Διεθνές Πανεπιστήμιο Της Ελλάδος
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΗΜΕΙΩΣΕΙΣ
Εισαγωγή στην SQL

Επικ. Καθ. Καζανίδης Ιωάννης

Καβάλα 2021

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

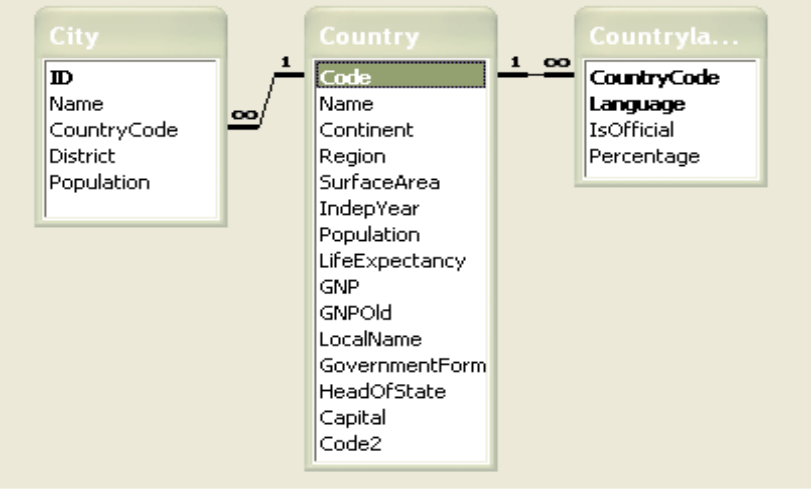
1.1 Εισαγωγή

Η SQL Structured Query Language (Γλώσσα Δομημένων Ερωτημάτων) αποτελεί

1.2 Περιγραφή Βάσης Δεδομένων για τα παραδείγματα

Στα παραδείγματα που συνοδεύουν τις σημειώσεις χρησιμοποιείται η βάση δεδομένων world η οποία ανακτήθηκε από τον "Επίσημο Οδηγό της MySQL 5.0". Αποτελείται από τρεις πίνακες με δεδομένα τα οποία αφορούν στατιστικά στοιχεία των χωρών αλλά και των πόλεών τους. Συγκεκριμένα έχουμε τους παρακάτω πίνακες

ΠΙΝΑΚΑΣ	ΠΕΡΙΓΡΑΦΗ
Coyntry	Περιέχει πληροφορίες για κάθε χώρα
City	Περιέχει πληροφορίες για κάθε πόλη
CoyntryLanguage	Περιέχει πληροφορίες για τις γλώσσες που ομιλούνται σε κάθε χώρα



Σχήμα 1.1: Παρουσίαση των πινάκων της βάσης world και των συσχετίσεών τους

Πίνακας 1.1: Περιγραφή του πίνακα City της βάσης world

City		
Πεδίο	**Περιγραφή**	**Τύπος Δεδομένων**
ID	Κωδικός πόλης	int
Name	Όνομα πόλης	char(35)
CountryCode	Κωδικός Χώρας	int
District	Περιοχή που βρίσκεται η πόλη	char(20)
Population	Πλυθησμός	int

Πίνακας 1.2: Περιγραφή του πίνακα Country της βάσης world

Country		
Πεδίο	**Περιγραφή**	**Τύπος Δεδομένων**
Code	Τριψήφιος κωδικός χώρας	char(3)
Name	Όνομα πόλης	char(52)
Continent	Κωδικός Χώρας	enum('Asia',…)
Region	Περιοχή που βρίσκεται η πόλη	char(26)
SurfaceArea	Έκταση της χώρας	float(10,2)
IndepYear	Χρονιά ανεξαρτησίας	smallint
Population	Πληθυσμός	int
LifeExpectancy	Μέσο προσδόκιμο ζωής	float(3,1)
GNP	Ακαθάριστο Εθνικό Προιόν (ΑΕΠ)	float(10,2)
GNPOld	ΑΕΠ προηγούμενης χρονιάς	float(10,2)
LocalName	Τοπικό όνομα χώρας	char(45)
GovernmentForm	Πολίτευμα	char(45)
HeadOfState	Αρχηγός κράτους	char(60)
Capital		int
Code2	Διψήφιος κωδικός χώρας	char(2)

Πίνακας 1.3: Περιγραφή του πίνακα CountryLanguage της βάσης world

CountryLanguage		
Πεδίο	**Περιγραφή**	**Τύπος Δεδομένων**
CountryCode	Κωδικός χώρας	char(3)
Language	Γλώσσα	char(30)
IsOfficial	Αν είναι επίσημη ή όχι	enum('T','F')
Percentage	Ποσοστό του πληθυσμού που ομιλάει τη γλώσσα	float(4,1)

ΚΕΦΑΛΑΙΟ 2

Η SQL και η MySQL

2.1 Η SQL

Η SQL Structured Query Language (Γλώσσα Δομημένων Ερωτημάτων) αποτελεί ένα παγκόσμιο πρότυπο για τον χειρισμό Σχεσιακών Βάσεων Δεδομένων (relational databases).Αρχικά η SQL ονομαζόταν SEQUEL (Structured English Query Language) και υλοποιήθηκε για το System R που ήταν ένα πειραματικό σχεσιακό σύστημα της IBM. Η SQL περιλαμβάνει χαρακτηριστικά της σχεσιακής άλγεβρας αλλά με σύνταξη περισσότερο φιλική στο χρήστη.

Κοινή προσπάθεια των οργανισμών ANSI (American National Standards Institute)και ISO (International Standards Organization) οδήγησαν σε διαδοχικές τυποποιημένες εκδόσεις της SQL. Η πρώτη τυποποιημένη έκδοση της SQL που λέγεται και SQL1 έγινε το 1986. Το πρότυπο αυτό αναθεωρήθηκε και επεκτάθηκε το 1992, και ονομάστηκε SQL2 (ή SQL-92). Το νεότερο πρότυπο SQL3 (αναφέρεται και ως SQL-99) αποτελεί μία επέκταση της SQL με αντικειμενοστραφή στοιχεία αλλά και σύγχρονα χαρακτηριστικά των βάσεων δεδομένων.

Η SQL αποτελεί μία πλήρης γλώσσα βάσεων δεδομένων. Δίνει δυνατότητες τόσο για ορισμό δεδομένων όσο και για αναζήτηση δεδομένων και ενημερώσεις. Επιπροσθέτως είναι δυνατή η ενσωμάτωσή της μέσα σε γενικής χρήσης γλώσσες προγραμματισμού (εμφωλιασμένη SQL). Είναι ταυτόχρονα λοιπόν Γλώσσα Ορισμού Δεδομένων (ΓΟΔ) όπως και Γλώσσα Χειρισμού Δεδομένων (ΓΧΔ). Επιπλέον επιτρέπει τον ορισμό περιορισμών ακεραιότητας και διευκολύνει τον ορισμό όψεων σε μία Βάση Δεδομένων (ΒΔ). Προσφέρει δυνατότητες για την διαχείριση των χρηστών και των δικαιωμάτων τους σε μία ΒΔ και παρέχει έλεγχο των δοσοληψιών. Ακολουθεί μία λίστα με βασικές εντολές της SQL οι οποίες υποστηρίζονται από σχεδόν όλες τις υλοποιήσεις της SQL.

Εντολές Ορισμού ΒΔ	Εντολές Διαχείρισης Δεδομένων
CREATE	INSERT
ALTER	UPDATE
DROP	DELETE

2.2 Η MySQL

Υπάρχουν αρκετές υλοποιήσεις της SQL όπως οι MySQL, Oracle, Microsoft SQL Server, Infomix και άλλες. Σε αυτές τις σημειώσεις θα αναφερθούμε ιδιαίτερα στην MySQL. Η MySQL προσφέρει μία από τις καλύτερες υλοποιήσεις της SQL. Δουλεύει σε πολλά λειτουργικά συστήματα (Windows, Linux, Unix) και με πολλές γλώσσες όπως οι PHP, Java, C, C++ κλπ. Είναι ανοικτού κώδικα οπότε έχει μηδενικό κόστος απόκτησης και είναι σταθερή. Η τεκμηρίωσή της είναι πάρα πολύ καλή και υπάρχει άφθονο σχετικό υλικό στο διαδίκτυο όπως και βιβλιογραφία. Για περισσότερες λεπτομέρειες μπορείτε να απευθυνθείτε στη διεύθυνση www.mysql.com.

2.3 Πίνακες

Στα σχεσιακά συστήματα βάσεων δεδομένων τα δεδομένα αναπαρίστανται χρησιμοποιώντας πίνακες. Κάθε πίνακας αναφέρεται επίσης και ως σχέση (relation). Ένα ερώτημα SQL προς την βάση δεδομένων έχει ως αποτέλεσμα επίσης έναν πίνακα.

Ένας πίνακας έχει την εξής δομή:

Στήλη 1	Στήλη 2	...	Στήλη N
...	...	...	...

Κάθε πίνακας έχει ένα μοναδικό όνομα και αποτελείται από γραμμές (rows) στις οποίες αποθηκεύεται η πληροφορία. Κάθε γραμμή του πίνακα περιέχει μόνο μια εγγραφή. Ένας πίνακας μπορεί να έχει μία ή περισσότερες στήλες. Κάθε στήλη έχει ένα όνομα και έναν τύπο δεδομένων και περιγράφει ένα πεδίο του πίνακα. Οι τύποι των πληροφοριών που θα

αποθηκευτούν σε έναν πίνακα καθορίζονται από τους τύπους δεδομένων των πεδίων που έχουν οριστεί κατά την δημιουργία του πίνακα.

2.4 Τύποι Δεδομένων

Ένας πίνακας μπορεί να έχει ως 254 πεδία του ίδιου ή διαφορετικού τύπου δεδομένων. Ως γενικούς τύπους δεδομένων θα μπορούσαμε να αναφέρουμε τα αλφαριθμητικά, τους αριθμούς και τις ημερομηνίες.

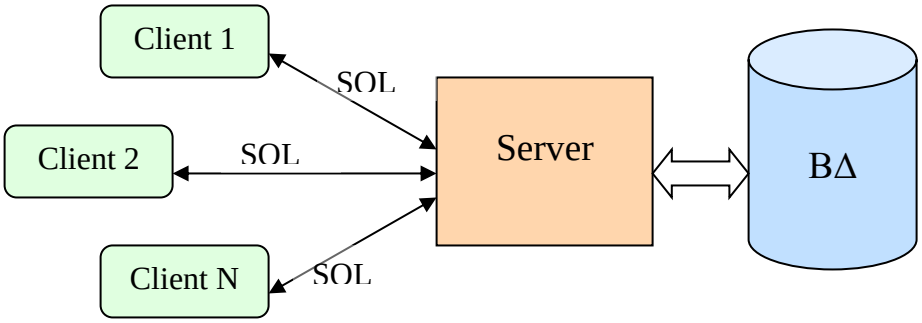
Οι τύποι δεδομένων που υποστηρίζονται από την MySQL παρατίθενται στον παρακάτω πίνακα

Τύποι Δεδομένων	Περιγραφή	Εύρος / Μορφή
Αριθμητικοί τύποι δεδομένων		
INT	Ακέραιος	(-2^{31} έως 2^{31} -1), ή (0 έως 2^{32} -1) χωρίς πρόσημο
TINYINT	Πολύ μικρός ακέραιος	(-2^{7} έως 2^{7} -1), ή (0 έως 2^{8} -1) χωρίς πρόσημο
SMALLINT	Μικρός ακέραιος	(-2^{15} έως 2^{15} -1), ή (0 έως 2^{8} -1) χωρίς πρόσημο
MEDIUMINT	Μέσου μεγέθους ακέραιος	(-2^{23} έως 2^{23} -1), ή (0 έως 2^{24} -1) χωρίς πρόσημο
BIGINT	Μεγάλος ακέραιος	(-2^{63} έως 2^{63} -1), ή (0 έως 2^{64} -1) χωρίς πρόσημο
FLOAT	Αριθμός μονής ακρίβειας κινητής υποδιαστολής	(-3.4x10^{+38},-1.176x10^{-38}) για αρνητικούς αριθμούς και (1.176x10^{-38}, 3.403x10^{+38}) για θετικούς αριθμούς
DOUBLE	Αριθμός διπλής ακρίβειας κινητής υποδιαστολής	(-1.798x10^{+308},-2.225x10^{-308}) για αρνητικούς αριθμούς και (2.225x10^{-308}, 1.798x10^{+308}) για θετικούς αριθμούς
DECIMAL	Αριθμός που αποθηκεύεται ως αλφαριθμητικό (String)	Το εύρος είναι όμοιο με τους αριθμούς DOUBLE
Χρονικοί τύποι δεδομένων		
DATE	Μία ημερομηνία	Μορφή: YYYY-MM-DD. Εύρος 1000-01-01 έως 9999-12-31
DATETIME	Ημερομηνία και ώρα	Μορφή: YYYY-MM-DD hh:mm:ss. Εύρος 1000- 01-01 00:00:00 έως 9999-12-31 23:59:59

