ΙΝΣΤΙΤΟΥΤΟ ΕΠΑΓΓΕΛΜΑΤΙΚΗΣ ΚΑΤΑΡΤΙΣΗΣ ΣΕΡΡΩΝ

ΕΙΔΙΚΟΤΗΤΑ ΤΕΧΝΙΚΩΝ ΕΦΑΡΜΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

**ΣΗΜΕΙΩΣΕΙΣ ΜΑΘΗΜΑΤΟΣ**

# ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΙΙ

**Διδάσκων**

**ΛΙΒΑΘΙΝΟΣ ΝΙΚΟΛΑΟΣ**

**ΣΗΜΕΙΩΣΕΙΣ ΜΑΘΗΜΑΤΟΣ**

# ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΙΙ

Λιβαθινός Νικόλαος

MSc, Μηχανικός Η/Υ και Πληροφορικής

www.livathinos.gr

Οκτώβριος 2007

## Περιεχόμενα

# 1 Εισαγωγή στις Βάσεις Δεδομένων

## 1.1 Τι είναι μια βάση δεδομένων

Μολονότι θα μπορούσαν να δοθούν αρκετοί ορισμοί, μια βάση δεδομένων είναι μια οργανωμένη συλλογή δεδομένων για γρήγορη αναζήτηση και ανάκτηση από έναν υπολογιστή. Και μολονότι στο παρελθόν υπήρξαν διάφορα μοντέλα για βάσεις δεδομένων, αυτό που τελικά επικράτησε είναι το *σχεσιακό μοντέλο* βάσεων δεδομένων.

Το σχεσιακό μοντέλο παριστάνει τη βάση δεδομένων ως μια συλλογή από σχέσεις, και μπορούμε να θεωρήσουμε την κάθε σχέση ως ένα *πίνακα*, ένα σύνολο εγγραφών, όπως φαίνεται στο ακόλουθο σχήμα:

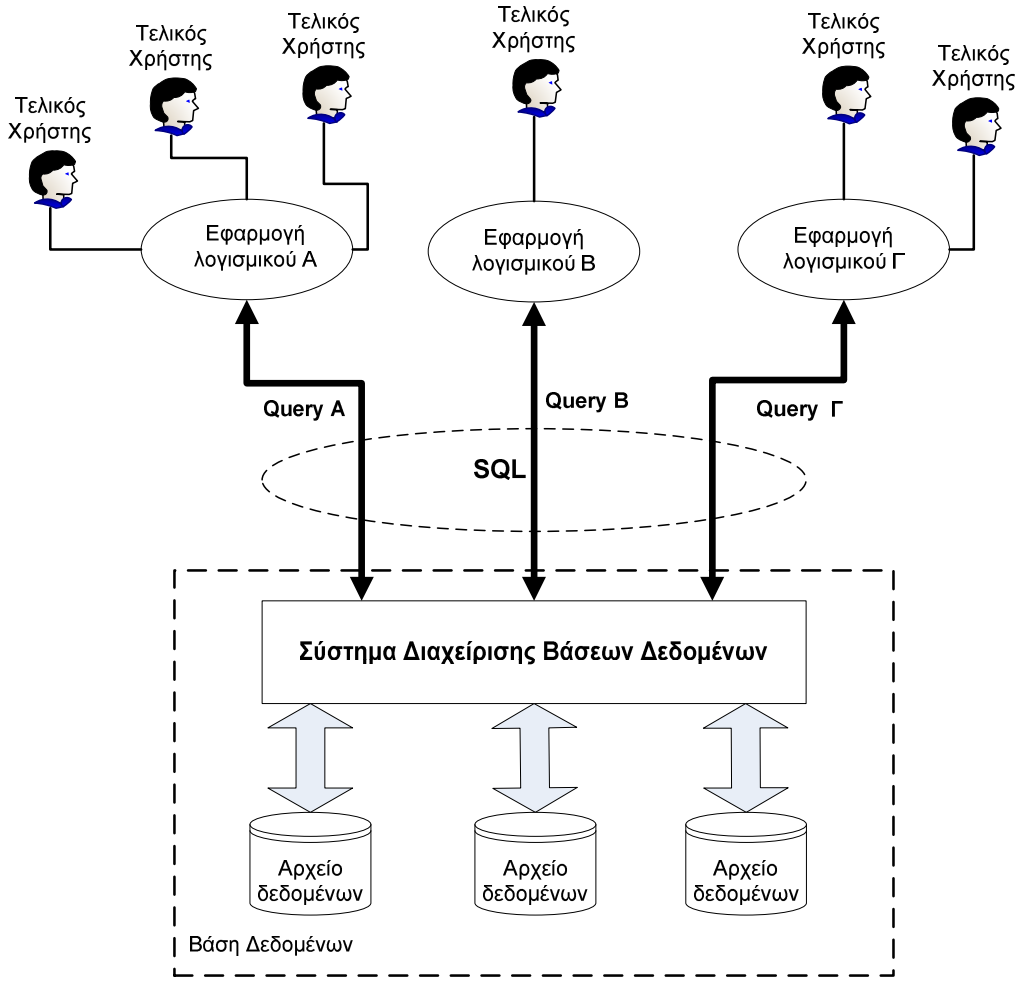
| **Πίνακας «Υπάλληλος»** | | | | |
|---|---|---|---|---|
| **Όνομα** | **Επώνυμο** | **Μισθός** | **Αριθ. Ταυτότητας** | **Τμήμα** |
| Νικόλαος | Παπαδόπουλος | 800 | Α812596 | Ανάπτυξης |
| Ανδρέας | Αβραμόπουλος | 1000 | Χ897425 | Έρευνας |
| Ιωάννης | Βασίδης | 900 | Λ125896 | Ανάπτυξης |

**Πίνακας 1: Παράδειγμα πίνακα βάσης δεδομένων**

Παρατηρούμε επομένως πως σε έναν πίνακα διακρίνουμε στήλες που διαφορετικά ονομάζονται *πεδία* και γραμμές που ονομάζονται *εγγραφές*. Κάθε πίνακας αντιστοιχεί σε μια *οντότητα* του πραγματικού κόσμου (στην προκειμένη περίπτωση έναν υπάλληλο) και τα πεδία του αντιστοιχούν στα χαρακτηριστικά *γνωρίσματα* της οντότητας που μας ενδιαφέρουν να αποθηκεύσουμε.

## 1.2 Αρχιτεκτονική ενός Συστήματος Διαχείρισης Βάσεων Δεδομένων

Ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) είναι ένα σύστημα λογισμικού που μας δίνει τη δυνατότητα να δημιουργούμε, να διαχειριζόμαστε, να τροποποιούμε, και να προσπελαύνουμε βάσεις δεδομένων. Δηλαδή είναι το πρόγραμμα που διαχειρίζεται την ίδια τη βάση και τα δεδομένα που είναι αποθηκευμένα σε αυτή. Στο ακόλουθο σχήμα φαίνεται η αρχιτεκτονική ενός απλού ΣΔΒΔ:




**Διάγραμμα 1: Αρχιτεκτονική ενός Συστήματος Διαχείρισης Βάσεων Δεδομένων**

Όπως βλέπουμε στο Διάγραμμα 1, μια βάση δεδομένων αποτελείται από το ΣΔΒΔ και ένα σύνολο εσωτερικών αρχείων που αποθηκεύονται τα δεδομένα. Ωστόσο η πρόσβαση στα δεδομένα γίνεται αποκλειστικά μέσω του ΣΔΒΔ και όχι άμεσα στα αρχεία. Επίσης βλέπουμε ότι η επικοινωνία με το ΣΔΒΔ γίνεται μέσω της γλώσσας SQL. Δηλαδή τα διάφορα προγράμματα εφαρμογών που χρειάζονται δεδομένα από τη βάση, επικοινωνούν με το ΣΔΒΔ μέσω της SQL και τελικά παρουσιάζουν στον χρήστη τις ζητούμενες πληροφορίες.

Με αυτόν τον τρόπο η πραγματική αποθήκευση των δεδομένων, δηλαδή τα εσωτερικά αρχεία της βάσης, είναι εντελώς αόρατα για τις εφαρμογές που χρησιμοποιούν τη βάση. Όλη η διαχείριση της πληροφορίας γίνεται με έμμεσο τρόπο μέσω του ΣΔΒΔ και της γλώσσας SQL. Το ΣΔΒΔ, είναι υπεύθυνο ώστε τα δεδομένα να αποθηκεύονται στα αρχεία με τέτοιο τρόπο που να καθίσταται γρήγορη η αναζήτηση και η ανάκτηση της πληροφορίας από αυτά. Για αυτό το σκοπό χρησιμοποιούνται εξειδικευμένες δομές δεδομένων που ονομάζονται *ευρετήρια*.

Πρέπει να κατανοήσουμε ότι το ΣΔΒΔ λειτουργεί ως ένα ανεξάρτητο πρόγραμμα εξυπηρετητή (server) πάνω στον οποίο συνδέονται προγράμματα πελάτες (clients). Οι clients δημιουργούν μια σύνδεση με τον server και στη συνέχεια επικοινωνούν στέλνοντάς του εντολές SQL. Κάθε περίοδος σύνδεσης του client με τον server ορίζει ένα *session* (περίοδος σύνδεσης). Έτσι στη γενική περίπτωση έχουμε έναν server και διάφορους clients που γενικά βρίσκονται διάσπαρτοι πάνω σε ένα δίκτυο δεδομένων και εξυπηρετούν διαφορετικές χρήσεις ο καθένας.

Με αυτόν τον τρόπο πολλοί clients μπορούν να είναι ταυτόχρονα συνδεδεμένοι σε έναν server και να χειρίζονται τα δεδομένα μιας βάσης στέλνοντας SQL εντολές. Για παράδειγμα θα μπορούσε ένας client να είναι ένας ο διαχειριστής της βάσης που εκτελεί εργασίες συντήρησης (π.χ. μέσω του εργαλείου MySQL Query Browser αν πρόκειται για την MySQL). Ένας άλλος client θα μπορούσε να είναι μια web εφαρμογή που αντλεί δεδομένα από τη βάση και μέσω ενός web server τα παρουσιάζει σε ένα site στο internet. Ή θα μπορούσε να είναι το κινητό τηλέφωνο ενός χρήστη που συνδέεται ασύρματα στο ΣΔΒΔ. Επομένως υπάρχουν πάμπολες εφαρμογές client που όλες τους συνδέονται στο ΣΔΒΔ, χρησιμοποιώντας τη βάση δεδομένων.

Προκειμένου να διευκολύνεται η διατήρηση της σωστής δομής των δεδομένων καθώς διάφοροι clients στέλνουν ασύγχρονα εντολές ανάκτησης και ενημέρωσης των δεδομένων, μπορούμε να ομαδοποιήσουμε τις SQL εντολές του κάθε client σε ομάδες εντολών που ονομάζονται *transactions* (συναλλαγές). Το ιδιαίτερο χαρακτηριστικό τους είναι ότι οι SQL εντολές που περιλαμβάνονται σε κάθε transaction μπορούν να εκτελεστούν για λογαριασμό του κάθε client σαν να χρησιμοποιούσε κατ' αποκλειστικότητα το ΣΔΒΔ.

Από αυτή την πλευρά ένα ΣΔΒΔ μοιάζει αρκετά με ένα λειτουργικό σύστημα. Το ΣΔΒΔ πρέπει να *χρονοπρογραμματίζει* κατάλληλα τα queries (εντολές SQL) ώστε να διατηρείται η σωστή δομή της βάσης. Επίσης είναι υπεύθυνο για την ασφάλεια των δεδομένων, για αυτό και η κάθε εφαρμογή (που από την σκοπιά του ΣΔΒΔ είναι ένας *χρήστης της βάσης*), έχει ανάλογα δικαιώματα πρόσβασης στη βάση (ανάγνωση, εγγραφή, τροποποίηση, δημιουργία κτλ).

Τέλος πρέπει να αναφέρουμε ότι τα περισσότερα ΣΔΒΔ παρέχουν ένα πλούσιο σύνολο από εργαλεία και βοηθητικά προγράμματα για την διαχείριση των δεδομένων όπως τη δημιουργία αντιγράφων ασφαλείας (backup) ή την εισαγωγή δεδομένων από backup, την μετατροπή των βάσεων σε μορφές συμβατές με άλλα ΣΔΒΔ, ή ακόμη και πιο προχωρημένες λειτουργίες όπως η δημιουργία γραφικού περιβάλλοντος για αλληλεπίδραση του ΣΔΒΔ με τον τελικό χρήστη (π.χ. οι φόρμες στην Access).

## 1.3 Τι είναι η SQL

Η SQL (Structured Query Language – Δομημένη Γλώσσα Ερωτήσεων) είναι το πρότυπο γλώσσας μέσω της οποίας χειριζόμαστε ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων. Η SQL είναι μια πλήρης γλώσσα βάσεων δεδομένων που διαθέτει εντολές για ορισμό δεδομένων, ερωτήσεις και ενημερώσεις. Επομένως μπορούμε να διακρίνουμε δύο υποσύνολα εντολών στην SQL, τις εντολές που αφορούν τον ορισμό και τη διαχείριση της δομής της βάσης (γλώσσα ορισμού δεδομένων) και τις εντολές που αφορούν την διαχείριση των ίδιων των δεδομένων (γλώσσας χειρισμού δεδομένων). Στους επόμενους υπότιτλους θα δούμε πιο αναλυτικά τι περιλαμβάνεται σε κάθε ένα από αυτά τα υποσύνολα της SQL.

Είναι ενδιαφέρον να σημειώσουμε πως τα περισσότερα ΣΔΒΔ έχουν κάνει προσθήκες πάνω στην standard SQL, δηλαδή επιπρόσθετες εντολές στο πρότυπο σύνολο εντολών της SQL που υποστηρίζονται μόνο από το συγκεκριμένο ΣΔΒΔ. Έτσι μπορούμε να αξιοποιήσουμε πλήρως όλες τις δυνατότητες που μας παρέχει το ΣΔΒΔ και να εξειδικεύσουμε τη σχεδίαση της βάσης ανάλογα με τις λεπτομέρειες υλοποίησης του εκάστοτε ΣΔΒΔ.

### 1.3.1 Γλώσσα Ορισμού Δεδομένων (ΓΟΔ)

Εφόσον τα δεδομένα σε μια (σχεσιακή) βάση δεδομένων αποθηκεύονται σε πίνακες, η ΓΟΔ αναλαμβάνει την δημιουργία και διαχείριση των πινάκων. Αυτό περιλαμβάνει τον προσδιορισμό του τύπου δεδομένων και των ιδιοτήτων των πεδίων του πίνακα, αλλά και επιπρόσθετων ιδιοτήτων πάνω στον ίδιο τον πίνακα (π.χ. αν οι χαρακτήρες θα είναι στα ελληνικά ή τα αγγλικά, κ.ά.). Επίσης διαχειρίζεται τις ίδιες τις βάσεις μέσα στο ΣΔΒΔ, ορίζει χρήστες βάσης και τα δικαιώματά τους πάνω στους πίνακες κ.ά. Το σύνολο όλων αυτών των οντοτήτων ονομάζεται *σχέδιο* (scheme).

### 1.3.2 Γλώσσα Χειρισμού Δεδομένων (ΓΧΔ)

Ο χειρισμός των δεδομένων αφορά εντολές για την ένθεση, την ενημέρωση την ανάκτηση και την διαγραφή των δεδομένων μέσα στη βάση. Έχοντας δηλαδή σχεδιάσει τους πίνακες της βάσης (δηλ. τη δομή της) μέσω της ΓΟΔ, τώρα μπορούμε με την ΓΧΔ να προσθέσουμε εγγραφές στους πίνακες ή να αναζητήσουμε δεδομένα ορίζοντας κριτήρια αναζήτησης, τα πεδία που θέλουμε να επιστραφούν, πως θέλουμε να συνδυαστούν οι πληροφορίες από τους διάφορους πίνακες κτλ. Μια εντολή SQL για αναζήτηση δεδομένων ονομάζεται *query* (δηλ. ερώτημα).

Για παράδειγμα με βάση τα δεδομένα του Πίνακας 1, θα μπορούσαμε να κάνουμε ερωτήσεις όπως:

- «Ποιοι υπάλληλοι εργάζονται στο τμήμα ανάπτυξης;»
- «Ποιο υπάλληλοι έχουν μισθό μεταξύ 750 και 900 ευρώ;»
- «Ποιοι υπάλληλοι έχουν επώνυμο που καταλήγει σε –όπουλος;»

## 1.4 MySQL και άλλα ΣΔΒΔ

Το πρόγραμμα σπουδών του μαθήματος προβλέπει την χρήση της MySQL ως ΣΔΒΔ. Η MySQL είναι ένα πολύ δημοφιλές ΣΔΒΔ, και το ιδιαίτερο χαρακτηριστικό της είναι πως αποτελεί ένα λογισμικό ανοιχτού κώδικα (open source). Η ελεύθερη – δωρεάν έκδοση της MySQL διατίθεται στην διεύθυνση http://www.mysql.org. Λόγω της έντονης ανάπτυξης που έχει η MySQL είναι σημαντικό να γνωρίζουμε σε ποια έκδοσή της αναφερόμαστε, καθώς οι νεότερες εκδόσεις υποστηρίζουν πολλές νέες δυνατότητες. Στο παρόν μάθημα αναφερόμαστε στην έκδοση 5.0.45. Όπως αναφέρθηκε και προηγουμένως το κάθε ΣΔΒΔ έχει τις δικές του προσθήκες στην SQL, ομοίως και η MySQL. Για αυτό και όποτε η εντολή που παρουσιάζεται δεν υπακούει στο πρότυπο SQL αλλά είναι προσθήκη, τότε αυτό θα διευκρινίζεται, διαφορετικά η εντολή θα εφαρμόζεται σε οποιοδήποτε ΣΔΒΔ.

Όπως θα δούμε και στη συνέχεια η MySQL ορισμένες φορές παρουσιάζει αποκλίσεις από την standard SQL, άλλες φορές εισάγει δυνατότητες που ξεπερνούν το πρότυπο και άλλες φορές υστερεί σε δυνατότητες που περιγράφονται από το πρότυπο (όπως άλλωστε συμβαίνει και με τα περισσότερα ΣΔΒΔ). Όποτε συμβαίνει αυτό θα το σημειώνουμε κατάλληλα.

Μερικές άλλες περιπτώσεις από ΣΔΒΔ που ξεχωρίζουν παρουσιάζονται στον ακόλουθο πίνακα:

| **Όνομα** | **Κατασκευαστής** | **URL** | **Τύπος λογισμικού** |
|---|---|---|---|
| Oracle | Oracle | http://www.oracle.com | Εμπορικό/ κλειστό |
| SQL Server | Microsoft | http://www.microsoft.com/sql | Εμπορικό/ κλειστό |
| DB2 | IBM | http://www.ibm.com/db2 | Εμπορικό/ κλειστό |
| MySQL | MySQL | http://www.mysql.com | Δωρεάν/ ανοιχτού κώδικα (η community version μόνο) |
| PostGreSQL | PostgreSQL | http://www.postgresql.com | Δωρεάν/ ανοιχτού κώδικα |
| Access | Microsoft | http://office.microsoft.com/access | Εμπορικό/κλειστό |

# 2 Ορισμός δεδομένων στην SQL

## 2.1 Τύποι δεδομένων στα πεδία

Όπως είδαμε στο προηγούμενο κεφάλαιο οι σχεσιακές βάσεις δεδομένων αναπαρίστανται ως ένα σύνολο πινάκων με τον κάθε πίνακα να αποτελείται από ένα σύνολο στηλών. Η κάθε στήλη (πεδίο) αποθηκεύει δεδομένα ενός προκαθορισμένου τύπου. Συγκεκριμένα το πρότυπο της SQL ορίζει τους ακόλουθους τύπους δεδομένων:

| **Κατηγορία τύπου δεδομένων** | **Όνομα τύπου** | **Περιγραφή τύπου** |
|---|---|---|
| Αριθμητικοί | int | Ακέραιος από -2147483648 έως 2147483647 |
| | smallint | Ακέραιος από -32768 έως 32767 |
| | float(d) | Κινητής υποδιαστολής με d ψηφία στο κλασματικό, όπου D<=24 |
| | double precision | Κινητής υποδιαστολής με M ψηφία στο ακέραιο μέρος και D στο κλασματικό, όπου D<=53 |
| | numeric(p,d) | Αριθμός ακριβούς αναπαράστασης (κατάλληλος για χρηματικά ποσά) με συνολικά p ψηφία εκ των οποίων τα d είναι δεξιά της υποδιαστολής στο κλασματικό μέρος. |

| | | |
|---|---|---|
| Χαρακτήρων | char(n) | Σταθερού μεγέθους συμβολοσειρά n χαρακτήρων |
| | varchar(n) | Μεταβλητού μεγέθους συμβολοσειρά έως n χαρακτήρων |
| | text | Κείμενο μεγάλου μήκους |
| Ημερομηνία / ώρα | date | Ημερομηνία της μορφής YYYY-MM-DD |
| | time | Ώρα της μορφής<br>HH:MM:SS |

**Πίνακας 2: Τύποι δεδομένων στην SQL**

Αυτό σημαίνει ότι όλα τα ΣΔΒΔ πρέπει να υποστηρίζουν τους τύπους του Πίνακας 2. Ωστόσο το κάθε ΣΔΒΔ κάνει και τις δικές του προσθήκες ή εξειδικεύσεις, συγκεκριμένα η MySQL προσθέτει επιπλέον και τους ακόλουθους τύπους (καθώς και άλλους που δεν φαίνονται στον πίνακα):

| **Κατηγορία τύπου δεδομένων** | **Όνομα τύπου** | **Περιγραφή τύπου** |
|---|---|---|
| Αριθμητικοί | bigint | Ακέραιος από -9223372036854775808 έως 9223372036854775807 |
| | tinyint | Ακέραιος από -128 έως 127 |
| Χαρακτήρων | enum(<string1>,…,<string n>) | Το κάθε string ορίζει μια από τις προκαθορισμένες επιλογές |

**Πίνακας 3: Επιπρόσθετοι τύποι δεδομένων που ισχύουν στη MySQL**

Η επιλογή του τύπου δεδομένων που θα προσθέσουμε σε ένα πεδίο θα πρέπει να εξαρτηθεί από το είδος των δεδομένων που θα αποθηκεύσουμε σε αυτό. Συγκεκριμένα θα πρέπει να λάβουμε υπόψη μας τόσο τον τύπο τους (ακέραιος, χαρακτήρας, κινητής υποδιαστολής, ημερομηνία κτλ) όσο και το *εύρος* τους. Για παράδειγμα αν θέλουμε να αποθηκεύσουμε την ηλικία ενός ανθρώπου τότε ένα πεδίο `smallint` είναι αρκετό (ή αν χρησιμοποιούμε mysql ακόμα και ένα πεδίο `tinyint`). Αν όμως θέλουμε να αποθηκεύσουμε κωδικούς για το σύνολο των τραπεζικών συναλλαγών μιας τράπεζας (που είναι εκατομμύρια) θα πρέπει να βάλουμε `int`.

Μεγάλη προσοχή χρειάζεται όταν αποθηκεύουμε αριθμούς με κλασματικό μέρος. Σε αυτή την περίπτωση πρέπει να ελέγξουμε κατά πόσο μας ενδιαφέρει η μεγάλη ακρίβεια των αριθμών ή όχι. Ιδιαίτερη προσοχή χρειάζεται στην αποθήκευση χρηματικών ποσών, όπου θα πρέπει να χρησιμοποιούμε τον τύπο `numeric(p,d)`. Αν για παράδειγμα θέλουμε να έχουμε 8 ψηφία στο ακέραιο μέρος και 2 στο κλασματικό, τότε θα ορίσουμε numeric(10,2), δηλαδή πρώτα γράφουμε το άθροισμα των ψηφίων και μετά το πλήθος των ψηφίων μετά την υποδιαστολή. Προσοχή ποτέ δεν πρέπει να χρησιμοποιούμε τους τύπους float/double όταν θέλουμε να αποθηκεύσουμε αριθμούς με μεγάλη ακρίβεια στα ψηφία τους, διαφορετικά θα συσσωρεύονται σφάλματα αναπαράστασης που θα μας δίνουν μεγάλη απόκλιση. Επειδή όμως οι τύποι float / double είναι πολύ πιο γρήγοροι στον

υπολογισμό τους, μπορούμε να τους χρησιμοποιούμε όταν έχουμε περιπτώσεις που μια μικρή απόκλιση δεν είναι σημαντική (π.χ. το ύψος ενός ανθρώπου).

