Α.Τ.Ε.Ι. Κρήτης
Τμήμα
Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Πτυχιακή Εργασία

Θέμα:
Σχεσιακές Βάσεις Δεδομένων, η γλώσσα SQL και Αποθηκευμένες Διαδικασίες.

Εισηγητής: Δρ. Αναστάσιος Καμπουρέλης
Επιμέλεια : Ευαγγελία Μπαγκέρη
Α.Μ. 1153

Περιεχόμενα:

Σελ.:

Ευχαριστήρια

Για αρχή θα ήθελα να ευχαριστήσω τον διδάσκων καθηγητή Δρ. Αναστάσιο Καμπουρέλη, πρώτον για την υπομονή του και δεύτερον για την πολύτιμη βοήθεια που μου έδωσε για την ολοκλήρωση της πτυχιακής μου εργασίας.

Τέλος θα ευχαριστήσω την οικογένεια μου για την συμπαράσταση και την υπομονή που έδειξε μέχρι την εκπλήρωση της εργασίας!

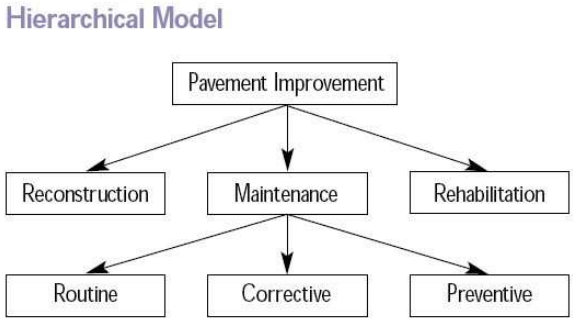
Στοιχεία:
Ευαγγελία Μπαγκέρη
τηλ.: 6976934176
e-mail: epp_1153@yahoo.gr

Κεφάλαιο 1 :
Σχεσιακές Βάσεις Δεδομένων

1.1 Τα τέσσερα Μοντέλα

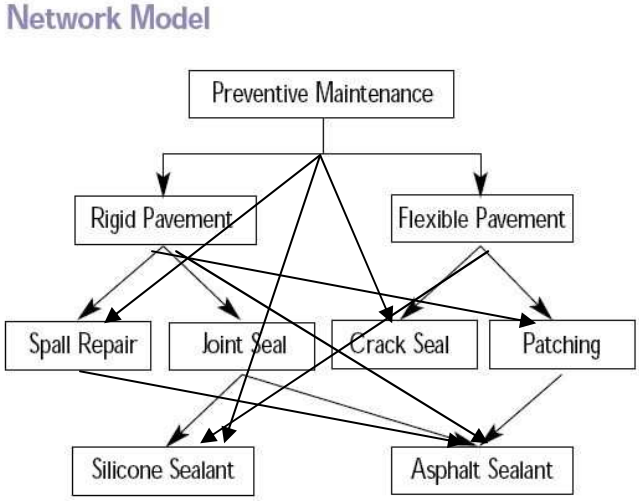
1.1.1 Το Ιεραρχικό Μοντέλο

Χρησιμοποιεί την δομή του δένδρου. Αποδείχτηκε ότι είναι περιοριστικό στη χρήση γιατί δεν μπορούν όλες οι δομές των πληροφοριών να μοντελοποιηθούν με τη δομή του δένδρου.



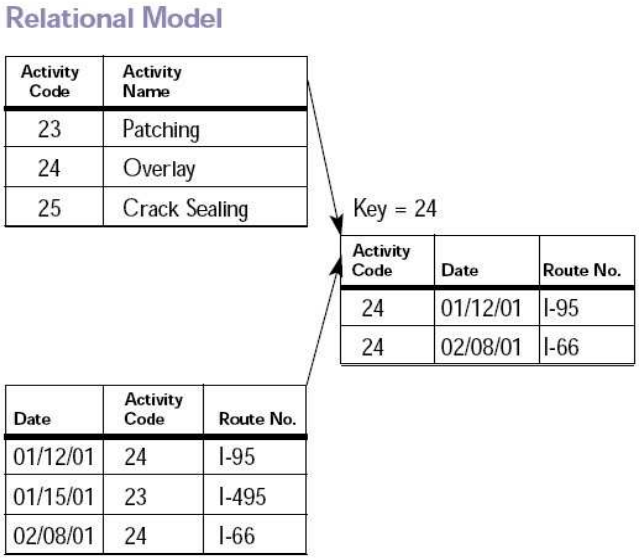
1.1.2 Το Μοντέλο Δικτύου

Το μοντέλο αυτό χρησιμοποιεί αυθαίρετα γραφήματα. Αποδείχτηκε ότι είναι σύνθετο με περίπλοκα διαγράμματα (μερικές φορές ονομαζόμενα διαγράμματα «σπαγγέτι») .



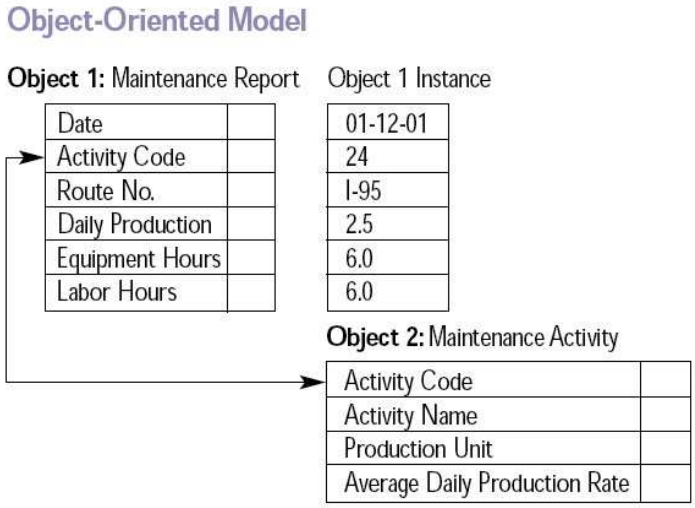
1.1.3 Το Σχεσιακό Μοντέλο

Σ' αυτό το μοντέλο η πληροφορία αποθηκεύεται σε πίνακες (σχέσεις). Χρησιμοποιώντας την Σχεσιακή Άλγεβρα (προβολές, ενώσεις κτλ.) μπορούμε να επιτύχουμε όλες τις αναγκαίες συνδέσεις. Το μοντέλο αυτό χρησιμοποιεί την γλώσσα SQL και είναι το βασικότερο μοντέλο για τη δουλειά μας.



1.1.4 Μοντέλο βασισμένο σε αντικείμενα

Αυτό είναι ένα σχετικά καινούριο μοντέλο δομής. Οι ερευνητές προσπαθούν να ενοποιήσουν την προσέγγιση από την μεριά του αντικειμένου με την προσέγγιση από την μεριά του πίνακα.



1.2 Το Σχεσιακό Μοντέλο Δεδομένων

Ένα σύστημα διαχείρισης βάσης δεδομένων (ΣΔΒΔ) (*database management system (DBMS)*) αποτελείται από ένα σύνολο δεδομένων και προγράμματα πρόσβασης στα δεδομένα αυτά. Το σύνολο των δεδομένων καλείται βάση δεδομένων (*database*). Στόχος του ΣΔΒΔ είναι η εύκολη και γρήγορη χρήση και ανάκτηση των δεδομένων. Η διαχείριση των δεδομένων περιλαμβάνει:

- τον ορισμό δομών για τη αποθήκευση των δεδομένων
- τον ορισμό μεθόδων για τη διαχείριση των δεδομένων

Ο ορισμός της δομής της βάσης δεδομένων βασίζεται σε ένα μοντέλο δεδομένων το οποίο ορίζει τον τρόπο που περιγράφονται τα δεδομένα, οι σχέσεις τους, η σημασία τους και οι περιορισμοί πάνω στα δεδομένα αυτά.

Το **σχεσιακό μοντέλο (*relational model*)** δεδομένων παριστάνει δεδομένα και τις σχέσεις τους ως ένα σύνολο πινάκων. Κάθε πίνακας (*table*) αποτελείται από στήλες (*columns*) με μοναδικά ονόματα. Μια γραμμή (*row*) του πίνακα παριστάνει μια σχέση (*relationship*) ανάμεσα σε ένα σύνολο από τιμές. Οι πίνακες που ακολουθούν θα χρησιμοποιηθούν σε όλη τη διάρκεια της εργασίας και παριστάνουν πελάτες μια τράπεζας (PEL) και τις συναλλαγές τους με την τράπεζα αυτή (SYN).

PEL

ID	NAME	CITY
100	GEORGE	ATHENS
101	MARY	CHANIA
102	THANOS	HERAKLION
103	ELENH	ATHENS
104	PETER	SALONIKA

SYN

AA	EURO	ID
1	100	100
2	150	100
3	200	102

Τύποι δεδομένων

Τα δεδομένα κάθε στήλης ενός πίνακα πρέπει να έχουν ένα συγκεκριμένο τύπο. Οι βασικοί τύποι που υποστηρίζονται από την SQL είναι οι παρακάτω:
- BIT : *Ναι ή Όχι*
- CURRENCY: *Τιμή που παριστάνει με ακρίβεια αριθμούς.*
- DATETIME: *Χρόνος*
- SINGLE: *Αριθμός κινητής υποδιαστολή μονής ακρίβειας*
- DOUBLE: *Αριθμός κινητής υποδιαστολή διπλής ακρίβειας*
- SHORT: *Ακέραιος 2 byte*
- LONG: *Ακέραιος 4 byte*
- TEXT: *Κείμενο μέχρι 255 χαρακτήρες*
- LONGTEXT: *Κείμενο μέχρι 1.2GB*
- BLOB*: binary large object, περιεχόμενα σε οποιαδήποτε μορφή πχ εικόνες*

Βασικές εντολές επεξεργασίας πινάκων

Οι εντολές θα αναλυθούν εκτενέστερα στο δεύτερο κεφάλαιο. Εδώ απλά κάνουμε μια εισαγωγή σε αυτές.