TIMESTAMP	Ένα Timestamp	Μορφή: YYYYMMDDhhmmss. Εύρος 19700101000000 έως μέση του έτους 2037
TIME	Μία ώρα	hh:mm:ss format. Range -838:59:59 έως 838:59:59
YEAR	Ένα έτος	YYYY format. Range 1900 έως 2155
Αλφαριθμητικοί τύποι δεδομένων		
CHAR	Αλφαριθμητικό σταθερού μήκους	0-255 χαρακτήρες
VARCHAR	Αλφαριθμητικό μεταβλητού μήκους	0-255 χαρακτήρες
TEXT	Κείμενο	0-65535 bytes
TINYTEXT	Small text field	0-255 bytes
MEDILTMTEXT	Medium-sized text	0-16777215 bytes
LONGTEXT	Large text field	0-4294967295 bytes
Δυαδικοί τύποι δεδομένων αλφαριθμητικών		
BLOB	Μεταβλητού μήκους δυαδική συμβολοσειρά	Δυαδική τιμή έως 0-65535 bytes
TINYBLOB	Μικρή τιμή BLOB	Δυαδική τιμή έως 0-255 bytes
MEDIUMBLOB	Μεσαία τιμή BLOB	Δυαδική τιμή έως 0-16777215 bytes
LONGBLOB	Μεγάλη τιμή BLOB	Δυαδική τιμή έως 0-4294967295 bytes
Τύποι δεδομένων με λίστες τιμών		
ENUM	Λίστα Τιμών	Αλφαριθμητικό με μία μόνο από τις τιμές της λίστας
SET	Λίστα τιμών	Αλφαριθμητικό με καμία ή πολλές από τις τιμές της λίστας (έως 64)

2.5. Εκτέλεση προγραμμάτων client της MySQL

Η MySQL λειτουργεί σε ένα δικτυακό περιβάλλον χρησιμοποιώντας την αρχιτεκτονική Client/Server. Αυτό σημαίνει ότι ένα κεντρικό πρόγραμμα θα λειτουργεί ως server και διάφορα προγράμματα client συνδέονται με τον server για να θέσουν ερωτήματα. Ο server είναι αυτός που επικοινωνεί με τη ΒΔ και επιστρέφει τις ζητούμενες πληροφορίες στα προγράμματα client.



Τα παραδείγματα αυτών των σημειώσεων χρησιμοποιούν το πρόγραμμα mysql, αλλά οι γενικές αρχές εφαρμόζονται και σε άλλα προγράμματα MySQL client γραμμής εντολών.

2.5.1 Σύνδεση με τη MySQL

Για να εκτελέσουμε το πρόγραμμα και να συνδεθούμε στον server με τις προκαθορισμένες τιμές για το όνομα χρήστη και χωρίς κωδικό πρόσβασης γράφουμε στην οθόνη του τερματικού το παρακάτω:

```
mysql
```

Μετά το mysql είναι δυνατό να ακολουθήσει μία λίστα παραμέτρων. Για να δούμε τις παραμέτρους που προσδιορίζονται από ένα πρόγραμμα MySQL το εκτελούμε με την παράμετρο –help.

```
mysql –help
```

2.5.2 Ορισμός host

Η παρακάτω παράμετρο προσδιορίζει τον υπολογισστή στον οποίο εκτελείται ο MySQL server. Η τιμή μπορεί να είναι ένα όνομα host ή ένας αριθμός IP. Το όνομα host localhost ορίζει ως host τον υπολογιστή στον οποίο εκτελείται το πρόγραμμα client και είναι η προκαθορισμένη τιμή της παραμέτρου host σε περίπτωση που παραληφθεί να δηλωθεί η τιμή της.

```
--host=όνομα_host          ή     -h όνομα_host
```

2.5.3 Ορισμός χρήστη και κωδικού πρόσβασης

Δύο παράμετροι για τις οποίες θα γίνει λόγος παρακάτω παρέχουν πληροφορίες ταυτότητας του χρήστη. Δείχνουν το όνομα χρήστη και τον κωδικό πρόσβασης που πιθανόν χρειάζεται για να προσπελάσουμε τον server.

Ο ορισμός χρήστη μπορεί να γίνει μέσο της παρακάτω παραμέτρου

```
--user=όνομα_χρήστη      ή      –u όνομα_χρήστη
```

Ο προσδιορισμός του κωδικού χρήστη γίνεται με την επόμενη παράμετρο

```
--password=τιμή_κωδικού  ή     -pτιμή_κωδικού
```

Αν παραληφθεί αυτή η επιλογή, ο λογαριασμός του χρήστη στη MySQL θα πρέπει να έχει οριστεί έτσι ώστε να επιτρέπει την σύνδεση χωρίς κωδικό πρόσβασης. Να σημειώσουμε ότι αν χρησιμοποιηθεί η παράμετρος –p τότε δε θα πρέπει να παρεμβάλλεται κενό μεταξύ του –p και της τιμής του κωδικού.

Έτσι για να συνδεθούμε με τον server με όνομα χρήστη student και κωδικό πρόσβασης teikav γράφουμε

```
mysql –u  student –pteikav    ή
mysql --user=student --password=teikav
```

Επιλογή προκαθορισμένης βάσης δεδομένων

Για να ορίσουμε τη βάση δεδομένων που θέλουμε να χρησιμοποιήσουμε μπορούμε στο τέλος των παραμέτρων να γράψουμε το όνομά της. Στο παρακάτω παράδειγμα ορίζουμε ως προκαθορισμένη βάση δεδομένων την βάση με το όνομα mydatabase

```
mysql –u student mydatabase
```

2.5.4 Σύνδεση με τη MySQL ως χρήστης root

Για να συνδεθούμε στη MySQL γράφουμε την παρακάτω εντολή

```
mysql –u root
```

Με την παραπάνω εντολή συνδεθήκαμε ως ο προκαθορισμένος χρήστης root για τον οποίο δεν απαιτείται η σύνδεση μέσω κωδικού πρόσβασης.

Αν έχουμε συνδεθεί σωστά στη MySQL στην οθόνη του τερματικού θα εμφανιστεί το σύμβολο προτροπής

mysql>

που υποδηλώνει ότι το πρόγραμμα είναι έτοιμο να δεχτεί εσωτερικές εντολές (ερωτήματα).

2.6 Εντολές

Για να εκδώσουμε κάποια εντολή πρέπει να την εισάγουμε μετά από την παραπάνω προτροπή. Κάθε εντολή πρέπει να τελειώνει με τον χαρακτήρα ; (ερωτηματικό). Αυτός ο χαρακτήρας δηλώνει ότι η εντολή είναι έτοιμη για αποστολή στον server. Ο server επεξεργάζεται την εντολή και στέλνει την απάντηση πίσω στον client ο οποίος προβάλλει το αποτέλεσμα.

Οι εντολές της SQL που ακολουθούν γράφονται με κεφαλαία γράμματα αλλά είναι δυνατή και η σύνταξή τους με μικρά γράμματα. Παραθέτουμε μερικές χρήσιμες εντολές της MySQL

`SHOW DATABASES;`

Εμφανίζει όλες τις βάσεις που υπάρχουν στο server (όσες τουλάχιστον έχουμε δικαιοδοσία να δούμε)

`USE sample_db;`

Επιλέγουμε τη βάση sample_db για να δουλέψουμε με αυτή.

`SELECT DATABASE( );`

Εμφανίζει την τρέχουσα βάση δεδομένων.

`SHOW TABLES;`

Εμφανίζει όλους τους «πίνακες» που υπάρχουν στη βάση που έχουμε επιλέξει

`DESCPIBE sample_table;`

Εμφανίζει τη δομή των δεδομένων του πίνακα sample_table.

`\h`

Συνοπτική βοήθεια.

`\q`

Έξοδος από τη MySQL.

```
CREATE DATABASE sample_db;
```

Δημιουργία της βάσης sample_db

```
DROP DATABASE sample_db;
```

Διαγραφή της βάσης sample_db

ΚΕΦΑΛΑΙΟ 3

ΕΡΩΤΗΜΑΤΑ ΕΠΙΛΟΓΗΣ

3.1 Τι είναι ένα ερώτημα

Ένα ερώτημα είναι η διαδικασία της ανάκτησης δεδομένων από μία ΒΔ, χρησιμοποιώντας την εντολή SELECT. Ένα ερώτημα χρησιμοποιείται για να εξάγει δεδομένα από μία ΒΔ σε ευανάγνωστη μορφή ανάλογα με την αίτηση του χρήστη. Οι ανακτήσεις είναι οι πιο συνηθισμένη πράξη στις ΒΔ, είναι σημαντικό λοιπόν να κατανοήσουμε πλήρως πως λειτουργεί η SELECT και τι μπορούμε να κάνουμε με αυτήν.

Η γενική συντακτική δομή της εντολής SELECT είναι:

```
SELECT [ DISTINCT | ALL ] { * | <πεδία> }
FROM <όνομα πίνακα> [ { , <όνομα πίνακα> } ... ]
[ WHERE <συνθήκη> ]
[ GROUP BY <πεδία> ]
[ HAVING <συνθήκη σε σχέση με το GROUP BY> ]
[ ORDER BY <πεδία [ASC|DESC]> ]
[ LIMIT <αριθμός εμφανιζόμενων γραμμών>] ;
```

3.2 Προβολή και προσδιορισμός των προς ανάκτηση στηλών

3.2.1 Χρήση της SELECT για ανάκτηση δεδομένων

Η παραπάνω σύνταξη περιλαμβάνει επιπλέον όρους που δεν θα μας απασχολήσουν σε αυτό το κεφάλαιο. Η απλούστερη σύνταξη της εντολής SELECT είναι η ακόλουθη:

```
SELECT <πεδία>
FROM <πίνακες>
```

Η πρόταση SELECT καθορίζει τις στήλες που θέλουμε να ανακτήσουμε. Η πρόταση του FROM καθορίζει τους πίνακες από όπου θα πάρουμε τις στήλες. Η λίστα του SELECT αποτελείται από ονόματα στηλών χωρισμένα μεταξύ τους με κόμμα ή με το σύμβολο * που δηλώνει την επιλογή όλων των στηλών με τη σειρά που ορίστηκαν κατά τη δημιουργία του πίνακα.

☞ Στα παραδείγματα που ακολουθούν θα αναφερόμαστε στη ΒΔ που παρουσιάστηκε στο πρώτο κεφάλαιο των σημειώσεων. Υποθέτουμε ότι έχουμε εισέλθει στο πρόγραμμα mysql και έχουμε επιλέξει ως ΒΔ την ΒΔ **world**.

Για να δούμε τα περιεχόμενα του πίνακα City εισάγουμε την εντολή:

```
SELECT *
FROM City;                                                    (3.1)
```

```
+-----+----------------------------------+-------------+----------------------+------------+
| ID  | Name                             | CountryCode | District             | Population |
+-----+----------------------------------+-------------+----------------------+------------+
|   1 | Kabul                            | AFG         | Kabol                |    1780000 |
|   2 | Qandahar                         | AFG         | Qandahar             |     237500 |
|   3 | Herat                            | AFG         | Herat                |     186800 |
|   4 | Mazar-e-Sharif                   | AFG         | Balkh                |     127800 |
|   5 | Amsterdam                        | NLD         | Noord-Holland        |     731200 |
|   6 | Rotterdam                        | NLD         | Zuid-Holland         |     593321 |
|   7 | Haag                             | NLD         | Zuid-Holland         |     440900 |
```

- Εδώ γράψαμε τις δεσμευμένες λέξεις της SQL (SELECT, FROM) με κεφαλαία γράμματα. Αυτό δεν είναι απαραίτητο. Θα μπορούσαν να γραφτούν και με μικρά:

```
select *
from City;                                                    (3.2)
```

- Ωστόσο δεν ισχύει το ίδιο και για τα ονόματα των πινάκων και των στηλών. Αυτά θα πρέπει να γράφονται όπως ακριβώς έχουν δηλωθεί κατά τη δημιουργία του πίνακα. Έτσι το όνομα CITY δεν θα γίνει αποδεκτό από την SQL εφόσον ο πίνακας είχε δηλωθεί κατά τη δημιουργία του ως City.

- Επίσης δεν είναι δυνατό να γίνει αλλαγή σειράς μεταξύ του SELECT και του FROM.