Σε περίπτωση που έχουμε χαρακτήρες θα πρέπει να επιλέξουμε μεταξύ των τύπων char(n), varchar(n) και text. Ο τύπος char(n) προσδιορίζει ένα *σταθερό* σύνολο από n χαρακτήρες, οπότε τον χρησιμοποιούμε όταν έχουμε ένα προκαθορισμένο πλήθος χαρακτήρων. Για παράδειγμα ο αριθμός ταυτότητας είναι ένα σύνολο από 7 χαρακτήρες σταθερά. Όταν όμως δεν γνωρίζουμε το πλήθος των χαρακτήρων που θέλουμε να αποθηκεύσουμε τότε χρησιμοποιούμε τον τύπο `varchar(n)`. Σε αυτή την περίπτωση ορίζουμε ότι το πεδίο θα αποθηκεύει *έως* n χαρακτήρες. Για παράδειγμα το όνομα ενός ανθρώπου δεν έχει σταθερό μήκος, ωστόσο κάνουμε μια εκτίμηση για το μέγιστο μήκος του και ορίζουμε πχ. `varchar(10)`. Αν πάλι θέλουμε να αποθηκεύσουμε ένα πεδίο που έχει μεν μεταβλητό μήκος αλλά αναμένουμε να είναι πολύ μεγάλο (π.χ. ένα βιβλίο) τότε θα χρησιμοποιήσουμε τον τύπο `text`.

Τέλος όσον αφορά τα πεδία για ημερομηνία / ώρα, πρέπει να προσέχουμε την σωστή μορφοποίηση των δεδομένων. Για παράδειγμα η ημερομηνία πρέπει να δίνεται στη μορφή `'EEEE-MM-HH'`, δηλαδή 4 ψηφία για το έτος, κατόπιν 2 ψηφία για τον μήνα και 2 ψηφία για την ημέρα. Επίσης η ώρα πρέπει να δίνεται στη μορφή `'ΩΩ:ΛΛ:ΔΔ'`, δηλαδή 2 ψηφία για την ώρα, 2 για τα λεπτά και 2 για τα δευτερόλεπτα.

### 2.1.1 Τιμή NULL

Ειδική περίπτωση στα δεδομένα ενός πίνακα αποτελεί η τιμή `NULL`. Όταν ένα πεδίο είναι «άδειο» τότε κατέχει την τιμή `NULL`, δηλαδή τίποτα. Για παράδειγμα ας θεωρήσουμε τον ακόλουθο πίνακα:

| **Μαθητής** | | |
|---|---|---|
| **Όνομα** | **Πατρώνυμο** | **Επώνυμο** |
| Ιωάννης | NULL | Βασίδης |
| Μιχάλης | NULL | Παρασκευόπουλος |

Σε αυτή την περίπτωση απλά δεν έχει συμπληρωθεί το πατρώνυμο στις εγγραφές του πίνακα, οπότε το πατρώνυμο είναι `NULL`. Αξίζει να σημειωθεί ότι η τιμή NULL δεν έχει τύπο, αποτελεί μια ειδική περίπτωση και όπως θα δούμε και στο επόμενο κεφάλαιο ο έλεγχος αν ένα πεδίο έχει την τιμή `NULL` γίνεται με ξεχωριστό τρόπο.

## 2.2 Ορισμοί πινάκων

### 2.2.1 Δημιουργία της δομής ενός πίνακα

Για να δημιουργήσουμε έναν πίνακα χρησιμοποιούμε την εντολή `CREATE TABLE`. Ένα πρότυπο σύνταξης αυτής της εντολής είναι:

```
CREATE TABLE <όνομα πίνακα>(
	<όνομα πεδίου 1> <τύπος πεδίου 1>,
	<όνομα πεδίου 2> <τύπος πεδίου 2>,
…
	<όνομα πεδίου N> <τύπος πεδίου N>
);
```

Στο παραπάνω πρότυπο (καθώς και στα επόμενα που θα παρουσιαστούν) όταν παρουσιάζεται κάτι εντός των τριγωνικών αγκύλων <>, εννοείται ότι θα αντικατασταθεί (μαζί με τις αγκύλες) με αυτό που πρέπει κάθε φορά. Ας υποθέσουμε ότι θέλουμε να συντάξουμε μια εντολή για τον πίνακα που ακολουθεί:

| **Προϊόν** | |
|---|---|
| **Όνομα πεδίου** | **Τύπος πεδίου** |
| Κωδικός | int |
| Όνομα | varchar(20) |
| Ημερ_Παραγωγής | date |
| Κόστος | numeric(6,2) |

Έχουμε δηλαδή τον πίνακα για ένα προϊόν που έχει 4 πεδία. Έναν ακέραιο κωδικό, το όνομά του, την ημερομηνία παραγωγής του και το κόστος του. Αντίστοιχα έχουμε επιλέξει τους τύπους πεδίων `int`, `varchar(20)`, `date`, `numeric(6,2)`. Αυτό σημαίνει ότι το όνομα του πεδίου θα πρέπει να είναι το πολύ έως 20 χαρακτήρες, και το κόστος του να έχει 4 ψηφία στο ακέραιο μέρος και 2 στο κλασματικό (άρα συνολικά 6). Οπότε θα έχουμε την ακόλουθη εντολή `CREATE`:

```
CREATE TABLE product(
	id INT,
	name VARCHAR(20),
	production DATE,
```

```
    cost   NUMERIC(6,2)
);
```

Με αυτά υπόψη θα μπορούσαμε να έχουμε το ακόλουθο στιγμιότυπο του πίνακα που μόλις ορίσαμε, γεμάτο με δεδομένα:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |
| 3 | CPU P4 3.2GHz | 2006-08-20 | 250.00 |
| 4 | Μνήμη 1GB | 2007-05-23 | 70.00 |

## 2.2.2 Διαγραφή ενός πίνακα

Μπορούμε να διαγράψουμε έναν ολόκληρο πίνακα, μαζί με τα δεδομένα που τυχόν έχει, χρησιμοποιώντας την εντολή `DROP`, σύμφωνα με το ακόλουθο πρότυπο:

```
DROP TABLE <όνομα πίνακα>
```

Έτσι αν θέλουμε να διαγράψουμε τον πίνακα product τότε θα γράψουμε:

```
DROP TABLE product
```

## 2.2.3 Αλλαγή της δομής ενός πίνακα

Η SQL μας δίνει τη δυνατότητα να τροποποιούμε την δομή ενός πίνακα ακόμα και μετά τη δημιουργία του. Μπορούμε δηλαδή να προσθαφαιρέσουμε στήλες ή να τροποποιήσουμε τον τύπο μιας στήλης ενός ήδη υπάρχοντα πίνακα. Αυτό γίνεται μέσω της εντολής `ALTER TABLE`. Συγκεκριμένα όταν θέλουμε να προσθέσουμε ακόμα ένα πεδίο τότε ακολουθούμε το πρότυπο:

```
ALTER TABLE <όνομα πίνακα> ADD <όνομα πεδίου> <τύπος πεδίου>
```

Π.χ. Αν θέλουμε στον πίνακα product να προσθέσουμε ακόμα ένα πεδίο με όνομα manufacturer (δηλ. κατασκευαστής) και τύπο `varchar(20)`. Τότε θα γράψουμε την ακόλουθη εντολή:

```
ALTER TABLE product ADD manufacturer VARCHAR(20);
```

Επίσης μπορούμε να διαγράψουμε το πεδίο ενός πίνακα ακολουθώντας το πρότυπο:

```
ALTER TABLE <όνομα πίνακα> DROP <όνομα πεδίου>
```

## 2.3 Κλειδί πίνακα

Τις περισσότερες φορές θέλουμε να μπορούμε να αναγνωρίζουμε τις εγγραφές ενός πίνακα, προσδιορίζοντας ένα ή περισσότερα πεδία τους ως μοναδικά. Για παράδειγμα στον πίνακα προϊόν, ο κωδικός είναι μοναδικός για κάθε προϊόν και έτσι αν γνωρίζουμε μόνο αυτό το πεδίο θα μπορούμε να αναγνωρίζουμε όλη την εγγραφή. Λέμε λοιπόν ότι το πεδίο κωδικός είναι *πρωτεύον κλειδί* του πίνακα προϊόν.

Σε άλλες περιπτώσεις θα μπορούσαμε να θέσουμε ένα σύνολο από πεδία ως κλειδί για ένα πίνακα (δηλ. ένα σύνθετο κλειδί). Για παράδειγμα στον πίνακα των μαθητών:

| **Μαθητής** | | |
|---|---|---|
| **Όνομα** | **Πατρώνυμο** | **Επώνυμο** |
| Ιωάννης | Σπύρος | Βασίδης |
| Μιχάλης | Σπύρος | Παρασκευόπουλος |

Θα μπορούσε κάποιος να ορίσει ότι ο συνδυασμός των πεδίων όνομα και επώνυμο αποτελούν το κλειδί για κάθε εγγραφή του πίνακα. Ωστόσο γενικά είναι καλό να χρησιμοποιείται ένα ξεχωριστό πεδίο για κλειδί, το οποίο είναι εξ' ορισμού μοναδικό (π.χ. ο αριθμός μητρώου του μαθητή). Έστω λοιπόν ότι συμπληρώνουμε τον πίνακα, προσθέτοντας ένα πεδίο για τον αριθμό μητρώου και ένα για τον αριθμό ταυτότητας του μαθητή:

| **Μαθητής** | | | | |
|---|---|---|---|---|
| **Αρ. Ταυτότητας** | **Αρ. Μητρώου** | **Όνομα** | **Πατρώνυμο** | **Επώνυμο** |
| Χ234567 | 2589 | Ιωάννης | Σπύρος | Βασίδης |
| Λ987654 | 2473 | Μιχάλης | Σπύρος | Παρασκευόπουλος |

Σε αυτή την περίπτωση βλέπουμε ότι έχουμε πολλές δυνατότητες για να επιλέξουμε κλειδί. Για παράδειγμα μερικές ιδέες για το ποια πεδία θα μπορούσαν να γίνουν το κλειδί είναι:

- Αρ. Ταυτότητας.
- Αρ. Μητρώου.

Βλέπουμε δηλαδή ότι έχουμε δύο πεδία με εξασφαλισμένη μοναδικότητα, άρα το καθένα είναι ένα *υποψήφιο κλειδί* και αυτό που τελικά θα επιλέξουμε είναι το *πρωτεύον κλειδί* (primary key).

Επομένως συνοψίζουμε τα συμπεράσματά μας για τα κλειδιά στον ακόλουθο πίνακα:

| **Ορισμός** | **Περιγραφή** |
|---|---|

| | |
|---|---|
| Κλειδί | Ένα ή περισσότερα πεδία που εγγυημένα θα είναι διαφορετικά σε κάθε εγγραφή |
| Υποψήφιο κλειδί | Ένα από τα κλειδιά του πίνακα |
| Πρωτεύον κλειδί | Ένα από τα υποψήφια κλειδιά που τελικά θα χρησιμοποιηθεί ως κλειδί |

Εφόσον το κλειδί προσδιορίζει μοναδικά την κάθε εγγραφή είναι λογικό ότι δεν μπορεί να παίρνει την τιμή NULL, διότι αυτό θα σήμαινε ότι η εγγραφή δεν θα υπήρχε. Επομένως για να ορίσουμε τον πίνακα «Μαθητής» στην τελική του μορφή (λαμβάνοντας υπόψιν και ότι ο αρ. ταυτότητας είναι κλειδί) θα έχουμε την ακόλουθη εντολή CREATE:

```
CREATE TABLE student(
        ar_tautotitas CHAR(7) NOT NULL,
        ar_mitrou int NOT NULL UNIQUE,
        name VARCHAR(20),
        middleName VARCHAR(20),
        surname VARCHAR(20),
        PRIMARY KEY (ar_tautotitas)
);
```

Στην εντολή αυτή προσέχουμε 3 σημεία (μέσα σε πλαίσιο). Αρχικά στο πεδίο ar_tautotitas (που έχουμε αποφασίσει να το θέσουμε ως πρωτεύον κλειδί), προσθέτουμε την επιλογή NOT NULL μετά τον τύπο του. Με αυτόν τον τρόπο απαγορεύουμε στο πεδίο ar_tautotitas να είναι κενό. Κάνουμε το ίδιο και για το πεδίο ar_mitroou, γιατί ομοίως δεν επιτρέπουμε ένας μαθητής να μην έχει αριθμό μητρώου. Επιπλέον όμως, επειδή ο αριθμός μητρώου είναι και αυτός μοναδικός (ως υποψήφιο κλειδί) προσθέτουμε και τον χαρακτηρισμό UNIQUE, ότι δηλαδή πρέπει να είναι μοναδικός. Τέλος μέσω της εντολής PRIMARY KEY ορίζουμε ποιο πεδίο θέλουμε να είναι το πρωτεύον κλειδί. Τα πεδία που αποτελούν το πρωτεύον κλειδί παρατίθενται εντός παρένθεσης χωρισμένα με κόμμα. Έτσι εφόσον τώρα έχουμε το πεδίο ar_tautotitas ως κλειδί βάζουμε αυτό μόνο.

Αν για παράδειγμα θέλαμε να ορίσουμε τον συνδυασμό του ονόματος, πατρωνύμου και επωνύματος ως πρωτεύον κλειδί τότε ο ορισμός του πίνακα θα ήταν:

```
CREATE TABLE student(
        ar_tautotitas CHAR(7) NOT NULL UNIQUE,
        ar_mitrou int NOT NULL UNIQUE,
        name VARCHAR(20) NOT NULL,
        middleName VARCHAR(20) NOT NULL,
        surname VARCHAR(20) NOT NULL,
```

```
        PRIMARY KEY (name, surname, middleName)
);
```

Δηλαδή βάζουμε στο primary key όλα τα πεδία που θέλουμε να είναι κλειδιά, για καθένα από αυτά ορίζουμε ότι δεν είναι `NULL` και αν θέλουμε ορίζουμε και ότι ο αρ. ταυτότητας πρέπει να είναι μοναδικός.

# 3 Χειρισμός δεδομένων στην SQL

## 3.1 Προσθήκη δεδομένων

### 3.1.1 Εντολή INSERT

Μπορούμε να εισάγουμε δεδομένα σε έναν πίνακα μέσω της εντολής INSERT. Η εντολή INSERT δίνει τη δυνατότητα να ορίσουμε εμείς ποια πεδία θέλουμε να εισαγάγουμε ανά γραμμή ή ακόμα και να εισαγάγουμε πολλές γραμμές ταυτόχρονα. Το γενικό πρότυπο που ακολουθείται είναι:

```
INSERT INTO <όνομα πίνακα> VALUES(<τιμή πεδίου 1>, <τιμή πεδίου 2>, … , <τιμή πεδίου n>)
```

Για παράδειγμα προκειμένου να εισάγουμε μια εγγραφή στον πίνακα product δίνουμε την ακόλουθη εντολή:

```
INSERT INTO product VALUES (1, 'Ποντίκι Optical', '2007-09-15', 23.50);
```

Πρέπει να προσέχουμε ώστε να βάζουμε τα πεδία με τη σωστή σειρά. Θα πρέπει δηλαδή να γράφουμε τα πεδία της ένθεσης με τη σειρά που έχουν δηλωθεί στην CREATE. Ωστόσο υπάρχει και μια εναλλακτική σύνταξη της εντολής insert που μας δίνει τη δυνατότητα να προσδιορίσουμε δική μας σειρά στα πεδία:

```
INSERT INTO <όνομα πίνακα>(<πεδίο Α>, <πεδίο Β>, … ,<πεδίο Ν>) VALUES (<τιμή πεδίου Α>, <τιμή πεδίου Β>, …, <τιμή πεδίου Ν>)
```

Η διαφορά που έχει αυτή η μορφή της INSERT είναι ότι μετά το όνομα του πίνακα παραθέτουμε μέσα σε παρένθεση το όνομα του πεδίου που θέλουμε. Κατόπιν οι τιμές των πεδίων θα πρέπει να ακολουθούν αυτή την διάταξη. Για παράδειγμα την προηγούμενη εγγραφή θα μπορούσαμε να την εισάγουμε με τον εξής τρόπο:

```
INSERT INTO product(cost, id, name, production) VALUES (23.50, 1, 'Ποντίκι Optical', '2007-09-15')
```

Βλέπουμε δηλαδή ότι μετά το όνομα του πίνακα παραθέτουμε τα πεδία του με τη σειρά που επιθυμούμε μέσα σε παρένθεση. Κατόπιν στο VALUES εισάγουμε τις τιμές των πεδίων με την νέα σειρά. Με αυτόν τον τρόπο δεν χρειάζεται να θυμόμαστε τη σειρά με την οποία έχουν τοποθετηθεί τα πεδία μέσα στον πίνακα, αλλά απλά το όνομά τους.

Έχοντας εκτελέσει την προηγούμενη εντολή INSERT ο πίνακας «προϊόν» θα έχει τα ακόλουθα δεδομένα (θεωρώντας ότι αρχίζουμε με τον πίνακα άδειο).

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |

Θέλοντας να εισαγάγουμε πολλές εγγραφές στον πίνακα έχουμε δύο εναλλακτικές λύσεις, είτε να επαναλαμβάνουμε εντολές INSERT μια για κάθε εγγραφή, είτε να χρησιμοποιήσουμε μια παραλλαγή της INSERT που επιτρέπει με μια εντολή να γίνονται πολλαπλές ενθέσεις. Για παράδειγμα ας δούμε πως θα μπορούσαμε να ενθέσουμε ακόμη τρεις εγγραφές:

```
INSERT INTO proion VALUES (2, 'Σκληρός δίσκος 300GB', '2007-09-15', 85.00),
(3, 'CPU P4 3.2GHz, '2006-08-20', 250.00),
(4, 'Μνήμη 1GB', '2007-05-23', 70.00);
```

Οπότε ο πίνακας θα έχει τις ακόλουθες εγγραφές:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |
| 3 | CPU P4 3.2GHz | 2006-08-20 | 250.00 |
| 4 | Μνήμη 1GB | 2005-12-23 | 70.00 |

Παρατηρούμε δηλαδή πως μπορούμε να επαναλαμβάνουμε εντός παρενθέσεων όσες εγγραφές θέλουμε μετά το VALUES. Η πολλαπλή ένθεση μπορεί να συνδυαστεί και με ορισμό της σειράς των πεδίων. Π.χ.

```
INSERT INTO proion(cost, id, name, production) VALUES
 (85.00, 2, 'Σκληρός δίσκος 300GB', '2007-09-15'),
 (250.00, 3, 'CPU P4 3.2GHz, '2006-08-20'),
 (70.00, 4, 'Μνήμη 1GB', 2007-05-23);
```

## 3.2 Ανάκτηση δεδομένων από έναν πίνακα

Έχοντας δει πως μπορούμε να εισάγουμε εγγραφές σε έναν πίνακα, τώρα μας ενδιαφέρει το ουσιαστικότερο μέρος της SQL που είναι η ανάκτηση των δεδομένων. Θέλουμε να δίνουμε εντολές στη βάση και να μας επιστρέφει τις πληροφορίες που έχουμε εισάγει στους πίνακες βάση κριτηρίων επιλογής, συνδυασμών και συναρτήσεων που ορίζουμε εμείς. Για παράδειγμα χρησιμοποιώντας τον πίνακα των προϊόντων θα μπορούσαμε να κάνουμε μια ερώτηση του τύπου:

- Ποια προϊόντα έχουν κόστος μικρότερο από 100 ευρώ;

Σε αυτή την περίπτωση το ΣΔΒΔ πρέπει να επιστρέψει το ακόλουθο αποτέλεσμα:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |
| 4 | Μνήμη 1GB | 2007-05-23 | 70.00 |

Ή αν κάνουμε την ερώτηση:

- Ποιο είναι το όνομα του ακριβότερου προϊόντος;

Τότε πρέπει να μας επιστρέψει την απάντηση:

| **Προϊόν** |
|---|
| **Όνομα** |
| Σκληρός δίσκος 300GB |

Τέλος αν του κάνουμε την ερώτηση:

- Ποια προϊόντα έχουν κατασκευαστεί πριν το έτος 2000;

Θα μας επιστρέψει την απάντηση:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |

Επομένως σε κάθε περίπτωση η απάντηση είναι και αυτή ένας πίνακας! Τα πεδία του πίνακα είναι αυτά που έχουμε ζητήσει στην ερώτηση (δηλ. όλα τα πεδία του πίνακα ή μόνο μερικά από αυτά ή ίσως και κάποια τεχνητά πεδία) και οι εγγραφές που επιστρέφονται είναι αυτές που ικανοποιούν τα κριτήρια αναζήτησης (αν έχουμε βάλει κάποιο κριτήριο).

Επειδή ακριβώς η ανάκτηση των δεδομένων είναι η απάντηση σε «ερωτήσεις», μια εντολή ανάκτησης δεδομένων ονομάζεται και query, που σημαίνει ερώτηση (ή και επερώτηση).

### 3.2.1 Επιλογή πεδίων με την SELECT

Η εντολή στην SQL μέσω της οποίας ανακτούμε πληροφορίες (συντάσσουμε queries) είναι η SELECT. Η γενική σύνταξη της SELECT είναι αρκετά σύνθετη, ωστόσο ένα απλό πρότυπο είναι το ακόλουθο:

```
SELECT <πεδία που θέλουμε να φαίνονται>
FROM <πίνακες από τους οποίους θα αντληθούν τα δεδομένα>
WHERE <κριτήρια επιλογής των εγγραφών>
```

Για παράδειγμα μπορούμε να πάρουμε το όνομα του προϊόντος με κωδικό 1, δίνονται την εντολή:

```
SELECT name FROM product WHERE id=1
```

Αν θέλουμε να πάρουμε μαζί με το όνομα και το κόστος, τότε η εντολή γίνεται

```
SELECT name, cost FROM product WHERE id=1
```

Δηλαδή το πρότυπο είναι:

| Επιλογή πεδίων στο αποτέλεσμα<br>(τα πεδία onoma και kostos) | Επιλογή πηγής δεδομένων<br>(από τον πίνακα proion) | Επιλογή εγγραφών στο αποτέλεσμα<br>(ο κωδικός του να είναι 1) |
|---|---|---|
| SELECT name, cost | FROM proion | WHERE kodikos=1 |

Αν θέλουμε να πάρουμε όλα τα πεδία μιας εγγραφής έχουμε δύο επιλογές, είτε παραθέτουμε στο SELECT όλα τα πεδία που θέλουμε, είτε χρησιμοποιούμε τον ειδικό χαρακτήρα *. Για παράδειγμα προκειμένου να πάρουμε όλα τα πεδία της εγγραφής με τον κωδικό 1, θα γράψουμε την εντολή:

```
SELECT * FROM product WHERE id=1
```

Αν πάλι θέλουμε να πάρουμε όλα τα πεδία όλων των εγγραφών (ουσιαστικά να δούμε τα περιεχόμενα του πίνακα), τότε απλώς παραλείπουμε εντελώς το τμήμα WHERE, και η εντολή γίνεται:

```
SELECT * FROM product
```

Επίσης μπορούμε να μετονομάσουμε ένα πεδίο χρησιμοποιώντας τον προσδιορισμό AS. Για παράδειγμα αν θέλουμε να εμφανίσουμε το κόστος της πρώτης εγγραφής, αλλά το όνομα της στήλης αντί για cost να είναι timi, τότε θα γράψουμε την εντολή:

```
SELECT cost AS timi FROM proion WHERE id=1
```

Σε αυτή την περίπτωση το αποτέλεσμα που θα πάρουμε θα είναι:

| **product** | | | |
|---|---|---|---|
| **id** | **name** | **production** | **timi** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |

Βλέπουμε δηλαδή ότι το όνομα του πεδίου cost είναι πλέον timi. Λέμε ότι το νέο όνομα αποτελεί ένα *ψευδώνυμο* (alias) για το κανονικό όνομα.