- Δημιουργία πινάκων:

Νέοι πίνακες δημιουργούνται με την εντολή CREATE TABLE.

```
CREATE TABLE PEL (ID INTEGER [10], NAME VARCHAR[30], CITY VARCHAR [30])
```

- Αλλαγές σε πίνακες:

Η εντολή ALTER TABLE επιτρέπει την προσθήκη νέων στηλών ή τη διαγραφή υπαρχόντων. Η προσθήκη γίνεται με την εντολή ADD COLUMN και η διαγραφή με την εντολή DROP COLUMN.

```
ALTER TABLE PEL
ADD COLUMN PHONE INTEGER [30]
```

- Προθήκη στοιχείων

Η προσθήκη δεδομένων σε έναν πίνακα γίνεται με την εντολή INSERT INTO.

```
INSERT INTO PEL (ID, NAME, CITY)
VALUES (105, "JOHN", "ATHENS")
```

- Επιλογή στοιχείων

Από τις πιο βασικές εντολές είναι η SELECT , η οποία μας επιτρέπει απλά να επιλέγουμε δεδομένα από έναν ή περισσότερους πίνακες ή και με κάποια κριτήρια.

```
SELECT ID, NAME
FROM PEL
WHERE CITY = ATHENS
```

Τα πεδία μπορούν να είναι ονόματα πεδίων ή συναρτήσεις της SQL πάνω σε πεδία. Τέτοιες συναρτήσεις είναι :
Avg: Μέσος όρος
Count: Μέτρηση
Min: Ελάχιστο
Max: Μέγιστο
Sum: Σύνολο

Ο αστερίσκος ως ορισμός πεδίου επιλέγει όλα τα πεδία.
Τα κριτήρια αναζήτησης είναι εκφράσεις πάνω στα πεδία. Ορισμένοι βασικοί τελεστές είναι οι:

- αριθμητικοί τελεστές: + - * / mod
- σύγκρισης: < <= ⇔ > >= = <> like
- λογικοί: and or not

- Αλλαγή

Η αλλαγές στα στοιχεία γίνονται με την εντολή UPDATE σύμφωνα με τη σύνταξη:

```
UPDATE PEL
SET CITY = 'PATRA'
WHERE ID = 4;
```

- Διαγραφή

Η διαγραφή γραμμών ενός πίνακα γίνεται με την εντολή DELETE:

```
DELETE
FROM PEL
WHERE CITY = "ATHENS"
```

Χαρακτηριστικά των Σχέσεων

Μια Σχεσιακή Βάση Δεδομένων δεν είναι τίποτα περισσότερο από ένα σύνολο από Σχέσεις.

Μια σχέση (relation) είναι ένας πίνακας τιμών. Κάθε στήλη στον πίνακα έχει επικεφαλίδα (όνομα) και ονομάζεται πεδίο (field). Κάθε γραμμή ονομάζεται πλειάδα και παριστά τα χαρακτηριστικά μιας οντότητας στο μοντέλο.

Το κενό (NULL), χρησιμοποιείται για την αναπαράσταση μιας τιμής στη βάση που είναι μη εφαρμόσιμη (non-applicable) ή άγνωστη (unknown). Εδώ το null δεν χρησιμοποιείται όπως σε μια γλώσσα προγραμματισμού, για παράδειγμα δεν γίνονται συγκρίσεις ανάμεσα σε δεδομένα και στο null.

Πχ. Η τιμή για τον PHONE_NUMBER για κάποιον που δεν έχει τηλέφωνο. Η τιμή για την ADDRESS κάποιου που δεν έχει προσκομίσει διεύθυνση.

Περιορισμοί

- Δομικοί περιορισμοί

Υπάρχουν 3 ειδών περιορισμοί που είναι έμφυτοι στο μοντέλο:

1. Κλειδί (Key)
2. Ακεραιότητα Οντότητας (Entity Integrity)
3. Αναφορική Ακεραιότητα (Referential Integrity)

Υπάρχουν επίσης 3 είδη ρητών περιορισμών:

1. Πεδίο τιμών (Domain)
2. Στηλών (Column)
3. Ορισμένων από το χρήστη (User - Defined)

Άλλος ρητός περιορισμός είναι και οι Συναρτησιακές Εξαρτήσεις (Functional Dependences)

- Περιορισμοί Κλειδιών

Στο σχεσιακό μοντέλο ισχύουν τα διαφορετικά κλειδιά, όπως και το μοντέλο E-R (Entity-Relationship). Το κλειδί είναι ιδιότητα του Σχεσιακού Σχήματος, οπότε ισχύει για όλα τα στιγμιότυπα – σχέσεις.

1. Ένα γνώρισμα ενός τύπου Οντοτήτων/Συσχετίσεων για το οποίο κάθε οντότητα/συσχέτιση στο σύνολο πρέπει να έχει μοναδική τιμή είναι **ΚΛΕΙΔΙ** (Key) ή (superkey)
2. Ένα **Υποψήφιο Κλειδί** (Candidate key) είναι ένα ελάχιστο κλειδί (δηλαδή, κανένα υποσύνολο των γνωρισμάτων του δεν είναι και αυτό κλειδί).
3. Το **Κύριο/Πρωτεύον Κλειδί** (Primary key) είναι ένα από τα υποψήφια κλειδιά που ορίζεται σαν προσδιοριστής των πλειάδων του πίνακα. Επιλέγεται από τον προγραμματιστή της β.δ.
4. Ένα **Ξένο Κλειδί** (Foreign key) είναι ένα πεδίο ή συνδυασμός πεδίων σε έναν πίνακα που παίρνει τιμές από το πεδίο κλειδί ενός άλλου πίνακα.

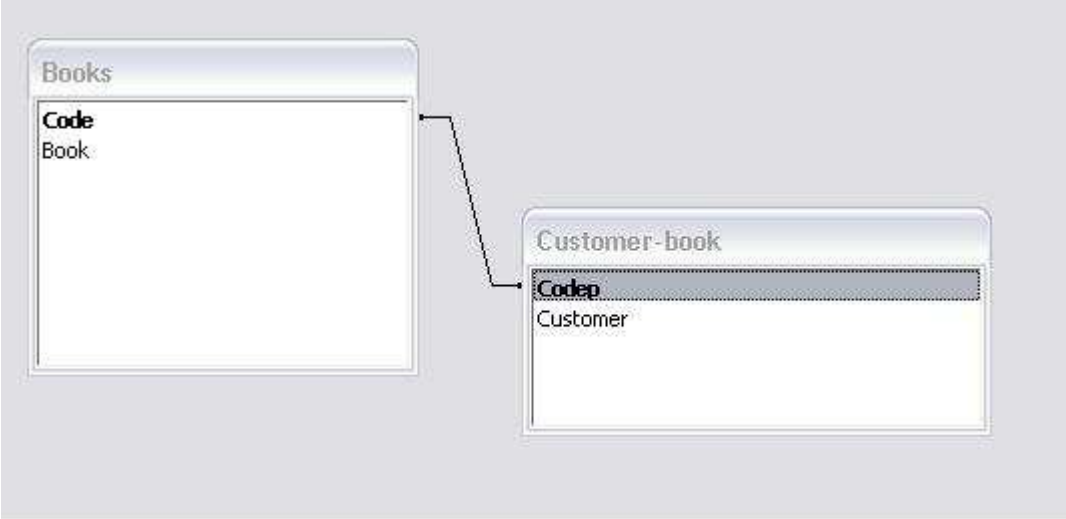
Συσχετίσεις πινάκων

Οι συσχετίσεις πινάκων είναι οι συσχετίσεις (relationships) βάση των οποίων δημιουργούμε συνδέσεις μεταξύ πινάκων. Έχουμε τέσσερα είδη συσχετίσεων:

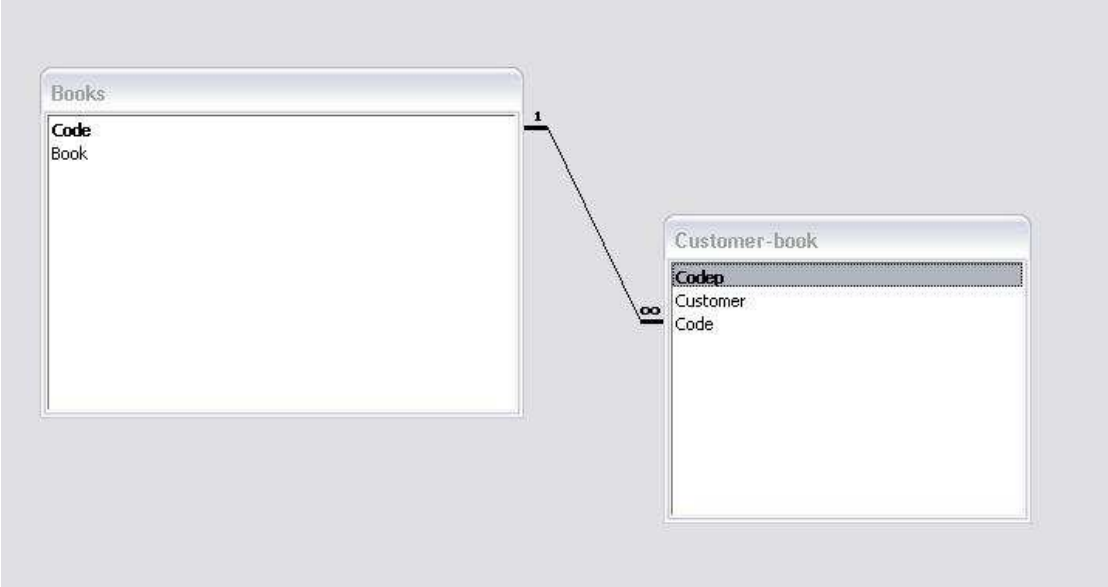
- 1 προς πολλά
- Πολλά προς 1
- Πολλά προς πολλά
- 1 προς 1

- **<u>1 - ∞</u> : ένα προς πολλά**

Η συσχέτιση αυτή επιτρέπει την ένωση ενός στοιχείου μιας οντότητας με πολλά στοιχεία άλλης οντότητας. Παρόμοιο είναι και το **<u>∞ - 1:</u>** πολλά προς ένα.

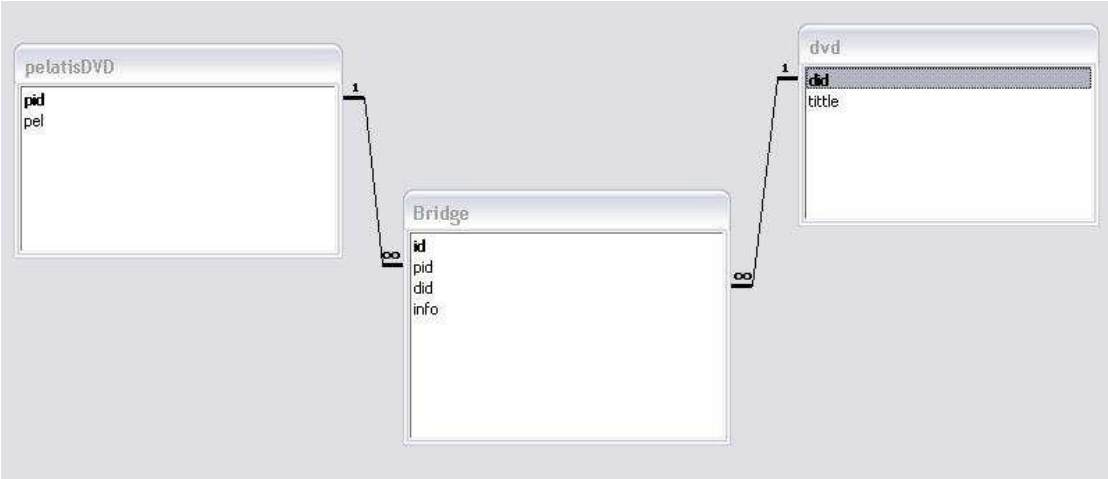


Εδώ έχουμε δύο πίνακες, τον Books, με δύο πεδία, τον κωδικό και τον τίτλο των βιβλίων και τον Customer-book που περιλαμβάνει τον κωδικό και το ονοματεπώνυμο των πελατών.
Πως επιτυγχάνουμε αυτήν τη συσχέτιση στους πίνακες ?
Στον δεύτερο πίνακα προσθέτω ένα πεδίο όπου θα κρατάει τις τιμές του Code του πρώτου, εκεί θα είναι απλά **ξένο κλειδί** και θα επιτρέπει τις επαναλήψεις (προσθήκη ευρετηρίου με διπλότυπα), κάτι που δεν γίνεται στον Books. Σ' αυτήν τη συσχέτιση έχουμε Referential Integrity (αναφορική ακεραιότητα όπου στο «∞» δεν μπορώ να έχω μεμονωμένα πεδία ενώ στο «1» μπορώ.)



