.
5. Αλλάξτε τον τύπο δεδομένων για το χαρακτηριστικό username σε varchar(8).
6. Μετονομάστε τον πίνακα Theatis σε Viewer.
7. Δημιουργήστε τον πίνακα Krites, με χαρακτηριστικά kritisID, βαθμολογία και ηλικία.

Άσκηση 3

Δημιουργήστε μια βάση δεδομένων για ένα τουρνουά τένις. Για αυτή τη βάση δεδομένων θέλουμε να κρατάμε στοιχεία για τους παίκτες, το τουρνουά και τα αποτελέσματα. Για κάθε παίκτη θέλουμε το ονοματεπώνυμο, την ηλικία και έτος νίκης σε κάθε τουρνουά. Για κάθε τουρνουά θέλουμε την επωνυμία (Roland Garros, US Open, Australian Open), τη χώρα που διεξάγεται, έτος ίδρυσης και έτος διεξαγωγής. Για τα αποτελέσματα θέλουμε να κρατάμε το νικητή του κάθε τουρνουά και πόσα σετ πήρε ο ηττημένος.

Ζητούμενα:

1. Δημιουργήστε το διάγραμμα ER της βάσης δεδομένων.
2. Δημιουργήστε τη βάση δεδομένων.
3. Δημιουργήστε τους πίνακες του προβλήματος.
4. Εισάγετε από 5 εγγραφές σε κάθε πίνακα.
5. Προσθέστε στον πίνακα τουρνουά το νικητή για το έτος 2015.
6. Αλλάξτε τον τύπο δεδομένων για το επώνυμο του παίκτη σε varchar(40).
7. Δημιουργήστε τον πίνακα Kritis, με χαρακτηριστικά ονοματεπώνυμο και εμπειρία. Κάθε κριτής μπορεί να συμμετέχει μόνο σε ένα τουρνουά το χρόνο.
8. Διαγράψτε από τον πίνακα Kritis τη στήλη εμπειρία.
9. Προσθέστε στον πίνακα Kritis το χαρακτηριστικό ηλικία.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαιο 8.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
Data Definition. http://dev.mysql.com/doc/refman/5.7/en/sql-syntax-data-definition.html
Data Manipulation.
http://dev.mysql.com/doc/refman/5.7/en/sql-syntax-data-manipulation.html
LOAD DATA. http://dev.mysql.com/doc/refman/5.7/en/load-data.html
ALTER TABLE. http://www.w3schools.com/sql/sql_alter.asp
DELETE FROM TABLE. http://www.w3schools.com/sql/sql_delete.asp
UPDATE TABLE. http://www.w3schools.com/sql/sql_update.asp
INSERT. http://www.w3schools.com/sql/sql_insert.asp
ON DELETE, ON UPDATE. http://dev.mysql.com/doc/refman/5.7/en/create-table-foreign-keys.html

Κεφάλαιο 5 Βασική Δομή Ερωτήματος SQL

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθεί η βασική δομή SQL ερωτήματος: SELECT..FROM..WHERE. Θα παρουσιαστούν οι βασικοί τελεστές σύγκρισης και οι λογικές πράξεις που περιλαμβάνονται στο πρότυπο της SQL. Το εργαστηριακό μέρος θα επικεντρωθεί στη σύνταξη ερωτημάτων που εμπλέκουν έναν πίνακα μόνο.

Προαπαιτούμενη γνώση
Αν και δε χρειάζεται απαραίτητα να γνωρίζει κανείς τα της δημιουργίας, επισκόπησης και αλλαγής πινάκων για να ασχοληθεί με SQL ερωτήματα, εντούτοις θα διευκόλυνε τη διαδικασία η επισκόπηση της ύλης που παρουσιάστηκε στα Κεφ. 3 και 4.

5.1 Εισαγωγικές Έννοιες

Έχοντας δημιουργήσει μία ΒΔ μπορούμε να θέσουμε ερωτήματα (queries) για να ανακτήσουμε εκείνο το μέρος της αποθηκευμένης πληροφορίας που μας ενδιαφέρει κάθε φορά. Η βασική δομή ενός SQL ερωτήματος είναι η ακόλουθη:

SELECT *πεδία*
FROM *πίνακες*
WHERE *λογική_συνθήκη*

Το παραπάνω ερώτημα επιστρέφει τις εγγραφές από το *πίνακες*, που πληρούν τη *λογική_συνθήκη* και τις εμφανίζει μόνο ως προς τα *πεδία*. Δυνητικά η εκτέλεση του παραπάνω ερωτήματος γίνεται ως εξής. Πρώτα θα εκτελεστεί το FROM. Στο FROM μπορεί να έχω μία πολύπλοκη έκφραση που να εμπλέκει περισσότερους από έναν πίνακες. Το αποτέλεσμα του υπολογισμού της έκφρασης του FROM θα είναι ένας τελικός πίνακας. Ο πίνακας αυτός δεν υπάρχει στη αποθηκευμένος στη ΒΔ. Είναι μόνο προσωρινός και χρησιμεύει για την απάντηση του ερωτήματος. Στη συνέχεια θα εκτελεστεί το WHERE. Μία προς μία οι εγγραφές του πίνακα που δημιουργήθηκε από το FROM θα υπολογιστεί αν πληρούν τη *λογική_συνθήκη*. Όσες την πληρούν κρατιούνται στο τελικό αποτέλεσμα ενώ οι υπόλοιπες διαγράφονται. Τέλος εκτελείται το SELECT όπου προβάλλονται στο τελικό αποτέλεσμα οι εγγραφές που επεστράφησαν από το WHERE όχι όμως ως προς όλα τα πεδία του πίνακα που προέκυψε από το FROM αλλά μόνο ως προς τα *πεδία* που καθορίζει το SELECT. Ο παραπάνω τρόπος εκτέλεσης (πρώτα το FROM μετά το WHERE και τέλος το SELECT) είναι δυνητικός και θα χρησιμέψει μόνο στο να συντάσσουμε ερωτήματα με ορθό τρόπο. Το DBMS μπορεί να ακολουθήσει διαφορετικό τρόπο εκτέλεσης ενός ερωτήματος (περισσότερα σε επόμενο κεφάλαιο). Τέλος θα πρέπει να σημειωθεί ότι τα αποτελέσματα ενός ερωτήματος είναι και αυτά σε μορφή πίνακα, ο οποίος είναι προσωρινός και δεν αποθηκεύεται στη ΒΔ.

5.1.1 Αριθμητικοί λογικοί και τελεστές σύγκρισης

Οι *λογικοί τελεστές* που υποστηρίζονται είναι οι γνωστοί μας από την άλγεβρα Bool και τις γλώσσες προγραμματισμού και έχουν ως εξής:

OR ή ||: Λογικό Ή
XOR: Αποκλειστικό Ή
AND ή &&: Λογικό ΚΑΙ
NOT ή !: Λογική άρνηση

Οι *τελεστές σύγκρισης* είναι οι: <, >, <=, >=, =, <>, !=, με τους δύο τελευταίους να είναι ίδιοι (υλοποιούν το διάφορο).

Οι *αριθμητικές πράξεις* που υποστηρίζονται είναι οι: +, -, *, /, DIV, MOD, %, με το DIV να είναι η ακέραια διαίρεση και τα MOD και % το υπόλοιπο της διαίρεσης ακεραίων.

Οι δυαδικές αριθμητικές πράξεις είναι οι:

&: bitwise ΚΑΙ
|: bitwise Ή
^: bitwise XOR
~: αντιστροφή όλων των bits
<<: μετατόπιση αριστερά (shift left)
>>: μετατόπιση δεξιά (shift right)

Σημείωση 1: Η προτεραιότητα μεταξύ των τελεστών ακολουθεί γενικά τους ίδιους κανόνες που ισχύουν στις περισσότερες γλώσσες προγραμματισμού δηλ. πρώτα οι παρενθέσεις, μετά οι πράξεις, στη συνέχεια οι συγκρίσεις και τέλος οι λογικοί τελεστές.

Σημείωση 2: Το αποτέλεσμα μιας λογικής έκφρασης μπορεί να είναι 1 (TRUE), 0 (FALSE) ή NULL (UNKNOWN). Συγκρίσεις όπου τουλάχιστον το ένα μέρος είναι NULL δίνουν ως αποτέλεσμα NULL.

Παραδείγματα:

SELECT 7=NULL; Δίνει NULL.
SELECT 7<NULL; Δίνει NULL.
SELECT NULL=NULL; Δίνει NULL.
SELECT 7=7; Δίνει 1.
SELECT 7<>7; Δίνει 0.

Παρατηρήστε ότι τόσο το FROM όσο και το WHERE έχουν παραλειφθεί στις παραπάνω εντολές. Το WHERE όποτε παραλείπεται υπονοείται WHERE TRUE, ενώ όσον αφορά το FROM η MySQL παρέχει έναν dummy πίνακα τον DUAL ώστε αν θέλει κάποιος να μπορεί να γράψει τις παραπάνω πράξεις σε μορφή ερωτήματος σε πίνακα πχ.:

```
SELECT 7=NULL
FROM DUAL
WHERE TRUE;
```

Σημείωση 3: Επειδή το αποτέλεσμα λογικών εκφράσεων μπορεί να είναι NULL, η SQL παρέχει τον τελεστή σύγκρισης: <=> (NULL safe equality). Ο τελεστής λειτουργεί όπως η ισότητα, με τη διαφορά ότι σύγκριση μεταξύ μη NULL και NULL δίνει FALSE ενώ μεταξύ NULL και NULL TRUE.

Παραδείγματα:

SELECT 7<=>NULL; Δίνει 0.
SELECT NULL<=>NULL; Δίνει 1.

Σημείωση 4: Στη γενική περίπτωση αλφαριθμητικά μετέχουν ως ορίσματα μέσα σε single quotes πχ. 'abc'. Οι συγκρίσεις μεταξύ αλφαριθμητικών δεν είναι case sensitive ενώ αγνοούνται τα κενά στο τέλος του αλφαριθμητικού (trailing blanks). Όταν ένα αλφαριθμητικό αναπαριστά αριθμό και μετέχει σε αριθμητικές πράξεις γίνεται αυτόματη μετατροπή του στην αντίστοιχη αριθμητική τιμή.

Παραδείγματα:

SELECT 'abc' = 'ABC'; Δίνει 1.
SELECT 'abc' = 'ABC '; Δίνει 1.
SELECT 'abc' = ' ABC'; Δίνει 0.
SELECT 'abc' + 1; Δίνει 1 (0+1).
SELECT '10' + 1; Δίνει 11.
SELECT '10.87' - 1; Δίνει 9.87.

Σημείωση 5: Οι δυαδικές πράξεις γίνονται σε ακεραίους των 64-bit (BIGINT). Αν τα ορίσματα δεν είναι BIGINT μετατρέπονται σε τέτοια. Η μετατροπή γίνεται ακόμα και αν τα ορίσματα είναι αλφαριθμητικά, στην οποία περίπτωση η τιμή τους είναι 0 και λαμβάνουμε warning.

Παραδείγματα:

SELECT 31^10; Δίνει 21 (11111 XOR 01010 = 10101 = 21).
SELECT 20&11; Δίνει 0 (10100 AND 01011 = 00000 = 0).
SELECT 20|11; Δίνει 31 (10100 AND 01011 = 11111 = 31).
SELECT ~0; Δίνει 18446744073709551615 (ο μέγιστος 64-bit integer).
SELECT 10^'abc'; Δίνει 10 (01010 XOR 00000 = 01010 = 10).
SELECT 10>>1; Δίνει 5 (διαίρεση με το 2. shift left (01010) = 0101 = 5);
SELECT 10<<3; Δίνει 80 (3 διαδοχικοί πολλαπλασιασμοί με το 2. 01010000 = 80).

5.1.2 Άλλοι τελεστές σύγκρισης

Η MySQL υποστηρίζει επιπλέον τελεστές για χειρισμό τύπων και συγκρίσεις. Για περισσότερες πληροφορίες μπορεί κανείς να ανατρέξει στο Κεφ.13.2 του εγχειριδίου. Στη συνέχεια παρουσιάζουμε κάποιες βασικές που πρόκειται να χρησιμοποιηθούν στα πλαίσια του εργαστηρίου και δίνουμε παραδείγματα χρήσης.

- IS και IS NOT. Ελέγχει την Boolean τιμή μιας έκφρασης. Οι Boolean τιμές είναι: TRUE, FALSE, UNKNOWN.
  Παράδειγμα:
  SELECT 5>NULL IS UNKNOWN; Δίνει 1.
  SELECT (5>NULL) <=> (5<NULL) IS NOT TRUE; Δίνει 0.

- IS NULL και IS NOT NULL. Ελέγχει αν μία έκφραση είναι NULL ή όχι.
  Παράδειγμα:
  SELECT (5>NULL) IS NOT NULL; Δίνει 0.
  SELECT 5<7 IS NULL; Δίνει 0.

- BETWEEN *τιμή1* AND *τιμή2*. Ελέγχει αν μία έκφραση είναι μεγαλύτερη ίση από *τιμή1* και μικρότερη ίση από *τιμή2*.
  Παράδειγμα:
  SELECT 5 BETWEEN 5 AND 7; Δίνει 1.
  SELECT 6 BETWEEN 5 AND 7; Δίνει 1.
  SELECT 7 BETWEEN 5 AND 7; Δίνει 1.
  SELECT 8 BETWEEN 5 AND 7; Δίνει 0.

- LIKE και NOT LIKE. Ελέγχει αν ένα όρισμα ταιριάζει με κάποιο αλφαριθμητικό πρότυπο. Η σύγκριση γίνεται χαρακτήρα, χαρακτήρα και επομένως τα κενά στο τέλος παίζουν ρόλο. Στο πρότυπο μπορούν να χρησιμοποιηθούν τα εξής δύο wildcards: ο χαρακτήρας '_' που σημαίνει οποιοσδήποτε χαρακτήρας και ο χαρακτήρας '%' που σημαίνει οποιαδήποτε σειρά χαρακτήρων.
  Παράδειγμα:
  SELECT 'Robert' LIKE '_o%t'; Δίνει 1.
  SELECT 'Robert' NOT LIKE '%r_'; Δίνει 0.

- CAST (*όρισμα* AS *τύπος*). Μετατρέπει το *όρισμα* στον προσδιοριζόμενο *τύπο*. Οι τύποι στους οποίους μπορεί να γίνει μετατροπή είναι: DATE, TIME, DATETIME, SIGNED [INTEGER], UNSIGNED [INTEGER], CHAR, CHAR(N) (μετατροπή σε αλφαριθμητικό συγκεκριμένου μήκους), BINARY (μετατροπή σε bitstring), BINARY(N) (μετατροπή σε bitstring μεγέθους N). Η μετατροπή ενός αλφαριθμητικού σε BINARY σημαίνει ότι οι συγκρίσεις γίνονται bit προς bit και το character set είναι το latin1. Για περισσότερες πληροφορίες μπορεί κανείς να ανατρέξει στα Κεφ.12.10 και Κεφ.11.4.2 του εγχειριδίου.
  Παράδειγμα:
  Η συνάρτηση NOW() επιστρέφει την τρέχουσα ημ/νια και ώρα (DATETIME).
  SELECT NOW(); Δίνει πχ. 2015-11-12 16:39:49.
  SELECT CAST(NOW() AS DATE); Δίνει πχ. 2015-11-12.

SELECT CAST(NOW() AS TIME); Δίνει πχ. 16:39:49.
SELECT CAST('123' AS SIGNED); Δίνει 123.
SELECT CAST('The working place where I work' AS CHAR(10)); Δίνει The workin.
SELECT 'The ' = 'The'; Δίνει 1.
SELECT CAST('The ' AS BINARY) = CAST('The' AS BINARY); Δίνει 0.

5.1.3 Συναρτήσεις αλφαριθμητικών

Η MySQL5.7 υποστηρίζει μία πληθώρα τελεστών και συναρτήσεων όσον αφορά στο χειρισμό αλφαριθμητικών (Κεφ.12.5 στο εγχειρίδιο). Στη συνέχεια συνοψίζονται κάποιες από αυτές.

- CHAR_LENGTH(*αλφαριθμητικό*). Επιστρέφει το πλήθος των χαρακτήρων του *αλφαριθμητικού.*
  Παράδειγμα:
  SELECT CHAR_LENGTH('MySQL5.7'); Δίνει 8.

- UPPER(*αλφαριθμητικό*). Μετατροπή σε κεφαλαία.
  Παράδειγμα:
  SELECT UPPER('MySql5.7'); Δίνει MYSQL5.7.

- LOWER(*αλφαριθμητικό*). Μετατροπή σε πεζά.
  Παράδειγμα:
  SELECT LOWER('MySql5.7'); Δίνει mysql5.7.

- CONCAT(*α1*, *α2*, ..). Συνενώνει τα αλφαριθμητικά *α1*, *α2*, κλπ.
  Παράδειγμα:
  SELECT CONCAT('My', 'SQL5.7'); Δίνει MySQL5.7.

- ASCII(*αλφαριθμητικό*). Επιστρέφει τον ascii αριθμό του πρώτου χαρακτήρα του αλφαριθμητικού.
  Παράδειγμα:
  SELECT ASCII('M'); Δίνει 77.

- BIN(*ακέραιος*). Επιστρέφει αλφαριθμητικό με τη δυαδική αναπαράσταση του *ακεραίου*.
  Παράδειγμα:
  SELECT BIN(10); Δίνει 1010.

- HEX(*όρισμα*). Επιστρέφει αλφαριθμητικό με τη δεκαεξαδική αναπαράσταση του ορίσματος (ακεραίου ή αλφαριθμητικού). Σε περίπτωση που το όρισμα είναι αλφαριθμητικό επιστρέφονται για κάθε χαρακτήρα που το απαρτίζει, δύο δεκαεξαδικοί αριθμοί που αντιστοιχούν στο byte του χαρακτήρα.
  Παράδειγμα:
  SELECT HEX(255); Δίνει FF.
  SELECT HEX('My'); Δίνει 4D79.
  Εξήγηση: SELECT BIN(ASCII('M')); Δίνει 1001101 δηλ. 0100 1101. Η πρώτη τετράδα αντιστοιχεί στο 4 στο δεκαεξαδικό και η δεύτερη στο D. Ομοίως για το ‘y’.

- STRCMP(*α1*, *α2*). Συγκρίνει τα αλφαριθμητικά *α1* και *α2*. Αν *α1* και *α2* είναι ίδια επιστρέφει 0, αλλιώς αν το *α1* είναι μικρότερο του *α2* επιστρέφει -1, αλλιώς επιστρέφει 1. Η σύγκριση γίνεται όμοια με τον τελεστή =.
  Παράδειγμα:
  SELECT STRCMP('Anna', 'George'); Δίνει -1.
  SELECT STRCMP('George', 'Anna'); Δίνει 1.
  SELECT STRCMP('Anna ', 'Anna'); Δίνει 0.

- REPLACE(*a1*, *a2*, *a3*). Επιστρέφει το *α1* μόνο που όπου υπάρχει το *α2* αντικαθίσταται με το *α3*.
  Παράδειγμα:
  SELECT REPLACE('MyQL5.7', 'QL', 'SQL'); Δίνει MySQL5.7.

- LOCATE(*a1*, *a2*). Επιστρέφει τη θέση μέσα στο *α2* όπου εμφανίζεται το *α1* για πρώτη φορά.
  Παράδειγμα:
  SELECT LOCATE('work', 'The working place where I work'); Δίνει 5.

- SUBSTR ή SUBSTRING(*αλφαριθμητικό*, *θέση*, *μέγεθος*). Επιστρέφει από το *αλφαριθμητικό* το υποαλφαριθμητικό που ξεκινά από την προσδιοριζόμενη *θέση* και είναι μήκους ίσο με το *μέγεθος* που δίνεται. Αν δε δίνεται το *μέγεθος* επιστρέφεται το υποαλφαριθμητικό από τη *θέση* και μέχρι το τέλος του *αλφαριθμητικού*.
  Παράδειγμα:
  SELECT SUBSTR('The working place where I work', 5, 4); Δίνει work.

- TRIM. Ως TRIM([LEADING|TRAILING|BOTH] *a1* FROM *a2*) κόβει το α1 από το πρόθεμα ή το επίθεμα ή και τα δύο του α2. Ως TRIM(*αλφαριθμητικό*) κόβει τα κενά τόσο στην αρχή όσο και στο τέλος του *αλφαριθμητικού*. Αν θέλουμε να κοπούν μόνο τα κενά στο τέλος μπορούμε να χρησιμοποιήσουμε την RTRIM(*αλφαριθμητικό*).
  Παράδειγμα:
  SELECT TRIM(BOTH 'ab' FROM 'ababMySQL5.7abab'); Δίνει MySQL5.7.

- SOUNDEX(*αλφαριθμητικό*). Επιστρέφει τη soundex τιμή του *αλφαριθμητικού*. Ιδεατά δύο αλφαριθμητικά που προφέρονται το ίδιο έχουν και την ίδια soundex τιμή. Μπορούμε να συγκρίνουμε τις soundex τιμές δύο αλφαριθμητικών ή με τον τελεστή =, ή με την εντολή:
  a1 SOUNDS LIKE *a2*.
  Παράδειγμα:
  SELECT SOUNDEX('Thomas'); Δίνει T520 (την τιμή soundex).
  SELECT 'Thomas' SOUNDS LIKE 'Tomas'; Δίνει 1.

5.1.4 Συναρτήσεις ημ/νιας και χρόνου

Η MySQL 5.7 υποστηρίζει μια πληθώρα συναρτήσεων για χειρισμό ημ/νιών και χρόνου, που περιγράφονται στο Κεφ. 12.7 του εγχειριδίου. Στη συνέχεια συνοψίζονται κάποιες από αυτές.

- NOW(), CURRENT_TIMESTAMP(). Όπως είδαμε στην προηγούμενη υποενότητα η NOW() επιστρέφει την τρέχουσα ημ/νία και ώρα (τύπος DATETIME). Ισοδύναμη με τη NOW() είναι η CURRENT_TIMESTAMP. Ως όρισμα στις συναρτήσεις αυτές μπορούμε να περάσουμε την επιθυμητή ακρίβεια ως υποδιαίρεση του δευτερολέπτου. Η μέγιστη δυνατή ακρίβεια είναι της τάξης του μsec και ως εκ τούτου μέγιστη τιμή στο όρισμα μπορεί να είναι το 6. Αν δεν προσδιοριστεί η επιθυμητή ακρίβεια υπονοείται 0 δηλ. ακρίβεια δευτερολέπτου.
  Παράδειγμα:
  SELECT NOW(6); Δίνει πχ. 2015-10-14 09:27:05.913871.

- DATE(), TIME(), DAY() ή DAYOFMONTH(), DAYOFYEAR(), DAYOFWEEK(), MONTH(), WEEK(), YEAR(), HOUR(), MINUTE(), SECOND(), MICROSECOND(). Από έναν τύπο DATE ή DATETIME ή TIMESTAMP, εξάγει την πληροφορία που περιγράφεται από το όνομα της συνάρτησης.
  Παράδειγμα:
  SELECT MINUTE(NOW()); Αν η NOW() επέστρεφε το TIMESTAMP του προηγούμενου παραδείγματος δηλ. 2015-10-14 09:27:05 η εντολή θα επέστρεφε 27.

SELECT DATE(NOW()); Στο παράδειγμα επιστρέφει: 2015-10-14.
SELECT TIME(NOW()); Στο παράδειγμα επιστρέφει: 09:27:05.
SELECT MICROSECOND(NOW(6)); Στο παράδειγμα επιστρέφει: 913871.
SELECT MICROSECOND(NOW(1)); Στο παράδειγμα επιστρέφει: 900000.
SELECT MICROSECOND(NOW()); Επιστρέφει 0 ανεξαρτήτως του αποτελέσματος της NOW().
SELECT DAYOFMONTH(NOW()), DAYOFYEAR(NOW()), DAYOFWEEK(NOW()); Στο παράδειγμα επιστρέφει μία εγγραφή με τρία πεδία: 14, 287, 4.

- DAYNAME(), MONTHNAME(). Επιστρέφουν το όνομα της ημέρας και του μήνα αντίστοιχα.
  Παράδειγμα:
  SELECT DAYNAME('2015-10-14 09:27:05'), MONTHNAME('2015-10-14 09:27:05'); Επιστρέφει μία εγγραφή με δύο πεδία: Wednesday, October.

- ADDDATE(*όρισμα*, INTERVAL *τιμή τύπος*). Προσθέτει στο *όρισμα* το χρόνο που καθορίζεται από την τιμή και τον τύπο της τιμής. Επιτρεπτοί τύποι τιμών είναι όλες οι υποδιαιρέσεις χρόνου στο εύρος από YEAR μέχρι MICROSECOND καθώς και συνδυασμοί των DAY, HOUR, MINUTE, SECOND, MICROSECOND, ανά δύο με πρώτη τη μεγαλύτερη μονάδα μέτρηση πχ. DAY_HOUR, DAY_MINUTE κλπ. Επίσης υποστηρίζεται και το YEAR_MONTH. Τέλος μπορούμε να παραλείψουμε τον τύπο και τη λέξη INTERVAL στην οποία περίπτωση προστίθενται ημέρες.
  Παράδειγμα:
  SELECT ADDDATE('2015-10-14', INTERVAL '2-3' YEAR_MONTH);
  Επιστρέφει: 2018-01-14.
  SELECT ADDDATE('2015-10-14 09:27:05.913871', INTERVAL
  '2 0:0:0.5' DAY_MICROSECOND);
  Επιστρέφει: 2015-10-16 09:27:06.413871.
  SELECT ADDDATE('2015-10-14 09:27:05.913871', 10);
  Επιστρέφει: 2015-10-24 09:27:05.913871.

- ADDTIME(*όρισμα*, *διάστημα*). Προσθέτει στο *όρισμα* το χρονικό *διάστημα*.
  Παράδειγμα:
  SELECT ADDTIME('2015-10-14 09:27:05.999999', '0.000001'); Δίνει 2015-10-14 09:27:06.
  SELECT ADDTIME('2015-10-14 09:27:05.999999', '1 0:1:2.000001'); Δίνει 2015-10-15 09:28:08.
  SELECT ADDTIME('2015-10-14 09:27:05.9', '48:0:0'); Δίνει 2015-10-16 09:27:05.900000.

- TIMESTAMPADD(*τύπος*, *τιμή*, *όρισμα*). Όμοια με την ADDDATE μόνο που δεν υποστηρίζει ζευγάρια τύπων και η *τιμή* είναι αριθμός.
  Παράδειγμα:
  SELECT TIMESTAMPADD(SECOND, 1.5, '2015-10-14 09:27:05.913871');
  Επιστρέφει: 2015-10-14 09:27:07.413871.

- SUBDATE(), SUBTIME() . Συμμετρικές των ADDDATE() και ADDTIME() μόνο που αφαιρούν χρόνο.

- DATEDIFF(*όρισμα1*, *όρισμα2*). Επιστρέφει τη διαφορά σε ημέρες των δύο ορισμάτων (*όρισμα1 – όρισμα2*).
  Παράδειγμα:
  SELECT DATEDIFF(ADDTIME('2015-10-14 09:27:05.913871', '1 20:0:0'), '2015-10-14 09:27:05.913871'); Επιστρέφει 2.

- TIMEDIFF(*όρισμα1*, *όρισμα2*). Επιστρέφει τη διαφορά σε τύπο TIME των δύο ορισμάτων (*όρισμα1 – όρισμα2*). Τα ορίσματα πρέπει να είναι DATETIME ή TIME. Η μέγιστη δυνατή διαφορά που επιστρέφεται περιορίζεται από τον τύπο TIME και είναι στο εύρος: '-838:59:59' εώς '838:59:59'. Αν χρειαζόμαστε μεγαλύτερες τιμές θα πρέπει να χρησιμοποιηθεί η συνάρτηση TIMESTAMPDIFF().
  Παράδειγμα:
  SELECT TIMEDIFF('2015-10-14 09:27:05.913871', '2015-10-16 09:27:05.913871');
  Επιστρέφει: -48:00:00.000000.

- TIMESTAMPDIFF(*τύπος*, *όρισμα1*, *όρισμα2*). Επιστρέφει ακέραια τιμή ίση με τη διαφορά των δύο ορισμάτων (*όρισμα2* - *όρισμα1*) στη χρονική μονάδα που καθορίζεται από τον τύπο. Είναι δυνατή η αφαίρεση DATE και DATETIME τύπων στην οποία περίπτωση υπονοείται χρόνος 00:00:00 για το όρισμα τύπου DATE.
  Παράδειγμα:
  SELECT TIMESTAMPDIFF(MICROSECOND, '2015-10-14 09:27:05.913871', '2015-10-14 09:27:06.913871'); Επιστρέφει 1000000.
  SELECT TIMESTAMPDIFF(HOUR, '2015-10-15', '2015-10-14'); Επιστρέφει -24.

5.1.5 Εμφάνιση πεδίων

Όπως έχει ειπωθεί στο SELECT ενός ερωτήματος επιλέγουμε ένα ή περισσότερα πεδία για να εμφανιστούν. Αν και σε πίνακες που έχει οριστεί PRIMARY KEY δεν υπάρχει περίπτωση να υπάρξουν δύο ίδιες εγγραφές, εντούτοις στα αποτελέσματα των ερωτημάτων μπορεί να έχω διπλές εγγραφές. Αν αυτό δεν είναι επιθυμητό τότε μπορούμε να χρησιμοποιήσουμε την επιλογή DISTINCT.
Παράδειγμα:
SELECT name, sname
FROM Customer;
Αν υπάρχουν δύο πελάτες με το ίδιο ον/μο επιστρέφεται μία εγγραφή για τον καθένα.
SELECT DISTINCT name, sname
FROM Customer;
Οι διπλές εγγραφές κόβονται από το αποτέλεσμα.

Αν επιθυμούμε να εμφανίσουμε όλα τα πεδία του πίνακα που προκύπτει από το FROM, αντί να τα αναφέρουμε στο SELECT ένα προς ένα, μπορούμε να χρησιμοποιήσουμε το *.
Παράδειγμα:
SELECT *
FROM Customer;
Ισοδύναμο με το να έλεγα:
SELECT cid, afm, address, name, sname, dateOfBirth
FROM Customer;

Κάποιες φορές επιθυμούμε να αλλάξουμε το όνομα ενός πεδίου όπως αυτό εμφανίζεται στο αποτέλεσμα ενός ερωτήματος. Μπορεί να επιτευχθεί αυτό βάζοντας μετά το πεδίο AS *καινούργιο_όνομα*.
Παράδειγμα:
SELECT cid, CONCAT(name,' ', surname)
FROM Customer;
Εμφανίζει έναν πίνακα με τις εξής δύο στήλες: cid και CONCAT(name,' ', surname).
SELECT cid, CONCAT(name,' ', surname) AS fullName
FROM Customer;
Εμφανίζει τα ίδια αποτελέσματα με πριν σε έναν πίνακα με τις εξής δύο στήλες: cid και fullName.

Αν δώσουμε το ερώτημα:
SELECT name, cid, sname

FROM Customer;
θα δούμε ότι τα αποτελέσματα επιστρέφονται σε αύξουσα σειρά ως προς cid. Γενικά μιλώντας τα αποτελέσματα ενός ερωτήματος επιστρέφονται σε αύξουσα σειρά ως προς το PRIMARY KEY. Αν επιθυμούμε να επιστρέψουμε τα αποτελέσματα με άλλη σειρά μπορούμε να το κάνουμε προσθέτοντας στο τέλος του ερωτήματος την επιλογή ORDER BY.
Παράδειγμα:

```
SELECT *
FROM Customer
WHERE TRUE
ORDER BY sname ASC, dateOfBirth DESC;
```

Το ερώτημα θα εμφανίσει όλες τις εγγραφές του πίνακα Customer ταξινομημένες πρώτα ως προς αύξουσα σειρά επωνύμου (ASC από το ascending) και έπειτα (για τις εγγραφές με ίδιο επώνυμο) ταξινομημένες ως προς ημ/νια γέννησης σε φθίνουσα σειρά (DESC από το descending). Αν παραλείψουμε τους προσδιοριστές ASC/DESC το default είναι το ASC.

5.1.6 Παραδείγματα ερωτημάτων σε έναν πίνακα

Θεωρείστε το σχήμα που παρουσιάστηκε στο Κεφ. 3.1 το οποίο συνοψίζεται παρακάτω:
Customer (cid, afm, address, name, sname, dateOfBirth)
Phones (*cid*, pnum)
Account (accid, balance, dateOfCreation)
Owns (*cid*, *accid*)
Action (*accid*, actid, amount, type, dateOfAction)
Transfer (*accidSource*, *actidSource*, *accidDest*, *actidDest*)

Για να σχεδιάσουμε ένα ερώτημα το πρώτο πράγμα που πρέπει να κάνουμε είναι να κατανοήσουμε ποιόν ή ποιούς πίνακες εμπλέκει. Στη συνέχεια να εφαρμόσουμε έναν προς έναν τους περιορισμούς που περιγράφονται έχοντας υπόψη ότι η έκφραση που υπάρχει στο WHERE εφαρμόζεται για κάθε εγγραφή ξεχωριστά. Τέλος θα πρέπει να αποφασίσουμε τα πεδία που επιθυμούμε να επιστραφούν. Ακολουθούν προδιαγραφές ερωτημάτων και η σύνταξή τους σε SQL.

- Βρες τους πελάτες που τα ονόματά τους είναι 'Kostas' και 'Maria'.
  Κατ' αρχήν εντοπίζουμε ότι το ερώτημα αφορά τον πίνακα Customer. Το κριτήριο στο WHERE έχει σχέση με τα ονόματα. Η εκφώνηση μπορεί εσφαλμένα να οδηγήσει στο συμπέρασμα ότι το WHERE πρέπει να γραφτεί ως εξής:
  WHERE name='Kostas' AND name='Maria'.
  Κάτι τέτοιο θα επιστρέψει πάντα κενό αποτέλεσμα ασχέτως των εγγραφών που υπάρχουν στον πίνακα. Ο λόγος είναι γιατί για να επιστραφεί μία εγγραφή θα πρέπει στο πεδίο name να έχει και τις δύο τιμές, πράγμα αδύνατο. Ο σωστός λογικός τελεστής λοιπόν είναι το OR. Τέλος στο ερώτημα αυτό δεν προσδιορίζονται τα πεδία που θέλουμε να επιστραφούν άρα μπορούμε να υποθέσουμε ότι αφορά όλα τα πεδία.

  ```
  SELECT *
  FROM Customer
  WHERE name='Kostas' OR name='Maria';
  ```

- Βρες τους πελάτες των οποίων το ΑΦΜ έχει δεύτερο και τελευταίο ψηφίο το 8 και έχουν γεννηθεί τη δεκαετία του 80 με εξαίρεση αυτούς που γεννήθηκαν το 1985 ή το 1987.
  Όταν έχουμε πιο πολύπλοκα κριτήρια μπορεί να μας διευκολύνει το να γράψουμε ένα, ένα τα κριτήρια και στο τέλος να συνθέσουμε την τελική λογική έκφραση.
  ΑΦΜ8: afm LIKE '_8%8'
  δεκαετία80: YEAR(dateOfBirth) BETWEEN 1980 AND 1989
  ημΓεν8587: YEAR(dateOfBirth) = 1985 OR YEAR(dateOfBirth) = 1987
  Μπορούμε τώρα να σχεδιάσουμε το WHERE ως εξής:

ΑΦΜ8 AND δεκαετία80 AND NOT ημΓεν8587
Κατά συνέπεια το τελικό ερώτημα έχει ως εξής:

```
SELECT *
FROM Customer
WHERE  afm LIKE '_8%8' AND
        YEAR(dateOfBirth) BETWEEN 1980 AND 1989 AND
        NOT (YEAR(dateOfBirth) = 1985 OR YEAR(dateOfBirth) = 1987);
```

- Βρες τα id των λογαριασμών χωρίς διπλές εγγραφές, για τα οποία διενεργήθηκε πράξη ανάληψης ποσού μεγαλύτερο των 10000 κατά τις προηγούμενες δύο ημέρες.
  Ο πίνακας που θα χρησιμοποιηθεί είναι ο Action. Αυτό που θα επιστραφεί στο SELECT είναι οι κωδικοί λογαριασμών δηλ. acid χωρίς διπλές εγγραφές, επομένως χρησιμοποιώντας DISTINCT. Τα κριτήρια του WHERE έχουν ως εξής:
  πράξη: Υποθέτοντας ότι οι αναλήψεις έχουν κωδικό 2, type=2
  ποσό: amount>10000
  ημέρες: DATEDIFF(NOW(), dateOfAction)>=2. Αν το πεδίο dateOfAction είναι DATETIME και το κριτήριο «τις προηγούμενες δύο ημέρες» πρέπει να είναι μέχρι και με ακρίβεια sec. τότε το κριτήριο μπορεί να γραφτεί ως εξής:
  SUBTIME(NOW(), '2 0:0:0') < dateOfAction

```
SELECT DISTINCT accid
FROM Action
WHERE type=2 AND amount>10000 AND SUBTIME(NOW(), '2 0:0:0') <= dateOfAction;
```

Σημείωση: Υποδιαιρέσεις του sec αν και μπορεί να χρησιμοποιηθούν δεν μπορούν να σωθούν στην MySQL5.7. Αν θέλαμε να σώσουμε και τα MICROSECOND ενός χρόνου, θα έπρεπε να τα εξάγουμε με τη σχετική συνάρτηση και να τα σώσουμε σε ξεχωριστό πεδίο.

5.2 Παράδειγμα Εργαστηριακής Άσκησης

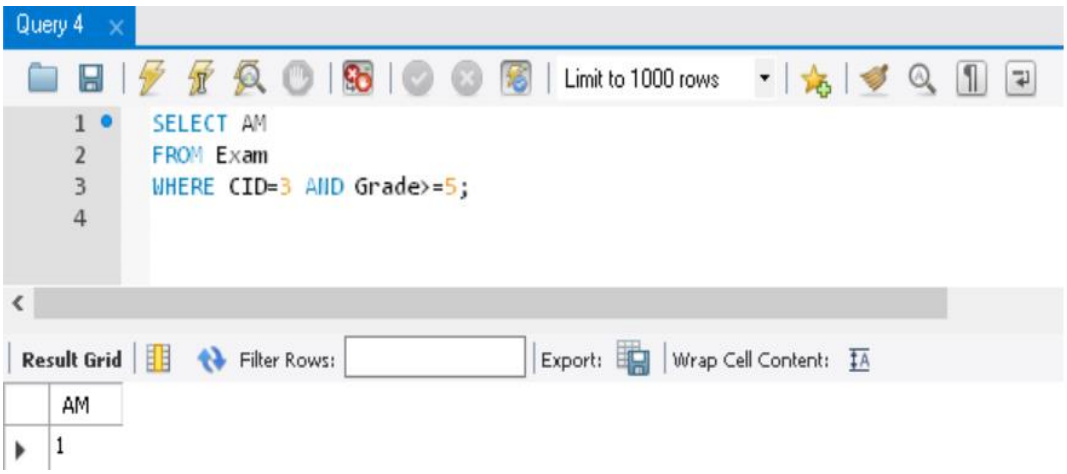
Εκφώνηση:	Στη Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 16. Επιλέξτε τους φοιτητές που έχουν περάσει το μάθημα με κωδικό 3. 17. Επιλέξτε τον καθηγητή που διδάσκει το μάθημα ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ. 18. Αναζητήστε τους φοιτητές του 4ου έτους. 19. Βρείτε τα ΑΜ των φοιτητών που έχουν κινητό τηλέφωνο. 20. Αναζητήστε τα ΑΜ των φοιτητών που έχουν περάσει τουλάχιστον ένα μάθημα. 21. Αναζητήστε τα ονόματα των καθηγητών των οποίων τα επώνυμα αρχίζουν από 'Τ' και τελειώνουν σε 's'. 22. Αναζητήστε τους καθηγητές των οποίων τα επώνυμα αρχίζουν από 'Τ' αλλά δεν τελειώνουν σε 's'.
Ζητούμενα:	Σύνταξη SQL ερωτημάτων σε έναν πίνακα.

Ζητούμενο 1: Επιλέξτε τους φοιτητές που έχουν περάσει το μάθημα με κωδικό 3.

Για τους φοιτητές μπορούμε να ζητήσουμε στο SELECT το ΑΜ ή το Lastname.
Δίνουμε:

```
SELECT AM
FROM Exam
WHERE CID=3 AND Grade>=5;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.1.



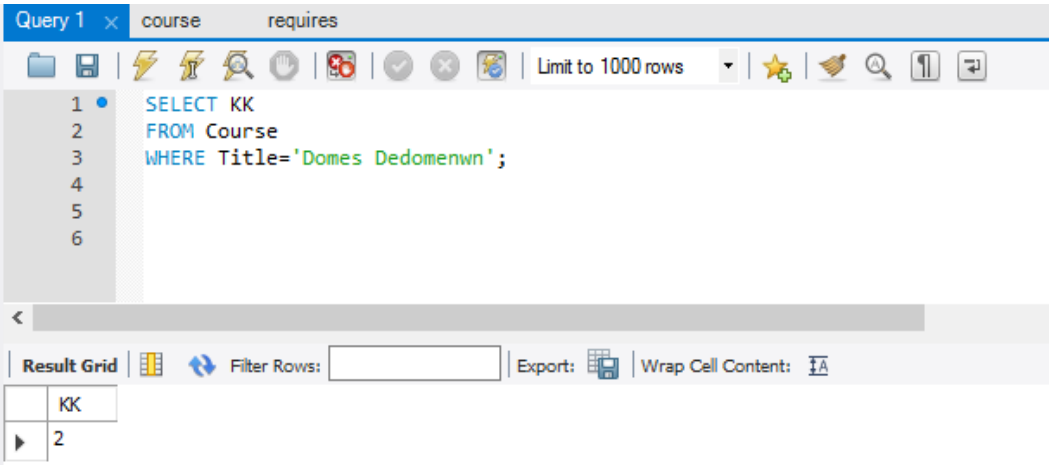
Εικόνα 5.1: *Στο παράδειγμα μόνο ένας φοιτητής έχει περάσει το μάθημα και έχει ΑΜ=1.*

Ζητούμενο 2: Επιλέξτε τον καθηγητή που διδάσκει το μάθημα ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ.

Χρησιμοποιώντας έναν πίνακα μπορώ να βρω τον κωδικό καθηγητή από τον πίνακα μαθημάτων.

Δίνουμε:
SELECT KK
FROM Course
WHERE Title='Domes Dedomenwn';

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.2.

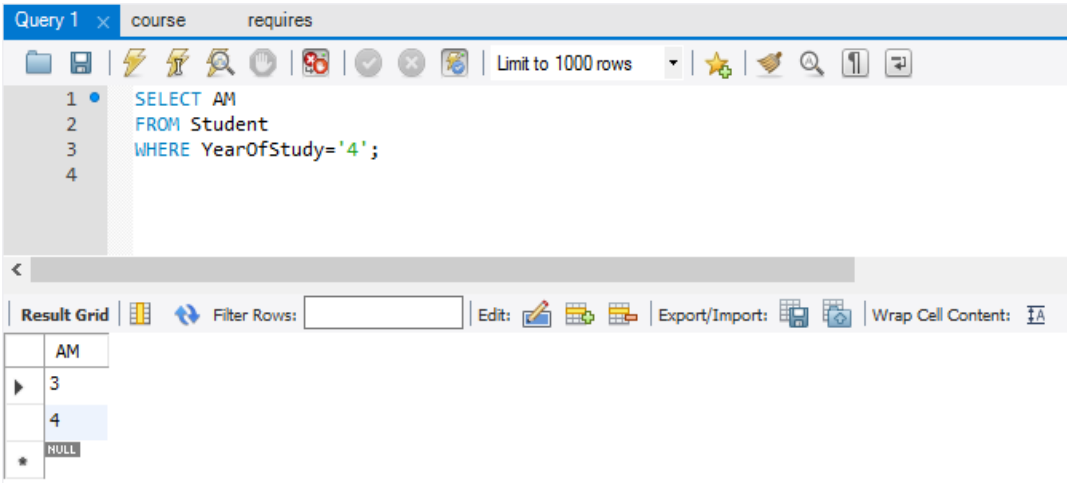


Εικόνα 5.2: *Μόνο ένας καθηγητής κάνει το μάθημα Δομές Δεδομένων και έχει κωδικό 2.*

Ζητούμενο 3: Αναζητήστε τους φοιτητές του 4^{ου} έτους.

Δίνουμε:
SELECT AM
FROM Student
WHERE YearOfStudy='4';

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.3.



Εικόνα 5.3: *Με τις εγγραφές που έχουμε μόνο δύο φοιτητές είναι στο 4ο έτος.*

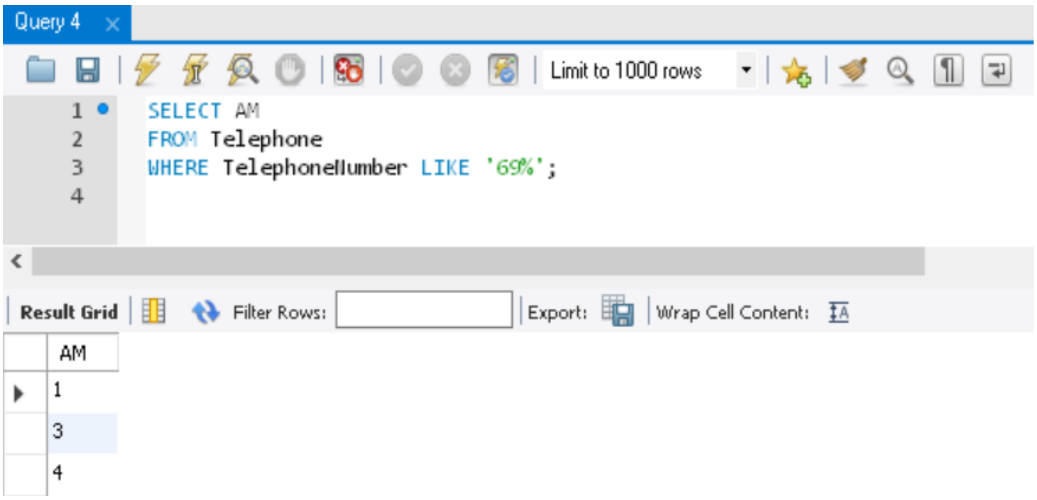
Ζητούμενο 4: Βρείτε τα ΑΜ των φοιτητών που έχουν κινητό τηλέφωνο.

Για να έχει ένας φοιτητής κινητό τηλέφωνο θα πρέπει το αντίστοιχο πεδίο να ξεκινάει από 6.

Δίνουμε:
SELECT AM

FROM Telephone
WHERE TelephoneNumber LIKE ‘69%’;

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.4.



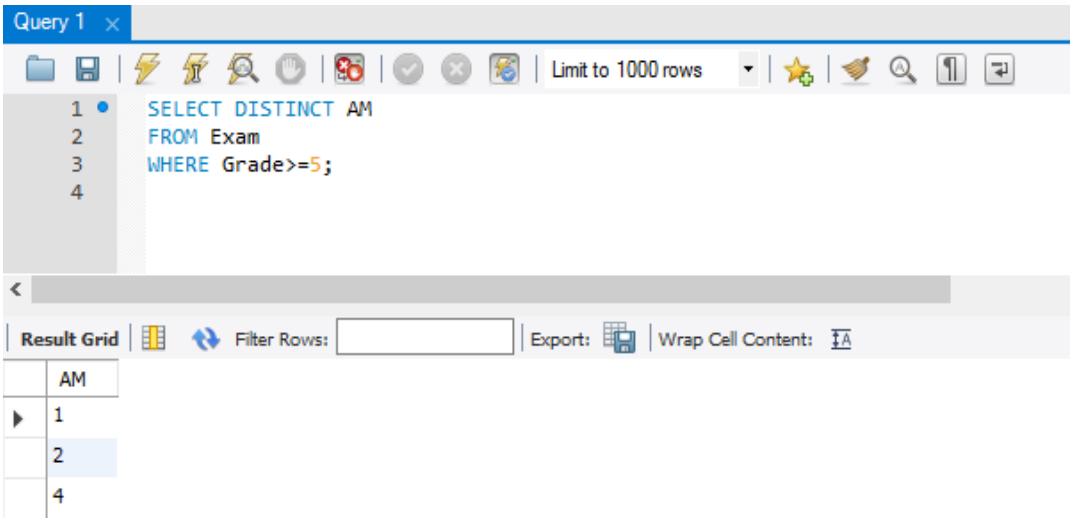
Εικόνα 5.4: *Τρεις φοιτητές από τους πέντε έχουν κινητό τηλέφωνο.*

Ζητούμενο 5: Αναζητήστε τα ΑΜ των φοιτητών που έχουν περάσει τουλάχιστον ένα μάθημα.

Χρησιμοποιώντας το DISTINCT, αν ένας φοιτητής έχει περάσει περισσότερα του ενός μαθήματα, θα επιστραφεί μόνο μια εγγραφή (για το πρώτο μάθημα που θα βρει) που είναι και το ζητούμενο.

Δίνουμε:
SELECT DISTINCT AM
FROM Exam
WHERE Grade>=5;

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.5.



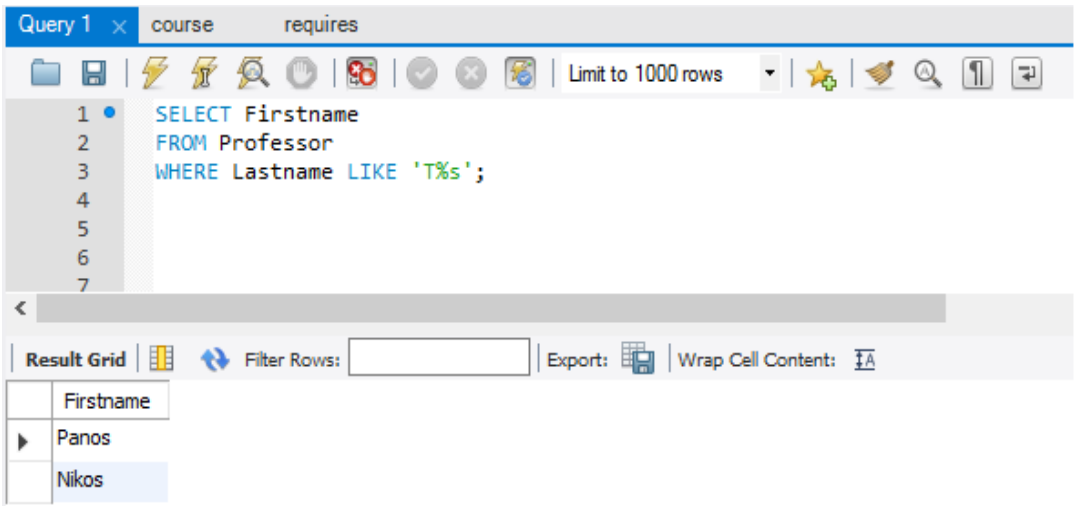
Εικόνα 5.5: *Δύο από τους πέντε φοιτητές έχουν περάσει κάποιο μάθημα.*

Ζητούμενο 6: Αναζητήστε τα ονόματα των καθηγητών των οποίων τα επώνυμα αρχίζουν από ‘T’ και τελειώνουν σε ‘s’.

Όπως έχουμε δει στη θεωρία, ο τελεστής % αναπαριστά 0 ή περισσότερους χαρακτήρες ενώ ο τελεστής _ αναπαριστά έναν χαρακτήρα. Στο συγκεκριμένο παράδειγμα δε μας ενδιαφέρουν τα ενδιάμεσα γράμματα στο επώνυμο, παρά μόνο το αρχικό και η κατάληξη, οπότε χρησιμοποιούμε ενδιάμεσα %.

Δίνουμε:
SELECT Firstname
FROM Professor
WHERE Lastname LIKE ‘T%s’;

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.6.



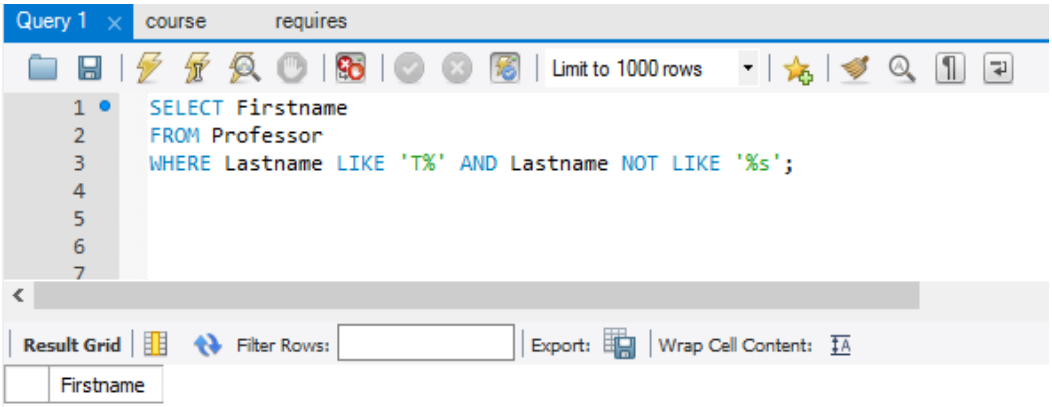
Εικόνα 5.6: *Οι καθηγητές που ικανοποιούν το ερώτημα είναι οι Tzimas και Theocharopoulos.*

Ζητούμενο 7: Αναζητήστε τους καθηγητές των οποίων τα επώνυμα αρχίζουν από 'T' αλλά δεν τελειώνουν σε 's'.

Εδώ ζητάμε δύο συνθήκες να ισχύουν στο WHERE η μία με LIKE και η άλλη με NOT LIKE.

Δίνουμε:
SELECT Firstname
FROM Professor
WHERE Lastname LIKE 'T%' AND Lastname NOT LIKE '%s';

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 5.7.



Εικόνα 5.7: *Κανένας καθηγητής δεν ικανοποιεί το ερώτημα.*

5.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

1. Βρείτε τους διευθυντές που έχουν 10 χρόνια προϋπηρεσίας.
2. Βρείτε όλους τους εργαζόμενους που το επώνυμό τους τελειώνει σε 'is'.
3. Βρείτε όλες τις εταιρείες που βρίσκονται στην οδό 'Μαιζώνος'.
4. Προσθέστε στον πίνακα Εργαζόμενος το πεδίο φύλο.
5. Βρείτε τους άνδρες εργαζόμενους που έχουν προσληφθεί από το 2000 και μετά.
6. Βρείτε πόσα χρόνια δουλεύει κάθε εργαζόμενος στην εταιρεία του.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2) απαντήστε στα ακόλουθα ερωτήματα:

1. Αναζητήστε όλες τις ταινίες που ο τίτλος τελειώνει σε 'ror'.
2. Βρείτε σε ποιες ταινίες πρωταγωνιστούσε ο ηθοποιός 'Tom Cruise'.
3. Βρείτε όλους τους σκηνοθέτες που είναι πάνω από 60 ετών.
4. Αναζητήστε όλους τους θεατές που έχουν βαθμολογήσει με 5 μια οποιαδήποτε ταινία.
5. Βρείτε σε ποιες ταινίες πρωταγωνιστούσε ο ηθοποιός 'Tom Cruise' ή πρωταγωνιστούσε η 'Natalie Portman' και είχαν παραπάνω από 10 εκατομμύρια εισητήρια.
6. Βρείτε τους κωδικούς των σκηνοθετών που είναι πάνω από 60 ετών ή σκηνοθέτησαν ταινία που πρωταγωνιστούσε ο ηθοποιός 'Tom Cruise'.

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3) απαντήστε στα ακόλουθα ερωτήματα.

1. Αναζητήστε όλα τα τουρνουά που ιδρύθηκαν μετά το 2000.
2. Αναζητήστε όλους τους ενήλικους παίκτες του τουρνουά.
3. Βρείτε τα αποτελέσματα του τουρνουά που κέρδισε ο 'Roger Federer'.
4. Βρείτε όλους τους παίκτες που έχουν πάνω από 5 νίκες στο 'Roland Garros'.
5. Αναζητήστε τους παίκτες που το όνομά τους αρχίζει από 'R'.
6. Βρείτε τα τουρνουά που διεξάγονται στη Γαλλία.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαιο 8.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
Data Manipulation.
http://dev.mysql.com/doc/refman/5.7/en/sql-syntax-data-manipulation.html
SELECT ... ORDER BY. http://www.w3schools.com/sql/sql_orderby.asp
SELECT DISTINCT. http://www.w3schools.com/sql/sql_distinct.asp
SELECT. http://www.w3schools.com/sql/sql_select.asp
SELECT AND OR. http://www.w3schools.com/sql/sql_and_or.asp
SELECT LIKE.
http://www.w3schools.com/sql/sql_like.asp
http://www.w3schools.com/sql/sql_wildcards.asp
SELECT WHERE. http://www.w3schools.com/sql/sql_where.asp

Κεφάλαιο 6 Βασικές Πράξεις Θεωρίας Συνόλων και SQL

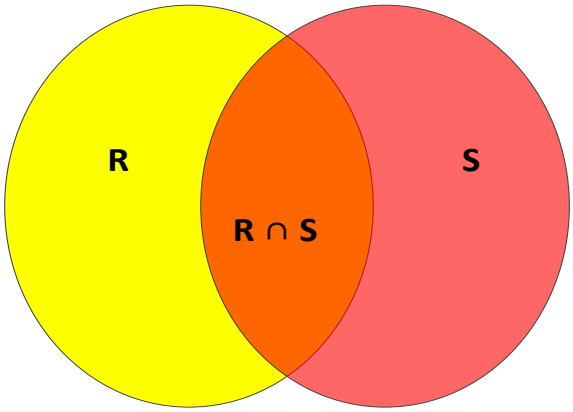
Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν οι βασικές πράξεις της θεωρίας συνόλων: καρτεσιανό γινόμενο, ένωση, τομή, διαφορά και η χρήση αυτών για να απαντηθούν ερωτήματα που εμπλέκουν παραπάνω από έναν πίνακα.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- Βασική δομή SQL ερωτήματος (Κεφ.5).

6.1 Εισαγωγικές Έννοιες

Έστω ότι έχουμε δύο σύνολα τα R και S που έχουν ιδίου τύπου στοιχεία. Θεωρείστε την παρακάτω αναπαράσταση των συνόλων R και S σε μορφή διαγραμμάτων Venn.



Ένωση συνόλων. Η πράξη της ένωσης συμβολίζεται με το ∪. Το αποτέλεσμά της R ∪ S είναι ένα νέο σύνολο που περιέχει τόσο τα στοιχεία του R όσο και του S. Στο διάγραμμα Venn είναι όλη η χρωματισμένη περιοχή ανεξαρτήτως χρώματος.
Τομή συνόλων. Η πράξη της τομής συμβολίζεται με το ∩. Το αποτέλεσμά της R ∩ S είναι ένα νέο σύνολο που περιέχει τα στοιχεία του R που είναι και στοιχεία του S, δηλ. τα κοινά στοιχεία των δύο συνόλων. Στο διάγραμμα Venn είναι η χρωματισμένη με πορτοκαλί περιοχή.
Διαφορά συνόλων. Η πράξη της διαφοράς συμβολίζεται με το -. Το αποτέλεσμά της R - S είναι ένα νέο σύνολο που περιέχει τα στοιχεία του R που δεν είναι στοιχεία του S, δηλ. τα στοιχεία του R που δεν ανήκουν στην τομή με το S. Στο διάγραμμα Venn είναι η χρωματισμένη με κίτρινο περιοχή, ενώ η χρωματισμένη με κόκκινο αντιστοιχεί στο S - R.

6.1.1 Ένωση, τομή και διαφορά στην SQL και MySQL5.7

Θεωρώντας τους πίνακες μιας σχεσιακής ΒΔ ως σύνολα εγγραφών, οι παραπάνω πράξεις βρίσκουν εφαρμογή κυρίως όποτε θέλουμε να συνδυάσουμε αποτελέσματα ερωτημάτων. Η υλοποίησή τους στην SQL γίνεται μέσω των εντολών UNION, INTERSECT και EXCEPT ή MINUS. Η γενικότερη μορφή σύνταξης είναι:

ερώτημα 1
UNION/INTERSECT/EXCEPT
ερώτημα 2

Προσοχή: Η MySQL στην έκδοση 5.7 δεν υποστηρίζει τις πράξεις INTERSECT και EXCEPT αλλά μόνο το UNION. Υπάρχει τρόπος για την υλοποίηση της τομής και της αφαίρεσης με χρήση υποερωτημάτων ο οποίος και θα συζητηθεί σε επόμενο κεφάλαιο. Παρόλα αυτά στα παραδείγματα που ακολουθούν και για λόγους εξάσκησης στις βασικές πράξεις συνόλων, περιλαμβάνονται και οι INTERSECT/EXCEPT.

Παραδείγματα χρήσης
Έστω το ακόλουθο σχήμα:
Customer (cid, afm, address, name, sname, dateOfBirth)
Phones (*cid*, pnum)
Account (accid, balance, dateOfCreation)
Owns (*cid*, *accid*)
Action (*accid*, actid, amount, type, dateOfAction)
Transfer (*accidSource*, *actidSource*, *accidDest*, *actidDest*)

- *Ερώτημα 1*: Βρες τους κωδικούς των πελατών που έχουν κινητό τηλέφωνο και το όνομά τους είναι Maria.
  Το ερώτημα εμπλέκει δύο πίνακες: τον Customer και τον Phones. Σκεπτόμενοι πάνω στο ερώτημα διαπιστώνουμε ότι στην ουσία αποτελείται από δύο υποερωτήματα (subqueries). Το πρώτο (έστω Query1) αφορά στην εύρεση των cid από τον πίνακα Phones που το pnum ξεκινά από 6, ενώ το δεύτερο (έστω Query2) αφορά στην εύρεση των cid από τον πίνακα Customer για τα οποία το name= 'Maria'. Έχοντας συντάξει τα Query1 και 2 το τελικό αποτέλεσμα θα το πάρουμε παίρνοντας την τομή τους αφού ζητάμε να ισχύουν και οι δύο συνθήκες.
  ```
  SELECT cid
  FROM Customer
  WHERE name='Maria'
  INTERSECT
  SELECT cid
  FROM Phones
  WHERE pnum LIKE '6%';
  ```
  Για οπτικούς λόγους μπορούμε να βάλουμε τα δύο υποερωτήματα μέσα σε παρενθέσεις και να συμπτύξουμε τις γραμμές, αφού κάθε υποερώτημα είναι μικρό, ως εξής:
  ```
  (SELECT cid FROM Customer WHERE name='Maria')
  INTERSECT
  (SELECT cid FROM Phones WHERE pnum LIKE '6%');
  ```

- *Ερώτημα 2*: Βρες τους κωδικούς των πελατών που έχουν κινητό τηλέφωνο ή το όνομά τους είναι Maria.
  Το ερώτημα είναι ίδιο με το παραπάνω μόνο που θέλουμε να ισχύει τουλάχιστον μία από τις δύο συνθήκες, πράγμα που σημαίνει ότι θέλουμε την ένωση των Query1 και Query2.
  ```
  (SELECT cid FROM Customer WHERE name='Maria')
  UNION
  (SELECT cid FROM Phones WHERE pnum LIKE '6%');
  ```

- *Ερώτημα 3*: Βρες τους κωδικούς των πελατών που έχουν κινητό τηλέφωνο και το όνομά τους δεν είναι Maria.
  Το ερώτημα τώρα απαιτεί αφαίρεση. Από το σύνολο όσων έχουν κινητό τηλέφωνο θα πρέπει να αφαιρεθεί το σύνολο όσων έχουν όνομα Maria.
  ```
  (SELECT cid FROM Customer WHERE name='Maria')
  EXCEPT
  (SELECT cid FROM Phones WHERE pnum LIKE '6%');
  ```

- *Ερώτημα 4*: Βρες τους κωδικούς των πελατών που έχουν κινητό τηλέφωνο ή το όνομά τους είναι Maria ή έχουν λογαριασμό που ο κωδικός του είναι στο εύρος [1000…2000].

Το ερώτημα εμπλέκει τώρα και τρίτο υποερώτημα (έστω Query3) που τρέχει στον πίνακα Owns. Από την εκφώνηση συνάδουμε ότι η τελική απάντηση είναι το UNION και των τριών υποερωτημάτων. Στην SQL μπορούμε να έχουμε περισσότερα των δύο υποερωτημάτων να συνδυάζονται μέσω των UNION, INTERSECT και EXCEPT.