## 3.2.2 Επιλογή εγγραφών με την WHERE

Όπως είδαμε στον προηγούμενο υπότιτλο μετά το WHERE, μπορούμε να ορίσουμε κριτήρια για την επιλογή των εγγραφών. Τα κριτήρια αυτά αφορούν αριθμητικές και λογικές συγκρίσεις, αναζήτηση σε χαρακτήρες, επιλογή από σύνολα τιμών κ.ά.

### 3.2.2.1 Λογικοί τελεστές και τελεστές σύγκρισης

Εάν θέλουμε να συγκρίνουμε πεδία τότε μπορούμε να χρησιμοποιήσουμε αριθμητικούς τελεστές για σύγκριση ( < , > , >= , <= ), ισότητα ( = ) και διαφορά ( <> ). Τα ακόλουθα παραδείγματα παρουσιάζουν ορισμένες περιπτώσεις:

Ποιες εγγραφές έχουν κόστος μεγαλύτερο ή ίσο με 30;

```
SELECT * FROM product WHERE cost>=30
```

Ποιες είναι οι εγγραφές που δεν έχουν τον κωδικό 1;

```
SELECT * FROM product WHERE id<>1
```

Επίσης μπορούμε να συνδυάζουμε κριτήρια με τους λογικούς τελεστές AND, OR, NOT. Για παράδειγμα:

Ποιες είναι οι εγγραφές που ο κωδικός τους είναι μεγαλύτερος από 1 και η ημερομηνία παραγωγής τους παλαιότερη από τον Ιούνιο του 2007;

```
SELECT * FROM product WHERE id>1 AND production<'2007-06-01'
```

Ποιες είναι οι εγγραφές με κωδικούς 1, 2, 3;

```
SELECT * FROM product WHERE id=1 OR id=2 OR id=3
```

Η SQL παρέχει επίσης μια διευκόλυνση όταν θέλουμε να ελέγξουμε αν ένα αριθμητικό πεδίο βρίσκεται μεταξύ δυο ακραίων τιμών. Για παράδειγμα έστω ότι θέλουμε να πάρουμε τα προϊόντα που το κόστος τους κυμαίνεται μεταξύ 20 και 50 ευρώ. Με βάση τα όσα έχουμε πει ως τώρα αυτό γίνεται ως:

```
SELECT * FROM product WHERE cost>=20 AND cost<=50
```

Ένας εναλλακτικός τρόπος για να πάρουμε το ίδιο αποτέλεσμα είναι χρησιμοποιώντας την εντολή `BETWEEN AND` ως εξής:

```
SELECT * FROM product WHERE cost BETWEEN 20 AND 50
```

Ποια είναι τα προϊόντα που δεν κατασκευάστηκαν μεταξύ της 1ης Ιουνίου και της 31ης Αυγούστου του 2007;

```
SELECT * FROM product WHERE production NOT BETWEEN '2007-06-01' AND '2007-08-31'
```

Τέλος μια πολύ σημαντική παρατήρηση αφορά τις συγκρίσεις με την τιμή NULL. Επειδή όπως έχουμε αναφέρει το NULL δεν υπάγεται σε κάποιο τύπο δεδομένων αλλά είναι ειδική περίπτωση, για αυτό όταν θέλουμε να ελέγξουμε αν μια τιμή είναι NULL βάζουμε `IS NULL` και όταν θέλουμε να ελέγξουμε αν μια τιμή δεν είναι NULL βάζουμε `IS NOT NULL`. Για παράδειγμα:

Ποιες εγγραφές δεν έχουν όνομα;

```
SELECT * FROM product WHERE name IS NULL
```

Ποιες εγγραφές έχουν όνομα;

```
SELECT * FROM product WHERE name IS NOT NULL
```

#### 3.2.2.2 Συγκρίσεις σε συμβολοσειρές

Μια άλλη κατηγορία συγκρίσεων αφορά συμβολοσειρές. Προφανώς μπορούμε να εφαρμόσουμε και στις συμβολοσειρές τους τελεστές σύγκρισης. Για παράδειγμα:

Ποιες είναι οι εγγραφές που το όνομά τους είναι «Ποντίκι Optical»;

```
SELECT * FROM product WHERE name='Ποντίκι Optical'
```

Επίσης μπορούμε να εφαρμόσουμε τους τελεστές > ή < με την έννοια της λεξικογραφικής σύγκρισης. Για παράδειγμα ας θεωρήσουμε τον ακόλουθο πίνακα δεδομένων:

| **Κατάλογος** | | |
|---|---|---|
| **Κωδικός** | **Επώνυμο** | **Όνομα** |
| 1 | ΑΒΡΟΣ | ΝΙΚΟΛΑΟΣ |

| | | |
|---|---|---|
| 2 | ΑΒΡΑΜΟΠΟΥΛΟΣ | ΙΩΑΝΝΗΣ |
| 3 | ΒΑΣΙΛΟΓΛΟΥ | ΙΩΑΝΝΗΣ |
| 4 | ΚΑΣΙΛΟΓΛΟΥ | ΝΙΚΟΛΑΟΣ |
| 5 | ΑΜΠΑΣΙΛΟΓΛΟΥ | ΧΡΗΣΤΟΣ |

Έστω ότι θέλουμε να δούμε ποια ονόματα ακολουθούν στον κατάλογο μετά το 'ΑΒ'

```
SELECT * FROM product WHERE name>'ΑΒΡΑΜ'
```

Οπότε τα αποτελέσματα που θα πάρουμε είναι:

| **Κατάλογος** | | |
|---|---|---|
| **Κωδικός** | **Επώνυμο** | **Όνομα** |
| 2 | ΑΒΡΑΜΟΠΟΥΛΟΣ | ΙΩΑΝΝΗΣ |
| 3 | ΒΑΣΙΛΟΓΛΟΥ | ΙΩΑΝΝΗΣ |
| 4 | ΚΑΣΙΛΟΓΛΟΥ | ΝΙΚΟΛΑΟΣ |
| 5 | ΑΜΠΑΣΙΛΟΓΛΟΥ | ΧΡΗΣΤΟΣ |

Όταν θέλουμε να ελέγξουμε αν μια συμβολοσειρά ταιριάζει με ένα *πρότυπο*, χρησιμοποιούμε τον τελεστή `LIKE` ακολουθούμενο από ένα αλφαριθμητικό που αποτελεί την σύγκριση. Στο αλφαριθμητικό σύγκρισης μπορούμε να γράφουμε είτε κανονικούς χαρακτήρες είτε τους ειδικούς χαρακτήρες _ και %. Το _ αντιστοιχεί σε ταίριασμα με έναν χαρακτήρα και το % σε ταίριασμα με περισσότερους χαρακτήρες. Προφανώς μπορούμε να κάνουμε και τον αρνητικό συνδυασμό `NOT LIKE`. Για παράδειγμα έστω ότι κάνουμε τις ακόλουθες ερωτήσεις με βάση τον κατάλογο των ονομάτων:

Ποιες είναι οι εγγραφές που το επώνυμό τους αρχίζει από 'ΑΒΡ';

```
SELECT * FROM catalog WHERE surname LIKE 'ΑΒΡ%'
```

Ποιες είναι οι εγγραφές που το επώνυμό τους καταλήγει σε «ΟΠΟΥΛΟΣ»;

```
SELECT * FROM catalog WHERE surname LIKE '%ΟΠΟΥΛΟΣ'΄
```

Ποιες είναι οι εγγραφές που έχουν οποιοδήποτε πρώτο χαρακτήρα αλλά μετά ακολουθεί η κατάληξη «ΑΣΙΛΟΓΛΟΥ»;

```
SELECT * FROM catalog WHERE surname LIKE '_ΑΣΙΛΟΓΛΟΥ'
```

Και το αποτέλεσμα είναι:

| **Κατάλογος** | | |
|---|---|---|
| **Κωδικός** | **Επώνυμο** | **Όνομα** |

| | | |
|---|---|---|
| 3 | ΒΑΣΙΛΟΓΛΟΥ | ΙΩΑΝΝΗΣ |
| 4 | ΚΑΣΙΛΟΓΛΟΥ | ΝΙΚΟΛΑΟΣ |

Παρατηρούμε πως η εγγραφή 5 δεν εμφανίζεται στο αποτέλεσμα, καθώς πριν το «ΑΣΙΛΟΓΛΟΥ» δεν υπάρχει ένας χαρακτήρας αλλά περισσότεροι.

Ένα άλλο ερώτημα θα μπορούσε να είναι: Ποιες είναι οι εγγραφές που το επώνυμό τους δεν έχει πουθενά το τμήμα «ΒΡ»;

```
SELECT * FROM catalog WHERE surname NOT LIKE '%BP%'
```

Οπότε στο αποτέλεσμα θα εμφανιστούν οι εγγραφές 3, 4, 5.

Επομένως βλέπουμε πως μπορούμε να συγκρίνουμε συμβολοσειρές με πρότυπα που ο χαρακτήρας _ αντικαθιστά έναν οποιονδήποτε άλλο και ο χαρακτήρας % αντικαθιστά ένα οσοδήποτε μεγάλο πλήθος χαρακτήρων.

### 3.2.3 Διάταξη των αποτελεσμάτων με ORDER BY

Ένα άλλο θέμα που μας ενδιαφέρει στην παρουσίαση των αποτελεσμάτων είναι η διάταξη με την οποία θα φαίνονται οι εγγραφές. Αυτό μπορούμε να το ελέγξουμε χρησιμοποιώντας την εντολή `ORDER BY`. Έτσι επεκτείνοντας τον ορισμό που έχουμε δώσει για την εντολή `SELECT`, έχουμε:

```
SELECT <πεδία που θέλουμε να φαίνονται>
FROM <πίνακες από τους οποίους θα αντληθούν τα δεδομένα>
WHERE <κριτήρια επιλογής των εγγραφών>
ORDER BY <πεδίο 1>, <πεδίο 2> …, <πεδίο ν>
```

Έστω λοιπόν η ερώτηση βασισμένη στον πίνακα των προϊόντων: Παρουσίασε τα προϊόντα ταξινομημένα από το φθηνότερο στο ακριβότερο

```
SELECT * FROM product ORDER BY cost
```

Αυτό θα δώσει το ακόλουθο αποτέλεσμα:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |
| 4 | Μνήμη 1GB | 2007-05-23 | 70.00 |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |
| 3 | CPU P4 3.2GHz | 2006-08-20 | 250.00 |

Ή και συνδυάζοντας όλα όσα έχουμε δει έως τώρα: «Παρουσιάσε τα ποντίκια από το νεότερο στο αρχαιότερο»

```
SELECT * FROM  product WHERE name LIKE 'Ποντίκι%' ORDER BY production
```

Βλέπουμε δηλαδή πως κάθε φορά συμπληρώνοντας στο τέλος το ORDER BY και επιλέγοντας το πεδίο ταξινόμησης, οι εγγραφές του αποτελέσματος ταξινομούνται βάσει του πεδίου ταξινόμησης από το μικρότερο στο μεγαλύτερο, δηλαδή με αύξουσα σειρά. Αν ωστόσο θέλουμε να ταξινομηθούν με φθίνουσα σειρά (από το μεγαλύτερο στο μικρότερο), τότε θα πρέπει στο τέλος του query να συμπληρώσουμε τον προσδιορισμό DESC (από το descending = φθίνων). Αν δεν συμπληρώσουμε τίποτα (όπως στα προηγούμενα παραδείγματα) εννοείται ο προσδιορισμός ASC (από το ascending = αύξων). Έτσι για παράδειγμα μπορούμε να ταξινομήσουμε τα προϊόντα από το ακριβότερο στο φθηνότερο με την ακόλουθη εντολή:

```
SELECT * FROM product ORDER BY cost DESC
```

Τέλος μπορούμε να ορίσουμε περισσότερα από ένα πεδία ταξινόμησης. Σε αυτή την περίπτωση τα αποτελέσματα θα παρουσιαστούν ταξινομημένα βάσει του πρώτου πεδίου και αν κάποιες εγγραφές έχουν ίδια τιμή σε αυτό το πεδίο τότε θα ταξινομηθούν με βάση το δεύτερο κ.ο.κ. Αν χρησιμοποιήσουμε περισσότερα από ένα πεδία τότε μπορούμε για καθένα από αυτά να ορίσουμε ανεξάρτητα αν η ταξινόμηση θα είναι αύξουσα ή φθίνουσα. Έστω για παράδειγμα η ακόλουθη ερώτηση:

```
SELECT * FROM catalog ORDER BY name ASC, surname DESC
```

Αυτό θα δώσει το ακόλουθο αποτέλεσμα:

| **Κατάλογος** | | |
|---|---|---|
| **Κωδικός** | **Επώνυμο** | **Όνομα** |
| 3 | ΒΑΣΙΛΟΓΛΟΥ | ΙΩΑΝΝΗΣ |
| 2 | ΑΒΡΑΜΟΠΟΥΛΟΣ | ΙΩΑΝΝΗΣ |
| 4 | ΚΑΣΙΛΟΓΛΟΥ | ΝΙΚΟΛΑΟΣ |
| 1 | ΑΒΡΟΣ | ΝΙΚΟΛΑΟΣ |
| 5 | ΑΜΠΑΣΙΛΟΓΛΟΥ | ΧΡΗΣΤΟΣ |

Βλέπουμε δηλαδή ότι οι εγγραφές έχουν αρχικά ταξινομηθεί με αύξουσα (λεξικογραφική) σειρά βάση του ονόματος. Κατόπιν για εκείνες τις εγγραφές που το όνομα είναι ίδιο ταξινομούνται βάσει του επωνύμου τους με φθίνουσα σειρά.

### 3.2.4 Συναρτήσεις στα πεδία της SELECT (count, min, max, avg, sum)

Έως τώρα έχουμε δει πως μπορούμε να επιλέξουμε υποπεριοχές ενός πίνακα ορίζοντας ποιες εγγραφές θέλουμε να επιστραφούν, ποια πεδία και με ποια σειρά. Ωστόσο υπάρχουν ερωτήσεις που δεν μπορούν να απαντηθούν με αυτές τις δυνατότητες μόνο. Για παράδειγμα:

«Πόσες είναι οι εγγραφές ενός πίνακα;»

Σε αυτή την περίπτωση δεν είναι δυνατόν να πάρουμε κάποια απάντηση με τις εντολές που έχουμε δει έως τώρα. Για αυτό η SQL δίνει τη δυνατότητα να εφαρμόζουμε συναρτήσεις στα αποτελέσματα των queries. Όπως έχουμε δει προηγουμένως μπορούμε να πάρουμε όλες τις εγγραφές του πίνακα με την εντολή:

```
SELECT * FROM product
```

Τώρα για να δούμε πόσες εγγραφές έχει ο πίνακας θα χρησιμοποιήσουμε την συνάρτηση count

```
SELECT count(*) FROM product
```

Οπότε το αποτέλεσμα που θα πάρουμε θα είναι ο πίνακας:

| product |
|---|
| Count(*) |
| 4 |

Με όμοιο τρόπο από με την count υπάρχουν οι συναρτήσεις min, max, avg, που υπολογίζουν αντίστοιχα το ελάχιστο, το μέγιστο και τον μέσο όρο μιας στήλης. Ας δούμε για παράδειγμα τι αποτέλεσμα θα επέστρεφαν τα ακόλουθα queries

```
SELECT min(cost) FROM product

SELECT max(cost) FROM product

SELECT avg(cost) FROM product

SELECT sum(cost) FROM product
```

| product |
|---|
| Min(cost) |
| 23.50 |

| product |
|---|
| Max(cost) |
| 250.00 |

| Product |
|---|
| Avg(cost) |
| 107.125 |

| Prduct |
|---|
| Sum(cost) |
| 428.50 |

Έτσι το min επιστρέφει την μικρότερη τιμή του πεδίου κόστος, το max την μεγαλύτερη, το avg την μέση τιμή και το sum το άθροισμα των τιμών.

Εννοείται βέβαια ότι μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις count, max, min, avg, sum με οτιδήποτε άλλο έχουμε μάθει σε σχέση με την επιλογή των εγγραφών. Για παράδειγμα έστω το ακόλουθο ερώτημα:

Πόσα προϊόντα έχουν κατασκευαστεί εντός του 2006;

```
SELECT COUNT(*) FROM product WHERE production>='2006-01-01' AND production<='2006-12-31'
```

Με την παραπάνω εντολή επιλέγουμε τα προϊόντα που η ημερομηνία παραγωγής τους είναι εντός της 1ης Ιανουαρίου του 2006 και της 31ης Δεκεμβρίου του 2006 (άρα εντός του έτους) και εφαρμόζοντας τη συνάρτηση count στο select παίρνουμε, όχι τις εγγραφές, αλλά το πλήθος τους. Ομοίως θα μπορούσαμε να βρούμε το ακριβότερο προϊόν που κατασκευάστηκε από την αρχή του 2007 και μετά με την εντολή:

```
SELECT max(cost) FROM product WHERE production>='2007-01-01'
```

Οπότε συνοψίζοντας μπορούμε να εφαρμόσουμε τις ακόλουθες συναρτήσεις στα πεδία του select:

| **Όνομα συνάρτησης** | **Λειτουργία** |
|---|---|
| Count(<πεδίο>) | Καταμέτρηση των εγγραφών που επιστρέφει το query. |
| Max(<πεδίο>) | Εφαρμόζεται επί ενός πεδίου και επιστρέφει την μεγαλύτερη τιμή του πεδίου από τις εγγραφές που επιλέγει το query. |
| Min(<πεδίο>) | Εφαρμόζεται επί ενός πεδίου και επιστρέφει την μικρότερη τιμή του πεδίου από τις εγγραφές που επιλέγει το query. |
| Avg(<πεδίο>) | Εφαρμόζεται επί ενός αριθμητικού πεδίου και επιστρέφει την μέσο όρο των τιμών του πεδίου από τις εγγραφές που επιλέγει το query. |
| Sum(<πεδίο>) | Εφαρμόζεται επί ενός αριθμητικού πεδίου και επιστρέφει το άθροισμα των τιμών του πεδίου από τις εγγραφές που επιλέγει το query. |

Ένα ενδιαφέρον συμπέρασμα που προκύπτει από τα όσα αναφέραμε παραπάνω είναι πως οι παραπάνω συναρτήσεις ενώ εφαρμόζονται σε ένα σύνολο από εγγραφές, επιστρέφουν *ένα* αποτέλεσμα (π.χ. το μεγαλύτερο κόστος)!

Επίσης ας προσέξουμε ότι μέσα στο count δεν είναι υποχρεωτικό να βάζουμε το * αλλά μπορούμε να βάλουμε το όνομα οποιουδήποτε πεδίου του πίνακα. Για παράδειγμα έστω ο ακόλουθος πίνακας:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |

| | | | |
|---|---|---|---|
| 3 | CPU P4 3.2GHz | 2006-08-20 | 250.00 |
| 4 | *NULL* | 2005-12-23 | 70.00 |

Παρατηρούμε δηλαδή ότι έχουμε διαγράψει το όνομα της εγγραφής 4. Και έστω λοιπόν ότι έχουμε τις ακόλουθες 2 εντολές:

```
SELECT count(*) FROM product
```

```
SELECT count(name) FROM product
```

Η πρώτη εντολή θα μετρήσει το πλήθος των εγγραφών και θα επιστρέψει 4. Η δεύτερη θα μετρήσει το πλήθος των εμφανίσεων του πεδίου name στον πίνακα οπότε θα επιστρέψει 3 (γιατί η τελευταία είναι κενή)!

Τέλος πρέπει να αναφέρουμε πως αυτές είναι μόνο λίγες από τις πολλές συναρτήσεις που συνήθως ορίζονται στα ΣΔΒΔ. Για παράδειγμα στην MySQL υπάρχουν εξειδικευμένες συναρτήσεις για τον υπολογισμό μαθηματικών συναρτήσεων, συναρτήσεις πάνω σε strings κ.ά.

### 3.2.5 Απαλοιφή διπλοεγγραφών με DISTINCT

Ας επιστρέψουμε πάλι στον πίνακα με τον κατάλογο των ονομάτων, όπως αυτός φαίνεται παρακάτω:

| **Κατάλογος** | | |
|---|---|---|
| **Κωδικός** | **Επώνυμο** | **Όνομα** |
| 1 | ΑΒΡΟΣ | ΝΙΚΟΛΑΟΣ |
| 2 | ΑΒΡΑΜΟΠΟΥΛΟΣ | ΙΩΑΝΝΗΣ |
| 3 | ΒΑΣΙΛΟΓΛΟΥ | ΙΩΑΝΝΗΣ |
| 4 | ΚΑΣΙΛΟΓΛΟΥ | ΝΙΚΟΛΑΟΣ |
| 5 | ΑΜΠΑΣΙΛΟΓΛΟΥ | ΧΡΗΣΤΟΣ |

Και έστω ότι θέλουμε να απαντήσουμε στο ερώτημα:

Ποια είναι τα ονόματα των εγγραφών του πίνακα; Μια λογική εντολή θα ήταν η :

```
SELECT name FROM catalog
```

Ωστόσο αυτή η εντολή θα επέστρεφε το αποτέλεσμα:

| **Catalog** |
|---|
| **Name** |
| ΝΙΚΟΛΑΟΣ |

| |
|---|
| ΙΩΑΝΝΗΣ |
| ΝΙΚΟΛΑΟΣ |
| ΙΩΑΝΝΗΣ |
| ΧΡΗΣΤΟΣ |

Βλέπουμε δηλαδή πως αν και όντως επέστεψε τα ονόματα, επειδή ήταν διπλογραμμένα τα επέστρεψε πολλές φορές το καθένα. Αν όμως θέλουμε απλά να πάρουμε τα *μοναδικά* ονόματα, τότε πρέπει να χρησιμοποιήσουμε τον τελεστή `DISTINCT` (μοναδικό) πριν την δήλωση των πεδίων στο SELECT:

```
SELECT DISTINCT name FROM catalog
```

Οπότε με αυτή την εντολή θα πάρουμε το ακόλουθο αποτέλεσμα:

| **catalog** |
|---|
| **Name** |
| ΝΙΚΟΛΑΟΣ |
| ΙΩΑΝΝΗΣ |
| ΧΡΗΣΤΟΣ |

Επομένως όταν θέλουμε να εγγυηθούμε ότι στο αποτέλεσμα του query δεν θα υπάρχουν διπλοεγγραφές τότε πρέπει να χρησιμοποιούμε το DISTINCT.

Τέλος μπορούμε να χρησιμοποιήσουμε το DISTINCT και με έναν διαφορετικό τρόπο. Για παράδειγμα έστω η ερώτηση:

“Πόσα διαφορετικά ονόματα εμφανίζονται στον κατάλογο;”

Προσέξτε ότι η εντολή

```
SELECT count(*) FROM catalog
```

Θα επέστρεφε το 5, εφόσον 5 ονόματα υπάρχουν. Όταν όμως θέλουμε να μάθουμε πόσα διαφορετικά ονόματα υπάρχουν τότε πρέπει να βάλουμε το DISTINCT μέσα στο count όπως φαίνεται στην ακόλουθη εντολή:

```
SELECT count(DISTINCT name) FROM catalog
```

Θυμόμαστε από τον προηγούμενο υπότιτλο πως δίνοντας ένα όρισμα μέσα στις παρενθέσεις της count, αυτή θα μετρήσει αυτό ακριβώς το πεδίο. Στην προκειμένη περίπτωση ζητούμε να μετρήσει τα διαφορετικά ονόματα. Έτσι θα επιστρέψει 3 που είναι το σωστό αποτέλεσμα.