- **∞ - ∞ : Πολλά προς πολλά**

Σαν παράδειγμα συσχέτισης πολλά προς πολλά μπορούμε να θεωρήσουμε μια ΒΔ ενός DVD Club. Κάθε πελάτης μπορεί να πάρει πολλά DVD καθώς και το ίδιο DVD μπορεί να δοθεί σε πολλούς πελάτες. Η συσχέτιση πολλά προς πολλά μπορεί να υλοποιηθεί με τον **πίνακα «γέφυρα» - bridge table**. Όπως βλέπουμε και στο επόμενο σχήμα, έχουμε τρεις πίνακες : τον PELATISDVD, όπου κρατάει τον κωδικό και το όνομα των πελατών, τον πίνακα DVD με πεδία τον κωδικό και τον τίτλο των ταινιών και τον πίνακα Bridge, με πεδία τους κωδικούς των πελατών και των ταινιών καθώς και πληροφορίες και έναν κωδικό id. Μεταξύ των πινάκων και του πίνακα γέφυρα δημιουργείται συσχέτιση 1 προς πολλά.

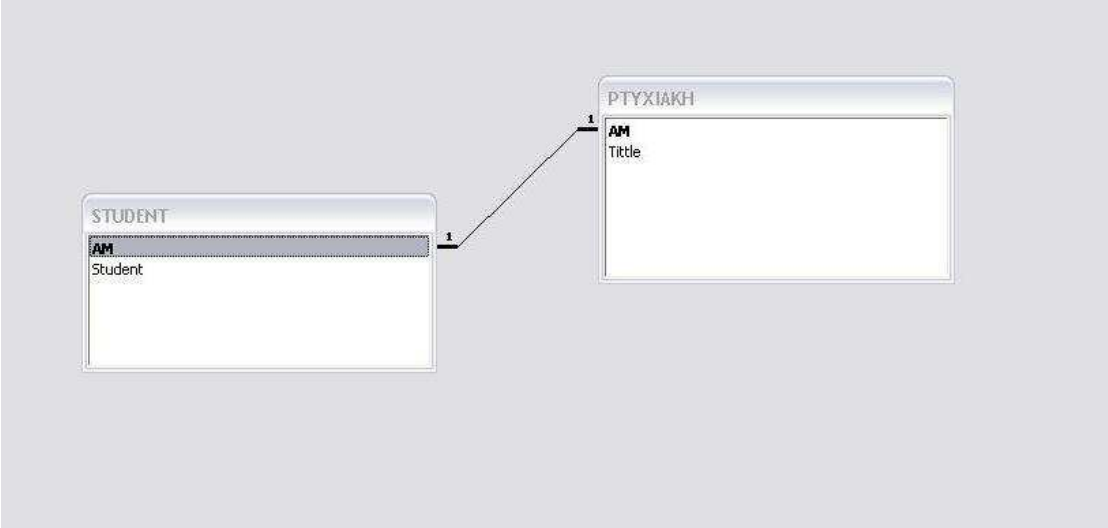


Οπότε πετύχαμε συσχέτιση ∞ - ∞. Αυτό μπορεί να θεωρηθεί ∞ - ∞ = (1 - ∞) + (∞ - 1).

- **1 – 1 : Ένα προς ένα**

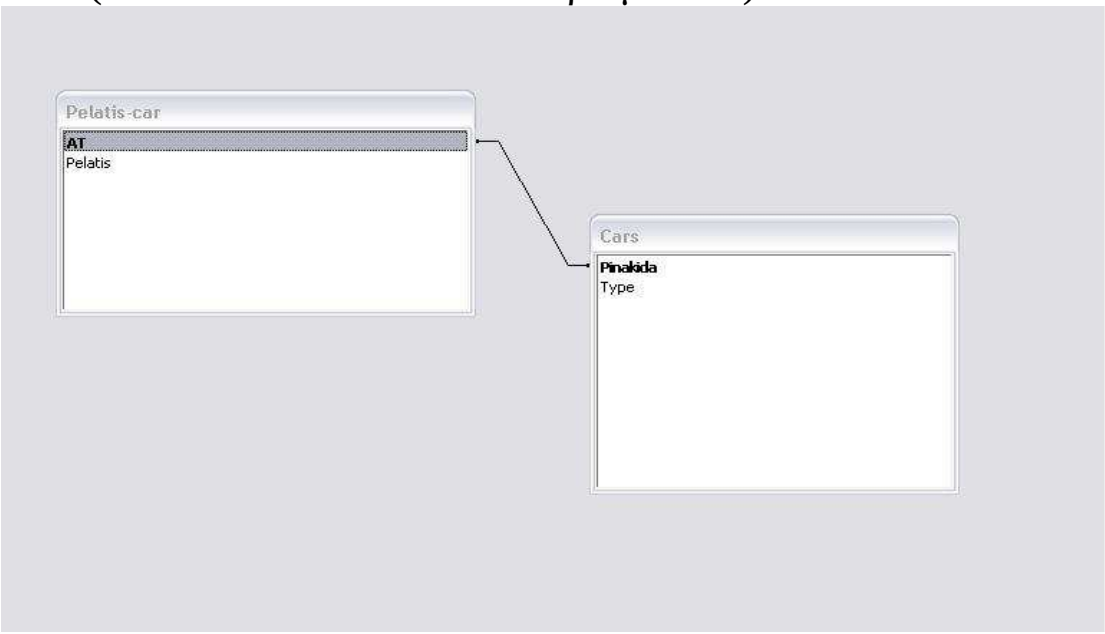
Αυτή η συσχέτιση ποτέ δεν επιτρέπει ενώσεις δύο διαφορετικών στοιχείων της οντότητας. Υπάρχουν δύο περιπτώσεις συσχέτισης 1 – 1.

- Ίδιο κλειδί: όταν δύο πίνακες έχουν το ίδιο πεδίο ως primary key.

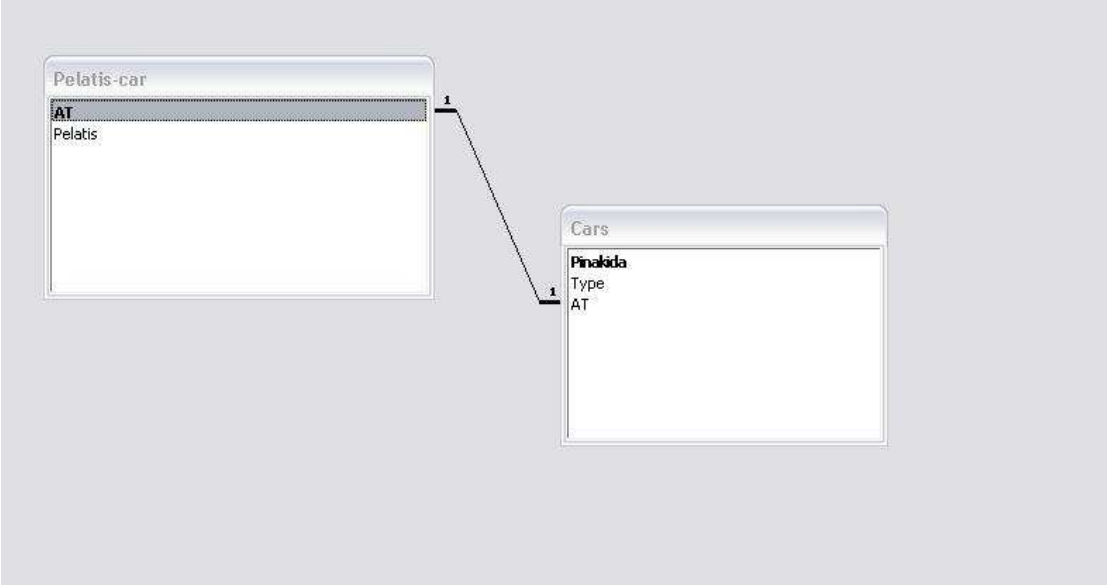


Εδώ έχουμε δύο πίνακες, τον Student, με δύο πεδία, το AM και το ονοματεπώνυμο των φοιτητών, και τον Ptyxiaki που περιλαμβάνει το AM και τον τίτλο της πτυχιακής. Και στους δύο πίνακες δεν επιτρέπονται επαναλήψεις, εφόσον το AM κάθε φοιτητή δεν μπορεί να δοθεί σε άλλον, καθώς επίσης στον κάθε φοιτητή αντιστοιχεί μια πτυχιακή εργασία.

- Different keys: όταν δύο πίνακες έχουν συσχέτιση 1-1 με διαφορετικά primary keys. Αυτό λύνεται ως ειδική περίπτωση 1- ∞ (όπου θα αποκλειστούν οι αριθμοί <2.)



Εδώ έχουμε δύο πίνακες, τον Pelatis-cars, με δύο πεδία, τον αριθμό ταυτότητας και το ονοματεπώνυμο των πελατών, και τον Cars που περιλαμβάνει την πινακίδα και τον τύπο του αυτοκινήτου.
Στον δεύτερο πίνακα προσθέτω το πεδίο AT, που είναι **ξένο κλειδί (εφόσον είναι primary key στον πρώτο πίνακα)**. Για να αποφύγουμε τις επαναλήψεις θα προσθέσουμε στο πεδίο AT του δεύτερου πίνακα ένα INDEX 1-1. Αυτό το ευρετήριο δεν επιτρέπει διπλότυπα.



Κεφάλαιο 2 :
Η γλώσσα SQL(Structured Query Language)

2.1 Η ιστορία της SQL και των Σχεσιακών Βάσεων Δεδομένων

Η SQL βασίστηκε σε ιδέες που προτάθηκαν στις αρχές του 1970 από τον ερευνητή Tedd Codd των εργαστηρίων IBM San Josi Research Laboratories. Δημιουργήθηκε με σκοπό να παρέχει μια ημι – φυσική γλώσσα για το IBM System Relational database system. Αρχικά οι σχεσιακές βάσεις δεδομένων είχαν πολλούς και πιο αποδοτικούς ανταγωνιστές π.χ. συστήματα διαχείρισης δεδομένων τα οποία βασίζονταν σε δικτυακά μοντέλα δεδομένων. Ωστόσο, όλο και πιο αποδοτικά σχεσιακά συστήματα άρχισαν να εμφανίζονται στα τέλη του 1980 κι αυτό οδήγησε στην επικράτηση των σχεσιακών βάσεων δεδομένων και της SQL.

Διάλεκτοι της SQL.