- Στα παραπάνω παραδείγματα η εντολή συντάχτηκε σε δύο σειρές. Θα μπορούσε να συνταχτεί και σε μία ή σε παραπάνω χωρίς κανένα πρόβλημα αρκεί στο τέλος της εντολής να υπήρχε ο χαρακτήρας τέλους **;**

- Το ίδιο αποτέλεσμα θα παίρναμε αν βάζαμε αναλυτικά όλα τα πεδία του πίνακα μετά τη λέξη select χωρίζοντάς τα με κόμμα.

```
SELECT ID, Name, CountryCode, District, Population
FROM City;                                          (3.3)
```

3.2.2 Επιλέγοντας πεδία στην ερώτηση

Αν θέλουμε να ανακτήσουμε τα δεδομένα από συγκεκριμένες στήλες ενός πίνακα τότε μετά τη SELECT εισάγουμε μόνο τα ονόματα των επιλεγμένων στηλών με τη σειρά που θέλουμε να εμφανιστούν. Π.χ.

```
SELECT Name, Population
FROM City;                                          (3.4)
```

με αποτέλεσμα

```
+--------------------------------+------------+
| Name                           | Population |
+--------------------------------+------------+
| Kabul                          |    1780000 |
| Qandahar                       |     237500 |
| Herat                          |     186800 |
| Mazar-e-Sharif                 |     127800 |
| Amsterdam                      |     731200 |
| Rotterdam                      |     593321 |
```

3.2.3 Μετονομασία ανακτηθεισών στηλών

Όταν εμφανίζονται τα αποτελέσματα ενός ερωτήματος το προκαθορισμένο όνομα της κάθε στήλης είναι αυτό το οποίο δόθηκε κατά τη δημιουργία του πίνακα. Μπορούμε να μετονομάσουμε το εμφανιζόμενο όνομα μίας στήλης δίνοντας ένα ψευδώνυμο μετά τη στήλη στη λίστα με τα πεδία.

```
SELECT Name AS Poli, Population 'Plythismos Polis'
FROM City;                                          (3.5)
```

```
+--------------------------------+------------------+
| Poli                           | Plythismos Polis |
+--------------------------------+------------------+
| Kabul                          |          1780000 |
| Qandahar                       |           237500 |
| Herat                          |           186800 |
| Mazar-e-Sharif                 |           127800 |
| Amsterdam                      |           731200 |
| Rotterdam                      |           593321 |
| Haag                           |           440900 |
```

- Η λέξη AS είναι προαιρετική
- Το ψευδώνυμο πρέπει να τοποθετείται οπωσδήποτε μέσα σε εισαγωγικά αν αποτελείται από παραπάνω από μία λέξεις χωρισμένες με κενό. Σε αντίθετη περίπτωση μπορεί να δηλωθεί και χωρίς εισαγωγικά
- Μπορούμε να αναφερόμαστε στο ψευδώνυμο μίας στήλης οπουδήποτε μέσα στο ερώτημα εκτός του όρου WHERE.

3.2.4 Εμφάνιση υπολογιζόμενων τιμών

Μπορούμε να εκτελέσουμε υπολογισμούς σε στήλες που έχουν αριθμητικά δεδομένα. Το επόμενο παράδειγμα εμφανίζει τον διπλάσιο πληθυσμό κάθε πόλης.

```
SELECT Name AS Poli, Population*2 'Diplasios Plythismos'
FROM City;                                              (3.6)
```

Ας δούμε ακόμα ένα παράδειγμα:

```
SELECT 120 As Timi, 120*0.19 AS FPA                     (3.7)
```

```
+------+-------+
| Timi | FPA   |
+------+-------+
|  120 | 22.80 |
+------+-------+
```

Στο παράδειγμα (3.7) εμφανίζονται δύο στήλες μία με την Τιμή ενός προϊόντος , και η δεύτερη με το ΦΠΑ που αντιστοιχεί σε αυτή την τιμή.

3.2.5 Προσδιορισμός της ΒΔ που περιέχει έναν πίνακα

Όταν ονομάζουμε έναν πίνακα σε μία εντολή SELECT θεωρούμε ότι ανήκει στην προκαθορισμένη βάση δεδομένων. Αν όμως δεν υπάρχει προκαθορισμένη βάση δεδομένων η εντολή δεν μπορεί να βρει τον ζητούμενο πίνακα επιστρέφοντας μήνυμα λάθους :

```
SELECT * FROM City;                                     (3.8)
ERROR 1046 (3D00): No database selected
```

Παρόμοιο μήνυμα λάθους συμβαίνει όταν βρισκόμαστε σε άλλη βάση από αυτήν την οποία περιέχει τον πίνακα τον οποίο θέλουμε να διαβάσουμε.

```
USE vasi2;
SELECT * FROM City;                                    (3.9)
ERROR 1146 (42502): Table 'vasi2.City' doesn't exist
```

Μπορούμε να προσδιορίσουμε ρητά τη βάση δεδομένων από την οποία θα προσπαθήσει η SELECT να ανακτήσει τον πίνακα. Αυτό μπορεί να γίνει αν πριν το όνομα του πίνακα δώσουμε το όνομα της βάσης δεδομένων και μία τελεία :

```
SELECT * FROM world.City;                              (3.10)
```

3.3 Προβολή και προσδιορισμός των προς ανάκτηση γραμμών

3.3.1 Ταξινόμηση αποτελεσμάτων με χρήση της ORDER BY

Εξ ορισμού τα αποτελέσματα που επιστρέφονται από μία εντολή SELECT εμφανίζονται χωρίς κάποια συγκεκριμένη σειρά. Ωστόσο μπορούμε να τα ταξινομήσουμε με την βοήθεια της **ORDER BY**. Η ταξινόμηση μπορεί να είναι αλφαβητική, αριθμητική ή με βάση τη σειρά δήλωσης συγκεκριμένων τιμών σε ένα πεδίο. Το παράδειγμα 3.11 εμφανίζει τις Πόλεις κατά αλφαβητική σειρά.

```
SELECT Name FROM City
ORDER BY Name;                                         (3.11)
```

Η ταξινόμηση εξορισμού γίνεται κατά αύξουσα σειρά. Προκειμένου να εμφανιστούν τα αποτελέσματα κατά φθίνουσα σειρά χρησιμοποιούμε τον όρο DESC μετά το όνομα της στήλης ταξινόμησης. Το παρακάτω παράδειγμα ταξινομεί τις πόλεις με βάση τον πληθυσμό τους εμφανίζοντας πρώτη αυτή με τον μεγαλύτερο πληθυσμό.

```
SELECT Name FROM City
ORDER BY Population DESC;                              (3.12)
```

Η ταξινόμηση μπορεί να γίνει σε περισσότερες από μία στήλες. Σε αυτή την περίπτωση οι γραμμές με την ίδια τιμή στην πρώτη στήλη ταξινόμησης εμφανίζονται συνεχόμενα και ταξινομούνται περαιτέρω με βάση την δεύτερη στήλη ταξινόμησης κ.ο.κ. Στο παράδειγμα

3.13 εμφανίζονται οι γλώσσες κάθε χώρας ταξινομημένες πρώτα κατά κωδικό χώρας και έπειτα κατά όνομα γλώσσας.

```
SELECT CountryCode, Language FROM CountryLanguage
ORDER BY CountryCode, Language;                    (3.13)
```

Αν η ταξινόμηση γίνεται με βάση στήλες τύπου ENUM ή SET τότε η σειρά ταξινόμησης θα αντιστοιχεί στη σειρά με την οποία έχουν δηλωθεί η πιθανές τιμές των στηλών κατά την δημιουργία του πίνακα. Η παρακάτω εντολή θα εμφανίσει τις ηπείρους όχι κατά αλφαβητική σειρά αλλά κατά σειρά δήλωσης στη δημιουργία του πίνακα.

```
SELECT Continent FROM Country
ORDER BY Continent;                                (3.14)
```

Αν θέλουμε η ταξινόμηση να γίνει αλφαβητικά τότε θα χρησιμοποιήσουμε την CAST( ) έτσι ώστε να μετατραπούν οι τιμές του πεδίου σε τιμές CHAR.

```
SELECT Continent FROM Country
ORDER BY CAST(Continent AS CHAR);                  (3.15)
```

3.3.2 Περιορισμός πολλαπλών αποτελεσμάτων με χρήση της DISTINCT

Αν ζητήσουμε να εμφανιστούν τα ονόματα των ηπείρων θα παρατηρήσουμε ότι υπάρχουν πολλαπλές εμφανίσεις αυτών ανάλογα με το πόσες χώρες υπάρχουν σε κάθε ήπειρο.

```
SELECT Continent FROM Country;                     (3.16)
```

```
+---------------+
| Continent     |
+---------------+
| Asia          |
| Europe        |
| North America |
| Europe        |
| Africa        |
| Oceania       |
| Europe        |
| Africa        |
| North America |
| North America |
| Asia          |
| South America |
| Asia          |
| North America |
| Oceania       |
```

Προκειμένου να εξαλείψουμε τις πολλαπλές τιμές και κάθε ήπειρο να εμφανίζεται μόνο μία φορά χρησιμοποιούμε την DISTINCT

```
SELECT DISTINCT Continent FROM Country;          (3.17)
```

```
+---------------+
| Continent     |
+---------------+
| Asia          |
| Europe        |
| North America |
| Africa        |
| Oceania       |
| South America |
| Antarctica    |
+---------------+
```

3.3.3 Καθορισμός πλήθους εγγραφών προς εμφάνιση

Με τον όρο **LIMIT** μπορούμε να δηλώσουμε τον αριθμό των γραμμών που θα εμφανιστούν. Αυτό είναι χρήσιμο για την ανάκτηση εγγραφών βάση της θέσης τους μέσα στο σύνολο των επιλεγμένων γραμμών. Η LIMIT μπορεί να δοθεί με τη μορφή ενός ή δύο ορισμάτων.

Στην μορφή με το ένα όρισμα δηλώνουμε τον αριθμό των γραμμών που θέλουμε να εμφανιστούν. Η παρακάτω εντολή εμφανίζει μόνο τις δέκα πρώτες γραμμές των αποτελεσμάτων.

```
SELECT Name FROM City LIMIT 10;                  (3.25)
```

Μπορούμε να χρησιμοποιήσουμε την LIMIT σε συνδυασμό με την ORDER BY για να εμφανίσουμε αποτελέσματα με βάση συγκεκριμένα στατιστικά. Έτσι αν θέλουμε να εμφανίσουμε τις 3 πολυπληθέστερες πόλεις του κόσμου συντάσσουμε την ακόλουθη εντολή.

```
SELECT Name FROM City
ORDER BY Population DESC LIMIT 3;                (3.26)
```

Αν θέλουμε να εμφανιστούν οι πόλεις που με βάση τον πληθυσμό τους κατατάσσονται από την 4 έως την 10 θέση τότε θα χρησιμοποιήσουμε την LIMIT με δύο ορίσματα. Το πρώτο

θα δηλώνει τις γραμμές που αφήνουμε χωρίς να εμφανιστούν και το δεύτερο τον αριθμό των γραμμών που θα εμφανιστούν. Η εντολή θα έχει την παρακάτω μορφή.

```
SELECT Name FROM City
ORDER BY Population DESC LIMIT 3,7;                (3.27)
```

3.3.4 Επιλογή γραμμών με την πρόταση WHERE

Γενική μορφή της πρότασης **WHERE** είναι

WHERE <έκφραση> συγκριτικός τελεστής <έκφραση>

Στον πίνακα 3.1 που ακολουθεί εμφανίζονται οι συγκριτικοί τελεστές που χρησιμοποιούνται από τη MySQL.

Πίνακας 3.1 Συγκριτικοί Τελεστές

Τελεστής	Σημασία
=	ίσον
>	μεγαλύτερο
<	μικρότερο
>=	μεγαλύτερο ή ίσο
<=	μικρότερο ή ίσο
!=	όχι ίσο
<>	διάφορο
!>	όχι μεγαλύτερο
!<	όχι μικρότερο

Έστω ότι θέλουμε να εμφανίσουμε όλες τις χώρες με πληθυσμό μεγαλύτερο των 100.000.000 κατοίκων.

```
SELECT Name FROM Country
WHERE Population > 100000000;                      (3.18)
```

```
+--------------------+
| Name               |
+--------------------+
| Bangladesh         |
| Brazil             |
| Indonesia          |
| India              |
| Japan              |
| China              |
| Nigeria            |
| Pakistan           |
| Russian Federation |
| United States      |
+--------------------+
```

3.3.5 Σύνδεση συνθηκών με λογικούς τελεστές

Οι τελεστές **AND, OR** και **NOT** χρησιμοποιούνται για να συνδέσουμε συνθήκες σε προτάσεις που περιέχουν το WHERE. Ακολουθούνε σχετικά παραδείγματα.