### 3.2.6 Ομαδοποίηση των αποτελεσμάτων με GROUP BY

Χρησιμοποιώντας πάλι τα δεδομένα του καταλόγου των ονομάτων έστω ότι θέλουμε να απαντήσουμε στο ακόλουθο ερώτημα:

Πόσες φορές εμφανίζεται κάθε όνομα στον κατάλογο;

Στον προηγούμενο υπότιτλο είδαμε πως με το DISTINCT μπορούμε να απαλείφουμε τις διπλοεγγραφές, και πως μπορούμε να μετρήσουμε τα διαφορετικά ονόματα του πίνακα. Τώρα όμως θέλουμε να μετρήσουμε πόσες φορές εμφανίζεται το κάθε όνομα. Για αυτό θα χρησιμοποιήσουμε την εντολή GROUP BY (δηλ. ομαδοποίησε σύμφωνα με):

```
SELECT name, count(*) FROM catalog GROUP BY name
```

Προσθέτοντας το GROUP BY στο τέλος της εντολής, το ΣΔΒΔ προχωράει σε μια περαιτέρω ομαδοποίηση των αποτελεσμάτων σύμφωνα με το πεδίο που δίνουμε (το όνομα στην προκειμένη περίπτωση). Οπότε δημιουργούνται 3 ομάδες εγγραφών, όπως φαίνονται στο ακόλουθο σχήμα.

| **Κατάλογος** | | |
|---|---|---|
| **Κωδικός** | **Επώνυμο** | **Όνομα** |
| 2 | ΑΒΡΑΜΟΠΟΥΛΟΣ | ΙΩΑΝΝΗΣ |
| 3 | ΒΑΣΙΛΟΓΛΟΥ | ΙΩΑΝΝΗΣ |
| 1 | ΑΒΡΟΣ | ΝΙΚΟΛΑΟΣ |
| 4 | ΚΑΣΙΛΟΓΛΟΥ | ΝΙΚΟΛΑΟΣ |
| 5 | ΑΜΠΑΣΙΛΟΓΛΟΥ | ΧΡΗΣΤΟΣ |

Το πλεονέκτημα της δημιουργίας εσωτερικών ομάδων στο αποτέλεσμα του query είναι ότι οι συναρτήσεις του SELECT εφαρμόζονται πλέον εσωτερικά στην κάθε ομάδα και όχι στο γενικό σύνολο. Έτσι όταν γράφουμε count(*), αυτό μετράει το πλήθος των εγγραφών της κάθε ομάδας. Άρα το αποτέλεσμα της παραπάνω εντολής θα είναι:

| **catalog** | |
|---|---|
| **Name** | **Count(*)** |
| ΙΩΑΝΝΗΣ | 2 |
| ΝΙΚΟΛΑΟΣ | 2 |
| ΧΡΗΣΤΟΣ | 1 |

Δηλαδή για κάθε ομάδα που έχει δημιουργηθεί από το GROUP BY ζητάμε το όνομα και το πλήθος των εγγραφών της. Κατά συνέπεια απαντήσαμε στην ερώτηση που είχαμε!

### 3.2.7 Χρησιμοποιώντας την SELECT ως εντολή προβολής

Τέλος σε αυτόν τον υπότιτλο θέλουμε να δούμε την εντολή SELECT από μια άλλη οπτική γωνία. Μπορούμε να χρησιμοποιούμε την SELECT ως μια εντολή προβολής, με όμοιο τρόπο που θα χρησιμοποιούσαμε μια εντολή print σε μια γλώσσα προγραμματισμού. Για παράδειγμα αν θέλουμε να παρουσιάσουμε το μήνυμα «Καλημέρα κόσμε», θα αρκούσε η εντολή:

```
SELECT "Καλημέρα κόσμε"
```

Επίσης μπορούμε να εφαρμόζουμε αριθμητικές και άλλες πράξεις στα πεδία ενός πίνακα. Για παράδειγμα έστω η ακόλουθη ερώτηση:

Ποιες είναι οι τιμές των προϊόντων αν κάνουμε μια αύξηση 5%

```
SELECT name, cost*1.05 AS newCost FROM product
```

Δηλαδή πολλαπλασιάζουμε το κόστος του κάθε προϊόντος με τον συντελεστή 1.05 (για να πετύχουμε αύξηση 5%), οπότε θα πάρουμε το ακόλουθο αποτέλεσμα:

| **product** | |
|---|---|
| **Name** | **newCost** |
| Ποντίκι optical | 24.675 |
| Σκληρός Δίσκος 300GB | 89.25 |
| CPU P4 3.2GHz | 262.50 |
| Μνήμη 1GB | 73.50 |

Σε αυτό το παράδειγμα κάναμε χρήση και των ψευδωνύμων χρησιμοποιώντας το AS. Έτσι ονομάσαμε το γινόμενο cost*1.05 ως newCost. Για αυτό και η δεύτερη στήλη στον πίνακα του αποτελέσματος έχει αυτό το όνομα.

Επίσης θα μπορούσαμε να εμφανίσουμε ως αποτέλεσμα το αποτέλεσμα μιας λογικής (ή άλλης σύγκρισης). Για παράδειγμα έστω ότι θέλουμε να απαντήσουμε στην ακόλουθη ερώτηση:

«Αν στη βάση υπάρχουν περισσότερα από 2 προϊόντα με κόστος μικρότερο των 100 ευρώ, εμφάνισε 1 διαφορετικά εμφάνισε 0»

```
SELECT count(*)>2 FROM product WHERE cost<100
```

Εδώ η λογική είναι ότι πρώτα κάνουμε μια καταμέτρηση για το πλήθος των προϊόντων που ικανοποιούν το κριτήριο αναζήτησης (το κόστος μικρότερο των 100 ευρώ). Κατόπιν κάνουμε μια σύγκριση της μέτρησης με το 2, οπότε το αποτέλεσμα θα είναι 0 αν είναι ψευδές και 1 αν είναι αληθές.

## 3.3 Ενημέρωση – διαγραφή δεδομένων

### 3.3.1 Εντολή UPDATE

Έχοντας ήδη ενθέσει μια εγγραφή στον πίνακα μπορούμε να αλλάξουμε (πχ. να διορθώσουμε) την τιμή ενός πεδίου χρησιμοποιώντας την εντολή UPDATE. Το γενικό πρότυπο της εντολής είναι:

```
UPDATE <όνομα πίνακα> SET <όνομα πεδίου>=<νέα τιμή πεδίου> WHERE <κριτήρια επιλογής εγγραφών>
```

Για παράδειγμα αν θέλουμε στον πίνακα των προϊόντων να θέσουμε την τιμή του επεξεργαστή στα 200 ευρώ, τότε θα γράψουμε

```
UPDATE product SET cost=200.00 WHERE id=3
```

Επίσης έχουμε τη δυνατότητα να αλλάξουμε ταυτόχρονα πολλά πεδία ανά εγγραφή παραθέτοντας τις νέες τιμές των πεδίων της διαχωρισμένες με κόμμα. Για παράδειγμα αν θέλουμε να αλλάξουμε και την τιμή και την ημερομηνία κατασκευής, μπορούμε να γράψουμε:

```
UPDATE product SET cost=200.00, production='2007-10-01' WHERE id=3
```

Είναι σημαντικό στην εντολή UPDATE να ορίσουμε προσεκτικά τα κριτήρια επιλογής των εγγραφών. Στα δύο προηγούμενα παραδείγματα χρησιμοποιούμε το πεδίο id του πίνακα που είναι κλειδί και έτσι βεβαιωνόμαστε ότι η αλλαγή θα γίνει στην συγκεκριμένη γραμμή. Ωστόσο θα μπορούσαμε να ορίσουμε κάτι πιο σύνθετο. Για παράδειγμα έστω η ακόλουθη εντολή:

```
UPDATE product SET cost=cost*1.05 WHERE production>'2007-01-01'
```

Με την εντολή αυτή ορίζουμε ότι για τα προϊόντα που έχουν κατασκευαστεί μετά το 2007, θα κάνουμε μια αύξηση στην τιμή τους κατά 5% (δηλ. το cost=cost*1.05). Τέλος να σημειώσουμε ότι στην περίπτωση που δεν ορίσουμε μια πρόταση WHERE τότε οι αλλαγές των πεδίων θα εφαρμοστούν σε όλες τις εγγραφές του πίνακα. Για παράδειγμα αν θέλουμε να μειώσουμε τις τιμές όλων των προϊόντων κατά 20% (π.χ. επειδή έχουμε γενικές εκπτώσεις), θα γράψουμε την εντολή:

```
UPDATE product SET cost=cost*0.80
```

Εδώ ορίζουμε το κάθε προϊόν να πωλείται στο 80% της αρχικής του τιμής.

### 3.3.2 Εντολή DELETE

Παρόμοια με την εντολή UPDATE λειτουργεί και η εντολή DELETE. Ο σκοπός της είναι πολύ απλός, διαγράφει εγγραφές από έναν πίνακα. Και πάλι έχουμε τη δυνατότητα να ορίσουμε ποιες εγγραφές θέλουμε να διαγραφούν (ή και όλες). Έστω για παράδειγμα η ακόλουθη εντολή:

```
DELETE FROM product WHERE id=1
```

Η παραπάνω εντολή απλά διαγράφει την εγγραφή με id=1, οπότε ο πίνακας product θα έχει τα ακόλουθα δεδομένα:

| Προϊόν | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |
| 3 | CPU P4 3.2GHz | 2006-08-20 | 250.00 |
| 4 | Μνήμη 1GB | 2005-12-23 | 70.00 |

Αν πάλι θέλουμε να κάνουμε μια εκκαθάριση της βάσης μας από παλιά προϊόντα με ημερομηνία κατασκευής πριν το 2007, θα μπορούσαμε να γράψουμε την εντολή:

```
DELETE FROM product WHERE production<'2007-01-01'
```

Και τέλος αν θέλουμε να διαγράψουμε όλα τα δεδομένα του πίνακα product, γράφουμε την εντολή:

```
DELETE FROM product
```

Είναι ενδιαφέρον να σημειώσουμε ότι η εντολή DELETE επηρεάζει μόνο τα δεδομένα ενός πίνακα και όχι τη δομή του. Δηλαδή ακόμα και μετά την εκτέλεση της τελευταίας εντολής ο πίνακας product όπως τον έχουμε ορίσει με τα πεδία του θα εξακολουθεί να υπάρχει, απλά θα είναι άδειος.

## 3.4 Περιορισμός των αποτελεσμάτων με την LIMIT της MySQL

Σε αυτόν τον υπότιτλο θα εξετάσουμε μια ενδιαφέρουσα εντολή της MySQL, που όμως δεν αποτελεί μέρος της πρότυπης SQL. Έτσι δεν μπορούμε να εγγυηθούμε την χρήση της σε άλλα ΣΔΒΔ, μολονότι σχεδόν όλα τα ΣΔΒΔ έχουν κάποια αντίστοιχη εντολή. Ας υποθέσουμε ότι θέλουμε να απαντήσουμε στην ακόλουθη ερώτηση:

«Από τον πίνακα των προϊόντων, εμφάνισε τα στοιχεία των δύο φθηνότερων προϊόντων»

Προσέχουμε ότι η παραπάνω ερώτηση θέτει έναν περιορισμό στο πλήθος των στοιχείων που θέλουμε να εμφανίσουμε. Αν θέλαμε να εμφανίσουμε όλα τα προϊόντα ταξινομημένα από το φθηνότερο στο ακριβότερο θα χρησιμοποιούσαμε την ακόλουθη εντολή:

```
SELECT * FROM product ORDER BY cost
```

Για να περιορίσουμε τις εγγραφές του αποτελέσματος στις δύο πρώτες θα γράψουμε:

```
SELECT * FROM product ORDER BY cost LIMIT 2
```

Ο προσδιορισμός LIMIT τοποθετείται στο τέλος του query και ορίζει το πλήθος των αποτελεσμάτων που θέλουμε να εμφανιστούν (στην προκειμένη περίπτωση 2) και προφανώς μπορούμε να το συνδυάσουμε με όσα έχουμε μάθει έως τώρα. Αν ο αριθμός που θα ορίσουμε είναι μεγαλύτερος από το σύνολο των εγγραφών, τότε απλά θα εμφανιστούν όλες οι εγγραφές.

# 4 Ανάκτηση δεδομένων από περισσότερους πίνακες

## 4.1 Συσχετισμοί πινάκων

Έως τώρα έχουμε εξετάσει διάφορες εντολές SQL που αφορούν τον ορισμό ενός πίνακα και τον χειρισμό των δεδομένων του (ένθεση, ενημέρωση, ανάκτηση και διαγραφή). Ωστόσο η πραγματική δύναμη των βάσεων δεδομένων οφείλεται στην ικανότητά τους να συσχετίζουν δεδομένα από περισσότερους πίνακες και να τα παρουσιάζουν ενοποιημένα σε ένα αποτέλεσμα.

Όταν στη βάση μας έχουμε περισσότερους από έναν πίνακες (δηλαδή πάντα), απαιτείται να σχεδιάσουμε σωστά πως αυτοί θα συσχετίζονται μεταξύ τους. Για παράδειγμα έστω ότι έχουμε το ακόλουθο σενάριο:

«Ένα ΙΕΚ αποτελείται από καθηγητές, μαθητές και μαθήματα και το κάθε μάθημα υπάγεται σε ένα εξάμηνο. Ο κάθε καθηγητής μπορεί να διδάσκει περισσότερα από ένα μαθήματα, ενώ το κάθε μάθημα μπορεί να διδάσκεται από περισσότερους από έναν καθηγητές».

Στους επόμενους υπότιτλους θα δούμε τα βήματα που θα ακολουθήσουμε για να σχεδιάσουμε τη δομή μιας βάσης που να μοντελοποιεί το παραπάνω πρόβλημα. Ωστόσο στη συνέχεια θα εξετάσουμε μερικές βασικές σχεδιαστικές αρχές στις συσχετίσεις πινάκων.

### 4.1.1 Σχεδιάσεις με διαφορετικούς βαθμούς πληθικότητας

Έστω ότι έχουμε έναν πίνακα για τον καθηγητή και έναν για το μάθημα όπως φαίνεται στο ακόλουθο σχήμα. Τα υπογραμμισμένα πεδία είναι κλειδί στον πίνακα:

| **Καθηγητής** | | | |
|---|---|---|---|
| Αρ. Ταυτότητας | Όνομα | Επώνυμο | Ειδικότητα |

| **Μάθημα** | | | |
|---|---|---|---|
| Κωδικός | Τίτλος | Τύπος | Περιγραφή |

Άσχετα με την επιλογή των πεδίων του κάθε πίνακα, αυτό που μας ενδιαφέρει εδώ είναι πως θα συνδέσουμε τον καθηγητή με το μάθημα. Αυτό εξαρτάται από τον *βαθμό συσχέτισης* που έχουν μεταξύ τους οι πίνακες[*]. Συγκεκριμένα υπάρχουν οι εξής δυνατότητες:

| **Συσχέτιση πινάκων** | **Πληθικότητα** |
|---|---|
| Ένας καθηγητής διδάσκει ένα μάθημα | 1:1 |
| Ένας καθηγητής διδάσκει πολλά μαθήματα | 1:N |
| Πολλοί καθηγητές διδάσκουν πολλά μαθήματα | M:N |

Είναι θεμελιώδες να κατανοούμε τον τρόπο που συσχετίζονται οι πίνακες μεταξύ τους γιατί αυτό καθορίζει δραστικά τον τρόπο που θα σχεδιάσουμε την βάση μας. Στη συνέχεια θα δούμε πως και γιατί πρέπει να σχεδιάζουμε τη βάση ανάλογα με τον βαθμό πληθικότητας που έχουμε.

#### 4.1.1.1 Συσχέτιση 1:1

Όταν η συσχέτιση είναι 1:1, τότε κάθε εγγραφή του ενός πίνακα συσχετίζεται με μια εγγραφή του άλλου πίνακα. Για παράδειγμα έστω ότι οι πίνακες έχουν τα ακόλουθα δεδομένα:

| **teacher** | | | |
|---|---|---|---|
| **ar_tautotitas** | **name** | **surname** | **speciality** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS |
| P458652 | STATHIS | KIOSSES | FILOLOGOS |

[*] Πολλές φορές ο βαθμός συσχέτισης των πινάκων ονομάζεται και πληθικότητα (στα αγγλικά cardinality).

| | | | |
|---|---|---|---|
| R586323 | MARIA | IWANNOY | PLHROFORIKOS |

| **subject** | | |
|---|---|---|
| **id** | **title** | **Type** |
| 1 | DataBases | THEORY |
| 2 | MySQL | LAB |
| 3 | C | THEORY |
| 4 | English | THEORY |

Όταν ένας καθηγητής διδάσκει αποκλειστικά ένα μάθημα και κάθε μάθημα διδάσκεται αποκλειστικά από έναν καθηγητή τότε το μόνο που χρειάζεται είναι να προσθέσουμε στον ένα από τους δύο πίνακες ένα ακόμη πεδίο που θα αναφέρεται στο κλειδί του άλλου πίνακα. Για παράδειγμα μπορούμε να προσθέσουμε στον καθηγητή ένα πεδίο που θα αφορά το id του μαθήματος που θα διδάσκει, οπότε θα έχουμε το ακόλουθο σχήμα:

| **teacher** | | | | |
|---|---|---|---|---|
| **ar_tautotitas** | **name** | **surname** | **speciality** | **Subject_id** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS | 1 |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS | 2 |
| P458652 | STATHIS | KIOSSES | FILOLOGOS | 4 |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS | 3 |

Με αυτόν τον τρόπο ορίζουμε ποιο μάθημα διδάσκει ο κάθε καθηγητής. Εναλλακτικά θα μπορούσαμε να προσθέσουμε στον πίνακα των μαθημάτων ένα πεδίο που να ορίζει το κλειδί του καθηγητή που το διδάσκει. Το πεδίο subject_id του πίνακα teacher που αναφέρεται στο κλειδί του πίνακα subject, για τον teacher είναι **ξένο κλειδί** (foreign key). Δηλαδή ξένο κλειδί χαρακτηρίζεται το πεδίο ενός πίνακα που είναι κλειδί για έναν άλλο πίνακα.

Επομένως ο κανόνας είναι ότι όταν η συσχέτιση των πινάκων είναι 1:1 τότε το κλειδί το ενός πίνακα θα γίνει ξένο κλειδί στον άλλον πίνακα. Και μπορούμε να επιλέξουμε όποιον από τους δύο πίνακες θέλουμε για να βάλουμε το κλειδί του ξένο κλειδί στον άλλο πίνακα. Σε επόμενο υπότιτλο θα δούμε πως ορίζεται το ξένο κλειδί σε έναν πίνακα.

#### 4.1.1.2 Συσχέτιση 1:N

Η δεύτερη περίπτωση συσχέτισης πινάκων είναι όταν μια εγγραφή του πρώτου πίνακα αντιστοιχεί σε πολλές εγγραφές του άλλου πίνακα. Σύμφωνα με το παράδειγμά μας ας υποθέσουμε ότι επιτρέπεται ένας καθηγητής να διδάσκει πολλά μαθήματα αλλά κάθε μάθημα να διδάσκεται αποκλειστικά από έναν καθηγητή.

Σε αυτήν την περίπτωση δεν έχουμε επιλογές, πρέπει να βάλουμε το κλειδί του καθηγητή ως ξένο κλειδί στο μάθημα. Δηλαδή γενικεύοντας όταν έχουμε δύο πίνακες Α και Β και κάθε εγγραφή του Α αντιστοιχεί σε πολλές εγγραφές του Β, τότε βάζουμε το κλειδί του Α ως ξένο κλειδί στον Β. Βάσει αυτού του κανόνα ο Α είναι ο teacher και ο Β ο subject, οπότε η τελική μορφή των πινάκων θα πρέπει να είναι:

| **teacher** | | | |
|---|---|---|---|
| **ar_tautotitas** | **name** | **surname** | **speciality** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS |
| P458652 | STATHIS | KIOSSES | FILOLOGOS |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS |

| **subject** | | | |
|---|---|---|---|
| **id** | **title** | **Type** | **tar_tautotitas** |
| 1 | DataBases | THEORY | X256526 |
| 2 | MySQL | LAB | X256526 |
| 3 | C | THEORY | R586323 |
| 4 | English | THEORY | P458652 |

Δηλαδή έχουμε ορίσει τον X256526 να διδάσκει τα μαθήματα 1 και 2, τον R586323 να διδάσκει το μάθημα 3 και τον P458652 να διδάσκει το μάθημα 4, ενώ ο R589643 δεν διδάσκει τίποτα. Με αυτή τη σχεδίαση έχουμε τον ελάχιστο αριθμό επαναλήψεων πληροφορίας.

Βλέπουμε ότι η μόνη επανάληψη αφορά το πεδίο ar_tautotitas του πίνακα subject και αυτό στην περίπτωση που ένας καθηγητής διδάσκει περισσότερα από ένα μαθήματα (όπως συμβαίνει δηλαδή με τις δύο πρώτες εγγραφές). Προσέξτε πως αν είχαμε επιλέξει να θέσουμε το κλειδί του subject ως ξένο κλειδί στον teacher τότε θα είχαμε πάρα πολλές επαναλήψεις, δοκιμάζοντας βλέπουμε ότι θα επαναλαμβάνονταν όλα τα πεδία εκτός από ένα, ενώ τώρα επαναλαμβάνεται μόνο ένα πεδίο.

#### 4.1.1.3 Συσχέτιση M:N

Η τρίτη και τελευταία περίπτωση συσχέτισης είναι όταν πολλές εγγραφές του ενός πίνακα συσχετίζονται με πολλές εγγραφές του άλλου πίνακα. Η περίπτωση αυτή είναι ιδιόμορφη γιατί απαιτεί την προσθήκη ενός επιπλέον πίνακα που θα συσχετίζει τους δύο βασικούς. Αν λοιπόν θεωρήσουμε πάλι ότι έχουμε τους πίνακες Α και Β όπου πολλές εγγραφές τους ενός συσχετίζονται με εγγραφές του άλλου, τότε δημιουργούμε έναν τρίτο πίνακα (έστω Γ) και βάζουμε τα κλειδιά των πινάκων Α και Β ως πεδία στον πίνακα Γ.

Επιστρέφοντας στο παράδειγμά μας, θεωρούμε τώρα την γενική περίπτωση όπου ένας καθηγητής μπορεί να διδάσκει πολλά μαθήματα και κάθε μάθημα να διδάσκεται από πολλούς καθηγητές. Σε αυτή την περίπτωση αφήνουμε τους πίνακες teacher και subject όπως είναι και προσθέτουμε έναν τρίτο πίνακα (έστω assignments δηλ. αναθέσεις), όπως φαίνεται στο ακόλουθο σχήμα:

| **Assignments** | |
|---|---|
| **ar_tautotitas** | **Subject_id** |
| X256526 | 1 |
| X256526 | 2 |
| R586323 | 3 |
| P458652 | 4 |
| R589643 | 1 |

Οπότε και τα δύο πεδία του πίνακα είναι ξένα κλειδιά, το ένα (ar_tautotitas) προς τον πίνακα teacher και το άλλο (subject_id) προς τον πίνακα subject. Με βάση τα δεδομένα του πίνακα assignments οι αναθέσεις μαθημάτων περιλαμβάνουν την συνδιδασκαλία του μαθήματος 1 από τους X256526 και R589643 ενώ τα υπόλοιπα δεδομένα είναι όπως και πριν. Είναι ενδιαφέρον να σημειώσουμε πως στον πίνακα assignments έχουμε τη δυνατότητα να προσθέσουμε και άλλα πεδία αν αυτό είναι απαραίτητο. Θα μπορούσαμε για παράδειγμα να προσθέσουμε ένα ακόμη πεδίο που να ορίζει την ημερομηνία της ανάθεσης, οπότε ο πίνακας assignments θα είχε την μορφή:

| **Assignments** | | |
|---|---|---|
| **ar_tautotitas** | **Subject_id** | **Assign_date** |
| X256526 | 1 | 2007-10-01 |
| X256526 | 2 | 2007-10-05 |
| R586323 | 3 | 2007-10-05 |
| P458652 | 4 | 2007-10-04 |
| R589643 | 1 | 2007-10-01 |

Τέλος να σημειώσουμε ότι όσον αφορά τον πίνακα assignments το κλειδί του είναι ο συνδυασμός των ξένων κλειδιών του, δηλαδή το (ar_tautotitas, subject_id). Ωστόσο θα μπορούσαμε αν θέλουμε να εισάγουμε και ένα ακόμη πεδίο που να λειτουργεί ως πρωτεύον κλειδί στον πίνακα assignments, σε αυτή την περίπτωση ο πίνακας θα γινόταν:

| **Assignments** | | | |
|---|---|---|---|
| **id** | **ar_tautotitas** | **Subject_id** | **Assign_date** |
| 1 | X256526 | 1 | 2007-10-01 |
| 2 | X256526 | 2 | 2007-10-05 |
| 3 | R586323 | 3 | 2007-10-05 |
| 4 | P458652 | 4 | 2007-10-04 |
| 5 | R589643 | 1 | 2007-10-01 |

Οπότε το πεδίο id έχει την έννοια του «κωδικού ανάθεσης μαθήματος». Ωστόσο στις περισσότερες περιπτώσεις η απλή εκδοχή του πίνακα assignments με τα δύο ξένα κλειδιά αρκεί για την σχεδίαση της βάσης.