Δυστυχώς υπάρχουν πολλές διάλεκτοι της SQL που διαφέρουν ελαφρώς από το ένα σύστημα στο άλλο. Οι κυριότερες είναι οι τυποποιημένες ANSI/ISO SQL. Υπήρξαν δύο πρότυπα: το SQL–89 ή SQL1 και το SQL–92 ή SQL2. Οι περισσότερες εφαρμογές προσπαθούν να μείνουν πιστές στην SQL–92.

2.2 DML (Data Manipulation Language) DDL (Data Definition Language) DCL (Data Control Language)

Η δομημένη γλώσσα ερωταποκρίσεων SQL (Structured Query Language, SQL) χρησιμοποιείται για τη διαχείριση των δεδομένων μιας βάσης δεδομένων. Η διαχείριση περιλαμβάνει τόσο τη δημιουργία και μεταβολή των πινάκων της εφαρμογής, όσο και την καταχώρηση και ανάκτηση δεδομένων με βάση συγκεκριμένα κριτήρια επιλογής.Μια τυπική γλώσσα SQL περιλαμβάνει τις επόμενες δομικές μονάδες:

Γλώσσα ορισμού δεδομένων (Data Definition Language, DDL): Η γλώσσα αυτή περιλαμβάνει εντολές που μας επιτρέπουν να υλοποιήσουμε πίνακες, σχέσεις ανάμεσα σε πίνακες και γενικά όλη τη δομή μιας βάσης δεδομένων.

Γλώσσα χειρισμού δεδομένων (Data Manipulation Language, DML): Η γλώσσα αυτή επιτρέπει τη διαχείριση των δεδομένων της εφαρμογής, όπως την εισαγωγή, διαγραφή, ανάκτηση και τροποποίηση δεδομένων.

Ορισμός όψεων της βάσης (View Definition): Επιτρέπει τη δημιουργία όψεων της βάσης δεδομένων οι οποίες ορίζονται ως **εικονικοί πίνακες (virtual tables)** οι οποίοι περιέχουν δεδομένα από ένα ή περισσότερους πίνακες της βάσης.

Ορισμός εξουσιοδοτήσεων (Authorization): Επιτρέπει τη δημιουργία ομάδων χρηστών και την απόδοση διαφορετικών δικαιωμάτων πρόσβασης σε κάθε έναν από αυτούς, προκειμένου η κάθε ομάδα χρηστών, να διαχειρίζεται μόνο τα δικά της δεδομένα.

Διαχείρισης ακεραιότητας (Integrity): Επιτρέπει το λεπτομερή έλεγχο των δεδομένων που καταχωρούνται στη βάση, έτσι ώστε να μην παραβιάζονται οι **κανόνες ακεραιότητας (integrity constrains)** που έχουμε ορίσει και οι οποίοι όταν τηρούνται απομακρύνουν τον κίνδυνο καταχώρησης **ασυνεπών δεδομένων (inconsistent data).**

Η γλώσσα διαχείρισης δεδομένων (Data Control Language, DCL): χειρίζεται τις εξουσιοδοτήσεις των δεδομένων

DML Γλώσσα Διαχείρισης Δεδομένων

Η γλώσσα χειρισμού δεδομένων (Data Manipulation Language DML), επιτρέπει την διαχείριση των δεδομένων των πινάκων της βάσης, και πιο συγκεκριμένα, την εισαγωγή, διαγραφή και τροποποίηση των εγγραφών των πινάκων. Επιπλέον, έχουμε την δυνατότητα να ανακτήσουμε από τους πίνακες, δεδομένα τα οποία πληρούν κάποια κριτήρια. Η πραγματοποίηση αυτών των διαδικασιών, γίνεται χρησιμοποιώντας τις εντολές INSERT, DELETE, UPDATE και SELECT.

Η εντολή SELECT

Η δύναμη της SQL είναι η εντολή SELECT. Χρησιμοποιείται για να ανακαλεί δεδομένα κάνοντας ερωτήματα στην βάση δεδομένων, επιλέγοντας γραμμές από έναν πίνακα, που ταιριάζουν με συγκεκριμένα κριτήρια.

Η σύνταξη της SELECT:

```
SELECT <πεδία>
FROM <πίνακες>
WHERE <κριτήριο>
```

Ένα παράδειγμα , πολύ απλό είναι το επόμενο, χρησιμοποιώντας τους πίνακες μας.

PEL

ID	NAME	CITY
100	GEORGE	ATHENS
101	MARY	CHANIA
102	THANOS	HERAKLION
103	ELENH	ATHENS
104	PETER	SALONIKA

SYN

AA	EURO	ID
1	100	100
2	150	100
3	200	102

```
SELECT *
FROM PEL;
```

Η απάντηση είναι :

ID	NAME	CITY
100	GEORGE	ATHENS
101	MARY	CHANIA
102	THANOS	HERAKLION
103	ELENH	ATHENS
104	PETER	SALONIKA

Επιλέγει όλες τις γραμμές του πίνακα PEL. Ο αστερίσκος * μετά το SELECT κατευθύνει την sql να συμπεριλάβει όλες τις στήλες (πεδία) από τον πίνακα PEL. Όπως βλέπουμε αυτό το ερώτημα (query) αναπαράγει τον ίδιο τον πίνακα.
Εάν ξέρουμε όλες τις στήλες του πίνακα μπορούμε απλά να γράψουμε :

```
SELECT ID, NAME, CITY
FROM PEL;
```

Σε μια SELECT μπορούμε να ανταλλάξουμε ή να αντιγράψουμε στήλες.

```
SELECT CITY, ID, NAME, CITY, ID
FROM PEL;
```

Θα μας δώσει:

CITY	ID	NAME	Expr1000	Exprp1001
ATHENS	100	GEORGE	ATHENS	100
CHANIA	101	MARY	CHANIA	101
HERAKLION	102	THANOS	HERAKLION	102
ATHENS	103	ELENH	ATHENS	103
SALONIKA	104	PETER	SALONIKA	104

Τα περίεργα ονόματα μπορούν να διορθωθούν με την εντολή AS που θα δούμε παρακάτω.

Μπορούμε να επιλέξουμε μόνο μια στήλη:

```
SELECT CITY
FROM PEL;
```

CITY
ATHENS
CHANIA
HERAKLION
ATHENS
SALONIKA

Για να αποφύγουμε διπλές καταχωρίσεις χρησιμοποιούμε την παράμετρο DISTINCT.

```
SELECT DISTINCT CITY
FROM PEL;
```

CITY
ATHENS
CHANIA
HERAKLION
SALONIKA

– Υπάρχουν περιπτώσεις ένα ερώτημα να χρησιμοποιεί έναν ή περισσότερους πίνακες που να έχουν στήλες με το ίδιο όνομα. (όπως η στήλη ID στον πίνακα PEL και στον πίνακα SYN).
Για να ξεχωρίζουμε τις στήλες με το ίδιο όνομα χρησιμοποιούμε την εξής σύνταξη:
Πίνακας.Πεδίο δηλαδή PEL.id και SYN.id
– Κάποιες φορές χρειάζεται να αλλάξουμε το όνομα της στήλης. Του δίνουμε δηλαδή alias (ψευδώνυμο). Γι' αυτό χρησιμοποιούμε την εντολή AS.

```
SELECT CITY AS PelCity, ID, CITY AS
CityRepeat
FROM PEL;
```

PelCity	ID	CityRepeat
ATHENS	100	ATHENS
CHANIA	101	CHANIA
HERAKLION	102	HERAKLION
ATHENS	103	ATHENS
SALONIKA	104	SALONIKA

Aliases πινάκων

Κάποιες φορές είναι χρήσιμο ή και απαραίτητο να δίνουμε νέο όνομα στον πίνακα.
Στο επόμενο παράδειγμα μετονομάζουμε τον πίνακα PEL σε r.

```
SELECT r.NAME
FROM PEL AS r;
```

Επιλογές υπό συνθήκη – η παράμετρος WHERE

```
SELECT *
FROM PEL
WHERE ID =100;
```

Θα μας εμφανίσει μόνο μια γραμμή: τον πελάτη με ID = 100

ID	NAME	CITY
100	GEORGE	ATHENS

ή χρησιμοποιούμε aliases:

```
SELECT r.NAME
FROM PEL AS r
WHERE r.ID = 103;
```

Με την παράμετρο WHERE μπορούμε να χρησιμοποιήσουμε:

- Συγκριτικούς τελεστές : =, >, <, >=, <=, <> (ή !=)
- Τον τελεστή IN
- Το τελεστή BETWEEN
- Τον τελεστή LIKE
- Το NULL ή το NOT NULL
- Τους λογικούς τελεστές : NOT, AND, OR

Παραδείγματα:

```
SELECT *
FROM PEL
WHERE ID <> 103 AND CITY = 'ATHENS';

SELECT *
FROM PEL
WHERE ID IN (100,106);
```

το ID IN (100,106) είναι ισότιμο με ID = 100 OR ID = 106

```
SELECT *
FROM PEL
WHERE ID BETWEEN 100 AND 106;
```

το ID BETWEEN 100 AND 106 είναι ισότιμο με ID >=100 AND ID <=106.

```
SELECT *
FROM PEL
WHERE NAME LIKE 'E%';
```

Θα εμφανίσει μόνο την ELENH.

```
SELECT *
FROM PEL
WHERE NAME LIKE '%a%';
```

Θα εμφανίσει ονόματα που περιλαμβάνουν το γράμμα 'α'.

ID	NAME	CITY
101	MARY	CHANIA
103	THANOS	HERAKLION

Επιλέγοντας από δύο ή περισσότερους πίνακες.

```
SELECT *
FROM PEL, SYN;
```

θα μας εμφανίσει όλες τις γραμμές. Μαθηματικά σωστό αλλά χωρίς κανένα νόημα εδώ.

Αν υπάρχουν στήλες με ίδιο όνομα σε διαφορετικούς πίνακες τότε χρειαζόμαστε ολόκληρα τα ονόματα τους(PEL.ID ή SYN.ID) ή aliases πινάκων.

```
SELECT PEL.ID, PEL.NAME, PEL.CITY,
SYN.EURO, SYN.AA
FROM PEL, SYN
WHERE PEL.ID = SYN.ID
```

ID	NAME	CITY	EURO	AA
100	GEORGE	ATHENS	100	1
100	GEORGE	ATHENS	150	2
102	THANOS	HERAKLION	200	3

Αυτή η εντολή μπορεί να πραγματοποιηθεί με ένα INNER JOIN των δυο πινάκων.

```
SELECT PEL.ID, PEL.NAME, PEL.CITY,
SYN.EURO, SYN.AA
FROM PEL INNER JOIN SYN
ON PEL.ID = SYN.ID
```

Μπορούμε να γράψουμε την παραπάνω εντολή με aliases πινάκων.

```
SELECT p.ID, p.NAME, p.CITY, s.EURO, s.AA
FROM PEL AS p INNER JOIN SYN AS s
ON p.ID = s.ID
```

Εκτός από το INNER μπορούμε να χρησιμοποιήσουμε LEFT ή RIGHT για να πάρουμε δυο εξωτερικές ενώσεις.

```
SELECT p.ID, p.NAME, p.CITY, s.EURO, s.AA
FROM PEL AS p LEFT JOIN SYN AS s
ON p.ID = s.ID
```

Θα μας εμφανίσει το JOIN των δυο πινάκων καθώς και όλα τα στοιχεία του αριστερού πίνακα (LEFT).