Στο ερώτημα 3.19 εμφανίζουμε όλες τις χώρες με πληθυσμό από 10.000.000 έως 150.000.000. Επίσης τις ταξινομούμε με φθίνουσα σειρά πληθυσμού.

```
SELECT Name FROM Country
WHERE Population >= 10000000
AND Population<=150000000
ORDER BY Population DESC;                                    (3.19)
```

```
+--------------------+
| Name               |
+--------------------+
| Russian Federation |
| Bangladesh         |
| Nigeria            |
| Japan              |
+--------------------+
```

Για εμφάνιση τιμών σε ένα διάστημα μπορεί να χρησιμοποιηθεί και ο όρος **BETWEEN**. Το παράδειγμα 3.20 μπορεί να γραφτεί λοιπόν ως εξής

```
SELECT Name FROM Country
WHERE Population BETWEEN 10000000 AND 150000000;   (3.20)
```

Το παράδειγμα 3.21 εμφανίζει όλες τις χώρες εκτός Ευρώπης χρησιμοποιώντας τον τελεστή NOT.

```
SELECT Name FROM Country
WHERE NOT Continent='Europe';                                (3.21)
```

Το παράδειγμα 3.22 εμφανίζει όλες τις χώρες της Ευρώπης και της Ασίας χρησιμοποιώντας τον τελεστή OR.

```
SELECT Name FROM Country
WHERE Continent='Europe' OR Continent='Asia';      (3.22)
```

Όταν υπάρχουν περισσότεροι από ένας λογικοί τελεστές σε μία συνθήκη τότε η σειρά προτεραιότητας είναι

1. NOT
2. AND
3. OR

Για να ελέγξουμε τη σειρά αποτίμησης μπορούμε να χρησιμοποιήσουμε παρενθέσεις για να ομαδοποιήσουμε τους όρους εκφράσεων.

```
SELECT Name, Population FROM Country
WHERE Population >10000000
AND Continent = 'Europe'
OR Continent = 'Asia';                                    (3.23)
```

Η παραπάνω εντολή βρίσκει όλες τις χώρες της Ευρώπης που έχουν πληθυσμό πάνω από 10.000.000 όπως και τις χώρες της Ασίας. Εναλλακτικά θα μπορούσε να γραφτεί ως εξής

```
SELECT Name, Population FROM Country
WHERE (Population >10000000 AND Continent = 'Europe')
OR Continent = 'Asia';                                    (3.24)
```

3.3.6 Χρησιμοποίηση του LIKE για ταίριασμα κειμένου

Έως τώρα ελέγχαμε αν οι τιμές ενός πεδίου ταυτίζονται με αυτές ενός δοθέντος κειμένου. Αν είναι απαραίτητο να βρεθούν τιμές βάση ομοιότητας κάποιων κειμένων τότε είναι χρήσιμη μία ταύτιση σε κάποιο πρότυπο. Για να εκτελέσουμε μία ταύτιση με βάση κάποιο πρότυπο χρησιμοποιούμε το

πεδίο|τιμή LIKE 'πρότυπο'

όπου πεδίο είναι το όνομα της στήλης την οποία θέλουμε να ελέγξουμε ή κάποια τιμή προς έλεγχο και 'πρότυπο' η συμβολοσειρά ταύτισης που περιγράφει την γενική μορφή τιμών. Τα πρότυπα μπορούν να έχουν δύο ειδικούς χαρακτήρες

- Τον χαρακτήρα **%** όπου υποκαθιστά πολλούς χαρακτήρες
- Τον χαρακτήρα _ όπου υποκαθιστά έναν οποιοδήποτε χαρακτήρα

Ακολουθούν μερικά παραδείγματα χρήσης του LIKE.

```
Select * FROM Country WHERE Name LIKE 'G%'                (3.25)
+---------------+
| Name          |
+---------------+
| Gabon         |
| Gambia        |
```

```
| Georgia       |
| Ghana         |
| Gibraltar     |
| Grenada       |
| Greenland     |
| Guadeloupe    |
| Guam          |
| Guatemala     |
| Guinea        |
| Guinea-Bissau |
| Guyana        |
| Greece        |
| Germany       |
+---------------+
```

```
Select * FROM Country WHERE Name LIKE 'G%e'        (3.26)
+------------+
| Name       |
+------------+
| Guadeloupe |
| Greece     |
+------------+
```

```
Select * FROM Country WHERE Name LIKE 'Gr__c_'     (3.27)

+--------+
| Name   |
+--------+
| Greece |
+--------+
```

Οι δύο ειδικοί χαρακτήρες μπορούν να συνδυαστούν και μαζί σε ένα πρότυπο. Ακολούθως παρατίθενται διάφορα πρότυπα με τις επεξηγήσεις τους.

'_r%'	ψάχνει για συμβολοσειρές τουλάχιστον δύο χαρακτήρων με τον δεύτερο χαρακτήρα να είναι ο r.
'_%_s' ή ´_ _%s'	ψάχνει για συμβολοσειρές που τελειώνουν με s και έχουν τουλάχιστον τρεις χαρακτήρες
'_ _ _ _as%'	ψάχνει για συμβολοσειρές που περιέχουν τους χαρακτήρες 'as' στην 5η και 6η θέση των χαρακτήρων τους.

3.3.7 Απροσδιόριστες τιμές (NULL values)

Όταν υπάρχουν απροσδιόριστες τιμές (NULL) σε ένα πεδίο σημαίνει είτε ότι οι τιμές είναι άγνωστες ή ότι δεν είναι διαθέσιμες. Για να εμφανιστούν οι απροσδιόριστες ή οι μη απροσδιόριστες τιμές χρησιμοποιούμε στην πρόταση του WHERE το παρακάτω

WHERE <όνομα στήλης> IS [NOT] NULL

Στο παράδειγμα που ακολουθεί εμφανίζονται οι χώρες στις οποίες δεν έχει καταχωρηθεί το προηγούμενο ΑΕΠ (GNPOld)

```
Select Name FROM Country
WHERE GNPOld IS NULL;                                    (3.28)
```

Στο παράδειγμα 3.29 εμφανίζονται μόνο τα κράτη για τα οποία έχει καταχωρηθεί το μέσο προσδόκιμο ζωής (LifeExpectancy) όπως και το προσδόκιμο ζωής αυτών.

```
Select Name, LifeExpectancy FROM Country
WHERE LifeExpectancy IS NOT NULL;                        (3.29)
```

3.3.8 Λίστες με (IN και NOT IN)

Οι λέξεις κλειδιά IN και NOT IN μας επιτρέπουν να ελέγχουμε αν κάποιες τιμές ταιριάζουν ή όχι με μία λίστα τιμών αντίστοιχα.

Αν για παράδειγμα θέλουμε να εμφανίσουμε τις χώρες της Ευρώπης της Ασίας και της Αφρικής σύμφωνα με αυτά που έχουμε δει θα δίναμε την εντολή

```
SELECT Name FROM Country
WHERE Continent='Europe'
OR Continent='Asia'
OR Continent = 'Africa';                                 (3.30)
```

Χρησιμοποιώντας το IN μπορούμε να πάρουμε το ίδιο αποτέλεσμα με το ερώτημα

```
SELECT Name FROM Country
WHERE Continent IN ('Europe','Asia','Africa');           (3.31)
```

Με το επόμενο παράδειγμα επιστρέφονται όλες οι χώρες που βρίσκονται εκτός της Ευρώπης και της Ασίας

```
SELECT Name FROM Country
WHERE Continent NOT IN ('Europe','Asia');          (3.32)
```

ΚΕΦΑΛΑΙΟ 4

ΣΥΝΔΙΑΣΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

4.1 Συνδυαστικές συναρτήσεις

Κάποιες φορές θέλουμε να εμφανίσουμε κάποια συγκεντρωτικά αποτελέσματα από μία ομάδα εγγραφών. Π.χ. τον συνολικό πληθυσμό των χωρών της Ευρώπης ή το μέσο προσδόκιμο ζωής όλων των χωρών. Σε αυτές τις περιπτώσεις όπου θέλουμε να υπολογιστούν κάποιες τιμές σύνοψης χρησιμοποιούμε τις συνδυαστικές συναρτήσεις.

Υπάρχουν έξι συνδυαστικές συναρτήσεις για να μπορούμε να βρίσκουμε ποσότητες όπως το πλήθος, άθροισμα, μεγαλύτερος, μικρότερος, μέσος. Δέχονται ως ορίσματα ονόματα πεδίων και επιστρέφουν τις αντίστοιχες ποσότητες. Η σύνταξη των συνδυαστικών συναρτήσεων και τα αποτελέσματα που παράγουν φαίνονται στον πίνακα 4.1 που ακολουθεί:

Πίνακας 4.1 Συνδυαστικές συναρτήσεις και αποτελέσματα

Συνδυαστική Συνάρτηση	*Αποτελέσματα*
SUM ([all/distinct]έκφραση)	Το άθροισμα τιμών αριθμητικής στήλη
AVG ([all/distinct]έκφραση)	Ο μέσος όρος τιμών αριθμητικής στήλης
COUNT ([all/distinct]έκφραση)	Το πλήθος των εγγραφών μίας στήλης με NOT NULL τιμές
COUNT (*)	Το πλήθος των εγγραφών του αποτελέσματος
MAX (έκφραση)	Η μεγαλύτερη τιμή της στήλης
MIN (έκφραση)	Η μικρότερη τιμή της στήλης

4.1.1 Υπολογίζοντας το άθροισμα και τον μέσο όρο

Η συνάρτηση SUM( ) υπολογίζει το άθροισμα των τιμών ενός αριθμητικού πεδίου (int, float κτλ) και επιστρέφει το αποτέλεσμα στον ίδιο τύπο δεδομένων. Στο παράδειγμα 4.1 βρίσκουμε τον συνολικό πληθυσμό όλων των χωρών της γης ενώ στο παράδειγμα 4.2 βρίσκουμε τον πληθυσμό της Ευρώπης

```
SELECT SUM(Population) FROM Country;                    (4.1)
```

```
SELECT SUM(Population) FROM Country
WHERE Continent='Europe' ;                              (4.2)
```

Αντίστοιχα η συνάρτηση AVG( ) υπολογίζει το μέσο τιμών ενός αριθμητικού πεδίου. Η AVG( ) είναι δυνατό να επιστρέψει δεκαδικό αριθμό ακόμα και αν ο τύπος του πεδίου στο όρισμά της είναι ακέραιος. Στο παρακάτω παράδειγμα υπολογίζονται ο συνολικός πληθυσμός της γης και ο μέσος πληθυσμός ανά χώρα στην ίδια εντολή.

```
SELECT SUM(Population),AVG(Population) FROM Country;  (4.3)
```

4.1.2 Υπολογίζοντας τη μικρότερη και μεγαλύτερη τιμή

Οι συναρτήσεις **MIN( )** και **MAX( )** επιστρέφουν την μικρότερη και μεγαλύτερη τιμή ενός πεδίου. Έχουν τη δυνατότητα να εφαρμοστούν εκτός από αριθμητικά πεδία και σε πεδία κειμενου. Στην δεύτερη περίπτωση βρίσκουν τη πρώτη και τελευταία λέξη με αλφαβητική σειρά αντίστοιχα. Έτσι το παράδειγμα 4.4 βρίσκει τον μικρότερο και το μεγαλύτερο πληθυσμό που έχει κάποια χώρα ενώ το 4.5 εμφανίζει την πρώτη και την τελευταία χώρα αλφαβητικά.

```
SELECT MIN(Population),MAX(Population) FROM Country; (4.4)
+-----------------+-----------------+
| MIN(Population) | MIN(Population) |
+-----------------+-----------------+
|               0 |      1277558000 |
+-----------------+-----------------+
```

```
SELECT MIN(Name),MAX(Name) FROM Country;                (4.5)
+------------+-----------+
| MIN(Name)  | MAX(Name) |
+------------+-----------+
| Afganistan | Zimbabwe  |
+------------+-----------+
```

4.1.3 Καταμετρώντας το πλήθος

Η **COUNT( )** μπορεί με πολλούς τρόπους να μετρήσει γραμμές ή τιμές. Η **COUNT(*)** δεν χρειάζεται σαν όρισμα κάποια έκφραση αφού εξ ορισμού δεν χρησιμοποιεί πληροφορία για κάποια στήλη. Επιστρέφει τον συνολικό αριθμό γραμμών σε ένα συγκεκριμένο πίνακα

χωρίς να περιορίζει τα διπλότυπα. Απαριθμεί κάθε γραμμή χωρίς να αποκλείει τις τιμές NULL. Η παρακάτω εντολή βρίσκει το σύνολο των εγγραφών του πίνακα Country.

```
Select COUNT( *) FROM Country        (4.6)
+----------+
| COUNT(*) |
+----------+
|      239 |
+----------+
```