## 4.1.2 Δημιουργία ξένων κλειδιών κατά τη δημιουργία πινάκων

Έχοντας εξετάσει τι είναι ένα ξένο κλειδί και πότε το τοποθετούμε σε έναν πίνακα, θα δούμε τώρα τις εντολές SQL που μας επιτρέπουν να εισαγάγουμε ένα ξένο κλειδί σε ένα πίνακα. Η δημιουργία ενός ξένου κλειδιού γίνεται (όπως και για το πρωτεύον κλειδί) στη φάση της αρχικής δημιουργίας του πίνακα. Για παράδειγμα αν θέλουμε να ορίσουμε ότι ο πίνακας subject θα έχει ως ξένο κλειδί το πεδίο tar_tautotitas, για το αντίστοιχο πεδίο ar_tautotitas του πίνακα teacher, τότε οι εντολές CREATE για την δημιουργία των δύο πινάκων, θα είναι ως εξής:

```
CREATE TABLE teacher(
ar_tautotitas CHAR(7) NOT NULL,
name VARCHAR(10),
surname VARCHAR(20),
speciality VARCHAR(20),
PRIMARY KEY(ar_tautotitas)
);

CREATE TABLE subject(
```

```
id INT NOT NULL,
        title VARCHAR(20),
        type ENUM('THEORY','LAB'),
        tar_tautotitas CHAR(7),
        PRIMARY KEY(id),
        FOREIGN KEY (tar_tautotitas) REFERENCES teacher(ar_tautotitas)
);
```

Μέσα στο πλαίσιο βλέπουμε τον τρόπο που ορίζουμε ότι το πεδίο tar_tautotitas του πίνακα subject θα είναι ξένο κλειδί. Χρησιμοποιούμε τον προσδιορισμό FOREIGN KEY, ορίζουμε μέσα σε παρένθεση το όνομα του πεδίου και στη συνέχεια γράφουμε σε ποιανού πίνακα κλειδί αναφέρεται (REFERENCES) το ξένο κλειδί (στην προκειμένη περίπτωση στον πίνακα teacher και στο πεδίο ar_tautotitas).

## 4.2 Εσωτερικό (καρτεσιανό) γινόμενο πινάκων

Ας θεωρήσουμε την περίπτωση όπου ένας καθηγητής διδάσκει αποκλειστικά ένα μάθημα αλλά ένα μάθημα διδάσκεται από πολλούς καθηγητές. Δηλαδή έχουμε τα ακόλουθα δεδομένα:

| **teacher** | | | |
|---|---|---|---|
| **id** | **name** | **surname** | **speciality** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS |
| P458652 | STATHIS | KIOSSES | FILOLOGOS |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS |

| **subject** | | | |
|---|---|---|---|
| **ıd** | **title** | **type** | **tar_tautotitas** |
| 1 | DataBases | THEORY | X256526 |
| 2 | MySQL | LAB | X256526 |
| 3 | C | THEORY | R586323 |
| 4 | English | THEORY | P458652 |

Το *εσωτερικό γινόμενο* (ή και *καρτεσιανό* γινόμενο) αυτών των δύο πινάκων είναι ένας συνδυασμένος πίνακας που έχει το σύνολο των πεδίων των δύο πινάκων και που κάθε γραμμή του ενός πίνακα

συνδυάζεται με όλες τις γραμμές του άλλου πίνακα. Δηλαδή στην προκειμένη περίπτωση το εσωτερικό γινόμενο των πινάκων teacher και subject είναι ο ακόλουθος πίνακας:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Ar_tautotitas** | **name** | **surname** | **speciality** | **Id** | **title** | **Type** | **tar_tautotitas** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS | 1 | DataBases | THEORY | X256526 |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS | 2 | MySQL | LAB | X256526 |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS | 3 | C | THEORY | R586323 |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS | 4 | English | THEORY | P458652 |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS | 1 | DataBases | THEORY | X256526 |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS | 2 | MySQL | LAB | X256526 |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS | 3 | C | THEORY | R586323 |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS | 4 | English | THEORY | P458652 |
| P458652 | STATHIS | KIOSSES | FILOLOGOS | 1 | DataBases | THEORY | X256526 |
| P458652 | STATHIS | KIOSSES | FILOLOGOS | 2 | MySQL | LAB | X256526 |
| P458652 | STATHIS | KIOSSES | FILOLOGOS | 3 | C | THEORY | R586323 |
| P458652 | STATHIS | KIOSSES | FILOLOGOS | 4 | English | THEORY | P458652 |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS | 1 | DataBases | THEORY | X256526 |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS | 2 | MySQL | LAB | X256526 |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS | 3 | C | THEORY | R586323 |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS | 4 | English | THEORY | P458652 |

Βλέπουμε δηλαδή ότι τα πεδία του εσωτερικού γινομένου αποτελούνται από την ένωση των πεδίων των δύο πινάκων και οι εγγραφές του δημιουργούνται καθώς κάθε γραμμή του πρώτου πίνακα συνδυάζεται με κάθε γραμμή του δεύτερου πίνακα. Προφανώς η λογική αυτή μπορεί να επεκταθεί και σε περισσότερους από δύο πίνακες. Κάθε φορά συνδυάζουμε όλες με όλες τις γραμμές. Το πλεονέκτημα του εσωτερικού γινομένου είναι ότι περιέχει *όλη τη συνδυασμένη* πληροφορία των δύο πινάκων.

Μπορούμε να πάρουμε το εσωτερικό γινόμενο των πινάκων teacher και subject με την ακόλουθη εντολή:

```
SELECT * FROM teacher, subject
```

Δηλαδή παραθέτουμε μέσα στο FROM τους πίνακες των οποίων θέλουμε να υπολογίσουμε το εσωτερικό γινόμενο διαχωρισμένους με κόμμα.

## 4.3 Συνένωση (JOIN) πινάκων

Στον προηγούμενο υπότιτλο εξετάσαμε τι είναι το εσωτερικό γινόμενο. Τώρα θα εξετάσουμε πως μπορούμε να εκμεταλλευτούμε αυτή την ιδιότητα για να αντλήσουμε συνδυασμένα δεδομένα από περισσότερους πίνακες. Θεωρώντας και πάλι την διαμόρφωση δεδομένων του προηγούμενου υπότιτλου (1:N), έστω ότι θέλουμε να απαντήσουμε στην ακόλουθη ερώτηση:

«Ποια είναι οι τίτλοι των μαθημάτων και τα ονοματεπώνυμα των καθηγητών που τα διδάσκουν»

Το θέμα με αυτή την ερώτηση είναι ότι για να απαντηθεί απαιτείται ο συνδυασμός και των δύο πινάκων. Έτσι πρέπει να δώσουμε την ακόλουθη εντολή:

```
SELECT title, name, surname
FROM teacher JOIN subject ON tar_tautotitas=ar_tautotitas
```

Αυτή η εντολή θα επιστρέψει το ακόλουθο αποτέλεσμα:

| **Title** | **Name** | **Surname** |
|---|---|---|
| Databases | NIKOS | PAPPAS |
| MySQL | NIKOS | PAPPAS |
| C | MARIA | IOANNOU |
| English | STATHIS | KIOSSES |

Αρχικά ορίζουμε στο SELECT τα πεδία που θέλουμε να εμφανιστούν (title, name, surname), στη συνέχεια μέσα στο FROM ορίζουμε τους πίνακες που θα συμπεριλάβουμε και τον τρόπο που θα γίνει αυτό. Βλέπουμε ότι χρησιμοποιούμε το *JOIN*, που σημαίνει *συνένωση*, δηλαδή συνδυασμός των δεδομένων από δύο πίνακες τους teacher και subject. Μετά από αυτό ορίζουμε *την συνθήκη της συνένωσης*, δηλαδή ότι θα πρέπει το πεδίο tar_tautotitas του subject να είναι ίδιο με το πεδίο ar_tautotitas του teacher. Η λογική δηλαδή είναι ότι ταιριάζουμε εκείνες τις γραμμές του subject με τις αντίστοιχες του teacher βάσει του κριτηρίου που ορίζει η συνθήκη της συνένωσης.

Μπορούμε να εκλάβουμε την συνένωση δύο πινάκων ως μια περικοπή των αποτελεσμάτων που θα επέστρεφε το εσωτερικό γινόμενο των πινάκων όταν εφαρμόσουμε την συνθήκη συνένωσης. Πράγματι αν ψάξουμε στον πίνακα του εσωτερικού γινομένου για τις εγγραφές που ικανοποιούν το κριτήριο tar_tautotitas=ar_tautotitas, τότε θα πάρουμε τα αποτελέσματα που φαίνονται παραπάνω. Με αυτή τη λογική θα μπορούσαμε να γράψουμε μια ισοδύναμη εντολή με την προηγούμενη:

```
SELECT title, name, surname FROM teacher, subject WHERE tar_tautotitas=ar_tautotitas
```

Δηλαδή εφαρμόζουμε στο εσωτερικό γινόμενο, την συνθήκη της συνένωσης ως κριτήριο επιλογής στο SELECT.

Επίσης έχουμε τη δυνατότητα να ορίσουμε και επιπλέον κριτήρια αναζήτησης (επιλογές ταξινόμησης και ότι άλλο έχουμε στην SQL) μαζί με το JOIN. Για παράδειγμα θα μπορούσαμε να έχουμε την ακόλουθη ερώτηση:

«Ποια είναι τα στοιχεία του καθηγητή που διδάσκει το μάθημα Βάσεις Δεδομένων;»

```
SELECT name, surname FROM teacher JOIN subject ON tar_tautotitas=ar_tautotitas WHERE title='Databases'
```

Ή ισοδύναμα

```
SELECT name, surname FROM teacher, subject WHERE tar_tautotitas=ar_tautotitas AND title='Databases'
```

Μια άλλη ερώτηση θα μπορούσε να είναι:

«Ποια είναι τα ονοματεπώνυμα των καθηγητών που διδάσκουν εργαστηριακά μαθήματα;»

```
SELECT name, surname FROM teacher, subject WHERE type='LAB' AND tar_tautotitas=ar_tautotitas
```

Όπως και προηγουμένως έτσι και εδώ ορίζουμε στο FROM τους πίνακες από τους οποίους θα αντλήσουμε τα δεδομένα και κατόπιν στο WHERE ένα κριτήριο επιλογής για τα μαθήματα (ότι ο τύπος τους θα είναι 'LAB' δηλαδή εργαστήρια) και ένα δεύτερο κριτήριο που αποτελεί τη συνθήκη της συνένωσης.

#### 4.3.1.1 Επίλυση ασαφειών στα ονόματα των πεδίων

Επειδή κατά τη συνένωση τα πεδία των πινάκων αναμειγνύονται, ενδεχομένως ορισμένες φορές να δημιουργηθούν ασάφειες. Για παράδειγμα αν και οι δύο πίνακες έχουν ένα πεδίο με το ίδιο όνομα, τότε απαιτείται κάποιος μηχανισμός για να προσδιορίζουμε σε ποιο πεδίο αναφερόμαστε την κάθε φορά, αυτό γίνεται ορίζοντας το όνομα του πεδίου ως <όνομα πινακα>.<όνομα πεδίου>. Για παράδειγμα θα μπορούσαμε να είχαμε γράψει την προηγούμενη εντολή ως:

```
SELECT teacher.name, teacher.surname FROM teacher, subject WHERE subject.type='LAB' AND subject.tar_tautotitas=teacher.ar_tautotitas
```

Δηλαδή βλέπουμε ότι πριν από το όνομα του κάθε πεδίου αναγράφεται και το όνομα του πίνακα που αυτό ανήκει διαχωρισμένο με τελεία. Στην πραγματικότητα αυτός ο τρόπος γραφής των πεδίων είναι και ο *πλήρης τρόπος* αναγραφής τους. Για δική μας διευκόλυνση έχουμε επίσης τη δυνατότητα να ορίσουμε ψευδώνυμα στους πίνακες της συνένωσης χρησιμοποιώντας το AS, και να χρησιμοποιούμε τα ψευδώνυμα των πινάκων για να προσδιορίζουμε τα πεδία τους. Για παράδειγμα η προηγούμενη εντολή θα μπορούσε να είχε γραφτεί ως εξής:

```
SELECT t.name, t.surname FROM teacher AS t, subject AS s WHERE s.type='LAB' AND s.tar_tautotitas=t.ar_tautotitas
```

Βλέπουμε δηλαδή ότι εκχωρήσαμε το ψευδώνυμο t αντί για το teacher και το s αντί για το subject.

Τέλος αξίζει να σημειώσουμε πως ο ειδικός χαρακτήρας *, εξακολουθεί να ισχύει και όταν χρησιμοποιούμε το πλήρες όνομα των πεδίων (ή και με ψευδώνυμα). Για παράδειγμα στην

προηγούμενη εντολή αντί για το ονοματεπώνυμο θέλαμε να πάρουμε όλα τα πεδία από τον πίνακα teacher θα γράφαμε:

```
SELECT t.* FROM teacher AS t, subject AS s WHERE s.type='LAB' AND s.tar_tautotitas=t.ar_tautotitas
```

Δηλαδή το t.* σημαίνει table.*, δηλαδή όλα τα πεδία του πίνακα table.

#### 4.3.1.2 Εξωτερική συνένωση

Μια ιδιαιτερότητα εμφανίζεται όταν η συνθήκη συνένωσης εφαρμόζεται σε πεδία που είναι NULL. Για παράδειγμα ας θεωρήσουμε ότι έχουμε ακόμη ένα μάθημα που όμως δεν έχει ανατεθεί σε κανέναν καθηγητή, σύμφωνα με την ακόλουθη διαμόρφωση των δεδομένων:

| **teacher** | | | |
|---|---|---|---|
| **ar_tautotitas** | **name** | **surname** | **speciality** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS |
| P458652 | STATHIS | KIOSSES | FILOLOGOS |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS |

| **subject** | | | |
|---|---|---|---|
| **id** | **title** | **Type** | **tar_tautotitas** |
| 1 | DataBases | THEORY | X256526 |
| 2 | MySQL | LAB | X256526 |
| 3 | C | THEORY | R586323 |
| 4 | English | THEORY | P458652 |
| 5 | Java | LAB | NULL |

Βάσει αυτών των πινάκων ας υποθέσουμε ότι θέλουμε να απαντήσουμε την ακόλουθη ερώτηση:

«Ποιοι είναι όλοι οι τίτλοι των μαθημάτων και τα αντίστοιχα ονόματα των καθηγητών που τα διδάσκουν;»

Αν γράψουμε όπως και προηγουμένως την εντολή:

```
SELECT title, name, surname
FROM teacher JOIN subject ON tar_tautotitas=ar_tautotitas
```

Τότε όπως και προηγουμένως θα πάρουμε την απάντηση:

| Title | Name | Surname |
|---|---|---|
| Databases | NIKOS | PAPPAS |
| MySQL | NIKOS | PAPPAS |
| C | MARIA | IOANNOU |
| English | STATHIS | KIOSSES |

Που δεν είναι όλοι οι τίτλοι των μαθημάτων, αλλά οι τίτλοι των μαθημάτων *που έχουν ανατεθεί*, δεν έχει επιστραφεί το μάθημα "Java". Ο λόγος είναι ότι η συνένωση δεν μπόρεσε να ταιριάξει το πεδίο tar_tautotitas της εγγραφής με id=5 του πίνακα subject με κανένα από τα πεδία ar_tautotitas του πίνακα teacher, γιατί απλά ήταν NULL. Δηλαδή η συνθήκη συνένωσης δεν περιέλαβε κανένα NULL πεδίο. Αν όμως θέλουμε να περιληφθούν στο αποτέλεσμα και οι εγγραφές του πίνακα subject που έχουν NULL στο πεδίο tar_tautotitas, τότε πρέπει να γράψουμε την ακόλουθη εντολή:

```
SELECT title, name, surname
FROM teacher RIGHT OUTER JOIN subject ON tar_tautotitas=ar_tautotitas
```

Αυτό που έχει αλλάξει είναι ότι αντί για JOIN, έχουμε RIGHT OUTER JOIN. Η διαφορά τους είναι ότι το RIGHT OUTER JOIN θα συμπεριλάβει στο αποτέλεσμα όλες τις εγγραφές του πίνακα που βρίσκεται στα δεξιά (δηλ. του subject) που για το πεδίο της συνθήκης συνένωσης (δηλ. το tar_tautotitas) έχουν NULL. Οπότε θα έχουμε το ακόλουθο επιδιωκόμενο αποτέλεσμα:

| Title | Name | Surname |
|---|---|---|
| Databases | NIKOS | PAPPAS |
| MySQL | NIKOS | PAPPAS |
| C | MARIA | IOANNOU |
| English | STATHIS | KIOSSES |
| Java | NULL | NULL |

Βλέπουμε δηλαδή ότι για κάθε εγγραφή που δεν ικανοποιεί την συνθήκη συνένωσης, περιλαμβάνεται στο τελικό αποτέλεσμα συνδυασμένη με NULL για τα πεδία του άλλου πίνακα. Έτσι περιλαμβάνουμε την "Java" που ανήκει στον πίνακα subject αλλά εφόσον δεν ικανοποιεί το tar_tautotitas=ar_tautotitas, τα πεδία name, surname του πίνακα teacher παραμένουν NULL. Συμμετρικά με το RIGHT OUTER JOIN υπάρχει και το LEFT OUTER JOIN που συμπεριλαμβάνει τις NULL εγγραφές του πίνακα που βρίσκεται στα αριστερά. Και προφανώς εναλλάσσοντας τις θέσεις των πινάκων μπορούμε να χρησιμοποιούμε πάντα το LEFT OUTER JOIN (ή το RIGHT OUTER JOIN). Τέλος αν θέλουμε να συμπεριληφθούν οι NULL εγγραφές και των δύο πινάκων (και εξ' αριστερών και εκ' δεξιών) τότε έχουμε ένα FULL OUTER JOIN.

Επίσης πρέπει να σημειώσουμε πως ο χαρακτηρισμός OUTER δεν είναι υποχρεωτικός. Θα μπορούσαμε απλά να γράψουμε LEFT JOIN ή RIGHT JOIN, ωστόσο χρησιμοποιείται για να τονίζει τη διαφορά που έχει το εξωτερικό (δηλ. OUTER) JOIN με το INNER (δηλ. εσωτερικό) JOIN. Δηλαδή αν δεν προσδιορίσουμε ότι πρόκειται για OUTER JOIN τότε εννοείται INNER JOIN. Επίσης αν η συνθήκη συνένωσης αφορά έλεγχο ισότητας (όπως ήταν στα παραδείγματα που είδαμε έως τώρα), πρόκειται για EQUI JOIN. Αν η συνθήκη συνένωσης αφορά συνθήκη ισότητας σε πεδία που έχουν το ίδιο όνομα και στους δύο πίνακες, τότε μιλάμε για ένα NATURAL JOIN. Τέλος στη γενική περίπτωση που έχουμε μια συνθήκη συνένωσης οποιοδήποτε τύπου, μιλάμε για THETA JOIN

Στον επόμενο πίνακα φαίνονται τα διαφορετικά είδη JOIN:

| **Τύπος JOIN** | **Εξήγηση** |
|---|---|
| INNER JOIN | Join που δεν συμπεριλαμβάνονται οι NULL εγγραφές |
| NATURAL JOIN | Join που η συνθήκη συνένωσης αφορά πεδία που και τους δύο πίνακες έχουν το ίδιο όνομα. |
| EQUI JOIN | Join που η συνθήκη συνένωσης είναι ισότητα μεταξύ πεδίων. |
| THETA JOIN | Join με γενικού τύπου συνθήκη συνένωσης. |
| LEFT OUTER JOIN | Join που συμπεριλαμβάνονται οι NULL εγγραφές του αριστερού πίνακα. |
| RIGHT OUTER JOIN | Join που συμπεριλαμβάνονται οι NULL εγγραφές του δεξιού πίνακα. |
| FULL OUTER JOIN | Join που συμπεριλαμβάνονται οι NULL εγγραφές του αριστερού και του δεξιού πίνακα. |

Φυσικά στην πράξη δεν έχει και πολύ νόημα το θεωρητικό όνομα του JOIN. Η ουσία είναι πως κάθε φορά θα πρέπει να προσδιορίζουμε την συνθήκη συνένωσης που θέλουμε. Μόνη ουσιαστική εξαίρεση είναι τα OUTER JOIN όταν θέλουμε να συμπεριληφθούν και NULL εγγραφές.

## 4.4 Επιλογή από σύνολα τιμών

### 4.4.1 Τελεστές IN, ANY, SOME, ALL

Από το προηγούμενο κεφάλαιο έχουμε δει πως μπορούμε να απαντήσουμε μια ερώτηση σαν την ακόλουθη:

«Ανέκτησε τις εγγραφές από τον πίνακα των μαθημάτων που έχουν τα id 1, 3, 4”

Είδαμε ότι αυτό γίνεται με μια εντολή όπως:

```
SELECT * FROM subject WHERE id=1 OR id=3 OR id=4
```

Ωστόσο σε μια περίπτωση όπως αυτή, όπου θέλουμε ένα πεδίο να λαμβάνει τιμές από ένα προκαθορισμένο σύνολο τιμών, τότε μπορούμε εναλλακτικά να χρησιμοποιήσουμε τον τελεστή IN όπως αυτό φαίνεται στο ακόλουθο παράδειγμα:

```
SELECT * FROM subject WHERE id IN (1,3,4)
```

Δηλαδή παραθέτουμε μετά το IN μέσα σε παρενθέσεις τις τιμές εντός των οποίων επιτρέπεται να κινείται το πεδίο. Συμπληρωματικά θα μπορούσαμε να ορίσουμε την ερώτηση:

«Ανέκτησε τις εγγραφές από τον πίνακα των μαθημάτων που δεν έχουν τα id 1, 3, 4”

```
SELECT * FROM subject WHERE id NOT IN (1,3,4)
```

Την ίδια λειτουργικότητα με το IN έχουν και οι τελεστές = ANY και = SOME. Δηλαδή θα μπορούσαμε να γράψουμε την παραπάνω εντολή με τους ακόλουθους ισοδύναμους τρόπους:

```
SELECT * FROM subject WHERE id = ANY(1,3,4)
```

Δηλαδή φέρε τα id που είναι ίσα με οποιοδήποτε από τα 1, 3,4

```
SELECT * FROM subject WHERE id = SOME (1,3,4)
```

Δηλαδή φέρε τα id που είναι ίσα με κάποια από τα 1, 3, 4

Φυσικά θα μπορούσαμε να συνδυάσουμε το ANY και με άλλους τελεστές όπως >, <, <=, >=. Επίσης μπορούμε να συνδυάσουμε τους τελεστές IN, ANY, SOME με το ALL. Για παράδειγμα θα μπορούσαμε να δώσουμε την ακόλουθη εντολή:

```
SELECT * FROM subject WHERE id > ALL (1, 2, 5)
```

Εδώ ζητούμε να φέρει τα μαθήματα που το id τους είναι μεγαλύτερο από *όλες* τις τιμές εντός της παρένθεσης (ουσιαστικά μεγαλύτερο από το 5).