ID	NAME	CITY	EURO	AA
100	GEORGE	ATHENS	100	1
100	GEORGE	ATHENS	150	2
102	THANOS	HERAKLION	200	3
101	MARY	CHANIA		
103	ELENH	ATHENS		
104	PETER	SALONIKA		

Ταξινόμηση – η παράμετρος ORDER BY

```
SELECT *
FROM PEL
ORDER BY NAME;
```

ID	NAME	CITY
103	ELENH	ATHENS
100	GEORGE	ATHENS
101	MARY	CHANIA
104	PETER	SALONIKA
102	THANOS	HERAKLION

Η εντολή ταξινομεί κατά αύξουσα σειρά (ASC). Εάν θέλουμε να ταξινομήσουμε κατά φθίνουσα σειρά γράφουμε DESC.

```
SELECT *
FROM PEL
ORDER BY NAME DESC;
```

Μπορούμε να ταξινομήσουμε τις γραμμές με βάση διάφορων στηλών εφαρμόζοντας ASC ή DESC.
Παρακάτω είναι ένα πιο περίπλοκο παράδειγμα. Ταξινομούμε το JOIN των δυο πινάκων με βάση το πεδίο NAME (ascending) και με βάση το πεδίο EURO (descending).

```
SELECT p.ID, p.NAME, p.CITY, s.EURO, s.AA
FROM PEL AS p INNER JOIN SYN AS s
ON p.ID = s.ID
ORDER BY p.NAME, s.EURO DESC;
```

ID	NAME	CITY	EURO	AA
100	GEORGE	ATHENS	150	2
100	GEORGE	ATHENS	100	1
102	THANOS	HERAKLION	200	3

Φωλιασμένες SELECT

Η πραγματική δύναμη της SQL είναι η δυνατότητα να χρησιμοποιούμε ερωτήματα μέσα σε άλλα ερωτήματα. Στην πράξη βέβαια υπάρχουν και όρια.
Υποθέτουμε ότι θέλουμε να εμφανίσουμε τους πελάτες που έχουν τουλάχιστον μια συναλλαγή. Ένας τρόπος θα ήταν να πάρουμε την ένωση των πινάκων, επιλέγοντας μόνο τις στήλες του PEL και μετά χρησιμοποιώντας DISTINCT για να αποφύγουμε διπλότυπα (την εμφάνιση των ονομάτων δυο φορές). Αυτό είναι ανέφικτο!
Η σωστή λύση είναι να εμφανίσουμε μόνο τις γραμμές του PEL όπου το PEL.ID βρίσκεται στο SYN.ID. Ή να εμφανίσουμε εκείνα τα PEL.ID που εμφανίζονται στο αποτέλεσμα της ερώτησης: SELECT ID FROM SYN.

Αυτό μπορεί να γραφεί ως εξής:

```
SELECT *
FROM PEL AS p
WHERE p.ID IN (SELECT s.ID FROM SYN AS s);
```

ID	NAME	CITY
100	GEORGE	ATHENS
102	THANOS	HERAKLION

Οι εντολές INSERT, UPDATE και DELETE

Μ'αυτές τις εντολές μπορούμε να τροποποιήσουμε τους πίνακες μας.

- Η εντολή *INSERT*, εισάγει μια νέα γραμμή (ή και περισσότερες)

```
INSERT INTO SYN
VALUES (4,300,103);
```

Με αυτή την εντολή εισάγαμε μια νέα γραμμή στον πίνακα SYN. Όμως πρέπει να γνωρίζουμε τις στήλες του SYN και την σειρά τους.
Η επόμενη φόρμα είναι πιο χρήσιμη:

```
INSERT INTO PEL (NAME, ID)
VALUES ('TASOS', 105)
```

Το πεδίο city είναι εδώ NULL. Εκτός και αν έχουμε δώσει κάποια default τιμή.

Μπορούμε να δημιουργήσουμε μαζικά INSERT χρησιμοποιώντας μέσα SELECT. Υποθέτουμε ότι μια τράπεζα στα Χανιά μας στέλνει τους πελάτες της με τον πίνακα PEL2. Θέλουμε λοιπόν να τους προσθέσουμε στον PEL.

```
INSERT INTO PEL
VALUES (SELECT * FROM PEL2 WHERE CITY= 'CHANIA');
```

- Η εντολή *UPDATE*, ενημερώνει τα δεδομένα σε μια ή περισσότερες σειρές.

```
UPDATE SYN
SET EURO = EURO * 10
WHERE AA = 3;
```

Η εντολή αλλάζει την 3η συναλλαγή. Το πεδίο EURO ανέρχεται στις 2000. Πρέπει να είμαστε πολύ προσεκτικοί με την εντολή UPDATE. Σκεφτείτε την εντολή

```
UPDATE PEL
SET NAME = 'MANOLIS';
```

Αύτη θα άλλαζε όλα τα ονόματα σε Μανόλης!

Πρέπει να σημειώσουμε την περίεργη συμπεριφορά την παραμέτρου SET. Γράφοντας EURO = EURO * 10 σίγουρα μας θυμίζει γλώσσα προγραμματισμού (την γλώσσα C για παράδειγμα). Όμως οι αλλαγές δεν εφαρμόζονται όπως στα προγράμματα. Το όνομα της μεταβλητής πριν από το = είναι

καινούρια μεταβλητή. Η μεταβλητή στα δεξιά του = είναι η παλιά.

- Η εντολή *DELETE*, διαγράφει μια η περισσότερες γραμμές. Σίγουρα αυτή η εντολή πρέπει να χρησιμοποιείται με προσοχή. Γράφοντας:

```
DELETE *
FROM PEL
WHERE ID = 104;
```

Διαγράφει τον PETER. Όμως αν γράψουμε :

```
DELETE *
FROM PEL
WHERE ID = 100;
```

Θα διαγράψει τον GEORGE που όμως έχει συναλλαγές με την τράπεζα και συνδέεται με τον πίνακα SYN. Συνήθως αυτό δεν γίνεται τόσο εύκολα. Αν όμως επιτρέψουμε CASCADE DELETE , την ώρα που συνδέουμε τους πίνακες, τότε η εντολή μας θα διαγράψει πρώτα τις συναλλαγές του πελάτη από τον πίνακα SYN και μετά τον ίδιο από τον πίνακα PEL.
Η εντολή που ακολουθεί

```
DELETE *
FROM PEL;
```

Διαγράφει τα περιεχόμενα του πίνακα και όχι τον ίδιο τον πίνακα. Αν θέλουμε να τον διαγράψουμε από την βάση μας θα γράψουμε DROP PEL;

Αθροιστικές Συναρτήσεις

Σε ένα σχεσιακό σχήμα βάσεων δεδομένων, υπάρχει η δυνατότητα να εφαρμόσουμε πάνω στα δεδομένα των πινάκων, ένα σύνολο συναρτήσεων οι οποίες επιστρέφουν κάποιες τιμές.

- **COUNT (*) = επιστρέφει το πλήθος των εγγραφών ενός πίνακα ή το πλήθος κάποιων τιμών**

```
SELECT COUNT (*) AS HowMany
FROM PEL
WHERE CITY = 'ATHENS';
```

HowMany
2

COUNT (πεδίο)

```
SELECT COUNT (CITY)
FROM PEL;
```

Επιστρέφει 4 γιατί μετράει δυο φορές την ATHENS. Χρησιμοποιούμε την παράμετρο DISTINCT για να πάρουμε 3.

```
SELECT COUNT (DISTINCT CITY)
FROM PEL;
```

*** SUM (πεδίο)** = άθροισμα των τιμών σε μη μηδενικές στήλες.
*** AVG (πεδίο)** = μέσος όρος (= SUM/COUNT) των τιμών.

SELECT SUM (EURO) AS Total, AVG (EURO) AS Mean
FROM SYN
WHERE ID = 100;

Total	Mean
250,00 €	125,00 €

*** MAX (πεδίο)** = η μέγιστη τιμή ενός συνόλου.
*** MIN (πεδίο)** = η ελάχιστη τιμή ενός συνόλου.

Αυτές οι συναρτήσεις μπορούν να χρησιμοποιηθούν και σε αλφαριθμητικά δεδομένα. Η επόμενη εντολή επιλέγει την MARY.

```
SELECT MIN (NAME) AS First
FROM PEL;
```

Η παράμετρος **group by** χρησιμοποιείται για να εφαρμόσουμε τις παραπάνω συναρτήσεις όχι μόνο σε ένα σύνολο από πλειάδες αλλά σε ομάδες από σύνολα πλειάδων.
Η παράμετρος **having** χρησιμοποιείται για να εφαρμόσουμε σε μια συγκεκριμένη ομάδα από πλειάδες. Η συνθήκη του having εφαρμόζεται αφού σχηματιστούν οι ομάδες και υπολογιστούν οι αθροιστικές συναρτήσεις.
Η βασική δομή και η σειρά των παραμέτρων είναι η εξής:

```
SELECT A1, A2, ..., AN
FROM   R1, R2, ..., RN
WHERE
GROYP BY
HAVING
ORDER BY
```

Και ένα μικρό παράδειγμα που συνδυάζει τις παραπάνω παραμέτρους…

```
SELECT PEL.NAME, PEL.CITY, SYN.AA
FROM PEL, SYN
WHERE PEL.ID = SYN.ID
GROUP BY PEL.CITY
HAVING SYN.EURO > 150
ORDER BY PEL.NAME;
```

DDL Γλώσσα Ορισμού Δεδομένων

Η γλώσσα ορισμού δεδομένων (Data Definition Language, DDL), επιτρέπει τη διαχείριση πινάκων (tables), σε μια βάση δεδομένων.

Οι πίνακες είναι το δομικό χαρακτηριστικό μιας σχεσιακής βάσης δεδομένων, καθώς περιέχουν τα δεδομένα που καταχωρούνται σε αυτήν.