```
(SELECT cid FROM Customer WHERE name='Maria')
UNION
(SELECT cid FROM Phones WHERE pnum LIKE '6%')
UNION
(SELECT cid FROM Owns WHERE accid BETWEEN 1000 AND 2000);
```

- *Ερώτημα 5 (Αποκλειστικό Ή)*: Βρες τους κωδικούς των πελατών που <u>είτε</u> έχουν κινητό τηλέφωνο <u>ή</u> το όνομά τους είναι Maria <u>αλλά όχι και τα δύο</u> συγχρόνως.
  Στην ουσία αυτό που ζητάμε είναι Query1 XOR Query2. Κατ' αντιστοιχία στο Venn διάγραμμα ζητάμε τις περιοχές που είναι κίτρινες, κόκκινες αλλά όχι πορτοκαλί. Επομένως ζητάμε να αφαιρέσουμε από την ένωση την τομή. Θα θέλαμε λοιπόν να μπορούσαμε να γράψουμε κάτι σαν:

```
((SELECT cid FROM Customer WHERE name='Maria')
UNION
(SELECT cid FROM Phones WHERE pnum LIKE '6%'))
EXCEPT
((SELECT cid FROM Customer WHERE name='Maria')
INTERSECT
(SELECT cid FROM Phones WHERE pnum LIKE '6%'));
```

  Με άλλα λόγια να περικλείσουμε μέσα σε παρενθέσεις το αποτέλεσμα κάποιων πράξεων συνόλων για να δώσουμε προτεραιότητα. Δυστυχώς κάτι τέτοιο δεν μπορεί να γίνει (με αυτόν τον τρόπο) και θα θεωρηθεί συντακτικό λάθος. Για παράδειγμα αν και η εντολή:

```
(SELECT cid FROM Customer WHERE name='Maria')
UNION
(SELECT cid FROM Phones WHERE pnum LIKE '6%');
```

  εκτελείται χωρίς πρόβλημα, η εντολή:

```
((SELECT cid FROM Customer WHERE name='Maria')
UNION
(SELECT cid FROM Phones WHERE pnum LIKE '6%'));
```

  γεννά σφάλμα.
  Ο λόγος είναι ο εξής. Περικλείοντας μέσα σε παρενθέσεις το αποτέλεσμα μιας πράξης δηλώνουμε στο DBMS ότι αυτό το αποτέλεσμα είναι ένας πίνακας ο οποίος πρέπει να μετονομαστεί πριν χρησιμοποιηθεί. Περισσότερα για πίνακες που προκύπτουν από υποερωτήματα σε επόμενο κεφάλαιο.

<u>Σημείωση 1</u>: Η MySQL5.7 υποστηρίζει δύο μορφές UNION το UNION ALL και UNION DISTINCT που είναι και το default αν παραληφθεί το ALL/DISTINCT. Με το DISTINCT κόβονται επαναλήψεις ίδιου στοιχείου ενώ με το ALL κρατιούνται. Για παράδειγμα το ακόλουθο ερώτημα:

```
SELECT * FROM Customer
UNION ALL
SELECT * FROM Customer;
```

επιστρέφει τις εγγραφές του Customer δύο φορές.

<u>Σημείωση 2</u>: Η MySQL5.7 δε δίνει εγγύηση σχετικά με την ταξινόμηση του αποτελέσματος ενός UNION ακόμα και αν οι πίνακες είναι και οι δύο ταξινομημένοι ως προς τα επιθυμητά πεδία. Για παράδειγμα η ακόλουθη εντολή:

```
(SELECT * FROM Customer ORDER BY sname)
UNION ALL
(SELECT * FROM Customer ORDER BY sname);
```

δεν είναι απαραίτητο ότι θα έχει αποτέλεσμα ταξινομημένο ως προς sname. Για να το επιτύχουμε μπορούμε μετά την πράξη να βάλουμε ORDER BY. Για παράδειγμα:

```
SELECT * FROM Customer
UNION ALL
SELECT * FROM Customer
ORDER BY surname;
```

Επισημαίνεται το εξής. Καθώς είναι δυνατή η χρήση της ORDER BY για να ταξινομηθεί το αποτέλεσμα της UNION στην εντολή:

```
(SELECT * FROM Customer ORDER BY sname)
UNION ALL
(SELECT * FROM Customer ORDER BY sname);
```

θα δημιουργούνταν συντακτικό λάθος αν παραλείπαμε τις παρενθέσεις.

Σημείωση 3: Η MySQL5.7 απαιτεί τα δύο υποερωτήματα που γίνονται UNION να έχουν ίδιο πλήθος πεδίων και με αντίστοιχους τύπους. Δεν υπάρχει απαίτηση να έχουν το ίδιο όνομα. Σε περίπτωση διαφορετικών ονομάτων το αποτέλεσμα εμφανίζεται με ονόματα πεδίων τα ονόματα του πρώτου υποερωτήματος.

Παράδειγμα

- *Ερώτημα 6*. Βρες τους κωδικούς των λογαριασμών που έχουν εκτελέσει μεταφορά χρημάτων προς το λογαριασμό με κωδικό 100, ή ήταν αποδέκτες μεταφοράς χρημάτων από το λογαριασμό 200.

  Query1: SELECT accidSource FROM Transfer WHERE accidDest=100

  Query2: SELECT accidDest FROM Transfer WHERE accidSource=200

  Όπως προαναφέρθηκε μπορούν τα δύο ερωτήματα να ενωθούν, καθώς τα accidSource και accidDest είναι ίδιου τύπου. Το αποτέλεσμα θα έχει ένα πεδίο με όνομα accidSource. Αν θέλουμε άλλο όνομα στο αποτέλεσμα μετονομάζουμε το πεδίο που επιστρέφει η Query1.

  ```
  (SELECT accidSource AS results FROM Transfer WHERE accidDest=100)
  UNION
  (SELECT accidDest FROM Transfer WHERE accidSource=200);
  ```

  Υποθέτοντας ότι στον πίνακα Transfer υπάρχει μία μόνο εγγραφή η: <200,1,100,1>, το αποτέλεσμα του παραπάνω ερωτήματος θα είναι:

  ```
  +---------+
  | results |
  +---------+
  |     200 |
  |     100 |
  +---------+
  ```

6.1.2 Καρτεσιανό γινόμενο

Δοθέντος δύο συνόλων R και S που δεν έχουν απαραίτητα στοιχεία ιδίου τύπου, το καρτεσιανό γινόμενο των δύο συνόλων (RXS) ορίζεται σαν το σύνολο που αποτελείται από όλα τα ζευγάρια της μορφής (a, b) όπου το a ανήκει στο R και το b στο S. Με άλλα λόγια είναι ένα σύνολο που περιέχει όλους τους συνδυασμούς στοιχείων του R με το S.

Το καρτεσιανό γινόμενο είναι η βασική πράξη συνδυασμού δύο η περισσοτέρων πινάκων σε ένα ερώτημα. Για να εφαρμόσουμε καρτεσιανό γινόμενο δύο πινάκων στο FROM του αντίστοιχου ερωτήματος μπορούμε ή να γράψουμε τα ονόματα των δύο πινάκων χωρισμένα με κόμμα ή να χρησιμοποιήσουμε την εντολή CROSS JOIN.

Παράδειγμα

- *Ερώτημα 7*. Εμφάνισε το καρτεσιανό γινόμενο των πινάκων Customer και Phones.

  Πρώτος τρόπος:

```
SELECT *
FROM Customer, Phones;
```

Δεύτερος τρόπος:

```
SELECT *
FROM Customer CROSS JOIN Phones;
```

Ο δεύτερος τρόπος ενέχει την πράξη της συνένωσης (JOIN) στην οποία θα αναφερθούμε εκτενώς στο επόμενο κεφάλαιο. Τα παραδείγματα που δίνονται στη συνέχεια της ενότητας θα χρησιμοποιούν τον πρώτο τρόπο γραφής ερωτήματος.

Έστω οι ακόλουθοι πίνακες Customer και Phones:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333

Το καρτεσιανό γινόμενό τους είναι:

cid	afm	address	name	sname	dateOfBirth	cid	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	2231011111
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	1	2231011111
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	1	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	1	6944444444
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	1	6944444444
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	2	2103333333
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2	2103333333
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	2	2103333333

Παρατηρήστε ότι:

A) Οι εγγραφές του καρτεσιανού γινομένου είναι όλοι οι δυνατοί συνδυασμοί των εγγραφών των δύο πινάκων. Στη γενικότερη περίπτωση αν έχω στον πρώτο πίνακα Μ εγγραφές και στο δεύτερο Ν, στο καρτεσιανό θα υπάρχουν ΜΝ εγγραφές. Αυτό σημαίνει ότι ενδέχεται κάποιες εγγραφές που επιστρέφονται να είναι ίδιες.

B) Τα πεδία του καρτεσιανού γινομένου είναι τα πεδία και των δύο πινάκων. Σε περίπτωση που υπάρχουν κοινά πεδία στους δύο πίνακες (το cid στο παράδειγμα) εμφανίζονται δύο φορές στο αποτέλεσμα. Για να χρησιμοποιήσουμε κάποιο κοινό πεδίο στο ερώτημα θα πρέπει να

αναφερθούμε συγκεκριμένα στον πίνακα από τον οποίο προέρχεται, γράφοντας το όνομα του πίνακα μαζί με του πεδίου χωρισμένα με τελεία.

Παράδειγμα
SELECT cid, pnum
FROM Customer, Phones;
Γυρνάει το ακόλουθο σφάλμα:
ERROR 1052 (23000): Column 'cid' in field list is ambiguous.

SELECT Phones.cid, pnum
FROM Customer, Phones;
Γυρνάει:

cid	pnum
1	2231011111
1	2231011111
1	2231011111
1	6944444444
1	6944444444
1	6944444444
2	2103333333
2	2103333333
2	2103333333

Τέλος θα πρέπει να σημειωθεί ότι αν επιθυμούμε να κάνουμε καρτεσιανό γινόμενο ενός πίνακα με τον εαυτό του, τουλάχιστον ένας από τους δύο πίνακες θα πρέπει να μετονομαστεί μέσα στο ερώτημα. Για να μετονομάσουμε έναν πίνακα στο FROM μπορούμε να βάλουμε το επιθυμητό όνομα μετά το όνομα του πίνακα με ή χωρίς χρήση του AS. Προφανώς η μετονομασία αφορά στο ερώτημα και μόνο και όχι στον πίνακα που υπάρχει σωσμένος στη ΒΔ.

Παράδειγμα:
SELECT *
FROM Customer, Customer;
Επιστρέφει: ERROR 1066 (42000): Not unique table/alias: 'Customer'

SELECT C1.sname, C2.sname
FROM Customer AS C1, Customer C2;
Με τον Customer του παραδείγματος, επιστρέφεται:

sname	sname
Kostantinou	Kostantinou
Kostantinou	Kostantinou
Papantoniou	Kostantinou
Kostantinou	Kostantinou
Kostantinou	Kostantinou
Papantoniou	Kostantinou
Kostantinou	Papantoniou
Kostantinou	Papantoniou
Papantoniou	Papantoniou

6.1.3 Παραδείγματα ερωτημάτων με πολλούς πίνακες

Ακολουθούν παραδείγματα ερωτημάτων με στόχο την εξάσκηση στη σύνταξη ερωτημάτων που εμπλέκουν πολλούς πίνακες.

- *Ερώτημα 8*. Βρες τα ΑΦΜ των πελατών με κινητά τηλέφωνα.
  Το ερώτημα απαιτεί τον πίνακα Customer για να επιστραφεί το ΑΦΜ και τον πίνακα Phones για τον έλεγχο του τηλεφωνικού αριθμού. Για να απαντηθεί το ερώτημα στην ουσία θα ήθελα να επεκτείνω τον πίνακα Phones με τα αντίστοιχα στοιχεία των πελατών που αφορούν τους συγκεκριμένους τηλεφωνικούς αριθμούς. Για το λόγο αυτό θα κάνουμε καρτεσιανό του Customer με το Phones. Όπως φαίνεται και από το παράδειγμα του ερωτήματος 7, που αναπαράγεται εδώ για ευκολία, αφού στο καρτεσιανό παίρνουμε όλους τους συνδυασμούς θα πάρουμε και συνδυασμούς που αντιστοιχούν ένα τηλεφωνικό νούμερο με τα στοιχεία λάθος πελάτη.

cid	**afm**	**address**	**name**	**sname**	**dateOfBirth**	**cid**	**pnum**
1	*077783234*	*56 Baltetsiou st.*	*Kostas*	*Kostantinou*	*1990-10-30*	*1*	*2231011111*
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	1	2231011111
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	1	2231011111
1	*077783234*	*56 Baltetsiou st.*	*Kostas*	*Kostantinou*	*1990-10-30*	*1*	*6944444444*
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	1	6944444444
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	1	6944444444
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	2	2103333333
2	*175783239*	*107 Diakou st.*	*Eleni*	*Kostantinou*	*1985-11-02*	*2*	*2103333333*
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	2	2103333333

  Για να κρατήσουμε μόνο τους συνδυασμούς στους οποίους υπάρχει αντιστοιχία μεταξύ των στοιχείων του πελάτη και του τηλεφωνικού αριθμού (εγγραφές μαρκαρισμένες με bold και italics στο παράδειγμα) θα ζητήσουμε να ισχύει στο WHERE η εξής συνθήκη επιπλέον αυτής που αφορά τα κινητά τηλέφωνα: Customer.cid=Phones.cid.

```
SELECT afm
FROM Customer, Phones
WHERE Customer.cid=Phones.cid AND pnum LIKE '6%';
```

  Να σημειωθεί ότι ερωτήματα που απαιτούν καρτεσιανό γινόμενο και κάποιας μορφής συσχέτιση ανάμεσα σε ένα ή περισσότερα κοινά πεδία είναι τόσο συνηθισμένα που για το λόγο αυτό εισήχθηκε η πράξη της συνένωσης (JOIN) πινάκων για την οποία θα μιλήσουμε εκτενώς σε επόμενο κεφάλαιο. Τα κοινά πεδία θα λέμε ότι είναι τα πεδία στα οποία γίνεται η συνένωση των πινάκων.

- *Ερώτημα 9*. Βρες τα ΑΦΜ των πελατών που έχουν λογαριασμούς με υπόλοιπο μεγαλύτερο των 10000.
  Το ερώτημα απαιτεί τους πίνακες Customer για το ΑΦΜ και τον Account για το υπόλοιπο του λογαριασμού. Αν συνδυάσουμε αυτούς τους πίνακες μόνο είναι βέβαιο ότι δε θα μπορέσουμε να απαντήσουμε στο ερώτημα καθώς λείπει η πληροφορία σχετικά με το ποιος

πελάτης έχει ποιο λογαριασμό. Η πληροφορία αυτή υπάρχει στον πίνακα Owns. Κατά συνέπεια θα πάρουμε το καρτεσιανό γινόμενο και των τριών πινάκων παραθέτοντας και τους τρεις πίνακες χωρισμένους με κόμμα στο FROM. Η σειρά παράθεσης παίζει ρόλο μόνο στη σειρά εμφάνισης των πεδίων του αποτελέσματος (αν δεν έχουμε άλλη επιλογή στο SELECT) καθώς και στη σειρά με την οποία θα εμφανιστούν οι εγγραφές στο αποτέλεσμα. Κατά τα άλλα η σειρά με την οποία εκτελούνται τα καρτεσιανά γινόμενα δεν παίζει κανένα ρόλο. Υποθέστε ότι ο Customer έχει τις εγγραφές του ερωτήματος 7 και οι πίνακες Account και Owns έχουν ως εξής:

Account

accid	balance	dateOfCreation
100	10000	2014-01-01
200	40000	2013-02-01
300	30000	2005-03-10

Owns

cid	accid
1	100
2	200
1	300

Το καρτεσιανό γινόμενο των 3 πινάκων Customer, Owns, Account έχει ως εξής (αναγράφεται μόνο το cid, afm από τον Customer για εξοικονόμηση χώρου):

cid	afm	cid	accid	accid	balance	dateOfCreation
1	*077783234*	*1*	*100*	*100*	*10000*	*2014-01-01*
1	077783234	2	200	100	10000	2014-01-01
1	077783234	1	300	100	10000	2014-01-01
1	077783234	1	100	200	40000	2013-02-01
1	077783234	2	200	200	40000	2013-02-01
1	077783234	1	300	200	40000	2013-02-01
1	077783234	1	100	300	30000	2005-03-10
1	077783234	2	200	300	30000	2005-03-10
1	*077783234*	*1*	*300*	*300*	*30000*	*2005-03-10*
2	175783239	1	100	100	10000	2014-01-01
2	175783239	2	200	100	10000	2014-01-01
2	175783239	1	300	100	10000	2014-01-01
2	175783239	1	100	200	40000	2013-02-01
2	*175783239*	*2*	*200*	*200*	*40000*	*2013-02-01*
2	175783239	1	300	200	40000	2013-02-01
2	175783239	1	100	300	30000	2005-03-10
2	175783239	2	200	300	30000	2005-03-10
2	175783239	1	300	300	30000	2005-03-10
3	095111139	1	100	100	10000	2014-01-01
3	095111139	2	200	100	10000	2014-01-01
3	095111139	1	300	100	10000	2014-01-01
3	095111139	1	100	200	40000	2013-02-01
3	095111139	2	200	200	40000	2013-02-01
3	095111139	1	300	200	40000	2013-02-01
3	095111139	1	100	300	30000	2005-03-10
3	095111139	2	200	300	30000	2005-03-10
3	095111139	1	300	300	30000	2005-03-10

Αν και στο καρτεσιανό υπάρχουν 27 εγγραφές (3*3*3) μόνο 3 εγγραφές (μαρκαρισμένες με bold και italics) αντιστοιχούν σωστά λογαριασμούς με πελάτες και είναι εκεί που υπάρχει ισότητα τόσο στο κοινό πεδίο των Customer, Owns όσο και στο κοινό πεδίο των Owns και Account. Ως εκ τούτου η σύνταξη του ερωτήματος σε SQL έχει ως εξής:

```
SELECT afm
FROM Customer, Owns, Account
WHERE   Customer.cid=Owns.cid AND
        Owns.accid=Account.accid AND
        balance>10000;
```

- *Ερώτημα 10*. Βρες τα ΑΦΜ των πελατών που έχουν στείλει χρήματα στο λογαριασμό με κωδικό 100.
  Οι κωδικοί των λογαριασμών που απαντούν στο ερώτημα είναι τα accidSource στον πίνακα Transfer για τα οποία το accidDest=100. Από αυτούς τους κωδικούς πρέπει να βρούμε τους πελάτες που σημαίνει καρτεσιανό με το Owns και το Customer όπως στο προηγούμενο ερώτημα μόνο που θα ζητήσουμε accid=accidSource (δε χρειάζεται να προσδιορίσουμε τους πίνακες καθώς τα πεδία αυτά υπάρχουν μόνο στο Owns και στο Transfer).

```
SELECT afm
FROM Customer, Owns, Transfer
WHERE   Customer.cid=Owns.cid AND
        accid=accidSource AND
        accidDest=100;
```

- *Ερώτημα 11*. Βρες τα ΑΦΜ των πελατών που έχουν στείλει χρήματα στο λογαριασμό με κωδικό 100 είτε απευθείας είτε έμμεσα δηλ. έστειλαν σε κάποιον που έστειλε στο λογαριασμό με κωδικό 100.
  Για το απευθείας, η απάντηση βρίσκεται στο προηγούμενο ερώτημα. Για έμμεσα, θα πρέπει στην ουσία να βρω τις εγγραφές t1 του Transfer για τις οποίες υπάρχουν εγγραφές t2 έτσι ώστε το t1[accidDest]=t2[accidSource] και το t2[accidDest]=100. Σε αυτήν την περίπτωση το t1[accidSource] ανήκει στο αποτέλεσμα και συνδυάζεται κατόπιν με Owns και Customer. Για να βρούμε τα παραπάνω t1 ένας απλός τρόπος (εμπνευσμένος από τους πίνακες σε γλώσσες προγραμματισμού) θα ήταν σε ένα nested loop κάθε μία εγγραφή του Transfer να συγκρίνεται με όλες τις υπόλοιπες. Κάτι τέτοιο δεν μπορεί να γίνει με χρήση απλών SQL ερωτημάτων της μορφής SELECT FROM WHERE. Η MySQL υποστηρίζει stored procedures και cursors (περισσότερα για αυτά σε επόμενο κεφάλαιο) βάσει των οποίων θα μπορούσε να υλοποιηθεί το ερώτημα με τον παραπάνω τρόπο. Υπάρχει όμως και άλλη εναλλακτική. Έχουμε ήδη δει ότι για να συγκρίνω εγγραφές που ανήκουν σε διαφορετικούς πίνακες μπορώ να κάνω καρτεσιανό και να εφαρμόσω κατάλληλα κριτήρια στο WHERE. Το αντίστοιχο ισχύει και για να συγκρίνω εγγραφές του ίδιου πίνακα, δηλ. κάνω καρτεσιανό γινόμενο του πίνακα με τον εαυτό του (SELF JOIN). Στο καρτεσιανό γινόμενο κάθε εγγραφή του πίνακα εμφανίζεται μαζί με όλες τις εγγραφές του πίνακα συμπεριλαμβανομένης αυτής. Κατά συνέπεια μπορώ να εφαρμόσω τα προαναφερθέντα κριτήριο και να εξάγω τους κωδικούς που ανήκουν στο αποτέλεσμα ως εξής:

  ```
  SELECT T1.accid
  FROM Transfer T1, Transfer T2
  WHERE T1.accidDest=T2.accidSource AND T2.accidDest=100;
  ```

  Εφόσον αυτή query μου επιστρέφει τους σωστούς λογαριασμούς εφαρμόζοντας καρτεσιανό γινόμενο με το Owns και το Customer παίρνω τους πελάτες που έστειλα έμμεσα χρήματα. Το τελικό αποτέλεσμα θα παρθεί από την ένωση αυτών με αυτούς που έστειλαν άμεσα.

  ```
  (SELECT afm
  FROM Customer, Owns, Transfer T1, Transfer T2
  ```

```
WHERE   Customer.cid=Owns.cid AND
        accid=T1.accidSource AND
        T1.accidDest=T2.accidSource AND
        T2.accidDest=100)
UNION
(SELECT afm
FROM Customer, Owns, Transfer
WHERE   Customer.cid=Owns.cid AND
        accid=accidSource AND
        accidDest=100);
```

Αν θέλαμε να βρούμε αυτούς που έστειλαν έμμεσα με δύο βήματα δηλαδή αυτούς που έστειλαν σε έναν λογαριασμό από τον οποίο στάλθηκε σε λογαριασμό που έστειλε στο 100 θα κάναμε 3 φορές καρτεσιανό γινόμενο του Transfer με τον εαυτό του ως εξής:

```
SELECT afm
FROM Customer, Owns, Transfer T1, Transfer T2, Transfer T3
WHERE   Customer.cid=Owns.cid AND
        accid=T1.accidSource AND
        T1.accidDest=T2.accidSource AND
        T2.accidDest=T3.accidSource AND
        T3.accidDest=100;
```

Γενικά μιλώντας, αν μας δοθεί το επιθυμητό «βάθος» της έμμεσης διαδρομής μπορούμε να χτίσουμε αντίστοιχο ερώτημα. Επίσης αν θέλουμε στο τελικό αποτέλεσμα να συμπεριλάβουμε διαφορετικά «βάθη» διαδρομής μπορούμε εφαρμόζοντας UNION στα υποερωτήματα. Αυτό που δεν μπορούμε να εκφράσουμε είναι ερώτημα που ζητά τους πελάτες που έχουν στείλει έμμεσα χρήματα στο λογαριασμό 100, για οποιοδήποτε βάθος έμμεσης διαδρομής. Για να απαντηθούν τέτοιου είδους ερωτήματα, που ονομάζονται ιεραρχικά καθώς προκύπτουν κατά κόρον στο ιεραρχικό μοντέλο και σε XML δεδομένα, υπάρχει το RECURSIVE JOIN που δεν υποστηρίζεται όμως στη MySQL5.7. Η ενασχόληση με τέτοιου είδους ερωτήματα και οι τρόποι οργάνωσης ιεραρχικών δεδομένων στη MySQL5.7 αποτελούν πιο προχωρημένα αντικείμενα μελέτης και ξεφεύγουν από τους σκοπούς του παρόντος συγγράμματος.

- *Ερώτημα 12*. Βρες τα ΑΦΜ των πελατών που έχουν στείλει χρήματα σε κάποιον από τους λογαριασμούς του πελάτη με ΑΦΜ 077777777.
  Όμοιο με το προηγούμενο ερώτημα μόνο που τώρα δεν έχουμε συγκεκριμένο accidDest για να αναζητήσουμε, αλλά πελάτη του οποίου οι λογαριασμοί παίζουν το ρόλο του προορισμού. Επομένως εκτός από το καρτεσιανό γινόμενο του Customer, Owns, Transfer ως προς τα cid και accid=accidSource, θέλουμε και τα αντίστοιχα για το accidDest.

```
SELECT C1.afm
FROM Customer C1, Owns O1, Transfer, Owns O2, Customer C2
WHERE   C1.cid=O1.cid AND
        O1.accid=accidSource AND
        accidDest=O2.accid AND
        O2.cid=C2.cid AND
        C2.afm='077777777';
```

- *Ερώτημα 13*. Βρες τους κωδικούς των λογαριασμών με τα μεγαλύτερα υπόλοιπα.
  Για να βρούμε τα accid που θέλουμε πρέπει να συγκρίνουμε εγγραφές του Account, επομένως και σύμφωνα με Ερωτήματα 11 και 12 θα κάνουμε καρτεσιανό του Account με το Account. Έστω A1 και A2 οι ονομασίες των δύο Account στο FROM και ας υποθέσουμε ότι επιστρέφουμε το A1.accid. Το ερώτημα είναι τι να ζητήσουμε στη σύγκριση. Αν ζητήσουμε A1.balance>A2.balance τότε για να επιστραφεί ένα A1.accid θα πρέπει να υπάρχει τουλάχιστον μία υποεγγραφή προερχόμενη από τον A2 με μικρότερο balance. Κατά

συνέπεια με τον τρόπο αυτό θα επιστρέψουμε τα accid για τα οποία υπάρχει κάποιος λογαριασμός με μικρότερο από αυτά balance. Αλλάζοντας τη φορά της σύγκρισης και ζητώντας A1.balance<A2.balance για να επιστραφεί ένα A1.accid θα πρέπει να υπάρχει τουλάχιστον μία υποεγγραφή προερχόμενη από τον A2 με μεγαλύτερο balance. Επομένως επιστρέφονται όσα δεν είναι τα μεγαλύτερα. Έχοντας τα accid που δεν έχουν το μεγαλύτερο balance μπορώ να τα αφαιρέσω από τα συνολικά accid του Account και να πάρω αυτά που έχουν το μεγαλύτερο balance. Η παραπάνω είναι βασική τεχνική σχεδιασμού ερωτημάτων δηλ. αν δεν μπορούμε να βρούμε απευθείας τις εγγραφές που απαντούν στο ερώτημα, ίσως μπορούμε να βρούμε εύκολα τις εγγραφές που δεν ανήκουν στη λύση οπότε με αφαίρεση παίρνουμε το ζητούμενο. Το πώς μπορούμε να κάνουμε αφαίρεση στη MySQL θα εξηγηθεί σε επόμενο κεφάλαιο. Στο ερώτημα εδώ θα χρησιμοποιηθεί το EXCEPT.

```
(SELECT accid
FROM Account)
EXCEPT
(SELECT A1.accid
FROM Account A1, Account A2
WHERE A1.balance<A2.balance);
```

Η SQL προδιαγράφει διάφορες συναρτήσεις συνάθροισης (aggregator functions) που υποστηρίζονται από την MySQL5.7. Μία από αυτές, η MAX(), θα μπορούσε να χρησιμοποιηθεί εδώ για να απαντηθεί το ερώτημα πιο εύκολα. Οι συναρτήσεις συνάθροισης εξετάζονται σε επόμενο κεφάλαιο.

6.2 Παράδειγμα Εργαστηριακής Άσκησης

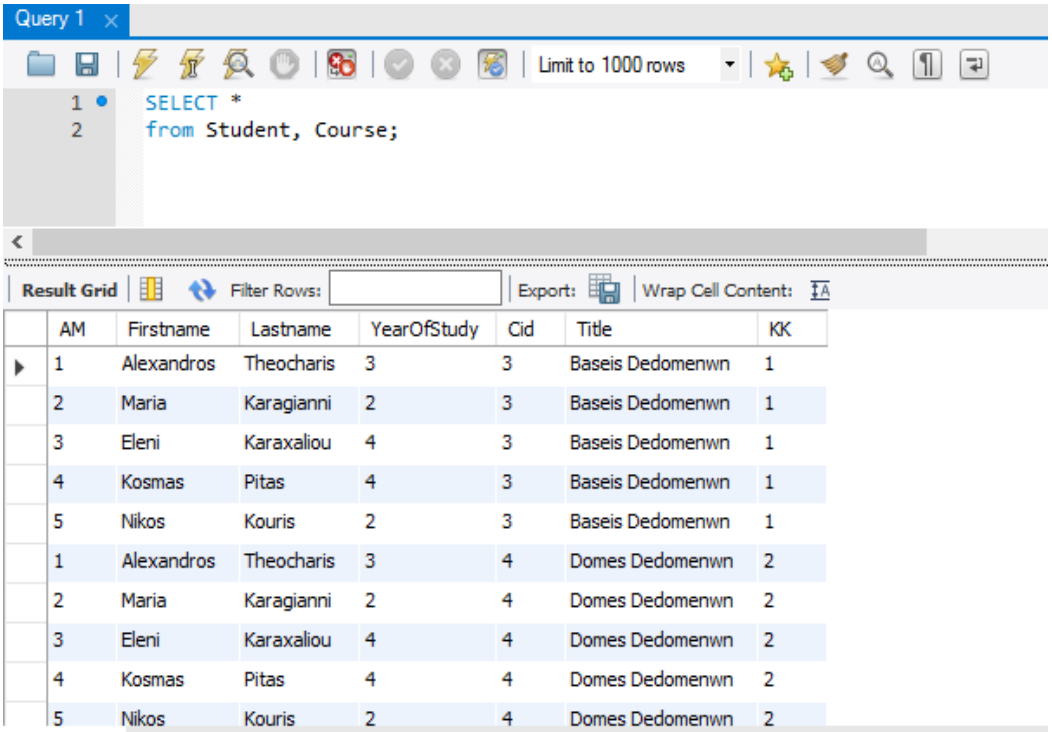
Εκφώνηση:	Στη Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 23. Δημιουργήστε το καρτεσιανό γινόμενο των φοιτητών με τα μαθήματα. 24. Βρείτε τα ΑΜ των φοιτητών που έχουν κινητό τηλέφωνο ή το όνομά τους είναι Γιώργος. 25. Βρείτε τους φοιτητές που πέρασαν το μάθημα Δομές Δεδομένων το 2014 ή το 2015. 26. Βρείτε τους καθηγητές που διδάσκουν το μάθημα Προγραμματισμός Ι ή Προγραμματισμός ΙΙ.
Ζητούμενα:	Καρτεσιανό γινόμενο, ένωση, τομή

Ζητούμενο 1: Δημιουργήστε το καρτεσιανό γινόμενο των φοιτητών με τα μαθήματα.

Δίνουμε:

```
SELECT *
FROM Student, Course;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 6.1 όπου παρατηρούμε ότι έχουμε όλους τους συνδυασμούς των εγγραφών των πινάκων.



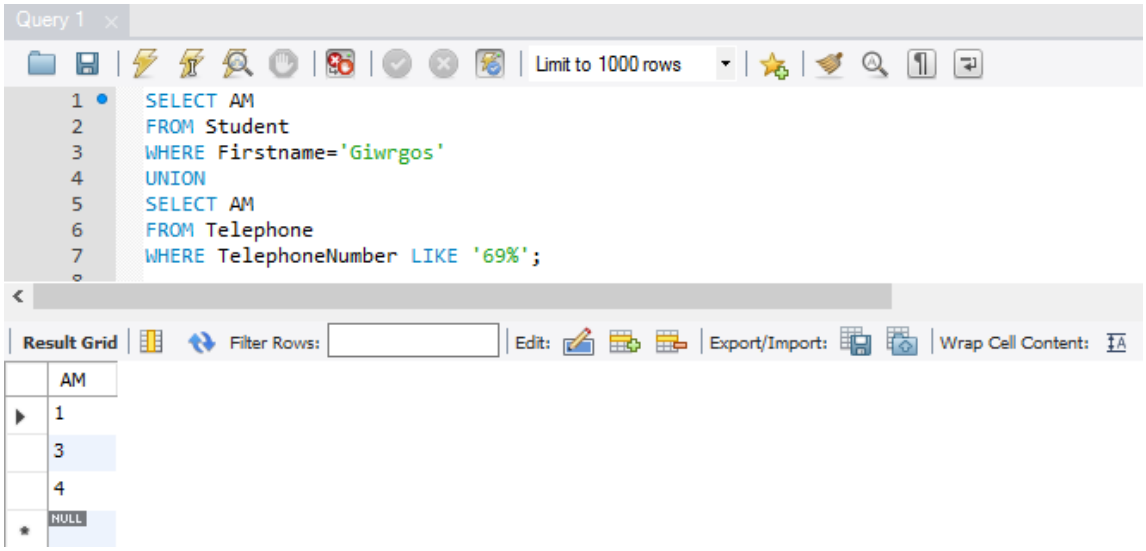
AM	Firstname	Lastname	YearOfStudy	Cid	Title	KK
1	Alexandros	Theocharis	3	3	Baseis Dedomenwn	1
2	Maria	Karagianni	2	3	Baseis Dedomenwn	1
3	Eleni	Karaxaliou	4	3	Baseis Dedomenwn	1
4	Kosmas	Pitas	4	3	Baseis Dedomenwn	1
5	Nikos	Kouris	2	3	Baseis Dedomenwn	1
1	Alexandros	Theocharis	3	4	Domes Dedomenwn	2
2	Maria	Karagianni	2	4	Domes Dedomenwn	2
3	Eleni	Karaxaliou	4	4	Domes Dedomenwn	2
4	Kosmas	Pitas	4	4	Domes Dedomenwn	2
5	Nikos	Kouris	2	4	Domes Dedomenwn	2

Εικόνα 6.1: *Καρτεσιανό γινόμενο.*

Ζητούμενο 2: Βρείτε τα ΑΜ των φοιτητών που έχουν κινητό τηλέφωνο ή το όνομά τους είναι Γιώργος.

```
SELECT AM
FROM Student
WHERE Firstname=’Giwrgos’
UNION
SELECT AM
FROM Telephone
WHERE TelephoneNumber LIKE ‘69%’;
```

Παράδειγμα αποτελέσματος βλέπουμε στην Εικόνα 6.2 όπου δεν έχουμε φοιτητή που να λέγεται Γιώργος, αλλά έχουμε τρεις φοιτητές με κινητό τηλέφωνο.

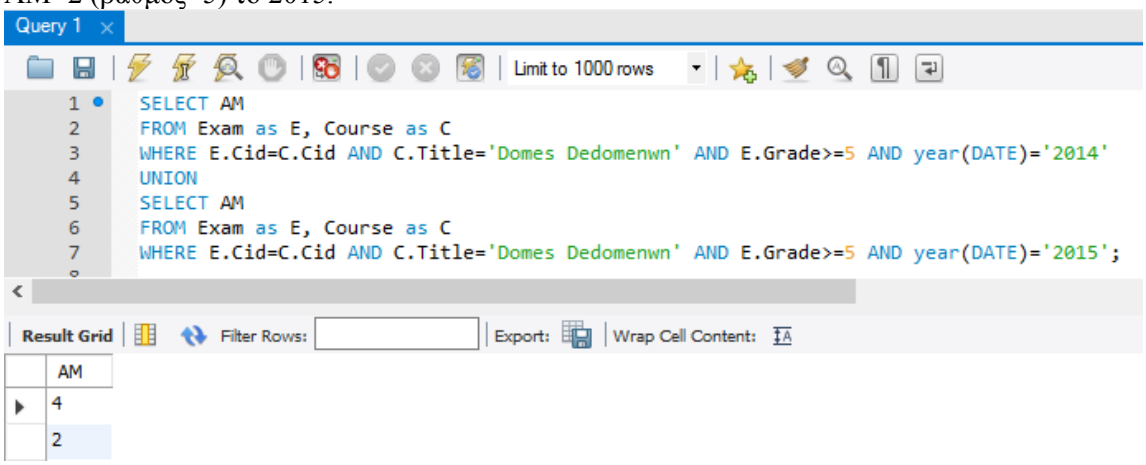


Εικόνα 6.2: *Η εντολή UNION.*

Ζητούμενο 3: Βρείτε τους φοιτητές που πέρασαν το μάθημα Δομές Δεδομένων το 2014 ή το 2015.

SELECT AM
FROM Exam as E, Course as C
WHERE E.Cid=C.Cid AND C.Title=’Domes Dedomenwn’ AND E.Grade>=5 AND year(DATE)=’2014’
UNION
SELECT AM
FROM Exam as E, Course as C
WHERE E.Cid=C.Cid AND C.Title=’Domes Dedomenwn’ AND E.Grade>=5 AND year(DATE)=’2015’;

Παράδειγμα αποτελέσματος του ερωτήματος βλέπουμε στην Εικόνα 6.3, όπου το μάθημα Δομές δεδομένων το έχει περάσει ο φοιτητής με ΑΜ=4 (βαθμός=8) το έτος 2014 και ο φοιτητής με ΑΜ=2 (βαθμός=5) το 2015.



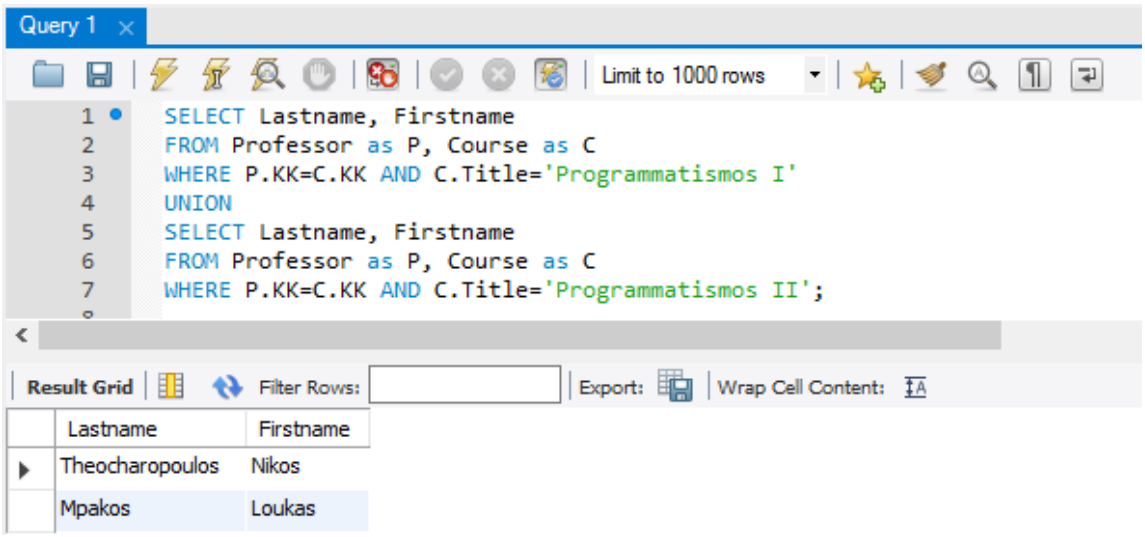
Εικόνα 6.3: *Η εντολή UNION.*

Ζητούμενο 4: Βρείτε τους καθηγητές που διδάσκουν το μάθημα Προγραμματισμός Ι ή Προγραμματισμός ΙΙ.

SELECT Lastname, Firstname
FROM Professor as P, Course as C

```
WHERE P.KK=C.KK AND C.Title=’Programmatismos I’
UNION
SELECT Lastname, Firstname
FROM Professor as P, Course as C
WHERE P.KK=C.KK AND C.Title=’Programmatismos II’;
```

Παράδειγμα αποτελέσματος του ερωτήματος βλέπουμε στην Εικόνα 6.4, όπου επιστρέφει τους καθηγητές που διδάσκουν τα μαθήματα Προγραμματισμός I και II. Στη συγκεκριμένη περίπτωση είναι δύο καθηγητές.



Εικόνα 6.4: *Η εντολή UNION.*

6.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

1. Φτιάξτε μια κατάσταση με όλους τους εργαζόμενους που να συνοδεύεται από τις εταιρίες.
2. Βρείτε τους εργαζόμενους που απασχολούνταν στην εταιρία Advance ή Info.
3. Δημιουργήστε το καρτεσιανό γινόμενο των εργαζομένων με τις εταιρίες.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2), απαντήστε στα ακόλουθα ερωτήματα:

1. Δημιουργήστε το καρτεσιανό γινόμενο των ταινιών με τους θεατές.
2. Βρείτε τις ταινίες που σκηνοθέτησε ο Steven Spielberg ή ο Quentin Tarantino.
3. Φτιάξτε μια κατάσταση με τους θεατές και τις ταινίες.

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3), απαντήστε στα ακόλουθα ερωτήματα:

1. Δημιουργήστε το καρτεσιανό γινόμενο των τουρνουά και των αποτελεσμάτων.
2. Φτιάξτε μια λίστα με όλα τα τουρνουά με τους νικητές τους.
3. Βρείτε τα τουρνουά που νίκησε ο Federer το 2014 ή το 2015.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαια 8 και 9.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαια 5
JOIN. https://dev.mysql.com/doc/refman/5.7/en/join.html
UNION. https://dev.mysql.com/doc/refman/5.7/en/union.html
INTERSECT.
http://www.tutorialspoint.com/sql/sql-intersect-clause.htm
http://www.techonthenet.com/sql/intersect.php
EXCEPT.
http://www.tutorialspoint.com/sql/sql-except-clause.htm
http://www.techonthenet.com/sql/except.php
MINUS. http://www.techonthenet.com/sql/minus.php

Κεφάλαιο 7 Η Πράξη της Συνένωσης

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθεί η χρήση εσωτερικών και εξωτερικών συνενώσεων(join, η συνένωση που μεταφράζεται σε άλλα συγγράμματα ως σύζευξη) στην απάντηση ερωτημάτων που συνδυάζουν πολλαπλούς πίνακες.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- *Βασικές πράξεις θεωρίας συνόλων (Κεφ.6).*

.

7.1 Εισαγωγικές Έννοιες

Η πράξη του καρτεσιανού γινομένου δύο πινάκων T1 και T2 και η συνακόλουθη εφαρμογή στο WHERE κριτηρίων της μορφής aθb, όπου το a είναι πεδίο του T1, το b πεδίο του T2 και το θ κάποιος τελεστής σύγκρισης, είναι τόσο συχνή που αναφερόμαστε ξεχωριστά σε αυτή ως συνένωση-θ (θ-join) των πινάκων T1 και T2. Θεωρήστε τους ακόλουθους πίνακες Customer και Phones:

Customer

cid	**afm**	**address**	**name**	**sname**	**dateOfBirth**
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333

Το παρακάτω ερώτημα:
SELECT afm
FROM Customer, Phones
WHERE Customer.cid=Phones.cid AND Phones.pnum LIKE '6%';
βρίσκει τα ΑΦΜ των πελατών που έχουν κινητά τηλέφωνα. Ο τρόπος σύνταξης του ερωτήματος υπονοεί ότι πρώτα θα γίνει το καρτεσιανό γινόμενο των δύο πινάκων και στη συνέχεια θα εφαρμοστούν τα κριτήρια του WHERE. Με άλλα λόγια πρώτα θα δημιουργηθούν οι 9 εγγραφές του καρτεσιανού και στη συνέχεια θα επιλεγούν αυτές που πληρούν το WHERE. Αντίθετα η χρήση της συνένωσης-θ στην παραπάνω περίπτωση θα υπονοούσε ότι δε χρειάζεται να παραχθεί πρώτα το καρτεσιανό γινόμενο αλλά το μέρος αυτού που πληροί τη συνθήκη Customer.cid=Phones.cid (3 εγγραφές) και στη συνέχεια οι εγγραφές αυτές θα ελέγχονταν ως προς τη συνθήκη Phones.pnum LIKE '6%'.

Όταν η συνθήκη θ σε ένα θ-join είναι ισότητα τότε θα μιλάμε για ισοσυνένωση (equijoin). Όταν έχω ισοσυνένωση σε όλα τα κοινά πεδία των δύο πινάκων τότε μιλάμε για φυσική ισοσυνένωση (natural join). Ξαναγυρνώντας στο προηγούμενο παράδειγμα, εφαρμόζοντας θ-join στους πίνακες Customer και Phones, στην ουσία εφαρμόζουμε equijoin, ενώ το ερώτημα θα μπορούσε να απαντηθεί και με natural join των δύο πινάκων.

Σημείωση 1: Όλα τα DBMS τρέχουν ειδικούς αλγορίθμους με στόχο τη γρήγορη εκτέλεση joins. Ιδιαίτερη έμφαση δίνεται στη γρήγορη εκτέλεση των equijoins που είναι και η πιο συχνή περίπτωση σε ερωτήματα. Ιδιαίτερα όταν έχω equijoin σε πεδίο για το οποίο υπάρχει ευρετήριο (index), πχ., όταν το πεδίο είναι primary key σε κάποιον πίνακα ο υπολογισμός είναι πολύ γρήγορος.
Σημείωση 2: Τα περισσότερα DBMS έχουν query optimizer που αποφασίζει για το βέλτιστο τρόπο εκτέλεσης ενός ερωτήματος. Στο παράδειγμα αν και ζητάμε καρτεσιανό γινόμενο των Customer και Phones (comma join) ο optimizer της MySQL5.7 θα το εκτελέσει σαν equijoin στο πεδίο cid. Αυτό δε σημαίνει ότι πάντα ο optimizer βρίσκει το καλύτερο πλάνο εκτέλεσης. Καλό είναι εφεξής για λόγους αναγνωσιμότητας και ευκολίας στη συντήρηση αλλά και για να βοηθούμε τον optimizer στις αποφάσεις του να μη χρησιμοποιούμε comma joins ή αλλιώς cross joins όταν αυτό που θέλουμε ουσιαστικά είναι θ-join. Η χρήση τους θα περιορίζεται μόνο στις περιπτώσεις που θέλουμε καρτεσιανό γινόμενο. Σαν αντιπαράδειγμα θεωρείστε την περίπτωση που έχω μια πολύπλοκη query με 10 ή 20 πίνακες η οποία πρέπει να ελεγχθεί/αλλαχθεί. Ο χρόνος που θα καταναλώσει ο προγραμματιστής για να κατανοήσει από το WHERE σε ποια πεδία συνενώνονται οι πίνακες δε θα είναι αμελητέος.

7.1.1 Συνενώσεις στην SQL και MySQL5.7

Η βασική δομή εντολής για να κάνουμε συνένωση δύο πινάκων είναι:
πίνακας1 [INNER | CROSS] JOIN *πίνακας2* [*join_συνθήκη*]
όπου στη *join_συνθήκη* μπορούμε να έχουμε τις ακόλουθες δύο επιλογές:
ON *λογική_συνθήκη*: κατά τα πρότυπα του WHERE.
USING (*λίστα_πεδίων*): ισότητα στα πεδία της λίστας με την προϋπόθεση να είναι κοινά και για τους δύο πίνακες.

Παραδείγματα
Οι ακόλουθες queries βρίσκουν τα afm των πελατών που έχουν κινητό τηλέφωνο.
(I) Με comma join.

```
SELECT afm
FROM Customer, Phones
WHERE Customer.cid=Phones.cid AND Phones.pnum LIKE '6%';
```

(II) Με INNER JOIN.

```
SELECT afm
FROM Customer INNER JOIN Phones ON Customer.cid=Phones.cid
WHERE Phones.pnum LIKE '6%';
```

(III) Με JOIN (ισοδύναμο του INNER JOIN)

```
SELECT afm
FROM Customer JOIN Phones ON Customer.cid=Phones.cid
WHERE Phones.pnum LIKE '6%';
```

(IV) Με CROSS JOIN

```
SELECT afm
FROM Customer CROSS JOIN Phones ON Customer.cid=Phones.cid
WHERE Phones.pnum LIKE '6%';
```

(V) INNER JOIN χρησιμοποιώντας το USING αντί του ON.

```
SELECT afm
FROM Customer INNER JOIN Phones USING (cid)
WHERE Phones.pnum LIKE '6%';
```

(VI) Ισοδύναμη με την (I)
```
SELECT afm
FROM Customer CROSS JOIN Phones
WHERE Customer.cid=Phones.cid AND Phones.pnum LIKE '6%';
```

(VII) INNER JOIN χωρίς WHERE
```
SELECT afm
FROM
Customer INNER JOIN Phones ON
Customer.cid=Phones.cid AND
Phones.pnum LIKE '6%';
```

Στη MySQL JOIN, INNER JOIN και CROSS JOIN είναι ισοδύναμα και πραγματοποιούν αυτό που ονομάζεται εσωτερική συνένωση πινάκων. Περισσότερα για τις διαφορές μεταξύ εσωτερικών και εξωτερικών συνενώσεων στην επόμενη υποενότητα. Χρησιμοποιώντας το ON μπορώ να γράψω οποιαδήποτε συνθήκη επομένως μπορώ να γράψω οποιοδήποτε θ-join ανεξαρτήτως αν το θ είναι ισότητα ή όχι. Με το USING μπορώ να γράψω μόνο equijoins. Η διαφορά μεταξύ του USING και του ON με ισότητα είναι ότι το πρώτο γυρνάει τις στήλες που περιγράφονται στη λίστα του μία φορά ενώ το δεύτερο δύο φορές μία για κάθε πίνακα. Για παράδειγμα αν στις queries (II) και (V) είχαμε αντί για SELECT afm, SELECT * τα πεδία που θα γυρνούσαν θα ήταν τα εξής:
(II) cid, afm, address, name, sname, dateOfBirth, cid, pnum
(V) cid, afm, address, name, sname, dateOfBirth, pnum

Στο πρότυπο της SQL η χρήση του CROSS JOIN περιορίζεται στον υπολογισμό του καρτεσιανού γινομένου και ως εκ τούτου δε δέχεται επιπλέον όρισμα ON ή USING. Στη MySQL όπως αναφέρθηκε το CROSS JOIN είναι ισοδύναμο με τα JOIN και INNER JOIN και ως εκ τούτου μπορεί να ακολουθείται από ON ή USING. Αν ένα JOIN, INNER JOIN ή CROSS JOIN δεν ακολουθείται από ON ή USING και στο WHERE δεν υπάρχει κριτήριο συνένωσης (Customer.cid=Phones.cid στο παράδειγμα) τότε επιστρέφεται το καρτεσιανό γινόμενο. Για λόγους αναγνωσιμότητας καλό είναι να χρησιμοποιείται το INNER JOIN όταν θέλουμε εσωτερική συνένωση με τα κριτήρια συνένωσης στο ON ή στο USING, αναλόγως αν θέλουμε να κρατήσουμε ή όχι σε ένα equijoin διπλές στήλες. Ο τρόπος γραφής της query (VII) θα πρέπει να αποφεύγεται καθώς πέραν της μειωμένης αναγνωσιμότητας (πιο δύσκολα καταλαβαίνει κάποιος ποιο είναι το κριτήριο συνένωσης), εισάγει αχρείαστα επιπλέον πολυπλοκότητα στον query optimizer. Τέλος για λόγους αναγνωσιμότητας καλό είναι να μη χρησιμοποιούμε CROSS JOIN όταν δε θέλουμε το καρτεσιανό γινόμενο αλλά INNER JOIN.

7.1.2 Εξωτερικές συνενώσεις στην SQL και MySQL5.7

Ας υποθέσουμε ότι στο Phones δεν είχαμε ορίσει ως ξένο κλειδί το cid και είχαμε το ακόλουθο στιγμιότυπο για τους πίνακες Customer και Phones:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333
4	6999999999

Έστω ότι θέλουμε να εμφανίσουμε τα στοιχεία των πελατών μαζί με τα τηλέφωνά τους (αν έχουν). Το ερώτημα δεν μπορεί να απαντηθεί χρησιμοποιώντας εσωτερική συνένωση όπως στην προηγούμενη υποενότητα καθώς με αυτό τον τρόπο δε θα επιστραφούν τα στοιχεία του πελάτη με cid=3 καθώς δε συνδυάζεται με καμία από τις εγγραφές στο Phones. Κάτι αντίστοιχο θα προέκυπτε εάν ζητούσαμε τα τηλέφωνα μαζί με στοιχεία για τους πελάτες που ανήκουν (αν υπάρχουν ιδιοκτήτες). Για να απαντηθούν τέτοια ερωτήματα χρησιμοποιούνται εξωτερικές συνενώσεις. Στην SQL περιγράφονται τριών ειδών εξωτερικές συνενώσεις: αριστερή (LEFT), δεξιά (RIGHT) και πλήρης (FULL). Η σύνταξη είναι αντίστοιχη της εσωτερικής συνένωσης: *πίνακας1* {LEFT | RIGHT | FULL} [OUTER] JOIN *πίνακας2 join_συνθήκη*. Η αναγραφή του OUTER είναι προαιρετική και στην πράξη πολλές φορές παραλείπεται. Οι κανόνες για τα ON και USING όσον αφορά τις επιστρεφόμενες στήλες είναι ίδιοι όπως περιγράφηκαν στην προηγούμενη υποενότητα.