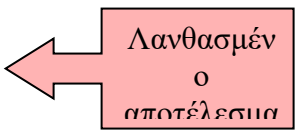
Αν θέλουμε να βρούμε τον αριθμό των μη απροσδιόριστων τιμών (not NULL values) τότε θα πρέπει να χρησιμοποιήσουμε την COUNT(έκφραση). Το παρακάτω παράδειγμα βρίσκει τον αριθμό των χωρών για τις οποίες έχει εισαχθεί το μέσο προσδόκιμο ζωής των κατοίκων τους.

```
Select COUNT( LifeExpectancy) FROM Country          (4.7)
+----------------------+
| COUNT(LifeExpectancy)|
+----------------------+
|                  222 |
+----------------------+
```

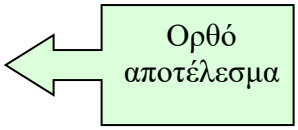
4.1.4 Χρήση των συνδυαστικών συναρτήσεων με DISTINCT

Η DISTINCT είναι προαιρετική με τις συναρτήσεις SUM, AVG και COUNT. Ωστόσο δεν επιτρέπεται να χρησιμοποιηθεί με τις MIN, MAX. και COUNT(*). Η DISTINCT περιορίζει τις διπλότυπες τιμές πριν τον υπολογισμό των αποτελεσμάτων από τις συναρτήσεις. Έτσι το αποτέλεσμα δεν θα επηρεάζεται από διπλότυπες τιμές. Το DISTINCT μπαίνει μέσα στις παρενθέσεις πριν το όνομα της στήλης. Στο παρακάτω παράδειγμα εμφανίζεται ο αριθμός των ηπείρων με ή χωρίς χρήση της Distinct. Παρατηρούμε ότι αν δεν εισάγουμε το DISTINCT το αποτέλεσμα είναι λανθασμένο.

```
Select COUNT( Continent) FROM Country;              (4.8)
+------------------+
| COUNT(Continent) |
+------------------+
|              239 |
+------------------+
```




```
Select COUNT( DISTINCT Continent) FROM Country;     (4.9)
+--------------------------+
| COUNT(DISTINCT Continent) |
+--------------------------+
|                        7 |
+--------------------------+
```



```
+--------------------------+
```

Στο παράδειγμα 4.8 η COUNT μέτρησε τις μη απροσδιόριστες τιμές του πεδίου Continent. Τον αριθμό δηλαδή των εγγραφών που έχουν τιμή σε αυτό το πεδίο. Αντίθετα στο 4.9 η COUNT απαριθμεί τις μοναδικές μη απροσδιόριστες τιμές του πεδίου Continent για αυτό το λόγο επιστρέφει και το σωστό αποτέλεσμα.

4.2 Ομαδοποίηση και συνδυαστικές συναρτήσεις

4.2.1 Ομαδοποίηση με χρήση της GROUP BY

Η GROUP BY χρησιμοποιείται για τον διαχωρισμό ενός πίνακα σε ομάδες (groups). Η ομαδοποίηση των εγγραφών γίνεται με βάση το όνομα ενός ή περισσοτέρων πεδίων ή ακόμα και με βάση τα αποτελέσματα υπολογιζόμενων πεδίων όπου χρησιμοποιούνται αριθμητικοί τύποι δεδομένων. Για παράδειγμα μία βάση δεδομένων μπορεί να περιέχει πληροφορίες για υπαλλήλους. Η εταιρία έχει πολλά τμήματα (departments) και κάθε υπάλληλος ανήκει σε ένα τμήμα. Μπορεί λοιπόν να θέλουμε να εκτελέσουμε ένα ερώτημα που να δείχνει πληροφορίες υπαλλήλων για κάθε συγκεκριμένο τμήμα. Σε αυτή την περίπτωση θα πρέπει να ομαδοποιήσουμε κατά τμήμα και να δημιουργήσουμε μία αναφορά σύνοψης.

Ας υποθέσουμε λοιπόν ότι ένας πίνακας με το όνομα Prosopiko περιέχει τις παρακάτω πληροφορίες

Πίνακας 4.2 Πεδία και δεδομένα του πίνακα Prosopiko

ID	Onoma	Tmima	Poli	Misthos
1	Nikos	Posliseis	Thiva	660
2	Maria	Service	Thessaloniki	680
3	Pavlos	Service	Kavala	1500
4	Spyros	Posliseis	Athina	750
5	Xristina	Posliseis	Drama	700
6	Xrysa	Service	Thessaloniki	680
7	Pantelis	Service	Kavala	2000
8	Mixalis	Service	Athina	2200
9	Sofia	Service	Komotini	1250

Αν εφαρμόσουμε ομαδοποίηση κατά πόλη μπορούμε να υπολογίσουμε ξεχωριστά στοιχεία για τους υπαλλήλους κάθε πόλης. Το παράδειγμα που ακολουθεί βρίσκει πόσους υπαλλήλους έχει η εταιρία σε κάθε πόλη.

```
Select Poli, COUNT(*) FROM Prosopiko
GROUP BY Poli;                                          (4.10)
```

```
+--------------+----------+
| Poli         | COUNT(*) |
+--------------+----------+
| Athina       |        2 |
| Drama        |        1 |
| Kavala       |        2 |
| Komotini     |        1 |
| Thessaloniki |        2 |
| Thiva        |        1 |
+--------------+----------+
```

Είναι δυνατό να γίνει ομαδοποίηση σε περισσότερα του ενός πεδία. Για να βρούμε π.χ. πόσους υπαλλήλους απασχολεί το κάθε τμήμα από την ίδια πόλη θα ομαδοποιήσουμε πρώτα κατά τμήμα και έπειτα κατά πόλη.

```
Select Tmima,Poli, COUNT(*) FROM Prosopiko
GROUP BY Tmima, Poli;                                   (4.11)
```

```
+----------+--------------+----------+
| Tmima    | Poli         | COUNT(*) |
+----------+--------------+----------+
| Poliseis | Athina       |        1 |
| Poliseis | Drama        |        1 |
| Poliseis | Thiva        |        1 |
| Service  | Athina       |        1 |
| Service  | Kavala       |        2 |
| Service  | Komotini     |        1 |
| Service  | Thessaloniki |        2 |
+----------+--------------+----------+
```

Το επόμενο παράδειγμα βρίσκει τον μεγαλύτερο και τον μικρότερο μισθό σε κάθε τμήμα της εταιρίας.

```
Select Tmima, MAX(Misthos),MIN(Misthos) FROM Prosopiko
GROUP BY Tmima;                                         (4.12)
```

```
+----------+--------------+--------------+
| Tmima    | MAX(Misthos) | MIN(Misthos) |
+----------+--------------+--------------+
| Poliseis |          750 |          660 |
| Service  |         2200 |          680 |
+----------+--------------+--------------+
```

☞ Όλες οι στήλες και οι εκφράσεις στην SELECT πρέπει να αναφέρονται στην φράση GROUP BY, με την εξαίρεση των στηλών των συνδυαστικών συναρτήσεων. Δηλαδή αν δεν είχαμε κάνει την ομαδοποίηση κατά τμήμα στο παράδειγμα 4.12 και προσπαθούσαμε να εμφανίσουμε το τμήμα μαζί με το μέγιστο και το ελάχιστο τότε θα επιστρεφόταν μήνυμα λάθους.

4.2.2 Επιλογή ομάδων δεδομένων με την πρόταση HAVING

Η φράση HAVING χρησιμοποιείται όταν στην συνθήκη υπάρχει μία συνδυαστική συνάρτηση. Τοποθετείται πάντα μετά την GROUP BY και ενεργεί όπως περίπου και το WHERE, χρησιμοποιείται όμως σε ένα διαφορετικό στάδιο της διαδικασίας ερωτημάτων. Με άλλα λόγια η φράση WHERE εφαρμόζει συνθήκες σε επιλιγμένες στήλες ενώ η HAVING εφαρμόζει συνθήκες σε ομάδες που δημιουργούνται από την φράση GROUP BY. Συμπερασματικά μπορούμε να πούμε ότι η HAVING είναι για την GROUP BY ότι είναι η WHERE για την SELECT.

Η επόμενη εντολή επιστρέφει το όνομα των πόλεων όπου κατοικούν περισσότεροι του ενός υπάλληλοι της εταιρίας.

```
Select Poli, Count(*) FROM Prosopiko
GROUP BY Poli
HAVING COUNT(*)>1;                                          (4.13)
+--------------+----------+
| Poli         | COUNT(*) |
+--------------+----------+
| Athina       |        2 |
| Kavala       |        2 |
| Thessaloniki |        2 |
+--------------+----------+
```

4.2.3 GROUP BY και ταξινόμηση

Παρατηρούμε στα παραπάνω παραδείγματα ότι τα αποτελέσματα εμφανίζονται ταξινομημένα κατά αλφαβητική σειρά των πεδίων κατά τα οποία ομαδοποιήθηκαν. Η MySQL ταξινομεί αυτόματα κατά αύξουσα σειρά τα αποτελέσματα με βάση τα πεδία που βρίσκονται μετά την GROUP BY. Ωστόσο επειδή αυτό δεν είναι γενικός κανόνας της SQL καλό είναι αν επιθυμούμε ταξινόμηση να χρησιμοποιούμε πάντα το ORDER BY. Οι δύο αυτές εκφράσεις όταν χρησιμοποιούνται στην ίδια πρόταση SELECT ακολουθούν μία συγκεκριμένη σειρά. Η GROUP BY τοποθετείται πρώτα και ακολουθεί η ORDER BY.

Στο παράδειγμα που ακολουθεί ταξινομούμε τα δεδομένα πρώτα κατά τμήμα και έπειτα κατά φθίνουσα σειρά πόλης.

```
Select Tmima,Poli, COUNT(*) FROM Prosopiko
GROUP BY Tmima, Poli
ORDER BY Tmima, Poli DESC;                                    (4.14)
```

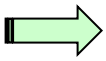
```
+----------+--------------+----------+
| Tmima    | Poli         | COUNT(*) |
+----------+--------------+----------+
| Poliseis | Thiva        |        1 |
| Poliseis | Drama        |        1 |
| Poliseis | Athina       |        1 |
| Service  | Thessaloniki |        2 |
| Service  | Komotini     |        1 |
| Service  | Kavala       |        2 |
| Service  | Athina       |        1 |
+----------+--------------+----------+
```

4.2.4 GROUP BY και WITH ROLLUP

Με την χρήση της WITH ROLLUP στο τέλος του όρου GROUP BY μπορούμε να δημιουργήσουμε πολλά επίπεδα τιμών σύνοψης. Για να γίνει καλύτερα αντιληπτό αν στο παράδειγμα 4.10 προσθέσουμε τον όρο WITH ROLLUP τότε θα προστεθεί ακόμα μία γραμμή η οποία θα εμφανίζει τον συνολικό αριθμό υπαλλήλων όλης της εταιρίας.

```
Select Poli, COUNT(*) FROM Prosopiko
GROUP BY Poli WITH ROLLUP;                                    (4.15)
```

```
+--------------+----------+
| Poli         | COUNT(*) |
+--------------+----------+
| Athina       |        2 |
| Drama        |        1 |
| Kavala       |        2 |
| Komotini     |        1 |
| Thessaloniki |        2 |
| Thiva        |        1 |
| NULL         |        9 |
+--------------+----------+
```



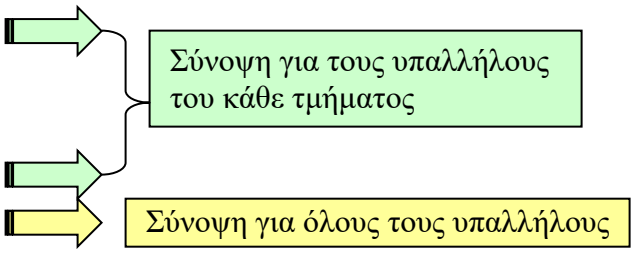
Σύνοψη για όλους τους υπαλλήλους

Αν εφαρμόσουμε το ίδιο στο παράδειγμα 4.14 όπου έχουμε διπλή ομαδοποίηση τότε θα εμφανιστεί σύνοψη αποτελεσμάτων για κάθε επίπεδο ομαδοποίησης.

```
Select Tmima,Poli, COUNT(*) FROM Prosopiko
GROUP BY Tmima, Poli WITH ROLLUP
ORDER BY Tmima, Poli DESC;                                    (4.16)
```

```
+----------+--------------+----------+
| Tmima    | Poli         | COUNT(*) |
```

```
+----------+-------------+----------+
| Poliseis | Thiva       |        1 |
| Poliseis | Drama       |        1 |
| Poliseis | Athina      |        1 |
| Poliseis | NULL        |        3 |
| Service  | Thessaloniki |        2 |
| Service  | Komotini    |        1 |
| Service  | Kavala      |        2 |
| Service  | Athina      |        1 |
| Service  | NULL        |        6 |
| NULL     | NULL        |        9 |
+----------+-------------+----------+
```



4.3 Περισσότερες συναρτήσεις στην SQL

4.3.1 Μαθηματικές συναρτήσεις

Υπάρχουν αρκετές μαθηματικές συναρτήσεις στην SQL που πραγματοποιούν ένα πλήθος μαθηματικών πράξεων. Σε αυτές τις σημειώσεις θα αναλυθούν συναρτήσεις για στρογγυλοποίηση αριθμητικών τιμών όπως η ROUND η FLOOR και η CEILING.