Η διαχείριση πινάκων περιλαμβάνει τις εντολές για τη δημιουργία και τη διαγραφή πινάκων από τη βάση καθώς και την προσθήκη πεδίων στους πίνακες μετά τη δημιουργία τους. Οι εντολές που πραγματοποιούν τις διαδικασίες αυτές είναι οι **CREATE TABLE, DROP TABLE** και **ALTER TABLE.**

Η εντολή CREATE

Δημιουργία πινάκων.

```
CREATE TABLE PEL (
    ID NUMBER (3) PRIMARY KEY,
    NAME VARCHAR (40) NOT NULL,
    CITY VARCHAR (20) DEFAULT 'HERAKLION');

CREATE TABLE SYN (
    AA NUMBER (4) PRIMARY KEY,
    EURO NUMBER (10,2) NOT NULL,
    ID NUMBER (3) NOT NULL,
    FOREIGN KEY (ID) REFERENCES PEL (ID));
```

Δημιουργία ευρετηρίων.

```
CREATE INDEX NameIndex
ON PEL (NAME);

CREATE INDEX CityName
ON PEL (CITY, NAME);

CREATE INDEX Syn_id
ON SYN (ID);
```

Η εντολή DELETE

Εάν θέλουμε να διαγράψουμε έναν πίνακα

```
DROP TABLE PEL;
```

Η εντολή ALTER

Η εντολή αυτή επιτρέπει την τροποποίηση της δομής του πίνακα μετά τη δημιουργία του.

```
ALTER TABLE PEL ADD PHONE NUMBER (10);
```

DCL Γλώσσα Διαχείρισης Δεδομένων

Το τρίτο μέρος της SQL αποτελείται από την Γλώσσα Διαχείρισης Δεδομένων (DCL). Η DCL χειρίζεται τις εξουσιοδοτήσεις των δεδομένων και επιτρέπει στον σχεδιαστή της Β.Δ. να ελέγχει ποιος έχει πρόσβαση να δει ή να διαχειριστεί τα δεδομένα μέσα στη βάση. Οι δύο κύριες εντολές είναι:

- GRANT: επιτρέπει σε έναν ή περισσότερους χρήστες να εκτελέσουν μια εφαρμογή ή περισσότερες σε ένα αντικείμενο της βάσης.
- REVOKE: αφαιρεί ή ελαττώνει τη δυνατότητα ενός χρήστη να εκτελέσει μια εφαρμογή.

```
GRANT SELECT, UPDATE ON my_table TO some_user;
```

Κεφάλαιο 3 :
Αποθηκευμένες Διαδικασίες (Stored Procedures) & Triggers

Μια αποθηκευμένη διαδικασία είναι ένα κομμάτι κώδικα, σαν τις συναρτήσεις ή μεθόδους που υπάρχουν σε όλες τις γλώσσες προγραμματισμού. Μπορούν να γραφούν σε διάφορες γλώσσες συμπεριλαμβανομένων των SQL, C, C++ και Java.
Είναι ένα ανεξάρτητο σετ από εκτεταμένες SQL δηλώσεις που βρίσκονται αποθηκευμένες σε μια βάση, σαν μέρος των μεταδεδομένων της.
Οι εφαρμογές μπορούν να αλληλεπιδράσουν με τις αποθηκευμένες διαδικασίες με τους εξής τρόπους:

- Μπορούν να περάσουν παραμέτρους και να λάβουν σαν αποτέλεσμα τιμές από τις αποθηκευμένες διαδικασίες.
- Μπορούν να καλέσουν μια Α.Δ. απευθείας για εκτελέσει μια αποστολή.
- Μπορούν να υποκαταστήσουν μια κατάλληλη Α.Δ. για έναν πίνακα ή όψη σε μια δήλωση SELECT.

Τα πλεονεκτήματα της χρήσης Α.Δ. είναι:

- Οι εφαρμογές μπορούν να μοιράζονται τμήματα κώδικα. Ένα κοινό κομμάτι κώδικα SQL γράφεται μια φορά, αποθηκεύεται στην βάση και χρησιμοποιείται από κάθε εφαρμογή που έχει πρόσβαση στη βάση.
- Σχεδίαση υπομονάδων. Οι Α.Δ. μπορούν να μοιραστούν μεταξύ εφαρμογών, εξαλείφοντας διπλότυπο κώδικα και μειώνοντας το μέγεθος των εφαρμογών.
- Διατήρηση της απλοποίησης. Κάθε φορά που μια διαδικασία αναβαθμίζεται, οι αλλαγές αυτόματα εμφανίζονται σε όλες τις εφαρμογές και την χρησιμοποιούν χωρίς να χρειάζεται να την ξανά μεταγλωττίσουν ή ξανά συνδέσουν.
- Βελτιωμένη απόδοση, ειδικά για πρόσβαση από απομακρυσμένους πελάτες. Οι Α.Δ. τρέχουν από τον server και όχι τους πελάτες.

Χρησιμοποιώντας αποθηκευμένες διαδικασίες

Υπάρχουν δυο είδη διαδικασιών που μπορεί μια εφαρμογή να καλέσει.

- *Διαδικασίες τύπου Select,* μια εφαρμογή μπορεί να τις χρησιμοποιήσει στη θέση ενός πίνακα ή όψης σε μια δήλωση SELECT. Μια διαδικασία τέτοιου τύπου κυρίως επιστρέφει μια ή περισσότερες τιμές ή κάποιο μήνυμα σφάλματος.
- *Εκτελέσιμες διαδικασίες,* όπου μια εφαρμογή μπορεί να εκτελέσει απευθείας, με μια δήλωση εκτέλεσης διαδικασίας.

Και οι δυο τύποι ορίζονται με τη δήλωση CREATE PROCEDURE και έχουν το ίδιο συντακτικό. Η διαφορά βρίσκεται στον τρόπο που η διαδικασία γράφεται και στο πως πρόκειται να χρησιμοποιηθεί. Η διαδικασίες τύπου select πάντα επιστρέφουν μηδέν ή περισσότερες στήλες, έτσι ώστε να εμφανιστεί κάτι σαν πίνακας ή όψη. Οι εκτελέσιμες διαδικασίες είναι απλά ρουτίνες καλούμενες από το πρόγραμμα και επιστρέφουν ένα απλό σετ από τιμές.

Διαδικασίες και εκτελέσεις

Οι Α.Δ. λειτουργούν μέσα σε ένα περιβάλλον από εκτελέσεις του προγράμματος που τις χρησιμοποιεί. Εάν μια διαδικασία χρησιμοποιείται σε μια εκτέλεση και αυτή η εκτέλεση ανακαλεστεί, τότε και όλες οι ενέργειες της διαδικασίας θα ανακληθούν επίσης.

Ασφάλεια & Α.Δ.

Όταν μια εφαρμογή καλέσει μια Α.Δ., το άτομο που τρέχει την εφαρμογή πρέπει να έχει δικαίωμα εκτέλεσης (EXECUTE privilege) της Α.Δ. Μια επέκταση στη δήλωση GRANT ενεργοποιεί το δικαίωμα στην εντολή EXECUTE και μια επέκταση στη δήλωση REVOKE απενεργοποιεί το δικαίωμα.

Χρησιμοποιώντας διαδικασίες τύπου SELECT

Μια τέτοια διαδικασία χρησιμοποιείται σαν ‘εικονικός’ πίνακα; ή όψης σε μια δήλωση SELECT και επιστρέφει μηδέν ή περισσότερες στήλες.
Τα πλεονεκτήματα της Α.Δ. αντί για πίνακες ή όψεις είναι:

1. Βρίσκεται τον server
2. Είναι μια ολόκληρη γλώσσα προγραμματισμού. Επιπλέον:
    - Μπορούν να δεχτούν εσωτερικές παραμέτρους οι οποίες επηρεάζουν το εξωτερικό αποτέλεσμα.
    - Μπορούν να περιέχουν δηλώσεις ελέγχου, τοπικές μεταβλητές και δηλώσεις ελέγχου δεδομένων, προσφέροντας μεγάλη ευελιξία στο χρήστη.

Καλώντας διαδικασίες τύπου SELECT

Για να χρησιμοποιήσουμε τέτοιου είδους διαδικασίες στη θέση ενός πίνακα ή όψης σε μια εφαρμογή, χρησιμοποιούμε το όνομα της διαδικασίας όπου θα χρησιμοποιούσαμε το όνομα του πίνακα ή της όψης.

Εκτελέσιμες διαδικασίες

Μια τέτοιου είδους διαδικασία καλείται ευθέως από μια εφαρμογή και συχνά εκτελεί μια αποστολή σύνηθες σε εφαρμογές που χρησιμοποιούν την ίδια β.δ. Οι εκτελέσιμες διαδικασίες μπορούν να δεχτούν εσωτερικές μεταβλητές από το καλούμενο πρόγραμμα και μπορούν επιλεκτικά να του επιστρέψουν μια και μοναδική σειρά.
Οι παράμετροι περνιούνται στην διαδικασία μέσα σε κόμματα ακολουθώντας το όνομα της διαδικασίας.
Πρέπει να σημειώσουμε ότι οι εκτελέσιμες διαδικασίες δεν μπορούν να επιστρέψουν πολλαπλές σειρές.

Μπορούμε να καθορίσουμε μια αποθηκευμένη διαδικασία με τη δήλωση CREATE PROCEDURE σε isql. Μια Α.Δ. συγκροτείται από την *επικεφαλίδα* και το *σώμα*.

Η επικεφαλίδα περιλαμβάνει:
* Το όνομα της Α.Δ. το οποίο πρέπει να είναι μοναδικό μεταξύ των ονομάτων της διαδικασίας, των όψεων και των πινάκων μέσα στη β.δ.
* Μια προαιρετική λίστα εισαγόμενων μεταβλητών και τους τύπους τους, όπου η διαδικασία λαμβάνει από το καλούμενο πρόγραμμα.
* Τέλος, αν η διαδικασία επιστρέφει μεταβλητές, μετά από τη δήλωση RETURNS ακολουθεί μια λίστα από τις εξαγόμενες μεταβλητές και τους τύπους τους.

Το σώμα της διαδικασίας περιλαμβάνει:

- Μια προαιρετική λίστα μεταβλητών με τους τύπους τους.
- Ένα τμήμα από δηλώσεις σε Interbase και trigger language μεταξύ των λέξεων BEGIN και END. Ένα τμήμα μπορεί να συμπεριλάβει και άλλα τμήματα έτσι ώστε να υπάρχουν πολλά επίπεδα ενσωμάτωσης.

Σύνταξη της CREATE PROCEDURE

```
CREATE PROCEDURE name
[(param datatype [, param datatype …])]
[RETURNS (param datatype [, param datatype …])]
AS
<procedure_body>;
<procedure_body> =
[<variable_declaration_list>]
<block>
<variable_declaration_list> =
DECLARE VARIABLE var datatype;
[DECLARE VARIABLE var datatype; …]
<block> =
BEGIN
<compound_statement>
[<compound_statement> …]
END
```

Κι ένα μικρό παράδειγμα το οποίο αθροίζει αριθμούς από το 1 έως την παράμετρο i που δίνουμε ως εισαγόμενη μεταβλητή.

```
CREATE PROCEDURE SUM_INT (I INTEGER) RETURNS (S
INTEGER)
AS
BEGIN
s = 0;
WHILE (i > 0) DO
BEGIN
s = s + i;
i = i - 1;
END
END
```

Διαδικασίες & Trigger Language

Η InterBase διαδικασία και η trigger language είναι μια ολοκληρωμένη γλώσσα προγραμματισμού για αποθηκευμένες διαδικασίες και για τα triggers. Συμπεριλαμβάνει

- SQL δηλώσεις διαχείρισης δεδομένων: INSERT, UPDATE, DELETE και απλό SELECT.
- SQL συναρτήσεις και εκφράσεις.
- Ισχυρές επεκτάσεις σε SQL, συμπεριλαμβανομένου δηλώσεων εκχώρησης, ελέγχου ροής, πλαισίου μεταβλητών, εξαιρέσεων και error – handling.