## 4.4.2 Εμφωλευμένες ερωτήσεις

Ένα πολύ ενδιαφέρον χαρακτηριστικό της SQL είναι η δυνατότητα που μας παρέχει να ενθέτουμε queries το ένα μέσα στο άλλο, αυτά ονομάζονται εμφωλευμένα (nested) queries. Για παράδειγμα ας δούμε έναν εναλλακτικό τρόπο που θα μπορούσαμε να απαντήσουμε την ακόλουθη ερώτηση:

«Ποια είναι τα ονοματεπώνυμα των καθηγητών που διδάσκουν εργαστηριακά μαθήματα;»

```
SELECT name, surname FROM teacher

WHERE ar_tautotitas IN (SELECT tar_tautotitas FROM subject WHERE type='LAB')
```

Εδώ χρησιμοποιούμε το εσωτερικό query για να συγκεντρώσουμε τους αριθμούς ταυτότητας των καθηγητών που διδάσκουν εργαστηριακά μαθήματα. Κατόπιν το εξωτερικό query χρησιμοποιεί αυτό

το σύνολο των Α.Τ. για να επιλέξει τις εγγραφές που πρέπει να εμφανίσει το όνομα και επώνυμο. Μια άλλη ερώτηση από τον πίνακα των προϊόντων, θα μπορούσε να είναι:

“Εμφάνισε όλα τα στοιχεία του ακριβότερου και του φθηνότερου προϊόντος”

```
SELECT * FROM product WHERE cost = (SELECT max(cost) FROM product) OR cost = (SELECT min(cost) FROM product)
```

Εδώ βλέπουμε ότι χρησιμοποιούμε δύο εμφωλευμένα queries για να ανακτήσουμε την φθηνότερη και την ακριβότερη τιμή προϊόντος.

Για να δούμε περισσότερο την αξία των εμφωλευμένων ερωτήσεων ας κάνουμε λίγο πιο περίπλοκη τη βάση με τους καθηγητές και τα μαθήματα. Θεωρούμε λοιπόν ότι θέλουμε να προσθέσουμε τα παιδιά που έχει ο κάθε καθηγητής και τα ορίσουμε σε ξεχωριστό πίνακα:

| **teacher** | | | |
|---|---|---|---|
| **ar_tautotitas** | **name** | **surname** | **speciality** |
| X256526 | NIKOS | PAPAS | PLHROFORIKOS |
| R589643 | ELENH | NIKOLAOY | PLHROFORIKOS |
| P458652 | STATHIS | KIOSSES | FILOLOGOS |
| R586323 | MARIA | IWANNOY | PLHROFORIKOS |

| **children** | | | |
|---|---|---|---|
| **Id** | **Name** | **Surname** | **Parent** |
| 1 | SPIROS | KIOSSES | P458652 |
| 2 | NIKOS | PAPAS | X256526 |
| 3 | GIOTA | IWANNOY | R586323 |
| 4 | GIOTA | PAPA | X256526 |

Έτσι όπως φαίνεται στους πίνακες έχουμε ότι ο Α897423 έχει το παιδί με id=1, ο Χ256526 τα παιδιά με id 2 και 4, και ο R586323 το παιδί με id 3. Έστω λοιπόν ότι θέλουμε να απαντήσουμε την ερώτηση:

«Ποια είναι τα στοιχεία των καθηγητών που έχουν παιδί που έχει το ίδιο όνομα με το δικό τους;»

```
SELECT * FROM teacher
WHERE ar_tautotitas IN (SELECT parent FROM children where teacher.name=name)
```

Αυτό το query ορίζει μια ιδιαίτερη κατηγορία εμφωλευμένων ερωτήσεων, που ονομάζονται *συσχετιζόμενες εμφωλευμένες ερωτήσεις* (correlated nested queries). Η ιδιομορφία τους είναι ότι το

εσωτερικό query χρησιμοποιεί πεδία από το εξωτερικό query, και στην προκειμένη περίπτωση αυτό είναι το όνομα. Σε αυτό το παράδειγμα το αποτέλεσμα του εσωτερικού query εξαρτάται από την τιμή του πεδίου για το εξωτερικό query. Μπορούμε να κατανοήσουμε καλύτερα ένα συσχετιζόμενο εμφωλευμένο query θεωρώντας ότι η εσωτερική ερώτηση υπολογίζεται για κάθε εγγραφή της εξωτερικής ερώτησης.

## 4.5 Πράξεις σε αποτελέσματα ερωτήσεων

Έως τώρα έχουμε δει δύο τρόπους για να συνδυάζουμε δεδομένα από περισσότερα queries. Τα joins και τα εμφωλευμένα queries. Υπάρχουν όμως και περιπτώσεις που δεν θέλουμε να συνδυάσουμε δεδομένα αλλά να ενώσουμε ή να διαχωρίσουμε δεδομένα. Για παράδειγμα έστω το ακόλουθο ερώτημα:

«Δώσε μια λίστα με όλα τα μοναδικά ονόματα που υπάρχουν στους πίνακες teacher και children»

```
SELECT name FROM teacher

UNION

SELECT name FROM children
```

Εδώ βλέπουμε ότι ενώνουμε τα αποτελέσματα από δυο εντελώς ανεξάρτητα queries χρησιμοποιώντας την πράξη UNION (δηλ. ένωση συνόλων). Το αποτέλεσμα της παραπάνω εντολής θα είναι ένας ενιαίος πίνακας που θα φαίνονται σε μια λίστα τα ονόματα που υπάρχουν και στους δύο πίνακες. Επίσης σημειώνουμε ότι το UNION απαλείφει αυτόματα τις διπλοεγγραφές από το τελικό αποτέλεσμα. Οπότε τα αποτελέσματα που θα επιστραφούν θα είναι:

| **Name** |
|---|
| STATHIS |
| MARIA |
| ELENH |
| NIKOS |
| SPIROS |
| GIOTA |

Δύο άλλες πράξεις που μπορούμε να εφαρμόσουμε στα σύνολα των δεδομένων είναι η INTERSECTION και η EXCEPT. Η INTERSECTION υπολογίζει την τομή δύο συνόλων δηλαδή τα δεδομένα που είναι κοινά και στα δύο σύνολα. Αν εφαρμόσουμε αυτή την πράξη στα ονόματα των πινάκων teacher και children θα πάρουμε το ακόλουθο αποτέλεσμα:

| Name |
|---|
| NIKOS |

Δηλαδή παίρνουμε τα ονόματα που υπάρχουν και στους δύο πίνακες.

Τέλος η EXCEPT επιστρέφει τις τιμές του αριστερού αποτελέσματος που δεν υπάρχουν στο δεξιό αποτέλεσμα. Αν δηλαδή εφαρμόζαμε την EXCEPT αριστερό πίνακα τον teacher και δεξιό τον children θα παίρναμε το ακόλουθο αποτέλεσμα:

| Name |
|---|
| ELENH |
| STATHIS |
| MARIA |

Τέλος πρέπει να σημειώσουμε ότι οι πράξεις INTERSECTION και EXCEPT δεν υποστηρίζονται από την MySQL.

## 4.5.1 Τελεστής EXISTS

Ορισμένες φορές αυτό που μας ενδιαφέρει δεν είναι το ακριβές αποτέλεσμα μιας ερώτησης, αλλά αν η ερώτηση επιστρέφει αποτελέσματα ή όχι. Για παράδειγμα έστω η ακόλουθη ερώτηση:

«Ανέκτησε τα στοιχεία των καθηγητών που έχουν παιδιά»

```
SELECT * FROM teacher

WHERE EXISTS (SELECT * FROM children WHERE ar_tautotitas=parent)
```

Το εσωτερικό query επιλέγει τα παιδιά που έχει ο κάθε γονέας. Εμείς όμως δεν ενδιαφερόμαστε για τα δεδομένα των παιδιών, αλλά απλώς για την ύπαρξή τους. Έτσι αν ένας γονέας έχει έστω και ένα παιδί (δηλαδή το εσωτερικό select επιστρέψει κάτι), τότε το EXISTS θα επιστρέψει true. Ομοίως αν θέλαμε να πάρουμε τους καθηγητές που δεν έχουν παιδιά θα χρησιμοποιούσαμε το NOT EXISTS.

Και πάλι φυσικά μπορούμε να συνδυάσουμε τον τελεστή EXISTS με άλλα κριτήρια αναζήτησης. Για παράδειγμα η ακόλουθη ερώτηση:

«Ανέκτησε τα στοιχεία των καθηγητών που είναι πληροφορικοί και έχουν παιδιά»

```
SELECT * FROM teacher

WHERE speciality='PLHROFORIKOS' AND EXISTS (SELECT * FROM children WHERE
ar_tautotitas=parent)
```

## 4.6 Πολύπλοκες ερωτήσεις

Σε αυτόν τον υπότιτλο αναπτύσσουμε ορισμένα χαρακτηριστικά παραδείγματα πολύπλοκων ερωτήσεων, καθώς μπορούμε να αντλήσουμε ενδιαφέροντα μαθήματα από την κάθε μια.

### 4.6.1 Query σε αποτέλεσμα query

Αρχικά θέλουμε να εκμεταλλευτούμε την δυνατότητα που έχουμε να τοποθετήσουμε ένα εμφωλευμένο query στο FROM. Δηλαδή να εφαρμόσουμε ένα query στο αποτέλεσμα ενός άλλου query. Για παράδειγμα έστω ότι θέλουμε να απαντήσουμε στο ακόλουθο ερώτημα, που αφορά τον πίνακα των προϊόντων:

«Αν κάνουμε μια έκπτωση 30% στα προϊόντα που κατασκευάστηκαν πριν το 2007 μόνο (δηλαδή αυτά που είναι εντός του 2007 διατηρούν την τιμή τους), ποια από αυτά θα έχουν κόστος κάτω των 50 ευρώ;»

Αρχικά παρατηρούμε ότι η ερώτηση «ποια από τα προϊόντα έχουν κόστος κάτω των 50 ευρώ» δεν αφορά τις τιμές των προϊόντων, ούτε καν τις μειωμένες τιμές των προϊόντων, αλλά τις μειωμένες τιμές *μερικών* προϊόντων. Έτσι λοιπόν πρέπει πρώτα να δημιουργήσουμε τις νέες τιμές και μετά να επιλέξουμε αυτά που θέλουμε. Προκειμένου να δημιουργήσουμε τις νέες τιμές πρέπει για μεν τα παλιά να κάνουμε μείωση, για τα νέα να μείνουν όπως είναι. Αυτό μπορεί να γίνει με το ακόλουθο query:

```
SELECT id, cost*0.7 AS newCost FROM product WHERE production<'2007-01-01'

UNION

SELECT id, cost AS newCost FROM product WHERE production>='2007-01-01'
```

Εδώ χρησιμοποιούμε τον τελεστή UNION προκειμένου να ενώσουμε τα δύο σύνολα των παλιών προϊόντων με την μείωση και των νέων προϊόντων δίχως την μείωση. Και τα δύο σύνολα έχουν την σωστή τιμή στο πεδίο newCost. Καταλήγουμε λοιπόν στο ακόλουθο σύνολο δεδομένων:

| **Id** | **newCost** |
|---|---|
| 3 | 175.00 |
| 4 | 49.00 |
| 1 | 23.50 |
| 2 | 85.00 |

Βλέπουμε δηλαδή ότι έγινε έκπτωση στα προϊόντα με id 3 και 4 που έχουν κατασκευαστεί πριν το 2007, αλλά τα υπόλοιπα έχουν μείνει ανέπαφα. Στη συνέχεια από αυτό το σύνολο θα επιλέξουμε εκείνα που έχουν κόστος κάτω των 50 ευρώ. Άρα καταλήγουμε στο ακόλουθο query

```
SELECT id, newCost FROM
```

```
(SELECT id, cost*0.7 AS newCost FROM product WHERE production<'2007-01-01' UNION
SELECT id, cost AS newCost FROM product WHERE production>='2007-01-01') AS q
WHERE q.newCost<50
```

Εδώ λοιπόν βλέπουμε ότι πήραμε όλο το προηγούμενο query και το βάλαμε μέσα στο FROM του νέου query. Επειδή θέλουμε έναν τρόπο να αναφερόμαστε στο πεδίο newCost του νέου query για να κάνουμε την σύγκριση <50, του εκχωρούμε το ψευδώνυμο q. Έτσι το τελικό query θα επιστρέψει τα id 4 και 1.

## 4.6.2 JOIN ενός πίνακα με τον εαυτό του

Έστω η ακόλουθη ερώτηση:

«Με βάση τον πίνακα των προϊόντων, απαντήστε ποιο είναι το ακριβότερο προϊόν δίχως όμως να χρησιμοποιήσετε τα min, max (ούτε και την limit)»

Η δυσκολία σε αυτό το ερώτημα είναι ότι δεν επιτρέπεται να χρησιμοποιήσουμε τις συναρτήσεις min, max. Για αυτό θα ακολουθήσουμε μια διαφορετική προσέγγιση. Μπορούμε να σκεφτούμε πως το προϊόν με την μεγαλύτερη τιμή είναι αυτό που συγκρινόμενο με τα άλλα προϊόντα δεν υπάρχει κανένα άλλο που να έχει μεγαλύτερη τιμή από αυτό. Άρα θέλουμε ένα query που να παίρνει δύο προϊόντα κάθε φορά και να συγκρίνει την τιμή τους. Αφού θέλουμε να πάρουμε δύο, αναγκαστικά θα κάνουμε JOIN γιατί είναι ο μόνος τρόπος να συσχετίσουμε δυο εγγραφές μεταξύ τους, και αφού θέλουμε και οι δύο εγγραφές να αναφέρονται στον ίδιο πίνακα θα κάνουμε JOIN τον product με τον εαυτό του! Μια ιδέα λοιπόν είναι να γράψουμε το ακόλουθο query:

```
SELECT p1.* FROM product AS p1, product AS p2 WHERE p1.cost>p2.cost
```

Εδώ λοιπόν δημιουργούμε δύο ψευδώνυμα του πίνακα product, τα p1 και p2 και ορίζουμε συνθήκη το κόστος του p1 να είναι μεγαλύτερο από το κόστος του p2. Ας δούμε πως θα λειτουργήσει αυτό, για τα δεδομένα του ακόλουθου πίνακα:

| **Προϊόν** | | | |
|---|---|---|---|
| **Κωδικός** | **Όνομα** | **Ημερ_παραγωγής** | **Κόστος** |
| 1 | Ποντίκι Optical | 2007-09-15 | 23.50 |
| 2 | Σκληρός δίσκος 300GB | 2007-09-15 | 85.00 |
| 3 | CPU P4 3.2GHz | 2006-08-20 | 250.00 |

Όπως γνωρίζουμε το ΣΔΒΔ θα πάρει μια μία τις εγγραφές του πίνακα και θα συγκρίνει αν ικανοποιούν την συνθήκη που έχουμε ορίσει. Έτσι τα βήματα που θα εκτελέσει φαίνονται στον ακόλουθο πίνακα:

| **id για το p1** | **id για το p2** | **Το παίρνω;** | **Γιατί;** |
|---|---|---|---|
| **1** | 1 | Όχι | 23.50 δεν είναι μεγαλύτερο από 23.50 |
| **1** | 2 | Όχι | 23.50 δεν είναι μεγαλύτερο από 85.00 |
| **1** | 3 | Όχι | 23.50 δεν είναι μεγαλύτερο από 250.00 |
| **2** | 1 | **Ναι** | 85.00 είναι μεγαλύτερο από το 23.50 |
| **2** | 2 | Όχι | 85.00 δεν είναι μεγαλύτερο από το 85.00 |
| **2** | 3 | Όχι | 85.00 δεν είναι μεγαλύτερο από το 250.00 |
| **3** | 1 | **Ναι** | 250.00 είναι μεγαλύτερο από 23.50 |
| **3** | 2 | **Ναι** | 250.00 είναι μεγαλύτερο από 85.00 |
| **3** | 3 | Όχι | 250.00 δεν είναι μεγαλύτερο από 250.00 |

Άρα εφόσον το query επιστρέφει το p1 τα αποτελέσματα θα περιλαμβάνουν τις εγγραφές με id=3 (που είναι σωστό) και με id=2 (που είναι λάθος). Το πρόβλημα είναι ότι συμπεριλαμβάνεται στο αποτέλεσμα το 2 που όμως είναι λάθος. Το λάθος οφείλεται στο ότι η εγγραφή με id=2 είναι μεγαλύτερη αλλά όχι από όλες τις άλλες μόνο από την id=1. Δηλαδή το p1 είμαι μεγαλύτερος από τον p2 σημαίνει ότι βρέθηκε μια εγγραφή του p1 είναι μεγαλύτερη από μια εγγραφή του p2 και όχι ότι είναι μεγαλύτερη από όλες. Η βαθύτερη αιτία του προβλήματος είναι ότι για να αποφασίσουμε ότι κάτι είναι το μεγαλύτερο πρέπει να το συγκρίνουμε με όλα τα άλλα. Όμως για να αποφασίσουμε ότι κάτι δεν είναι το μεγαλύτερο αρκεί να βρεθεί έστω και ένα άλλο που να είναι μεγαλύτερο από αυτό. Έτσι λοιπόν το σωστό query είναι:

```
SELECT * FROM product WHERE id NOT IN (SELECT p1.id FROM product AS p1, product AS p2 WHERE p1.cost<p2.cost)
```

Το εμφωλευμένο query βρίσκει όλα τα id που έχουν έστω και ένα άλλο που είναι μικρότερο από αυτό. Το εξωτερικό επιλέγει εκείνες τις εγγραφές που δεν έχουν μεγαλύτερό τους, αλλά επιλέγει τη μέγιστη που είναι και το ζητούμενο αποτέλεσμα.

# 5 Προχωρημένα χαρακτηριστικά των ΣΔΒΔ

## 5.1 Περιορισμοί στις βάσεις

Στα προηγούμενα κεφάλαια είδαμε πως μπορούμε να δημιουργήσουμε έναν πίνακα (CREATE TABLE), να ορίσουμε πρωτεύοντα (PRIMARY KEY) και ξένα κλειδιά (FOREIGN KEY) καθώς και πώς να εισάγουμε (INSERT), ενημερώσουμε (UPDATE) ή διαγράψουμε (DELETE) εντολές. Ωστόσο η προσθαφαίρεση δεδομένων είναι δυνατόν να προκαλέσει προβλήματα στην *ακεραιότητα των δεδομένων* της βάσης. Για να διατηρείται σωστή η ακεραιότητα των δεδομένων πρέπει να ικανοποιούνται οι ακόλουθοι περιορισμοί (constraints):

1. Περιορισμοί τύπου δεδομένων. Δηλαδή θα πρέπει οι τιμές των πεδίων να αντιστοιχούν στον τύπο δεδομένων του πεδίου.
2. Περιορισμοί κλειδιού. Κάθε κλειδί θα πρέπει να είναι μοναδικό.
3. Περιορισμοί ακεραιότητας οντοτήτων. Ποτέ δεν πρέπει να είναι NULL ένα πρωτεύον κλειδί.
4. Περιορισμοί αναφορικής ακεραιότητας. Ένα ξένο κλειδί θα πρέπει να αντιστοιχεί σε υπαρκτό κλειδί του πίνακα στον οποίο αναφέρεται.

Μόνο οι εντολές που αλλάζουν τα δεδομένα της βάσης είναι πιθανό να παραβιάσουν κάποιον από τους παραπάνω περιορισμούς, δηλαδή οι INSERT, UPDATE και DELETE. Στις επόμενες ενότητες θα δούμε πως μπορεί η κάθε μια από αυτές τις εντολές να παραβιάσει τους περιορισμούς και πως μπορούμε να αντιμετωπίσουμε την κάθε περίπτωση.

## 5.1.1 Περιορισμοί στην εισαγωγή δεδομένων

Η INSERT μπορεί να παραβιάσει όλους τους περιορισμούς. Για παράδειγμα έστω ότι έχουμε τους πίνακες teacher και subject του προηγούμενου κεφαλαίου όπως φαίνονται στο ακόλουθο σχήμα:

| **teacher** | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Όνομα** | **Τύπος** | **Όνομα** | **Τύπος** | **Όνομα** | **Τύπος** | **Όνομα** | **Τύπος** |
| Ar_tautotitas | Char(7) | Name | Varchar(10) | Surname | Varchar(20) | speciality | Varchar(20) |

| **subject** | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Όνομα** | **Τύπος** | **Όνομα** | **Τύπος** | **Όνομα** | **Τύπος** | **Όνομα** | **Τύπος** |
| Id | Int | Title | Varchar(20) | Type | Enum('THEORY', 'LAB') | tar_tautotitas | Char(7) |

Ας δούμε πως θα μπορούσε η INSERT να παραβιάσει καθέναν από τους 4 περιορισμούς, δίνοντας 4 εσφαλμένα queries.

1. INSERT INTO subject VALUES(1,**1**,'THEORY', NULL, 'X256526'). Εδώ βάζουμε στο δεύτερο πεδίο τιμή ακεραίου ενώ είναι τύπου VARCHAR(20), επομένως παραβιάζουμε τον πρώτο περιορισμό για τον σωστό τύπο δεδομένων του πεδίου. Παραβιάζεται ο περιορισμός του τύπου δεδομένων.
2. INSERT INTO subject VALUES (**1**, 'MySQL', 'LAB', NULL, 'X256526'). Θεωρώντας ότι έχουμε ήδη εισάγει την πρώτη εγγραφή με id1, προσπαθούμε να εισάγουμε την δεύτερη εγγραφή δίνοντας πάλι το ίδιο id. Παραβιάζεται ο περιορισμός του κλειδιού.
3. INSERT INTO subject VALUES (**NULL**, 'C', 'THEORY', NULL, 'R586323'). Εδώ προσπαθούμε να εισάγουμε μια εγγραφή δίχως να δώσουμε τιμή στο κλειδί της. Παραβιάζεται ο περιορισμός ακεραιότητας οντοτήτων.
4. INSERT INTO subject VALUES (4,'English', 'THEORY', NULL, **'QWERTYU'**).Εδώ εισάγουμε ένα ξένο κλειδί που όμως δεν αντιστοιχεί πουθενά. Δηλαδή η τιμή 'QWERTYU' δεν υπάρχει σαν πρωτεύον κλειδί στον πίνακα teacher. Παραβιάζεται ο περιορισμός αναφορικής ακεραιότητας.

Ο συνήθης τρόπος χειρισμού της παραβίασης ενός περιορισμού από την INSERT είναι η απόρριψη της εισαγωγής. Δηλαδή το ΣΔΒΔ απαγορεύει στον χρήστη να εισάγει μια εγγραφή που θα προκαλούσε παραβίαση των παραπάνω περιορισμών. Για αυτό και αν προσπαθήσουμε να εκτελέσουμε τις παραπάνω εντολές για τον πίνακα subject θα πάρουμε ένα μήνυμα λάθους.