Η ROUND( ) στρογγυλοποιεί το όρισμά της με τον παρακάτω τρόπο. Αν το κλασματικό τμήμα του ορίσματος είναι μεγαλύτερο ή ίσο του .5 τότε στρογγυλοποιεί στον αμέσως μεγαλύτερο ακέραιο αριθμό. Στην αντίθετη περίπτωση στρογγυλοποιεί στον αμέσως κατώτερο ακέραιο αριθμό. Π.χ.

```
Select ROUND(32.5), ROUND(21.4) ;                    (4.17)
+-------------+-------------+
| ROUND(32.5) | ROUND(21.4) |
+-------------+-------------+
|          33 |          21 |
+-------------+-------------+
```

Η FLOOR( ) επιστρέφει τον αμέσως μικρότερο ακέραιο από το όρισμά της όπως στο παράδειγμα 4.18 ενώ αντίθετα η CEILING επιστρέφει τον αμέσως μεγαλύτερο ακέραιο από το όρισμά της

```
Select FLOOR(32.5), CEILING(32.5) ;                  (4.18)
+-------------+---------------+
| FLOOR(32.5) | CEILING(32.5) |
+-------------+---------------+
|          32 |            33 |
+-------------+---------------+
```

4.3.2 Συναρτήσεις ελέγχου ροής

Οι συναρτήσεις ελέγχου ροής μας δίνουν τη δυνατότητα να επιλέγουμε την επιστρεφόμενη τιμή ανάλογα με το αποτέλεσμα μίας έκφρασης. Η πιο συνηθισμένη συνάρτηση αυτού του είδους είναι η IF( ) η οποία συντάσσεται ως ακολούθως:

```
IF( όρισμα 1, όρισμα 2, όρισμα 3)
όπου
όρισμα 1: συνθήκη,
όρισμα 2: επιστροφή αν συνθήκη αληθής,
όρισμα 3: επιστροφή αν συνθήκη ψευδής.
```

Στο επόμενο παράδειγμα ελέγχει αν ο βαθμός είναι μεγαλύτερος ή ίσος του 5 και επιστρέφει το ανάλογο μήνυμα

```
Select IF(vathmos>=5,'Pernaei','Kovetai') FROM grades;(4.19)
```

4.3.3 Συναρτήσεις συμβολοσειράς

Δύο από τις συναρτήσεις συμβολοσειράς που θα δούμε στη συνέχεια είναι η UPPER( ) και η LOWER( ). Αυτές μετατρέπουν τους πεζούς χαρακτήρες μίας μη δυαδικής συμβολοσειράς σε κεφαλαίους και αντίστροφα.

```
Select UPPER(Name), LOWER(Name) FROM Students;     (4.20)
+-------------+-------------+
| UPPER(Name) | LOWER(NAME) |
+-------------+-------------+
| NIKOLAOS    | nikolaos    |
| MARIA       | maria       |
+-------------+-------------+
```

Στην περίπτωση που η συμβολοσειρά είναι δυαδική τότε η εφαρμογή των UPPER( ) και LOWER( ) δεν έχει κάποιο αποτέλεσμα. Σε αυτή την περίπτωση θα πρέπει πρώτα να γίνει μετατροπή της συμβολοσειράς σε μη δυαδική όπως στο παράδειγμα 4.21 χρησιμοποιώντας την CONVERT( ) και τον όρο BINARY.

```
Select UPPER(CONVERT(BINARY Name USING latin1)) FROM
Students;                                          (4.21)
```

ΚΕΦΑΛΑΙΟ 5

ΟΡΙΣΜΟΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ

5.1 Δημιουργία και διαχείριση Βάσεων Δεδομένων

5.1.1 Δημιουργία Βάσης Δεδομένων

Για να δημιουργήσουμε μία νέα βάση δεδομένων χρησιμοποιούμε την εντολή CREATE DATABASE με την παρακάτω σύνταξη

```
CREATE DATABASE OnomaVasis;                    (5.1)
```

Στην περίπτωση όπου υπάρχει ήδη μία βάση με το όνομα το οποίο έχουμε δηλώσει τότε θα εμφανιστεί μήνυμα λάθους. Για να αποφεχθεί μία τέτοια περίπτωση μπορούμε να χρησιμοποιήσουμε τον όρο IF NOT EXISTS. Με αυτό τον τρόπο η βάση θα δημιουργηθεί μόνο εάν δεν υπάρχει ήδη κάποια με το ίδιο όνομα..

```
CREATE DATABASE IF NOT EXISTS imd;             (5.2)
```

Η εντολή 5.2 θα δημιουργήσει μία ΒΔ με το όνομα imd αν δεν υπάρχει ήδη κάποια στον server βάση με αυτό το όνομα.

Με την εντολή CREATE DATABASE είναι δυνατό να δηλώσουμε το προκαθορισμένο σύνολο χαρακτήρων της βάσης (CHARACTER SET) και το τύπο του συνόλου χαρακτήρων (COLLATE). Στο επόμενο παράδειγμα δημιουργείται η βάση δεδομένων imd που έχει ως προκαθορισμένο σύνολο χαρακτήρων το utf8 και τύπο του συνόλου το utf8_danish_ci:

```
CREATE DATABASE imd
CHARACTER SET utf8
COLLATE utf8_danish_ci;                        (5.3)
```

5.1.2 Προβολή και επιλογή Βάσεων Δεδομένων

Για να δούμε ποιες είναι οι διαθέσιμες ΒΔ γράφουμε

```
SHOW DATABASES ;                                    (5.4)
```

Για να χρησιμοποιήσουμε μία βάση δεδομένων γράφουμε την παρακάτω εντολή:

```
USE DATABASE OnomaVasis;                            (5.5)
```

Για να δούμε ποια βάση χρησιμοποιούμε την δεδομένη στιγμή γράφουμε:

```
SELECT DATABASE();                                  (5.6)
```

5.1.3 Αλλαγή στοιχείων Βάσεων Δεδομένων

Είναι δυνατό να αλλάξουμε τις επιλογές του συνόλου χαρακτήρων μίας βάσης και του τύπου του συνόλου με την εντολή ALTER DATABASE. Η εντολή αυτή αλλάζει τις επιλογές σύμφωνα με τη δήλωση που θα κάνουμε. Τυχόν όμως αλλαγές θα εφαρμόζονται μόνο στους πίνακες της βάσης που θα δημιουργηθούνε από εκείνο το σημείο και μετά.

Η παρακάτω εντολή αλλάζει το σύνολο χαρακτήρων σε latin1 τύπου latin1_swedish_ci.

```
ALTER DATABASE imd
CHARACTER SET latin1
COLLATE latin1_swedish_ci;                          (5.7)
```

5.1.4 Διαγραφή Βάσεων Δεδομένων

Τέλος μπορούμε να διαγράψουμε μία βάση δεδομένων με την εντολή DROP DATABASE συμπληρώνοντας με το όνομα της βάσης προς διαγραφή. Το παρακάτω παράδειγμα διαγράφει την ΒΔ imd.

```
DROP DATABASE imd                                   (5.8)
```

5.2 Δημιουργία πινάκων

Για τη δημιουργία πίνακα χρησιμοποιούμε την εντολή CREATE TABLE η οποία δίνει τη δυνατότητα να περιγραφούν οι τύποι δεδομένων και οι περιορισμοί για κάθε πεδίο του

πίνακα, καθώς και να ορισθούν τα κλειδιά και το αν θα επιτρέπονται κενές τιμές για κάποιο πεδίο. Μία γενική σύνταξη της εντολής είναι η ακόλουθη:

```
CREATE TABLE ΌνομαΠίνακα (ορισμός στηλών πίνακα);
Create table <όνομα πίνακα> (
     <πεδίο1> <τύπος δεδομένων> [not null][default <τιμή>],
     <πεδίο2> <τύπος δεδομένων> [not null][default <τιμή>],
     ....................................
     <πεδίοN> <τύπος δεδομένων> [not null][default <τιμή>],
     [<περιορισμοί πίνακα>]
);
```

Για την δημιουργία ενός πίνακα ισχύουν τα εξής:

- Για κάθε πεδίο του πίνακα πρέπει να οριστεί ένα μοναδικό όνομα και ένας τύπος δεδομένων.
- Οι ορισμοί των πεδίων διαχωρίζονται με **κόμμα** (,)
- Δεν υπάρχει διαχωρισμός κεφαλαίων-πεζών (case insensitive)
- Ο περιορισμός **NOT NUL** πρέπει να μπαίνει αμέσως μετά τον τύπο δεδομένων

Στο όρισμα του πίνακα δηλώνουμε τα ονόματα των πεδίων του πίνακα, τον τύπο τους και τυχόν περιορισμούς που θα ισχύουν για το κάθε πεδίο. Έτσι το ακόλουθο παράδειγμα δημιουργεί τον πίνακα students ο οποίος αποτελείται από δύο πεδία: το id που θα αποθηκεύει ακέραιους αριθμούς και το name τύπου varchar έως 50 χαρακτήρες το καθένα.

```
CREATE TABLE students (id INT, name varchar(50));        (5.9)
```

Αντίστοιχα με την δημιουργία ΒΔ αν υπάρχει ήδη ένας πίνακας στη ΒΔ με το όνομα το οποίο έχουμε δηλώσει τον νέο πίνακα τότε θα εμφανιστεί μήνυμα λάθους. Μπορούμε να το αποφύγουμε αυτό το πρόβλημα χρησιμοποιώντας τον όρο IF NOT EXISTS.

```
CREATE TABLE IF NOT EXISTS students (
id INT,
name VARCHAR (50)
);                                                    (5.10)
```

5.2.1 Δηλώσεις NOT NULL και DEFAULT

Προαιρετικά αν θέλουμε να απαγορευτούν σε κάποιο πεδίο οι τιμές NULL τότε εισάγουμε μετά τον τύπο του αντίστοιχου πεδίου τον όρο **NOT NULL**.

Ακόμα μπορούμε να ορίσουμε μία προκαθορισμένη τιμή **DEFAULT** σε κάποιο πεδίο η οποία θα εισαχθεί κατά την περίπτωση που δεν εισάγεται τιμή με άλλο τρόπο σε αυτό το πεδίο.

Στο παράδειγμα που ακολουθεί τα πεδίο ID και name θα απαγορεύεται να έχουν τιμές NULL ενώ το πεδίο gender θα έχει προεπιλεγμένη τιμή "Men". Παρατηρήστε τον τύπο του πεδίου gender (ENUM) και πως τον δηλώνουμε σε ένα πεδίο.

Επίσης μπορούμε να δηλώσουμε οι

```
CREATE TABLE students (
id NOT NULL,
name VARCHAR(50) NOT NULL,
gender ENUM('Men','Women') DEFAULT 'Women'
);                                                    (5.11)
```

5.2.2 Περιορισμοί πρωτεύοντος κλειδιού

Για να δηλώσουμε ένα πεδίο ως πρωτεύων κλειδί ενός πίνακα στη λίστα περιορισμών του πίνακα θα εισάγουμε τον όρο **PRIMARY KEY** (όνομα πεδίου). Αν δηλωθεί ένα πεδίο ως πρωτεύων κλειδί αυτό σημαίνει ότι οι τιμές του θα πρέπει οπωσδήποτε να είναι NOT NULL αλλά και μοναδικές.

```
CREATE TABLE students (
Id INT ,
name VARCHAR(50) NOT NULL,
PRIMARY KEY(id)
);                                                    (5.12)
```

Η δήλωση PRIMARY KEY μπορεί να εισαχθεί και αμέσως μετά τον ορισμό του πεδίου

```
CREATE TABLE students (
Id INT PRIMARY KEY,
name VARCHAR(50) NOT NULL,
);                                                    (5.13)
```

Αν έχουμε ένα σύνθετο κλειδί που αποτελείται από δύο (ή παραπάνω) πεδία μπορούμε να το δηλώσουμε ως εξής

```
CREATE TABLE students (
id INT ,
poli VARCHAR(35),
```

```
name VARCHAR(50) NOT NULL,
PRIMARY KEY(id, poli)
);                                                    (5.14)
```

5.2.3 Περιορισμοί ξένου κλειδιού

Με αντίστοιχο τρόπο με την PRIMARY KEY μπορούμε να δημιουργήσουμε ένα ξένο κλειδί σε έναν πίνακα χρησιμοποιώντας τον όρο FOREIGN KEY. Το ξένο κλειδί θα πρέπει να αναφέρεται σε καποιο πεδίο άλλου πίνακα ο οποίος θα πρέπει να έχει ήδη δημιουργηθεί. Έτσι αν έχουμε εκτελέσει την εντολή 5.13 μπορούμε να ορίσουμε ένα ξένο κλειδί στον νέο πίνακα το οποίο θα αναφέρεται στο πεδίο id του πίνακα students.

```
CREATE TABLE vathmoi (
id INT AUTO_INCREMENT,
mathima VARCHAR(80) NOT NULL,
vathmos FLOAT NOT NULL,
studentID INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(studentID) REFERENCES students(id)
);                                                    (5.15)
```

5.2.4 Περιορισμοί μοναδικότητας

Ο περιορισμός UNIQUE καθορίζει ότι η τιμή του πεδίου πρέπει να είναι μοναδική σε όλο τον πίνακα, δηλαδή δεν μπορεί δυο εγγραφές να έχουν την ίδια τιμή για αυτό το πείδίο. Αν όμως δεν έχει οριστεί για το ίδιο πεδίο και ο περιορισμός NOT NULL τότε δυο εγγραφές μπορεί να έχουν την τιμή NULL σε αυτό το πεδίο. Σε αυτή την περίπτωση δεν παραβιάζεται ο περιορισμός UNIQUE. Η χρήση του UNIQUE γίνεται με τρόπο παρόμοιο με του PRIMARY KEY όπως στα παρακάτω παραδείγματα.

```
CREATE TABLE students (
id INT NOT NULL,
name VARCHAR(50) NOT NULL,
UNIQUE(id)
);                                                    (5.16)
```

Στο παράδειγμα 5.16 επειδή το πεδίο id ορίζεται ως NOT NULL και UNIQUE ταυτόχρονα μπορεί να θεωρηθεί ως πρωτεύων κλειδί του πίνακα.