Κεφάλαιο 4 :
Όρια της γλώσσας SQL και λύσεις με Stored Procedures

4.1 Πρόβλημα 1 :

Έστω ότι έχουμε το παρακάτω πρόβλημα. Στο Τ.Ε.Ι. υπάρχουν πολλά μαθήματα τα οποία εξαρτώνται από άλλα μαθήματα, τα λεγόμενα 'αλυσίδες'.
Υποθέτουμε ότι έχουμε τα επόμενα πέντε μαθήματα:

1. ΕΡ (Εισαγωγή στην Πληροφορική)
2. PR (Προγραμματισμός)
3. DB (Βάσεις Δεδομένων)
4. DOM (Δομές Δεδομένων και Αλγόριθμοι)
5. JAVA (Προγραμματισμός στην JAVA)

Υποθέτουμε πάλι ότι έχουμε τις εξής εξαρτήσεις:
EP → PR → DOM → JAVA
EP → DB
Δηλαδή κάποιος φοιτητής για να παρακολουθήσει τη JAVA θα πρέπει πρώτα να έχει επιτύχει στα τρία προηγούμενα. Καθώς επίσης για να παρακολουθήσει τις Βάσεις Δεδομένων θα πρέπει να επιτύχει στην Πληροφορική.
Σημ.: Δεν υπάρχουν τέτοιες αλυσίδες στο Τ.Ε.Ι.

Μπορούμε να περιγράψουμε το πρόβλημα μας σε έναν πίνακα, έστω ονόματι ΜΑΤΗ, όπως ακολουθεί:

id	Title	pro
1	EP	0
2	PR	1
3	DB	1
4	DOM	2
5	JAVA	4

Αποτελείται από τρεις στήλες. Τον κωδικό του μαθήματος, τον τίτλο του και μια στήλη με τον κωδικό του προαπαιτούμενου μαθήματος του. Για απλότητα θα υποθέσουμε ότι κάθε μάθημα εξαρτάται από ένα μάθημα κάθε φορά. Η τιμή 0 στην στήλη pro σημαίνει ότι δεν έχει καμία εξάρτηση το συγκεκριμένο μάθημα από άλλο.

Να γραφτεί μια ερώτηση σε SQL η οποία, για δοσμένο id, να εκτυπώνει όλη την αλυσίδα των εξαρτήσεων. Δηλαδή όλες τις αλυσίδες ενός μαθήματος.
Για παράδειγμα, η αλυσίδα της JAVA (id=5) θα είναι :
JAVA, DOM, PR, EP.

Πώς όμως θα γράψουμε μια τέτοια ερώτηση?
Για να πάρουμε ως αποτέλεσμα την JAVA απλά γράφουμε:

```
SELECT *
FROM MATH
WHERE id = 5;
```

Για να πάρουμε την DOM (σαν προαπαιτούμενο της JAVA) απλά γράφουμε:

```
SELECT *
FROM MATH
WHERE id in (SELECT pro
             FROM MATH
             WHERE id = 5);
```

Για να εκτυπώσουμε τον PR γράφουμε:

```
SELECT *
FROM MATH
WHERE id in (SELECT pro
             FROM MATH
             WHERE id in (SELECT pro
                          FROM MATH
                          WHERE id = 5));
```

Για να εκτυπώσουμε την EP γράφουμε:

```
SELECT *
FROM MATH
WHERE id in(SELECT pro
            FROM MATH
            WHERE id in(SELECT pro
                        FROM MATH
                        WHERE id in (SELECT pro
                                     FROM MATH
                                     WHERE id=5);
```

Μπορούμε να χρησιμοποιήσουμε την UNION για να επιλέξουμε όλα τα αποτελέσματα:

```
SELECT *
FROM MATH
WHERE id = 5
UNION
SELECT *
FROM MATH
WHERE id in (SELECT pro
             FROM MATH
             WHERE id = 5)
UNION
SELECT *
FROM MATH
WHERE id in (SELECT pro
             FROM MATH
             WHERE id in (SELECT pro
                          FROM MATH
                          WHERE id = 5));
UNION
SELECT *
FROM MATH
WHERE id in(SELECT pro
            FROM MATH
            WHERE id in(SELECT pro
                        FROM MATH
                        WHERE id in (SELECT pro
                                     FROM MATH
                                     WHERE id=5);
```

Βλέπουμε όμως ότι όλο αυτό εξελίσσεται σε ένα μεγάλο κομμάτι κώδικα και ότι λειτουργεί για ένα δοσμένο βάθος της αλυσίδας.
Η πρώτη ερώτηση έχει βάθος = 0 (η ίδια η JAVA).
Η δεύτερη έχει βάθος 1 (DOM), η τρίτη βάθος 2 (PR) κ.τ.λ.

Δεν μπορούμε να γράψουμε ένα ερώτημα το οποίο είναι ανεξάρτητο του βάθους.

Τώρα, μπορεί κάποιος να επιχειρηματολογήσει λέγοντας ότι στο ΤΕΙ έχουμε μόνο 8 εξάμηνα, έτσι κάθε αλυσίδα θα έχει βάθος μέχρι το πολύ 8 και συνεπώς μπορούμε να γράψουμε ένα τέτοιο (τεράστιο) ερώτημα!

Να σημειώσουμε όμως πως παρόμοιες αλυσίδες υπάρχουν και σε πολλές άλλες περιπτώσεις όπου δεν μπορούμε να προβλέψουμε το βάθος. Για παράδειγμα στα forum διάφορων ιστοσελίδων όπου οι χρήστες απαντούν στα μηνύματα άλλων χρηστών, άλλοι απαντάν στις απαντήσεις κ.ο.κ. Εκεί δεν υπάρχει δυνατότητα πρόβλεψης.
Μπορεί επίσης κάποιος να υποθέσει ότι ένα τέτοιο ερώτημα μπορεί να γραφτεί με άλλα SLQ εργαλεία, όπως το JOIN, GROUP και άλλα…
Όμως, όχι! Το παραπάνω πρόβλημα είναι ειδική περίπτωση προβλήματος Μεταβατικής Ολοκλήρωσης (Transitive Closure – TC). Και κάθε TC δεν μπορεί να υπολογιστεί με SQL γιατί είναι μαθηματικό θεώρημα.

Εάν έχουμε στοιχειώδης γνώσεις σε γλώσσες προγραμματισμού, τότε η λύση είναι πολύ απλή! Είναι μόνο μια απλή επανάληψη while.
Παρακάτω θα παρουσιάσουμε μια λύση σε γλώσσα C. Για απλότητα θα χρησιμοποιήσω δυο παράλληλους πίνακες, τον title και τον pro, για να κρατάω τα δεδομένα.
Υποθέτω λοιπόν ότι:

```
title[1] = "EP";      pro[1] = 0;
title[2] = "PR";      pro[2] = 1;
title[3] = "DB";      pro[3] = 1;
title[4] = "DOM";     pro[4] = 2;
title[5] = "JAVA";    pro[5] = 4;
```

```
main()
{ int id;

  printf("give id -> "); scanf( "%d", &id );

  while (id>0)
     {
        printf( "lecture = %s \n", title[id] );
        id = pro[id]; // για να πάρουμε το επόμενο pro
     }
}
```

Όπως βλέπουμε το παραπάνω πρόβλημα είναι πολύ απλό. Και λειτουργεί για οποιοδήποτε βάθος αλυσίδας. Συμπέρασμα: η SQL έχει προβλήματα γιατί δεν είναι γλώσσα προγραμματισμού!

Στα περισσότερα σημερινά συστήματα βάσεων δεδομένων RDBMS υπάρχουν προεκτάσεις της SQL, καλούμενες Αποθηκευμένες Διαδικασίες (Stored Procedures) και Triggers, τα οποία είναι γλώσσες προγραμματισμού (με μεταβλητές, επαναλήψεις κτλ.)
Σαν παράδειγμα θα παραθέσουμε μια λύση σε FireBird χρησιμοποιώντας μια Stored Procedure ονόματι ALYSIDA.

```
 CREATE PROCEDURE ALYSIDA (CID INTEGER)
RETURNS (ID INTEGER, TITLE VARCHAR(30))
AS
  DECLARE VARIABLE PRO INTEGER;
BEGIN
  WHILE (CID>0) DO
  BEGIN

    FOR SELECT ID, TIT, PRO
    FROM MATH
    WHERE ID = CID
    INTO :ID, :TITLE, :PRO
    DO; // διάβασε δεδομένα & επέστρεψέ τα στον Client

    CID = PRO; // πάρε το επόμενο pro

    SUSPEND; // περίμενε για το επόμενο Client's FETCH

  END
END
```

Σχόλια:

1. Το Create καθοδηγεί την FireBird να δημιουργήσει την ALYSIDA και να την αποθηκεύσει στην πλευρά του server. Είναι κάτι όπως την εντολή CREATE TABLE.
2. Στην επανάληψη του while έχουμε δήλωση FOR SELECT, το οποίο δείχνει πώς να χρησιμοποιήσουμε SQL ερωτήματα στις αποθηκευμένες διαδικασίες.
3. Για να διακρίνουμε μεταξύ των στηλών που επιστρέφονται από το SELECT και των μεταβλητών, μπορούμε να χρησιμοποιήσουμε το σύμβολο (:) , έτσι το ID είναι στήλη , ενώ το :ID είναι μια επιστρεφόμενη μεταβλητή.
4. Το SUSPEND περιμένει για το επόμενη εντολή FETCH.