Παραδείγματα

(I)

```
SELECT *
FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid;
```

Επιστρέφει:

cid	afm	address	name	sname	dateOfBirth	cid	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2	2103333333
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	NULL	NULL

Όπως παρατηρούμε η εγγραφή με cid=3 του πίνακα Customer που δε συνδυάζεται με καμία εγγραφή από τον πίνακα Phones βρίσκεται στο αποτέλεσμα του LEFT JOIN, με NULL στα πεδία του Phones.

```
SELECT *
FROM Customer LEFT JOIN Phones USING(cid);
```

Επιστρέφει:

cid	afm	address	name	sname	dateOfBirth	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2103333333
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	NULL

(II)

```
SELECT *
FROM Customer RIGHT JOIN Phones ON Customer.cid=Phones.cid;
```

Επιστρέφει:

cid	afm	address	name	sname	dateOfBirth	cid	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2	2103333333
NULL	NULL	NULL	NULL	NULL	NULL	4	6999999999

SELECT *
FROM Customer RIGHT JOIN Phones USING(cid);

Επιστρέφει:

cid	pnum	afm	address	name	sname	dateOfBirth
1	2231011111	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
1	6944444444	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	2103333333	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
4	6999999999	NULL	NULL	NULL	NULL	NULL

Στη MySQL5.7 όταν κάνουμε RIGHT JOIN με χρήση του USING επιστρέφεται το ισοδύναμο αποτέλεσμα με αυτό που θα είχαμε εάν γράφαμε τους πίνακες με ανάποδη σειρά και κάναμε LEFT JOIN.

(III)
SELECT *
FROM Customer FULL JOIN Phones ON Customer.cid=Phones.cid;
Επιστρέφει:

cid	afm	address	name	sname	dateOfBirth	cid	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	1	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2	2103333333
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	4	6999999999

Παρατηρούμε ότι τόσο οι εγγραφές του Customer που δε συνδυάζονται, όσο και οι αντίστοιχες εγγραφές του Phones εμφανίζονται στο αποτέλεσμα με NULL στα πεδία του άλλου πίνακα.

SELECT *
FROM Customer FULL JOIN Phones USING(cid);
Επιστρέφει:

cid	afm	address	name	sname	dateOfBirth	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2103333333
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	NULL
4	NULL	NULL	NULL	NULL	NULL	6999999999

Όπως και στο LEFT/RIGHT JOIN USING έτσι και στο FULL εμφανίζεται η στήλη cid μία φορά. Ο κανόνας είναι ότι αν οι δύο στήλες που αντιστοιχούν στο πεδίο του join έχουν τιμή στη στήλη του αποτελέσματος κρατιέται η τιμή, πχ. στην πρώτη εγγραφή Customer.cid=Phones.cid=1. Αν και οι δύο στήλες είναι NULL, η στήλη του αποτελέσματος παίρνει τιμή NULL (δεν υπάρχει στο παραπάνω παράδειγμα παρόμοια περίπτωση). Αν η μία στήλη είναι NULL και η άλλη έχει τιμή τότε κρατιέται η τιμή. Για παράδειγμα η εγγραφή με cid

3 του Customer στην αντίστοιχή στήλη Phones.cid έχει NULL για το λόγο αυτό στο τελικό cid του αποτελέσματος υπάρχει η τιμή 3. Ομοίως για την εγγραφή με Phones.cid=4.

Σημείωση 1: Η ύπαρξη *join_συνθήκης* (ON ή USING) στη MySQL5.7, όπως συζητήθηκε, είναι προαιρετική για τις πράξεις JOIN, INNER JOIN, CROSS JOIN, στην οποία περίπτωση επιστρέφεται το καρτεσιανό γινόμενο (εκτός και υπάρχει αντίστοιχη της join_συνθήκης στο WHERE. Στις εξωτερικές συνενώσεις όμως είναι υποχρεωτική. Για παράδειγμα το:
SELECT *
FROM Customer LEFT JOIN Phones;
οδηγεί σε συντακτικό σφάλμα.
Σημείωση 2: Η MySQL5.7 δεν υποστηρίζει την πράξη του FULL OUTER JOIN. Υπάρχει τρόπος να υλοποιηθεί μέσω άλλων πράξεων όπως θα εξηγηθεί στη συνέχεια.
Σημείωση 3: Καλό είναι για λόγους αναγνωσιμότητας να μη γράφουμε queries που έχουν και LEFT και RIGHT JOIN. Καλό είναι οι queries μας να γράφονται ή δυνατόν με LEFT JOIN.
Σημείωση 4: Χρειάζεται προσοχή όταν πραγματοποιούμε INNER JOIN σε πεδίο που περιέχει NULL τιμές. Αν γραφτεί *join_συνθήκη* και δε γραφτεί στο WHERE τότε οι σύγκριση μεταξύ NULL και τιμής θα είναι UNKNOWN ή FALSE (αναλόγως του τελεστή σύγκρισης) και ως εκ τούτου δε θα επιστραφεί η εγγραφή. Η κατάσταση αυτή εύκολα προκύπτει όταν χρησιμοποιούμε το αποτέλεσμα μιας εξωτερικής συνένωσης για να πραγματοποιήσουμε εσωτερική συνένωση.

7.1.3 NATURAL JOIN και STRAIGHT_JOIN

Οι φυσικές συνενώσεις είναι συνενώσεις σε όλα τα κοινά πεδία των πινάκων. Τα κοινά πεδία αναγράφονται μία φορά το καθένα στο αποτέλεσμα. Η MySQL5.7 υποστηρίζει την πράξη NATURAL JOIN τόσο για εξωτερικές όσο και για εσωτερικές συνενώσεις. Ακολουθούν αντίστοιχα παραδείγματα χρήσης.
Παραδείγματα:
(I)
SELECT *
FROM Customer NATURAL JOIN Phones;
Φυσική εσωτερική συνένωση. Ισοδύναμο με:
SELECT *
FROM Customer INNER JOIN Phones USING(cid);

(II)
SELECT *
FROM Customer NATURAL LEFT JOIN Phones;
Αριστερή εξωτερική φυσική συνένωση. Ισοδύναμο με:
SELECT *
FROM Customer LEFT JOIN Phones USING(cid);
Αντίστοιχα θα μπορούσε να είχε γραφτεί: NATURAL LEFT OUTER JOIN.

(III)
SELECT *
FROM Customer NATURAL RIGHT JOIN Phones;
Αριστερή εξωτερική φυσική συνένωση. Ισοδύναμο με:
SELECT *
FROM Customer RIGHT JOIN Phones USING(cid);
Αντίστοιχα θα μπορούσε να είχε γραφτεί: NATURAL RIGHT OUTER JOIN.

Η χρήση του NATURAL JOIN καθιστά τον κώδικα δυσκολότερα συντηρήσιμο. Για παράδειγμα θεωρήστε τους ακόλουθους πίνακες:

Customer (cid, afm, address, name, sname, dateOfBirth)
Account (accid, balance, dateOfCreation)
Owns (*cid*, *accid*)
και τις εξής δύο queries:
(A)
```
SELECT Customer.* -- επιστρέφει τα πεδία του Customer
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid;
```
Με -- και κενό μπαίνουν σχολιάζεται γραμμή και είναι αντίστοιχο του // στη C++.
Με /* */ σχολιάζεται οτιδήποτε είναι ανάμεσα όπως στη C.

(B)
```
SELECT Customer.*
FROM Customer NATURAL JOIN Owns NATURAL JOIN Account;
```

Οι δύο παραπάνω queries επιστρέφουν τους πελάτες που έχουν κάποιο λογαριασμό. Αν αποφασίζαμε να προσθέσουμε στο Account ένα πεδίο address με τη διεύθυνση του υποκαταστήματος που ανοίχθηκε ο λογαριασμός, τότε ενώ η πρώτη query θα εξακολουθούσε να λειτουργεί σωστά η δεύτερη θα έπρεπε να αλλαχθεί καθώς το Account δε θα συνενώνονταν μόνο ως προς accid αλλά και ως προς address.

Οι συνενώσεις στην MySQL5.7 στη γενικότερη περίπτωση εκτελούνται με nested loop όπου κάθε εγγραφή του ενός πίνακα ελέγχεται αν ανήκει στον άλλο πίνακα. Η σειρά με την οποία θα διαβαστούν οι πίνακες δηλ. ποιος θα είναι στο εξωτερικό loop και ποιος στο εσωτερικό, παίζει ρόλο στον τελικό χρόνο εκτέλεσης. Καθοριστικό ρόλο επίσης παίζει η ύπαρξη ευρετηρίου στο/ά πεδίο/α του join. Συνήθως ευρετήρια είναι δομές δεδομένων που δεικτοδοτούν στο δίσκο και απαντούν γρήγορα σε ερώτημα ισότητας. Τέτοιου είδους ευρετήρια φτιάχνονται αυτόματα στα πεδία των PRIMARY και UNIQUE KEY. Σε επόμενο κεφάλαιο θα δούμε πώς μπορούμε να δημιουργούμε και άλλα ευρετήρια.

Έστω τώρα ότι έχω το join: T1 INNER JOIN T2 ON T1.a=T2.a. Αν για το a υπάρχει ευρετήριο στον T1 αλλά όχι στο T2, τότε το να έχω στο εξωτερικό loop το T2 θα είναι τις περισσότερες φορές καλύτερο αφού ο έλεγχος κάθε εγγραφής του T2 για το αν συνδυάζεται με το T1 εκμεταλλεύεται το ευρετήριο, ενώ στην αντίθετη περίπτωση που στο εξωτερικό loop ήταν το T1 το ευρετήριο θα έμενε δυνητικά ανεκμετάλλευτο. Μπορούμε να δούμε το πλάνο εκτέλεσης μιας query χρησιμοποιώντας την εντολή EXPLAIN, γράφοντας EXPLAIN ή EXPLAIN EXTENDED, πριν το SELECT.
Παράδειγμα:
```
EXPLAIN EXTENDED
SELECT Customer.*
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
        INNER JOIN Transfer ON Account.accid=Transfer.accidSource;
```
Επιστρέφονται διάφορα πεδία των οποίων η πλήρης εξήγηση ξεφεύγει από το σκοπό του συγγράμματος (ο αναγνώστης που επιθυμεί περισσότερες πληροφορίες μπορεί να αναφερθεί στο Κεφ. 8.8.2 του εγχειριδίου). Αν παρατηρήσουμε το πεδίο table που δείχνει τον πίνακα στον οποίο δουλεύει η query θα παρατηρήσουμε ότι η σειρά με την οποία διαβάζονται οι πίνακες είναι η εξής: Transfer, Owns, Account, Customer.

Καθώς η εσωτερική συνένωση είναι πράξη αντιμεταθετική και προσεταιριστική ο optimizer της MySQL5.7 θα εκτελέσει τις συνενώσεις με τη σειρά που θεωρεί ότι θα είναι βέλτιστη. Μπορούμε να επιβάλουμε τη σειρά με την οποία θα εκτελεστούν τα join σε μία query με τη χρήση του STRAIGHT_JOIN που έχει την ίδια σύνταξη και λειτουργία με το INNER JOIN, με

τη διαφορά ότι επιβάλλει πρώτα να διαβαστεί ο πίνακας που βρίσκεται αριστερά του STRAIGHT_JOIN και μετά αυτός από τα δεξιά.

Παραδείγματα:

```
EXPLAIN EXTENDED
SELECT Customer.*
FROM Customer
        STRAIGHT_JOIN Owns ON Customer.cid=Owns.cid
        STRAIGHT_JOIN Account ON Owns.accid=Account.accid
        STRAIGHT_JOIN Transfer ON Account.accid=Transfer.accidSource;
```

Η σειρά με την οποία διαβάζονται οι πίνακες είναι η εξής: Customer, Owns, Account, Transfer καθώς έχουμε προσδιορίσει πλήρως την επιθυμητή σειρά.

```
EXPLAIN EXTENDED
SELECT Customer.*
FROM Customer
        STRAIGHT_JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
        INNER JOIN Transfer ON Account.accid=Transfer.accidSource;
```

Η σειρά με την οποία διαβάζονται οι πίνακες είναι η εξής: Transfer, Account, Customer, Owns. Παρατηρήστε ότι το STRAIGHT_JOIN επέβαλε τη σειρά μεταξύ Customer και Owns και όχι τη γενικότερη σειρά του Customer.

Εναλλακτικά μπορούμε να καθορίσουμε τη σειρά με την οποία θα προσπελαστούν οι πίνακες, χρησιμοποιώντας STRAIGHT_JOIN στο SELECT. Για παράδειγμα:

```
EXPLAIN EXTENDED
SELECT STRAIGHT_JOIN Customer.*
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
        INNER JOIN Transfer ON Account.accid=Transfer.accidSource;
```

Επιβάλει ότι τα joins θα γίνουν με τη σειρά που αναγράφονται, είναι δηλαδή ισοδύναμο με το να έγραφα STRAIGHT_JOIN στη θέση κάθε INNER_JOIN. Ως εκ τούτου η σειρά με την οποία διαβάζονται οι πίνακες είναι η εξής: Customer, Owns, Account, Transfer.

Τέλος αξίζει να σημειωθεί ότι η MySQL5.7 επιβάλει συγκεκριμένη σειρά με την οποία θα εκτελεστούν οι εξωτερικές συνενώσεις καθώς στη γενικότερη περίπτωση δεν είναι αντιμεταθετικές και προσεταιριστικές πράξεις. Έτσι στην έκφραση: T1 LEFT JOIN T2 ON T1.a=T2.a πρώτα θα προσπελαστεί ο αριστερός πίνακας (T1) και μετά ο δεξιά (T2). Αυτό σημαίνει ότι σε μία σειρά από LEFT JOIN οι προσπελάσεις θα είναι σύμφωνα με τον τρόπο που γράφονται. Για παράδειγμα:

```
EXPLAIN EXTENDED
SELECT Customer.*
FROM Customer
        LEFT JOIN Owns ON Customer.cid=Owns.cid
        LEFT JOIN Account ON Owns.accid=Account.accid
        LEFT JOIN Transfer ON Account.accid=Transfer.accidSource;
```

Η σειρά με την οποία διαβάζονται οι πίνακες είναι η: Customer, Owns, Account, Transfer. Μοναδική εξαίρεση στον κανόνα είναι αν στο ON απαιτείται προσπέλαση σε άλλον πίνακα.

Παράδειγμα

```
EXPLAIN EXTENDED
SELECT *
FROM Owns INNER JOIN Phones
LEFT JOIN Customer ON Customer.cid=Owns.cid;
```

Η σειρά με την οποία διαβάζονται οι πίνακες είναι η: Owns, Customer, Phones.

7.1.4 Υλοποίηση INTERSECT, EXCEPT και FULL JOIN

Οι πράξεις INTERSECT, EXCEPT και FULL JOIN όπως έχει ήδη αναφερθεί δεν υποστηρίζονται από τη MySQL5.7, μπορούν όμως να υλοποιηθούν μέσω άλλων πράξεων. Σε αυτήν την υποενότητα δείχνουμε πώς μπορούν να υλοποιηθούν με χρήση άλλων join.

Η πράξη INTERSECT
Το INTERSECT είναι αντίστοιχο του INNER JOIN με τη διαφορά ότι στο INTERSECT δεν επιστρέφονται διπλές εγγραφές, ενώ στο αντίστοιχο INNER JOIN αν μία εγγραφή από τον αριστερό πίνακα συνδυάζεται με πολλές εγγραφές από το δεξιό πίνακα στο αποτέλεσμα θα εμφανίζεται η εγγραφή πολλές φορές.
Παράδειγμα
Το παρακάτω ερώτημα βρίσκει τους πελάτες που τους λένε Maria και έχουν κινητό τηλέφωνο.

```
(SELECT cid FROM Customer WHERE name='Maria')
INTERSECT
(SELECT cid FROM Phones WHERE pnum LIKE '6%');
```

Ισοδύναμο ερώτημα με INNER JOIN

```
SELECT DISTINCT cid
FROM Customer INNER JOIN Phones USING(cid)
WHERE name='Maria' AND pnum LIKE '6%';
```

Η πράξη EXCEPT
Το EXCEPT μπορεί να υλοποιηθεί με χρήση LEFT JOIN. Το LEFT JOIN μας επιστρέφει όλες τις εγγραφές του αριστερού πίνακα. Αυτές που υπάρχουν και στο δεξιό θα εμφανίζονται με ίδιες τιμές στα πεδία του join. Επομένως ζητώντας στο WHERE τα πεδία του join στο δεξιό πίνακα να είναι NULL θα κρατηθούν οι εγγραφές του αριστερού πίνακα που δεν υπάρχουν στο δεξιό.
Παράδειγμα
Το παρακάτω ερώτημα βρίσκει τους πελάτες που τους λένε Maria και δεν έχουν κινητό τηλέφωνο.

```
(SELECT cid FROM Customer WHERE name='Maria')
EXCEPT
(SELECT cid FROM Phones WHERE pnum LIKE '6%');
```

Ισοδύναμο ερώτημα με LEFT JOIN και subqueries:

```
SELECT T1.cid AS cid -- DISTINCT not needed
FROM (SELECT cid FROM Customer WHERE name='Maria') T1
        LEFT JOIN
        (SELECT cid FROM Phones WHERE pnum LIKE '6%') T2 ON T1.cid=T2.cid
WHERE T2.cid IS NULL;
```

Κάνω την πρώτη subquery LEFT JOIN με τη δεύτερη. Στο αποτέλεσμα θα υπάρχουν εγγραφές της μορφής (T1.cid, T2.cid) με όλες τις εγγραφές της T1 να εμφανίζονται. Δύο περιπτώσεις θα έχουμε όσον αφορά το T2.cid. Ή θα ισχύει T2.cid=T1.cid στην οποία περίπτωση το cid αυτό θα το έπαιρνα με το INNER JOIN των δύο subquery, που σημαίνει ότι αντιστοιχεί σε κινητό τηλέφωνο ή το T2.cid θα είναι NULL. Η τελευταία περίπτωση είναι αυτή που θα μας επιστρέψει και το σωστό αποτέλεσμα της αφαίρεσης. Προσέξτε ότι δεν υπάρχει περίπτωση διπλών εγγραφών καθώς το join γίνεται σε PRIMARY KEY οπότε δε χρειάζεται το DISTINCT. Γενικά μιλώντας καλό είναι να χρησιμοποιούμε το DISTINCT μόνο όταν χρειάζεται απαραίτητα, καθώς η εκτέλεσή του κοστίζει σε χρόνο (μπορεί να επιβάλει την ταξινόμηση των εγγραφών του αποτελέσματος).

Ισοδύναμο ερώτημα με LEFT JOIN χωρίς subqueries:

```
SELECT Customer.cid AS cid
FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid AND pnum LIKE '6%'
WHERE name='Maria' AND Phones.cid IS NULL;
```

Στην παραπάνω query μεταφέραμε τα κριτήρια του WHERE της subquery που αφαιρείται στην *join_συνθήκη* επιτρέποντάς μας να γράψουμε την query χωρίς να χρησιμοποιήσουμε subqueries. Αναλόγως της περίπτωσης η χρήση ή όχι υποερωτημάτων μπορεί να οδηγήσει σε αργότερη ή γρηγορότερη εκτέλεση. Στο παράδειγμά μας αν χρησιμοποιούσαμε την EXPLAIN EXTENDED για την προηγούμενη query που χρησιμοποιεί υποερωτήματα και κατόπιν δίναμε θα παρατηρούσαμε ότι βγαίνουν warnings. Με SHOW WARNINGS; θα διαπιστώναμε ότι η query μετατράπηκε από τον optimizer στην παραπάνω μορφή δηλ. χωρίς υποερωτήματα.

Η πράξη FULL JOIN
Μπορούμε να υλοποιήσουμε FULL JOIN στη MySQL5.7 ενώνοντας τα αποτελέσματα ενός LEFT και ενός RIGHT JOIN. Έστω ότι θέλουμε να υλοποιήσουμε το ακόλουθο:

```
SELECT *
FROM Phones FULL JOIN Customer ON Phones.cid=Customer.cid;
```

Πρώτη προσπάθεια:

```
(SELECT * FROM Phones LEFT JOIN Customer on Phones.cid=Customer.cid)
UNION
(SELECT * FROM Phones RIGHT JOIN Customer on Phones.cid=Customer.cid);
```

Τρέχοντας την παραπάνω query κόβονται από το αποτέλεσμα οι διπλές εγγραφές. Έτσι αν οι εγγραφές που υπήρχαν στους πίνακες ήταν όπως στους παρακάτω πίνακες το αποτέλεσμα θα ήταν σωστό.

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333
4	6999999999

Ας θεωρήσουμε χάριν παραδείγματος ότι στον πίνακα Phones υπάρχει μία διπλή εγγραφή (μπορούμε να το πετύχουμε απενεργοποιώντας το primary key) έστω:

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333
4	6999999999
4	6999999999

Σε αυτήν την περίπτωση στο FULL JOIN θα πρέπει να υπάρξουν δύο εγγραφές <4, 6999999999, NULL, .., NULL>, αλλά με την προηγούμενη query θα υπάρξει μόνο μία. Αν και το να υπάρξουν διπλές εγγραφές σε πίνακα είναι εξαιρετικά απίθανο στην πράξη, εντούτοις μπορεί να προκύψει παρόμοια περίπτωση αν το FULL JOIN γίνεται σε πεδίο/α που δεν είναι μοναδικό/α.

Ο τρόπος για να κρατήσουμε τις διπλές εγγραφές με το UNION είναι να χρησιμοποιήσουμε UNION ALL αντί του DISTINCT που είναι το default. Σε αυτήν την περίπτωση όμως θα

εμφανιστούν δύο φορές οι εγγραφές που ανήκουν στο INNER JOIN μία ως αποτέλεσμα του πρώτου υποερωτήματος και μία του δεύτερου. Θα πρέπει λοιπόν τώρα να διώξουμε από το δεύτερο υποερώτημα τα αποτελέσματα του πρώτου. Ο τρόπος είναι αυτός που παρουσιάστηκε στην υλοποίηση του EXCEPT δηλ. με τη χρήση του IS NULL στο WHERE.

Τελική απάντηση:

```
(SELECT * FROM Phones LEFT JOIN Customer on Phones.cid=Customer.cid)
UNION ALL
(SELECT * FROM Phones RIGHT JOIN Customer on Phones.cid=Customer.cid
WHERE Phones.cid IS NULL);
```

7.2 Παράδειγμα Εργαστηριακής Άσκησης

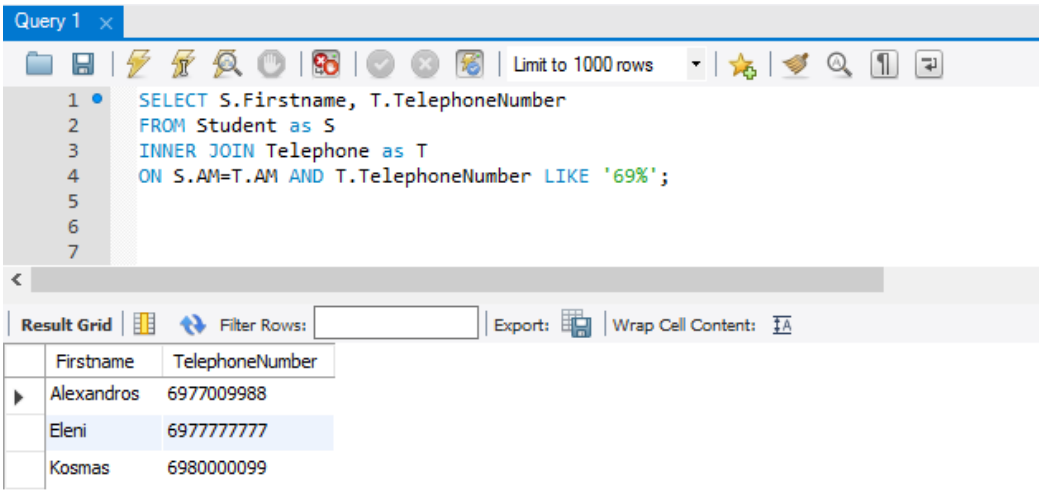
Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: **Σετ ερωτήσεων Α (Εσωτερικές Συνενώσεις – INNER Joins)** 1. Εμφανίστε τα ονόματα όλων των φοιτητών μαζί με τα τηλέφωνά τους. Δείξτε μόνο τους φοιτητές που διαθέτουν κινητό τηλέφωνο. 2. Εμφανίστε τους φοιτητές που έχουν περάσει το μάθημα ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ μαζί με το βαθμό τους. 3. Εμφανίστε τους καθηγητές μαζί με τα μαθήματα που διδάσκουν. **Σετ ερωτήσεων Β (Εξωτερικές Συνενώσεις – Outer Joins)** 1. Εμφανίστε όλους τους φοιτητές μαζί με τα τηλέφωνά τους. 2. Εμφανίστε όλα τα μαθήματα μαζί με τους κωδικούς τυχόν προαπαιτούμενων που έχουν.
Ζητούμενα:	Εσωτερικές συνενώσεις, εσωτερικές φυσικές συνενώσεις, σύγκριση μεταξύ καρτεσιανού γινομένου και φυσικής συνένωσης, εξωτερικές συνενώσεις

Σετ ερωτήσεων Α (Εσωτερικές Συνενώσεις – INNER Joins)

Ζητούμενο Α1: Εμφανίστε τα ονόματα όλων των φοιτητών μαζί με τα τηλέφωνά τους. Δείξτε μόνο τους φοιτητές που διαθέτουν κινητό τηλέφωνο.

Δίνουμε:
SELECT S.Firstname, T.TelephoneNumber
FROM Student as S
INNER JOIN Telephone as T ON S.AM=T.AM AND T.TelephoneNumber LIKE '69%';

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 7.1.



Εικόνα 7.1: *Οι φοιτητές που έχουν κινητά τηλέφωνα.*

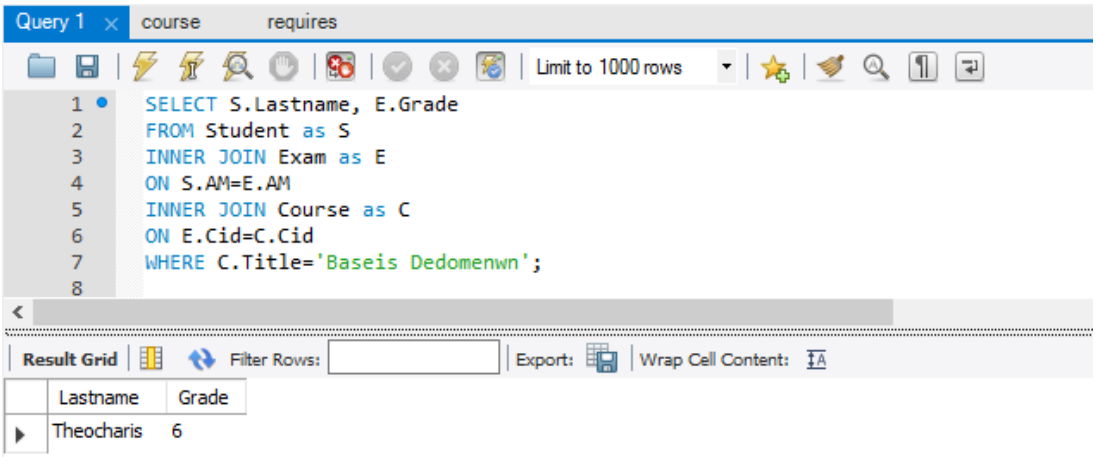
Ζητούμενο Α2: Εμφανίστε τους φοιτητές που έχουν δώσει το μάθημα ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ μαζί με το βαθμό τους.

Για την υλοποίηση του συγκεκριμένου ερωτήματος χρειαζόμαστε στοιχεία από τρεις τους: Student, Exam και Course.

Δίνουμε:

```
SELECT S.Lastname, E.Grade
FROM Student as S
INNER JOIN Exam as E ON S.AM=E.AM
INNER JOIN Course as C ON E.Cid=C.Cid
WHERE C.Title='Baseis Dedomenwn';
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 7.2.



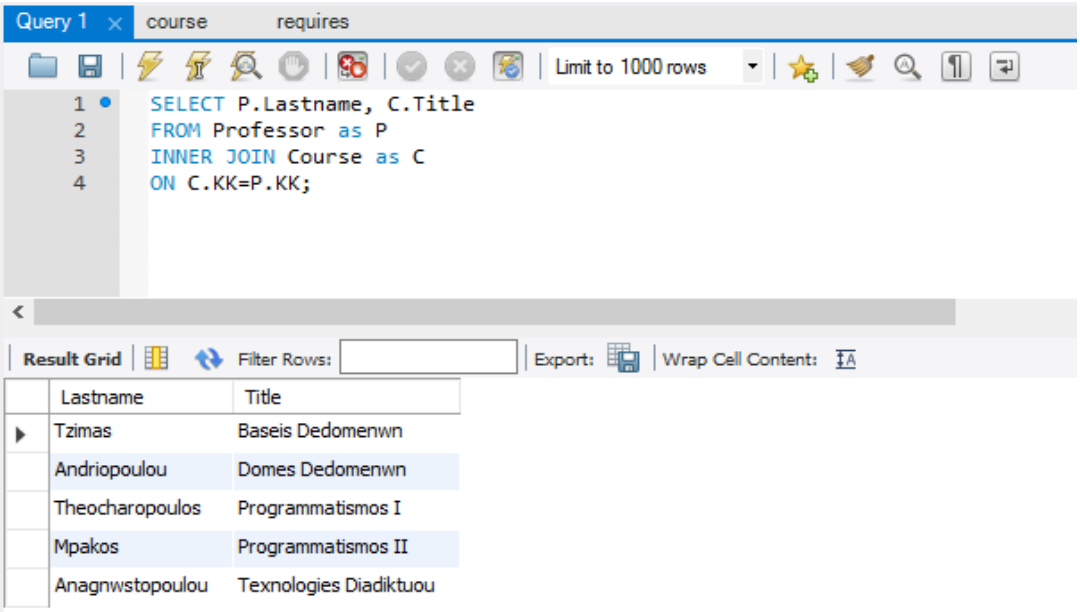
Εικόνα 7.2: *Ένας φοιτητής έχει περάσει το μάθημα ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ.*

Ζητούμενο Α3: Εμφανίστε τους καθηγητές μαζί με τα μαθήματα που διδάσκουν.

Δίνουμε:

```
SELECT P.Lastname, C.Title
FROM Professor as P
INNER JOIN Course as C ON C.KK=P.KK;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 7.3.



Εικόνα 7.3: *Οι καθηγητές και τα μαθήματά τους.*

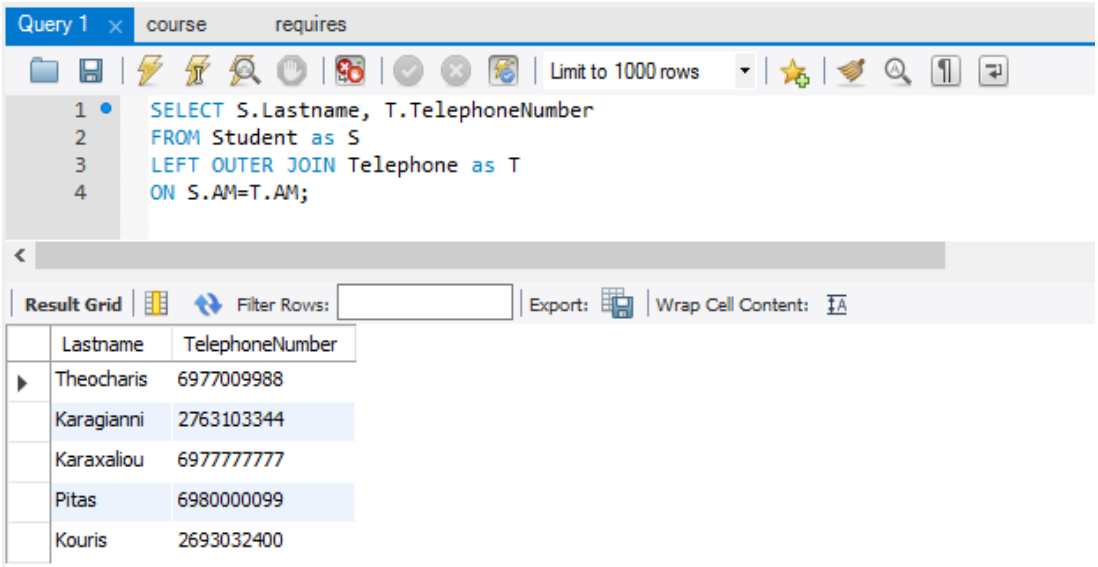
Σετ ερωτήσεων Β (Εξωτερικές Συνενώσεις – Outer Joins)

Ζητούμενο Β1: Εμφανίστε όλους τους φοιτητές μαζί με τα τηλέφωνά τους.

Το LEFT JOIN επιστρέφει όλες τις γραμμές των πεδίων του αριστερού πίνακα (Student) και όλες τις ταιριαστές γραμμές του δεξιού πίνακα (Telephone). Όταν δεν υπάρχει ταίριασμα στις γραμμές τότε επιστρέφει NULL.

Δίνουμε:
SELECT S.Lastname, T.TelephoneNumber
FROM Student as S
LEFT OUTER JOIN Telephone as T ON S.AM=T.AM;

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 7.4.



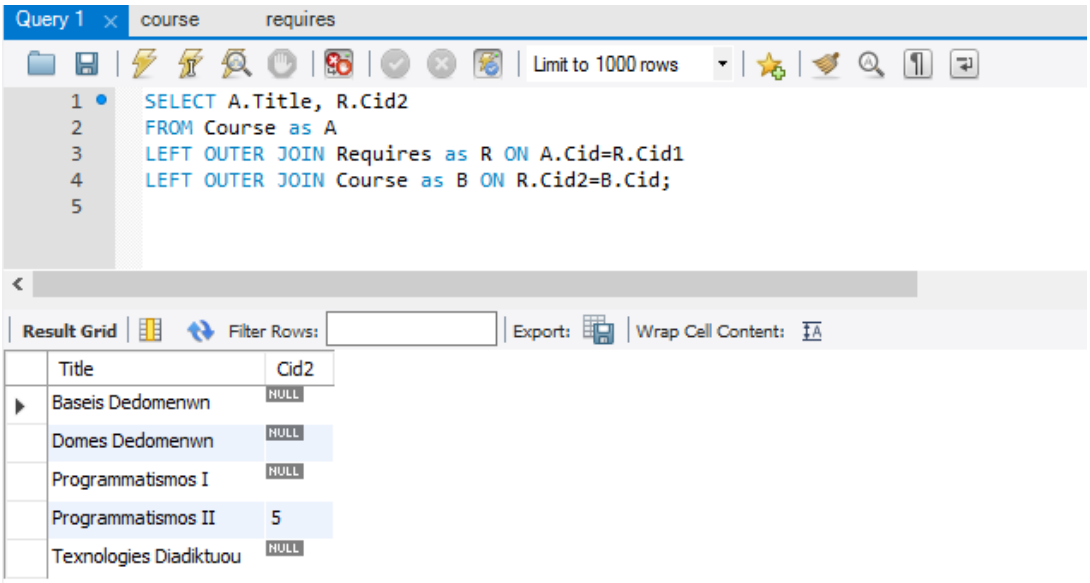
Εικόνα 7.4: *Όλοι οι φοιτητές έχουν τηλέφωνο κατά συνέπεια στο αποτέλεσμα δεν υπάρχει NULL στο τηλέφωνο.*

Ζητούμενο Β2: Εμφανίστε όλα τα μαθήματα μαζί με τους κωδικούς τυχόν προαπαιτούμενων που έχουν.

Δίνουμε:

```
SELECT A.Title, R.Cid2
FROM Course as A
LEFT OUTER JOIN Requires as R ON A.Cid=R.Cid1
LEFT OUTER JOIN Course as B ON R.Cid2=B.Cid;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 7.5.



Εικόνα 7.5: *Το μόνο μάθημα που έχει προαπαιτούμενο είναι ο Προγραμματισμός ΙΙ που έχει ως προαπαιτούμενο το μάθημα Προγραμματισμός Ι (Cid2=5).*

7.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

1. Εμφανίστε τους εργαζόμενους που δουλεύουν στην εταιρία ‘Advance’.
2. Εμφανίστε τους προμηθευτές και τις εταιρίες με τις οποίες έχουν συνεργαστεί.
3. Εμφανίστε τις εταιρίες με τις διευθύνσεις τους(η διεύθυνση είναι ξεχωριστός πίνακας).
4. Εμφανίστε τις εταιρίες που δουλεύουν διευθυντές με 13 χρόνια προϋπηρεσίας.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2), απαντήστε στα ακόλουθα ερωτήματα:

1. Εμφανίστε τους σκηνοθέτες και τις ταινίες που έχουν σκηνοθετήσει.
2. Εμφανίστε τους θεατές και τις ταινίες που έχουν βαθμολογήσει με βαθμό μεγαλύτερο του 3.
3. Εμφανίστε τους κριτές μαζί με τις ταινίες που έχουν βαθμολογήσει.
4. Εμφανίστε τις ταινίες που είχαν τους περισσότερους θεατές(και πόσοι ήταν αυτοί).

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3), απαντήστε στα ακόλουθα ερωτήματα:

1. Εμφανίστε τα τουρνουά μαζί με τους νικητές τους για το έτος 2015.
2. Εμφανίστε τους παίκτες που συμμετείχαν σε τουρνουά της Γαλλίας.
3. Εμφανίστε τους νικητές του ‘Roland Garros’ από το 2010 μέχρι σήμερα.
4. Εμφανίστε τα τουρνουά στα οποία συμμετείχε ο παίκτης ‘Rafael Nadal’ το 2015.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαια 8 και 9.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
JOIN.
https://dev.mysql.com/doc/refman/5.7/en/join.html
http://www.w3schools.com/sql/sql_join.asp
INNER JOIN. http://www.w3schools.com/sql/sql_join_inner.asp
LEFT JOIN. http://www.w3schools.com/sql/sql_join_left.asp
RIGHT JOIN. http://www.w3schools.com/sql/sql_join_right.asp
FULL JOIN. http://www.w3schools.com/sql/sql_join_full.asp.
FULL JOIN και MySQL.
http://www.xaprb.com/blog/2006/05/26/how-to-write-full-outer-join-in-mysql/

Κεφάλαιο 8 Εμφωλευμένα Ερωτήματα

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθεί η χρήση εμφωλευμένων ερωτημάτων ως πίνακες στο FROM και ως τιμές ή σύνολα εγγραφών στο WHERE.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- *Η ύλη των Κεφ.5, 6 και 7.*

.

8.1 Εισαγωγικές Έννοιες

Εμφωλευμένα ερωτήματα ή αλλιώς υποερωτήματα (subqueries) είναι ερωτήματα (τύπου SELECT) τα οποία βρίσκονται μέσα σε άλλα ερωτήματα περικλειόμενα σε παρενθέσεις. Εμφωλευμένα ερωτήματα μπορώ να έχω και στα τρία μέρη ενός εξωτερικού ερωτήματος SELECT FROM WHERE. Επίσης μπορούν να χρησιμοποιηθούν μέσα σε άλλες εντολές και όχι κατά ανάγκη σε εντολή SELECT πχ. σε INSERT ή UPDATE και DELETE. Οι δύο τελευταίες εντολές θα εξηγηθούν εκτενώς σε επόμενο κεφάλαιο.

Ένα εμφωλευμένο ερώτημα μπορεί να επιστρέφει μία τιμή, ή μία ή περισσότερες εγγραφές. Εν γένει (πλην εξαιρέσεων που δε θα μας απασχολήσουν) ένα υποερώτημα μπορεί να περιλαμβάνει ότι περιγράφεται στο συντακτικό του SELECT (βλέπε Κεφ. 13.2.9 του εγχειριδίου της MySQL5.7). Στη συνέχεια περιγράφουμε τη χρήση εμφωλευμένων ερωτημάτων στο SELECT, το FROM και το WHERE ενός εξωτερικού ερωτήματος. Περισσότερα για τα εμφωλευμένα ερωτήματα μπορεί να διαβάσει ο αναγνώστης στο Κεφ. 13.2.10 του εγχειριδίου της MySQL5.7.

Στη συνέχεια του κεφαλαίου δείχνουμε τη χρήση των υποερωτημάτων χρησιμοποιώντας τους ακόλουθους πίνακες ως παράδειγμα αναφοράς:

Customer (cid, afm, address, name, sname, dateOfBirth)
Phones (*cid*, pnum)
Account (accid, balance, dateOfCreation)
Owns (*cid*, *accid*)
Action (*accid*, actid, amount, type, dateOfAction)
Transfer (*accidSource*, *actidSource*, *accidDest*, *actidDest*)

Υποθέτουμε επίσης το ακόλουθο στιγμιότυπο για τους πίνακες:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333

Account

accid	balance	dateOfCreation
100	10000	2014-01-01
200	40000	2013-02-01
300	30000	2005-03-10

Owns

cid	accid
1	100
2	200
1	300

Action

accid	actid	amount	type	dateOfAction
100	1	500	2	2014-01-14
200	1	500	2	2014-01-14
200	2	1000	2	2015-11-11
300	1	1000	2	2015-11-11

Transfer

accidSource	actidSource	accidDest	actidDest
200	1	100	1
300	1	200	2

8.1.1 Εμφωλευμένα ερωτήματα στο SELECT

Είναι δυνατόν να χρησιμοποιήσουμε εμφωλευμένο ερώτημα για να καθορίσουμε τα πεδία που επιστρέφονται στο SELECT. Σε αυτήν την περίπτωση το υποερώτημα θα πρέπει να επιστρέφει ένα πεδίο και μία τιμή.
Παραδείγματα:
(I)

```
SELECT
(SELECT name FROM Customer WHERE cid=1),
(SELECT sname FROM Customer WHERE cid=3);
```

Επιστρέφει:

SELECT name FROM Customer WHERE cid=1	SELECT surname FROM Customer WHERE cid=3
Kostas	Papantoniou

Όπως μπορούμε να παρατηρήσουμε τα ονόματα των πεδίων είναι τα δύο υποερωτήματα. Συνήθως σε τέτοιες περιπτώσεις θα έπρεπε να υπάρξει μετονομασία.

(II)

```
SELECT
(SELECT name, sname FROM Customer WHERE cid=1),
(SELECT sname FROM Customer WHERE cid=3);
```

Το ερώτημα θα δημιουργήσει το ακόλουθο λάθος:
ERROR 1241 (21000): Operand should contain 1 column(s)

Το πρόβλημα είναι ότι το πρώτο υποερώτημα επιστρέφει δύο πεδία, ενώ το εξωτερικό SELECT αναμένει ένα πεδίο από κάθε υποερώτημα.

(III)
SELECT
(SELECT name FROM Customer WHERE cid=1),
(SELECT cid FROM Customer WHERE sname='Kostantinou');

Το ερώτημα θα δημιουργήσει το ακόλουθο λάθος:
ERROR 1242 (21000): Subquery returns more than 1 row
Το πρόβλημα είναι ότι το δεύτερο υποερώτημα επιστρέφει δύο εγγραφές μία με cid=1 και μία με cid=2.

(IV)
SELECT
(SELECT name FROM Customer WHERE cid=1),
(SELECT sname FROM Customer WHERE cid=3)
FROM Customer;

Επιστρέφει:

SELECT name FROM Customer WHERE cid=1	**SELECT surname FROM Customer WHERE cid=3**
Kostas	Papantoniou
Kostas	Papantoniou
Kostas	Papantoniou

Για κάθε μία εγγραφή του Customer (3 συνολικά) επιστρέφεται ότι περιγράφεται από τα δύο υποερωτήματα του εξωτερικού SELECT.

Ως γενικό κανόνα καλό είναι να αποφεύγουμε τη χρήση υποερωτημάτων ως επιστρεφόμενα πεδία για λόγους αναγνωσιμότητας και αν είναι δυνατόν να μετατρέπουμε το αντίστοιχο ερώτημα χρησιμοποιώντας joins. Για παράδειγμα το ερώτημα (I) μπορεί να επαναδιατυπωθεί χρησιμοποιώντας self join:
SELECT C1.name, C2.surname
FROM Customer C1, Customer C2
WHERE C1.cid=1 AND C2.cid=3;
Εξαίρεση του παραπάνω γενικού κανόνα αποτελεί η χρήση υποερωτημάτων που εμπλέκουν ερωτήματα συνάθροισης. Για παράδειγμα έστω το ακόλουθο ερώτημα: *Επέστρεψε το πλήθος των εγγραφών που υπάρχουν στο Customer και το Phones*.
Σημείωση: Για να πάρουμε το πλήθος των εγγραφών ενός πίνακα που δεν είναι NULL ως προς κάποια πεδία, χρησιμοποιούμε τη συνάρτηση COUNT() με όρισμα τα πεδία αυτά. Κατ' επέκταση μπορούμε να μετρήσουμε τις εγγραφές ενός πίνακα χρησιμοποιώντας COUNT(*). Περισσότερα για την COUNT() και ερωτήματα συναθροίσεων σε επόμενο κεφάλαιο.

1ος τρόπος
SELECT
(SELECT COUNT(*) FROM Customer) AS CustomerRecords,
(SELECT COUNT(*) FROM Phones) AS PhonesRecords ;

2ος τρόπος
SELECT *
FROM
(SELECT COUNT(*) AS CustomerRecords FROM Customer) T1,
(SELECT COUNT(*) AS PhonesRecords FROM Phones) T2;

3ος τρόπος
```
(SELECT COUNT(*) FROM Customer)
UNION ALL
(SELECT COUNT(*) FROM Phones);
```

Για πολλούς ο πρώτος τρόπος γραφής είναι καλύτερος από το δεύτερο καθώς ο μόνος λόγος για τον οποίο συνδυάζονται οι πίνακες T1 και T2 είναι για να μπορεί να εμφανιστεί το αποτέλεσμα σε μία γραμμή αντί σε μία στήλη όπως κάνει ο τρίτος τρόπος που δεν απαντά ουσιαστικά στο ερώτημα.

8.1.2 Εμφωλευμένα ερωτήματα στο WHERE

Μία από τις πιο συνηθισμένες χρήσεις εμφωλευμένων ερωτημάτων είναι για δημιουργία μίας εγγραφής ή συνόλου εγγραφών έναντι του οποίου συγκρίνονται οι εγγραφές ενός εξωτερικού πίνακα. Ως τελεστές σύγκρισης μπορούμε να έχουμε οποιονδήποτε από τους: <, >, <=, >=, <>, !, =, <=> όταν το υποερώτημα επιστρέφει μία μόνο εγγραφή και τους παραπάνω τελεστές μαζί με τις επιλογές ALL, ANY και SOME όταν το υποερώτημα επιστρέφει περισσότερες από μία εγγραφές. Η επιλογή ALL επιστρέφει TRUE αν ο τελεστής σύγκρισης ισχύει για όλες τις εγγραφές που επιστρέφει το υποερώτημα αλλιώς αν δεν ισχύει έστω και για μία επιστρέφεται FALSE. Οι επιλογές ANY και SOME είναι ισοδύναμες και επιστρέφουν TRUE αν ο τελεστής σύγκρισης ισχύει για τουλάχιστον μία εγγραφή που επιστρέφεται από το υποερώτημα, αλλιώς αν δεν ισχύει για καμία επιστρέφεται FALSE. Ακολουθούν παραδείγματα χρήσης.

Ερώτημα 1. Βρες τα τηλέφωνα του πελάτη με afm='077783234'.

(I) Χωρίς εμφωλευμένο με χρήση INNER JOIN.
```
SELECT pnum
FROM Phones INNER JOIN Customer ON Phones.cid=Customer.cid
WHERE afm='077783234';
```

(II) Με εμφωλευμένο ερώτημα και =.
```
SELECT pnum
FROM Phones
WHERE cid =
        (SELECT cid
        FROM Customer
        WHERE afm='077783234');
```

(III) Με εμφωλευμένο ερώτημα και =ANY.
```
SELECT pnum
FROM Phones
WHERE cid =ANY
        (SELECT cid
        FROM Customer
        WHERE afm='077783234');
```

Η λογική εκτέλεση των ερωτημάτων (II) σύμφωνα με την SQL σύνταξη έχει ως εξής: για κάθε εγγραφή του Phones ελέγχεται αν το cid της εγγραφής ισούται με το cid που επιστρέφει το εμφωλευμένο ερώτημα. Σε περίπτωση ισότητας η εγγραφή του Phones επιστρέφεται και προβάλλεται το πεδίο της pnum. Παρατηρώντας προσεκτικότερα το εμφωλευμένο ερώτημα διαπιστώνουμε ότι το αποτέλεσμά του είναι ανεξάρτητο από την εγγραφή του Phones πάνω στην οποία «τρέχει». Αυτό σημαίνει ότι δυνητικά δε χρειάζεται να τρέξει το εμφωλευμένο ερώτημα για κάθε μία από τις εγγραφές του Phones αλλά μπορεί να εκτελεστεί μία φορά και τα αποτελέσματα του να συγκριθούν με τις εγγραφές του Phones. Αυτού του είδους τα

υποερωτήματα ονομάζονται *ασυσχέτιστα* και η παραπάνω βελτιστοποίηση που περιγράφηκε (δηλ. η εκτέλεση του υποερωτήματος μία μόνο φορά) πραγματοποιείται από όλα τα κύρια DBMS και την MySQL5.7. Αντίστοιχη είναι και η εκτέλεση του ερωτήματος (III).

Ερώτημα 2. Βρες τα τηλέφωνα του πελάτη με sname='Kostantinou'.

(I) Χωρίς εμφωλευμένο με χρήση INNER JOIN.

```
SELECT pnum
FROM Phones INNER JOIN Customer ON Phones.cid=Customer.cid
WHERE sname='Kostantinou';
```

(II) Με εμφωλευμένο ερώτημα και =.

```
SELECT pnum
FROM Phones
WHERE cid =
        (SELECT cid
        FROM Customer
        WHERE sname='Kostantinou');
```

Το ερώτημα θα δημιουργήσει το ακόλουθο λάθος:
ERROR 1242 (21000): Subquery returns more than 1 row
Ο λόγος είναι γιατί το εσωτερικό ερώτημα επιστρέφει παραπάνω από μία εγγραφές (για την ακρίβεια 3). Στο εξωτερικό WHERE μπορούμε να συγκρίνουμε χρησιμοποιώντας κάποιον από τους τελεστές σύγκρισης χωρίς ANY ή ALL μόνο αν το εσωτερικό ερώτημα επιστρέφει μία εγγραφή.

(III) Με εμφωλευμένο ερώτημα και =ANY.

```
SELECT pnum
FROM Phones
WHERE cid =ANY
        (SELECT cid
        FROM Customer
        WHERE sname='Kostantinou');
```

Επιστρέφει κανονικά το αποτέλεσμα.

Ερώτημα 3. Βρες τους κωδικούς των λογαριασμών που δεν έχουν το μεγαλύτερο υπόλοιπο.

(I) Χωρίς εμφωλευμένο με χρήση self-join (αυτό-συνένωσης).

```
SELECT A1.accid
FROM Account A1, Account A2
WHERE A1.balance<A2.balance;
```

(II) Με εμφωλευμένο ερώτημα και <ANY.

```
SELECT accid
FROM Account
WHERE balance <ANY
        (SELECT balance
        FROM Account);
```

Ερώτημα 4. Βρες τους κωδικούς των λογαριασμών που έκαναν μεταφορά χρημάτων προς το λογαριασμό με κωδικό 100, μαζί με τις κινήσεις και τα ποσά.

(I) Χωρίς εμφωλευμένο.

```
SELECT accid, actid, amount
```

```
FROM Action INNER JOIN Transfer ON accid=accidSource AND actid=actidSource
WHERE accidDest=100;
```

(II) Με ασυσχέτιστο εμφωλευμένο ερώτημα.

```
SELECT accid, actid, amount
FROM Action
WHERE (accid, actid) =ANY
        (SELECT accidSource, actidSource
        FROM Transfer
        WHERE accidDest=100);
```

(III) Με συσχετισμένο εμφωλευμένο ερώτημα.

```
SELECT accid, actid, amount
FROM Action
WHERE accid =ANY
        (SELECT accidSource
        FROM Transfer
        WHERE accidDest=100 AND actid=actidSource);
```

Όπως προαναφέρθηκε μπορούμε να συγκρίνουμε την εγγραφή που εξετάζει το εξωτερικό WHERE έναντι των αποτελεσμάτων ενός υποερωτήματος ως προς περισσότερα του ενός πεδία, στην οποία περίπτωση τα πεδία της σύγκρισης περιλαμβάνονται σε παρένθεση. Παράδειγμα τέτοιου ερωτήματος είναι το (II).

Το ερώτημα (III) είναι παράδειγμα συσχετισμένου εμφωλευμένου ερωτήματος. Συσχετισμένο καλείται ένα εμφωλευμένο ερώτημα όταν χρησιμοποιεί πεδίο του εξωτερικού ερωτήματος. Στο παράδειγμα το υποερώτημα εκτελεί τη σύγκριση actid=actidSource. Το αποτέλεσμα ενός συσχετισμένου υποερωτήματος (σε αντίθεση με τα ασυσχέτιστα) δυνητικά αλλάζει αναλόγως της εγγραφής που ελέγχεται στο εξωτερικό WHERE. Αυτό σημαίνει ότι το υποερώτημα δεν μπορεί να εκτελεστεί μόνο μία φορά αλλά θα πρέπει να εκτελείται για κάθε εγγραφή του εξωτερικού ερωτήματος. Αν και αυτό φαίνεται εκ πρώτης όψεως μη αποδοτικό θα συζητήσουμε στη συνέχεια του κεφαλαίου περιπτώσεις που συμφέρει η χρήση συσχετισμένου εμφωλευμένου ερωτήματος.

8.1.3 Οι τελεστές IN, NOT IN, EXISTS και NOT EXISTS

Δύο από τους πιο συχνούς συνδυασμούς των τελεστών σύγκρισης με τα ANY και ALL είναι το =ANY και το <>ALL, που υπολογίζουν αν η εγγραφή που ελέγχεται στο εξωτερικό WHERE ανήκει ή δεν ανήκει στα αποτελέσματα ενός υποερωτήματος. Για το λόγο αυτό και για λόγους αναγνωσιμότητας προσφέρονται οι τελεστές IN και NOT IN αντίστοιχα, τη χρήση των οποίων θα προτιμούμε έναντι των =ANY και <>ALL.

Δύο άλλοι χρήσιμοι τελεστές είναι οι EXISTS και NOT EXISTS που χρησιμοποιούνται συνήθως σε συνδυασμό με συσχετισμένα εμφωλευμένα ερωτήματα. Ο τελεστής EXISTS επιστρέφει TRUE αν το αποτέλεσμα του υποερωτήματος έχει έστω και μία εγγραφή, αλλιώς επιστρέφει FALSE. Το αντίθετο κάνει ο τελεστής NOT EXISTS. Ακολουθεί παράδειγμα χρήσης.

Ερώτημα 4 (συνέχεια). Βρες τους κωδικούς των λογαριασμών που έκαναν μεταφορά χρημάτων προς το λογαριασμό με κωδικό 100, μαζί με τις κινήσεις και τα ποσά.

(I) Με χρήση IN.

```
SELECT accid, actid, amount
FROM Action
```

```
WHERE (accid, actid) IN
        (SELECT accidSource, actidSource
        FROM Transfer
        WHERE accidDest=100);
```

(II) Με χρήση EXISTS.