5.2.4 Δημιουργία δείκτη σε πίνακα

Για να διατηρηθεί η καλή απόδοση των ερωτημάτων που αφορούν μεγάλουν σε όγκο εγγραφών πίνακες θεωρείται απαραίτητη η τοποθέτηση δεικτών στους πίνακες. Για να ορίσουμε δείκτες για ένα πίνακα όταν τον δημιουργούμε χρησιμοποιούμε τον όρο INDEX με την ακόλουθη σύνταξη

INDEX [ονομα INDEX] (όνομα πεδίου [,όνομα πεδίου][,όνομα πεδίου…])

Ακολουθεί σχετικό παράδειγμα

```
CREATE TABLE students (
am INT NOT NULL,
firstname CHAR(50) NOT NULL,
lastname  CHAR(50) NOT NULL,
registered DATE NOT NULL,
INDEX NameIndex (registered),
PRIMARY KEY (am)
);                                                    (5.17)
```

Η παραπάνω εντολή δημιουργεί ένα δείκτη με το όνομα NameIndex όπου εφαρμόζεται στην στήλη registered..

5.2.5 Αυτόματη αύξηση τιμών ενός πεδίου

Χρησιμοποιώντας τον όρο AUTO_INCREMENT μετά τον ορισμό ενός πεδίου που έχει δηλωθεί ως πρωτεύον κλειδί ή έχει ζητηθεί η μοναδικότητα των τιμών του με τον όρο UNIQUE μπορούμε να ορίζουμε να αριθμείται αυτόματα κατά την περίπτωση που δεν εισαχθεί ρητά κάποια τιμή σε αυτό.

```
CREATE TABLE students (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(50) NOT NULL,
PRIMARY KEY(id)
);                                                    (5.18)
```

5.2.6 Δημιουργία πινάκων βάση υπάρχοντος πίνακα

Υπάρχουν δύο τρόποι δημιουργίας πίνακα βάσει άλλου πίνακα με την εντολή CREATE TABLE.

5.2.6.1 Πίνακας που περιέχει τις εγγραφές κάποιου άλλου πίνακα

Χρησιμοποιώντας την εντολή

```
CREATE TABLE  όνομα_νέου_πίνακα SELECT … ;          (5.19)
```

εισάγουμε τις εγγραφές που θα προκύψουν από το αποτέλεσμα της SELECT στον νέο πίνακα. Σε αυτή την περίπτωση ο καινούριος πίνακας δεν κρατάει το πρωτεύων κλειδί και τους υπόλοιπους περιορισμούς του πηγαίου πίνακα αλλά μόνο το περιεχόμενο των εγγραφών. Για παράδειγμα οι επόμενες εντολές δημιουργούν έναν πίνακα που περιέχει όλα τα περιεχόμενα του πίνακα City (παρουσιάζεται στην αρχή των σημειώσεων) και ένα πίνακα που περιέχει μερικά από τα περιεχόμενα του City.

```
CREATE TABLE CityCopy1
SELECT * FROM City;                                 (5.20)

CREATE TABLE CityCopy2
SELECT * FROM City WHERE Population <150000         (5.21)
```

5.2.6.2 Πίνακας που αντιγράφει την δομή κάποιου άλλου πίνακα

Χρησιμοποιώντας την εντολή

```
CREATE TABLE  όνομα_νέου_πίνακα LIKE όνομα_πηγαίου_πίνακα;
                                                    (5.22)
```

Δημιουργούμε έναν νέο κενό πίνακα χρησιμοποιώντας τον ορισμό κάποιου άλλου. Σε αυτή την περίπτωση δεν αντιγράφωνται οι εγγραφές του αρχικού πίνακα. Για παράδειγμα η επόμενη εντολή δημιουργεί έναν νέο πίνακα ο οποίος έχει ακριβώς την ίδια δομή με τον πίνακα students

```
CREATE TABLE studentsCopy LIKE students;            (5.23)
```

5.3 Προβολή πινάκων και δομής πίνακα

Για να δούμε τους πίνακες που υπάρχουν σε μία βάση δεδομένων χρησιμοποιούμε την εντολή SHOW TABLES.

SHOW TABLES FROM onoma_vasis; (5.24)

Είναι δυνατό να παραλείψουμε τον όρο FROM και το όνομα της βάσης. Τότε θα εμφανιστούν οι πίνακες της προκαθορισμένης βάσης δεδομένων.

Για να δούμε τη δομή ενός πίνακα χρησιμοποιούμε την εντολή DESRCIBE TABLE ακολουθούμενη από το όνομα του πίνακα. Στο παράδειγμα 5.25 φαίνεται το αποτέλεσμα της DESCRIBE για τον πίνακα students του παραδείγματος 5.18.

DESCRIBE TABLE students ; (5.25)

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI |         | auto_increment |
+-------+-------------+------+-----+---------+----------------+
| name  | varchar(50) | NO   |     |         |                |
+-------+-------------+------+-----+---------+----------------+
```

Στην στήλη Field αναφέρεται το όνομα της κάθε στήλης του πίνακα. Η στήλη Type προβάλει τον τύπο δεδομένων του κάθε πεδίου. Η στήλη NULL δείχνει αν επιτρέπονται οι απροσδιόριστες τιμές και η στήλη KEY αν κάποιο πεδίο αποτελεί κλειδί του πίνακα. Στην στήλη Default βλέπουμε αν υπάρχει κάποια προκαθορισμένη τιμή για κάποιο πεδίο και τέλος στην στήλη Extra εμφανίζονται αν υπάρχουν κάποιες άλλες πληροφορίες για το πεδίο του πίνακα όπως αν αριθμείται αυτόματα οπότε και εμφανίζεται το auto_increment.

5.4 Τροποποίηση πινάκων

Η SQL μας δίνει τη δυνατότητα να τροποποιήσουμε τη δομή ενός πίνακα ή να τον διαγράψουμε. Η τροποποίηση της δομής ενός πίνακα μπορεί να γίνει με την εντολή ALTER TABLE ενώ η διαγραφή με την εντολή DROP TABLE. Ας δούμε αναλυτικά τι μπορούμε να κάνουμε με τις εντολές αυτές. Για τα επόμενα παραδείγματα θα χρησιμοποιήσουμε τον πίνακα students των παραδειγμάτων 5.18,5.25.

5.4.1 Εισαγωγή πεδίου σε πίνακα

Χρησιμοποιώντας την εντολή ALTER TABLE μπορούμε να προσθέσουμε ένα πεδίο σε ένα πίνακα. Για παράδειγμα για να προσθέσουμε ένα νέο πεδίο με το επίθετο των φοιτητών η οποία θα λέγεται surname θα δώσουμε την εξής εντολή:

```
ALTER TABLE students ADD surname VARCHAR(50) NOT NULL;
```
(5.26)

Το νέο πεδίο θα προστεθεί αμέσως μετά το τελευταία πεδίο του πίνακα. Αυτή είναι η προκαθορισμένη εισαγωγή νέου πεδίου. Μπορούμε να ορίσουμε την σειρά στην οποία θα προστεθεί το νέο πεδίο με τις εξής εκφράσεις: Γράφουμε FIRST στο τέλος της εντολής όταν θέλουμε να εισαχθεί το νέο πεδίο και να είναι προηγείται των υπολοίπων πεδίων του πίνακα και AFTER όνομα_πεδίου όταν θέλουμε να τοποθετηθεί το νέο πεδίο μετά από κάποιο άλλο. Για παράδειγμα η επόμενη εντολή τοποθετεί το νέο πεδίο μετά το πεδίο id.

```
ALTER TABLE students
ADD surname VARCHAR(50) NOT NULL
AFTER id;
```
(5.27)

Η νέα δομή του πίνακα τώρα θα είναι η παρακάτω:

```
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| id      | int(11)     | NO   | PRI |         | auto_increment |
+---------+-------------+------+-----+---------+----------------+
| surname | varchar(50) | NO   |     |         |                |
+---------+-------------+------+-----+---------+----------------+
| name    | varchar(50) | NO   |     |         |                |
+---------+-------------+------+-----+---------+----------------+
```

5.4.2 Τροποποίηση πεδίου ενός πίνακα

Για να αλλάξουμε το όνομα ή τον τύπο ενός πεδίου θα εισάγουμε στην ALTER TABLE τον όρο CHANGE αμέσως μετά το όνομα του πίνακα. Τον όρο CHANGE θα πρέπει να ακολουθεί το όνομα του πεδίου το οποίο θέλουμε να τροποοιήσουμε έπειτα το νέο όνομα για το πεδίο (αν δεν θέλουμε να αλλάξουμε το όνομα του πεδίου γράφουμε δύο φορές συνεχόμενα το όνομά του) και έπειτα ο νέος τύπος του πεδίου. Για παράδειγμα η παρακάτω εντολή μεταβάλλει τον τύπο δεδομένων της surname από VARCHAR(50) σε VARCHAR(40).

```
ALTER TABLE students
CHANGE surname surname VARCHAR(40) NOT NULL;      (5.28)
```

Ενώ η εντολή 5.29 αλλάζει μόνο το όνομα του πεδίου

```
ALTER TABLE students
CHANGE surname lastname VARCHAR(50) NOT NULL;     (5.29)
```

5.4.3 Μετονομασία πίνακα

Η μετονομασία ενός πίνακα μπορεί να γίνει με την χρήση του όρου RENAME. Οι παρακάτω δύο εντολές είναι ισοδύναμες και αλλάζουν το όνομα του πίνακα students σε foitites.

```
ALTER TABLE students RENAME TO foitites          (5.30)
```

```
RENAME TABLE students TO foitites                (5.31)
```

5.4.4 Διαγραφή πεδίων και πινάκων

Για να διαγράψουμε έναν ή περισσότερους πίνακες απλά γράφουμε DROP TABLE ακολουθούμενο από το όνομα ή τα ονόματα των πινάκων που θέλουμε να διαγράψουμε χωριζόμενα με κόμμα. Για παράδειγμα η παρακάτω εντολή διαγράφει τον πίνακα students.

```
DROP TABLE students;                             (5.32)
```

Για να διαγράψουμε ένα πεδίο ενός πίνακα χρησιμοποιούμε πάλι τον όρο DROP αλλά μέσα στην εντολή ALTER TABLE όπως στο επόμενο παράδειγμα όπου αφαιρείται η στήλη surname από τον πίνακα students.

```
ALTER TABLE students DROP surname;               (5.33)
```

5.2.6 Εισαγωγή και διαγραφή περιορισμών από υπάρχον πίνακα

Για να προσθέσουμε έναν περιορισμό σε ένα πίνακα χρησιμοποιούμε τον όρο ADD σε μία εντολή ALTER TABLE ακολουθούμενο από τον νέο περιορισμό. Για παράδειγμα αν είχαμε παραλείψει να δηλώσουμε πρωτεύων κλειδί στον πίνακα students θα το δημιουργούσαμε με την παρακάτω εντολή

```
ALTER TABLE students ADD PRIMARY KEY (ID)         (5.34)
```

Αν πάλι θέλαμε να δημιουργήσουμε κάποιο σύνθετο δείκτη θα δίναμε την επόμενη εντολή

```
ALTER TABLE students ADD INDEX (surname,firstname) (5.35)
```

Φυσικά η παραπάνω εντολή προϋποθέτει ότι προυπάρχουν τα πεδία surname και firstname στον πίνακα students.

Για να διαγράψουμε έναν περιορισμό χρησιμοποιούμε τον όρο DROP ακολουθούμενο από το όνομα του περιορισμού. Για παράδειγμα η παρακάτω εντολές διαγράφουν το πρωτεύων κλειδί και τον δείκτη NameIndex από τον πίνακα students αντίστοιχα.