Φυσικά για να λειτουργήσει σωστά αυτό, θα πρέπει να έχει γίνει σωστά και η δημιουργία του πίνακα. Κυρίως μας ενδιαφέρει να ορίζουμε σωστά τους τύπους δεδομένων των πεδίων, τα πρωτεύοντα

κλειδιά και τα ξένα κλειδιά. Ας προσέξουμε για παράδειγμα πάλι την εντολή CREATE που είχαμε ορίσει στο προηγούμενο κεφάλαιο για τον πίνακα subject:

```
CREATE TABLE subject(
        id INT NOT NULL,
        title VARCHAR(20),
        type ENUM('THEORY','LAB'),
        tar_tautotitas CHAR(7),
        PRIMARY KEY(id),
        FOREIGN KEY (tar_tautotitas) REFERENCES teacher(ar_tautotitas)
);
```

Προσέχουμε να δηλώνουμε τα πρωτεύονται κλειδιά ως NOT NULL, ώστε να ικανοποιούμε τον περιορισμό ακεραιότητας των οντοτήτων. Επίσης δηλώνουμε ποια πεδία είναι PRIMARY KEY ώστε να γνωρίζει το ΣΔΒΔ ποιο σε ποιο πεδίο θα πρέπει να εφαρμόσει περιορισμό κλειδιού και εφόσον υπάρχουν ξένα κλειδιά, όπως εδώ το πεδίο tar_tautotitas να το δηλώνουμε ως FOREIGN KEY ορίζοντας στο REFERENCES τον πίνακα και το πεδίο του στο οποίο αναφέρεται.

## 5.1.2 Περιορισμοί στην διαγραφή δεδομένων

Η εντολή DELETE μπορεί να παραβιάσει μόνο τον περιορισμό αναφορικής ακεραιότητας. Για παράδειγμα ας θεωρήσουμε την ακόλουθη εντολή:

```
DELETE FROM teacher WHERE ar_tautotitas='X256526'
```

Μολονότι αυτή η εντολή δεν επηρεάζει αρνητικά τον πίνακα teacher, ωστόσο δημιουργεί πρόβλημα στον πίνακα subject καθώς εκεί οι δύο πρώτες εγγραφές αναφέρονται στην εγγραφή του teacher που διαγράφτηκε μέσω του ξένου κλειδιού τους. Έτσι οι εγγραφές 1 και 2 στον subject θα έχουν στο πεδίου του tar_tautotitas την τιμή 'X256526', που όμως δεν θα αντιστοιχεί σε καμία τιμή του πίνακα teacher. Με αυτόν τον τρόπο παραβιάζεται ο περιορισμός αναφορικής ακεραιότητας.

Για να αποφύγουμε την παραβίαση του περιορισμού αναφορικής ακεραιότητας στην DELETE, έχουμε 3 επιλογές:

1. Να απαγορεύσουμε τις εσφαλμένες διαγραφές.
2. Να *διαδώσουμε* (cascade) την διαγραφή, διαγράφοντας και όλες τις εγγραφές που αναφέρονται στην εγγραφή που διαγράφτηκε.
3. Να τροποποιήσουμε τις τιμές των αναφορικών γνωρισμάτων που προκαλούν την παραβίαση.

Οι επιλογές αυτές ορίζονται κατά την δημιουργία ενός ξένου κλειδιού μέσω της επιλογής ON DELETE. Αν θέλουμε να απαγορεύσουμε την διαγραφή μιας εγγραφής που χρησιμοποιείται από έναν άλλο πίνακα ως ξένο κλειδί, τότε πρέπει στη δήλωση του ξένου κλειδιού να ορίσουμε την επιλογή ON

DELETE RESTRICT. Αν θέλουμε όμως να διαγραφούν και όλες οι εγγραφές που αναφέρονται σε αυτήν τότε θα χρησιμοποιήσουμε την ON DELETE CASCADE. Τέλος αν θέλουμε απλά να θέσουμε στην τιμή NULL τις εγγραφές που δείχνουν προς την διεγραμμένη εγγραφή τότε θα χρησιμοποιήσουμε την ON DELETE SET NULL.

Για παράδειγμα ας θεωρήσουμε τους πίνακες teacher και subject. Όπως είδαμε στο προηγούμενο κεφάλαιο το πεδίο tar_tautotitas του subject είναι ξένο κλειδί στον teacher. Οπότε αν θέλουμε με την διαγραφή μιας εγγραφής του teacher να γίνεται αυτόματα και η διαγραφή των εγγραφών του subject που δείχνουν προς αυτή, πρέπει να δηλώσουμε το ξένο κλειδί του subject με τον ακόλουθο τρόπο:

```
CREATE TABLE subject(
	id INT NOT NULL,
	title VARCHAR(20),
	type ENUM('THEORY','LAB'),
	tar_tautotitas CHAR(7),
	PRIMARY KEY(id),
	FOREIGN KEY (tar_tautotitas) REFERENCES teacher(ar_tautotitas) ON DELETE CASCADE
);
```

Η προεπιλεγμένη τιμή είναι η ON DELETE RESTRICT, οπότε απαγορεύεται η διαγραφή μιας εγγραφής που χρησιμοποιείται ως ξένο κλειδί.

## 5.1.3 Περιορισμοί στην ενημέρωση των δεδομένων

Τέλος στην εντολή UPDATE ισχύει ότι η τροποποίηση ενός πεδίου που δεν είναι ούτε πρωτεύον ούτε ξένο κλειδί συνήθως δεν προκαλεί προβλήματα. Το ΣΔΒΔ αρκεί μόνο να επιβεβαιώσει ότι η νέα τιμή είναι του σωστού τύπου δεδομένων και πεδίου ορισμού. Η τροποποίηση της τιμής του πρωτεύοντος κλειδιού είναι παρόμοια με τη διαγραφή μιας εγγραφής και την εισαγωγή μιας άλλης στη θέση της, διότι χρησιμοποιούμε το πρωτεύον κλειδί για να αναγνωρίζουμε τις εγγραφές. Επομένως τα όσα είδαμε προηγουμένως για την εισαγωγή και την διαγραφή ισχύουν και εδώ.

Η επιλογή για να ορίσουμε την συμπεριφορά του συστήματος όταν έχουμε ενημέρωση ενός πεδίου είναι η ON UPDATE. Και με τις επιλογές RESTRICT, CASCADE, SET NULL μπορούμε αντίστοιχα να απαγορεύσουμε την αλλαγή ενός πεδίου που χρησιμοποιείται ως ξένο κλειδί, ή να διαδώσουμε την αλλαγή ή να θέσουμε την τιμή NULL στο ξένο κλειδί. Η προεπιλεγμένη τιμή είναι η ON UPDATE CASCADE.

## 5.2 Όψεις (views) στην SQL

Ένα προχωρημένο χαρακτηριστικό των ΣΔΒΔ είναι η υποστήριξη των όψεων. Μια όψη (view) είναι ένας *εικονικός πίνακας* που προκύπτει από τους άλλους πίνακες της βάσης. Συνήθως η όψη δεν αντιστοιχεί σε κάποιο πραγματικό αρχείο πίνακα στο ΣΔΒΔ αλλά υπάρχει στη μνήμη του υπολογιστή και χρησιμοποιείται για να διευκολύνει την λειτουργία του συστήματος όταν απαιτείται τακτική πρόσβαση σε σύνθετα δεδομένα που προκύπτουν από join σε πίνακες.

Μια όψη δημιουργείται με συνδυασμό των εντολών CREATE και SELECT. Για παράδειγμα με βάση τους πίνακες teacher και subject θα μπορούσαμε να ορίσουμε την ακόλουθη όψη:

```
CREATE VIEW lab_teachers AS

SELECT * FROM teacher, subject WHERE ar_tautotitas=tar_tautotitas AND type='LAB'
```

Προσέχουμε ότι το εσωτερικό SELECT είναι ένα JOIN που ανακτά τους καθηγητές που έχουν αναλάβει εργαστηριακά μαθήματα. Η CREATE VIEW δημιουργεί έναν πίνακα με το όνομα lab_teachers που περιέχει τους καθηγητές που διδάσκουν εργαστηριακά μαθήματα. Το ενδιαφέρον χαρακτηριστικό είναι ότι η όψη θα έχει πάντα ενημερωμένα τα δεδομένα της ακόμα και αν τα δεδομένα των αρχικών πινάκων αλλάξουν μετά τη δημιουργία της. Δηλαδή αν εισάγουμε έναν ακόμη καθηγητή στον πίνακα teacher και του αναθέσουμε ένα εργαστηριακό μάθημα, τότε αυτόματα θα προστεθεί και στην όψη lab_teachers. Ή αν ένα μάθημα αλλάξει από εργαστηριακό και γίνει θεωρία, τότε αυτόματα θα αφαιρεθούν από την όψη οι καθηγητές που το διδάσκουν.

Μπορούμε να διαγράψουμε μια όψη χρησιμοποιώντας την εντολή DROP ή να την τροποποιήσουμε χρησιμοποιώντας την ALTER.

Μπορούμε να χρησιμοποιούμε μια όψη μέσα σε SELECT queries όπως θα χρησιμοποιούσαμε και έναν οποιονδήποτε άλλο πίνακα. Ωστόσο δεν είναι πάντα εφικτό να κάνουμε αλλαγές στα δεδομένα μιας όψης χρησιμοποιώντας τις εντολές UPDATE, DELETE, INSERT. Για να μπορούμε να ενημερώνουμε τα δεδομένα μιας όψης θα πρέπει αυτή να έχει μια ένα προς ένα αντιστοιχία με τα δεδομένα των πινάκων, δηλαδή να είναι εφικτό από την αλλαγή της όψης να συμπεράνουμε ποιες αλλαγές πρέπει να γίνουν στους πραγματικούς πίνακες από τους οποίους προέρχονται τα δεδομένα. Για παράδειγμα ας θεωρήσουμε την ακόλουθη όψη που βασίζεται στον πίνακα catalog του κεφαλαίου 3:

```
CREATE VIEW distinct_names AS SELECT DISTINCT name FROM catalog
```

Εδώ θέλουμε να πάρουμε τα μοναδικά ονόματα από τον πίνακα των ονομάτων. Οπότε η όψη θα έχει τα ακόλουθα δεδομένα:

| **distinct_names** |
|---|
| **Name** |
| NIKOLAOS |

| IOANNIS |
| --- |
| XRHSTOS |

Έστω λοιπόν ότι θέλουμε να εκτελέσουμε την ακόλουθη εντολή:

```
UPDATE distinct_names SET name='SPIROS' WHERE name='NIKOLAOS'
```

Εφόσον η όψη δεν είναι ένας πραγματικός πίνακας, κάθε αλλαγή των δεδομένων της όψης θα πρέπει να αντιστοιχίζεται στα δεδομένα των πινάκων από τους οποίους προέρχεται, επομένως στον πίνακα catalog. Ωστόσο στον catalog υπάρχουν πολλές εγγραφές με το όνομα 'NIKOLAOS' έτσι το ΣΔΒΔ δεν μπορεί να βγάλει συμπέρασμα για το ποια εγγραφή να ενημερώσει.

# 6 Διασύνδεση βάσεων δεδομένων και διαχείριση ΣΔΒΔ

## 6.1 Διασύνδεση προγραμμάτων με βάσεις δεδομένων

### 6.1.1 Διεπαφή προγραμματισμού εφαρμογών (API)

Έως τώρα έχουμε εξετάσει πως μπορούμε να επικοινωνούμε με ένα ΣΔΒΔ μέσω της γλώσσας SQL. Ωστόσο ο τελικός μας στόχος δεν είναι απλώς και μόνο να επικοινωνούμε με το ΣΔΒΔ αλλά να χρησιμοποιούμε την βάση δεδομένων μας στα πλαίσια μιας εφαρμογής.

Όπως γνωρίζουμε οι εφαρμογές λογισμικού αναπτύσσονται προγραμματίζοντας σε κάποια γλώσσα προγραμματισμού. Επομένως για να χρησιμοποιήσουμε την βάση θα πρέπει το πρόγραμμα που θα γράψουμε να έχει τη δυνατότητα να επικοινωνεί με την βάση. Υπάρχουν διάφοροι τρόποι για να επικοινωνήσει μια εφαρμογή λογισμικού με τη βάση. Συνήθως το ΣΔΒΔ λειτουργεί ως server (βλ. Διάγραμμα 1 στο πρώτο κεφάλαιο) και η εφαρμογή ως client. Οπότε για να επικοινωνήσει ο client με τον server χρησιμοποιεί ένα καθορισμένο σύνολο συναρτήσεων ως διεπαφή λογισμικού. Αυτό είναι γνωστό ως API (Application Programming Interface) και δεν είναι τίποτε άλλο από μια βιβλιοθήκη λογισμικού που μας παρέχει ένα σύνολο συναρτήσεων για τον χειρισμό της βάσης. Για παράδειγμα αν χρησιμοποιούμε τη γλώσσα προγραμματισμού C για την ανάπτυξη της εφαρμογής μας και τη MySQL για τη βάση μας, τότε ο ακόλουθος πίνακας παρουσιάζει ένα δείγμα από το API της MySQL για τη C.

| Function | Περιγραφή |
|---|---|
| **mysql_close()** | Closes a server connection. |
| **mysql_errno()** | Επιστρέφει τον κωδικό λάθους για το τελευταίο σφάλμα που έχει συμβεί. |
| **mysql_fetch_row()** | Ανακτά την επόμενη γραμμή από το αποτέλεσμα. |
| **mysql_free_result()** | Απελευθερώνει την μνήμη που χρησιμοποιεί ένα result set |
| **mysql_init()** | Αρχικοποίηση ενός MYSQL struct. |
| **mysql_query()** | Εκτέλεση ενός sql query που δίνεται σαν ένα c string. |
| **mysql_real_connect()** | Σύνδεση στον MYSQL server |
| **mysql_use_result()** | Αρχικοποιεί μια ανάκτηση των αποτελεσμάτων ενός query γραμμή ανά γραμμή |

Το ακόλουθο πρόγραμμα C δίνει μια εικόνα πως θα μπορούσαμε να γράψουμε ένα πρόγραμμα που συνδέεται στη βάση, εκτελεί ένα query και παρουσιάζει τα αποτελέσματα που αυτό επιστρέφει:

```
#include <stdio.h>
#include <mysql.h> //Σε αυτό το header υπάρχουν οι συναρτήσεις που θέλουμε

#define HOST "localhost"
#define USER "user1"
#define PASSWD "pwd1"
#define DB_NAME "demo"

int main(int argc, char* argv[]){
  MYSQL *mysql = NULL;
  MYSQL_RES *result;
  MYSQL_ROW row;

  mysql = mysql_init(NULL);//Αρχικοποίηση του mysql

  if(mysql == NULL)
    return 1; // Init failed.
```

```
  /*Εδώ γίνεται η πραγματική σύνδεση στη βάση δίνοντας ως παραμέτρους την διεύθυνση του server, το όνομα και τον κωδικό του χρήστη, το όνομα της βάσης και την tcp πόρτα που ακούει ο server. Το αποτέλεσμα είναι η σωστή αρχικοποίηση της δομής mysql*/
  if(!mysql_real_connect(mysql, HOST, USER, PASSWD, DB_NAME, MYSQL_PORT, NULL, 0))
    return 1; // Connection failed.

/*Χρησιμοποιώντας την mysql δομή που έχουμε δημιουργήσει, στέλνουμε ένα query στη βάση*/
  if(mysql_query(mysql, "select t_id, name from test1") != 0)
    return 1; // Select failed

/*Παίρνουμε το αποτέλεσμα του query σε μια MYSQL_RES δομή*/
  result = mysql_use_result(mysql);

/*Από το αποτέλεσμα ανακτούμε μια μια τις γραμμές του*/
  while((row = mysql_fetch_row(result)))
    printf("%s %s\n", row[0], row[1]);

  if(mysql_errno(mysql)) // Error when fetching rows
    return 1;

/*Απελευθερώνουμε τη μνήμη που καταλαμβάνει το αποτέλεσμα του query*/
  mysql_free_result(result);

/*Δεν ξεχνούμε να κλείσουμε τη σύνδεση με τον server*/
  mysql_close(mysql);
  return 0;
}
```

Το παραπάνω πρόγραμμα δείχνει ενδεικτικά μερικές βασικές λειτουργίες που μπορούμε να κάνουμε για να συνδεθούμε με την mysql και να ανακτήσουμε τα δεδομένα που θέλουμε. Ενδεικτικά αναφέρουμε ότι τα βασικά βήματα είναι:

1. Συμπερίληψη (include) του mysql.h αρχείου.
2. Αρχικοποίηση μιας MYSQL δομής.
3. Σύνδεση με τον mysql server.
4. Αποστολή ενός query.
5. Λήψη του αποτελέσματος σε ένα MYSQL_RES.
6. Ανάκτηση των γραμμών του αποτελέσματος μιας μιας.
7. Απελευθέρωση της μνήμης που καταλαμβάνει το αποτέλεσμα του query.
8. Κλείσιμο της σύνδεσης με τον server.

Στο εγχειρίδιο της mysql μπορούμε να βρούμε μια πλήρη λίστα των εντολών που υπάρχουν στο API. Επίσης η MySQL (όπως και τα περισσότερα ΣΔΒΔ) παρέχουν API για αρκετές γλώσσες προγραμματισμού όπως C, Java, .NET, C++, PHP, PERL, Python κ.ά.

## 6.1.2 Embedded SQL

Στον προηγούμενο υπότιτλο είδαμε έναν τρόπο για να συνδεθεί κάποιος σε μια βάση δεδομένων από το δικό του πρόγραμμα. Παρατηρήσαμε όμως ότι η προσέγγιση που παρουσιάστηκε αφορούσε ένα συγκεκριμένο ΣΔΒΔ (MySQL) και μια συγκεκριμένη γλώσσα προγραμματισμού (C). Μολονότι στην πράξη αυτό είναι εντελώς ικανοποιητικό, ωστόσο θέλουμε να έχουμε έναν μηχανισμό που θα μας παρέχει μια καθιερωμένη διεπαφή ανεξάρτητα από την βάση που χρησιμοποιούμε. Όπως δηλαδή η SQL είναι ένα πρότυπο που εφαρμόζεται σε όλες τις βάσεις δεδομένων έτσι θέλουμε να έχουμε και μια πρότυπη προγραμματιστική διεπαφή που θα χρησιμοποιείται για όλες τις βάσεις δεδομένων. Στην παραπάνω λογική υπακούει η embedded SQL (ενσωματωμένη SQL).

Η embedded SQL είναι μια ελαφρά τροποποίηση της SQL που έχουμε δει ως τώρα και όπως και το API που είδαμε πριν, μας δίνει την δυνατότητα να συνδεθούμε με τη βάση, να στέλνουμε queries και να παίρνουμε αποτελέσματα. Το ιδιαίτερο χαρακτηριστικό της embedded SQL είναι ότι αντί να καλούμε κάποιες συναρτήσεις, γράφουμε ανάμεικτα κώδικα SQL και C. Κατόπιν το ανάμεικτο πρόγραμμα οδηγείται στην είσοδο ενός pre-compiler που το μετατρέπει σε ένα νόμιμο C πρόγραμμα και τέλος το αποτέλεσμα του pre-compiler στην είσοδο του C compiler. Στη συνέχεια θα δούμε αναλυτικά πως μπορούμε να συντάξουμε ένα απλό C πρόγραμμα με embedded SQL. Δυστυχώς η MySQL δεν υποστηρίζει επίσημα την embedded SQL οπότε τα παραδείγματα που θα κάνουμε δεν μπορούν να γίνουν compile στην MySQL.

Κάθε εντολή στην Embedded SQL αρχίζει με το πρόθεμα EXEC SQL και τελειώνει υποχρεωτικά με ; Το πρώτο βήμα είναι να συνδεθούμε με τον server, οπότε πρέπει να γνωρίζουμε την ip διεύθυνση του μηχανήματος, το όνομα της βάσης, το όνομα του χρήστη και τον κωδικό του. Γράφουμε την εντολή (η εντολή αυτή επιδέχεται και άλλους τρόπους σύνταξης, ωστόσο μια συνήθης χρήση παρουσιάζεται εδώ):

```
EXEC SQL CONNECT TO <όνομα βάσης>@<διεύθυνση server> USER <όνομα χρήστη> IDENTIFIED BY <κωδικός χρήστη>;
```

Αντίστοιχα όταν θέλουμε να κλείσουμε τη σύνδεση με τον server γράφουμε την εντολή:

```
EXEC SQL DISCONNECT;
```

Έχοντας δημιουργήσει μια σύνδεση μπορούμε να στέλνουμε SQL εντολές. Οι εντολές που στέλνουμε θα εκτελεστούν μόνο μετά την εντολή επιβεβαίωσης που είναι η:

```
EXEC SQL COMMIT;
```

Οπότε προκειμένου να δημιουργήσουμε έναν πίνακα θα εκτελέσουμε μια εντολή:

```
EXEC SQL CREATE TABLE <όνομα πίνακα>(<όνομα πεδίου1> <τύπος πεδίου1>, <όνομα πεδίου2> <τύπος πεδίου2>…);

EXEC SQL COMMIT;
```

Με όμοιο τρόπο συντάσσουμε τις εντολές INSERT, DELETE, UPDATE. Προσέχουμε πως στο τέλος κάθε συνόλου εντολών που θέλουμε να εκτελεστεί πρέπει να προσθέτουμε την εντολή COMMIT. Ωστόσο υπάρχει η δυνατότητα να γίνεται αυτόματα COMMIT η κάθε εντολή, αν δώσουμε την εντολή:

```
EXEC SQL SET AUTOCOMMIT TO ON;
```

#### 6.1.2.1 Μεταβλητές

Ας δούμε τώρα πως θα μπορούσαμε να ανακτήσουμε κάποια δεδομένα από τη βάση, στέλνοντας μια εντολή SELECT:

```
EXEC SQL BEGIN DECLARE SECTION;

VARCHAR name[10];

int cost;

EXEC SQL END DECLARE SECTION;


EXEC SQL SELECT name, cost INTO :name, :cost FROM product WHERE id=1;
```

Αρχικά έχουμε ορίσει ένα block που αρχίζει με την εντολή EXEC SQL BEGIN DECLARE SECTION και τελειώνει με την εντολή EXEC SQL END DECLARE SECTION. Μέσα σε αυτή την περιοχή δηλώνουμε μεταβλητές που θα χρησιμοποιηθούν για να μεταφέρουμε δεδομένα από τη βάση στην C ή και το αντίστροφο. Οι μεταβλητές δηλώνονται με κανονικό κώδικα C. Στη συνέχεια εκτελείται μια τροποποιημένη εντολή SELECT, όπου μετά την λίστα των πεδίων εισάγουμε την λέξη INTO και απαριθμούμε τις δηλωμένες μεταβλητές που θέλουμε να αποθηκεύσουν τις τιμές των πεδίων. Έτσι η εντολή που έχουμε γράψει θα ανακτήσει από τη βάση τα πεδία name και cost της πρώτης εγγραφής του πίνακα product και στη συνέχεια θα αποθηκεύσει αυτά τα πεδία στις μεταβλητές name και cost. Σημειώνουμε ότι οι μεταβλητές θα πρέπει πρώτα να έχουν δηλωθεί μέσα στο DECLARE SECTION και ότι στην SELECT θα πρέπει να εισάγονται με τον χαρακτήρα : (άνω κάτω τελεία). Προφανώς μπορούμε να δηλώσουμε όσες C μεταβλητές θέλουμε, απλά εκείνες που θα χρησιμοποιηθούν για την

μεταφορά των δεδομένων μεταξύ του προγράμματος και της βάσης, θα πρέπει να δηλώνονται εντός του DECLARE SECTION.

Ενδιαφέρον παρουσιάζουν οι μεταβλητές που δηλώνονται ως τύπος VARCHAR. Φυσικά αυτός δεν είναι τύπος δεδομένων της C, για αυτό και ο pre-compiler θα μετατρέψει την μεταβλητή name σύμφωνα με το ακόλουθο πρότυπο:

```
struct{short len; char * arr[10]} name;
```

Βλέπουμε δηλαδή ότι η δομή έχει δύο πεδία, το πρώτο απλά ορίζει το πλήθος των χαρακτήρων που χρησιμοποιούνται (δηλ. το πραγματικό μήκος που θα έχει η τιμή του name και το δεύτερο έναν πίνακα χαρακτήρων που θα έχει τους χαρακτήρες του πεδίου.