```
SELECT accid, actid, amount
FROM Action
WHERE EXISTS
        (SELECT *
        FROM Transfer
        WHERE accidDest=100 AND accid=accidSource AND actid=actidSource);
```

Action

accid	actid	amount	type	dateOfAction
100	1	500	2	2014-01-14
200	1	500	2	2014-01-14
200	2	1000	2	2015-11-11
300	1	1000	2	2015-11-11

Transfer

accidSource	actidSource	accidDest	actidDest
200	1	100	1
300	1	200	2

Όπως γίνεται φανερό η χρήση του IN είναι αντίστοιχη της χρήσης του =ANY. Η χρήση του EXISTS όπως αναφέρθηκε γίνεται τις πιο πολλές φορές σε συνδυασμό με εμφωλευμένο ερώτημα. Αν υποθέσουμε το παραπάνω στιγμιότυπο για τους πίνακες Action και Transfer, η εκτέλεση του ερωτήματος (II) θα γίνονταν στο λογικό επίπεδο ως εξής (στην πράξη ο optimizer μπορεί να επιλέξει κάτι διαφορετικό). Αρχικά επιλέγεται η πρώτη εγγραφή του πίνακα Action δηλ. αυτή με: (accid, actid) = (100, 1). Ελέγχονται μία προς μία οι εγγραφές του Transfer ως προς τη συνθήκη WHERE στο εμφωλευμένο. Καμία από τις δύο εγγραφές δεν πληρεί τη συνθήκη επομένως το εμφωλευμένο δεν επιστρέφει εγγραφές, η συνθήκη EXISTS είναι FALSE και η πρώτη εγγραφή του Action δεν επιστρέφεται στο αποτέλεσμα. Στη συνέχεια ελέγχεται για τη δεύτερη εγγραφή του Action αν το εξωτερικό WHERE δίνει TRUE. Επομένως για την εγγραφή του Action με: (accid, actid) = (200, 1) υπολογίζεται ποιες εγγραφές του Transfer πληρούν το εσωτερικό WHERE. Υπάρχει μία τέτοια εγγραφή, η πρώτη με accidSource=200, επομένως το εμφωλευμένο ερώτημα επιστρέφει μία εγγραφή, άρα το EXISTS επιστρέφει TRUE και επομένως η δεύτερη εγγραφή του Action επιστρέφεται στο αποτέλεσμα. Με παρόμοιο τρόπο ελέγχονται και οι υπόλοιπες δύο εγγραφές του Action και το τελικό αποτέλεσμα είναι:

accid	actid	amount
200	1	500

Όπως γίνεται φανερό όταν έχω συσχετισμένο εμφωλευμένο ερώτημα με EXISTS τα πεδία που επιστρέφονται από το εμφωλευμένο δεν παίζουν ρόλο. Είθισται ή να γράφουμε το εμφωλευμένο με SELECT * ή ακόμα καλύτερα για να είναι εύκολα αναγνωρίσιμο το ότι δεν παίζουν κανένα ρόλο τα πεδία χρησιμοποιώντας SELECT 1. Στο παράδειγμα:

```
SELECT accid, actid, amount
FROM Action
WHERE EXISTS
        (SELECT 1
        FROM Transfer
        WHERE accidDest=100 AND accid=accidSource AND actid=actidSource);
```

Χρησιμοποιώντας SELECT 1 και αναλόγως του DBMS καθοδηγούμε επίσης καλύτερα τον optimizer όσον αφορά στη βελτιστοποίηση του ερωτήματος. Για παράδειγμα αν έχουμε joins στο εμφωλευμένο πιθανώς να χρησιμοποιήσει λιγότερη μνήμη για να τα πραγματοποιήσει καθώς δε μας ενδιαφέρουν όλα τα πεδία που προκύπτουν από τα joins. Η MySQL5.7 μετατρέπει το SELECT * ενός εμφωλευμένου με EXISTS σε SELECT 1.

8.1.4 Πολλαπλές εμφωλεύσεις, τομή, αφαίρεση και εσωτερική συνένωση

Πολλαπλά επίπεδα εμφώλευσης
Ένα SQL ερώτημα μπορεί να χρησιμοποιεί εμφωλευμένα ερωτήματα πολλών επιπέδων. Θεωρείστε το ακόλουθο παράδειγμα.

Ερώτημα 5. Βρες τα afm των πελατών που έχουν κάνει μεταφορά χρημάτων προς το λογαριασμό με κωδικό 100.

(I) Με χρήση INNER JOIN.
```
SELECT afm
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Transfer ON Owns.accid=Transfer.accidSource
WHERE accidDest=100;
```

(II) Με πολλαπλά επίπεδα εμφώλευσης χρησιμοποιώντας IN.
```
SELECT afm
FROM Customer
WHERE cid IN
        (SELECT cid
        FROM Owns
        WHERE accid IN
                (SELECT accidSource
                FROM Transfer
                WHERE accidDest=100));
```

(III) Με πολλαπλά επίπεδα εμφώλευσης χρησιμοποιώντας EXISTS.
```
SELECT afm
FROM Customer
WHERE EXISTS
        (SELECT 1
        FROM Owns
        WHERE Customer.cid=Owns.cid AND EXISTS
                (SELECT 1
                FROM Transfer
                WHERE accid=accidSource AND accidDest=100));
```

Η λογική σειρά εκτέλεσης στα (II) και (III) έχει ως εξής: για κάθε μία εγγραφή του Customer ελέγχεται το εξωτερικό WHERE. Κάθε έλεγχος του εξωτερικού WHERE ενέχει την εκτέλεση ενός υποερωτήματος με υποερώτημα. Αυτό με τη σειρά του υπολογίζεται ελέγχοντας κάθε

εγγραφή του Owns αν πληροί την αντίστοιχη συνθήκη στο WHERE κλπ. Αξίζει να ειπωθεί ότι στο (III) το εμφωλευμένο FROM Transfer είναι συσχετισμένο, επομένως θα πρέπει να υπολογίζεται για κάθε εγγραφή του FROM Owns που κι αυτό είναι συσχετισμένο και με τη σειρά του θα πρέπει να υπολογίζεται για κάθε εγγραφή του FROM Customer. Αντίθετα στο (II), το FROM Transfer δεν είναι συσχετισμένο επομένως μπορεί να εκτελεστεί μία φορά. Κατόπιν το αποτέλεσμα να χρησιμοποιηθεί για να εκτελεστεί μία φορά το FROM Owns και στη συνέχεια να υπολογιστεί το εξωτερικό ερώτημα FROM Customer.

Τομή και εσωτερική συνένωση
Εκ των προηγούμενων παραδειγμάτων του κεφαλαίου εύκολα μπορεί να συναχθεί ότι μπορούμε να υλοποιήσουμε την πράξη της τομής και της εσωτερικής συνένωσης χρησιμοποιώντας είτε ασυσχέτιστο εμφωλευμένο και τον τελεστή IN, είτε συσχετισμένο με χρήση του EXISTS. Γενικότερα μιλώντας, ερωτήματα που ενέχουν συνενώσεις μπορεί να ξαναγραφούν χρησιμοποιώντας εμφωλευμένα. Το αντίθετο, δηλ. η μετατροπή ενός ερωτήματος που ενέχει εμφωλευμένα υποερωτήματα σε ερώτημα που χρησιμοποιεί μόνο συνενώσεις δεν μπορεί να γίνει πάντα. ΄

Αφαίρεση
Ερωτήματα που ενέχουν αφαίρεση μπορεί να γραφτούν με χρήση υποερωτημάτων και των τελεστών NOT IN ή NOT EXISTS. Θεωρείστε το ακόλουθο παράδειγμα.

Ερώτημα 6. Βρες τους πελάτες που δεν έχουν κινητά τηλέφωνα.

(I) Με χρήση NOT IN.
```
SELECT *
FROM Customer
WHERE cid NOT IN
        (SELECT cid
        FROM Phones
        WHERE pnum LIKE '69%');
```

(II) Με χρήση NOT EXISTS.
```
SELECT *
FROM Customer
WHERE NOT EXISTS
        (SELECT 1
        FROM Phones
        WHERE pnum LIKE '69%' AND Customer.cid=Phones.cid);
```

(III) Υλοποιώντας αφαίρεση με χρήση LEFT JOIN.
```
SELECT Customer.*
FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid AND pnum LIKE '69%'
WHERE pnum IS NULL;
```

Στο (III) όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο θα ήταν λάθος να μεταφέρονταν ο έλεγχος του κινητού τηλεφώνου από την join συνθήκη στο WHERE. Ο λόγος είναι ότι θέλουμε να κρατήσουμε τους συνδυασμούς των εγγραφών που αντιστοιχούν σε κινητά έτσι ώστε οι υπόλοιπες εγγραφές του Customer που δεν έχουν να εμφανίζονται με NULL τιμές στο αντίστοιχο πεδίο pnum.

Κάτι που χρήζει προσοχής όταν χρησιμοποιούμε το NOT IN είναι η περίπτωση όπου το υποερώτημα επιστρέφει και NULL. Ακολουθεί παράδειγμα.

Έστω οι κάτωθι πίνακες Customer και Phones:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333

και το εξής ερώτημα:

```
SELECT pnum
FROM Customer INNER JOIN Phones USING (cid)
WHERE pnum NOT IN
        (SELECT pnum
        FROM Customer LEFT JOIN Phones USING (cid)
        WHERE pnum IS NULL OR pnum LIKE '69%');
```

Το εσωτερικό ερώτημα κάνει LEFT JOIN το Customer με το Phones. Ενδιάμεσος πίνακας:

Customer LEFT JOIN Phones USING (cid)

cid	afm	address	name	sname	dateOfBirth	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2103333333
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	NULL

Από το εσωτερικό WHERE κρατιούνται οι εξής εγγραφές:

cid	afm	address	name	sname	dateOfBirth	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	6944444444
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20	NULL

και προβάλλονται ως προς pnum δίνοντας:

pnum
6944444444
NULL

Το εξωτερικό ερώτημα σχηματίζει το INNER JOIN των δύο πινάκων:

Customer INNER JOIN Phones USING (cid)

cid	afm	address	name	sname	dateOfBirth	pnum
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	2231011111
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30	6944444444
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02	2103333333

και στη συνέχεια για κάθε μία από τις τρεις εγγραφές ελέγχεται αν το pnum δεν ανήκει στο αποτέλεσμα του εσωτερικού ερωτήματος. Επομένως θα περίμενε κάποιος το τελικό αποτέλεσμα του συνολικού ερωτήματος να είναι:

pnum
2231011111
2103333333

Παρόλα αυτά το συνολικό ερώτημα δεν επιστρέφει καμία εγγραφή στο αποτέλεσμα. Ο λόγος είναι γιατί το NOT IN συντακτικά μεταφράζεται σαν <> από όλες τις εγγραφές του εμφωλευμένου. Για παράδειγμα για την πρώτη εγγραφή του εξωτερικού INNER JOIN ο έλεγχος είναι ισοδύναμος με:
WHERE '2231011111'<>'2231011111' AND '6944444444'<>NULL
Το '6944444444'<>NULL δίνει NULL και TRUE AND NULL πάλι NULL, επομένως η συγκεκριμένη εγγραφή του Customer δεν επιστρέφεται (ομοίως και για τις άλλες).

Το παραπάνω πρόβλημα δηλ. η επιστροφή NULL από εμφωλευμένο ερώτημα δεν είναι τόσο σημαντικό όταν χρησιμοποιούμε τον τελεστή IN ο οποίος μεταφράζεται σαν:
WHERE c=t1 OR c=t2 OR c=t3 κλπ. όπου c το πεδίο που ελέγχεται από την εξωτερική query και t1, t2, t3 οι αντίστοιχες τιμές στις εγγραφές της εσωτερικής query. Ακόμα και αν πχ. το t3 είναι NULL, δεν επηρεάζει το αποτέλεσμα καθώς TRUE OR NULL=TRUE.

Στο παράδειγμα με το NOT IN μία λύση είναι να ξαναγράψουμε την query χρησιμοποιώντας NOT EXISTS ως εξής:

```
SELECT P1.pnum
FROM Customer INNER JOIN Phones P1 USING (cid)
WHERE NOT EXISTS
        (SELECT 1
        FROM Customer LEFT JOIN Phones P2 USING (cid)
        WHERE (P2.pnum IS NULL OR P2.pnum LIKE '69%') AND P2.pnum=P1.pnum);
```

8.1.5 Εμφωλευμένα ερωτήματα στο FROM

Μπορούμε να έχουμε εμφωλευμένο ερώτημα στο FROM ενός εξωτερικού ερωτήματος. Σε αυτήν την περίπτωση πολλές φορές αναφερόμαστε στο εμφωλευμένο ερώτημα ως παραγόμενος πίνακας (derived table). Όταν έχω εμφωλευμένο ερώτημα στο FROM ο παραγόμενος πίνακας θα πρέπει να μετονομαστεί. Ακολουθούν παραδείγματα:

Ερώτημα 7. Βρες τους πελάτες που δεν έχουν κινητά τηλέφωνα και έχουν τουλάχιστον ένα λογαριασμό με υπόλοιπο μεγαλύτερο των 15.000.

Το ερώτημα απαρτίζεται από δύο υποερωτήματα. Το πλάνο λοιπόν είναι να διατυπώσω ξεχωριστά σε SQL τα δύο υποερωτήματα που το απαρτίζουν φροντίζοντας να επιστρέφει το κάθε ένα το cid του πελάτη, να τα συνδυάσω κατάλληλα και στο τέλος να κάνω join με τον πίνακα Customer για να πάρω και τα υπόλοιπα στοιχεία.

Αυτοί που δεν έχουν κινητά (χρησιμοποιώντας LEFT JOIN):
```
SELECT Customer.cid
FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid AND pnum LIKE '69%'
WHERE pnum IS NULL;
```

Αυτοί που έχουν τουλάχιστον ένα λογαριασμό με υπόλοιπο μεγαλύτερο του 15000:
```
SELECT cid
FROM Owns INNER JOIN Account USING (accid)
WHERE balance>15000;
```

Για να απαντήσουμε στο ερώτημα χρειάζεται να κάνουμε τομή των δύο παραπάνω αποτελεσμάτων. Όπως έχει προαναφερθεί μπορούμε να κάνουμε τομή ή χρησιμοποιώντας IN ή με INNER JOIN

(a) Χρησιμοποιώντας IN:

```
SELECT cid
FROM Owns INNER JOIN Account USING (accid)
WHERE balance>15000 AND cid IN
        (SELECT Customer.cid
        FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
        WHERE pnum IS NULL);
```

(b) Χρησιμοποιώντας INNER JOIN:

```
SELECT T1.cid
FROM (SELECT Customer.cid
        FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
        WHERE pnum IS NULL) T1
        INNER JOIN
        (SELECT cid
        FROM Owns INNER JOIN Account USING (accid)
        WHERE balance>15000) T2
        ON T1.cid=T2.cid;
```

Για το τελικό αποτέλεσμα κάνουμε join με τον πίνακα Customer:

(I) Χρησιμοποιώντας το (a) και join με IN:

```
SELECT Customer.*
FROM Customer
WHERE cid IN
        (SELECT cid
        FROM Owns INNER JOIN Account USING (accid)
        WHERE balance>15000 AND cid IN
                (SELECT Customer.cid
                FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
                WHERE pnum IS NULL));
```

(II) Χρησιμοποιώντας το (a) και join με INNER JOIN:

```
SELECT Customer.*
FROM Customer
        INNER JOIN
        (SELECT cid
        FROM Owns INNER JOIN Account USING (accid)
        WHERE balance>15000 AND cid IN
                (SELECT Customer.cid
                FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
                WHERE pnum IS NULL)) T1
        ON Customer.cid=T1.cid;
```

(III) Χρησιμοποιώντας το (b) και join με IN:

```
SELECT Customer.*
FROM Customer
```

```
WHERE cid IN
        (SELECT T1.cid
        FROM (SELECT Customer.cid
                FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
                WHERE pnum IS NULL) T1
                INNER JOIN
                (SELECT cid
                FROM Owns INNER JOIN Account USING (accid)
                WHERE balance>15000) T2
                ON T1.cid=T2.cid);
```

(IV) Χρησιμοποιώντας το (b) και join με INNER JOIN:

```
SELECT Customer.*
FROM Customer
        INNER JOIN
        (SELECT T1.cid
        FROM (SELECT Customer.cid
                FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
                WHERE pnum IS NULL) T1
                INNER JOIN
                (SELECT cid
                FROM Owns INNER JOIN Account USING (accid)
                WHERE balance>15000) T2
                ON T1.cid=T2.cid) T3
        ON T3.cid=Customer.cid;
```

(V) Αλλάζοντας το (a) έτσι ώστε να επιστρέφεται το ζητούμενο:

```
SELECT Customer.*
FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
WHERE pnum IS NULL AND Customer.cid IN
        (SELECT cid
        FROM Owns INNER JOIN Account USING (accid)
        WHERE balance>15000);
```

(VI) Αλλάζοντας το (b) έτσι ώστε να επιστρέφεται το ζητούμενο:

```
SELECT T1.*
FROM (SELECT Customer.*
        FROM Customer LEFT JOIN Phones ON Customer.cid=Phones.cid
                                                AND pnum LIKE '69%'
        WHERE pnum IS NULL) T1
        INNER JOIN
        (SELECT cid
        FROM Owns INNER JOIN Account USING (accid)
        WHERE balance>15000) T2
        ON T1.cid=T2.cid;
```

(VII) Χωρίς χρήση εμφωλευμένων ερωτημάτων:

```
SELECT Customer.*
FROM Customer
        INNER JOIN Owns USING (cid)
        INNER JOIN Account USING (accid)
        LEFT JOIN Phones ON Customer.cid=Phones.cid AND pnum LIKE '69%'
```

WHERE pnum IS NULL AND balance>15000;

Τα ερωτήματα (I) εώς (IV) είναι χαρακτηριστικά του τρόπου με τον οποίο μπορούμε να δομήσουμε σταδιακά ένα ερώτημα χρησιμοποιώντας απευθείας αποτελέσματα υποερωτημάτων. Τα (V) εώς (VII) παρουσιάζουν πιο συμπτυγμένες queries. Τα (V) και (VI) που χρησιμοποιούν εμφωλευμένα είναι πιο κοντά σε σχέση με το (VII) στον αρχικό τρόπο σκέψης κατά τον οποίο αναλύσαμε το αρχικό ερώτημα σε υποερωτήματα.

Συγκρίνοντας κανείς στο επίπεδο της λογικής εκτέλεσης τα ερωτήματα (VI) και (VII) παρατηρεί κανείς ότι στο (VII) είναι πιο πιθανό να οδηγήσει σε μεγαλύτερο πίνακα μετά τα joins σε σχέση με το (VI). Ο λόγος είναι γιατί στον T2 του (VI) υπάρχει το κριτήριο balance>15000, επομένως ο T2 δυνητικά θα περιέχει λιγότερες εγγραφές σε σχέση με το inner join των Owns και Account. Κάτι τέτοιο ενδέχεται (θεωρητικά) να μειώσει το χρόνο εκτέλεσης των ερωτημάτων. Η τεχνική αυτή κατά την οποία πρώτα εφαρμόζονται οι συνθήκες που αλλιώς θα υπήρχαν στο WHERE και στη συνέχεια γίνονται join οι πίνακες ονομάζεται select before join (SBJ). Σε αντίθεση, η συνηθισμένη μορφή μιας query που ενέχει όλα τα φιλτραρίσματα να γίνονται στο WHERE αφού έχουν συνενωθεί οι πίνακες ονομάζεται projection selection join query (PSJ). Θεωρείστε το ακόλουθο (πιο απλό) παράδειγμα.

Ερώτημα 8. Βρες τους πελάτες που το afm τους τελειώνει σε 9 και έχουν τουλάχιστον ένα λογαριασμό με υπόλοιπο μεγαλύτερο των 15000.

(I) PSJ μορφή:

```
SELECT Customer.*
FROM Customer, Owns, Account
WHERE Customer.cid=Owns.cid AND
        Owns.accid=Account.accid AND
        afm LIKE '%9' AND
        balance>15000;
```

ή

```
SELECT Customer.*
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
WHERE afm LIKE '%9' AND balance>15000;
```

(II) SBJ μορφή:

```
SELECT T1.*
FROM (SELECT * FROM Customer WHERE afm LIKE '%9') T1
        INNER JOIN Owns ON T1.cid=Owns.cid
        INNER JOIN
        (SELECT * FROM Account WHERE balance>15000) T2
        ON Owns.accid=T2.accid;
```

Έστω ότι υπάρχουν 100.000 εγγραφές στο Customer, 200.000 εγγραφές στα Owns και Account και ότι μόνο το 20% των λογαριασμών έχουν υπόλοιπο πάνω από 15.000. Αγνοώντας λοιπά κριτήρια στο (I) οι τρεις πίνακες που θα συνενωθούν θα έχουν μέγεθος 100.000, 200.000 και 200.000. Με την πιο απλή εκτέλεση των joins, χρησιμοποιώντας nested loops χωρίς ευρετήρια ή οτιδήποτε άλλο θα επιτάχυνε την εκτέλεση θα έχουμε:

Βήμα	Πλήθος ελέγχων	Μέγεθος αποτελέσματος
1ο join	100.000x200.000	200.000
2ο join	200.000x200.000	200.000
WHERE	200.000	~4.000
Συνολικά:	60.000.200.000	~404.000

Αντίθετα στο (II):

Βήμα	Πλήθος ελέγχων	Μέγεθος αποτελέσματος
WHERE T1	100.000	~10.000
1ο join	10.000x200.000	~20.000
WHERE T2	200.000	40.000
2ο join	~20.000x40.000	~4.000
Συνολικά:	2.800.300.000	~74.000

Όπως γίνεται αντιληπτό το (II) αναμένουμε να είναι σαφώς πιο γρήγορο σε σχέση με το (I) θεωρητικά. Στην πράξη όπως μπορούμε να δούμε με τη χρήση της EXPLAIN ο optimizer της MySQL5.7 θα εκτελέσει με ακριβώς τον ίδιο τρόπο τόσο το (I) όσο και το (II).

8.1.6 Ζητήματα βελτιστοποίησης στην MySQL5.7

Τα εμφωλευμένα ερωτήματα παρέχουν έναν πιο εύκολο τρόπο να συντάσσουμε queries καθώς μπορούμε να δομήσουμε μια query βήμα προς βήμα σύμφωνα με τον τρόπο που αναλύουμε λογικά ένα σύνθετο ερώτημα σε απλούστερα. Όσον αφορά στην εκτέλεση των υποερωτημάτων (είτε στο WHERE, είτε στο FROM) η MySQL5.7 ακολουθεί κάποιους κανόνες που περιγράφονται στο Κεφ. 8.2.1.18 του εγχειριδίου. Στη συνέχεια συνοψίζουμε τη γενική εικόνα.

Ένα υποερώτημα στο WHERE που χρησιμοποιεί IN ή =ANY:
- Μπορεί να εκτελεστεί ως semi-join.
- Μπορεί να υλοποιηθεί παραγόμενος πίνακας.
- Μπορεί να μετατραπεί σε EXISTS.

Ένα υποερώτημα στο WHERE που χρησιμοποιεί NOT IN ή <>ALL:
- Μπορεί να υλοποιηθεί παραγόμενος πίνακας.
- Μπορεί να μετατραπεί σε EXISTS.

Ένα υποερώτημα (derived table) στο FROM:
- Μπορεί να ενσωματωθεί ενός (merging) στο εξωτερικό ερώτημα.
- Μπορεί να υλοποιηθεί παραγόμενος πίνακας.

Στην πράξη της ημισυνένωσης (semi-join) ενός πίνακα T1 με έναν T2 ζητούμε να επιστραφούν οι εγγραφές του T1 που συνδυάζονται με μία τουλάχιστον εγγραφή του T2. Η εκτέλεση ενός semi-join μπορεί να γίνει με INNER JOIN. Για παράδειγμα έστω ότι θέλουμε το semi-join του Customer με τον Phones.

(I) Με INNER JOIN και DISTINCT:
```
SELECT DISTINCT Customer.*
FROM Customer INNER JOIN Phones ON Customer.cid=Phones.cid;
```

Ο παραπάνω τρόπος εκτέλεσης είναι μη αποδοτικός καθώς από το INNER JOIN επιστρέφονται διπλές εγγραφές για κάποιον πελάτη που έχει πολλά τηλέφωνα μόνο και μόνο για να κοπούν με το DISTINCT. Αντίθετα με τη χρήση IN και εμφωλευμένου ερωτήματος μπορούμε να αποφύγουμε το DISTINCT.

(II) Με IN:
SELECT Customer.*
FROM Customer
WHERE cid IN (SELECT cid FROM Phones);

Η MySQL5.7 υποστηρίζει ισχυρές βελτιστοποιήσεις των semi-join που για να μπορεί να τις εκμεταλλευτεί μία query θα πρέπει να είναι στη μορφή εμφωλευμένου με IN ή =ANY και να πληρούνται και κάποιες επιπλέον προϋποθέσεις που περιγράφονται στο Κεφ. 8.2.1.18 οι βασικότερες εκ των οποίων είναι να μη χρησιμοποιείται στο εμφωλευμένο UNION, ο συνδυασμός των ORDER BY με LIMIT και GROUP BY (θα εξηγηθεί στο επόμενο κεφάλαιο). Η μορφή (I) δε μετατρέπεται σε semi-join.

Κατ' αντιστοιχία η αντι-ημισυνένωση (anti semi-join ή αλλιώς left anti semi-join) δύο πινάκων T1 και T2 επιστρέφει τις εγγραφές του T1 που δε συνδυάζονται με καμία εγγραφή του T2. Και εδώ ο πιο ενδεδειγμένος τρόπος σύνταξης είναι με εμφωλευμένο που χρησιμοποιεί NOT IN ή NOT EXISTS. Κυρίως όμως καλύτερα να προτιμούμε το NOT EXISTS εξαιτίας της περίπτωσης των NULL τιμών.

Η στρατηγική της υλοποίησης του παραγόμενου πίνακα συνήθως είναι πιο χρονοβόρα. Είναι σημαντικό να τονιστεί ότι μέχρι την έκδοση 5.7.5 της MySQL η υλοποίηση παραγόμενων πινάκων σε εμφωλευμένα στο FROM δε διατηρούσε τα όποια ευρετήρια υπήρχαν. Από την 5.7.6 μέχρι και την 5.7.9 που ήταν η τρέχουσα τη στιγμή που γράφονταν αυτό το κείμενο, το εγχειρίδιο αναφέρει ότι ο optimizer *μπορεί* να δημιουργήσει ευρετήριο για τον υλοποιημένο πίνακα (χωρίς αυτό να είναι βέβαιο).

Το παραπάνω σημαίνει ότι συνήθως ερωτήματα με εμφωλευμένα στο FROM που δεν ενσωματώνονται στο εξωτερικό ερώτημα δηλ. δεν μπορεί ο optimizer να ξαναγράψει το ερώτημα χωρίς χρήση εμφωλευμένου, με μεγάλη πιθανότητα θα εκτελούνται πιο αργά σε εκδόσεις προγενέστερες της 5.7.5 και ενδέχεται να είναι πιο αργά από την έκδοση 5.7.6 όσον αφορά στη μεγαλύτερη γκάμα ερωτημάτων.

Γενικά όσον αφορά στην MySQL θα πρέπει να έχουμε υπόψη μας τα εξής:

- Η MySQL5.7 έχει πολλές βελτιστοποιήσεις όσον αφορά στην εκτέλεση joins. Κατά συνέπεια τις περισσότερες φορές ερωτήματα που χρησιμοποιούν joins έναντι εμφωλευμένων τις περισσότερες φορές θα είναι πιο γρήγορα.
- Ερωτήματα που είναι semi-joins και anti semi-joins καλό είναι να γράφονται ως εμφωλευμένα.
- Ερωτήματα εμφωλευμένα με χρήση μόνο τελεστών σύγκρισης πχ. = οδηγούν σε γρηγορότερη εκτέλεση σε σχέση με εμφωλευμένα που χρησιμοποιούν και ANY/ALL.
- Η χρήση SBJ τακτικής με εμφωλευμένα στο FROM ενδέχεται να μην έχει καλή απόδοση καθώς ο optimizer ούτως ή άλλως λαμβάνει υπόψη την τακτική του να φιλτράρει κανείς τους πίνακες πριν τους κάνει join. Εκτελεί όμως και άλλες βελτιστοποιήσεις, ενώ με την SBJ σε περίπτωση που δεν μπορεί να ξαναγραφεί η query πάντα θα υλοποιείται ο παραγόμενος πίνακας (derived table materialization).

Τα προαναφερθέντα δε μειώνουν ούτε στο ελάχιστο την αξία της χρήσης υποερωτημάτων. Ο βασικότερος όλων των κανόνων (κυρίως για αρχάριους) είναι ο ακόλουθος: *Πρώτα γράφουμε ένα σωστό ερώτημα και στη συνέχεια αν η απόδοσή του δεν είναι ικανοποιητική προσπαθούμε να το βελτιστοποιήσουμε.*

Η διαδικασία βελτιστοποίησης ενός ερωτήματος ξεφεύγει από το σκοπό του συγγράμματος παρόλα αυτά κάποια κεντρικά σημεία είναι τα ακόλουθα:

- Εξετάζουμε το πλάνο εκτέλεσης χρησιμοποιώντας την EXPLAIN.

- Προσπαθούμε να ξαναγράψουμε το ερώτημα χρησιμοποιώντας semi-joins όπου είναι δυνατό.
- Προσπαθούμε να μετατρέψουμε εμφωλευμένα ερωτήματα σε αντίστοιχα που χρησιμοποιούν μόνο joins.

8.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στη Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: **Σετ ερωτήσεων Α (Εμφωλευμένα ερωτήματα στο WHERE)** Γράψτε τα ακόλουθα ερωτήματα σε SQL χρησιμοποιώντας εμφωλεύσεις: 1. Βρείτε τα ονόματα των φοιτητών που δεν έχουν κινητά τηλέφωνα 2. Βρείτε τα ονόματα των φοιτητών που πέρασαν το μάθημα 'Baseis Dedomenon' 3. Βρείτε τους κωδικούς των μαθημάτων που δεν έχουν περαστεί από κανένα φοιτητή **Σετ ερωτήσεων Β (Μετονομασία και εμφωλευμένα στο FROM)** 1. Εμφανίστε τους τίτλους των μαθημάτων μαζί με τους αντίστοιχους κωδικούς τυχόν προαπαιτούμενων που έχουν. 2. Βρείτε τα ονόματα των φοιτητών που δεν έχουν γράψει το μεγαλύτερο βαθμό στο μάθημα 'Domes Dedomenon'. **Σετ ερωτήσεων Γ (Πράξεις Συνόλων IN/NOT IN)** 1. Να βρεθούν οι φοιτητές που έχουν δώσει μάθημα και του καθηγητή 'Antonis' και του καθηγητή 'Lampsas'. 2. Να βρεθούν οι φοιτητές που έχουν δώσει μάθημα του καθηγητή 'Antonis' αλλά όχι του καθηγητή 'Lampsas'. 3. Βρείτε τα ονόματα των φοιτητών που έχουν γράψει το μεγαλύτερο βαθμό στο μάθημα 'Baseis Dedomenon'. **Σετ ερωτήσεων Δ (EXISTS)** 1. Βρείτε τα επώνυμα των φοιτητών που έχουν δώσει το μάθημα με κωδικό 4.
Ζητούμενα:	Εμφωλευμένα ερωτήματα στο WHERE, μετονομασία, συσχετισμένα εμφωλευμένα ερωτήματα, πράξεις συνόλων IN/NOT IN, EXISTS.

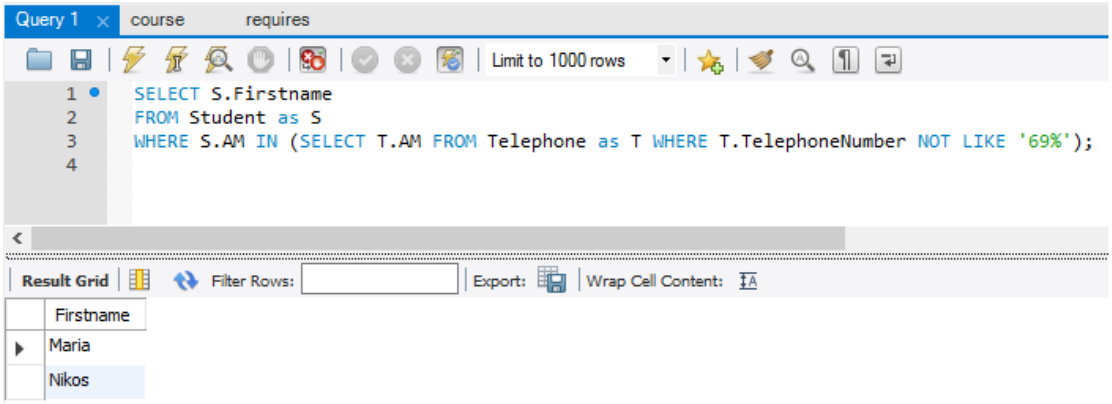
Σετ ερωτήσεων Α (Εμφωλευμένα ερωτήματα στο WHERE)

Ζητούμενο Α1: Βρείτε τα ονόματα των φοιτητών που δεν έχουν κινητά τηλέφωνα.

Δίνουμε:

```
SELECT S.Firstname
FROM Student AS S
WHERE S.AM IN
        (SELECT T.AM
        FROM Telephone AS T
        WHERE T.TelephoneNumber NOT LIKE '69%');
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.1.



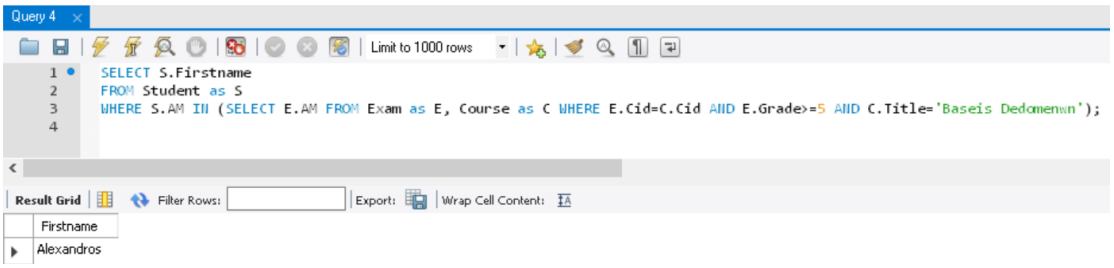
Εικόνα 8.1: *Συνένωση πινάκων με εμφωλευμένο στο WHERE.*

Ζητούμενο Α2: Βρείτε τα ονόματα των φοιτητών που πέρασαν το μάθημα 'Baseis Dedomenon'.

Δίνουμε:

```
SELECT S.Firstname
FROM Student AS S
WHERE S.AM IN
        (SELECT E.AM
        FROM Exam AS E, Course AS C
        WHERE E.Cid=C.Cid AND
                E.Grade>=5 AND
                C.Title='Baseis Dedomenwn');
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.2.



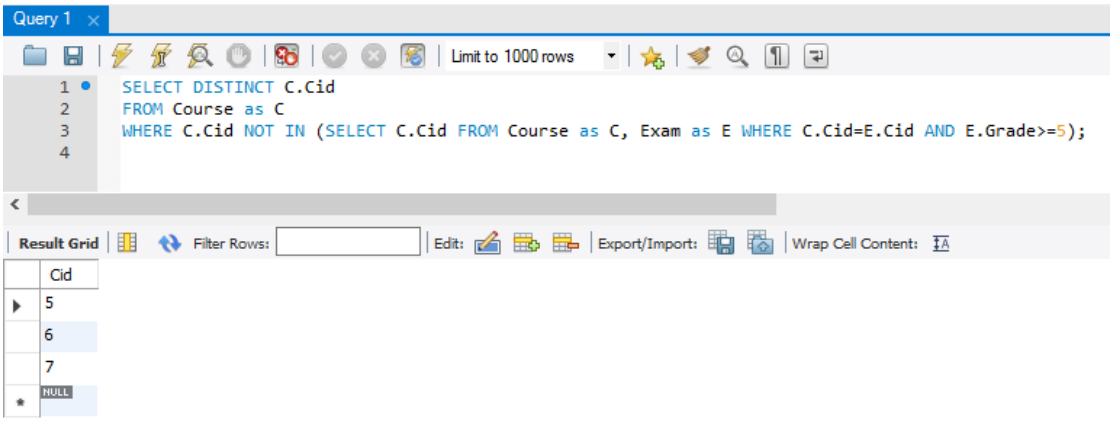
Εικόνα 8.2: *Χρησιμοποιώντας τρεις πίνακες και εμφωλευμένο ερώτημα.*

Ζητούμενο Α3: Βρείτε τους κωδικούς των μαθημάτων που δεν έχουν περαστεί από κανένα φοιτητή.

Δίνουμε:

```
SELECT DISTINCT C.Cid
FROM Course AS C
WHERE C.Cid NOT IN
        (SELECT C.Cid
        FROM Course AS C, Exam AS E
        WHERE C.Cid=E.Cid AND E.Grade>=5);
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.3.



Εικόνα 8.3: *Παράδειγμα αφαίρεσης.*

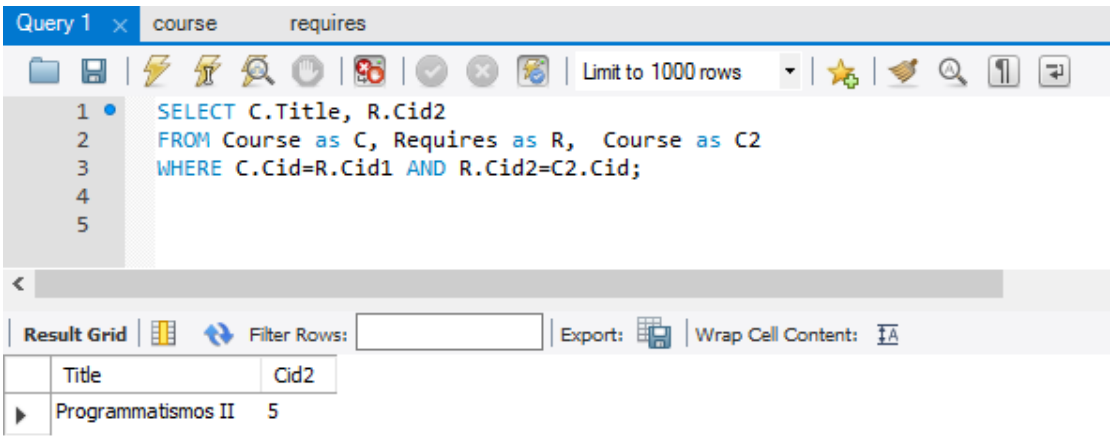
Σετ ερωτήσεων Β (Μετονομασία και Εμφωλευμένα στο FROM)

Ζητούμενο Β1: Εμφανίστε τους τίτλους των μαθημάτων μαζί με τους αντίστοιχους κωδικούς τυχόν προαπαιτούμενων που έχουν.

Δίνουμε:

```
SELECT C.Title, R.Cid2
FROM Course AS C, Requires AS R, Course AS C2
WHERE C.Cid=R.Cid1 AND R.Cid2=C2.Cid;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.4.



Εικόνα 8.4: *Μετονομασία πινάκων με χρήση του AS.*

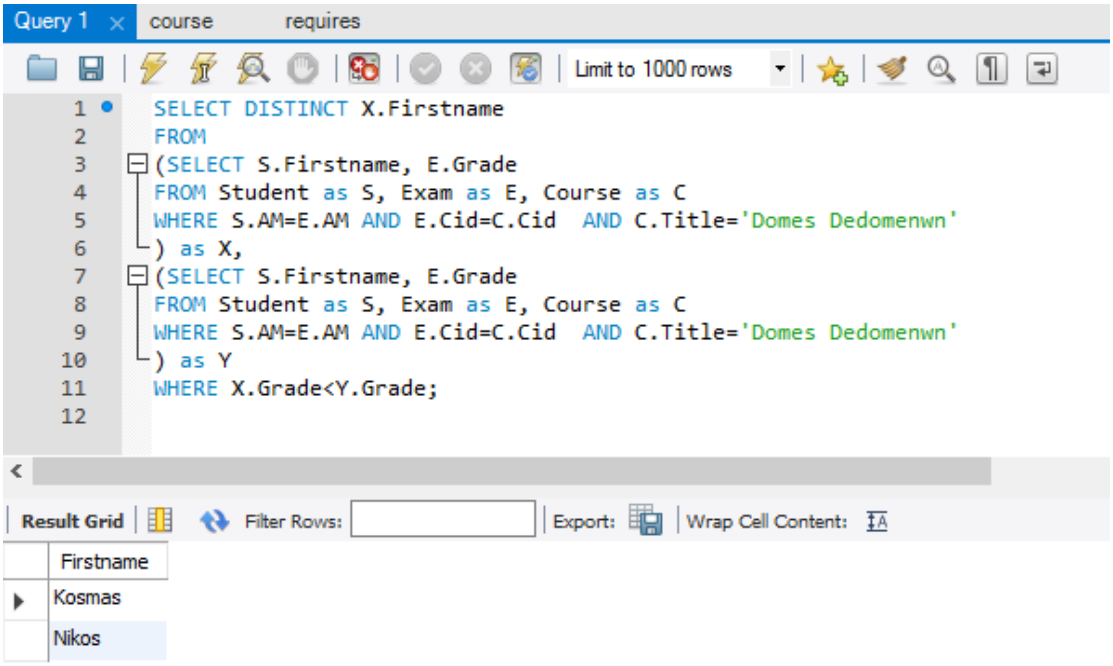
Ζητούμενο Β2: Βρείτε τα ονόματα των φοιτητών που δεν έχουν γράψει το μεγαλύτερο βαθμό στο μάθημα ‘Domes Dedomenon’.

Δίνουμε:

```
SELECT DISTINCT X.Firstname
FROM
```

```
        (SELECT S.Firstname, E.Grade
        FROM Student AS S, Exam AS E, Course AS C
        WHERE S.AM=E.AM AND E.Cid=C.Cid  AND C.Title='Domes Dedomenwn'
        ) AS X,
        (SELECT S.Firstname, E.Grade
        FROM Student AS S, Exam AS E, Course AS C
        WHERE S.AM=E.AM AND E.Cid=C.Cid  AND C.Title='Domes Dedomenwn'
        ) AS Y
WHERE X.Grade<Y.Grade;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.5.



Εικόνα 8.5: *Εμφωλευμένα στο FROM.*

Σετ ερωτήσεων Γ (Πράξεις Συνόλων IN/NOT IN)

Ζητούμενο Γ1: Να βρεθούν τα ονόματα των φοιτητών που έχουν δώσει μάθημα και του καθηγητή 'Tzimas' και της καθηγήτριας 'Andriopoulou'.

Δίνουμε:

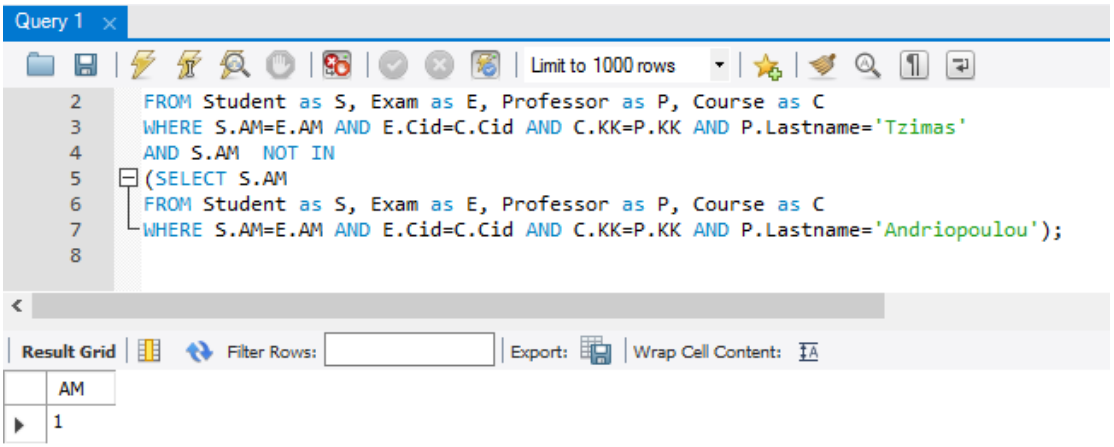
```
SELECT X.AM
FROM
        (SELECT S.AM
        FROM Student AS S, Exam AS E, Course AS C, Professor AS P
        WHERE S.AM=E.AM AND E.Cid=C.Cid AND
        C.KK=P.KK AND P.Lastname='Tzimas'
        ) AS X,
        (SELECT E.AM
        FROM Student AS S, Exam AS E, Course AS C, Professor AS P
        WHERE S.AM=E.AM AND E.Cid=C.Cid AND
        C.KK=P.KK AND P.Lastname='Andriopoulou'
        ) as Y
WHERE X.AM=Y.AM;
```

Ζητούμενο Γ2: Να βρεθούν οι φοιτητές που έχουν δώσει μάθημα του καθηγητή 'Tzimas' αλλά όχι της καθηγήτριας 'Andriopoulou'.

Δίνουμε:

```
SELECT S.AM
FROM Student AS S, Exam AS E, Professor AS P, Course AS C
WHERE S.AM=E.AM AND E.Cid=C.Cid AND
C.KK=P.KK AND P.Lastname='Tzimas' AND
S.AM  NOT IN
        (SELECT S.AM
        FROM Student AS S, Exam AS E, Professor AS P, Course AS C
        WHERE S.AM=E.AM AND E.Cid=C.Cid AND
        C.KK=P.KK AND P.Lastname='Andriopoulou');
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.6.



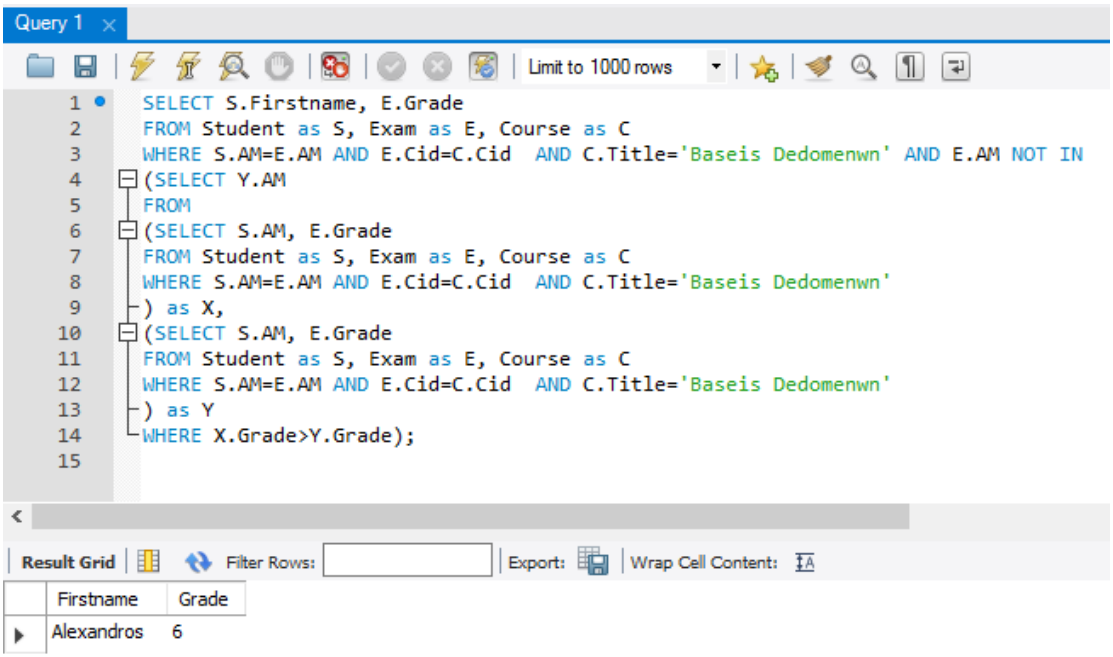
Εικόνα 8.6: *Αφαίρεση με χρήση NOT IN.*

Ζητούμενο Γ3: Βρείτε τα ονόματα των φοιτητών που έχουν γράψει μεγαλύτερο βαθμό στο μάθημα ‘Baseis Dedomenwn’ από άλλους συμφοιτητές τους.

Δίνουμε:

```
SELECT S.Firstname, E.Grade
FROM Student AS S, Exam AS E, Course AS C
WHERE S.AM=E.AM AND
E.Cid=C.Cid  AND
C.Title='Baseis Dedomenwn' AND
E.AM NOT IN
        (SELECT Y.AM
        FROM
                (SELECT S.AM, E.Grade
                FROM Student AS S, Exam AS E, Course AS C
                WHERE S.AM=E.AM AND
                E.Cid=C.Cid  AND
                C.Title='Baseis Dedomenwn'
                ) AS X,
                (SELECT S.AM, E.Grade
                FROM Student AS S, Exam AS E, Course AS C
                WHERE S.AM=E.AM AND
                E.Cid=C.Cid  AND
                C.Title='Baseis Dedomenwn'
                ) AS Y
        WHERE X.Grade>Y.Grade);
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.7.



Εικόνα 8.7: *Εύρεση μεγίστου με χρήση αφαίρεσης.*

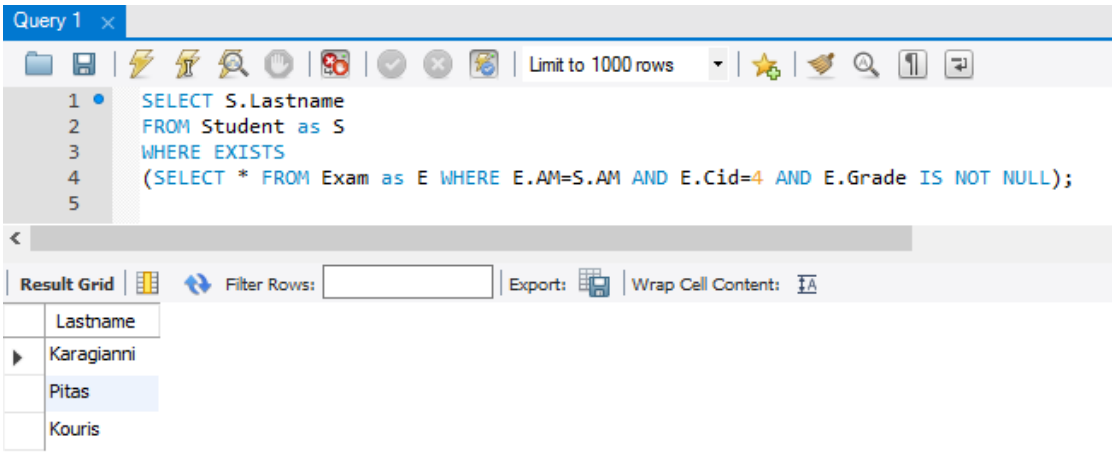
Σετ ερωτήσεων Δ (EXISTS)

Ζητούμενο Δ1: Βρείτε τα επώνυμα των φοιτητών που έχουν δώσει το μάθημα με κωδικό 4.

Δίνουμε:

```
SELECT S.Lastname
FROM Student as S
WHERE EXISTS
        (SELECT *
        FROM Exam AS E
        WHERE E.AM=S.AM AND E.Cid=4 AND E.Grade IS NOT NULL);
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 8.8.



Εικόνα 8.8: *Παράδειγμα χρήσης του EXISTS.*

8.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

5. Βρείτε την επωνυμία των εταιριών που έχουν εργαζόμενους χωρίς καθόλου προϋπηρεσία.
6. Εμφανίστε τα επώνυμα των διευθυντών μαζί με τις εταιρίες που δουλεύουν.
7. Βρείτε τα ονόματα των εργαζομένων που έχουν δουλέψει στην εταιρία ‘Advance’.
8. Βρείτε τους διευθυντές που δουλεύουν σε εταιρίες που έχουν ιδρυθεί μετά το 2000.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2), απαντήστε στα ακόλουθα ερωτήματα:

1. Βρείτε τις ταινίες που οι σκηνοθέτες τους είναι μεγαλύτεροι από 60 ετών.
2. Βρείτε τις ταινίες που τις παρακολούθησαν περισσότεροι από 100 θεατές.
3. Βρείτε τις ταινίες που σκηνοθέτησε ο ‘Steven Spielberg’, αλλά όχι ο ‘Cuiseppe Tornatore’.
4. Βρείτε τις ταινίες με βαθμολογία 5.

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3), απαντήστε στα ακόλουθα ερωτήματα:

1. Βρείτε τα ονόματα των παικτών που δεν έχουν νικήσει ποτέ σε κανένα τουρνουά.
2. Βρείτε τους παίκτες που συμμετείχαν στο τουρνουά της Γαλλίας το 2015.
3. Εμφανίστε τους παίκτες του 2015 που ήταν μεγαλύτεροι από 20 ετών.
4. Βρείτε τους παίκτες που έχουν κερδίσει το ‘Roland Garros’, αλλά όχι το ‘US Open’.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαια 8 και 9.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
Comparison Operators. https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html
IN/NOT IN Operator. http://www.w3schools.com/sql/sql_in.asp
SUBQUERIES.
http://zetcode.com/databases/mysqltutorial/subqueries/
http://dev.mysql.com/doc/refman/5.7/en/subqueries.html
EXISTS http://dev.mysql.com/doc/refman/5.7/en/exists-and-not-exists-subqueries.html

Κεφάλαιο 9 Συναθροίσεις

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιαστούν ερωτήματα συνάθροισης χρησιμοποιώντας τις βασικές συναρτήσεις MAX, MIN, AVG, SUM και COUNT. Θα παρουσιαστεί η χρήση της ομαδοποίησης εγγραφών με την εντολή GROUP BY και της επιλογής ομάδων με την εντολή HAVING. Τέλος θα γίνει εκτενέστερη αναφορά της εντολής ταξινόμησης αποτελεσμάτων ORDER BY.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- *Η ύλη του Κεφ. 8.*

9.1 Εισαγωγικές Έννοιες

Στη συνέχεια θα παρουσιαστεί μία σύνοψη του τρόπου σύνταξης ερωτημάτων συνάθροισης (aggregation queries). Στα ερωτήματα-παραδείγματα που θα δοθούν θα χρησιμοποιηθούν οι παρακάτω πίνακες με τις αντίστοιχες εγγραφές τους.

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333

Account

accid	balance	dateOfCreation
100	10000	2014-01-01
200	40000	2013-02-01
300	30000	2005-03-10
400	20000	2005-03-10

Owns

cid	accid
1	100
2	200
1	300
1	400

Action

accid	actid	amount	type	dateOfAction
100	1	500	2	2014-01-14
200	1	500	2	2014-01-14
200	2	1000	2	2015-11-11
200	3	2000	1	2015-12-11
300	1	1000	2	2015-11-11
300	2	500	1	2015-12-11

Transfer

accidSource	actidSource	accidDest	actidDest
200	1	100	1
300	1	200	2

9.1.1 Συναρτήσεις συνάθροισης

Όλες οι συναρτήσεις συνάθροισης χρησιμοποιούνται στο SELECT τμήμα ενός ερωτήματος. Παίρνουν ως όρισμα μία έκφραση και επιστρέφουν ως αποτέλεσμα μία τιμή. Η τιμή αυτή υπολογίζεται ως εξής: υπολογίζεται πρώτα η τιμή της έκφρασης για κάθε γραμμή του πίνακα και στη συνέχεια εφαρμόζεται η συνάρτηση που περιγράφεται για το σύνολο των τιμών που υπολογίστηκαν. Αν σε κάποια γραμμή η έκφραση αποτιμείται σε NULL, η συγκεκριμένη γραμμή δε μετέχει στον υπολογισμό.

Οι βασικές συναρτήσεις συνάθροισης αριθμητικών τιμών στην SQL είναι οι εξής:
SUM(έκφραση): επιστρέφει το άθροισμα.
AVG(έκφραση): επιστρέφει το μέσο όρο.
MAX(έκφραση): επιστρέφει τη μέγιστη τιμή.
MIN(έκφραση): επιστρέφει την ελάχιστη τιμή.
STDDEV_POP(έκφραση): επιστρέφει την τυπική απόκλιση.
COUNT(έκφραση): επιστρέφει το πλήθος γραμμών.

Ακολουθούν παραδείγματα. Για περισσότερες πληροφορίες αλλά και πλήρη κατάλογο των υποστηριζόμενων από τη MySQL συναρτήσεων, ο αναγνώστης μπορεί να απευθυνθεί στο Κεφ.12.20.1 του εγχειριδίου της MySQL5.7.

Ερώτημα 1. Βρες το άθροισμα, το μέσο όρο, την τυπική απόκλιση, το μεγαλύτερο και το μικρότερο ποσό χρημάτων στους λογαριασμούς.

```
SELECT SUM(balance) AS sum, AVG(balance) AS avg, STDDEV_POP(balance) AS stddev,
MAX(balance) AS max, MIN(balance) AS min
FROM Account;
```

Επιστρέφει:

sum	avg	stddev	max	min
100000	25000	11180.339887498949	40000	10000

Ερώτημα 2. Βρες το πλήθος των εγγραφών στον Customer για τις οποίες υπάρχουν διευθύνσεις.

```
SELECT COUNT(address)
FROM Customer;
```

Επιστρέφει: 3.

Αν υποθέσουμε το ακόλουθο στιγμιότυπο του πίνακα Customer:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	NULL	Maria	Papantoniou	1967-03-20

τότε θα επιστρέφονταν 2 καθώς η γραμμή με cid=3 έχει NULL στο address.

Για να μετρήσουμε τις γραμμές στον πίνακα Customer (ασχέτως αν σε κάποια πεδία υπάρχουν NULL τιμές) μπορούμε να δώσουμε:

```
SELECT COUNT(*)
FROM Customer;
```

ή

```
SELECT COUNT(TRUE)
FROM Customer;
```

ή

```
SELECT COUNT(1)
FROM Customer;
```

Εκ των τριών η συνηθέστερα χρησιμοποιούμενη μορφή είναι η πρώτη με την οποία στην ουσία λέμε ότι μία γραμμή θα «μετρηθεί» στο αποτέλεσμα αν κάποιο από τα πεδία της είναι διάφορο του NULL.

Ερώτημα 3 (απαλοιφή διπλότυπων). Βρες το πλήθος των διαφορετικών ον/μων των πελατών.