Μπορούμε να καλέσουμε την ALYSIDA όπως παρακάτω:

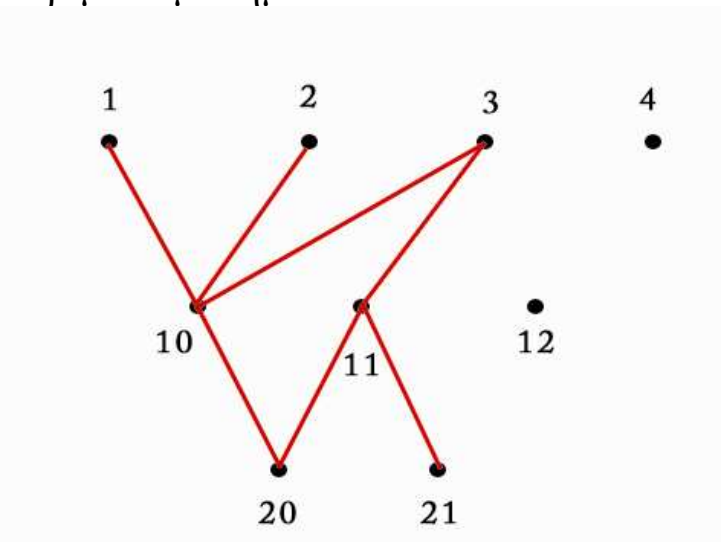
```
SELECT *
FROM ALYSIDA (5);
```

Το αποτέλεσμά μας…

ID	TITLE
5	JAVA
4	DOM
2	PR
1	EP

Πρόβλημα 2 :

Σαν δεύτερο πρόβλημα θα χρησιμοποιήσουμε μια γενίκευση του πρώτου προβλήματος. Στο Τ.Ε.Ι. υπάρχουν πολλά μαθήματα τα οποία εξαρτώνται από άλλα μαθήματα, τα λεγόμενα ‘αλυσίδες’. Πολλές φορές ένα μάθημα όμως εξαρτάται από περισσότερα από ένα μαθήματα.
Φανταστείτε ένα τέτοιο γράφημα όπου οι αριθμοί αντιστοιχούν σε συγκεκριμένα μαθήματα:



Η αλυσίδα του μαθήματος 20 είναι περίπλοκη γιατί εξαρτάται από το μάθημα 11, το οποίο εξαρτάται από ένα μάθημα και το μάθημα 10, το οποίο εξαρτάται με τη σειρά του από τρία άλλα μαθήματα.
Θα περιγράψουμε το πρόβλημα μας χρησιμοποιώντας τους παρακάτω πίνακες

MATH2

id	Title
1	m1
2	m2
3	m3
4	m4
10	m10
11	m11
12	m12
20	m20
21	m21

DEP

id	Pro
10	1
10	2
10	3
11	3
20	10
20	11
21	11

Ο πίνακας MATH αποτελείται από δυο στήλες, την id δηλαδή τον κωδικό του μαθήματος και την title με τον τίτλο του μαθήματος. Ο πίνακας DEP, ο οποίος αποτελείται από τις εξαρτήσεις μεταξύ των μαθημάτων και είναι ο πιο σημαντικός πίνακας του προβλήματος, αποτελείται από την στήλη id με τους κωδικούς των μαθημάτων και την στήλη pro όπου δείχνει το προαπαιτούμενο του συγκεκριμένου μαθήματος. Ο πίνακας αυτός έχει μια ιδιαιτερότητα διότι κλειδί του είναι ο συνδυασμός των δυο στηλών, γι'αυτό και παρατηρούμε διπλότυπα στην στήλη id.

Να γραφτεί μια SQL η οποία, για δοσμένο id, να εκτυπώνει όλη την αλυσίδα των εξαρτήσεων. Δηλαδή όλα τα προαπαιτούμενα ενός μαθήματος.
Για παράδειγμα, η αλυσίδα του μαθήματος m20 (id=20) θα είναι :
m10, m1, m2, m11 και m3.

Πώς όμως θα γράψουμε μια τέτοια ερώτηση?
Για να πάρουμε ως αποτέλεσμα το m20 απλά γράφουμε:

```
SELECT *
FROM MATH2
WHERE id = 20;
```

Για να πάρουμε το m10 και το m11 (σαν προαπαιτούμενα του m20) απλά γράφουμε:

```
SELECT *
FROM MATH2
WHERE id in (SELECT pro
             FROM DEP
             WHERE id = 20);
```

Για να εκτυπώσουμε το m1, m2 και m3 γράφουμε:

```
SELECT *
FROM MATH2
WHERE id in (SELECT pro
             FROM DEP
             WHERE id in (SELECT pro
                          FROM DEP
                          WHERE id = 20));
```

Και συνεχίζεται κάπως έτσι…

Μπορούμε να χρησιμοποιήσουμε την UNION για να επιλέξουμε όλα τα αποτελέσματα:

```
SELECT *
FROM MATH2
WHERE id = 20
UNION
SELECT *
FROM MATH2
WHERE id in (SELECT pro
             FROM DEP
             WHERE id = 20)
UNION
SELECT ...
```

Βλέπουμε όμως ότι όλο αυτό λειτουργεί για ένα δοσμένο βάθος της αλυσίδας.

Η πρώτη ερώτηση έχει βάθος = 0 (το ίδιο το m20).
Η δεύτερη έχει βάθος 1 (m10,m11), η τρίτη βάθος 2 (m1,m2,m3) κ.τ.λ.

Δεν υπάρχει λύση του προβλήματος σε γλώσσα SQL και ήταν προφανές διότι ούτε και στο πρώτο πρόβλημα υπήρχε λύση, το οποίο ήταν πιο εύκολο του δεύτερου.

Μια προσέγγιση στη γλώσσα C θα ήταν ο παρακάτω κώδικας…

```
#include<stdio.h>

int pinid [7] = {10,10,10,11,20,20,21};
int pinpro [7] = { 1, 2, 3, 3,10,11,11};

void alysida2 ( int id )
{ int i, pro;

  printf ("%d\n", id);

  for (i=0; i<7; i++)
  if (pinid[i] = = id)
{
    pro = pinpro[i];
    alysida2(pro);                    // παλινδρόμηση !!!
  }
}

main()
{
  alysida2(20);
}
```

Ο παραπάνω κώδικας χρησιμοποιεί εξωτερική συνάρτηση την alysida, η οποία καλεί τον εαυτό της, έχουμε δηλαδή παλινδρόμηση.
Μέσα στην συνάρτηση έχουμε έναν έλεγχο for και μια συνθήκη if, τα οποία σε συνδυασμό σαρώνουν τους πίνακες για να βρουν το αποτέλεσμα που να τα ικανοποιεί. Μέσα σ’ αυτόν τον έλεγχο έχουμε κλήση της ίδιας της συνάρτησης θέλοντας να ελέγξουμε τυχόν στοιχεία που ζητάμε.

Λύση του προβλήματος μας με Stored Procedures…

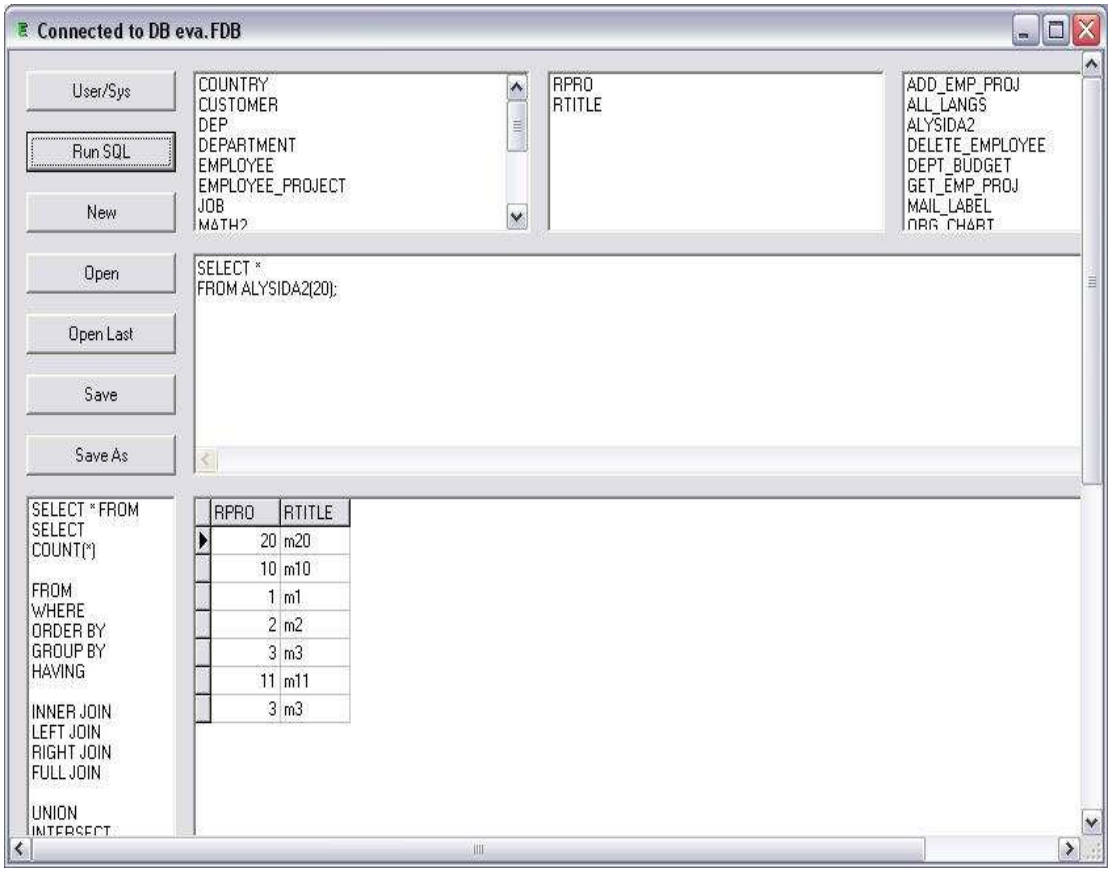
```
create or alter procedure ALYSIDA2 ( ID integer )
returns ( RPRO integer, RTITLE varchar(50) )
AS
 declare variable p integer;
 declare variable q integer;
 declare variable t varchar(50);

BEGIN

  SELECT title FROM MATH2 WHERE id=:ID into RTITLE;
  RPRO = ID;
  SUSPEND;

  FOR SELECT pro FROM DEP WHERE id = :ID INTO :p
  DO
  BEGIN
    FOR SELECT * FROM ALYSIDA2(:p) INTO :q,:t
    DO
    BEGIN
      RPRO = q;
      RTITLE = t;
      SUSPEND;
    END
  END

END
```

Θα καλέσουμε την διαδικασία απλά γράφοντας

```
SELECT *
FROM ALYSIDA2( 20 )
```

Στην παρακάτω εικόνα βλέπουμε το αποτέλεσμα που μας δίνει η διαδικασία, τρέχοντας την στη Firebird. Παίρνουμε ως αποτέλεσμα τον πίνακα με τα προαπαιτούμενα του μαθήματος m20.

RPRO	RTITLE
20	m20
10	m10
1	m1
2	m2
3	m3
11	m11
3	m3


Connected to DB eva.FDB
User/Sys
Run SQL
New
Open
Open Last
Save
Save As
COUNTRY
CUSTOMER
DEP
DEPARTMENT
EMPLOYEE
EMPLOYEE_PROJECT
JOB
RPRO
RTITLE
ADD_EMP_PROJ
ALL_LANGS
ALYSIDA2
DELETE_EMPLOYEE
DEPT_BUDGET
GET_EMP_PROJ
MAIL_LABEL
SELECT *
FROM ALYSIDA2(20);
SELECT * FROM
SELECT
COUNT(*)
FROM
WHERE
ORDER BY
GROUP BY
HAVING
INNER JOIN
LEFT JOIN
RIGHT JOIN
FULL JOIN
UNION
RPRO RTITLE
20 m20
10 m10
1 m1
2 m2
3 m3
11 m11
3 m3

Βιβλιογραφία

- http://en.wikipedia.org/wiki/
- Εισαγωγή στα συστήματα βάσεων δεδομένων.
  Συγγραφέας: C. J. Date
- Inside Relational Databases.
  Συγγραφέας: Mark Whitehorn & Bill Marklyn
- Manual της Firebird:
  Firebird-1.5-QuickStartGuide.pdf

Τα προγράμματα που χρησιμοποιήθηκαν κατά τη διάρκεια της πτυχιακής ήταν:

- Microsoft Access
- Microsoft Visual Studio
- Firebird