Ιδιαίτερη μεταχείριση χρειάζονται οι τιμές NULL. Προκειμένου να γνωρίζουμε αν μια τιμή είναι NULL στη βάση, απαιτείται να δηλώσουμε μια *μεταβλητή ένδειξης* (indicator variable). Για παράδειγμα αν θέλουμε να ανακτήσουμε μόνο το πεδίο name αλλά να γνωρίζουμε όταν αυτό είναι NULL θα πρέπει να γράψουμε τον ακόλουθο κώδικα:

```
EXEC SQL BEGIN DECLARE SECTION;

VARCHAR name;

int name_indicator;

EXEC SQL END DECLARE SECTION;


EXEC SQL SELECT name  INTO :name :name_indicator FROM product WHERE id=1;
```

Βλέπουμε δηλαδή ότι για ένα πεδίο (name) αντιστοιχίζουμε δύο μεταβλητές (name, name_indicator) εκ' των οποίων η δεύτερη είναι int. Έτσι η πρώτη μεταβλητή (name) θα αποθηκεύσει την τιμή του πεδίου αλλά η δεύτερη θα χρησιμοποιηθεί ως σημαία που θα υποδηλώνει την κατάσταση του πεδίου. Συγκεκριμένα αν είναι 0 τότε η τιμή δεν είναι NULL. Αν είναι μικρότερη του 0 τότε είναι NULL και αν είναι μεγαλύτερη του 0 τότε η τιμή του πεδίου έχει περικοπεί για να χωρέσει στον τύπο δεδομένων που παρέχουμε.

Ωστόσο τα προηγούμενα παραδείγματα ανακτούν μια εγγραφή μόνο. Στην περίπτωση που το αποτέλεσμα έχει πολλές εγγραφές, τότε πρέπει να χρησιμοποιήσουμε *δείκτες* (cursors). Για παράδειγμα αν θέλουμε να πάρουμε το όνομα και το κόστος όλων των προϊόντων, τότε θα γράψουμε:

```
EXEC SQL BEGIN DECLARE SECTION;

VARCHAR name;

int cost;

EXEC SQL END DECLARE SECTION;


EXEC SQL DECLARE deiktis CURSOR FOR SELECT name, cost FROM product;

EXEC SQL OPEN deiktis;
```

```
while(sqlca.sqlcode==0){
        ...
        EXEC SQL FETCH NEXT FROM deiktis INTO :name, :cost;
}
EXEC SQL CLOSE deiktis;
```

Παρατηρούμε ότι μετά τη δήλωση των μεταβλητών, ορίζουμε έναν δείκτη προσδιορίζοντας το query πάνω στο οποίο αυτός θα ενεργεί. Μετά «ανοίγουμε» τον δείκτη και στη συνέχεια ακολουθεί ένας κανονικός C βρόχος, μέσα στον οποίο χρησιμοποιούμε μια εντολή fetch που ενεργεί επί του δείκτη που έχουμε ορίσει. Κάθε φορά που εκτελείται η εντολή EXEC SQL FETCH NEXT FROM <όνομα δείκτη>, ανακτούμε και την επόμενη εγγραφή του αποτελέσματος. Τα πεδία της κάθε εγγραφής που ανακτάται αποθηκεύονται όπως και προηγουμένως στις μεταβλητές name, cost. Τέλος όταν διαβάσουμε όλα τα πεδία κλείνουμε τον δείκτη.

Επίσης προσέχουμε ότι ο έλεγχος στον βρόχο γίνεται επί της μεταβλητής sqlca. Η sqlca είναι μια δομή (struct) που τα πεδία της προσδιορίζουν την κατάσταση της βάσης. Στην προκειμένη περίπτωση μπορούμε να ελέγχουμε αν το πεδίο sqlcode είναι ίσο με μηδέν, οπότε και πρέπει να σταματήσουμε την ανάκτηση των δεδομένων.

### 6.1.2.2 Δυναμική embedded SQL

Έως τώρα τα παραδείγματα embedded SQL που έχουμε δει αφορούσαν στατικά queries. Δηλαδή αποφασίζουμε για την εντολή που θα στείλουμε στη βάση, όταν γράφουμε το πρόγραμμα. Ωστόσο αυτό δεν είναι πάντα εφικτό, πολλές φορές το ακριβές query που θα δώσουμε εξαρτάται από δεδομένα που δίνει ο τελικός χρήστης ή γενικά είναι γνωστά μόνο κατά την διάρκεια της εκτέλεσης του προγράμματος. Για παράδειγμα αν έχουμε μια μηχανή αναζήτησης τότε η λέξη της αναζήτησης – που προφανώς θα αποτελέσει κριτήριο επιλογής στο query που θα στείλουμε – δίνεται από τον τελικό χρήστη την στιγμή που τρέχει το πρόγραμμα. Προκειμένου να ξεπεραστεί αυτό το πρόβλημα υπάρχει η *δυναμική embedded SQL*. Η βασική ιδέα είναι να προσδιορίζουμε *παραμέτρους* σε κάθε query που θα ορίζονται κατά τον χρόνο εκτέλεσης του προγράμματος. Για παράδειγμα:

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR name;
int cost;
char *statement=”SELECT name, cost FROM product WHERE id=?”;
EXEC SQL END DECLARE SECTION;


EXEC SQL PREPARE myStatement FROM :stmt;
EXEC SQL EXECUTE myStatement USING 12 INTO  name, cost;
EXEC SQL DEALOCATE PREPARE myStatement;
```

Όπως φαίνεται από το παραπάνω παράδειγμα ετοιμάζουμε μια μεταβλητή string που περιέχει το query που θέλουμε να εκτελεστεί. Η διαφοροποίηση όμως έγκειται στο ότι αντί να προσδιορίσουμε για ποιο συγκεκριμένο id θέλουμε να γίνει η ανάκτηση, στη θέση της τιμής βάζουμε ένα αγγλικό ερωτηματικό. Με αυτόν τον τρόπο δημιουργούμε μια *παράμετρο* στο query. Όταν έχουμε queries με παραμέτρους τότε η εκτέλεσή τους γίνεται σε δυο βήματα. Πρώτα πρέπει να *προετοιμάσουμε* το query (μέσω της εντολής EXEC SQL PREPARE …) και στη συνέχεια να το εκτελέσουμε ορίζοντας τιμή για αυτή την παράμετρο. Αυτό γίνεται στο query:

```
EXEC SQL EXECUTE myStatement USING 12 INTO  name, cost;
```

Η τιμή που ακολουθεί μετά το USING ορίζει την τιμή της παραμέτρου, ενώ τα αποτελέσματα από την εκτέλεση του query αποθηκεύονται στις μεταβλητές που ορίζουμε (name, cost). Έτσι η παραπάνω διαδικασία ισοδυναμεί με την εκτέλεση του query:

```
SELECT name, cost FROM product WHERE id=12;
```

#### 6.1.2.3 Δημιουργία εκτελέσιμου από κώδικα C με embedded SQL

Έχοντας ολοκληρώσει την συγγραφή του κώδικα, έχουμε ένα αρχείο με ανάμεικτο κώδικα C και embedded SQL. Όπως αναφέραμε και στην αρχή του κυρίου υπότιτλου, αυτό το αρχείο δεν μπορεί να μεταγλωττιστεί απ' ευθείας από τον C compiler. Πρέπει πρώτα να περάσει από τον embedded SQL pre-compiler ώστε να μετατραπεί σε ένα νόμιμο C πρόγραμμα που θα περιέχει μόνο εντολές C. Στη συνέχεια το πρόγραμμα που θα δημιουργήσει ο pre-compiler θα εισαχθεί κανονικά στον C compiler. Φυσικά προσοχή χρειάζεται προκειμένου να συμπεριλάβουμε τις σωστές βιβλιοθήκες στον linker.

Επίσης έχουμε την δυνατότητα να συμπεριλάβουμε αρχεία embedded SQL μέσα στον κώδικά μας, με παρόμοιο τρόπο όπως συμπεριλαμβάνουμε αρχεία στην C. Αυτό μπορεί να γίνει με την εντολή:

```
EXEC SQL INCLUDE <όνομα αρχείου>;
```

Η διαφορά αυτής της εντολής include από την παραδοσιακή που έχει η C:

```
#include <όνομα αρχείου>
```

Είναι ότι η πρώτη θα αναγκάσει τον SQL pre-compiler να κάνει parsing και το αρχείο που ορίζουμε, ενώ η δεύτερη όχι.

## 6.2 XML

Πέρα από τις βάσεις δεδομένων που είναι ισχυρά δομημένες μορφές οργάνωσης της πληροφορίας, υπάρχουν και *ημιδομημένες* μορφές οργάνωσης της πληροφορίας. Σε αυτές τις μορφές διατηρείται η έννοια του πίνακα και των πεδίων του, αλλά όχι μέσω μιας αυστηρής γλώσσας ορισμού όπως η SQL. Ένα πολύ επιτυχημένο παράδειγμα ημιδομημένων δεδομένων είναι η XML (eXtensible Markup Language). Το σκεπτικό στην XML είναι η δημιουργία ενός απλού αρχείου κειμένου όπου η πληροφορία θα δομείται μέσα σε *ετικέτες* ή αλλιώς tags. Για παράδειγμα θα μπορούσαμε να ορίσουμε την ακόλουθη πρόταση xml:

```
<name language='greek'>Σπύρος</name>
```

Το συγκριτικό πλεονέκτημα της XML είναι η ευελιξία της. Για παράδειγμα η παραπάνω πρόταση ορίζει ένα tag που λέγεται name, έχει τιμή Σπύρος και έχει μια ιδιότητα language που έχει την τιμή greek. Αυτό θα μπορούσε κάλλιστα να σημαίνει ότι έχουμε το ελληνικό όνομα Σπύρος. Δεν υπάρχει κάποιος περιορισμός στον τρόπο που θα δομήσουμε τα δεδομένα, αρκεί βέβαια να ειμαστε σύμφωνοι με τους κανόνες σύνταξης της XML. Όπως φαίνεται και από αυτό το απλό παράδειγμα κάθε tag εμπίπτει στο πρότυπο

```
<όνομα ιδιότητα1=τιμή1 ιδιότητα2=τιμή2 ….> περιεχόμενο </όνομα>
```

Και αυτό μπορεί να εφαρμοστεί αναδρομικά, δηλαδή το περιεχόμενο ενός tag να είναι ένα άλλο tag κ.ο.κ. Έτσι μπορεί κάποιος να συντάξει ένα κείμενο δίχως να δεσμεύεται από κάποιο συγκεκριμένο πρότυπο αλλά ωστόσο να παραμένει δομημένο. Για παράδειγμα παρακάτω φαίνεται ένας απλός πίνακας με δεδομένα παρόμοια με αυτά που έχουμε δει στα παραδείγματα των προηγούμενων κεφαλαίων:

```
<table_structure name="catalog">
	<field Field="id" Type="int(11)" Null="NO" Key="PRI" Default="" Extra="" />
	<field Field="surname" Type="varchar(30)" Null="YES" Key="" Extra="" />
	<field Field="name" Type="varchar(10)" Null="YES" Key="" Extra="" />
</table_structure>


<table_data name="catalog">
<row>
	<field name="id">1</field>
	<field name="surname">ABROS</field>
	<field name="name">NIKOLAOS</field>
</row>
<row>
	<field name="id">2</field>
	<field name="surname">ABRAMOPOULOS</field>
	<field name="name">IOANNIS</field>
</row>
</table_data>
```

Ο παραπάνω κώδικας XML αποτελεί τμήμα του αυτόματα παραγόμενου κώδικα όταν η MySQL μετατρέπει τον πίνακα catalog σε δομή XML. Βλέπουμε ότι στο πάνω τμήμα γίνεται η δήλωση της δομής του πίνακα όπου προσδιορίζεται ο τύπος του κάθε πεδίου και διάφορες ιδιότητές του, ενώ στο κάτω τμήμα φαίνονται τα δεδομένα του πίνακα, όπως αυτά εκχωρούνται στα διάφορα πεδία. Το

ουσιαστικό θέμα είναι ότι στον παραπάνω κώδικα δεν έχουμε καμία υποχρέωση να τον συντάξουμε με αυτόν τον τρόπο. Αν για παράδειγμα χρησιμοποιηθεί ένα άλλο εργαλείο αυτόματης δημιουργίας XML, τότε για τα δεδομένα του ίδιου πίνακα θα πάρουμε τον ακόλουθο κώδικα XML:

```
<RECORDS>
<RECORD>
      <id>1</id>
      <surname>ABROS</surname>
      <name>NIKOLAOS</name>
</RECORD>
<RECORD>
      <id>2</id>
      <surname>ABRAMOPOULOS</surname>
      <name>IOANNIS</name>
</RECORD>
</RECORDS>
```

Αυτή ακριβώς η ευελιξία της XML είναι και η δύναμή της. Μέσω της XML μπορούμε εύκολα να μεταφέρουμε δεδομένα μεταξύ εντελώς διαφορετικών ΣΔΒΔ, διαφορετικών λειτουργικών συστημάτων, διαφορετικών αρχιτεκτονικών hardware και γενικά μεταξύ ακραίων ανομοιογενών περιβαλλόντων. Χαρακτηριστικό πεδίο εφαρμογής της XML αποτελεί το web, όπου διαφορετικοί ιστόχωροι μπορούν να διασυνδέονται μεταφέροντας δεδομένα σε μορφή XML σύμφωνα με την τεχνολογία των web services. Φυσικά η XML δεν μπορεί να αντικαταστήσει ουτε καν να συγκριθεί σε απόδοση με τις δυνατότητες και την ταχύτητα που παρέχουν τα παραδοσιακά ΣΔΒΔ όταν τα χρησιμοποιούμε για ένα συγκεκριμένο σύστημα.

## 6.3 Εντολές διαχείρισης του ΣΔΒΔ MySQL

Έως τώρα έχουμε εξετάσει τρόπους για να επικοινωνούμε με ΣΔΒΔ μέσω της γλώσσας SQL. Ωστόσο πολλές φορές ένας διαχειριστής ενός ΣΔΒΔ θέλει να κάνει πολύ περισσότερα πράγματα που αφορούν το τεχνικό θέμα των βάσεων. Σε αυτόν τον υπότιτλο θα εστιάσουμε την προσοχή μας στο πως μπορεί κάποιος διαχειριστεί λογαριασμούς χρηστών, να πάρει και να φορτώσει αντίγραφα ασφαλείας. Σε όλες τις περιπτώσεις η εφαρμογή θα γίνεται στο ΣΔΒΔ MySQL.

### 6.3.1 Διαχείριση χρηστών στη MySQL

Όπως έχουμε αναφέρει και στο πρώτο κεφάλαιο, κάθε διασύνδεση με μια βάση δεδομένων γίνεται για λογαριασμό ενός χρήστη, ακριβώς όπως η σύνδεση σε ένα πολυχρηστικό λειτουργικό σύστημα γίνεται

μέσω λογαριασμών χρηστών. Έτσι κατά την δημιουργία μιας βάσης ορίζουμε ποιοι χρήστες θα έχουν πρόσβαση σε αυτή και με τι είδους δικαιώματα ο καθένας. Για παράδειγμα ένας χρήστης μπορεί να έχει δικαίωμα μόνο ανάγνωσης από τους πίνακες (δηλ. μόνο SELECT) κάποιος άλλος να έχει δικαίωμα προσθήκης δεδομένων (INSERT), αλλαγής της δομής της βάσης (ALTER) ή να έχει δικαίωμα να δημιουργεί νέους λογαριασμούς χρηστών.

Σε κάθε περίπτωση υπάρχει πάντα ο super user που δεν διαγράφεται ποτέ από την βάση και έχει πάντα όλα τα δικαιώματα πρόσβασης σε όλες τις βάσεις. Το όνομά του είναι **root** και ο κωδικός πρόσβασής του καθορίζεται κατά την εγκατάσταση του ΣΔΒΔ.

Η δημιουργία ενός νέου χρήστη γίνεται με την εντολή:

```
CREATE USER <όνομα χρήστη> IDENTIFIED BY <κωδικός χρήστη>
```

Έτσι αν θέλουμε να δημιουργήσουμε ένα χρήστη με όνομα labuser και κωδικό trustno1, θα γράφαμε την ακόλουθη εντολή:

```
CREATE USER 'labuser' IDENTIFIED BY 'trustno1'
```

Στη συνέχεια μπορούμε να δώσουμε συγκεκριμένα δικαιώματα σε αυτόν τον χρήστη με την εντολή GRANT. Για παράδειγμα αν θέλουμε να δώσουμε όλα τα δικαιώματα χρήσης της βάσης dblab στον χρήστη labuser, θα γράφαμε την ακόλουθη εντολή:

```
GRANT ALL ON dblab.* TO labuser;
```

Ο προσδιορισμός ALL δίνει στον χρήστη όλα τα δικαιώματα στη βάση και ο προσδιορισμός dblab.* προσδιορίζει όλους τους πίνακες. Αν για παράδειγμα θέλουμε να δώσουμε μόνο δικαιώματα ανάγνωσης σε όλους τους πίνακες της βάσης, τότε θα γράφαμε:

```
GRANT SELECT ON dblab.* TO labuser;
```

Αν θέλουμε να αφαιρέσουμε όλα τα δικαιώματα χρήσης της βάσης dblab από τον χρήστη labuser, θα χρησιμοποιήσουμε την εντολή REVOKE:

```
REVOKE ALL PRIVILEGES ON dblab.* FROM labuser;.
```

Αν θέλουμε να αλλάξουμε τον κωδικό ενός χρήστη, θα χρησιμοποιήσουμε την εντολή SET PASSWORD. Για παράδειγμα αν θέλουμε να αλλάξουμε τον κωδικό του labuser σε trustonlyme, θα γράψουμε:

```
SET PASSWORD FOR labuser = password('trustonlyme');
```

Τέλος πρέπει να σημειώσουμε πως ο κάθε χρήστης έχει δικαιώματα όχι μόνο επί ορισμένων βάσεων και πινάκων αλλά ανάλογα και με το είδος της σύνδεσης που θα μπορεί να κάνει. Για παράδειγμα ένας χρήστης θα μπορούσε να έχει πρόσβαση μόνο τοπικά από το μηχάνημα που είναι εγκατεστημένος ο server ή θα μπορούσε να συνδέεται και με απομακρυσμένη πρόσβαση.

### 6.3.2 Παρουσίαση των βάσεων, των χρηστών και των πινάκων στην MySQL

Ένα από τα βασικότερα πράγματα που θέλει να κάνει ο διαχειριστής ενός ΣΔΒΔ είναι να βλέπει ποιες βάσεις υπάρχουν στο σύστημα, ποιοι χρήστες, ποιοι πίνακες, τι πεδία έχει ο καθένας κ.ο.κ. Για να δούμε ποιες βάσεις και ποιοι πίνακες υπάρχουν στη βάση χρησιμοποιούμε την εντολή show. Για παράδειγμα για να δούμε ποιες βάσεις υπάρχουν θα γράψουμε την ακόλουθη εντολή:

```
show databases;
```

Κατόπιν για να χρησιμοποιήσουμε μια βάση χρησιμοποιούμε την εντολή use. Για παράδειγμα για να χρησιμοποιήσουμε την βάση dblab, θα γράψουμε την ακόλουθη εντολή:

```
use dblab;
```

Αν θέλουμε να δούμε ποιους πίνακες έχει τότε:

```
show tables;
```

Αν θέλουμε να δούμε ποια πεδία έχει ο πίνακας product, τότε θα χρησιμοποιήσουμε την εντολή describe ως εξής:

```
describe product;
```

Αν πάλι θέλουμε να δούμε τι δικαιώματα πρόσβασης έχει ένας χρήστης θα χρησιμοποιήσουμε την εντολή show grants. Έστω για παράδειγμα πως θέλουμε να δούμε τα δικαιώματα χρήσης του χρήστη labuser:

```
show grants for labuser;
```

### 6.3.3 Αντίγραφα ασφαλείας και διασύνδεση με άλλες βάσεις για την MySQL

Τέλος θα εξετάσουμε πως μπορούμε να πάρουμε αντίγραφα ασφαλείας (backup), πώς να φορτώσουμε δεδομένα σε μια βάση από αντίγραφα ασφαλείας και πώς να μεταφέρουμε δεδομένα μεταξύ διαφορετικών βάσεων.

Η λήψη αντιγράφων ασφαλείας είναι προφανώς ένα κρίσιμης σημασίας ζήτημα για τον διαχειριστή μιας βάσης δεδομένων. Οι τρόποι για να επιτευχθεί αυτό είναι πολλοί και με διάφορα πλεονεκτήματα – μειονεκτήματα ο καθένας. Μολονότι η διαδικασία αποθήκευσης των δεδομένων μιας βάσης θα μπορούσε να γίνει με εντολές SQL, ωστόσο λόγω της πολυπλοκότητας της διαδικασίας χρησιμοποιούνται εξωτερικά βοηθητικά προγράμματα. Κατά συνέπεια υπάρχουν πολλές επιλογές για το ποιο εργαλείο θα χρησιμοποιηθεί και προφανώς το καθένα έχει τα δικά του χαρακτηριστικά ενώ νέα εργαλεία παρουσιάζονται συνεχώς. Εμείς εδώ ωστόσο θα επικεντρώσουμε την προσοχή μας στο στάνταρ εργαλείο που παρέχεται μαζί με την mysql και λειτουργεί από την γραμμή εντολών, το

mysqldump. Η λειτουργία του mysqldump βασίζεται στις κατάλληλες παραμέτρους κατά την κλήση του προγράμματος.

Για παράδειγμα έστω ότι θέλουμε να δημιουργήσουμε ένα backup της βάσης dblab σε ένα αρχείο με το όνομα dblab.sql. Η κλήση του mysqldump θα έχει τις ακόλουθες παραμέτρους (το πρόθεμα shell> υποδηλώνει την γραμμή εντολής στην κονσόλα εντολών και δεν είναι μέρος της εντολής):

```
shell> mysqldump dblab –u labuser –p > dblab.sql
```

Με αυτή την εντολή ορίζουμε την βάση που θέλουμε να πάρουμε backup και τον χρήστη που έχει δικαιώματα χρήστης σε αυτήν (μετά θα μας ζητήσει και το password). Το αποτέλεσμα της εντολής θα καταγραφεί στο αρχείο dblab.sql. Το αρχείο αυτό είναι ουσιαστικά ένα σύνολο από sql εντολές (που έχουν παραχθεί αυτόματα από το mysqldump) που αφορούν την δημιουργία των πινάκων της βάσης και το γέμισμά τους με τα δεδομένα που έχουν.

Αν αργότερα θέλουμε να επαναφέρουμε τη βάση από το backup που έχουμε κρατήσει τότε θα χρησιμοποιήσουμε τον MySQL client ως εξής:

```
shell>> more dblab.sql | mysql –u labuser –p dblab
```

Με αυτόν τον τρόπο φορτώνουμε πάλι την βάση με τα δεδομένα που έχουμε αποθηκεύσει. Βέβαια για να λειτουργήσει σωστά αυτή η εντολή θα πρέπει να έχουμε ήδη δημιουργήσει μια βάση με το όνομα που ορίζουμε (dblab στην προκειμένη περίπτωση).

Τέλος πρέπει να σημειώσουμε πως υπάρχουν πολλά εργαλεία που επιτρέπουν την διασύνδεση της MySQL με άλλες βάσεις δεδομένων ή και ανάκτηση των δεδομένων μιας βάσης από αρχεία XML, ή και από αρχεία κειμένου όπου τα πεδία ενός πίνακα διαχωρίζονται με ειδικούς χαρακτήρες. Για παράδειγμα το εργαλείο MySQL Migration Toolkit που δίνεται μαζί με τα εργαλεία γραφικού περιβάλλοντος για την MySQL έχει εύχρηστους οδηγούς που ολοκληρώνουν μια διαδικασία μεταφοράς δεδομένων.

# Βιβλιογραφία

[1]. R. Elmasri, S.B. Navathe, “Fundamentals of Database Systems” 3rd edition, 2000.

[2]. MySQL 5.0 Reference Manual.

[3]. PostgreSQL 8.2.3 Documentation.

[4]. DB2 Information Center.