```
SELECT COUNT(DISTINCT name, sname)
FROM Customer;
```

Το ερώτημα θα επιστρέψει το πλήθος των εγγραφών με διαφορετικά ον/μα (στο παράδειγμα 3). Αν δίναμε:

```
SELECT COUNT(DISTINCT sname)
FROM Customer;
```

θα λαμβάναμε ως απάντηση το 2, καθώς υπάρχουν δύο εγγραφές με ξεχωριστά επώνυμα.

9.1.2 Χρήση ομαδοποίησης

Ας υποθέσουμε ότι θέλουμε να γραφεί query για το παρακάτω ερώτημα:

Ερώτημα 4. Βρες το σύνολο των χρημάτων που έχει καταθέσει κάθε πελάτης.

Έστω ότι επιχειρούμε να δώσουμε την ακόλουθη query:

```
SELECT cid, SUM(balance)
FROM Owns INNER JOIN Account USING (accid);
```

Η απάντηση που παίρνουμε είναι η εξής:

cid	SUM(balance)
1	100000

Αυτό που συμβαίνει είναι ότι μετά την εκτέλεση του FROM ο πίνακας που προκύπτει είναι ο εξής:

accid	cid	balance	dateOfCreation
100	1	10000	2014-01-01
200	2	40000	2013-02-01
300	1	30000	2005-03-10
400	1	20000	2005-03-10

Σε αυτόν τον πίνακα υπολογίζεται το SUM(balance) για όλες τις εγγραφές και επιστρέφεται ως αποτέλεσμα (100000) μαζί με το cid της πρώτης εγγραφής. Γίνεται λοιπόν αντιληπτό ότι χρειάζεται ένας τρόπος να ομαδοποιηθούν οι εγγραφές (ως προς cid) και να υπολογιστεί το SUM ξεχωριστά για κάθε ομάδα. Αυτό γίνεται με χρήση του GROUP BY ως εξής:

```
SELECT cid, SUM(balance)
FROM Owns INNER JOIN Account USING (accid)
GROUP BY cid;
```

Η απάντηση που παίρνουμε είναι η εξής:

cid	SUM(balance)
1	60000
2	40000

Στο παραπάνω ερώτημα αφού φτιαχτεί ο τελικός πίνακας του FROM οι εγγραφές ομαδοποιούνται βάσει της τιμής του cid. Έτσι θα υπάρξουν δύο ομάδες η μία θα έχει τις εγγραφές που έχουν cid=1 (3 εγγραφές) και η άλλη αυτές που έχουν cid=2 (1 εγγραφή). Το SELECT θα επιστρέψει ένα αποτέλεσμα για κάθε ομάδα.

Ερώτημα 5. Βρες το σύνολο των χρημάτων που κινήθηκαν ανά λογαριασμό για τους λογαριασμούς που το άθροισμα των κινήσεών τους είναι >1000.

Ένα καταφανές λάθος θα ήταν να δίναμε:

```
SELECT accid, SUM(amount)
FROM Action
WHERE amount>1000
GROUP BY accid;
```

Με τον πίνακα Action ως ακολούθως:

Action

accid	actid	amount	type	dateOfAction
100	1	500	2	2014-01-14
200	1	500	2	2014-01-14
200	2	1000	2	2015-11-11
200	3	2000	1	2015-12-11
300	1	1000	2	2015-11-11
300	2	500	1	2015-12-11

η απάντηση που παίρνουμε είναι η εξής:

accid	SUM(amount)
200	2000

Ο λόγος είναι γιατί από τον πίνακα Account φιλτράρονται οι εγγραφές σύμφωνα με το WHERE και ως εκ τούτου μένει μόνο μία εγγραφή (η 4η). Στη συνέχεια εφαρμόζεται το GROUP BY και κατόπιν το SELECT.

Το προηγούμενο SQL ερώτημα ήταν εμφανώς λάθος καθώς η εκφώνηση ζητάει το άθροισμα και όχι η κάθε μία πράξη να είναι άνω των 1000. Αν γράψουμε:

```
SELECT accid, SUM(amount)
FROM Action
WHERE SUM(amount)>1000
GROUP BY accid;
```

θα λάβουμε ως απάντηση το εξής μήνυμα λάθους:
ERROR 1111 (HY000): Invalid use of group function
Ο λόγος είναι γιατί η συνθήκη του WHERE εφαρμόζεται ανά εγγραφή και ως εκ τούτου η χρήση του SUM(amount) δεν είναι ορθή καθώς δεν μπορεί να καθοριστεί η ομάδα εγγραφών στην οποία θα επενεργήσει. Χρειάζεται λοιπόν ένας τρόπος να μπορούμε να δηλώνουμε σε μία SQL query κριτήρια βάσει των οποίων μπορεί να φιλτραριστούν ολόκληρες ομάδες εγγραφών. Τον τρόπο αυτό προσφέρει η εντολή HAVING. Στο παράδειγμα η ορθή query θα ήταν ως εξής:

```
SELECT accid, SUM(amount)
FROM Action
GROUP BY accid
HAVING SUM(amount)>1000;
```

και θα επέστρεφε:

accid	SUM(amount)
200	3500
300	1500

καθώς η ομάδα με accid=100 δεν πληροί το HAVING αφού έχει σύνολο χρηματικών κινήσεων 500.

Συνοψίζοντας, η βασική μορφή ενός ερωτήματος συνάθροισης (aggregation query) έχει ως ακολούθως:

```
SELECT f1, f2, .., fn, a1, a2,..,am
FROM T1, T2, …. Tk
WHERE d
GROUP BY a1, a2, ..,am
HAVING p;
```

Η παραπάνω query εκτελείται (συντακτικά) ως εξής: πρώτα σχηματίζεται ο τελικός πίνακας στο FROM. Στη συνέχεια οι εγγραφές του πίνακα που προκύπτει φιλτράρονται βάσει της λογικής συνθήκης d στο WHERE. Οι εγγραφές που απομένουν ομαδοποιούνται βάσει των πεδίων a1,..,am. Από τις ομάδες που προκύπτουν κρατούνται οι ομάδες που πληρούν τη λογική συνθήκη p του HAVING. Για κάθε μία από τις ομάδες η SELECT επιστρέφει μία εγγραφή που έχει τα αποτελέσματα των f1,.,fn συναθροίσεων και τις αντίστοιχες τιμές στα πεδία ομαδοποίησης. Σημειώνουμε ότι δεν είναι απαραίτητο αν και είναι σύνηθες στο SELECT να επιστρέφονται τα πεδία ομαδοποίησης μαζί με τις συναθροίσεις. Επίσης η συνθήκη στο HAVING τυπικά πρέπει να έχει συγκρίσεις που εμπλέκουν συναθροίσεις. Για παράδειγμα, το HAVING accid<300 αν και επιτρεπτό δεν είναι δόκιμο καθώς η συνθήκη accid<300 μπορεί να υπολογιστεί στο επίπεδο των εγγραφών (ως μέρος του WHERE) και όχι των ομάδων.

9.1.3 Εμφωλευμένα ερωτήματα και συναθροίσεις

Στη συνέχεια δείχνουμε μερικά παραδείγματα ερωτημάτων στα οποία θέλουμε να βρούμε εγγραφές ενός πίνακα που πληρούν κριτήριο που εμπλέκει συναθροίσεις.

Ερώτημα 6. Βρες τους λογαριασμούς με υπόλοιπα μεγαλύτερα του μέσου όρου υπολοίπων.

1ος τρόπος
Βρίσκουμε το μέσο όρο των υπολοίπων σε εμφωλευμένο και στη συνέχεια προσθέτουμε αυτό το πεδίο στον πίνακα Account (με cross join). Επιλέγουμε τις εγγραφές που έχουν υπόλοιπο μεγαλύτερο του μέσου όρου.

```
SELECT Account.*
FROM  Account CROSS JOIN (SELECT AVG(balance) AS avb
                          FROM Account) T1
WHERE Account.balance>T1.avb;
```

2ος τρόπος
Με εμφωλευμένο στο WHERE.

```
SELECT *
FROM  Account
WHERE Account.balance > (SELECT AVG(balance) FROM Account);
```

Ερώτημα 7. Βρες τους λογαριασμούς που έχουν σύνολο χρηματικών κινήσεων μεγαλύτερο ή ίσο με το μέσο όρο.

```
SELECT accid
FROM Action
GROUP BY accid
HAVING SUM(amount) >= (SELECT AVG(amount) FROM Action);
```

Επιστρέφει:

accid
200
300

Σημείωση: Εάν γράφαμε την προηγούμενη query ως εξής:

```
SELECT accid
FROM Action
GROUP BY accid
HAVING SUM(amount) >= AVG(amount);
```

θα επιστρέφονταν:

accid
100
200
300

καθώς το AVG υπολογίζεται για κάθε ομάδα ξεχωριστά, επομένως είναι φυσικό όλοι οι λογαριασμοί να πληρούν τη συνθήκη ότι το άθροισμα των κινήσεών τους είναι μεγαλύτερο ή ίσο από το μέσο όρο τους.

9.1.4 Χρήση των ORDER BY και LIMIT

Μπορούμε να επιστρέψουμε ταξινομημένα τα αποτελέσματα ενός ερωτήματος με χρήση της ORDER BY. Η σύνταξη είναι:
ORDER BY <πεδίο> [ASC|DESC], ..

Με άλλα λόγια μετά την ORDER BY ακολουθεί ένα ή περισσότερα πεδία στα οποία θα γίνει η ταξινόμηση των τελικών αποτελεσμάτων χωρισμένα με κόμμα. Σε κάθε πεδίο μπορεί προαιρετικά να επιλεγεί η ταξινόμηση να γίνει σε αύξουσα σειρά (ASC) ή φθίνουσα (DESC).
Σημείωση: Η χρήση της ORDER BY εμπλέκει ταξινόμηση που στη γενικότερη περίπτωση (εξωτερική ταξινόμηση) είναι αρκετά ακριβή πράξη. Κατά συνέπεια η χρήση της πρέπει να γίνεται μόνο όταν είναι απολύτως επιβεβλημένη. Περισσότερες πληροφορίες περιλαμβάνονται στο Κεφ. 9.2.1.15 του εγχειριδίου της MySQL5.7.

Αν δεν επιλεγεί κανένας προσδιοριστής η σειρά ταξινόμησης στο συγκεκριμένο πεδίο θα είναι αύξουσα.
Είναι δυνατόν να περιορίσουμε το πλήθος των εγγραφών που επιστρέφονται στο αποτέλεσμα με χρήση της LIMIT. Η σύνταξη είναι:
LIMIT [<αριθμόςΠαραλειπόμενων>,] αριθμόςΕμφανιζόμενων
Για παράδειγμα δίνοντας: LIMIT 2, 5 σημαίνει ότι από τα αποτελέσματα πρέπει να παραληφθούν οι 2 πρώτες εγγραφές και να επιστραφούν οι επόμενες 5.

Σημείωση: Η χρήση της LIMIT συνήθως οδηγεί σε γρηγορότερη εκτέλεση ερωτημάτων καθώς στη γενικότερη περίπτωση δε χρειάζεται να υπολογιστεί όλο το αποτέλεσμα ενός ερωτήματος. Αντίθετα όταν έχουν υπολογιστεί στο αποτέλεσμα τόσες εγγραφές όσες προδιαγράφονται στο LIMIT η εκτέλεση του ερωτήματος μπορεί να περαιωθεί. Περισσότερες πληροφορίες περιλαμβάνονται στο Κεφ. 9.2.1.19 του εγχειριδίου της MySQL5.7.
Σημείωση 2: Στη σύνταξη SQL ερωτήματος το ORDER BY έπεται του HAVING και το LIMIT του ORDER BY.

Παράδειγμα 1:

```
SELECT accid, SUM(amount) AS sum
FROM Action
WHERE amount<2000
GROUP BY accid
HAVING COUNT(*)>1
ORDER BY accid DESC
LIMIT 1;
```

Επιστρέφει:

accid	sum
300	1500

Παράδειγμα 2:

```
SELECT accid, SUM(amount) AS sum
FROM Action
GROUP BY accid
ORDER BY COUNT(*) DESC
LIMIT 2,1;
```

Το αποτέλεσμα πριν το LIMIT είναι (ταξινομημένο ως προς πλήθος εγγραφών ανά ομάδα):

accid	sum
200	3500
300	1500
100	500

Το τελικό αποτέλεσμα που θα επιστραφεί μετά το LIMIT:

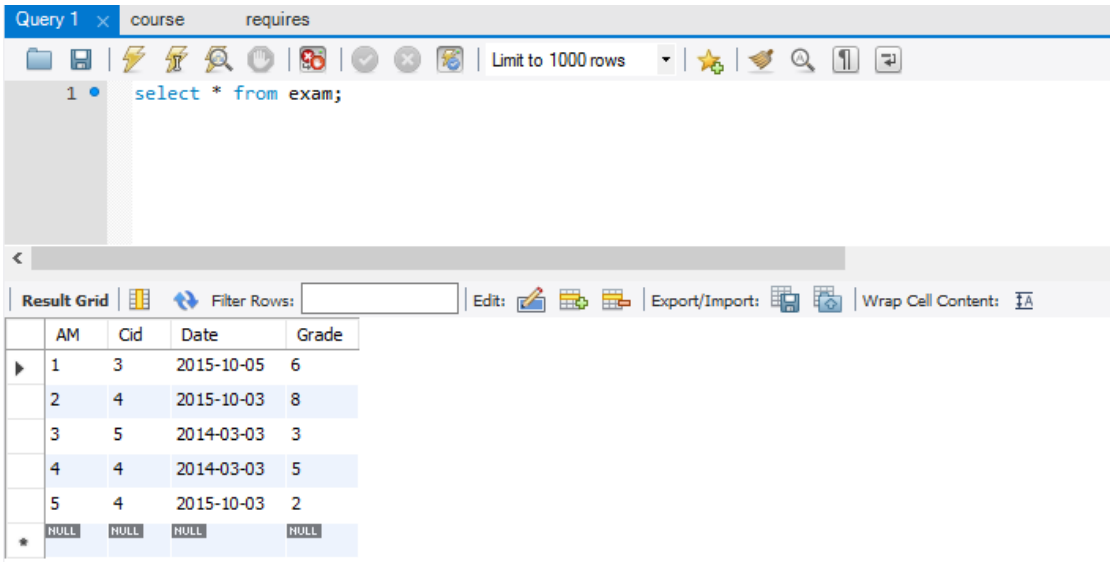
accid	sum
100	500

9.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 1. Εμφανίστε τη μέση βαθμολογία για το σύνολο των μαθημάτων. 2. Εμφανίστε τη μέση βαθμολογία σε κάθε μάθημα ξεχωριστά. 3. Εμφανίστε τα μαθήματα (κωδικούς), τη συμμετοχή και τη μέση βαθμολογία σε όλα τα μαθήματα που είχαν συμμετοχή μεγαλύτερη από το μάθημα «Baseis Dedomenwn».
Ζητούμενα:	Συναρτήσεις Συνάθροισης COUNT, AVG, MAX, MIN, SUM, GROUP BY.

Ζητούμενο 1: Εμφανίστε τη μέση βαθμολογία για το σύνολο των μαθημάτων.

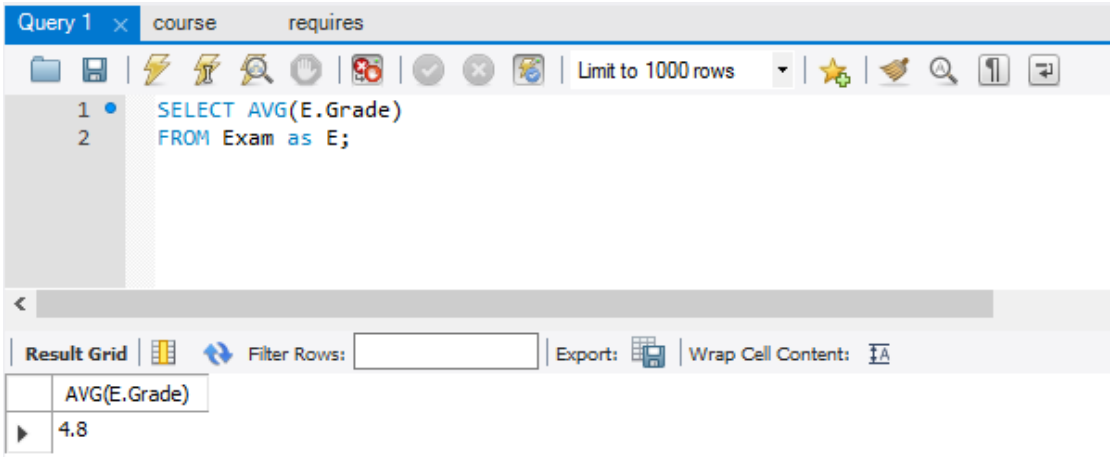
Η Εικόνα 9.1 δείχνει τα περιεχόμενα του πίνακα των βαθμολογιών.



Εικόνα 9.1: *Οι κωδικοί των μαθημάτων μαζί με τους βαθμούς των φοιτητών.*

Για το ζητούμενο δίνουμε:
SELECT AVG(E.Grade)
FROM Exam as E;

Το αποτέλεσμα φαίνεται στην Εικόνα 9.2.



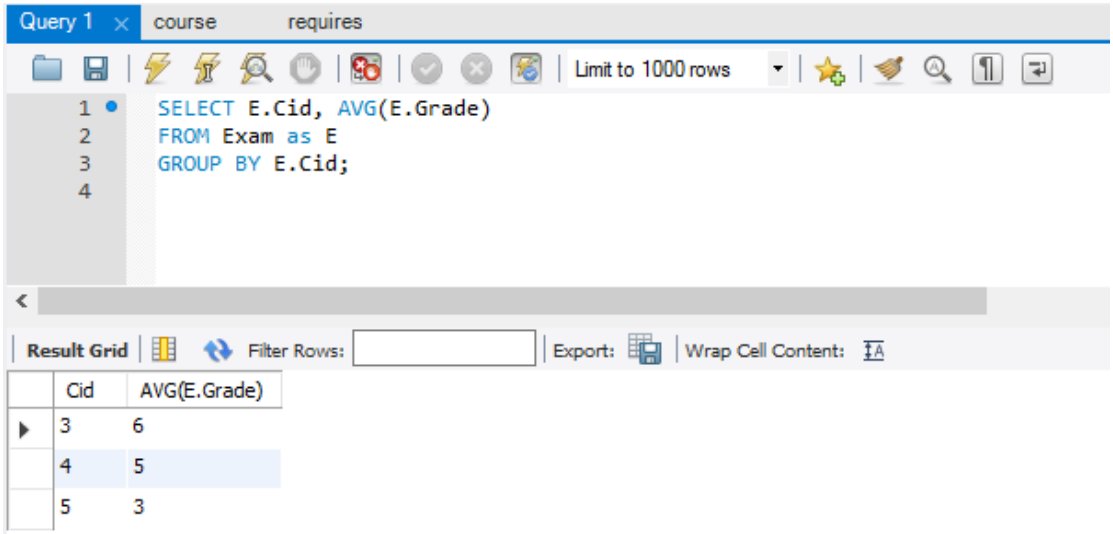
***Εικόνα 9.2**: Παράδειγμα συνάθροισης χωρίς GROUP BY.*

Ζητούμενο 2: Εμφανίστε τη μέση βαθμολογία σε κάθε μάθημα ξεχωριστά.

Για να βρούμε τη μέση βαθμολογία ανά μάθημα θα ομαδοποιήσουμε τις εγγραφές ως προς cid. Δίνουμε:

```
SELECT E.Cid, AVG(E.Grade)
FROM Exam as E
GROUP BY E.Cid;
```

Το αποτέλεσμα φαίνεται στην Εικόνα 9.3.



***Εικόνα 9.3:** Παράδειγμα συνάθροισης με GROUP BY.*

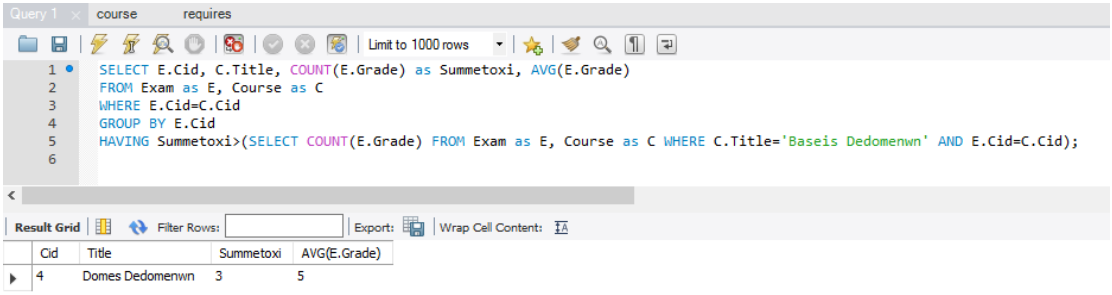
Ζητούμενο 3: Εμφανίστε τα μαθήματα (κωδικούς), τη συμμετοχή και τη μέση βαθμολογία σε όλα τα μαθήματα που είχαν συμμετοχή μεγαλύτερη από το μάθημα «Baseis Dedomenwn».

Δίνουμε:

```
SELECT E.Cid, C.Title, COUNT(E.Grade) as Summetoxi, AVG(E.Grade)
FROM Exam as E, Course as C
WHERE E.Cid=C.Cid
GROUP BY E.Cid
HAVING Summetoxi>
        (SELECT COUNT(E.Grade)
```

```
FROM Exam as E, Course as C
WHERE C.Title='Baseis Dedomenwn' AND E.Cid=C.Cid);
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 9.4.



Εικόνα 9.4: *Το μάθημα που έχει συμμετοχή μεγαλύτερη από τις Βάσεις Δεδομένων είναι οι Δομές Δεδομένων.*

9.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

1. Εμφανίστε τη μέση προϋπηρεσία για το σύνολο των εργαζομένων.
2. Εμφανίστε τη μέση προϋπηρεσία εργαζομένων για κάθε εταιρία.
3. Εμφανίστε το διευθυντή με τα περισσότερα έτη εμπειρίας σε κάθε εταιρία.
4. Εμφανίστε τους εργαζόμενους που έχουν προϋπηρεσία μεγαλύτερη από το μέσο όρο.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2), απαντήστε στα ακόλουθα ερωτήματα:

1. Εμφανίστε τις ταινίες μαζί με τη μέση βαθμολογία της κάθε ταινίας.
2. Βρείτε την ταινία με τη μέγιστη βαθμολογία και την ταινία με την ελάχιστη.
3. Εμφανίστε τις ταινίες μαζί με τον αριθμό των θεατών που παρακολούθησαν την κάθε ταινία.
4. Εμφανίστε τους σκηνοθέτες των ταινιών που τις παρακολούθησαν περισσότεροι θεατές από αυτούς που παρακολούθησαν την ταινία ‘Spectre’.

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3), απαντήστε στα ακόλουθα ερωτήματα:

1. Εμφανίστε τους παίκτες με τις περισσότερες και λιγότερες νίκες για το 2015.
2. Βρείτε τη μέση ηλικία των παικτών του τουρνουά ‘Australia Open’.
3. Εμφανίστε τα τουρνουά που είχαν συμμετοχή μεγαλύτερη από 3 αθλητές τα έτη 2013-2015.
4. Εμφανίστε τα τουρνουά που έχουν τα περισσότερα χρόνια ίδρυσης.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαια 8, 9
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
ORDER BY. http://www.w3schools.com/sql/sql_orderby.asp
GROYP BY.
http://www.w3schools.com/sql/sql_groupby.asp
http://dev.mysql.com/doc/refman/5.7/en/group-by-functions-and-modifiers.html
HAVING. http://www.w3schools.com/sql/sql_having.asp
LIMIT. http://www.mysqltutorial.org/mysql-limit.aspx
SUM. http://www.w3schools.com/sql/sql_func_sum.asp
MIN. http://www.w3schools.com/sql/sql_func_min.asp
MAX. http://www.w3schools.com/sql/sql_func_max.asp
AVG. http://www.w3schools.com/sql/sql_func_avg.asp
COUNT. http://www.w3schools.com/sql/sql_func_count.asp
EXISTS. http://dev.mysql.com/doc/refman/5.7/en/exists-and-not-exists-subqueries.html

Κεφάλαιο 10 Άλλες Πράξεις Θεωρίας Συνόλων

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιαστεί η πράξη της διαίρεσης. Στο κομμάτι των ασκήσεων θα γίνει συνολική επισκόπηση ερωτημάτων που εμπλέκουν πράξεις συνόλων.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- *Η ύλη των Κεφ. 6, 7, 8 και 9.*

10.1 Εισαγωγικές Έννοιες

10.1.1 Ορισμοί

Όλες οι συναρτήσεις Η πράξη της διαίρεσης στη σχεσιακή άλγεβρα μεταξύ δύο σχέσεων R και S συμβολίζεται με R÷S. Για να ορίζεται θα πρέπει τα πεδία του S να είναι υποσύνολο των πεδίων του R. Η πράξη επιστρέφει μία σχέση με πεδία τα πεδία του R που δεν ανήκουν στο S και πλειάδες τις υποπλειάδες του R (ως προς τα παραπάνω πεδία) που στο R συνδυάζονται με όλες τις πλειάδες του S. Για παράδειγμα θεωρείστε τις εξής σχέσεις R και S:

R

A	***B***	***C***	*D*
1	1	1	1
2	1	1	2
3	2	2	3
1	2	2	1
2	2	2	3
1	1	2	1
2	2	2	1
1	3	3	1
2	2	3	3
3	1	1	1

S

B	***C***
1	1
2	2

Το αποτέλεσμα της διαίρεσης του R με το S θα έχει ως πεδία τα A και D. Ως εγγραφές θα υπάρχουν οι υποεγγραφές ως προς (A,D) του R που συνδυάζονται και με τις δύο εγγραφές του S. Τέτοια στο παράδειγμα είναι μόνο η <1,1> που συνδυάζεται και με τις δύο εγγραφές του S. Συνδυάζεται και με μία ακόμα αλλά δεν παίζει ρόλο για το αποτέλεσμα επειδή το <3,3> δεν ανήκει στο S. Τελικό αποτέλεσμα:

R÷S

A	***D***
1	1

Υπό μία άλλη έννοια η πράξη της διαίρεσης μπορεί να ειδωθεί σαν την αντίστροφη πράξη του καρτεσιανού γινομένου (τυπικά δεν είναι), εφόσον με την προϋπόθεση ότι το Β δεν είναι κενό ισχύει: (A×B)÷B=A.

Η πράξη της διαίρεσης χρησιμοποιείται για να απαντηθούν ερωτήματα στα οποία ζητάμε να επιστραφεί το μέρος ενός πίνακα που συνδυάζεται με όλες τις εγγραφές ενός άλλου πίνακα.

Παραδείγματα:
1) Βρες τους φοιτητές που έχουν παρακολουθήσει τουλάχιστον ένα μάθημα κάθε καθηγητή.
Εκτέλεση: Βρίσκω όλους τους καθηγητές έστω Κ (ΚωδικόςΚαθηγητή). Βρίσκω έναν πίνακα Π με πεδία (ΚωδικόςΦοιτητή, ΚωδικόςΚαθηγητή) που έχει όλες τις εγγραφές που αντιστοιχούν στην πληροφορία ο φοιτητής με κωδικό Χ έχει παρακολουθήσει μάθημα του καθηγητή Υ. Το αποτέλεσμα το παίρνω διαιρώντας το Π με το Κ.

2) Βρες τις ομάδες που έχασαν από όλες τις ομάδες που έχασε η ομάδα Χ.
Εκτέλεση: Βρίσκω όλες τις ομάδες από τις οποίες έχασε η ομάδα Χ (έστω πίνακας Τ1 (ΚωδικόςΝικήτριαςΟμάδος)). Βρίσκω έναν πίνακα (έστω Τ2) της μορφής (ΚωδικόςΧαμένηςΟμάδος, ΚωδικόςΝικήτριαςΟμάδος) που έχει όλες τις εγγραφές της μορφής η ομάδα με κωδικό ΚωδικόςΧαμένηςΟμάδος έχασε από την ομάδα με κωδικό ΚωδικόςΝικήτριαςΟμάδος. Διαιρώ το Τ2 με το Τ1 για να βρω το αποτέλεσμα.

10.1.2 Υλοποίηση διαίρεσης στην SQL

Πιο τυπικά η πράξη της διαίρεσης στη σχεσιακή άλγεβρα μπορεί να οριστεί ως εξής:
$R \div S = \Pi_{R-S}(R) - \Pi_{R-S}((\Pi_{R-S}(R) \times S) - R)$, όπου $\Pi_{R-S}(R)$ είναι η πράξη της προβολής σύμφωνα με την οποία επιστρέφεται ο πίνακας R με πεδία όμως μόνο τα πεδία του R που δεν ανήκουν και στα πεδία του S. Σε SQL θυμίζουμε η προβολή είναι αντίστοιχη του τμήματος SELECT ενός ερωτήματος. Ο παραπάνω ορισμός λέει με απλά λόγια ότι το αποτέλεσμα της διαίρεσης προκύπτει αν από όλες τις υποεγγραφές (πεδία: R-S) του R (υποψήφιες εγγραφές για το τελικό αποτέλεσμα) αφαιρέσουμε αυτές που δε συνδυάζονται με όλες τις εγγραφές του S (πρώτη αφαίρεση). Για να βρούμε τις τελευταίες αρκεί να πάρουμε όλους τους συνδυασμούς (καρτεσιανό) των υποψήφιων για το αποτέλεσμα εγγραφών και του S και να αφαιρέσουμε το R. Οι υποεγγραφές που θα μείνουν είναι αυτές για τις οποίες στο αρχικό R δε συνδυάζονται με όλο το S.

Στην SQL και κατ'επέκταση στη MySQL δεν υποστηρίζεται απευθείας η πράξη της διαίρεσης, πχ. με κάποια εντολή DIVIDE. Ως εκ τούτου θα πρέπει να υλοποιηθεί χρησιμοποιώντας τις υπόλοιπες εντολές. Θα δείξουμε 3 διαφορετικές υλοποιήσεις χρησιμοποιώντας ως παράδειγμα το εξής ερώτημα: Βρέστε ποιοι λογαριασμοί έχουν στείλει χρήματα σε όλους τους λογαριασμούς του πελάτη με cid=1.

Για να υλοποιήσουμε το ερώτημα θα πρέπει πρώτα από όλα να γνωρίζουμε ποια είναι τα accid του πελάτη με cid=1. Αυτό βρίσκεται εύκολα από τον πίνακα Owns. Στη συνέχεια από τον πίνακα Transfer θα κρατήσουμε μόνο τα πεδία accidSource και accidDest. Το αποτέλεσμα προκύπτει διαιρώντας το δεύτερο πίνακα με τον πρώτο αφού στον πρώτο μετονομαστεί το accid σε accidDest.

Υλοποίηση 1

Μπορούμε να υλοποιήσουμε διαίρεση, υλοποιώντας απευθείας τον ορισμό της πράξης όπως δόθηκε παραπάνω. Η ακόλουθη SQL query υλοποιεί μία ελαφρώς παραλλαγμένη μορφή του παραπάνω:

```
SELECT DISTINCT accidSource
FROM Transfer
WHERE accidSource NOT IN (
        SELECT accidSource
        FROM
                (SELECT accidSource, T1.accid AS accidDest
                FROM Transfer, (SELECT accid FROM Owns WHERE cid=1) T1) T2
        WHERE (accidSource, accidDest) NOT IN (
                SELECT accidSource, accidDest FROM Transfer
                )
        );
```

ή σε πιο απλοποιημένη εκδοχή:

```
SELECT DISTINCT accidSource
FROM Transfer
WHERE accidSource NOT IN (
        SELECT accidSource
        FROM Transfer, (SELECT accid FROM Owns WHERE cid=1) T1
        WHERE (accidSource, accid) NOT IN (
                SELECT accidSource, accidDest FROM Transfer
                )
        );
```

Υλοποίηση 2

Ένας άλλος τρόπος να υλοποιηθεί διαίρεση είναι ο εξής: για κάθε μία πιθανή υποεγγραφή του αποτελέσματος βλέπουμε αν το σύνολο με το οποίο συνδυάζεται περιέχει όλες τις εγγραφές του διαιρέτη. Για να το πετύχουμε ελέγχουμε αν αφαιρώντας από το διαιρέτη το συνδυαζόμενο σύνολο προκύπτει κενό σύνολο. Στο παράδειγμα, για κάθε accidSource θα βρω το σύνολο των accidDest με τα οποία συνδυάζεται. Αφαιρώντας αυτό το σύνολο από τα accid του πελάτη με cid=1, αν αυτό που προκύπτει είναι κενό σημαίνει ότι το συγκεκριμένο accidSource που κοιτάζω συνδυάζεται με όλα τα ζητούμενα. Ακολουθεί η SQL query:

```
SELECT DISTINCT T1.accidSource
FROM Transfer T1
WHERE NOT EXISTS (
        SELECT *
        FROM Owns
        WHERE cid=1 AND accid NOT IN (
                SELECT T2.accidDest
                FROM Transfer T2
                WHERE T1.accidSource=T2.accidSource
                )
        );
```

Υλοποίηση 3

Στην υλοποίηση 2 στην ουσία υπολογίζεται για κάθε accidSource αν το σύνολο των accidDest με τα οποία συνδυάζεται περιέχει το σύνολο των ζητούμενων accid (του cid=1). Για να επιτευχθεί αυτό χρησιμοποιήθηκε αφαίρεση. Ένας άλλος τρόπος είναι με μέτρηση των εγγραφών. Δηλαδή για κάθε accidSource θα βρούμε με πόσα από τα ζητούμενα accid συνδυάζεται. Εάν αυτός ο αριθμός είναι ίσος με το συνολικό πλήθος των ζητουμένων το accidSource θα επιστρέφεται. Ακολουθεί η SQL query που υλοποιεί το παραπάνω.

```
SELECT DISTINCT accidSource
FROM Transfer INNER JOIN Owns ON accidDest=accid
WHERE cid=1
GROUP BY accidSource
HAVING COUNT(DISTINCT accidDest) = (
        SELECT COUNT(*)
        FROM Owns
        WHERE cid=1
        );
```

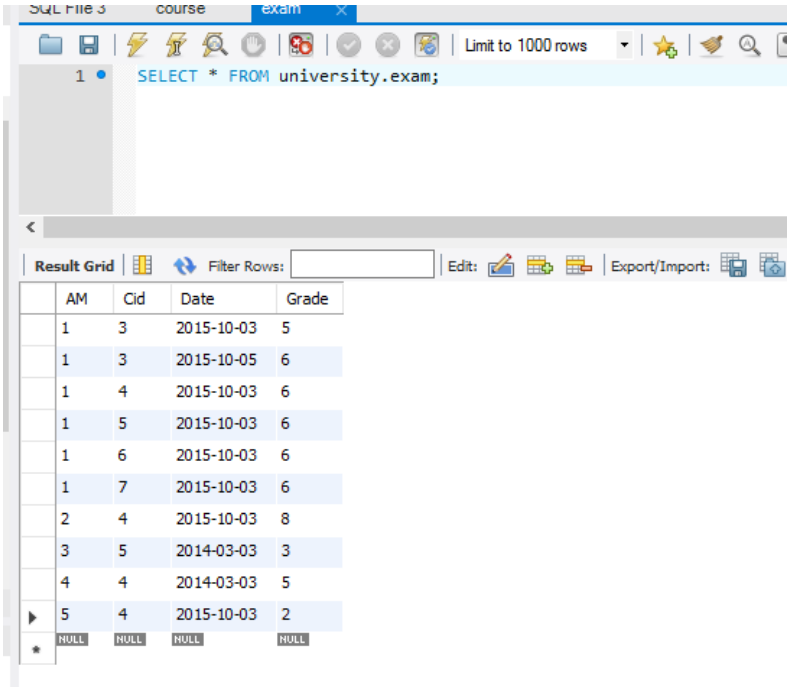
Σημείωση: Στο HAVING θα ήταν λάθος να μην είχαμε DISTINCT καθώς ένα accidSource μπορεί να έχει στείλει πολλές φορές χρήματα στο ίδιο accidDest. Επίσης ο λόγος για το COUNT(*) στο εμφωλευμένο αντί του πιο σωστού COUNT (DISTINCT accid) είναι γιατί στη συγκεκριμένη query είναι ισοδύναμα καθώς για cid=1 εγγυημένα τα accid θα είναι διαφορετικά μεταξύ τους. Στη γενικότερη περίπτωση όμως οφείλουμε να σημειώσουμε ότι χρειάζεται COUNT με DISTINCT παράμετρο και στο εμφωλευμένο υποερώτημα που μετράει τις εγγραφές του διαιρέτη.

10.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στη Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 4. Βρείτε τους φοιτητές που πέρασαν όλα τα μαθήματα. 5. Βρείτε τους φοιτητές που έχουν περάσει το μάθημα «Βάσεις Δεδομένων» το 2014 και «Δομές Δεδομένων» το 2015. 6. Βρείτε τους φοιτητές που πέρασαν κάποιο από τα μαθήματα «Βάσεις Δεδομένων» και «Δομές Δεδομένων» το 2015.
Ζητούμενα:	Διαίρεση, τομή και ένωση.

Ζητούμενο 1: Βρείτε τους φοιτητές που πέρασαν όλα τα μαθήματα.

Έστω ότι έχουμε τον ακόλουθο πίνακα Exam όπου ο φοιτητής με AM=1 έχει περάσει όλα τα μαθήματα.



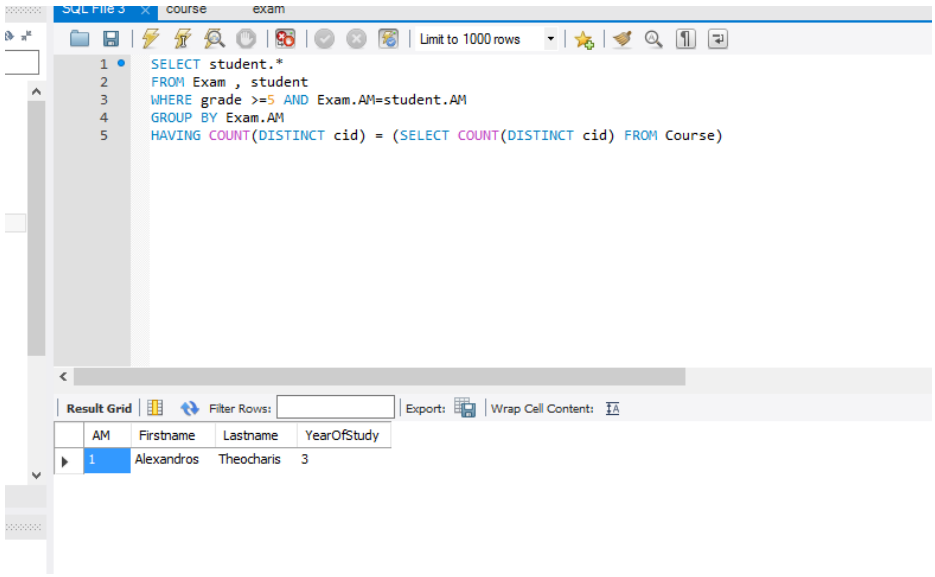
AM	Cid	Date	Grade
1	3	2015-10-03	5
1	3	2015-10-05	6
1	4	2015-10-03	6
1	5	2015-10-03	6
1	6	2015-10-03	6
1	7	2015-10-03	6
2	4	2015-10-03	8
3	5	2014-03-03	3
4	4	2014-03-03	5
5	4	2015-10-03	2
NULL	NULL	NULL	NULL

Εικόνα 10.1: *Πίνακας βαθμολογιών.*

Το ερώτημα που μας δίνει την απάντηση είναι το ακόλουθο:

```
SELECT Student.*
FROM Exam , Student
WHERE grade >=5 AND Exam.AM=Student.AM
GROUP BY Exam.AM
HAVING COUNT(DISTINCT cid) = (SELECT COUNT(DISTINCT cid) FROM Course)
```

Πάνω στον πίνακα Exam ομαδοποιούμε με βάση το AM για βαθμούς μεγαλύτερους ίσους του 5 (WHERE grade >=5 ….GROUP BY Exam.AM) και επιλέγουμε την ομάδα με πληθάριθμο ίσο με το πλήθος μαθημάτων (… HAVING COUNT(DISTINCT cid) = (SELECT COUNT(DISTINCT cid) FROM Course) …). Συνεπώς βρίσκουμε τους φοιτητές που έχουν περάσει όλα τα μαθήματα. Τέλος κάνουμε σύνδεση με τον πίνακα student για να συνδέσουμε τα AM με τα υπόλοιπα στοιχεία του φοιτητή. Τα αποτελέσματα φαίνονται στην Εικόνα 10.2.



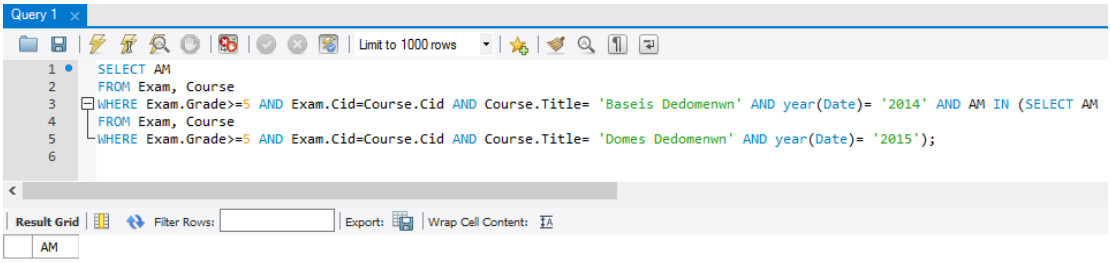
Εικόνα 10.2: *Υλοποίηση διαίρεσης χρησιμοποιώντας συνάρτηση συνάθροισης.*

Ζητούμενο 2: Βρείτε τους φοιτητές που έχουν περάσει το μάθημα «Βάσεις Δεδομένων» το 2014 και «Δομές Δεδομένων» το 2015.

Για να βρούμε το ζητούμενο θα βρούμε ξεχωριστά τους φοιτητές που πέρασαν τα δύο μαθήματα και θα πάρουμε την τομή τους.
Δίνουμε:

```
SELECT AM
FROM Exam, Course
WHERE Exam.Grade>=5 AND
Exam.Cid=Course.Cid AND
Course.Title= ‘Baseis Dedomenwn’ AND
year(Date)= ‘2014’ AND
AM IN (SELECT AM
        FROM Exam, Course
        WHERE Exam.Grade>=5 AND
        Exam.Cid=Course.Cid AND
        Course.Title= ‘Domes Dedomenwn’
        AND year(Date)= ‘2015’
        );
```

Στο παραπάνω ερώτημα υλοποιούμε την πράξη INTERSECT. Όπως μπορούμε να δούμε στην παρακάτω Εικόνα, το ερώτημα δεν επιστρέφει κάποια τιμή καθώς δεν έχουμε κάποιον φοιτητή που να έχει περάσει και τα δύο μαθήματα το 2014 και 2015.



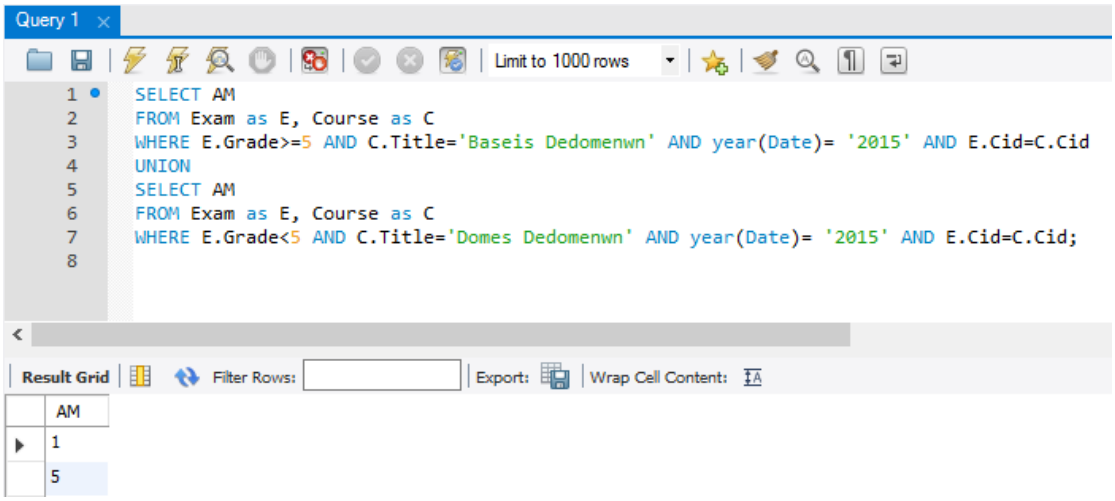
Εικόνα 10.3: *Παράδειγμα υλοποίησης της πράξης INTERSECT.*

Ζητούμενο 3: Βρείτε τους φοιτητές που πέρασαν κάποιο από τα μαθήματα «Βάσεις Δεδομένων» και «Δομές Δεδομένων» το 2015.

Ένας τρόπος για να απαντηθεί το ερώτημα είναι βρίσκοντας για το κάθε μάθημα τους επιτυχόντες ξεχωριστά και παίρνοντας την ένωση.
Δίνουμε:

SELECT AM
FROM Exam as E, Course as C
WHERE E.Grade>=5 AND C.Title='Baseis Dedomenwn' AND year(Date)= '2015' AND E.Cid=C.Cid
UNION
SELECT AM
FROM Exam as E, Course as C
WHERE E.Grade<5 AND C.Title='Domes Dedomenwn' AND year(Date)= '2015' AND E.Cid=C.Cid;

Τα αποτελέσματα φαίνονται στην Εικόνα 10.4.



Εικόνα 10.4: *Παράδειγμα ένωσης.*

Για το παραπάνω ζητούμενο δε χρειάζεται απαραίτητα να γράψουμε ερώτημα ένωσης (δόθηκε χάριν παραδείγματος). Ισοδύναμα μπορεί να δοθεί το ακόλουθο:

SELECT AM
FROM Exam as E, Course as C
WHERE E.Grade>=5 AND
(C.Title='Baseis Dedomenwn' OR C.Title='Domes Dedomenwn') AND
year(Date)= '2015' AND
E.Cid=C.Cid;

10.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

5. Φτιάξτε μια κατάσταση με όλους τους εργαζόμενους που να συνοδεύεται από την εταιρία στην οποία δουλεύουν.
6. Βρείτε τους διευθυντές που έχουν διευθύνει την εταιρία 'Info' αλλά όχι την εταιρία 'Advance'.
7. Βρείτε τους εργαζόμενους που το 2000 απασχολούνταν στην εταιρία 'Info' και το 2010 απασχολούνταν στην εταιρία 'Advance'.
8. Εμφανίστε το ονοματεπώνυμο των εργαζομένων και την εταιρία που εργάζονται.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2), απαντήστε στα ακόλουθα ερωτήματα:

5. Φτιάξτε μια κατάσταση με όλες τις ταινίες μαζί με τους σκηνοθέτες τους.
6. Βρείτε τις ταινίες που τις σκηνοθέτησε ο 'Steven Spielberg' αλλά όχι ο 'Cuiseppe Tornatore'.
7. Βρείτε τις ταινίες που τις σκηνοθέτησε ο 'Steven Spielberg' και ο 'Cuiseppe Tornatore'.
8. Εμφανίστε τους θεατές με τις ταινίες που παρακολούθησαν.

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3), απαντήστε στα ακόλουθα ερωτήματα:

5. Φτιάξτε μια κατάσταση με όλα τα τουρνουά μαζί με τους νικητές τους.
6. Βρείτε τα τουρνουά που έπαιξε ο 'Roger Federer' και ο 'Rafael Nadal'.
7. Βρείτε τα τουρνουά που έπαιξε ο 'Roger Federer', αλλά όχι ο 'Rafael Nadal'.
8. Εμφανίστε τους νικητές, έτος νίκης για κάθε τουρνουά το 2013-2015.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαια 8, 9
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.

Κεφάλαιο 11 Ενημέρωση Δεδομένων και Συνολική Επισκόπηση Ερωτημάτων

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν οι εντολές UPDATE και DELETE για ενημέρωση και διαγραφή εγγραφών σε έναν ή περισσότερους πίνακες.

Προαπαιτούμενη γνώση
Προαπαιτούμενα για την κατανόηση των εννοιών αλλά και την εργαστηριακή άσκηση είναι τα ακόλουθα:

- *Η ύλη των Κεφ. 3, 6-9.*

.

11.1 Εισαγωγικές Έννοιες

Στη συνέχεια θα παρουσιαστεί η σύνταξη και χρήση των εντολών UPDATE και DELETE. Στα ερωτήματα-παραδείγματα που θα δοθούν θα χρησιμοποιηθούν οι πίνακες που χρησιμοποιήθηκαν και σε προηγούμενα κεφάλαια, οι οποίοι έχουν ως ακολούθως:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
1	2231011111
1	6944444444
2	2103333333

Account

accid	balance	dateOfCreation
100	10000	2014-01-01
200	40000	2013-02-01
300	30000	2005-03-10
400	20000	2005-03-10

Owns

cid	accid
1	100
2	200
1	300
1	400

Action

accid	actid	amount	type	dateOfAction
100	1	500	2	2014-01-14
200	1	500	2	2014-01-14
200	2	1000	2	2015-11-11
200	3	2000	1	2015-12-11
300	1	1000	2	2015-11-11
300	2	500	1	2015-12-11

Transfer

accidSource	actidSource	accidDest	actidDest
200	1	100	1
300	1	200	2

11.1.1 Η εντολή UPDATE

Η ενημέρωση εγγραφών, δηλ. η αλλαγή τιμών σε ένα ή περισσότερα πεδία τους, γίνεται με τη χρήση της εντολής UPDATE. Η βασική σύνταξη της εντολής έχει ως εξής:

UPDATE *πίνακες*
SET *λίστα_αναθέσεων*
WHERE *λογική_έκφραση*
Το όρισμα στο UPDATE (*πίνακες*) είναι ισοδύναμο με το όρισμα που παίρνει το FROM σε ένα SELECT-FROM-WHERE ερώτημα. Με άλλα λόγια μπορεί να περιλαμβάνει συνενώσεις πολλαπλών πινάκων, πίνακες που προκύπτουν από εμφωλευμένα ερωτήματα κλπ. Κατά την εκτέλεση, όλες οι εγγραφές που ανήκουν σε κάποιον από τους πίνακες που αναγράφονται στο όρισμα *πίνακες* του UPDATE και μπορεί να ενημερωθούν (δηλ. υπάρχουν τα αντίστοιχα δικαιώματα πχ., πρόκειται για βασικούς πίνακες στη ΒΔ και όχι για πίνακες που προκύπτουν από εμφωλευμένα ερωτήματα) ενημερώνονται εφόσον πληρούν τη λογική συνθήκη του WHERE. Οι αλλαγές στις οποίες υπόκεινται περιγράφονται στο SET. Ακολουθούν παραδείγματα. Η πλήρης σύνταξη της εντολής μπορεί να βρεθεί στο Κεφ. 14.2.11 του εγχειριδίου της MySQL 5.7.

Ερώτημα 1: Τοκίστε με 1% όλους τους λογαριασμούς.

Θα πρέπει στον πίνακα Account να αλλάξουμε το περιεχόμενο του balance.

Query 1:
```
UPDATE Account
SET balance=balance+0.01*balance;
```

Προσέξτε ότι παραλείποντας το WHERE υπονοείται WHERE TRUE και επομένως το SET εφαρμόζεται σε όλες τις εγγραφές του Account. Επίσης παρατηρήστε ότι ο τελεστής '=' στο SET παίζει το ρόλο της ανάθεσης και όχι της σύγκρισης. Το αποτέλεσμα θα είναι:

Account

accid	balance	dateOfCreation
100	10100	2014-01-01
200	40400	2013-02-01
300	30300	2005-03-10
400	20200	2005-03-10

Ερώτημα 2: Εφαρμόστε έκτακτη εισφορά 500 ευρώ για τους λογαριασμούς με υπόλοιπο άνω των 30000.

Query 2:
```
UPDATE Account
SET balance=balance-500
WHERE balance >30000;
```

Ερώτημα 3: Αλλάξτε το ον/μο της: Maria Papantoniou σε Mary Papadoniou.

Query 3:
```
UPDATE Customer
SET name='Mary', sname='Papadoniou'
WHERE name='Maria' AND sname='Papantoniou';
```

Ερώτημα 4: Αλλάξτε το όνομα του: Kostas Kostantinou σε Costas Kostantinou και προσθέστε 1000 ευρώ σε κάθε λογαριασμό του.

1ος Τρόπος
Μπορούμε να δούμε το παραπάνω ερώτημα σαν δύο ξεχωριστά ερωτήματα ενημέρωσης ως εξής:

```
UPDATE Customer
SET name='Costas'
WHERE name='Kostas' AND sname='Kostantinou';
```

και στη συνέχεια χρησιμοποιώντας εμφωλευμένο ερώτημα:

```
UPDATE Account
SET balance=balance+1000
WHERE accid IN (SELECT accid
                FROM Owns INNER JOIN Customer ON Owns.cid=Customer.cid
                WHERE name='Costas' AND sname='Kostantinou');
```

2ος Τρόπος
Μπορούμε να χρησιμοποιήσουμε ένα ερώτημα ενημέρωσης με πολλούς πίνακες όπως παρακάτω:

```
UPDATE Customer, Owns, Account
SET name='Costas', balance=balance+1000
WHERE name='Kostas' AND sname='Kostantinou' AND Customer.cid=Owns.cid AND
Owns.accid=Account.accid;
```

ή ισοδύναμα με INNER JOIN:

```
UPDATE Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
SET name='Costas', balance=balance+1000
WHERE name='Kostas' AND sname='Kostantinou';
```

Το παραπάνω ερώτημα θα αλλάξει 4 εγγραφές, 3 στον πίνακα Account και 1 στον πίνακα Customer. Υποθέτοντας τους πίνακες του παραδείγματος με τις αρχικές εγγραφές, το τελικό αποτέλεσμα θα είναι (οι εγγραφές που αλλάζουν φαίνονται με italics):

Customer

cid	**afm**	**address**	**name**	**sname**	**dateOfBirth**
1	*077783234*	*56 Baltetsiou st.*	*Costas*	*Kostantinou*	*1990-10-30*
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Account

accid	**balance**	**dateOfCreation**
100	*11000*	*2014-01-01*
200	40000	2013-02-01
300	*31000*	*2005-03-10*
400	*21000*	*2005-03-10*

Ο τρόπος με τον οποίο εκτελείται ένα ερώτημα ενημέρωσης πολλαπλών πινάκων είναι ο ακόλουθος. Πρώτα υπολογίζεται ο τελικός πίνακας στο UPDATE. Στο παράδειγμα το καρτεσιανό γινόμενο (ή inner join) των Customer, Owns και Account. Για κάθε μία εγγραφή του τελικού πίνακα του UPDATE που πληροί τη συνθήκη του WHERE γίνονται οι αλλαγές που περιγράφονται στο SET στις εγγραφές των αντίστοιχων πινάκων από τις οποίες προέκυψε η εγγραφή που εξετάζεται.
Προσοχή: Σε ερώτημα ενημέρωσης πολλαπλών πινάκων κάθε εγγραφή ενημερώνεται μία και μόνο φορά και όχι κάθε φορά που βρίσκεται σε συνδυασμό που πληροί το WHERE. Για παράδειγμα, έστω το ακόλουθο ερώτημα ενημέρωσης:

```
UPDATE Account a1, Account a2
SET a1.balance=a1.balance + 1;
```

Υποθέτοντας το στιγμιότυπο του Account που προέκυψε από το προηγούμενο ερώτημα το αποτέλεσμα είναι:

Account

accid	**balance**	**dateOfCreation**
100	11001	2014-01-01
200	40001	2013-02-01
300	31001	2005-03-10
400	21001	2005-03-10

Με άλλα λόγια σε όλες τις εγγραφές έγινε η πρόσθεση 1 ευρώ στο υπόλοιπο μία φορά, παρότι όπως γίνεται αντιληπτό κάθε εγγραφή του Account εμφανίζεται 4 φορές στο καρτεσιανό γινόμενο.

Τέλος όπως αναφέρθηκε και στην αρχή, αν και στο UPDATE μπορούμε να έχουμε πίνακες που προκύπτουν από εμφωλευμένα ερωτήματα, δεν μπορούμε να αλλάξουμε τις εγγραφές αυτών των πινάκων. Για παράδειγμα το ακόλουθο ερώτημα:

```
UPDATE (SELECT * FROM Customer) C
SET name='Helen'
WHERE name= 'Eleni';
```

θα επιστρέψει το ακόλουθο σφάλμα:
ERROR 1288 (HY000): The target table C of the UPDATE is not updatable

Αντίθετα το ακόλουθο ερώτημα:

```
UPDATE (SELECT * FROM Customer) C, Customer
SET Customer.name='Helen'
WHERE Customer.name= 'Eleni';
```

θα πραγματοποιήσει ορθά την αλλαγή ονόματος καθώς τα πεδία που αλλάζουν ανήκουν σε βασικό πίνακα της ΒΔ και όχι στον πίνακα C που προέκυψε από το εμφωλευμένο ερώτημα.

Πριν προχωρήσουμε θα πρέπει να τονιστεί ότι η ενημέρωση πεδίου που είναι ξένο κλειδί απαγορεύεται σε περίπτωση που η τιμή στην οποία αλλάζει δεν υπάρχει στον πίνακα στον οποίο αναφέρεται. Για παράδειγμα η ακόλουθη εντολή:

```
UPDATE Owns
SET cid=100
WHERE cid=1;
```

οδηγεί σε σφάλμα καθώς δεν υπάρχει στον πίνακα Customer πελάτης με cid=100, ενώ η ακόλουθη:

```
UPDATE Owns
SET cid=3
WHERE cid=1;
```

θα εκτελεστεί κανονικά καθώς υπάρχει πελάτης στο Customer με cid=3. Ο πίνακας Customer προφανώς δε θα υποστεί αλλαγές ενώ ο Owns θα γίνει ως εξής:

Owns

cid	**accid**
3	100
2	200
3	300
3	400

Κατ' αντιστοιχία αν επιχειρηθεί αλλαγή πεδίου σε έναν πίνακα που είναι πίνακας στον οποίο αναφέρεται ξένο κλειδί άλλου πίνακα θα ισχύσουν οι περιορισμοί αναφοράς που έχουν τεθεί (δες και Κεφ. 3). Για παράδειγμα για τους αρχικούς πίνακες του παραδείγματος:

Customer

cid	**afm**	**address**	**name**	**sname**	**dateOfBirth**
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Owns

cid	**accid**
1	100
2	200
1	300
1	400

υποθέτοντας ότι στον πίνακα Owns το ξένο κλειδί cid έχει οριστεί με την επιλογή ON UPDATE CASCADE η ακόλουθη εντολή:

UPDATE Customer
SET cid=10
WHERE cid=1;

θα πραγματοποιηθεί επιστρέφοντας μήνυμα της μορφής:

Query OK, 1 row affected
Rows matched: 1 Changed: 1 Warnings: 0

που αντιστοιχεί στην αλλαγή του Customer σε:

Customer

cid	**afm**	**address**	**name**	**sname**	**dateOfBirth**
10	*077783234*	*56 Baltetsiou st.*	*Kostas*	*Kostantinou*	*1990-10-30*
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Εμφανίζοντας τις εγγραφές του Owns όμως, παρατηρούμε ότι έγιναν και άλλες 3 αλλαγές εξαιτίας της επιλογής ON UPDATE CASCADE στη δήλωση του ξένου κλειδιού:

Owns

cid	**accid**
10	*100*
2	200
10	*300*
10	*400*

11.1.2 Η εντολή DELETE

Η διαγραφή εγγραφών γίνεται με τη χρήση της εντολής DELETE της οποίας η βασική σύνταξη για έναν πίνακα έχει ως εξής:

DELETE FROM *πίνακας*
WHERE *λογική_έκφραση*

Η εκτέλεση της εντολής γίνεται εκτιμώντας για κάθε εγγραφή του πίνακα στο DELETE FROM (*πίνακας*) τη *λογική_έκφραση* του WHERE. Αν είναι TRUE τότε η εγγραφή διαγράφεται. Για παράδειγμα η ακόλουθη εντολή:
DELETE FROM Account;

σβήνει όλες τις εγγραφές του πίνακα Account, καθώς σε περίπτωση που δεν υπάρχει WHERE υπονοείται WHERE TRUE, ενώ η εντολή:

DELETE FROM Account
WHERE balance > 20000;

θα διαγράψει μόνο τις εγγραφές με υπόλοιπο άνω των 20000.

Η DELETE μπορεί να χρησιμοποιηθεί για να διαγραφούν εγγραφές από περισσότερους του ενός πίνακες. Η βασική σύνταξη σε αυτήν την περίπτωση έχει ως εξής:

DELETE *λίστα*
FROM *πίνακες*
WHERE *λογική_έκφραση*

όπου *λίστα* είναι ένας ή περισσότεροι πίνακες χωρισμένοι με κόμμα και το όρισμα του FROM (*πίνακες*) είναι όμοιο με το όρισμα του UPDATE στην εντολή UPDATE-SET-WHERE, δηλ. περιλαμβάνει έκφραση που ενέχει συνενώσεις πινάκων. Η εκτέλεση της εντολής έχει ως εξής: πρώτα υπολογίζεται το FROM, στη συνέχεια για κάθε μία από τις εγγραφές του πίνακα που παράγεται ελέγχεται αν πληρείται η *λογική_έκφραση* του WHERE. Για τις εγγραφές που πληρείται διαγράφονται οι αντίστοιχες εγγραφές των πινάκων από τις οποίες προέκυψαν μόνο για τους πίνακες που αναγράφονται στο όρισμα του DELETE (*λίστα*) και εφ' όσον αυτό είναι επιτρεπτό πχ. δεν πρόκειται για πίνακες που προκύπτουν από υποερωτήματα. Ακολουθούν παραδείγματα. Περισσότερα για τη σύνταξη και χρήση της DELETE μπορεί να βρεθούν στο Κεφ. 14.2.2 του εγχειριδίου της MySQL 5.7.

Παράδειγμα 1: Να διαγραφούν όλοι οι πελάτες από τον πίνακα Customer που δεν έχουν κάποιο λογαριασμό.

```
DELETE Customer
FROM Customer LEFT JOIN Owns ON Customer.cid=Owns.cid
WHERE Owns.cid IS NULL;
```

Παράδειγμα 2: Να διαγραφούν όλες οι εγγραφές των πινάκων Customer και Owns που αντιστοιχούν σε πελάτες που έχουν τουλάχιστον ένα λογαριασμό με υπόλοιπο μεγαλύτερο ή ίσο των 40000.

Υποθέτοντας τους εξής πίνακες:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Account

accid	balance	dateOfCreation
100	10000	2014-01-01
200	40000	2013-02-01
300	30000	2005-03-10
400	20000	2005-03-10

Owns

cid	accid
1	100
2	200
1	300
1	400

Το παρακάτω ερώτημα:

```
DELETE Owns, Customer
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
WHERE balance>=40000;
```

πετυχαίνει τις επιθυμητές διαγραφές αφήνοντας τους πίνακες στην ακόλουθη μορφή:

Customer

cid	afm	address	name	sname	dateOfBirth
1	077783234	56 Baltetsiou st.	Kostas	Kostantinou	1990-10-30
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Account

accid	balance	dateOfCreation
100	10000	2014-01-01
200	40000	2013-02-01
300	30000	2005-03-10
400	20000	2005-03-10

Owns

cid	accid
1	100
1	300
1	400

Παρόλα αυτά το ερώτημα δεν είναι σωστό γενικά. Θεωρείστε την περίπτωση στην οποία ζητάμε να απαλειφθούν οι εγγραφές στον Customer και Owns που αντιστοιχούν σε πελάτες που έχουν κάποιο λογαριασμό με υπόλοιπο μικρότερο των 15000. Το παρακάτω ερώτημα:

```
DELETE Owns, Customer
FROM Customer
        INNER JOIN Owns ON Customer.cid=Owns.cid
        INNER JOIN Account ON Owns.accid=Account.accid
WHERE balance<15000;
```

δε θα εκτελεστεί επιστρέφοντας μήνυμα σχετικό με παραβίαση ξένου κλειδιού. Ο λόγος είναι ο εξής. Η μόνη εγγραφή που πληροί το WHERE από τον πίνακα του FROM, είναι η σχετική με το accid=100. Κατά συνέπεια θα επιχειρηθεί η διαγραφή της εγγραφής <1, 100> από τον πίνακα Owns και η εγγραφή με cid=1 από τον πίνακα Customer. Η δεύτερη διαγραφή θα παραβιάσει την ύπαρξη ξένου κλειδιού όσον αφορά στις εγγραφές <1, 300> και <1, 400>. Ακόμα όμως και αν δεν υπήρχε περιορισμός ξένου κλειδιού το ερώτημα δε θα ήταν σωστό καθώς οι πίνακες Customer και Owns θα ήταν ως εξής:

Customer

cid	afm	address	name	sname	dateOfBirth
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Owns

cid	accid
2	200
1	300
1	400

αντί του ορθού:

Customer

cid	afm	address	name	sname	dateOfBirth
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Owns

cid	accid
2	200

Για να πετύχουμε να διαγραφούν και οι 3 εγγραφές με cid=1 από το Owns πιο σωστό θα ήταν να δίναμε:

```
DELETE  Owns, Customer
FROM Customer INNER JOIN Owns ON Customer.cid=Owns.cid
WHERE Owns.cid IN (SELECT OW.cid
                   FROM Owns OW INNER JOIN Account
                   WHERE balance<15000);
```

Το παραπάνω ερώτημα θα αποτύχει επιστρέφοντας το ακόλουθο μήνυμα λάθους:
ERROR 1093 (HY000): You can't specify target table 'Owns' for update in FROM clause

Στη MySQL 5.7 δεν επιτρέπεται η χρήση σε εμφωλευμένο ερώτημα του WHERE πίνακα που μετέχει στη λίστα του DELETE και ως εκ τούτου θα υποστεί αλλαγές. Μπορούμε όμως να γράψουμε την παρακάτω εντολή που χρησιμοποιεί εμφωλευμένο στο FROM και είναι και δίνει και την τελική απάντηση στο ερώτημα:

Εντολή 1

```
DELETE Owns, Customer
FROM (SELECT cid
      FROM Owns INNER JOIN Account ON Owns.accid=Account.accid
      WHERE balance<15000) T1, Owns, Customer
WHERE Customer.cid=Owns.cid AND Owns.cid=T1.cid;
```

Σε περίπτωση που το ξένο κλειδί cid έχει οριστεί με επιλογή ON DELETE RESTRICT η εντολή δε θα εκτελεστεί επιστρέφοντας σφάλμα ξένου κλειδιού. Ο λόγος είναι ότι η MySQL 5.7 δεν εξασφαλίζει ότι ο τρόπος εκτέλεσης μιας εντολής DELETE θα ακολουθεί τις εξαρτήσεις ξένου κλειδιού.

Αντίθετα αν δεν υπάρχει περιορισμός ξένου κλειδιού ή ο περιορισμός είναι με ON DELETE CASCADE η εντολή θα εκτελεστεί κανονικά διαγράφοντας τις επιθυμητές 3 εγγραφές από τον πίνακα Owns και τη 1 εγγραφή από το Customer που έχουν cid=1.

Να σημειωθεί ότι στην τελευταία περίπτωση δηλ. αυτή στην οποία το ξένο κλειδί έχει οριστεί με ON DELETE CASCADE θα μπορούσαμε να πετύχουμε το ίδιο αποτέλεσμα διαγράφοντας μόνο τις σχετικές εγγραφές από τον Customer και αφήνοντας την επιλογή ON DELETE CASCADE να κάνει τις αλλαγές στον πίνακα Owns:

Εντολή 2

```
DELETE FROM Customer
WHERE cid IN (SELECT cid
              FROM Owns INNER JOIN Account ON Owns.accid=Account.accid
              WHERE balance<15000);
```

Να σημειωθεί ότι αν όλα τα ξένα κλειδιά σε όλους τους πίνακες της ΒΔ είναι ορισμένα με ON DELETE CASCADE τότε και με τις δύο παραπάνω εντολές θα γίνουν παραπλεύρως αλλαγές και στον πίνακα Phones όπου θα σβηστούν οι εγγραφές (δύο συνολικά) με cid=1 αφήνοντας μία μόνο εγγραφή στον πίνακα:

Phones

cid	pnum
2	2103333333

Παράδειγμα 3: Να διαγραφούν όλες οι εγγραφές των πινάκων Customer, Account και Owns που αντιστοιχούν σε πελάτες που έχουν τουλάχιστον ένα λογαριασμό με υπόλοιπο μικρότερο των 15000.

Σύμφωνα και με το προηγούμενο παράδειγμα, αν δεν υπάρχουν ξένα κλειδιά δίνουμε:

```
DELETE Owns, Customer, Account
FROM (SELECT cid
        FROM Owns INNER JOIN Account ON Owns.accid=Account.accid
        WHERE balance<15000) T1, Owns, Customer, Account
WHERE Customer.cid=Owns.cid AND Owns.cid=T1.cid AND Account.accid=Owns.accid;
```

Αν υπάρχουν ξένα κλειδιά και είναι όλα ορισμένα με την επιλογή ON DELETE CASCADE σε όλους τους πίνακες, τότε θα γίνουν διαγραφές και στους πίνακες Phones, Action και Transfer. Στον πίνακα Phones θα σβηστούν οι εγγραφές με cid=1, στον Action οι εγγραφές με accid=100, 300 ή 400 (που αντιστοιχούν σε αυτές που διαγράφησαν από τον πίνακα Account), και στον πίνακα Transfer αυτές που έχουν είτε σαν accidSource ή σαν accidDest τις προαναφερθείσες τιμές accid. Η αρχική και τελική μορφή των πινάκων φαίνεται παρακάτω όπου έχουν διαγραμμιστεί οι εγγραφές που διαγράφονται:

Customer

cid	afm	address	name	sname	dateOfBirth
~~1~~	~~077783234~~	~~56 Baltetsiou st.~~	~~Kostas~~	~~Kostantinou~~	~~1990-10-30~~
2	175783239	107 Diakou st.	Eleni	Kostantinou	1985-11-02
3	095111139	12 Rodon st.	Maria	Papantoniou	1967-03-20

Phones

cid	pnum
~~1~~	~~2231011111~~
~~1~~	~~6944444444~~
2	2103333333

Account

accid	balance	dateOfCreation
~~100~~	~~10000~~	~~2014-01-01~~
200	40000	2013-02-01
~~300~~	~~30000~~	~~2005-03-10~~
~~400~~	~~20000~~	~~2005-03-10~~

Owns

cid	accid
~~1~~	~~100~~
2	200
~~1~~	~~300~~
~~1~~	~~400~~

Action

accid	actid	amount	type	dateOfAction
~~100~~	~~1~~	~~500~~	~~2~~	~~2014-01-14~~
200	1	500	2	2014-01-14
200	2	1000	2	2015-11-11
200	3	2000	1	2015-12-11
~~300~~	~~1~~	~~1000~~	~~2~~	~~2015-11-11~~
~~300~~	~~2~~	~~500~~	~~1~~	~~2015-12-11~~

Transfer

accidSource	actidSource	accidDest	actidDest
~~200~~	~~1~~	~~100~~	~~1~~
~~300~~	~~1~~	~~200~~	~~2~~

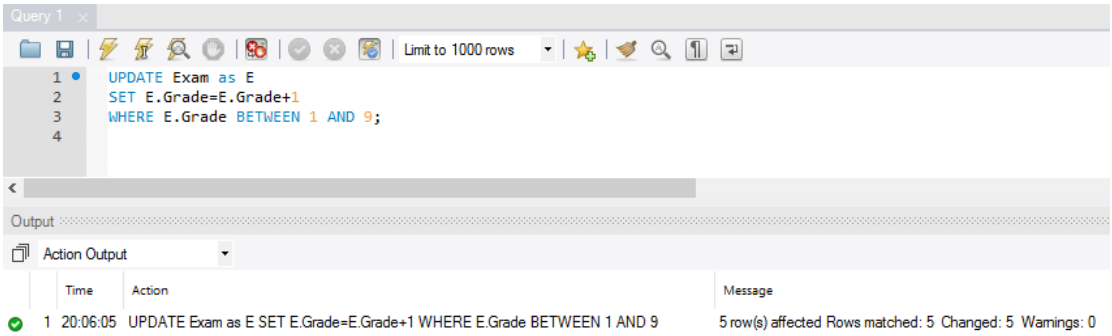
11.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στη Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 27. Αλλάξτε τους βαθμούς των εξετάσεων προσθέτοντας +1 βαθμό σε κάθε εγγραφή για όσες βαθμολογίες είναι μεταξύ 1 και 9. 28. Αλλάξτε την εγγραφή του φοιτητή 'Karpetas' σε AM=5 και YearOfStudy='4'. 29. Αλλάξτε το έτος σπουδών σε '5' για τους φοιτητές που πέρασαν το μάθημα 'Domes Dedomenwn'. 30. Σβήστε όλες τις εγγραφές του Requires. 31. Σβήστε όλα τα κινητά τηλέφωνα των φοιτητών. 32. Σβήστε από τον Exam τις εγγραφές που αντιστοιχούν στο μάθημα 'Baseis Dedomenwn'. Προτού υλοποιήσετε τα ακόλουθα ερωτήματα, εξάγετε τη βάση δεδομένων university.sql ώστε οι αλλαγές που θα γίνουν να μην την επηρεάσουν.
Ζητούμενα:	Τροποποίηση και διαγραφή εγγραφών (UPDATE, DELETE).

Ζητούμενο 1: Αλλάξτε τους βαθμούς των εξετάσεων προσθέτοντας +1 βαθμό σε κάθε εγγραφή για όσες βαθμολογίες είναι μεταξύ 1 και 9.
Δίνουμε:

```
UPDATE Exam as E
SET E.Grade=E.Grade+1
WHERE E.Grade BETWEEN 1 AND 9;
```

Παράδειγμα αποτελέσματος φαίνεται στην Εικόνα 11.1 όπου συμβαίνουν αλλαγές σε 5 εγγραφές.

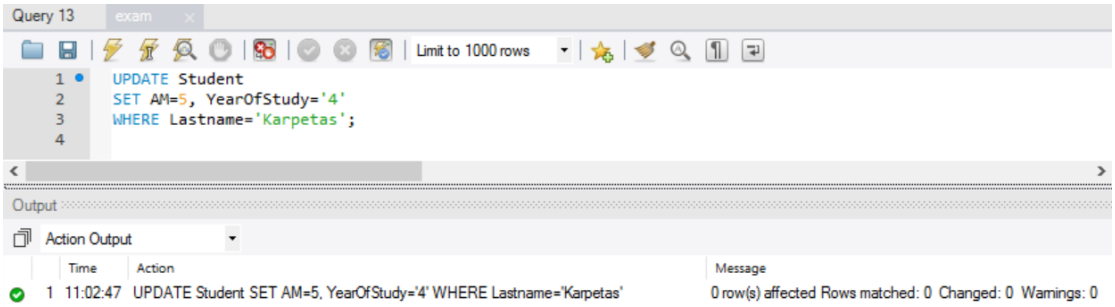


Εικόνα 11.1: *Παράδειγμα ενημέρωσης σε ένα πεδίο.*

Ζητούμενο 2: Αλλάξτε την εγγραφή του φοιτητή ‘Karpetas’ σε AM=5 και YearOfStudy=‘4’. Δίνουμε:

UPDATE Student
SET AM=5, YearOfStudy=’4’
WHERE Lastname=‘Karpetas’;

Στην Εικόνα 11.2 φαίνεται το αποτέλεσμα. Επειδή δεν υπήρχε φοιτητής με το συγκεκριμένο επώνυμο καμία εγγραφή δεν επηρεάζεται.

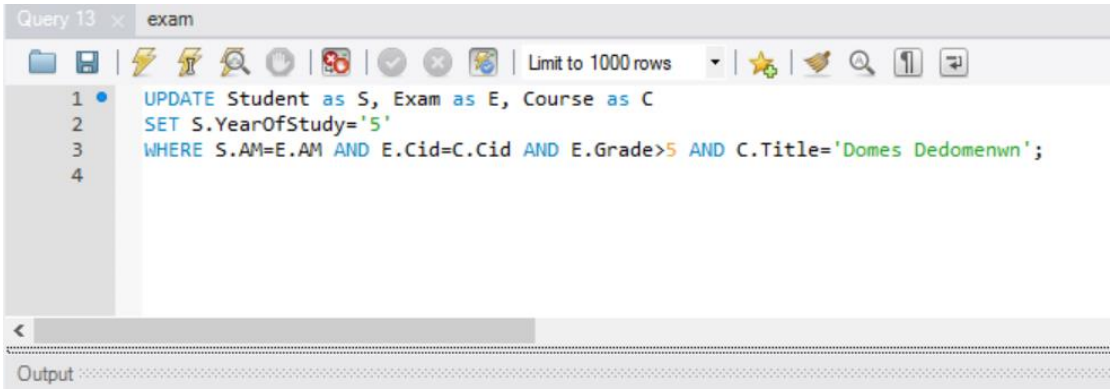


Εικόνα 11.2: *Παράδειγμα ενημέρωσης σε δύο πεδία.*

Ζητούμενο 3: Αλλάξτε το έτος σπουδών σε ‘5’ για τους φοιτητές που πέρασαν το μάθημα ‘Domes Dedomenon’.

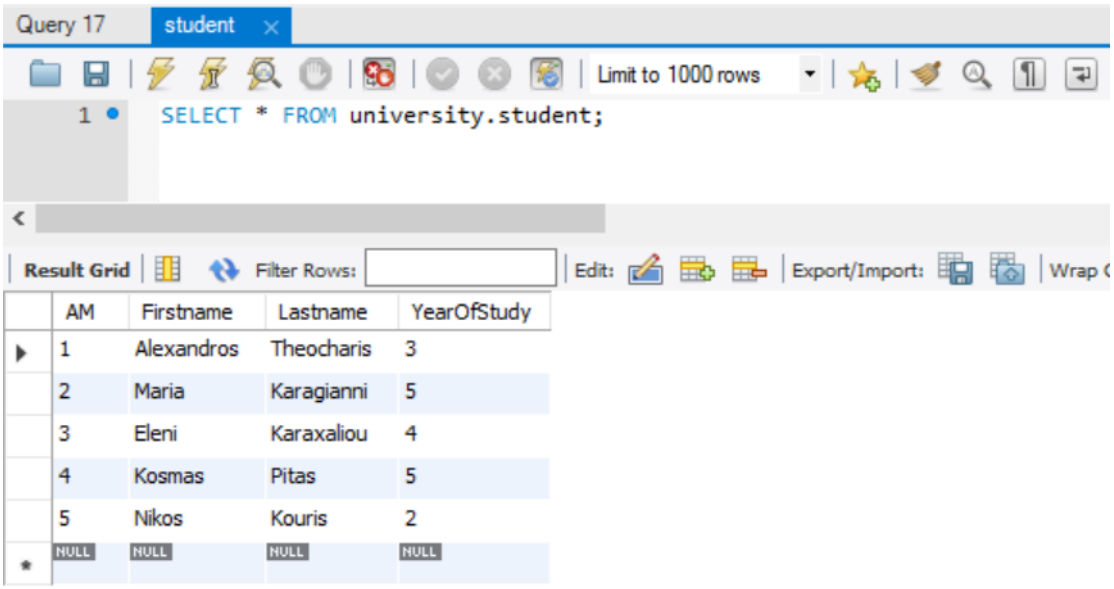
Σε αυτό το ερώτημα θα πρέπει να συνδυάσουμε τον πίνακα με τα στοιχεία των φοιτητών όπου εκεί θα γίνει η αλλαγή του έτους, τον πίνακα με τις βαθμολογίες για να γίνει έλεγχος βαθμού και τον πίνακα μαθημάτων για να ζητηθεί ο συγκεκριμένος τίτλος.
Δίνουμε (βλέπε και Εικόνα 11.3):

UPDATE Student as S, Exam as E, Course as C
SET S.YearOfStudy=’5’
WHERE S.AM=E.AM AND E.Cid=C.Cid AND E.Grade>5 AND C.Title=‘Domes Dedomenwn’;



Εικόνα 11.3: *Παράδειγμα ενημέρωσης με χρήση περισσοτέρων του ενός πινάκων.*

Τα αποτελέσματα φαίνονται στην Εικόνα 11.4 όπου άλλαξαν τα έτη σε δύο φοιτητές.

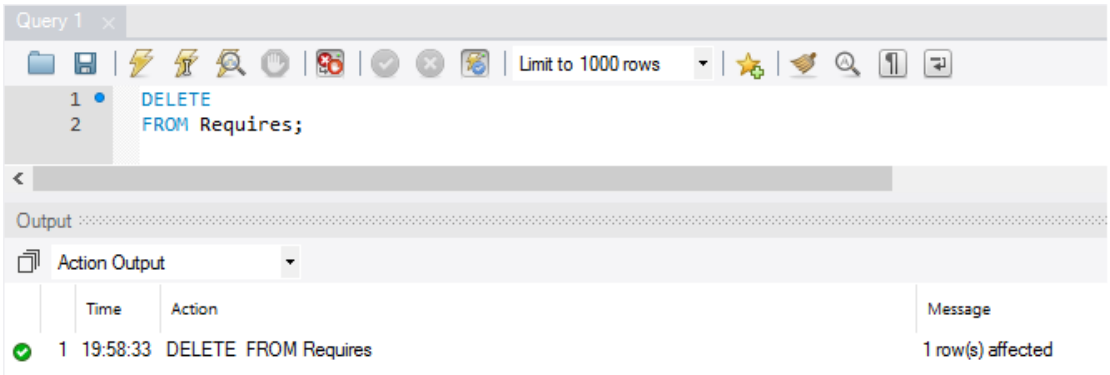


Εικόνα 11.4: *Αποτελέσματα της ενημέρωσης στο Ζητούμενο 3.*

Ζητούμενο 4: Σβήστε όλες τις εγγραφές του Requires.
Δίνουμε:

```
DELETE
FROM Requires;
```

Το αποτέλεσμα φαίνεται στην Εικόνα 11.5.

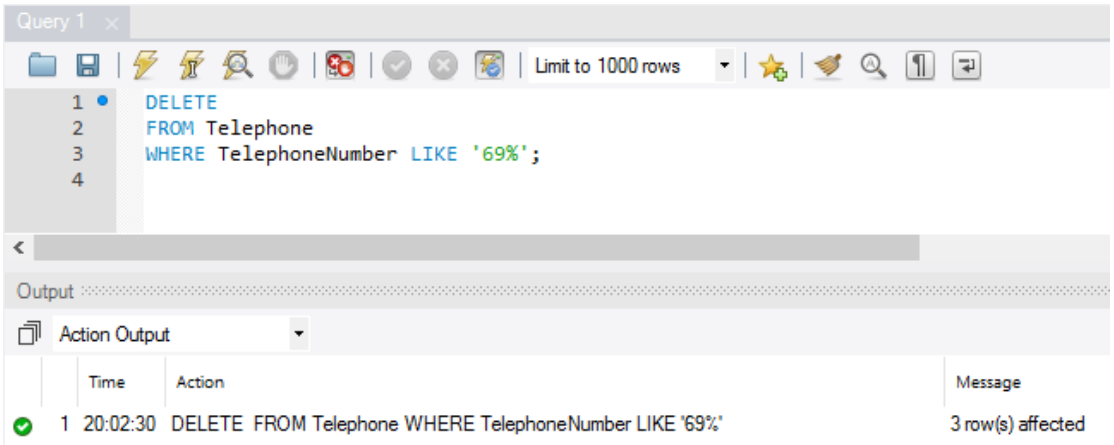


Εικόνα 11.5: *Παράδειγμα διαγραφής όλων των εγγραφών ενός πίνακα.*

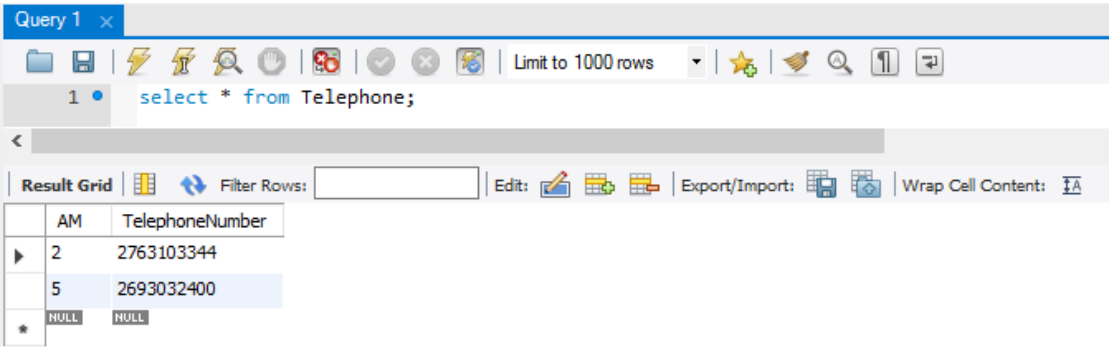
Ζητούμενο 5: Σβήστε όλα τα κινητά τηλέφωνα των φοιτητών.
Δίνουμε (βλέπε Εικόνα 11.6):

```
DELETE
FROM Telephone
WHERE TelephoneNumber LIKE ‘69%’;
```

Το αποτέλεσμα φαίνεται στην Εικόνα 11.7 η οποία δείχνει ότι έχουν απομείνει μόνο τα σταθερά τηλέφωνα στον πίνακα.



Εικόνα 11.6: *Παράδειγμα διαγραφής εγγραφών βάσει συνθήκης.*

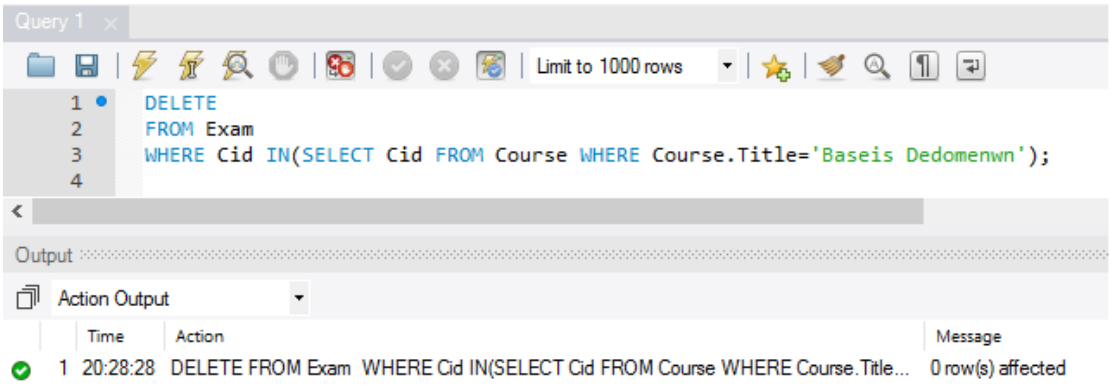


Εικόνα 11.7: *Αποτελέσματα της διαγραφής στο Ζητούμενο 5.*

Ζητούμενο 6: Σβήστε από τον Exam τις εγγραφές που αντιστοιχούν στο μάθημα ‘Baseis Dedomenwn’.

Το ερώτημα εμπλέκει δύο πίνακες, τον πίνακα βαθμολογιών και τον πίνακα μαθημάτων. Μπορεί να γραφεί χρησιμοποιώντας εμφωλευμένο ερώτημα ως εξής (βλέπε και Εικόνα 11.8):

```
DELETE
FROM Exam
WHERE Cid IN (SELECT Cid
              FROM Course
              WHERE Course.Title=’Baseis Dedomenwn’
              );
```



Εικόνα 11.8: *Παράδειγμα διαγραφής με χρήση εμφωλευμένου ερωτήματος.*

11.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 1), απαντήστε στα ακόλουθα ερωτήματα:

4. Προσθέστε ένα έτος εμπειρίας σε όλους τους διευθυντές.
5. Αλλάξτε την επωνυμία της εταιρίας Advance σε Computer Center.
6. Διαγράψτε τους εργαζόμενους που έχουν προϋπηρεσία μικρότερη από 10 χρόνια.
7. Διαγράψτε τις εταιρίες που ιδρύθηκαν πριν το 2000.

Άσκηση 2

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 2), απαντήστε στα ακόλουθα ερωτήματα:

4. Αφαιρέστε ένα βαθμό από όλες τις ταινίες.
5. Διαγράψτε τους σκηνοθέτες που έχουν σκηνοθετήσει τρεις ή περισσότερες ταινίες.
6. Διαγράψτε τον πίνακα Krites.
7. Αλλάξτε τα ονόματα δύο ταινιών.

Άσκηση 3

Χρησιμοποιώντας τη βάση δεδομένων που δημιουργήσατε στο κεφάλαιο 4 (άλυτη άσκηση 3), απαντήστε στα ακόλουθα ερωτήματα:

4. Διαγράψτε όλες τις εγγραφές του πίνακα Kritis.
5. Αλλάξτε τα ονόματα από δύο παίκτες.
6. Διαγράψτε όλα τα τουρνουά της Γαλλίας και Γερμανίας.
7. Προσθέστε ένα έτος(ηλικία) σε όλους τους παίκτες.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαια 8 και 9.
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
UPDATE.
http://dev.mysql.com/doc/refman/5.7/en/update.html
http://www.w3schools.com/sql/sql_update.asp
DELETE.
http://dev.mysql.com/doc/refman/5.7/en/delete.html
http://www.w3schools.com/sql/sql_delete.asp

Κεφάλαιο 12 Υλοποίηση Stored Procedures, Χρήση Ευρετηρίων

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν βασικά στοιχεία όσον αφορά τις stored procedures και τη χρήση ευρετηρίων.

Προαπαιτούμενη γνώση
Η ύλη των προηγούμενων κεφαλαίων του βιβλίου.

12.1 Εισαγωγικές Έννοιες

Μια stored procedure είναι ένα σύνολο από εντολές SQL που αποθηκεύεται μέσα στο σύστημα διαχείρισης βάσεων δεδομένων (πχ. MySQL server). Η διαδικασία αυτή έχει μοναδικό όνομα και μπορεί να κληθεί από άλλες διαδικασίες, εφαρμογές, εντολές χρηστών από τη διεπαφή (console, workbench) κλπ.

Συνήθως οι αποθηκευμένες διαδικασίες συμβάλλουν στην αύξηση της απόδοσης των εφαρμογών. Όταν καλείται μία διαδικασία, τότε μεταγλωττίζεται, φορτώνεται στην μνήμη και εκτελείται. Τα Συστήματα Διαχείρισης Βάσεων Δεδομένων, όπως η MySQL, τις αποθηκευμένες διαδικασίες που εκτελούνται πολύ συχνά τις διατηρούν σε κρυφή μνήμη cache. Αν κληθεί ένα ερώτημα που δεν βρίσκεται στην κρυφή μνήμη τότε εκτελείται σαν ένα απλό ερώτημα.

Οι διαδικασίες βοηθούν στο να μειωθεί η ανταλλαγή δεδομένων μεταξύ μίας εφαρμογής και του εξυπηρετητή. Αντί να στέλνονται ερωτήματα SQL πολλών χαρακτήρων, η εφαρμογή στέλνει μόνο το όνομα της διαδικασίας και τις κατάλληλες παραμέτρους. Επίσης οι διαδικασίες είναι επαναχρησιμοποιήσιμες από πολλές εφαρμογές ελαφρύνοντας την πολυπλοκότητα ανάπτυξης τους, μιας και οι προγραμματιστές απλώς καλούν λειτουργίες που παρέχονται.

Μέσω των αποθηκευμένων διαδικασιών μπορεί να ενισχυθεί η ασφάλεια, μιας και ο διαχειριστής μπορεί να δώσει τα κατάλληλα δικαιώματα σε εφαρμογές να καλέσουν τις διαδικασίες χωρίς να χρειάζεται να κάνει κάτι αντίστοιχο στους πίνακες που χρησιμοποιούνται από αυτές.

Αν χρησιμοποιούνται εκτεταμένα οι διαδικασίες, η μνήμη που χρησιμοποιείται από κάθε σύνδεση μπορεί να αυξηθεί σημαντικά. Επίσης αν υπάρχουν πάρα πολλές λογικές πράξεις μέσα στη διαδικασία, το βάρος της επεξεργασίας αυξάνει, μιας και ο επεξεργαστής δεν είναι βελτιστοποιημένος για κάθε δυνατή πράξη.

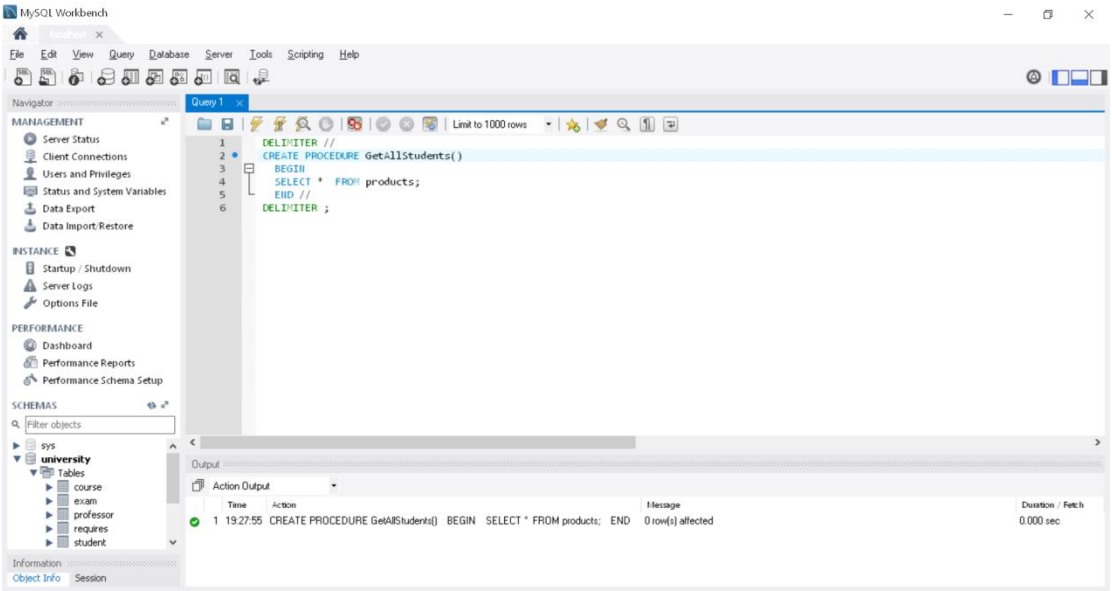
Επίσης, πολύ σύνθετες διαδικασίες που υλοποιούν πολύπλοκες λειτουργίες μιας εφαρμογής δεν είναι πάντα εύκολο να υλοποιηθούν. Ακόμα, η αποσφαλμάτωση των διαδικασιών δεν είναι πάντα εύκολη. Μερικά ΣΔΒΔ παρέχουν αυτή την δυνατότητα, σε άλλα θα πρέπει ο προγραμματιστής να εκτελεί αποσφάλματωση με την εκτύπωση μηνυμάτων στην console.

Ας δημιουργήσουμε μία απλή διαδικασία που περιέχει ένα ερώτημα select.

```
DELIMITER //

CREATE PROCEDURE GetAllStudents()

 BEGIN

        SELECT * FROM students;

 END //

 DELIMITER ;
```

Με χρήση της Delimeter ορίζουμε ότι η γραμμή μιας εντολής θα τερματίζει με το //. Εάν δεν τεθεί αυτό τότε θα χρησιμοποιηθεί σας τελεστής ο ;. Αν σε μία αποθηκευμένη διαδικασία θέλουμε να τοποθετήσουμε εντολές που τερματίζουν με το ; τότε θα πρέπει στον ορισμό της να αλλάξουμε τον χαρακτήρα αυτό. Σε αντίθετη περίπτωση θα εκτελεστεί ο κώδικας μέχρι το πρώτο ; που θα συναντηθεί.

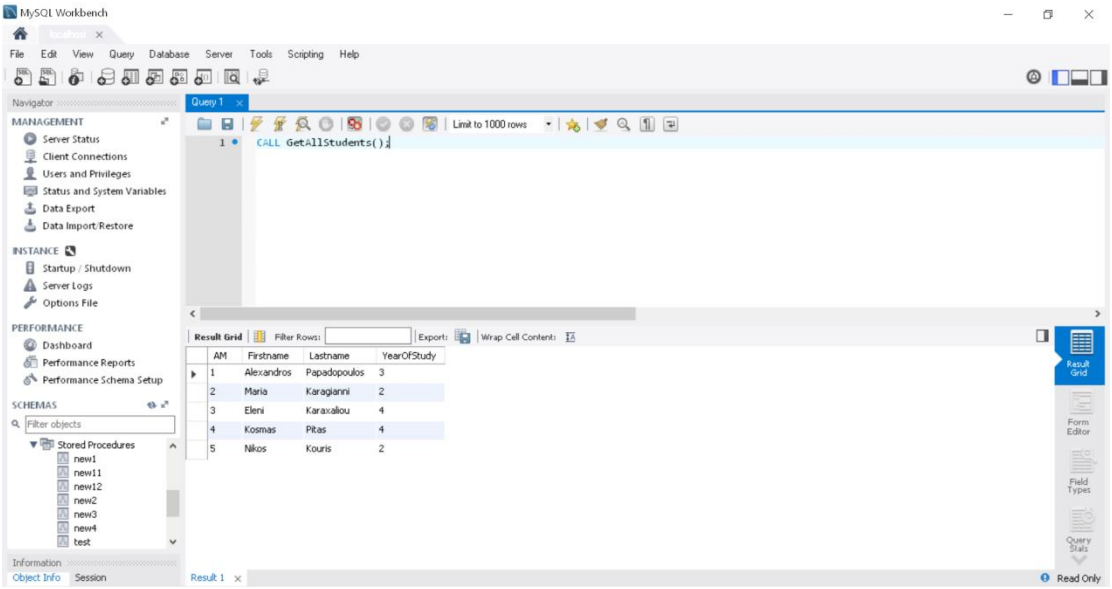
Στην παραπάνω διαδικασία δόθηκε ένα όνομα μετά την εντολή CREATE PROCEDURE, το οποίο πρέπει να είναι μοναδικό στο σχήμα(GetAllStudents). Δεν δόθηκαν ορίσματα στη διαδικασία μιας και δίπλα από το όνομα δεν χρησιμοποιήθηκαν παράμετροι παρά (). Στην συνέχεια εντός των εντολών BEGIN … END τοποθετούνται οι εντολές SQL που θα εκτελεστούν όταν κληθεί η διαδικασία. Στο παραπάνω παράδειγμα η SELECT * FROM students; Οι εντολές αυτές καλούνται σώμα της διαδικασίας.



Εικόνα 12.1: *Ορισμός διαδικασίας.*

Για την κλήση της διαδικασίας καλούμε CALL <όνομα της διαδικασίας> (<ορίσματα>). Για το παραπάνω παράδειγμα:

```
CALL GetAllStudents();
```



Εικόνα 12.2: *Κλήση διαδικασίας.*

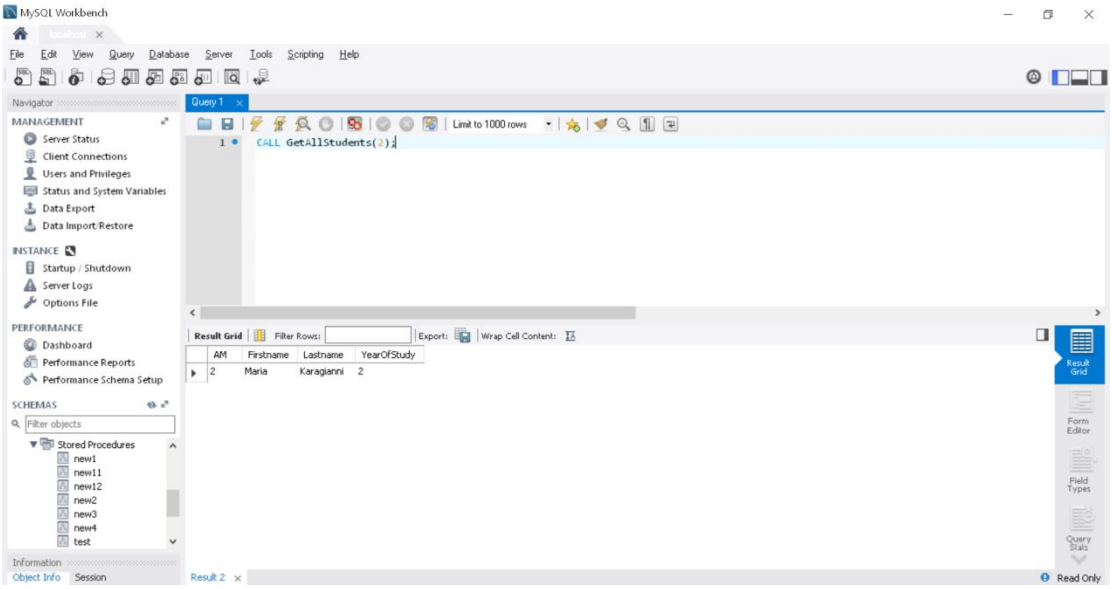
Στις αποθηκευμένες διαδικασίες υπάρχει η δυνατότητα να ορισθούν παράμετροι εισόδου

```
DELIMITER //
CREATE PROCEDURE GetAllStudents(IN am INT)
 BEGIN
 SELECT * FROM student where student.AM=am;
 END //
 DELIMITER ;
```

Καλώντας την διαδικασία με όρισμα:

Στο παραπάνω ερώτημα η παράμετρος χρησιμοποιείται στο where σκέλος της εντολής. Η κλήση της συνάρτησης θα είναι:

```
CALL GetAllStudents(2);
```



Εικόνα 12.3: *Κλήση διαδικασίας με όρισμα.*

Υπάρχει η δυνατότητα να χρησιμοποιηθούν και παράμετροι εξόδου. Αυτοί χαρακτηρίζονται ως OUT.

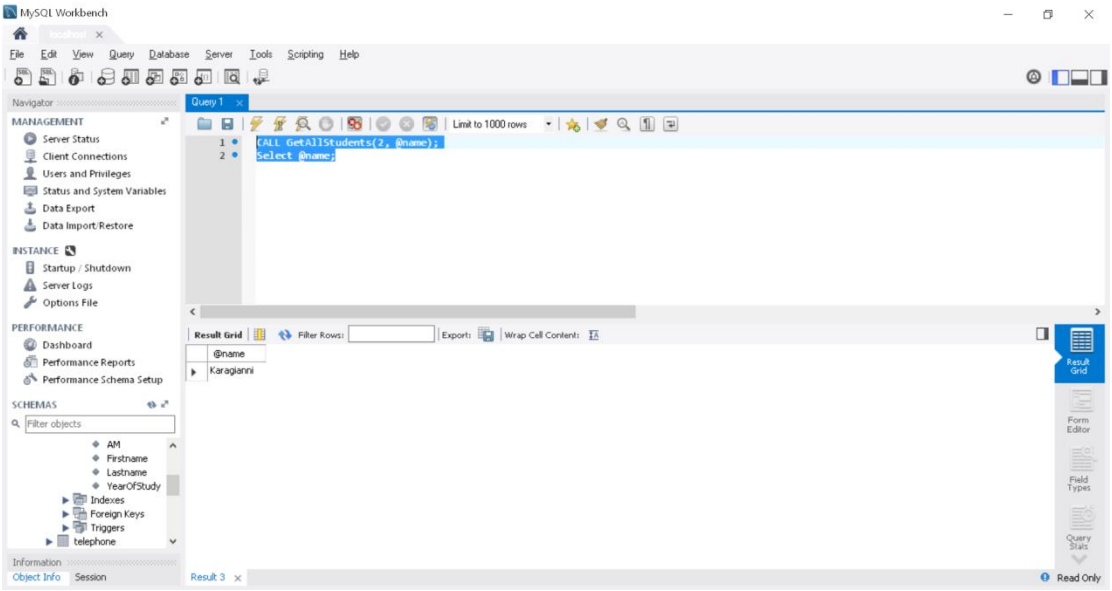
```
DELIMITER //
CREATE PROCEDURE GetAllStudents(IN am INT, OUT surname VARCHAR(24))
 BEGIN
        SELECT Lastname INTO surname FROM student where student.AM=am;
 END //
```

Στο παραπάνω παράδειγμα το χαρακτηριστικό Lastname θα ανατεθεί στο όρισμα εξόδου surname

Η κλήση της διαδικασίας θα πρέπει να γίνει ως εξής:

```
CALL GetAllStudents(2, @name);
Select @name;
```

ΜΕ @name δηλώνεται μία μεταβλητή περιβάλλοντος η οποία στην συνέχεια εκτυπώνεται με τη Select.



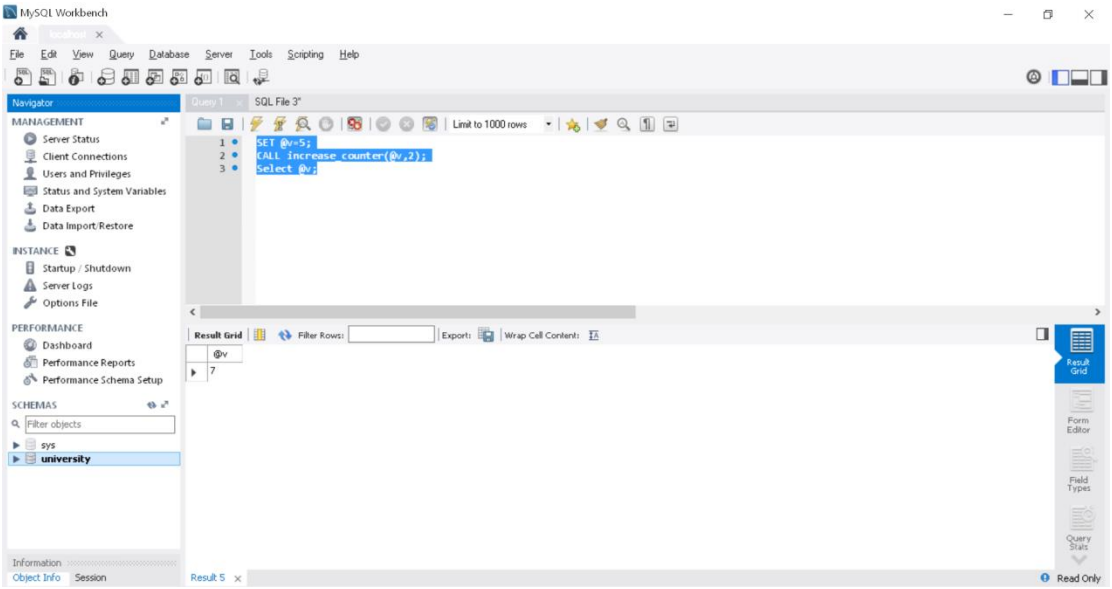
Εικόνα 12.4: *Κλήση διαδικασίας με όρισμα.*

Μία παράμετρος μπορεί να λειτουργήσει και ως IN και ως OUT και θα πρέπει να χαρακτηριστεί INOUT. Παραδείγματος χάριν, στο ακόλουθο παράδειγμα η παράμετρος count λειτουργεί ως INOUT αφού χρησιμοποιείται σαν είσοδος και στη συνέχεια επιστρέφεται αφού έχει προστεθεί σε αυτή το δεύτερο όρισμα.

```
DELIMITER //

CREATE PROCEDURE increase_counter(INOUT count INT(4),IN step INT(4))

BEGIN

 SET count = count + step;

END//

DELIMITER ;
```

Η κλήση θα γίνει ως εξής

```
SET @v=5;

CALL increase_counter(@v,2);

Select @v;
```



Εικόνα 12.5: *Κλήση διαδικασίας με όρισμα.*

Με την εντολή DECLARE είναι δυνατό να ορισθούν μεταβλητές στις διαδικασίες. Στην γενική περίπτωση η εντολή έχει την μορφή:

DECLARE variable_name datatype(size) DEFAULT default_value;

Π.χ. DECLARE lastname VARCHAR(25) DEFAULT ‘’;

Η ανάθεση τιμών γίνεται με την εντολή SET. Πχ.

Set lastname=’Kouris’;

Οι μεταβλητές όπως είδαμε παραπάνω, μπορούν να χρησιμοποιηθούν είτε εντός των διαδικασιών, είτε κατά την κλήση μιας διαδικασίας για να λάβουμε της παραμέτρους που επιστρέφονται.

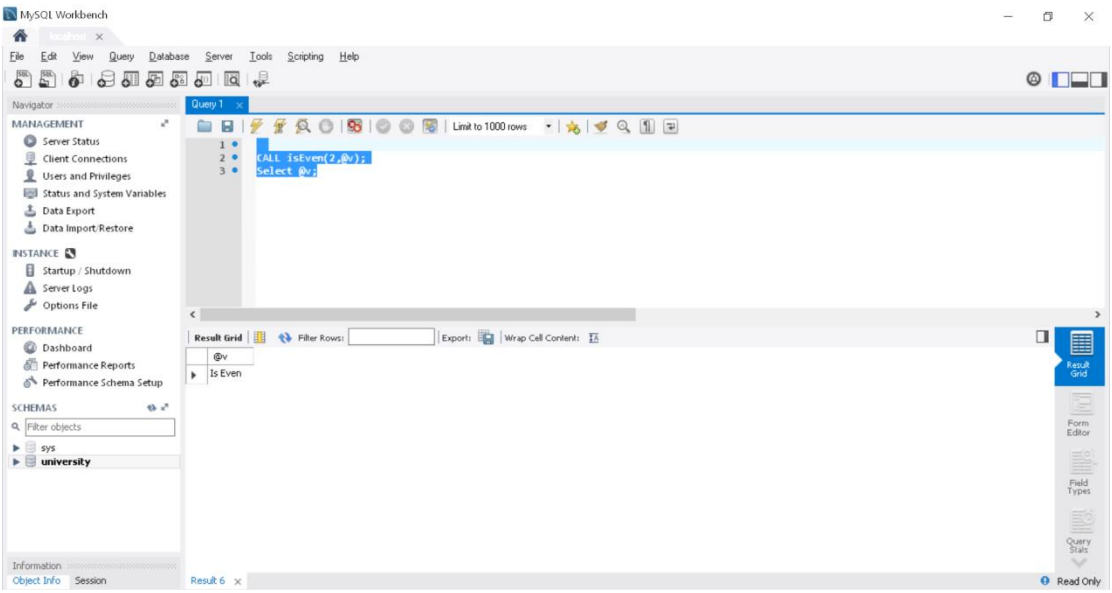
Στις αποθηκευμένες διαδικασίες, είναι δυνατό να υπάρξει έλεγχος ροής με χρήση της εντολής διακλάδωσης IF THEN ELSE.

IF expression THEN
  statements;
END IF;
	IF expression THEN
	  statements;
	ELSE
	  else-statements;
	END IF;
		IF expression THEN
		  statements;
		ELSEIF elseif-expression THEN
		  elseif-statements;
		...
		ELSE
		  else-statements;
		END IF;

```
DELIMITER //
CREATE PROCEDURE isEven(IN val INT, OUT result VARCHAR(11))
BEGIN
if mod(val,2)=0 then
 SET result='Is Even';
else
 SET result='Is Even';
end if;
END//
DELIMITER ;
```

Καλώντας

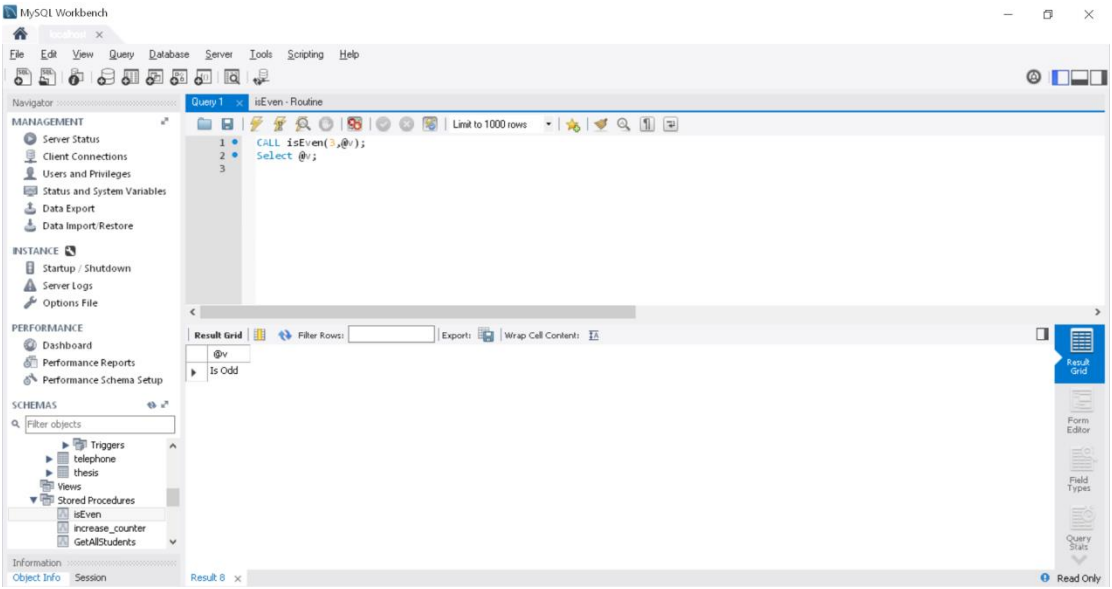
```
CALL isEven(2,@v);
Select @v;
```



Εικόνα 12.6: *Κλήση διαδικασίας.*

Καλώντας:

```
CALL isEven(3,@v);
Select @v;
```

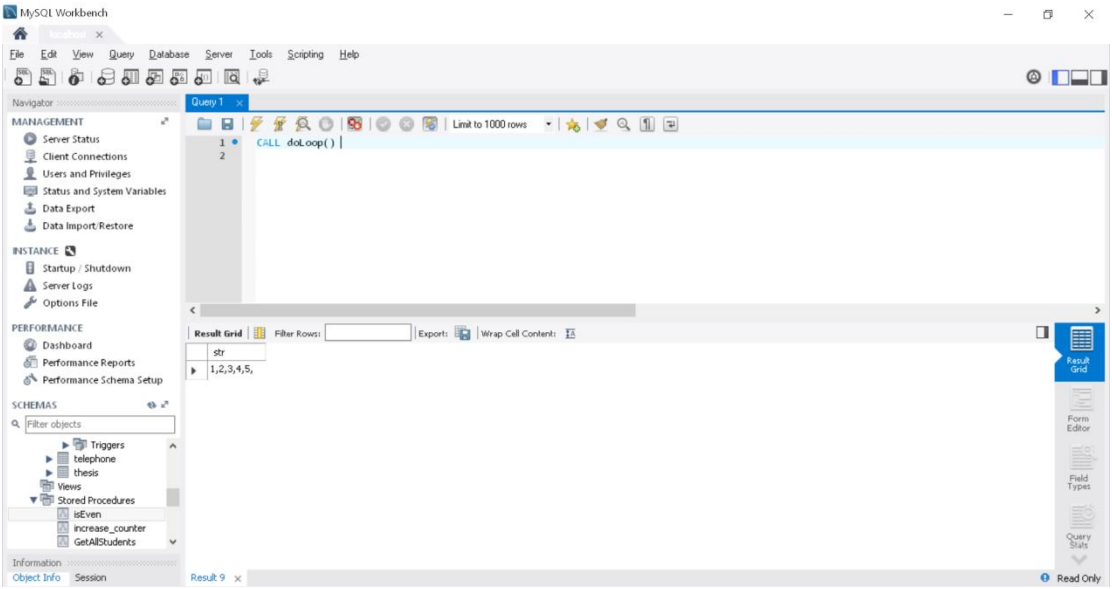


Εικόνα 12.7: *Κλήση διαδικασίας.*

Στις διαδικασίες είναι δυνατό να χρησιμοποιηθούν και εντολές επανάληψης πχ η WHILE με την μορφή:

```
WHILE expression DO
   statements
END WHILE
```

```
DELIMITER $$
 CREATE PROCEDURE doLoop()
 BEGIN
 DECLARE x  INT;
 DECLARE str  VARCHAR(255);
  SET x = 1;
 SET str =  '';
 WHILE x  <= 5 DO
          SET  str = CONCAT(str,x,',');
          SET  x = x + 1;
 END WHILE;
 SELECT str;
 END$$
DELIMITER ;
```



Εικόνα 12.8: *Κλήση διαδικασίας.*

Με αντίστοιχη λογική λειτουργεί η εντολή REPEAT … UNTIL μόνο που η εντολή ελέγχου εκτελείται στο τέλος κάθε επανάληψης.

REPEAT

    statements;

UNTIL expression

END REPEAT

Η επανάληψη με την χρήση cursor επιτρέπει να διατρέξουμε τα αποτελέσματα ενός ερωτήματος που είναι πίνακας. Ο ορισμός ενός cursor γίνεται με την εντολή DECLARE.

DECLARE cursor_name CURSOR FOR SELECT_statement;

OPEN cursor_name;

FETCH cursor_name INTO variables list;

CLOSE cursor_name;

Ας δούμε το παρακάτω παράδειγμα. Διατρέχει τα επώνυμα των φοιτητών και τα συνενώνει σε μια συμβολοσειρά.

DELIMITER $$

CREATE PROCEDURE getNames (INOUT v_student text)

BEGIN

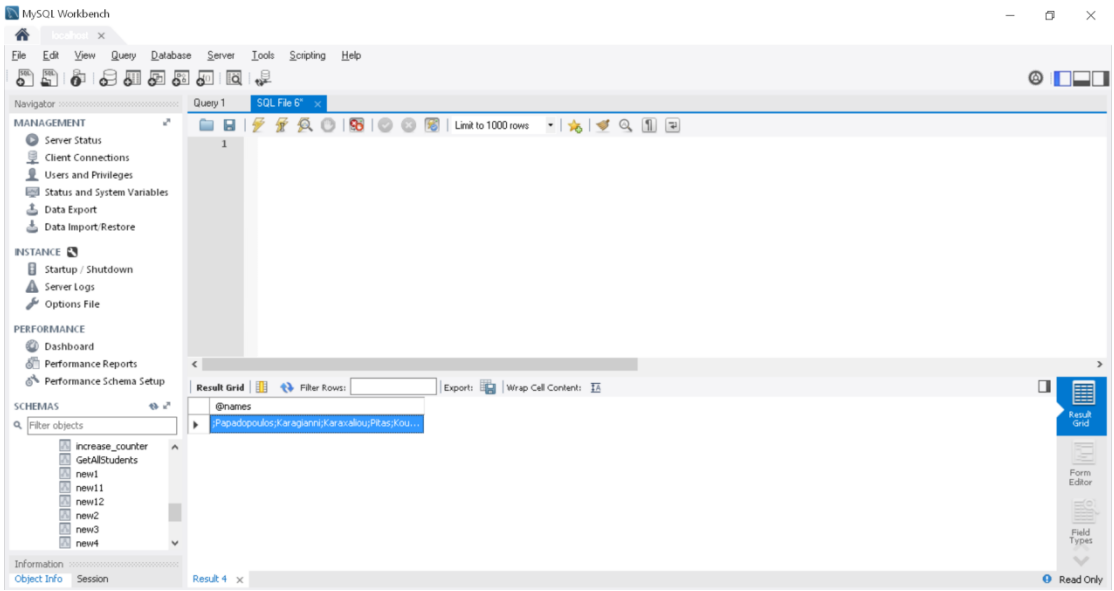
DECLARE done INT DEFAULT 0;

DECLARE Lastnames TEXT DEFAULT "";

DECLARE student_cursor CURSOR FOR SELECT Lastname FROM student;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

```
OPEN student_cursor;
get_lastname: LOOP
FETCH student_cursor INTO Lastnames;
SET v_student = CONCAT(v_student,";",Lastnames);
 IF done = 1 THEN
          LEAVE get_lastname;
       END IF;
END LOOP get_lastname;
CLOSE student_cursor;
END$$
DELIMITER ;

set @names='';
CALL getNames(@names);
Select @names;
```



Εικόνα 12.9: *Αποτέλεσμα κλήσης διαδικασίας.*

12.1.1 Ευρετήρια

Για την δημιουργία ενός ευρετηρίου χρησιμοποιείται η εντολή CREATE INDEX με γενική μορφή:

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name

USING [BTREE | HASH | RTREE]

ON table_name (column_name [(length)] [ASC | DESC],...)

- Unique σημαίνει ότι η MySQL θα δημιουργήσει τον περιορισμό όλες οι τιμές του καταλόγου/ευρετηρίου να εμφανίζονται από μια φορά.
- FULLTEXT υποστηρίζεται μόνο από το MyISAM μηχανισμό αποθήκευσης και λειτουργεί για τους τύπους CHAR, VARCHAR, TEXT.

Παράδειγμα:

CREATE INDEX officeCode ON employees(officeCode)

Ανάλογα με τον μηχανισμό αποθήκευσης δεδομένων υποστηρίζονται και άλλα ευρετήρια:

- MyISAM:BTREE, RTREE
- InnoDB: BTREE
- MEMORY/HEAP: HASH, BTREE

Η χρησιμοποίηση των ευρετηρίων Β-δέντρου και κατακερματισμού μπορεί να επιταχύνει την απόδοση ερωτημάτων. Ανάλογα με το τύπο του ευρετηρίου μπορούν να επιταχυνθούν διαφορετικά ερωτήματα.

12.1.2 B-trees

Ένα ευρετήριο Β-δέντρου μπορεί να χρησιμοποιηθεί για ερωτήματα που συγκρίνουν χαρακτηριστικά που χρησιμοποιούν τους τελεστές =,>,> =, <, <=,BETWEEN. Ο κατάλογος μπορεί επίσης να χρησιμοποιηθεί για LIKE συγκρίσεις, αν η παράμετρος σύγκρισης είναι ένα σταθερό αλφαριθμητικό που δεν ξεκινά με ένα χαρακτήρα μπαλαντέρ %. Για παράδειγμα, τα ακόλουθα ερωτήματα SELECT επιταχύνονται με την χρήση ευρετηρίων Β-Δέντρου:

SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';

SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';

Οι ακόλουθες SELECT δηλώσεις δεν χρησιμοποιούν ευρετήρια:

SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';

SELECT * FROM tbl_name WHERE key_col LIKE other_col;

Στην πρώτη δήλωσή της, η LIKE τιμή ξεκινά με ένα χαρακτήρα μπαλαντέρ(wildcard). Στη δεύτερη ανακοίνωση, η LIKE τιμή δεν είναι σταθερή.

Κάθε ευρετήριο που δεν καλύπτει όλους τους ελέγχους σε ένα σκέλος WHERE ενός ερωτήματος, δεν χρησιμοποιείται για την βελτιστοποίηση του ερωτήματος.

... WHERE index_part1=1 AND index_part2=2 AND other_column=3

/* index = 1 OR index = 2 */

... WHERE index=1 OR A=10 AND index=2

/* optimized like "index_part1='hello'" */

... WHERE index_part1='hello' AND index_part3=5

/* Can use index on index1 but not on index2 or index3 */

... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;

Στα παρακάτω ερωτήματα δεν χρησιμοποιούνται ερωτήματα

/* index_part1 is not used */

... WHERE index_part2=1 AND index_part3=2

/* Index is not used in both parts of the WHERE clause */

... WHERE index=1 OR A=10

/* No index spans all rows */

... WHERE index_part1=1 OR index_part2=10

12.1.3 Ευρετήριο HASH

Τα Hash ευρετήρια χρησιμοποιούνται για την βελτιστοποίηση άλλων ερωτημάτων.

Συγκεκριμένα:

- Χρησιμοποιούνται μόνο για συγκρίσεις ισότητας που χρησιμοποιούν τους = ή <=> τελεστές (και είναι πολύ γρήγορα). Δεν χρησιμοποιούνται για τελεστές σύγκρισης, όπως <, που δημιουργούν επιλογή σε ένα εύρος τιμών.
- Ένα ερώτημα δεν μπορεί να χρησιμοποιήσει ένα κατάλογο κατακερματισμού για να επιταχυνθεί όταν χρησιμοποιεί την ενέργεια ORDER BY.
- Μόνο ολόκληρα κλειδιά μπορούν να χρησιμοποιηθούν για την αναζήτηση μιας εγγραφής σε έναν πίνακα. (Με ευρετήριο B-δέντρου, κάθε πρόθεμα ενός κλειδιού μπορεί να χρησιμοποιηθεί για να βρουν γραμμές).

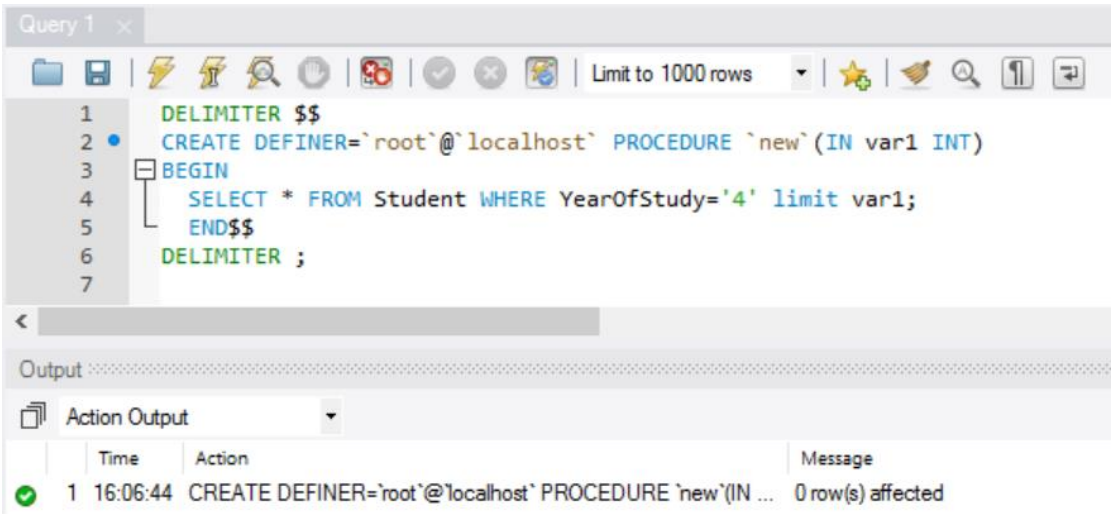
12.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 1. Χρησιμοποιώντας Procedure να επιστρέψετε τους 5 πρώτους φοιτητές του 4ου έτους. 2. Χρησιμοποιώντας Procedure(με OUT μεταβλητή) να βρείτε το μέγιστο βαθμό που έγραψαν οι φοιτητές στο μάθημα "Baseis Dedomenwn". 3. Χρησιμοποιώντας Procedure(με CASE), βρείτε πόσοι φοιτητές πέρασαν το μάθημα "Domes Dedomenwn" και πόσοι κόπηκαν. 4. Ποιο τύπο ευρετηρίου θα χρησιμοποιούσατε για τα ακόλουθα ερωτήματα: a. Αναζήτηση φοιτητή με βάση το ΑΜ b. Αναζήτηση φοιτητή με βάση το επίθετο c. Αναζήτηση φοιτητών με βάση ενός εύρους ΑΜ (πχ. Οι φοιτητές με ΑΜ από 1923-2312) d. Επιλογή μαθήματος με βάση τον τίτλο
Ζητούμενα:	Stored Procedures, βασικά ευρετήρια B-trees, Hash, χρήση ευρετηρίων σε βασικά ερωτήματα.

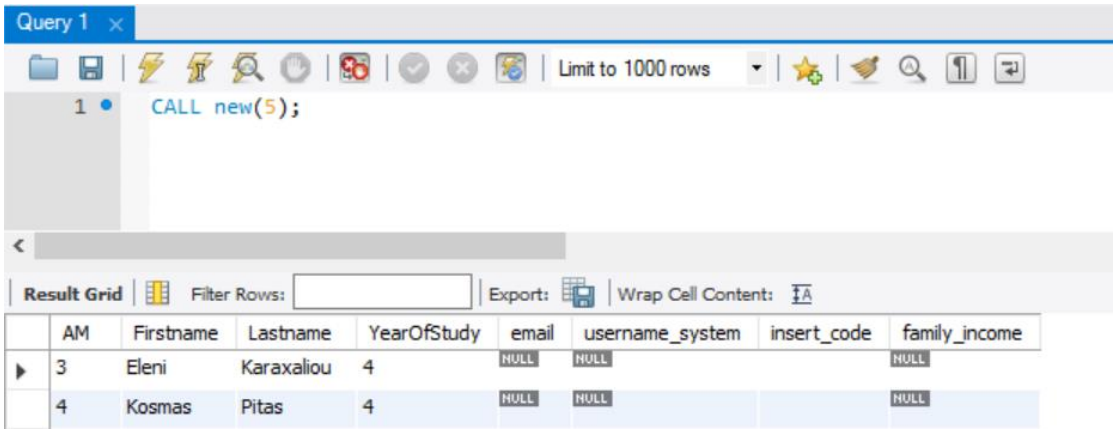
1. Χρησιμοποιώντας Procedure να επιστρέψετε τους 5 πρώτους φοιτητές του 4ου έτους.

Ορίζουμε τη Procedure "new" η οποία χρησιμοποιεί τη μεταβλητή var1, που δέχεται έναν αριθμό από το χρήστη. Στη συγκεκριμένη περίπτωση θέλουμε τους 5 πρώτους φοιτητές του 4ου έτους, οπότε στην κλήση την Procedure βάζουμε την τιμή 5.

```
DELIMITER $$
CREATE PROCEDURE new(IN var1 INT)
 BEGIN
 SELECT * FROM Student WHERE YearOfStudy='4' LIMIT var1;
 END$$
-------------------------------------------------------
CALL new(5);
```



Εικόνα 12.10: *Η διαδικασία new.*

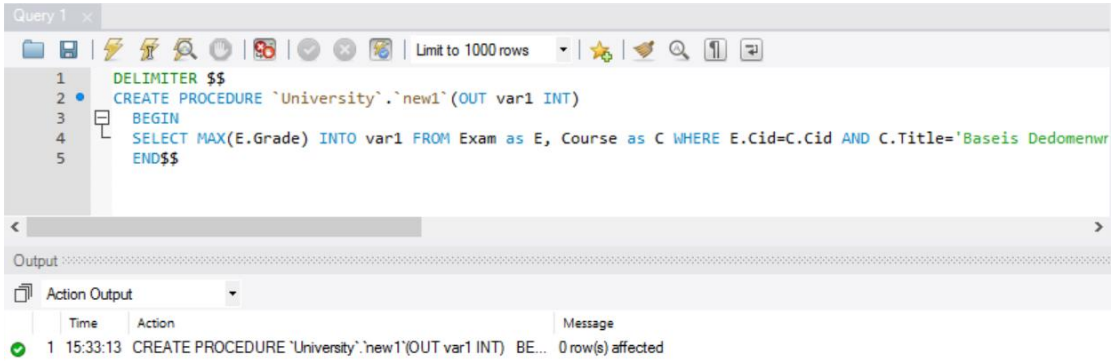


Εικόνα 12.11: *Κλήση διαδικασίας.*

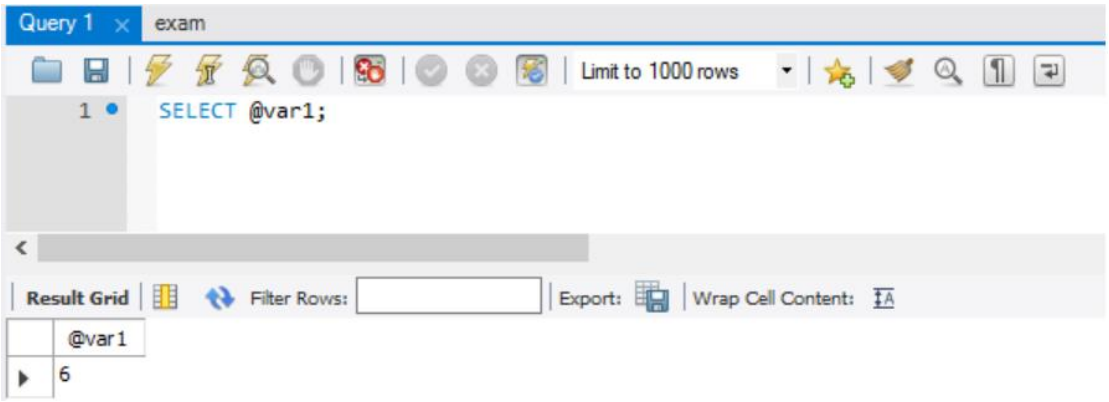
2. Χρησιμοποιώντας Procedure(με OUT μεταβλητή) να βρείτε το μέγιστο βαθμό που έγραψαν οι φοιτητές στο μάθημα “Baseis Dedomenwn”.

Ορίζουμε τη Procedure “new1” η οποία χρησιμοποιεί τη μεταβλητή var1. Μετά την κλήση της Procedure, η λέξη OUT λέει στη βάση δεδομένων ότι η τιμή βγαίνει έξω από τη διαδικασία. Για να δούμε το αποτέλεσμα, καλούμε τη διαδικασία βάζοντας ως όρισμα @var1 και για να εμφανίσουμε το αποτέλεσμα γράφουμε SELECT @var1.

```
DELIMITER $$
CREATE PROCEDURE `University`.`new1`(OUT var1 INT)
 BEGIN
 SELECT MAX(E.Grade) INTO var1 FROM Exam as E, Course as C WHERE E.Cid=C.Cid AND C.Title=’Baseis Dedomenwn’;
 END$$
-------------------------------------------------------------
CALL new1(@var1);

SELECT @var1;
```



Εικόνα 12.12: *Η διαδικασία new1.*

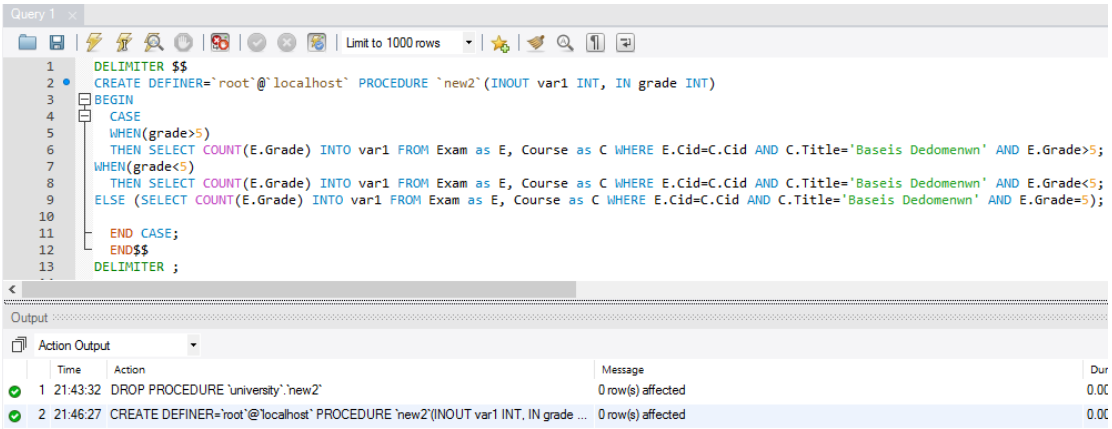


Εικόνα 12.13: *Κλήση διαδικασίας.*

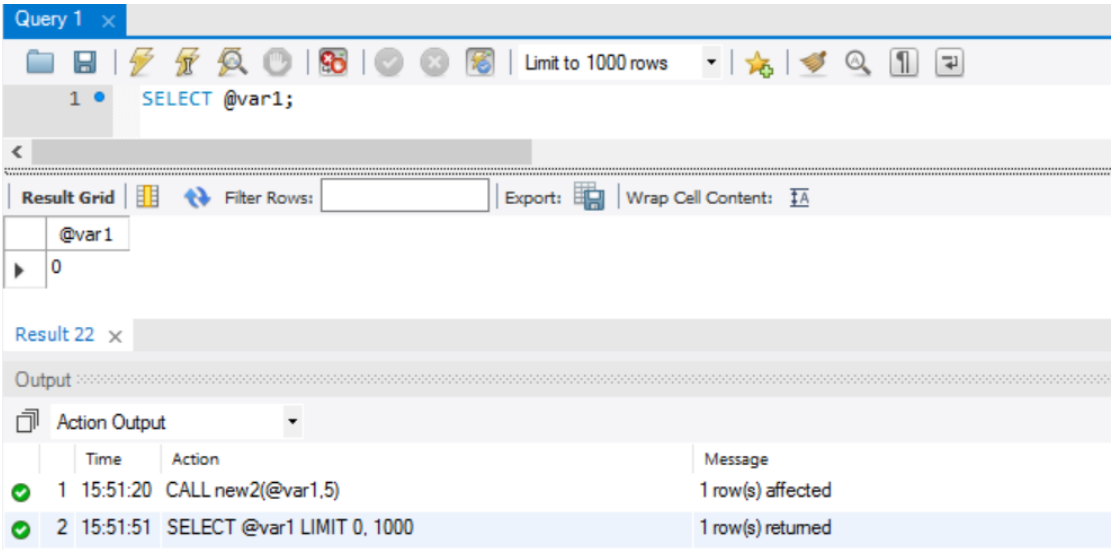
3. Χρησιμοποιώντας Procedure(με CASE), βρείτε πόσοι φοιτητές πέρασαν το μάθημα "Baseis Dedomenwn", πόσοι κόπηκαν και πόσοι πήραν 5.

Στην περίπτωση που θέλουμε να χρησιμοποιήσουμε εντολές που χρειάζονται ερωτηματικό (;) στο τέλος για να δηλωθούν τότε ορίζουμε delimeter στην αρχή του προγράμματος για να το διαφοροποιήσουμε από τις εσωτερικές εντολές. Για να βρούμε το πλήθος των φοιτητών που έγραψαν βαθμό μεγαλύτερο, μικρότερο ή ίσο με 5 χρησιμοποιούμε τη συνάρτηση COUNT(). Με την CASE εξετάζουμε τις διάφορες περιπτώσεις. Στην κλήση της διαδικασίας, στη θέση του ορίσματος grade βάζουμε 5 για να δούμε πόσοι φοιτητές έχουν γράψει 5, έναν αριθμό μεγαλύτερο του 5 για να δούμε πόσοι φοιτητές έχουν γράψει πάνω από 5 και έναν αριθμό μικρότερο του 5 για να δούμε πόσοι φοιτητές έχουν γράψει κάτω από 5.

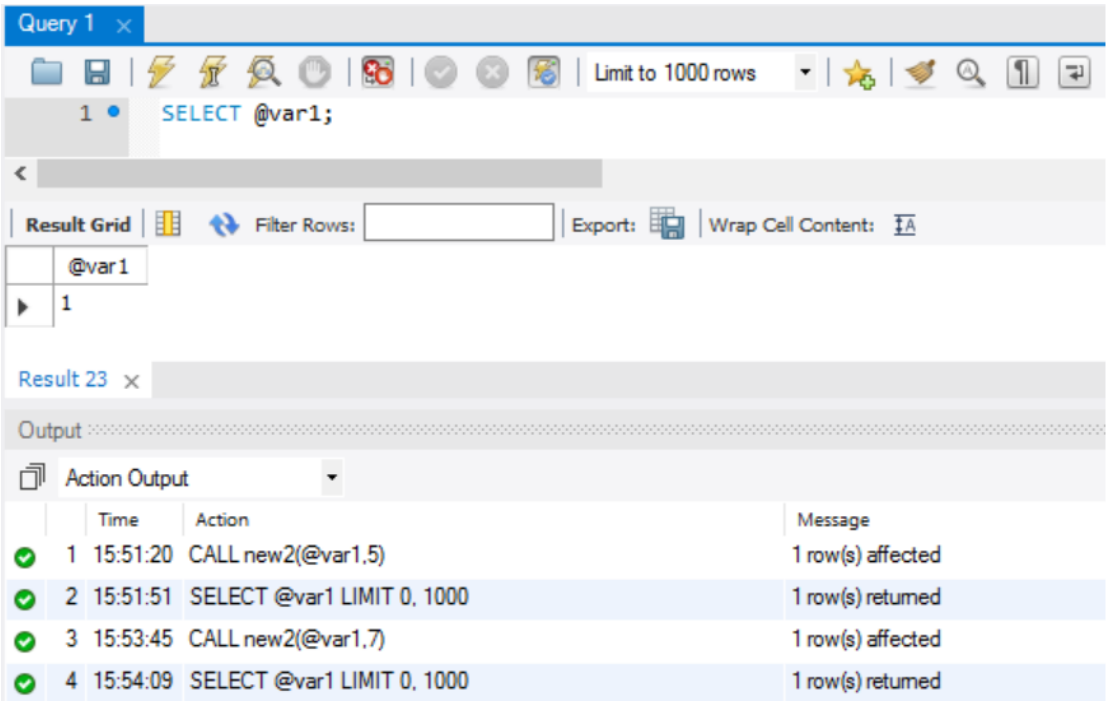
```
DELIMITER $$
CREATE PROCEDURE new2(INOUT var1 INT, IN grade INT)
 BEGIN
 CASE
 WHEN(grade>5)
 THEN SELECT COUNT(E.Grade) INTO var1 FROM Exam as E, Course as C WHERE E.Cid=C.Cid AND
C.Title='Baseis Dedomenwn' AND E.Grade>5;
WHEN(grade<5)
 THEN SELECT COUNT(E.Grade) INTO var1 FROM Exam as E, Course as C WHERE E.Cid=C.Cid AND
C.Title='Baseis Dedomenwn' AND E.Grade<5;
ELSE (SELECT COUNT(E.Grade) INTO var1 FROM Exam as E, Course as C WHERE E.Cid=C.Cid AND
C.Title='Baseis Dedomenwn' AND E.Grade=5);

 END CASE;
 END $$
-------------------------------------------------------
CALL new2(@var1,5);

SELECT @var1;
```



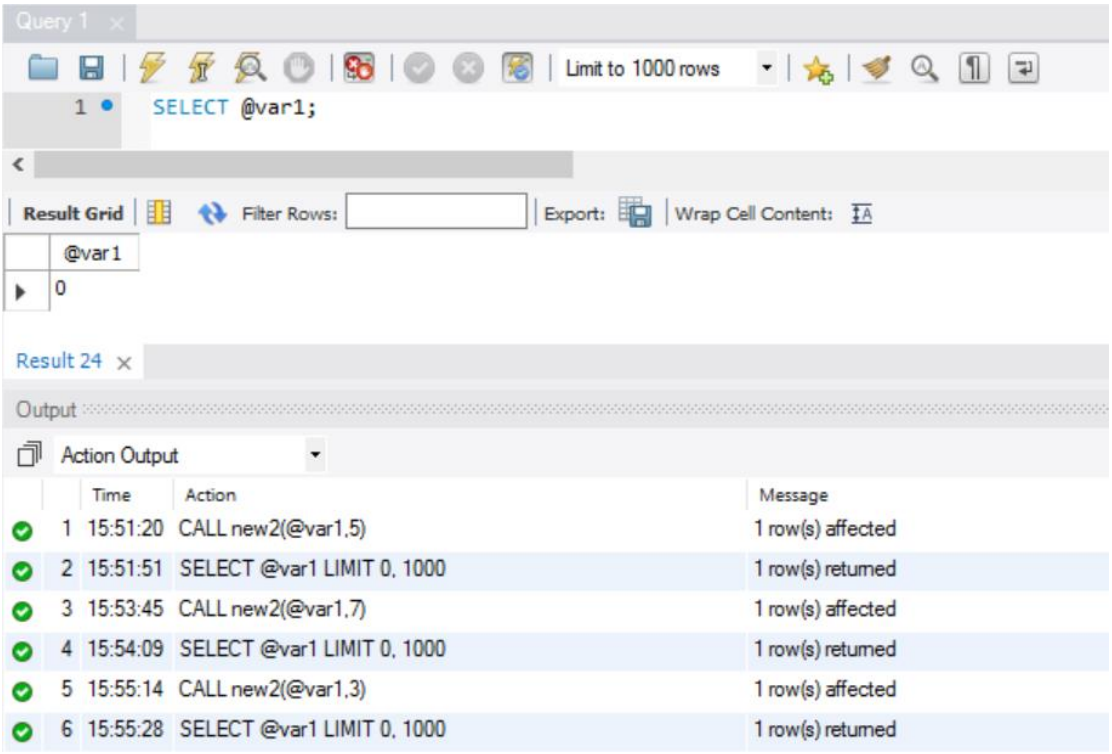
Εικόνα 12.14: *Η διαδικασία new2.*



Εικόνα 12.15: *Κλήση της διαδικασίας με όρισμα 5.*



Εικόνα 12.16: *Κλήση της διαδικασίας με όρισμα 7.*



Εικόνα 12.17: *Κλήση της διαδικασίας με όρισμα 3.*

4. Ποιο τύπο ευρετηρίου θα χρησιμοποιούσατε για τα ακόλουθα ερωτήματα:
    a. Αναζήτηση φοιτητή με βάση το ΑΜ
    b. Αναζήτηση φοιτητή με βάση το επίθετο
    c. Αναζήτηση φοιτητών με βάση ενός εύρους ΑΜ (πχ. Οι φοιτητές με ΑΜ από 1923-2312)
    d. Επιλογή μαθήματος με βάση τον τίτλο

Για τα ερωτήματα α) β) αναζήτησης που θα είναι της μορφής

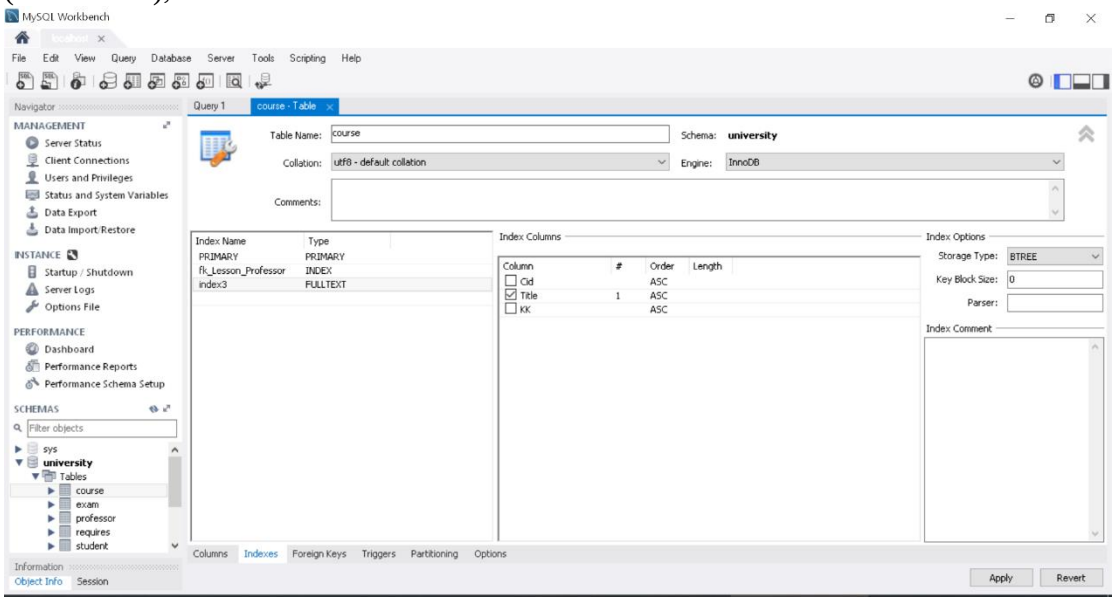
Select * from student where AM=1234;

Select * from student where Lastname = 'Theodoridis';

Θα επιλεγεί ευρετήριο κατακερματισμού.

ALTER TABLE `university`.`course` ADD FULLTEXT INDEX `index1` USING HASH (`AM` ASC);

ALTER TABLE `university`.`course` ADD FULLTEXT INDEX `index3` USING HASH (`Title` ASC);



Εικόνα 12.18: *Ευρετήρια.*

Για τα ερωτήματα γ) και δ) που είναι ερωτήματα εύρους
Select * from student where AM>1234 AND AM < 2543;

Select * from student where Lastname LIKE 'Theodoridis% ';

ALTER TABLE `university`.`course` ADD FULLTEXT INDEX `index1` USING BTREE (`AM` ASC);

ALTER TABLE `university`.`course` ADD FULLTEXT INDEX `index3` USING BTREE (`Title` ASC);

12.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

1. Χρησιμοποιώντας procedure εμφανίστε πόσοι εργαζόμενοι έχουν προϋπηρεσία μεγαλύτερη από 10 χρόνια, πόσοι έχουν μικρότερη από 10 χρόνια και πόσοι ίση με 10 χρόνια.
2. Χρησιμοποιώντας procedure βρείτε τους εργαζόμενους που δουλεύουν στην εταιρία “Advance”.
3. Χρησιμοποιώντας procedure βρείτε τους διευθυντές που έχουν τα λιγότερα έτη προϋπηρεσίας.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαια 9.
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 5.
CREATE PROCEDURE http://dev.mysql.com/doc/refman/5.7/en/create-procedure.html

Κεφάλαιο 13 Δοσοληψίες

Σύνοψη

Στο παρόν κεφάλαιο θα παρουσιασθούν βασικά στοιχεία όσον αφορά τις δοσοληψίες, την έννοια της σειριοποιησιμότητας και των διαφόρων επιπέδων απομόνωσης.

Προαπαιτούμενη γνώση

Η ύλη των προηγούμενων κεφαλαίων του βιβλίου.

13.1 Εισαγωγικές Έννοιες

Με τον όρο **δοσοληψία,** εννοούμε μια μονάδα εργασίας που μπορεί να έχει πρόσβαση σε διάφορα δεδομένα, διαβάζοντας ή ενημερώνοντάς τα. Είναι μια σειρά από ενέργειες, οι οποίες διαβάζουν ή γράφουν αντικείμενα της βάσης. Όταν εκτελείται μια δοσοληψία, η βάση δεδομένων μπορεί να αλλάζει. Με το πέρας της δοσοληψίας όμως, η βάση πρέπει να παραμένει σταθερή.

Δύο είναι τα βασικά προβλήματα με τις δοσοληψίες:

1. Τι θα γίνει αν κατά τη διάρκεια της εκτέλεσης, πέσει το σύστημα;
2. Τι θα γίνει αν δύο δοσοληψίες επιχειρούν να μεταβάλλουν το ίδιο αντικείμενο ταυτοχρόνως;

Για να διασφαλίσει την ακεραιότητα των δεδομένων το σύστημα βάσης δεδομένων πρέπει να εξασφαλίσει:

1. **Ατομικότητα**: είτε **όλες** οι πράξεις της δοσοληψίας επιτυγχάνουν, είτε **όλες** αποτυγχάνουν.
2. **Συνέπεια**: στο τέλος της δοσοληψίας, η βάση πρέπει να είναι σε συνεπή μορφή.
3. **Απομόνωση**: ακόμα κι αν τρέχουν πολλές δοσοληψίες ταυτόχρονα, **κάθε δοσοληψία πρέπει να νομίζει ότι τρέχει μόνη της.**
4. **Μονιμότητα**: αν η δοσοληψία επιτύχει, πρέπει το αποτέλεσμά της να επιβιώνει, ακόμα κι αν αποτύχει το σύστημα.

Το παραπάνω είναι διεθνώς γνωστό σαν **ACID test**

(A)tomicity	**Ατομικότητα**
(C)onsistency	**Συνέπεια**
(I)solation	**Απομόνωση**
(D)urability	**Μονιμότητα**

Καταστάσεις δοσοληψιών

- **Active:** στο ξεκίνημα και κατά τη διάρκειά της. Η δοσοληψία μένει σ' αυτή την κατάσταση ενώ εκτελείται
    - **Failed:** όταν το DBMS αντιληφθεί ότι η δοσοληψία δεν μπορεί να συνεχίσει
- **Aborted:** όταν η αποτυχημένη δοσοληψία έχει αναιρεθεί από το σύστημα και η ΒΔ είναι σε συνεπή μορφή

- **Committed:** όταν η δοσοληψία επιτύχει και η ΒΔ είναι σε συνεπή μορφή.
- **Partially committed:** όταν έχει εκτελεστεί η τελευταία εντολή της δοσοληψίας

Συμπεραίνουμε λοιπόν, ότι η δοσοληψία τελειώνει είτε με μια δήλωση **COMMIT** που ολοκληρώνει επιτυχώς τη δοσοληψία, μονιμοποιώντας τις αλλαγές στη βάση δεδομένων, είτε με δήλωση **ABORT (ROLLBACK για την MySQL)**.

Σειριοποιησιμότητα

Βασικό ζητούμενο είναι σε κάθε δοσοληψία να διατηρείται η συνέπεια της βάσης. Η σειριακή εκτέλεση των δοσοληψιών (σειριακό χρονοπρόγραμμα) εγγυάται τη συνέπεια της βάσης.

Σειριοποιήσιμο είναι το χρονοπρόγραμμα που εγγυημένα έχει το ίδιο αποτέλεσμα με ένα πλήρες σειριακό χρονοπρόγραμμα. Έτσι, αν μας δοθεί ένα χρονοπρόγραμμα, πρέπει να μπορέσουμε να αποφανθούμε αν είναι σειριοποιήσιμο ή όχι και αυτό μπορούμε να το πετύχουμε με δυο τεχνικές: την **σειριοποιησιμότητα συγκρούσεων** και την **σειριοποιησιμότητα όψεως.** Στο εξής θα αγνοούμε το processing στη μνήμη και θα μας απασχολούν μόνο οι read και write αλληλεπιδράσεις με τη βάση δεδομένων. Εδώ υποθέτουμε ότι οι δοσοληψίες μπορούν να εκτελέσουν τους αυθαίρετους υπολογισμούς τους στα δεδομένα σε τοπικούς buffer ανάμεσα σε read και write.

Σειριοποιησιμότητα συγκρούσεων

Έστω η εντολή I_1 της δοσοληψίας T_1 και η εντολή I_2 της T_2. Τότε αυτές συγκρούονται αν και μόνο αν και οι δυο έχουν πρόσβαση σε ένα αντικείμενο Q και τουλάχιστον μια απ' αυτές γράφει το Q.

Παράδειγμα

1. I_1 = **read** (Q) , I_2 = **read** (Q) Δεν Συγκρούονται
2. I_1 = **read** (Q) , I_2 = **write** (Q) Συγκρούονται
3. I_1 = **write** (Q) , I_2 = **read** (Q) Συγκρούονται
4. I_1 = **write** (Q) , I_2 = **read** (Q) Δεν Συγκρούονται

Αν ένα χρονοπρόγραμμα S μετασχηματιστεί σε ένα χρονοπρόγραμμα S' από μια σειρά εναλλασσόμενων - μη συγκρουόμενων εντολών, λέμε ότι τα S και S' είναι **ισοδύναμα συγκρούσεων.** Δηλαδή, δύο χρονοπρογράμματα είναι ισοδύναμα συγκρούσεων, αν για κάθε σύγκρουση οι συγκρουόμενες εντολές έχουν την ίδια σειρά στα δύο προγράμματα.

Ένα χρονοπρόγραμμα είναι **σειριοποιήσιμο συγκρούσεων,** αν είναι ισοδύναμο συγκρούσεων με ένα σειριακό (δηλαδή, αν όλες οι συγκρουόμενες πράξεις έχουν την ίδια σειρά που θα είχαν σε ένα σειριακό).

Για την καλύτερη κατανόηση των παραπάνω, σημειώνουμε τυπικά ότι:

- στο σειριακό χρονοπρόγραμμα, κάθε δοσοληψία «ξεμπερδεύει» ξεχωριστά (σε απομόνωση) με κάθε αντικείμενο και μετά το «αναλαμβάνει» μια άλλη.
- σε ένα σειριοποιήσιμο, το να ΜΗΝ υπάρχει σύγκρουση σημαίνει ότι το χρονοπρόγραμμα «ξεμπερδεύει» με τα αντικείμενα με την ίδια σειρά ανά δοσοληψία, με την οποία θα το έκανε και το σειριακό.

Σειριοποιησιμότητα όψεων

Έστω ότι S και S΄ είναι δυο χρονοπρογράμματα με το ίδιο set δοσοληψιών. Τα S και S΄ λέγονται **ισοδύναμα όψης** αν συμπίπτουν οι παρακάτω τρεις συνθήκες:

- Για κάθε δεδομένο αντικειμένου Q, αν η δοσοληψία T_1 διαβάζει την αρχική τιμή του Q στο χρονοπρόγραμμα S, τότε η δοσοληψία Ti πρέπει επίσης να διαβάζει την αρχική τιμή του Q στο χρονοπρόγραμμα S΄.
- Για κάθε δεδομένο αντικειμένου Q, αν η δοσοληψία T_1 εκτελέσει την **read**(*Q)* στο χρονοπρόγραμμα S και αυτή η τιμή παράχθηκε από τη δοσοληψία T_1, τότε πρέπει στο χρονοπρόγραμμα S΄ η δοσοληψία T_1 επίσης να διαβάζει την τιμή του Q που επίσης παράχθηκε από τη δοσοληψία T_1.
- Για κάθε δεδομένο αντικειμένου Q, η δοσοληψία που εκτελεί την τελική λειτουργία **write**(Q) στο χρονοπρόγραμμα *S* πρέπει να εκτελεί και την τελική **write**(*Q*) λειτουργία στο χρονοπρόγραμμα *S* ‘.

Ένα χρονοπρόγραμμα S είναι **σειριοποιήσιμο όψης** όταν είναι ισοδύναμο όψης με ένα σειριακό χρονοπρόγραμμα. Κάθε χρονοπρόγραμμα που είναι σειριοποιήσιμο σε σχέση με συγκρούσεις, είναι σειριοποιήσιμο όψεως . Το αντίστροφο όμως δεν ισχύει.

Το παρακάτω χρονοπρόγραμμα είναι σειριοποιήσιμο όψης αλλά όχι σειριοποιήσιμο σύγκρουσης (ισοδύναμο όψης σε σειριακό χρονοπρόγραμμα T3, T4, T6)

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Κάθε χρονοπρόγραμμα που είναι σειριοποιήσιμο όψεως, και ΔΕΝ είναι σειριοποιήσιμο σε σχέση με συγκρούσεις, περιέχει **τυφλές εγγραφές** (writes που δεν έχει προηγηθεί read γι’ αυτές στην δοσοληψία τους).

Παραδείγματα με MySQL

Αρχικά δημιουργήστε ένα πίνακα στη μηχανή αποθήκευσης InnoDB με όνομα T1 και γνωρίσματα L1, K1 που δείχνουν το pin λογαριασμού και το υπόλοιπο του κάθε λογαριασμού αντιστοίχως.

Απάντηση:

```
CREATE TABLE T1 (
        L1 INT NOT NULL,
        K1 INT,
PRIMARY KEY (L1)
)ENGINE=InnoDB;
```

και ας εισάγουμε σ΄ αυτόν τις παρακάτω εγγραφές :

```
INSERT INTO T1 VALUES(006, 600);
INSERT INTO T1 VALUES(007, 900);
INSERT INTO T1 VALUES(008, 1000);
INSERT INTO T1 VALUES(009, 1100);
```

Δίνοντας τώρα την εντολή:

```
SELECT * FROM T1;
```

βλέπουμε για το table T1 τον παρακάτω πίνακα που δείχνει πως έχουν μέχρι τώρα οι εγγραφές στο T1:

L1	K1
006	600
007	900
008	1000
009	1100

Το πρότυπο ISO καθορίζει ότι μια δοσοληψία της SQL ξεκινάει αυτόματα με μια δήλωση **εκκίνησης-δοσοληψίας** η οποία εκτελείται αυτόματα από ένα χρήστη ή ένα πρόγραμμα (πχ. από τις δηλώσεις **SELECT, INSERT, UPDATE**). Επίσης οι αλλαγές που γίνονται από μια δοσοληψία δεν είναι ορατές από άλλες δοσοληψίες που εκτελούνται ταυτόχρονα μέχρι την ολοκλήρωσή της.

Ας πειραματιστούμε τώρα με μια μόνο transaction:

Παράδειγμα

Ο κάτοχος του λογαριασμού με pin 009 έκανε κατάθεση και πλέον το υπόλοιπο του λογαριασμό του αντιστοιχεί σε 10000 ευρώ. Υλοποιήστε την ενημέρωση

Απάντηση:

```
START TRANSACTION;
UPDATE T1
SET K1=10000
WHERE K1=1100;
```

Με την παραπάνω πράξη ενημέρωσης της T1 αντικαθιστούμε την τιμή 1100 της k1 με την τιμή 10000. Αυτό φαίνεται από τον παρακάτω πίνακα που προκύπτει αν δώσουμε πάλι μια select μέσα στην transaction.

SELECT * **FROM** T1;

L1	K1
006	600
007	900
008	1000
009	10000

Όμως μια δοσοληψία μπορεί να ολοκληρωθεί με έναν από τους ακόλουθους τρόπους :

- Με μια δήλωση **COMMIT** που ολοκληρώνει επιτυχώς τη δοσοληψία, μονιμοποιώντας τις αλλαγές στη βάση δεδομένων. Μια νέα δοσοληψία ξεκινάει μετά τη δήλωση COMMIT με την επόμενη δήλωση εκκίνησης –δοσοληψίας
- Με μια δήλωση **ROLLBACK** η οποία ακυρώνει τη δοσοληψία, διαγράφοντας κάθε αλλαγή που έχει γίνει από τη δοσοληψία. Μια νέα δοσοληψία ξεκινάει μετά τη δήλωση ROLLBACK με την επόμενη δήλωση εκκίνησης –δοσοληψίας
- Για την προγραμματιζόμενη SQL, η επιτυχής ολοκλήρωση ενός προγράμματος ολοκληρώνει την τελική δοσοληψία με επιτυχία, ακόμη και αν η δήλωση COMMIT δεν έχει εκτελεστεί.
- Για την προγραμματιζόμενη SQL, η μη ομαλή ολοκλήρωση του προγράμματος ακυρώνει τη δοσοληψία.

Αν δηλαδή στη συνέχεια δώσουμε την εντολή ROLLBACK η εκτέλεση της δοσοληψίας θα ακυρωθεί και οι αλλαγές που έχουν γίνει απ’ αυτήν δεν θα αποθηκευτούν στη βάση. Ενώ αν δώσουμε COMMIT θα δούμε ότι οι αλλαγές από τη δοσοληψία θα μονιμοποιηθούν στη βάση δεδομένων και η T1 θα κλείσει επιτυχώς.

Παράδειγμα

Ο κάτοχος του λογαριασμού με pin κάρτας 007 βρίσκεται σε ένα τραπεζικό κατάστημα και προσπαθεί να κάνει κατάθεση μεταβάλλοντας το υπόλοιπο σε 10500ευρώ ενώ την ίδια στιγμή ο γιος του προσπαθεί να κάνει ανάληψη από ένα ATM και να μεταβάλλει τον ίδιο λογαριασμό από 900 σε 400 ευρώ. Υλοποιήστε την παραπάνω συναλλαγή

Απάντηση:

Ξεκινάμε την πρώτη δοσοληψία

START TRANSACTION;

UPDATE T1

SET K1=10500

WHERE K1=900;

Και συνεχίζουμε με την δεύτερη

START TRANSACTION;

(Αν εκτελέσουμε ως εδώ ένα Select * from T1; βλέπουμε ότι οι αλλαγές που έκανε η πρώτη transaction δεν είναι ορατές στη δεύτερη).

UPDATE T1

SET K1=10800

WHERE K1=900;

Ας δούμε τώρα πάλι με ένα Select * from T1; αν η δεύτερη δοσοληψία πραγματοποιήθηκε.

Το αποτέλεσμα που πήραμε φαίνεται στον παρακάτω στον πίνακα όπου διαπιστώνουμε ότι η **δεύτερη** δοσοληψία **δεν** έχει πραγματοποιηθεί!

L1	K1
006	600
007	10500
008	1000
009	10000

Αυτό οφείλεται σε μια τεχνική ελέγχου του ταυτοχρονισμού που ονομάζεται **κλείδωμα.** Η βασική ιδέα είναι απλή : όταν μια συναλλαγή χρειάζεται διαβεβαίωση ότι κάποιο αντικείμενο για το οποίο ενδιαφέρεται δε θα μεταβληθεί με κάποιον απρόβλεπτο τρόπο ενώ "η συναλλαγή έχει γυρισμένη την πλάτη της", τότε η συναλλαγή **αποκτά ένα κλείδωμα** σε αυτό το αντικείμενο. Το αποτέλεσμα του κλειδώματος είναι να αποκλειστούν οι άλλες συναλλαγές από το αντικείμενο, και έτσι να μην μπορούν να το μεταβάλουν. Η πρώτη συναλλαγή έχει έτσι τη δυνατότητα να πραγματοποιήσει την επεξεργασία της, γνωρίζοντας με βεβαιότητα ότι το δεδομένο αντικείμενο θα παραμείνει σε σταθερή κατάσταση για όσο χρόνο επιθυμεί η συναλλαγή.

Για να εκτελεστεί η δεύτερη transaction πρέπει να κλείσει πρώτα η α συναλλαγή με μια δήλωση COMMIT ή ROLLBACK. Τότε θα δούμε ότι η δεύτερη ξεμπλοκάρει και αρχίζει να εκτελείται και η ίδια. Η βάση δεδομένων την κάνει αυτόματα ROLLBACK(όπως αναφέρθηκε παραπάνω).

Γνωρίζοντας όμως τα παραπάνω προσπαθήστε να υλοποιήσετε και την παρακάτω δοσοληψία:

Παράδειγμα

Δυο άτομα την ίδια χρονική στιγμή προσπαθούν να κάνουν μια κατάθεση στο λογαριασμό με αριθμό pin 006 ο ένας μετατρέποντάς τον σε 10.700 ευρώ και ο άλλος στον λογαριασμό 007 μετατρέποντάς τον σε 11.500 ευρώ.

Απάντηση:

Εισάγουμε την πρώτη δοσοληψία
START TRANSACTION;
UPDATE T1
SET K1=10700
WHERE K1=600;
Εκτελούμε μια **COMMIT;** για να κλείσει η TR1 και να μονιμοποιηθούν οι αλλαγές στη βάση δεδομένων.
Κάνουμε εδώ μια
SELECT * **FROM** T1;
για να δούμε την κατάσταση που βρίσκεται το table T1

L1	K1
006	10700
007	10500
008	1000
009	10000

Ξεκινάμε τώρα την TR2
START TRANSACTION;
UPDATE T1
SET K1=11500
WHERE K1=10500;

Κάνοντας εδώ μια SELECT βλέπουμε όπως φαίνεται από τον πίνακα ότι πλέον η ενημέρωση στη βάση από την TR2 έχει γίνει:

L1	K1
006	11500
007	10500
008	1000
009	10000

Για να κλείσουμε τη δοσοληψία εισάγουμε μια δήλωση COMMIT ή ROLLBACK ανάλογα με το αν θέλουμε να διατηρήσουμε τις αλλαγές από την τρέχουσα δοσοληψία ή όχι. Ας προσέξουμε ότι οι παραπάνω δοσοληψίες υλοποιούνται στο K1 που είναι γνώρισμα μη πρωτεύοντος κλειδιού. Προσπαθήστε τώρα να πραγματοποιήσετε ενημέρωση σε transaction σε γνώρισμα πρωτεύοντος κλειδιού.

Παράδειγμα

Οι κάτοχοι των καρτών με pin 006 και 007 έχασαν την κάρτα τους και μόλις το αντιλήφθησαν έτρεξαν να αλλάξουν κωδικό. Ο ένας το μετατρέπει σε 001 ενώ εκείνος που είχε το 007 το μετατρέπει σε 002.

Απάντηση :

START TRANSACTION;
UPDATE T1
SET L1=001
WHERE L1=006;
Εκτελώντας τη δήλωση SELECT * FROM T1; βεβαιωνόμαστε ότι οι αλλαγές έχουν γίνει

L1	K1
001	11500
007	10500
008	1000
009	10000

Προχωράμε στην εκτέλεση της δεύτερης δοσοληψίας

START TRANSACTION;
UPDATE T1
SET L1=002
WHERE L1=007;
Ας εκτελέσουμε μια SELECT * FROM T1;

Το αποτέλεσμα που προκύπτει είναι ο παρακάτω πίνακας 6.7 όπου και διαπιστώνουμε ότι οι αλλαγές από την TR2 έχουν γίνει

L1	K1
001	11500
002	10500
008	1000
009	10000

Οι δυο δοσοληψίες εκτελούνται ταυτόχρονα όταν τα γνωρίσματα που ενημερώνουμε αποτελούν πρωτεύοντα κλειδιά. Εδώ **αποκτούν κλείδωμα μόνο οι συγκεκριμένες εγγραφές** ενώ στην περίπτωση που δεν δρούμε σε πρωτεύον κλειδί αποκτά κλείδωμα όλος ο πίνακας.

Πρόβλημα αδιεξόδου (deadlock)

Παράδειγμα

Δημιουργήστε έναν πίνακα ο οποίος θα περιέχει δυο γνωρίσματα, το ένα από τα δυο θα αντιπροσωπεύει τον κωδικό πτήσης μιας συγκεκριμένης αεροπορικής εταιρίας και το άλλο την ώρα αναχώρησης της συγκεκριμένης πτήσης.

Απάντηση :

```
CREATE TABLE F1 (
        Id INT ,
        hour  TIME,
PRIMARY KEY (Id));
```

Ας εισάγουμε σ' αυτόν μερικές τιμές

```
INSERT INTO T1 VALUES(533, 645);

INSERT INTO T1 VALUES(740, 2005);

INSERT INTO T1 VALUES(927, 615);

INSERT INTO T1 VALUES(348, 1700);
```

Παράδειγμα

Λόγω άσχημων καιρικών συνθηκών έχουμε ακύρωση των πτήσεων με id=740, id=348. Οι παραπάνω πτήσεις αντικαθίστανται η μια από την 632 με ώρα αναχώρησης 21:45 και η άλλη από την πτήση 283 και ώρα αναχώρησης 19:00 αντιστοίχως. Προσπαθήστε να υλοποιήσετε τις παραπάνω συναλλαγές.

Απάντηση:

Ξεκινάμε με τη δημιουργία της πρώτης συναλλαγής

```
START TRANSACTION;
UPDATE F1
SET Id=632
WHERE Id=740;
UPDATE F1
SET hour=2145
WHERE hour=2005;
```

Εκτελούμε και την δεύτερη

```
START TRANSACTION;
UPDATE F1
SET hour=1900
WHERE hour=1700;
```

```
UPDATE F1
SET  Id=283
WHERE Id=348;
```

Σ' αυτή την περίπτωση πέφτουμε σε ένα αδιέξοδο γιατί κάθε μια από τις δοσοληψίες περιμένει την άλλη να απελευθερώσει ένα κλείδωμα για να μπορέσει να προχωρήσει.

Το αδιέξοδο (deadlock) είναι μια κατάσταση όπου δυο ή περισσότερες συναλλαγές είναι **ταυτόχρονα** σε κατάσταση αναμονής, και η κάθε μια περιμένει μια από τις άλλες να απελευθερώσει ένα κλείδωμα, πριν μπορέσει να προχωρήσει. Αν συμβεί ένα αδιέξοδο είναι επιθυμητό το σύστημα να το εντοπίσει και να το σπάσει. Ο εντοπισμός του αδιεξόδου σημαίνει τον εντοπισμό ενός κύκλου στο **γράφημα αναμονών,** δηλαδή στο γράφημα που δείχνει "ποιος περιμένει ποιον". Σπάσιμο του αδιεξόδου σημαίνει να διαλέξουμε για **θύμα** μια από τις συναλλαγές που βρίσκονται σε αδιέξοδο και να την ανακατασκευάσουμε απελευθερώνοντας έτσι τα κλειδώματά της και επιτρέποντας να προχωρήσει σε κάποια άλλη συναλλαγή. Έτσι εδώ η MySQL εντοπίζει τα deadlock και από μόνη της κάνει rollback τις δυο δοσοληψίες.

Επίπεδο απομόνωσης

Ο όρος **επίπεδο απομόνωσης** χρησιμοποιείται για να υποδηλώσει αυτό που λέμε βαθμό παρεμβολής τον οποίο μπορεί να ανεχθεί μια δεδομένη συναλλαγή όσον αφορά τις ταυτόχρονες συναλλαγές. Στην πραγματικότητα, δεν υπάρχει κανένας λόγος να μη λειτουργεί μια συναλλαγή σε διαφορετικά επίπεδα απομόνωσης την ίδια στιγμή, σε διαφορετικά μέρη της βάσης δεδομένων.

Μια ειδική εντολή που ονομάζεται **SET TRANSACTION** χρησιμοποιείται για να ορίσει κάποια χαρακτηριστικά της νέας συναλλαγής που θα ξεκινήσει. Χαρακτηριστικά αυτής της εντολής είναι ο **τρόπος προσπέλασης** και το ***επίπεδο απομόνωσης.*** Η σύνταξη της εντολής είναι:

SET [GLOBAL | SESSION] **TRANSACTION** [READ ONLY | READ WRITE] ISOLATION LEVEL

{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }

Οι προτάσεις READ ONLY και READ WRITE υποδεικνύουν αν μια συναλλαγή είναι μόνο για ανάγνωση ή αν περιλαμβάνει και τις δυο πράξεις. Η προκαθορισμένη επιλογή είναι η READ WRITE αν δεν καθορίζεται καμία. Η δήλωση READ ONLY επιτρέπει σε μια δοσοληψία να εκτελέσει τις δηλώσεις INSERT, UPDATE και DELETE σε προσωρινούς και μόνο πίνακες.

Το επίπεδο απομόνωσης υποδεικνύει το βαθμό αλληλεπίδρασης που επιτρέπεται από άλλες δοσοληψίες κατά την εκτέλεση της συγκεκριμένης δοσοληψίας. Τα πιθανά επίπεδα απομόνωσης είναι τα:

1. **READ UNCOMMITTED** (ανάγνωση ανεπικύρωτων): επιτρέπει σε μια δοσοληψία να βλέπει ενδιάμεσα αποτελέσματα άλλων δοσοληψιών, πριν αυτές ολοκληρωθούν. Εδώ εισάγεται το πρόβλημα της **αναξιόπιστης ανάγνωσης** δηλ μια δοσοληψία μπορεί να διαβάσει μια γραμμή δοσοληψίας που μόλις έχει ενημερωθεί ενώ μετά αυτή κάνει ROLLBACK
2. **READ COMMITTED** (ανάγνωση επικυρωμένων): επιτρέπει σε μια δοσοληψία να βλέπει ενδιάμεσα αποτελέσματα άλλων δοσοληψιών όταν αυτά γίνουν COMMITED.Πρόβλημα **μη επαναλήψιμης ανάγνωσης**. Δηλαδή εδώ μια δοσοληψία διαβάζει αρχικά μια τιμή, μετά αυτή ενημερώνεται από μια commited transaction και όταν μετά πάει να ξαναδιαβάσει την τιμή έχει αλλάξει.
3. **REPEATABLE READ (επαναλήψιμη ανάγνωση) ή SERIALIZABLE** (σειριοποιήσιμο) :επιτρέπουν σε μια δοσοληψία να κάνει σταθερές αναγνώσεις. Όταν διαβάσει κάτι εξακολουθεί να το βλέπει ακόμα και αν μια άλλη δοσοληψία το έχει αλλάξει.(phantoms)
4. Η δήλωση GLOBAL σημαίνει ότι αλλάζεις το επίπεδο απομόνωσης καθολικά, για όλες τις επόμενες συνδέσεις ενώ με τη SESSION μόνο για τις επόμενες συνδέσεις.

Παράδειγμα

Τώρα σύμφωνα με τον παραπάνω πίνακα T1 που αφορά τραπεζικούς λογαριασμούς υλοποιήστε μια δοσοληψία που να προσθέτει 200ευρώ στον λογαριασμό με pin=008 και στη συνέχεια μια δεύτερη η οποία διαβάζει το συνολικό ποσό του λογαριασμού 008 και προσθέτει το 10% αυτού στο λογαριασμό με pin=009

Απάντηση:

Ξεκινάμε την πρώτη δοσοληψία
START TRANSACTION;
΄Ομως εδώ η δεύτερη δοσοληψία θέλουμε να κάνει σταθερές αναγνώσεις γιατί από αυτήν διαβάζει το συνολικό ποσό και το μεταφέρει στον άλλο λογαριασμό. Άρα εδώ πρέπει να ορίσουμε σ'αυτήν μια set transaction με προεπιλογή SERIALIZABLE:
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
Εκτελούμε την πράξη ενημέρωσης
UPDATE T1
SET K1=K1+200 **WHERE** K1=1000;
Με **SELECT** * **FROM** T1;
Βλέπουμε από τον παρακάτω πίνακα ότι η ενημέρωση πραγματοποιήθηκε.

L1	K1
001	11500
002	10500
008	1200
009	10000

Και προχωράμε στη δεύτερη:
START TRANSACTION;
UPDATE T1
SET K1=0.1 * K1 + 10000 WHERE K1=10000;(πρόσθεση 10% από Α)
UPDATE T1
SET K1=K1-0.1*K1 WHERE K1=1200; (αφαιρούμε το 0.1*K1)

Το αποτέλεσμα φαίνεται από τον πίνακα που ακολουθεί

L1	K1
001	11500
002	10500
008	1080
009	11080

Διαπιστώνουμε ότι η ενημέρωση έγινε παίρνοντας το σωστό ποσό από την TR1.

Κλείδωμα πίνακα

Τα locks χρησιμοποιούνται είτε για να γίνει ένα είδος συναγωνισμού με τα transactions είτε για να επιταχύνουμε όταν γίνεται ενημέρωση της βάσης.

Η εντολή με την οποία κλειδώνεις και ξεκλειδώνεις ένα πίνακα είναι η εξής

LOCK TABLES

tbl_name [AS *alias*] {READ [LOCAL] | [LOW_PRIORITY] WRITE}

[, *tbl_name* [AS *alias*] {READ [LOCAL] | [LOW_PRIORITY] WRITE}]

UNLOCK TABLES

Μπορούμε να έχουμε είτε **read** είτε **write locks.** Όταν κλειδώνουμε έναν πίνακα, πρέπει να καθορίσουμε αν θέλουμε ένα "read lock" ή ένα "write lock". Το πρώτο εμποδίζει άλλες διεργασίες από το να κάνουν αλλαγές στον πίνακα αλλά επιτρέπει σ' άλλες να διαβάσουν τον πίνακα. Ενώ όταν μια ακολουθία επιτρέψει ένα write lock στον πίνακα μόνο εκείνη μπορεί να διαβάσει απ' αυτόν τον πίνακα και να γράψει σ' αυτόν. Όλες οι άλλες διεργασίες μπλοκάρονται.

Παράδειγμα

Έστω ότι έχουμε δυο clients από τους οποίους ο πρώτος παίρνει ένα read lock στον πίνακα p1 που δείχνει τη βαθμολογία σε κάθε μάθημα. Έπειτα ο δεύτερος προσπαθεί να εισάγει μια εγγραφή στο ίδιο table. Υλοποιήστε το. Τί συμβαίνει;

Απάντηση:

Ξεκινάω με τη δημιουργία πίνακα σε MyISAM :

CREATE TABLE P1 (

ID INT ,

DEG INT,

PRIMARY KEY (ID)

)ENGINE=MyISAM;

Βάζω τιμές στον πίνακα

INSERT INTO P1 VALUES(160, 08);

INSERT INTO P1 VALUES(170, 04);

INSERT INTO P1 VALUES(190, 02);

INSERT INTO P1 VALUES(110, 06);

1os client

Κάνω το read lock

Lock table p1 read;

2os client

INSERT INTO P1 VALUES(185, 10);

Παραπάνω ο 2os client προσπαθεί να ενημερώσει τον πίνακα ρ1 που έχει πάρει read lock όμως παρατηρούμε ότι δεν γίνεται. Για να συμβεί κάτι τέτοιο πρέπει πρώτα ο 1os client να κάνει unlock το table.

Επίσης, εδώ παρατηρούμε ότι ο δεύτερος client μπορεί να κάνει ένα

SELECT * FROM p1;

Δηλαδή, όταν ένας πίνακας πάρει ένα read lock σε μια ακολουθία μπορεί η ίδια και οποιαδήποτε άλλη ακολουθία να διαβάσει αυτόν τον πίνακα αλλά δεν μπορεί να κάνει αλλαγές σ' αυτόν.

Παράδειγμα

Έστω ότι έχουμε δυο clients από τους οποίους ο πρώτος παίρνει ένα write lock στον πίνακα p1 που δείχνει τη βαθμολογία σε κάθε μάθημα. Έπειτα ο δεύτερος προσπαθεί να ενημερώσει το ίδιο table. Υλοποιήστε το. Τι συμβαίνει;

Απάντηση:

1os client

Κάνω το write lock

Lock table p1 write;

2os client

UPDATE P1 SET DEG=05 WHERE DEG=04;

Ο δεύτερος client προσπαθεί να ενημερώσει τον ρ1 αλλά επειδή είναι κλειδωμένος με write lock δεν γίνεται!

Επίσης, αν προσπαθήσει ο δεύτερος να διαβάσει από αυτό το table πάλι δεν μπορεί.

Αν όμως ο πρώτος client κάνει ένα

SELECT * FROM p1;

ή ένα

UPDATE p1 SET deg=05 WHERE deg=04;

Παρατηρούμε ότι γίνεται!

Άρα όταν μια ακολουθία επιτρέψει ένα write lock σε έναν πίνακα μόνο αυτή μπορεί να ενημερώσει ή να διαβάσει αυτόν τον πίνακα. Οποιαδήποτε άλλη ακολουθία μπλοκάρεται. Πρέπει να κάνει πρώτα unlock τον πίνακα.

Έτσι αν μια διεργασία επιτρέψει σε έναν πίνακα να πάρει ένα read lock τότε μπορεί και οποιαδήποτε άλλη να πάρει lock σ' αυτόν τον πίνακα αλλά μόνο read. Ενώ αν ένας πίνακας πάρει write lock σε μια διεργασία τότε καμία άλλη διεργασία δεν μπορεί να πάρει οποιοδήποτε lock σε αυτόν τον πίνακα.

Επίσης μπορούμε να έχουμε **read locαl** και **write locαl** παραμέτρους. Η **read local** επιτρέπει στις μη συγκρουόμενες INSERT δηλώσεις να εκτελούνται ενώ κρατείται το lock.

Τα **LOW_PRIORITY WRITE** locks χρησιμοποιούνται για να επιτρέψουν σε άλλες διεργασίες να αποκτήσουν read locks ενώ οι διεργασίες περιμένουν για το write lock. Τα **LOW_PRIORITY WRITE** locks πρέπει να χρησιμοποιούνται μόνο όταν υπάρχει σιγουριά

Κλείδωμα εγγραφών έναντι κλειδώματος πινάκων

Πλεονεκτήματα του row - locking επιπέδου (κλείδωμα σε εγγραφές):

- Λιγότερες lock συγκρούσεις καθώς προσπελαύνονται διαφορετικές γραμμές σε πολλές ακολουθίες
- Λιγότερες αλλαγές για rollbacks.
- Είναι δυνατό να κλειδώσει μια μόνο γραμμή για πολύ ώρα.

Παρ' όλα αυτά όμως το row - locking έχει και τα εξής **μειονεκτήματα:**

- Καταλαμβάνει περισσότερη μνήμη από τα table-locks επίπεδα

- Είναι πιο αργό από το επίπεδο table-lock όταν χρησιμοποιείται σε ένα μεγάλο μέρος του table γιατί πρέπει να αποκτήσεις πολύ περισσότερα locks
- Είναι πολύ χειρότερο από άλλα locks αν κάνεις συχνά GROUP BY λειτουργίες σε ένα μεγάλο μέρος δεδομένων ή αν πρέπει συχνά να εξετάζεις ολόκληρο το table.
- Με τα υψηλότερου επιπέδου locks μπορείς επίσης πιο εύκολα να υποστηρίξεις locks διαφορετικών τύπων για να συντονίσεις την εφαρμογή επειδή τα υψηλότερα locks είναι λιγότερα από τα row-level.

Όμως, για να επιτύχει μια πολύ υψηλή lock ταχύτητα , η MYSQL χρησιμοποιεί το table-locking , αντί για row-locking, για όλες τις μηχανές αποθήκευσης εκτός από την InnoDB και BDB.

Έτσι, τα table-locks είναι **ανώτερα** από τα row-locks στις ακόλουθες περιπτώσεις :

- Όταν οι περισσότερες δηλώσεις για τα tables είναι read
- Αναγνώσεις και ενημερώσεις σε strict κλειδιά,που ενημερώνεις ή διαγράφεις μια γραμμή που μπορεί να προκαλεσθεί με μια ανάγνωση μοναδικού κλειδιού

UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;

DELETE FROM tbl_name WHERE unique_key_col=key_value;

- Παράλληλες INSERT συνδυασμένες με SELECT δηλώσεις, και πολύ λίγες UPDATE και DELETE δηλώσεις
- Μερικές SCAN και ORDER BY λειτουργίες σε ολόκληρο το table χωρίς εγγραφές.

Για μεγάλα tables, το table locking είναι πολύ καλύτερο από το row για τις περισσότερες εφαρμογές

Δεν παύει όμως και το table locking να μειονεκτεί στις παρακάτω περιπτώσεις:

- Όταν ένας client εισάγει μια δήλωση SELECT που παίρνει πολύ χρόνο να τρέξει.
- Ένας δεύτερος client τότε εισάγει μια δήλωση UPDATE στο ίδιο table. Αυτός ο client θα περιμένει μέχρι η δήλωση SELECT να τελειώσει.
- Ένας άλλος client εισάγει μια άλλη δήλωση SELECT στο ίδιο table. Επειδή η UPDATE έχει μεγαλύτερη προτεραιότητα από τη SELECT, αυτή η SELECT θα περιμένει μέχρι η UPDATE να τελειώσει. Θα περιμένει επίσης μέχρι να ολοκληρωθεί και η πρώτη SELECT.

13.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 1. Χρησιμοποιείστε δοσοληψία για την ενημέρωση του βαθμού του φοιτητή «Kouris» στο μάθημα «Baseis Dedomenwn». Αρχικά ακυρώστε την δοσοληψία χρησιμοποιώντας την εντολή rollback και στην συνέχει υποβάλετε τον σωστό βαθμό ολοκληρώνοντας την συνεδρία με την εντολή commit. 2. Σε ένα υποθετικό σενάριο όπου δύο χρήστες τις γραμματείας θα πρέπει ταυτόχρονα να περάσουν βαθμούς μαθημάτων στην βάση δεδομένων και να υπολογίσουν και τον Μέσο Όρο κάποιων φοιτητών, χρησιμοποιήστε τα κατάλληλα επίπεδα απομόνωσης συνεδριών για την ορθή λειτουργία και την λήψη σωστών αποτελεσμάτων.
Ζητούμενα:	Χρήση των δοσοληψιών και κατανόηση των επιπέδων απομόνωσης (isolation levels).

Ζητούμενο 1: Χρησιμοποιείστε δοσοληψία για την ενημέρωση του βαθμού του φοιτητή «Kouris» στο μάθημα «Baseis Dedomenwn». Αρχικά ακυρώστε την δοσοληψία χρησιμοποιώντας την εντολή rollback και στην συνέχει υποβάλετε τον σωστό βαθμό ολοκληρώνοντας την συνεδρία με την εντολή commit.

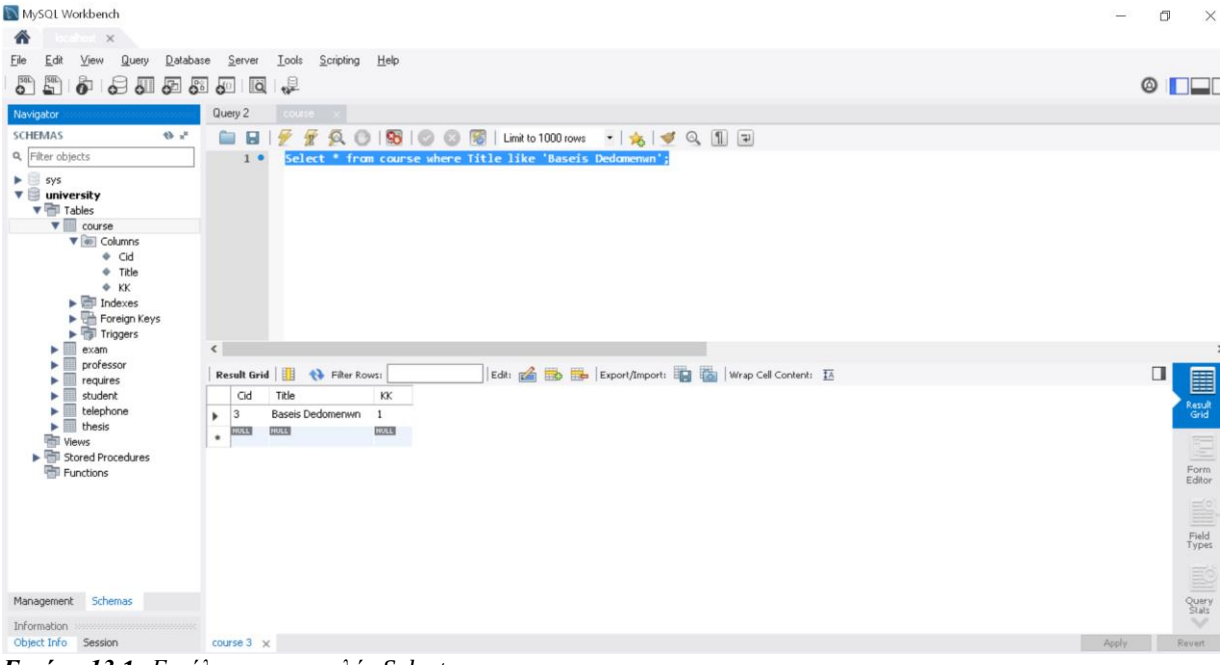
Αρχικά ξεκινάει η ενεργοποίηση της δοσοληψίας και ακύρωση της στην περίπτωση καταχώρησης λάθους βαθμού όπως φαίνεται στο ακόλουθο παράδειγμα:

Επιλέγουμε την Βάση δεδομένων:

```
Use university;
```

Αρχικά βρίσκουμε τον κωδικό του μαθήματος για το οποίο πρέπει να ενημερωθεί ο βαθμός. Ο κωδικός του επίμαχου μαθήματος είναι 3.

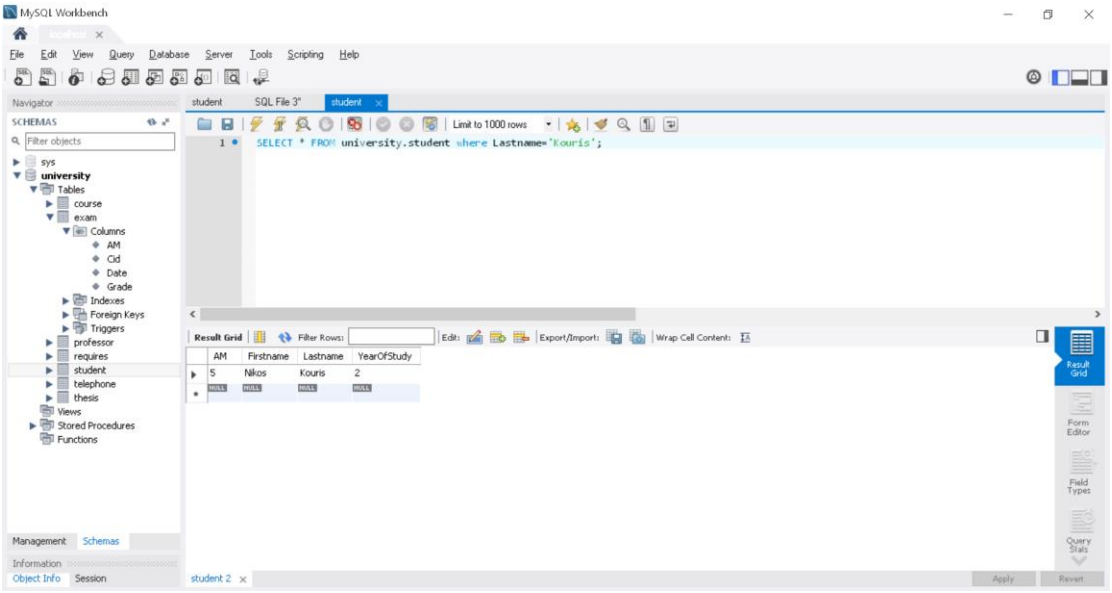
```
Select * from course where Title like 'Baseis Dedomenwn';
```



Εικόνα 13.1: *Εκτέλεση της εντολής Select..*

Στη συνέχεια βρίσκουμε το ΑΜ του μαθητή Kouris. Το ΑΜ του φοιτητή είναι 1.

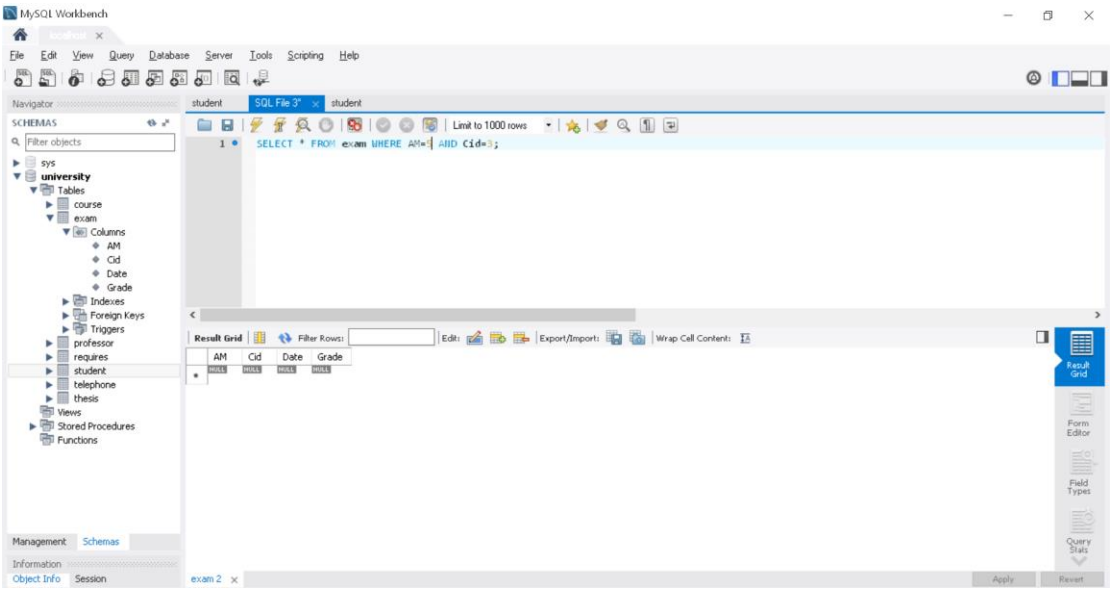
SELECT * FROM university.student where Lastname like Kouris;



Εικόνα 13.2: *Εκτέλεση της εντολής Select..*

Στην συνέχεια εξετάζουμε αν ο φοιτητής έχει ξαναδώσει κάποια εξέταση για το μάθημα.

SELECT * FROM exam WHERE AM=5 AND Cid=3;



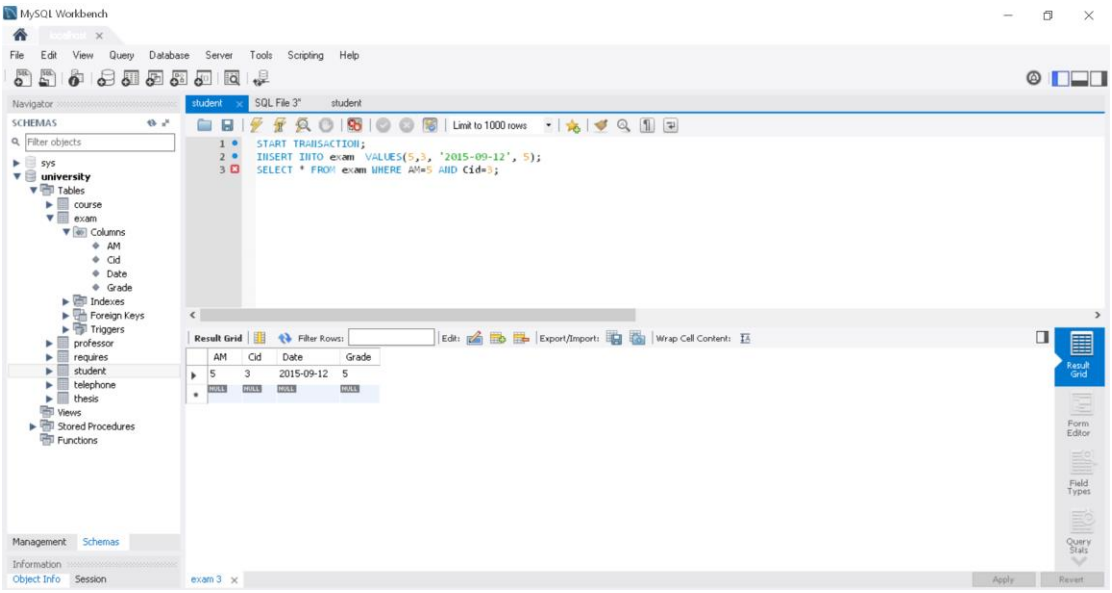
Εικόνα 13.3: *Εκτέλεση της εντολής Select.*

Ξεκινάμε την δοσοληψία που θα κάνει την ενημέρωση γράφουμε την εντολή εισαγωγής νέας εγγραφής και στη συνέχει με χρήση της select βλέπουμε αν η εισαγωγή υπάρχει στο σχήμα exam.

START TRANSACTION;

INSERT INTO exam VALUES(5,3, '2015-09-12', 5);

SELECT * FROM exam WHERE AM=5 AND Cid=3;

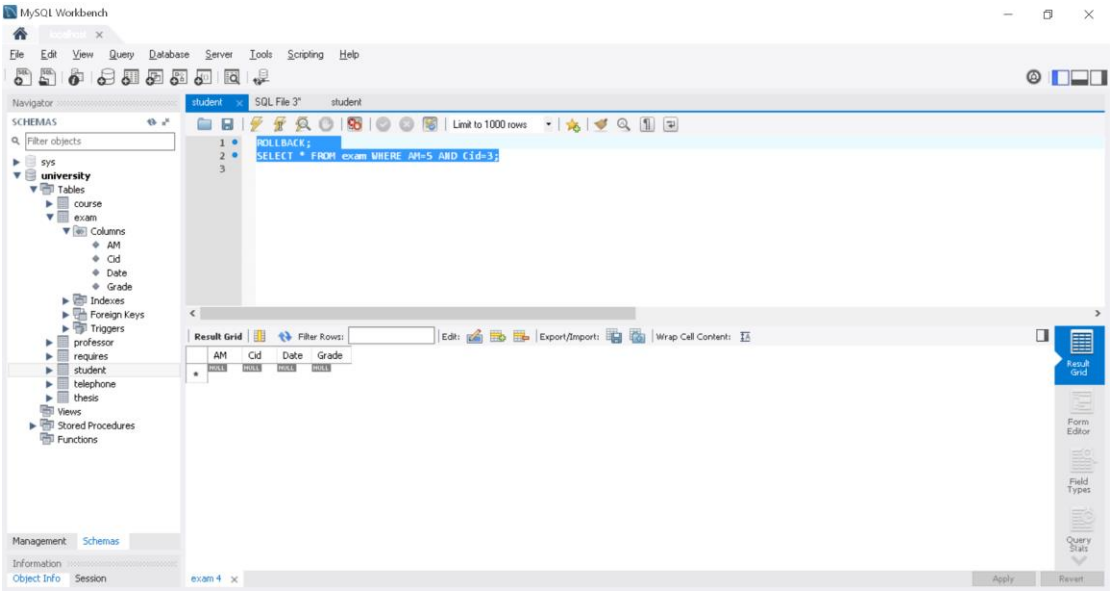


Εικόνα 13.4: *Εκτέλεση εντολών εντός Δοδοληψίας..*

Διαπιστώνουμε ότι ο βαθμός πέντε (5) είναι λάθος οπότε επιλέγουμε να τερματίσουμε την δοσοληψία χωρίς να εκτελεστούν οι αλλαγές στην βάση δεδομένων εκτελώντας την εντολή rollback.

ROLLBACK;

SELECT * FROM exam WHERE AM=5 AND Cid=3;



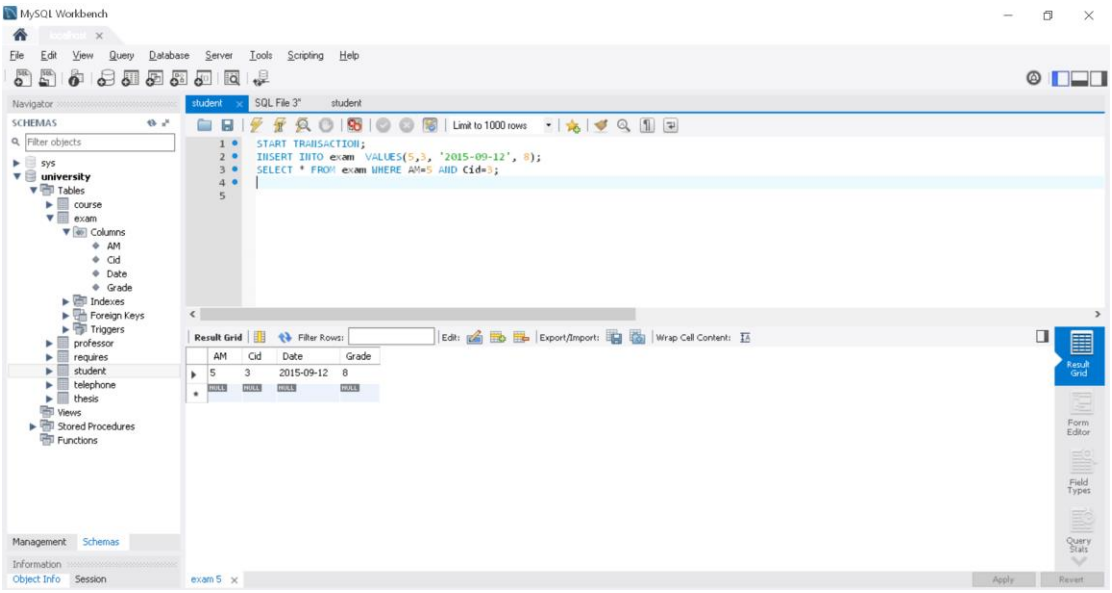
Εικόνα 13.5: *Αναίρεση Δοσοληψίας.*

Παρατηρούμε ότι η διαδικασία ολοκληρώθηκε χωρίς να περάσει η εγγραφή στην βάση δεδομένων. Συνεχίζουμε σε μία νέα συνεδρία όπου χρησιμοποιείται ο σωστός βαθμός 8 στην εξέταση του φοιτητή.

START TRANSACTION;

INSERT INTO exam VALUES (5, 3, '2015-09-12', 8);

SELECT * FROM exam WHERE AM=5 AND Cid=3;

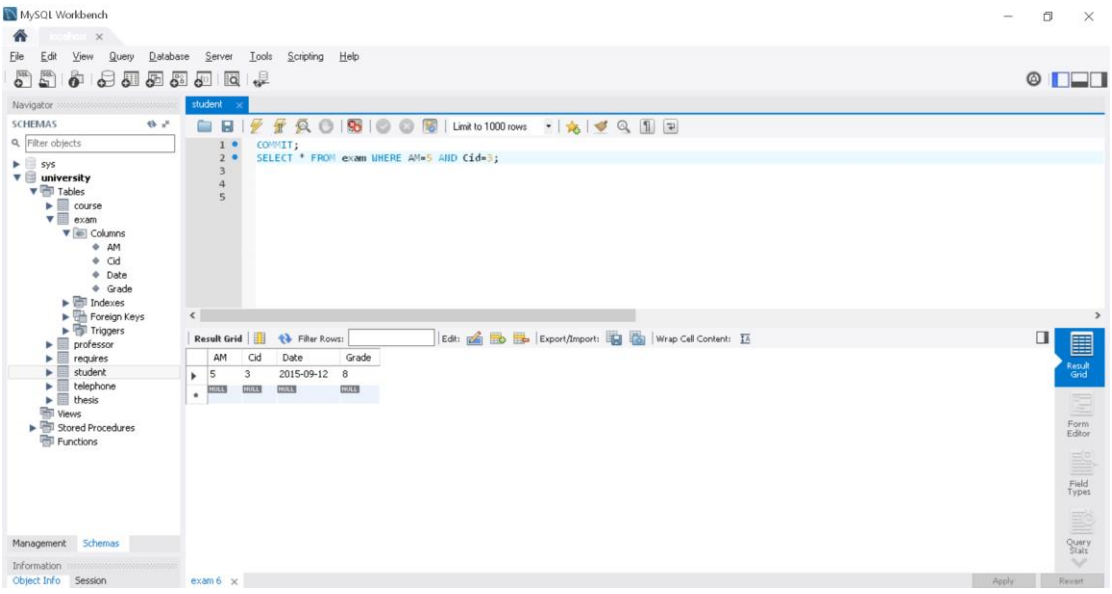


Εικόνα 13.6: *Εκτέλεση Δοσοληψίας..*

Στην συνέχεια ολοκληρώνουμε την συνεδρία με την εντολή commit και οι ενέργειες εκτελούνται τελικά στην βάση δεδομένων.

COMMIT;

SELECT * FROM exam WHERE AM=5 AND Cid=3;

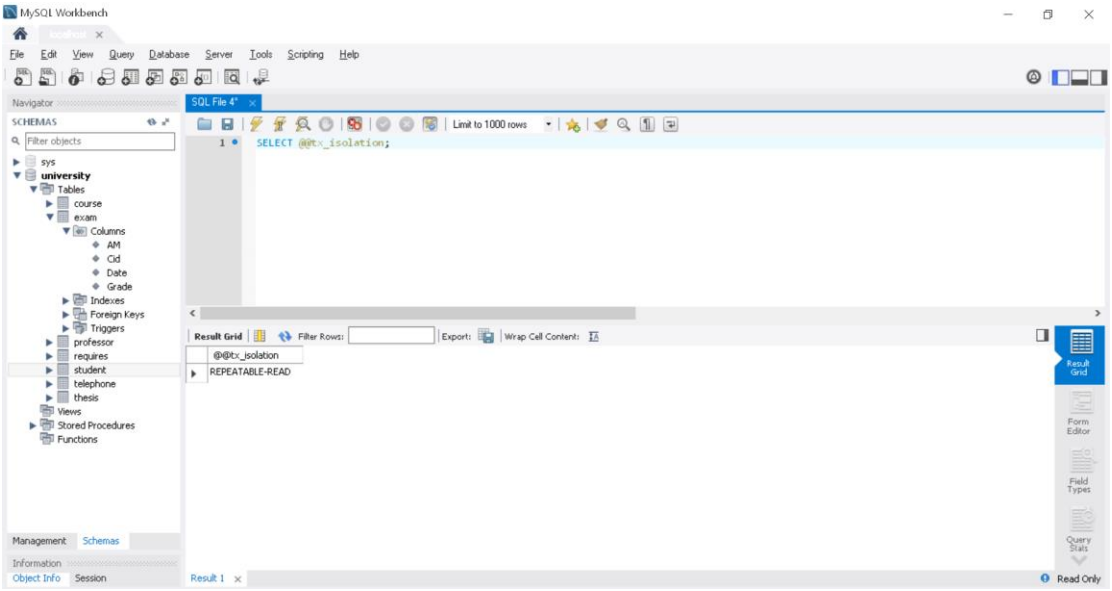


Εικόνα 13.7: *Επιτυχής ολοκλήρωση Δοσοληψίας*

Ζητούμενο 2: Σε ένα υποθετικό σενάριο όπου δύο χρήστες τις γραμματείας θα πρέπει ταυτόχρονα να περάσουν βαθμούς μαθημάτων στην βάση δεδομένων και να υπολογίσουν και τον Μέσο Όρο κάποιων φοιτητών, χρησιμοποιήστε τα κατάλληλα επίπεδα απομόνωσης συνεδριών για την ορθή λειτουργία και την λήψη σωστών αποτελεσμάτων.

Για να δούμε το επίπεδο απομόνωσης δίνουμε την εντολή

SELECT @@tx_isolation;



Εικόνα 13.8: *Επιλογή επιπέδου απομόνωσης.*

Για να ορίσουμε το επίπεδο απομόνωσης μίας δοσοληψίας χρησιμοποιούμε την εντολή:

SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{

REPEATABLE READ

| READ COMMITTED

| READ UNCOMMITTED

| SERIALIZABLE

}

Όπως για παράδειγμα:

SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

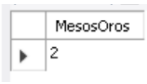
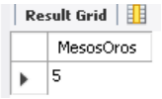
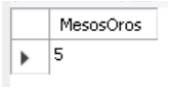
Έστω ότι υπάρχουν δύο χρήστες που θέλουν να εκτελέσουν τις ακόλουθες ενέργειες ταυτόχρονα:

- **Χρήστης Α:** Ενημερώνει τον βαθμό του φοιτητή Kouris στο μάθημα Baseis Dedomenwn χρησιμοποιώντας την εντολή: INSERT INTO exam VALUES(5,3, '2015-09-12', 8);

- **Χρήστης Β:** Να υπολογίσει τον μέσο όρο του φοιτητή Kouris χρησιμοποιώντας το ακόλουθο ερώτημα: SELECT avg(grade) as MesosOros FROM exam WHERE AM=5

ΣΕΝΑΡΙΟ 1---ISOLATION LEVEL READ UNCOMMITTED

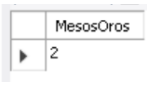
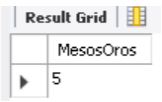
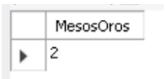
Επιτρέπει στον Χρήστη Β να δει αλλαγές από άλλους χρήστες που δεν έχουν γίνει commit.

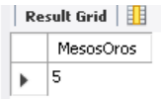
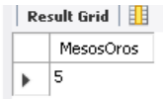
Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; SELECT @@tx_isolation; SET autocommit=0; SELECT @@autocommit; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	
	use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5  Παρατηρούμε ότι βλέπει την εισαγωγή της εγγραφής του Χρήστη Α πριν αυτός δώσει commit.	
Προσοχή καθώς δίνει ο Χρήστης Α rollback υπάρχει ο κίνδυνος ο Χρήστης Β να συνεχίσει να χρησιμοποιεί τον λανθασμένο Μέσο όρο.	Έστω ότι ο χρήστης δίνει την εντολή rollback για να αναιρέσει την επιλογή του. rollback; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 

Μόνο αν εκτελέσει ξανά το ερώτημα θα είναι σε θέση να διαπιστώσει ότι ο Χρήστης Α έκανε τελικά rollback στην ενέργεια του. SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	

ΣΕΝΑΡΙΟ 2---ISOLATION LEVEL READ COMMITTED

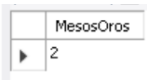
Επιτρέπει στον Χρήστη Β να δει μόνο αλλαγές από άλλους χρήστες που έχουν γίνει commit.

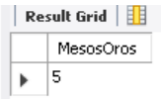
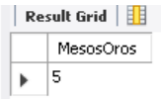
Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED; SELECT @@tx_isolation; SET autocommit=0; SELECT @@autocommit; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	
	use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5  Παρατηρούμε ότι δεν βλέπει την εισαγωγή της εγγραφής του Χρήστη Α πριν αυτός δώσει commit.	

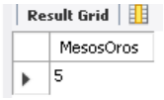
	Έστω ότι ο χρήστης δίνει την εντολή commit για να ενημερώσει την βάση δεδομένων. COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
Μόνο αν εκτελέσει ξανά το ερώτημα θα είναι σε θέση να διαπιστώσει ότι ο Χρήστης Α έκανε τελικά commit στην ενέργεια του και τα νέα δεδομένα. SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  **Φαινόμενο non-repeatable read το session του Χρήστη Β μέσα στο ίδιο transaction διαβάζει δύο διαφορετικές τιμές.**	

ΣΕΝΑΡΙΟ 3---ISOLATION LEVEL REPEATABLE READ

Επιτρέπει στον Χρήστη Β να δει μόνο αλλαγές από άλλους χρήστες που έχουν γίνει commit από τον χρήστη Α αλλά και ταυτόχρονα διατηρεί στην δοσοληψία του την ίδια τιμή. Δηλαδή θα δει τις αλλαγές του χρήστη Α μόλις ο ίδιος ο Χρήστης Β δώσει commit για να ολοκληρώσει την δική του δοσοληψία.

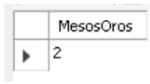
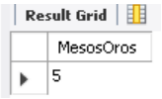
Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SET SESSION TRANSACTION ISOLATION LEVEL **REPEATABLE READ**; SELECT @@tx_isolation; SET autocommit=0; SELECT @@autocommit; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	

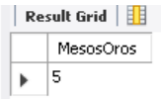
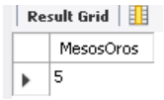
	use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5  Παρατηρούμε ότι δεν βλέπει την εισαγωγή της εγγραφής του Χρήστη Α πριν αυτός δώσει commit.	
	Έστω ότι ο χρήστης δίνει την εντολή commit για να ενημερώσει την βάση δεδομένων. COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  **Το φαινόμενο non-repeatable read δεν εμφανίζεται. Το session του Χρήστη Β μέσα στο ίδιο transaction εξακολουθεί να διαβάζει την ίδια τιμή. Μόνο αν εκτελέσει commit και ολοκληρώσει την δική του δοσοληψία θα είναι σε θέση να διαπιστώσει ότι ο Χρήστης Α έκανε τελικά commit στην ενέργεια του και τα νέα δεδομένα.**	

COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	

ΣΕΝΑΡΙΟ 4---ISOLATION LEVEL SERIALIZABLE

Επιτρέπει στον Χρήστη Β να δει μόνο αλλαγές από άλλους χρήστες που έχουν γίνει commit από τον χρήστη Α. Αν κατά την διάρκεια της ενημέρωσης από τον Α ο Β προσπαθήσει να εκτελέσει το ερώτημα του, τότε η δοσολοψία του θα «κλειδώσει» μέχρι να ολοκληρωθεί αυτή του Α.

Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	
	SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE; use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5 **Παρατηρούμε ότι εδώ η δοσοληψία «κλειδώνει» μέχρι να ολοκληρώσει την δοσοληψία του ο Α.**	

	Έστω ότι ο χρήστης δίνει την εντολή commit για να ενημερώσει την βάση δεδομένων. COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  Result Grid MesosOros 5
 Result Grid MesosOros 5 **Ολοκληρώνεται η εντολή του Χρήστη Β αφού ο Α ολοκλήρωσε την δική του δοσοληψία.**	

13.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Επεκτείνεται το ζητούμενο 2 της λυμένης εργαστηριακής άσκησης για τρεις χρήστες αυτή τη φορά/ Έστω ότι υπάρχουν τρεις χρήστες που θέλουν να εκτελέσουν τις ακόλουθες ενέργειες ταυτόχρονα:

- **Χρήστης Α:** Ενημερώνει τον βαθμό του φοιτητή Kouris στο μάθημα Baseis Dedomenwn χρησιμοποιώντας την εντολή: INSERT INTO exam VALUES(5,3, '2015-09-12', 8);
- **Χρήστης Β:** Να υπολογίσει τον μέσο όρο του φοιτητή Kouris χρησιμοποιώντας το ακόλουθο ερώτημα: SELECT avg(grade) as MesosOros FROM exam WHERE AM=5
- **Χρήστης C:** Ενημερώνει τον βαθμό του φοιτητή Kouris στο μάθημα Programmatismos IIχρησιμοποιώντας την εντολή: INSERT INTO exam VALUES(5,6, '2015-09-12', 4);

Δοκιμάστε και τα 4 επίπεδα απομόνωσης όπως και στην λυμένη άσκηση.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαιο 17.
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 16.
TRANSACTIONS http://dev.mysql.com/doc/refman/5.7/en/commit.html

Κεφάλαιο 14 Διασύνδεση με Εφαρμογές JAVA

Σύνοψη
Στο παρόν κεφάλαιο θα παρουσιασθούν βασικά στοιχεία όσον αφορά την σύνδεση ενός προγράμματος Java με την Mysql και την εκτέλεση εντολών σε αυτή.

Προαπαιτούμενη γνώση
Βασική γνώση συγγραφής προγραμμάτων σε Java καθώς και η ύλη των προηγούμενων κεφαλαίων του βιβλίου.

14.1 Εισαγωγικές Έννοιες

Το JAVA JDBC είναι ένα JAVA API με το οποίο μπορούμε να έχουμε πρόσβαση σε οποιαδήποτε μορφής δεδομένα πίνακα και ειδικότερα σε δεδομένα μιας σχεσιακής Βάσης Δεδομένων.

Με το JDBC μπορούμε να γράψουμε Java εφαρμογές, οι οποίες διαχειρίζονται τις παρακάτω λειτουργίες:

1. Σύνδεση στη Βάση Δεδομένων
2. Αποστολή ερωτημάτων και ενημέρωση δεδομένων στη Βάση
3. Ανάκτηση και επεξεργασία των αποτελεσμάτων που λαμβάνονται από τη βάση ως απάντηση στο ερώτημά μας.

Στο ακόλουθο παράδειγμα φαίνονται αυτές οι λειτουργίες:

```
public void connectToAndQueryDatabase(String username, String password) {
   Connection con = DriverManager.getConnection(
                    "jdbc:myDriver:myDatabase",
                    username,
                    password); // Εντολή σύνδεσης στο ΣΔΒΔ
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
   while (rs.next()) { // Επανάληψη στις εγγραφές που έχουν επιστραφεί
      int x = rs.getInt("a");          // Επιλογή του χαρακτηριστικού a σε μία
                                       //μεταβλητή τύπου int.
      String s = rs.getString("b");
      float f = rs.getFloat("c");
   }
}
```

Στο παραπάνω παράδειγμα χρησιμοποιείται το αντικείμενο DriveManager το οποίο κάνει τη σύνδεση με τη Βάση, το αντικείμενο Statement το οποίο μεταφέρει το SQL ερώτημα στη Βάση και το ResultSet αντικείμενο το οποίο ανακτά τα αποτελέσματα από τα ερωτήματά μας και εκτελεί ένα απλό while loop για την ανάκτηση και εμφάνιση αυτών των αποτελεσμάτων.

Το JDBC API χρησιμοποιεί JDBC drivers για να συνδεθεί με τη βάση. Υπάρχουν αρκετές βιβλιοθήκες που υλοποιούν το API. Συνήθως κάθε σύστημα διαχείρισης βάσεων δεδομένων (MySQL, Oracle, SQL Server) δίνει και μία υλοποίηση αυτού. Συνεπώς με την εγκατάσταση του εξυπηρετητή πρέπει να επιλεγεί και κατάλληλα η JDBC βιβλιοθήκη.

Τι είναι το API (Application Programming Interface)

Είναι ένα έγγραφο τεκμηρίωσης που περιέχει την περιγραφή όλων των χαρακτηριστικών ενός προϊόντος ή λογισμικού ή βιβλιοθήκης. Αναπαριστά κλάσεις και διεπαφές που ακολουθούν τα προγράμματα λογισμικού για να επικοινωνούν μεταξύ τους. Μπορούμε να δημιουργήσουμε ένα API για εφαρμογές, βιβλιοθήκες, λειτουργικά συστήματα, κ.α.

JDBC Drivers

Ένας JDBC driver είναι ένα κομμάτι λογισμικού που επιτρέπει σε μια java εφαρμογή να αλληλοεπιδράσει με τη βάση δεδομένων. Υπάρχουν 4 τύποι JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver
3. Network Protocol driver
4. Thin driver

Σύνδεση με τη Βάση Δεδομένων

Για να συνδεθεί μια java εφαρμογή με τη βάση δεδομένων πρέπει να γίνουν 5 βήματα:

1. Καταγραφή της driver class
2. Δημιουργία σύνδεσης
3. Δημιουργία statement
4. Εκτέλεση ερωτημάτων
5. Κλείσιμο σύνδεσης

1) Για να γίνει καταγραφή της κλάσης χρησιμοποιείται η μέθοδος forName(). Αυτή η μέθοδος φορτώνει τη driver class. Σύνταξη της forName() μεθόδου:

public static void forName(String className) **throws** ClassNotFoundException

2) Για να γίνει η σύνδεση με τη βάση δεδομένων χρησιμοποιείται η μέθοδος getConnection(). Σύνταξη της getConnection() μεθόδου:

public static Connection getConnection(String url) **throws** SQLException
public static Connection getConnection(String url, String name,
String password) **throws** SQLException

3) Για τη δημιουργία ενός statement χρησιμοποιούμε τη μέθοδο createStatement(). Σύνταξη της createStatement() μεθόδου:

public Statement createStatement() **throws** SQLException

4) Για να εκτελέσουμε ένα ερώτημα στη βάση δεδομένων χρησιμοποιούμε τη μέθοδο executeQuery(). Σύνταξη της executeQuery() μεθόδου:

public ResultSet executeQuery(String sql) **throws** SQLException

Παράδειγμα που εκτελεί ένα ερώτημα:

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5) Για να κλείσουμε τη σύνδεση χρησιμοποιούμε τη μέθοδο close(). Σύνταξη της μεθόδου close():

```
public void close()throws SQLException
```

Παράδειγμα:

```
con.close();
```

Σύνδεση με MySQL χρησιμοποιώντας JDBC

Για να συνδεθεί μια εφαρμογή με τη mysql βάση δεδομένων πρέπει να γνωρίζουμε τα παρακάτω στοιχεία:

1) **Driver class:** Η driver class για τη mysql είναι **com.mysql.jdbc.Driver**.
2) **Connection URL:** Το connection url για τη mysql ΒΔ είναι **jdbc:mysql://localhost:3306/databasename**, όπου jdbc είναι το API, mysql είναι η βάση δεδομένων, localhost είναι ο server στον οποίο τρέχει η mysql(μπορούμε να βάλουμε και IP διεύθυνση), 3306 είναι η θύρα και databasename είναι το όνομα της βάσης δεδομένων.
3) **Username:** Το username που χρησιμοποιούμε για τη mysql ΒΔ (π.χ. root).
4) **Password:** Είναι ο κωδικός που έχουμε βάλει κατά την εγκατάσταση της mysql βάσης δεδομένων.

Παράδειγμα:

Δημιουργούμε μια ΒΔ company και τον πίνακα emp.

```
create database company;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Σύνδεση της java εφαρμογής με τη mysql ΒΔ:

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/company","usename","password");

Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
```

```
while(rs.next())  {
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
}
con.close();
}catch(Exception e){
System.out.println(e);}
}
}
```

Το παραπάνω παράδειγμα θα επιστρέψει και εκτυπώνει όλες τις εγγραφές του πίνακα emp.

Για να γίνει η σύνδεση της java εφαρμογής με τη mysql ΒΔ, πρέπει να φορτωθεί το αρχείο mysqlconnector.jar. Υπάρχουν δύο τρόποι για να φορτωθεί το αρχείο:

1. Επικόλληση στο φάκελο jre/lib/ext όπου βρίσκονται οι jar βιβλιοθήκες
2. Ορισμός της διαδρομής αρχείου του jar στο classpath

DriveManager class

Η κλάση DriverManager λειτουργεί ως διεπαφή μεταξύ χρηστών και drivers. Ελέγχει ποιοι drivers είναι διαθέσιμοι και διαχειρίζεται μια σύνδεση μεταξύ της βάσης και του κατάλληλου driver. Οι κυριότερες μέθοδοι που χρησιμοποιεί η κλάση είναι:

1. public static void registerDriver(Driver driver)
2. public static void deregisterDriver(Driver driver)
3. public static Connection getConnection(String url)
4. public static Connection getConnection(String url,String userName,String password)

Connection Interface

Η σύνδεση γίνεται μεταξύ της java εφαρμογής και της βάσης δεδομένων. Οι κυριότερες μέθοδοι είναι:

1. public Statement createStatement()
2. public Statement createStatement(int resultSetType,int resultSetConcurrency)
3. public void setAutoCommit(boolean status)
4. public void commit()
5. public void rollback()
6. public void close()

Statement Interface

Το Statement interface παρέχει μεθόδους οι οποίες εκτελούν ερωτήματα στη βάση δεδομένων. Οι κυριότερες μέθοδοι που χρησιμοποιούνται είναι:

1. public ResultSet executeQuery(String sql)
2. public int executeUpdate(String sql)
3. public boolean execute(String sql)
4. public int[] executeBatch()

Παράδειγμα (εισαγωγή, ενημέρωση και διαγραφή μιας εγγραφής)

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("com.mysql.jdbc.Driver ");
```

```
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/company","usename","passwo
rd");
Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp values(33,'Papadopoylos',15000)");
//int result=stmt.executeUpdate("update emp set name='Papadopoulos', salary=10000 where
id=33");
int result=stmt.executeUpdate("delete from emp where id=33");
 System.out.println(result+" records affected");
con.close();
}}
```

PreparedStatement Interface

Η συγκεκριμένη διεπαφή εκτελεί παραμετροποιημένα ερωτήματα.

Παράδειγμα

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con= DriverManager.getConnection(
"jdbc:mysql://localhost:3306/company","usename","password");

PreparedStatement stmt=con.prepareStatement("insert into emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Papadopoylos Kostas");
int i=stmt.executeUpdate();

System.out.println(i+" records inserted");
con.close();
}catch(Exception e){
System.out.println(e);
}

}
}
```

14.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 3. Γράψτε μία κλάση που θα αναπαριστά τους φοιτητές. 4. Γράψτε μία κλάση που θα φορτώνει όλους τους φοιτητές από την ΒΔ στη κύρια μνήμη σε μία λίστα. 5. Γράψτε μέθοδο που θα αποθηκεύει ένα αντικείμενο χρήστη στη βάση. 6. Γράψτε μία κλάση mail που θα χρησιμοποιεί τις παραπάνω λειτουργίες.
Ζητούμενα:	Χρήση JDBC βιβλιοθήκης για εκτέλεση εντολών στην MySQL από προγράμματα java.

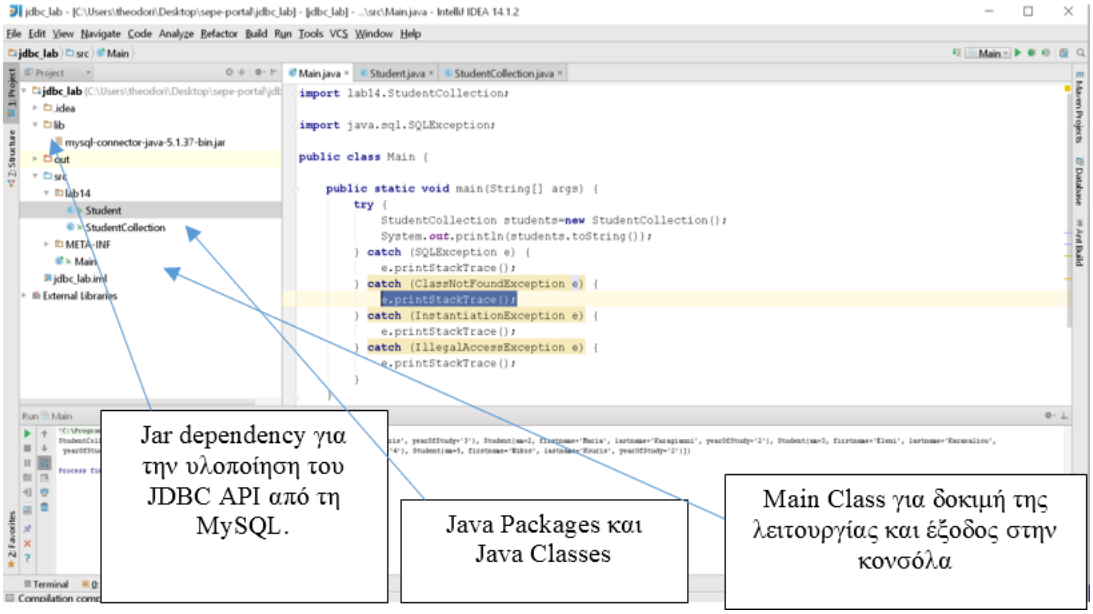
Κλάση αντικειμένων Φοιτητής (Student)

```
package lab14;


public class Student {
   private int am;
   private String firstname;
   private String lastname;
   private String yearOfStudy;

   public Student(int am, String firstname, String lastname, String yearOfStudy) {
      this.am = am;
      this.firstname = firstname;
      this.lastname = lastname;
      this.yearOfStudy = yearOfStudy;
   }

   public int getAm() {
      return am;
   }

   public void setAm(int am) {
      this.am = am;
   }

   public String getFirstname() {
      return firstname;
   }

   public void setFirstname(String firstname) {
      this.firstname = firstname;
   }

   public String getLastname() {
      return lastname;
```

```
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getYearOfStudy() {
        return yearOfStudy;
    }

    public void setYearOfStudy(String yearOfStudy) {
        this.yearOfStudy = yearOfStudy;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Student that = (Student) o;

        if (am != that.am) return false;
        if (firstname != null ? !firstname.equals(that.firstname) : that.firstname != null) return
false;
        if (lastname != null ? !lastname.equals(that.lastname) : that.lastname != null) return
false;
        if (yearOfStudy != null ? !yearOfStudy.equals(that.yearOfStudy) : that.yearOfStudy !=
null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = am;
        result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
        result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
        result = 31 * result + (yearOfStudy != null ? yearOfStudy.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Student{" +
                "am=" + am +
                ", firstname='" + firstname + '\'' +
                ", lastname='" + lastname + '\'' +
                ", yearOfStudy='" + yearOfStudy + '\'' +
                '}';
    }
}
```

Κλάση συλλογής αντικειμένων Φοιτητής (StudentCollection)

```
package lab14;

import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;


public class StudentCollection {
   List<Student> students;


   public StudentCollection() throws SQLException, ClassNotFoundException,
IllegalAccessException, InstantiationException {
       students=new ArrayList<Student>();
       Class.forName("com.mysql.jdbc.Driver").newInstance();
       Connection con = DriverManager.getConnection(
               "jdbc:mysql://localhost/university",
               "username",
               "password"); // Εντολή σύνδεσης στο ΣΔΒΔ
       Statement stmt = con.createStatement();
       ResultSet rs = stmt.executeQuery("SELECT * FROM student");

       while (rs.next()) { // Επανάληψη στις εγγραφές που έχουν επιστραφεί
          Student student=new
Student(rs.getInt("AM"),rs.getString("firstname"),rs.getString("lastname"),rs.getString("yea
rofstudy"));
          getStudents().add(student);
       }
con.close();


   }

   public List<Student> getStudents() {
      return students;
   }


   @Override
   public String toString() {
      return "StudentCollection{" +
              "students=" + Arrays.toString(students.toArray()) +
              '}';
   }
}
```



Εικόνα 14.1: *Στιγμιότυπο από το περιβάλλον ανάπτυξης.*

```
import lab14.StudentCollection;

import java.sql.SQLException;

public class Main {

   public static void main(String[] args) {
      try {
         StudentCollection students=new StudentCollection();
         System.out.println(students.toString());
      } catch (SQLException e) {
         e.printStackTrace();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      } catch (InstantiationException e) {
         e.printStackTrace();
      } catch (IllegalAccessException e) {
         e.printStackTrace();
      }
   }
}
```

Console Output (βλέπε Εικόνα 14.1 και 14.2):

StudentCollection{students=[Student{am=1, firstname='Alexandros', lastname='Theocharis', yearOfStudy='3'}, Student{am=2, firstname='Maria', lastname='Karagianni', yearOfStudy='2'}, Student{am=3, firstname='Eleni', lastname='Karaxaliou', yearOfStudy='4'}, Student{am=4, firstname='Kosmas', lastname='Pitas', yearOfStudy='4'}, Student{am=5, firstname='Nikos', lastname='Kouris', yearOfStudy='2'}]}

Μεταγλώττιση Main κλάσης:
javac.exe Main.java

Εκτέλεση:
java -cp ..\lib\mysql-connector-java-5.1.37-bin.jar; Main

StudentCollection{students=[Student{am=1, firstname='Alexandros', lastname='Theocharis', yearOfStudy='3'}, Student{am=2, firstname='Maria', lastname='Karagianni', yearOfStudy='2'}, Student{am=3, firstname='Eleni', lastname='Karaxaliou', yearOfStudy='4'}, Student{am=4, firstname='Kosmas', lastname='Pitas', yearOfStudy='4'}, Student{am=5, firstname='Nikos', lastname='Kouris', yearOfStudy='2'}]}

Εικόνα 14.2: *Αποτέλεσμα στην κονσόλα των MS Windows.*

Κλάση Φοιτητή που έχει επεκταθεί με την μέθοδο `toUpdateSQLCommand` που επιστρέφει την κατάλληλη εντολή update που ενημερώνει τις τιμές του αντίστοιχου φοιτητή στη ΒΔ.

```
package lab14;


public class Student {
   private int am;
   private String firstname;
   private String lastname;
   private String yearOfStudy;

   public Student(int am, String firstname, String lastname, String yearOfStudy) {
      this.am = am;
      this.firstname = firstname;
      this.lastname = lastname;
      this.yearOfStudy = yearOfStudy;
   }

   public int getAm() {
      return am;
   }

   public void setAm(int am) {
      this.am = am;
   }

   public String getFirstname() {
```

```
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getYearOfStudy() {
        return yearOfStudy;
    }

    public void setYearOfStudy(String yearOfStudy) {
        this.yearOfStudy = yearOfStudy;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Student that = (Student) o;

        if (am != that.am) return false;
        if (firstname != null ? !firstname.equals(that.firstname) : that.firstname != null) return
false;
        if (lastname != null ? !lastname.equals(that.lastname) : that.lastname != null) return
false;
        if (yearOfStudy != null ? !yearOfStudy.equals(that.yearOfStudy) : that.yearOfStudy !=
null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = am;
        result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
        result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
        result = 31 * result + (yearOfStudy != null ? yearOfStudy.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Student{" +
                "am=" + am +
                ", firstname='" + firstname + '\'' +
```

```
            ", lastname='" + lastname + '\'' +
            ", yearOfStudy='" + yearOfStudy + '\'' +
            '}';
    }


    public String toUpdateSQLCommand() {
        return "update student set " +
                "am=" + am +
                " , firstname='" + firstname + '\'' +
                " , lastname='" + lastname + '\'' +
                " , yearOfStudy='" + yearOfStudy + '\'' +
                " where am=" +am;
    }

}
```

Κλάση StudentCollection που έχει επεκταθεί με την μέθοδο updateStudent που ενημερώνει ένα αντικείμενο Student στην ΒΔ.

```
package lab14;

import java.sql.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;


public class StudentCollection {
    List<Student> students;


    public StudentCollection() throws SQLException, ClassNotFoundException,
IllegalAccessException, InstantiationException {
        students=new ArrayList<Student>();
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost/university",
                "root",
                "5662420!");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM student");

        while (rs.next()) {
            Student student=new
Student(rs.getInt("AM"),rs.getString("firstname"),rs.getString("lastname"),rs.getString("yea
rofstudy"));
            getStudents().add(student);
        }
        con.close();

    }
```

```
    public boolean updateStudent(Student s) throws SQLException, ClassNotFoundException,
IllegalAccessException, InstantiationException {

        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost/university",
                "username",
                "password");
        Statement stmt = con.createStatement();
        int result=stmt.executeUpdate(s.toUpdateSQLCommand());
        System.out.println(result+" records affected");
        con.close();
        if (result==1) return true;
        else return false;
    }

    public List<Student> getStudents() {
        return students;
    }


    @Override
    public String toString() {
        return "StudentCollection{" +
                "students=" + Arrays.toString(students.toArray()) +
                '}';
    }


}
```

Η κλάση Main που χρησιμοποιεί τις νέες λειτουργίες/μεθόδους.

```
import lab14.Student;
import lab14.StudentCollection;

import java.sql.SQLException;

public class Main {

    public static void main(String[] args) {
        try {
            StudentCollection students=new StudentCollection();
            System.out.println(students.toString());
            Student s=students.getStudents().get(0);
            System.out.println(s.toUpdateSQLCommand());
            s.setLastname("Papadopoulos");
            students.updateStudent(s);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}
```

14.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Δημιουργείστε και επεκτείνεται την παραπάνω εργαστηριακή άσκηση έτσι ώστε αυτή να εκτυπώνει στον χρήστη μία λίστα με τις ακόλουθες επιλογές:

1. Έξοδος
2. Αναζήτηση Φοιτητή: σε αυτή την επιλογή ο χρήστης θα πρέπει να δίνει είτε το ΑΜ είτε το Επώνυμο του φοιτητή και θα εκτυπώνονται στη κονσόλα τα στοιχεία του φοιτητή (ή φοιτητών) που βρέθηκαν
3. Επεξεργασία Φοιτητή: σε αυτή την επιλογή ο χρήστης θα μπορεί να επεξεργάζεται τα δεδομένα ενός φοιτητή
4. Διαγραφή Φοιτητή: δίνοντας το ΑΜ
5. Αποθήκευση στη ΒΔ: σε αυτή την επιλογή τα αντικείμενα-φοιτητές που έχουν επεξεργαστεί από τον χρήστη θα πρέπει να αποθηκεύονται στη ΒΔ.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση". Κεφάλαιο 9.
R. Ramakrishnan & J. Gehrke. 2002. Database Management Systems (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαια 6, 7.
MySQL JDBC Connector. http://dev.mysql.com/doc/connector-j/en/index.html
JDBC Tutorial. http://www.tutorialspoint.com/jdbc/index.htm
MySQL – Java. http://www.vogella.com/tutorials/MySQLJava/article.html