```
ALTER TABLE students DROP PRIMARY KEY;          (5.36)
```

```
ALTER TABLE students DROP INDEX NameIndex;      (5.37)
```

Ισοδύναμες με τις παραπάνω εντολές είναι και οι επόμενες οι οποίες χρησιμοποιούν την εντολή DROP INDEX. Να σημειωθεί ότι το όνομα του περιορισμού PRIMARY KEY είναι πάντα η δεσμευμένη λέξη PRIMARY για αυτό και αναφέρεται έτσι στο πρωτεύων κλειδί η εντολή 5.38 το οποίο μάλιστα πρέπει να το τοποθετούμε ανάμεσα σε εισαγωγικά.

```
DROP INDEX 'RIMARY' ON students;                (5.38)
```

```
DROP INDEX NameIndex ON students;               (5.39)
```

ΚΕΦΑΛΑΙΟ 6

ΠΡΟΣΘΗΚΗ, ΤΡΟΠΟΠΟΙΗΣΗ ΚΑΙ ΔΙΑΓΡΑΦΗ ΔΕΔΟΜΕΝΩΝ

6.1 Εισαγωγή δεδομένων σε πίνακα

6.1.1 Η εντολή INSERT

Για να εισάγουμε δεδομένα (δηλαδή εγγραφές) σε έναν πίνακα χρησιμοποιούμε την εντολή INSERT INTO. Η εντολή INSERT έχει δύο βασικές μορφές σύνταξης

```
INSET INTO όνομα_πίνακα (λίστα πεδίων πίνακα)
VALUES (λίστα τιμών);                                    (6.1)

INSERT INTO όνομα_πίνακα
SET πεδίο = τιμή [,πεδίο= τιμή]...
WHERE παράσταση ;                                        (6.2)
```

Για τις παραπάνω εντολές ισχύουν τα παρακάτω

- Οι τιμές στη λίστα τιμών πρέπει να είναι όσα είναι και τα πεδία στη λίστα πεδίων και αντιστοιχίζονται ένα προς ένα στα πεδία της λίστας πεδίων. Δηλαδή η πρώτη τιμή θα δοθεί στο πρώτο πεδίο, η δεύτερη στο δεύτερο κ.ο.κ.
- Η σειρά εμφάνισης στη λίστα πεδίων μπορεί να είναι οποιαδήποτε αρκεί να γίνεται σωστή αντιστοίχηση στη λίστα τιμών
- Η λίστα πεδίων πρέπει να συμπεριλαμβάνει όλα τα πεδία του πίνακα τα οποία δηλώθηκαν ως NOT NULL.
- Η λίστα πεδίων μπορεί να παραλειφθεί. Σε αυτή την περίπτωση η λίστα τιμών θα πρέπει να περιέχει τιμές για όλα τα πεδία του πίνακα με την ίδια σειρά που δηλώθηκαν κατά τη δημιουργία του πίνακα.
- Οι απροσδιόριστες τιμές (NULL) θα δηλώνονται με τον όρο NULL χωρίς εισαγωγικά
- Τέλος αν ένα πεδίο παραλειφθεί από τη λίστα πεδίων τότε σε αυτό θα δοθεί η προκαθορισμένη (DEFAULT) τιμή. Σε περίπτωση που δεν έχει οριστεί

προκαθορισμένη τιμή δίνεται η τιμή NULL εφόσον αυτό επιτρέπεται. Σε κάθε άλλη περίπτωση εμφανίζεται μήνυμα σφάλματος.

Για τα παραδείγματα που ακολουθούν στο κεφάλαιο αυτό χρησιμοποιούμε τον παρακάτω πίνακα

```
CREATE TABLE students
{
id INT NOT NULL AUTO_INCREMENT,
name CHAR(30) NOT NULL DEFAULT ' ',
age INT NOT NULL DEFAULT 0,
PRIMARY KEY(id)
};                                                    (6.3)
```

Τα επόμενα παραδείγματα εισάγουν μία εγγραφή στον πίνακα students

```
INSERT INTO students (id, name, age)
VALUES (10, 'Peter', 25);                             (6.4)

INSERT INTO students
SET id = 10, name = 'Peter', age = 25;                (6.5)
```

Είναι δυνατή και η εισαγωγή πολλών εγγραφών με μία εντολή INSERT. Αυτό μπορεί να γίνει με τον παρακάτω τρόπο

```
INSERT INTO students  (name, age)
VALUES('Peter', 25),('Nikos', 21),('Maria', 20);   (6.6)
```

Παρατηρούμε ότι μετά τον όρο VALUES μπορούν να ακολουθήσουν πολλές παρενθέσεις με τις λίστες τιμών όπου η κάθε μία αντιστοιχεί σε μία εγγραφή του πίνακα. Οι παρενθέσεις θα πρέπει να χωρίζονται από κόμμα.

Να σημειωθεί ότι στο παράδειγμα 6.6 δεν εισήγαμε τιμή για το πεδίο id. Αυτό δεν θα δημιουργήσει σφάλμα γιατί το πεδίο έχει δηλωθεί ως AUTO_INCREMENT οπότε θα αριθμείται αυτόματα.

6.1.2 Η εντολή REPLACE

Η REPLACE εισάγει και αυτή εγγραφές σε έναν πίνακα όπως και η INSERT και συντάσσεται με παρόμοιο τρόπο. Η βασική διαφορά τους έγκειται στην συμπεριφορά τους στην περίπτωση που υπάρχουν διπλές εγγραφές. Η INSERT αν προσπαθήσει να εισάγει

μία εγγραφή όπου η τιμή κάποιου πεδίου της το οποίο έχει περιορισμό μοναδικής τιμής (UNIQUE, PRIMARY KEY) συμπίπτει με την τιμή του ίδιου πεδίου κάποιας άλλης εγγραφής τότε η γραμμή δεν εισάγεται . Αντίθετα η REPLACE σε αυτή την περίπτωση πρώτα διαγράφει την παλαιά εγγραφή και έπειτα εισάγει την καινούρια. Ακολουθούν σχετικά παραδείγματα

```
REPLACE INTO students (id, name, age)
VALUES (10, 'Peter', 30);                    (6.7)

REPLACE INTO students
SET id = 10, name = 'Peter', age = 30;       (6.8)
```

6.1.3 Εισαγωγή δεδομένων από ένα άλλο πίνακα

Χρησιμοποιώντας ένα συνδυασμό της εντολής INSERT και της SELECT μπορούμε να εισάγουμε δεδομένα μέσα σε ένα πίνακα με βάση τα αποτελέσματα ενός ερωτήματος από άλλο πίνακα. Η επόμενη εντολή εισάγει τα δεδομένα του πίνακα students στον πίνακα graduated οι οποίοι έχουν αντίστοιχη δομή

```
INSERT INTO graduated SELECT * FROM students;    (6.9)
```

Στην παραπάνω εντολή οι δύο πίνακες είχαν ακριβώς την ίδια δομή. Δεν είναι αυτό απαραίτητο αρκεί να επιλέξουμε σωστά τις στήλες και τις τιμές που θα εισάγουμε.. Μπορούμε ακόμα να εισάγουμε τιμές σε συγκεκριμένες στήλες του πίνακα όπως και να περιορίσουμε τις εγγραφές που θα εισαχθούν με την χρήση συνθηκών. Ακολουθεί σχετικό παράδειγμα

```
INSERT INTO graduated (name, age)
SELECT (name,age) FROM students
WHERE age>22;                                (6.10)
```

6.2 Τροποποίηση δεδομένων

Για να τροποποιήσουμε δεδομένα από τις εγγραφές ενός πίνακα χρησιμοποιούμε την εντολή UPDATE με την ακόλουθη σύνταξη:

```
UPDATE όνομα_πίνακα
SET πεδίο = τιμή [,πεδίο = τιμή] …
WHERE παράσταση ;                            (6.11)
```

Για παράδειγμα αν θέλουμε να μεταβάλλουμε την τιμή του πεδίου name της εγγραφής που έχει id = 10 από τον πίνακα students γράφουμε

```
UPDATE students
SET age = 24
WHERE id = 10;                                          (6.12)
```

Ο όρος WHERE είναι προαιρετικός. Αν παραληφθεί τότε η νέα τιμή του πεδίου θα δοθεί σε όλες τις εγγραφές του πίνακα. Για παράδειγμα αν θέλουμε σε όλους τους φοιτητές να προσθέσουμε ένα χρόνο γράφουμε

```
UPDATE students
SET age = age + 1;                                      (6.13)
```

Η UPDATE δεν εγγειάται την σειρά με την οποία θα ενημερώνονται οι εγγραφές. Αυτό κάποιες φορές μπορεί να δημιουργήσει προβλήματα. Αν υποθέσουμε ότι έχουμε τον παρακάτω πίνακα όπου το id είναι το PRIMARY KEY.

```
Select * FROM people;                                   (6.14)
+----+-------+-----+
| id | name  | age |
+----+-------+-----+
|  2 | Peter |  19 |
|  3 | Maria |  20 |
+----+-------+-----+
```

Αν προσπαθήσουμε να αριθμήσουμε ξανά τις τιμές id ξεκινώντας από το 1 θα πρέπει να γράψουμε την εντολή

```
UPDATE people SET id=id-1;                              (6.15)
```

Όπως αναφέραμε παραπάνω η UPDATE δεν εγγυάται τη σειρά με την οποία θα κάνει τις αλλαγές. Αν λοιπόν προσπαθήσει να μεταβάλλει πρώτα την πρώτη έγγραφή και μετά την δεύτερη δε θα υπάρξει κανένα πρόβλημα. Στην περίπτωση όμως που προσπαθήσει να αλλάξει την δεύτερη εγγραφή πρώτα μεταβάλλοντας το 3 σε 2 τότε θα παραβιαζόταν ο περιορισμός της μοναδικότητας τιμής του κλειδιού. Αυτό το πρόβλημα λύνεται με την

ORDER BY όπου προσδιορίζουμε με ποια σειρά θα γίνουν οι ενημερώσεις. Η εντολή 6.15 λοιπόν θα γραφόταν ως εξής

```
UPDATE people
SET id=id-1
ORDER BY ID;                                                    (6.16)
```

Τέλος να αναφέρουμε ότι μπορούμε να προσθέσουμε και τον όρο LIMIT προκειμένου να περιοσρίσουμε τον αριθμό των εγγραφών που ενημερώνονται. Για παράδειγμα η παρακάτω εντολή αλλάζει μόνο την πρώτη εγγραφή που θα βρει με name 'Nikos' σε 'Nikolas'.

```
UPDATE people
SET name ='Nikolas'
WHERE Name ='Nikos'
LIMIT 1;                                                        (6.17)
```

6.3 Διαγραφή δεδομένων

Η διαγραφή των εγγραφών ενός πίνακα μπορεί να γίνει με τις εντολές DELETE και TRUNCATE TABLE. Η μεταξύ τους διαφορά είναι ότι η DELETE επιτρέπει έναν όρο WHERE οπότε μπορεί να προσδιορίσει ποιες εγγραφές θα διαγραφούν. Αντίθετα η TRUNCATE TABLE διαγράφει όλες τις εγγραφές του πίνακα. Οι εντολές έχουν την ακόλουθη σύνταξη:

```
DELETE FROM όνομα_πίνακα
WHERE παράσταση;                                                (6.18)

TRUNCATE TABLE όνομα_πίνακα;                                    (6.19)
```

Για παράδειγμα οι επόμενες εντολές είναι ισοδύναμες και διαγράφουν όλες τις εγγραφές του πίνακα students

```
DELETE FROM students;                                           (6.20)

TRUNCATE TABLE students                                         (6.21)
```

Όπως αναφέραμε η DELETE μπορεί να διαγράψει συγκεκριμένες εγγραφές χρησιμοποιώντας τον όρο WHERE. Για παράδειγμα η επόμενη εντολή διαγράφει όλες τις εγγραφές των οποίων η τιμή του πεδίου age είναι μικρότερη του 18.

```
DELETE FROM students
WHERE age<18;                                          (6.22)
```

Η εντολή DELETE επιτρέπει τους όρους ORDER BY και LIMIT όπως ακριβώς και η εντολή UPDATE που αναλύσαμε παραπάνω.

ΚΕΦΑΛΑΙΟ 7

ΕΡΩΤΗΜΑΤΑ ΜΕ ΕΝΩΣΗ ΠΙΝΑΚΩΝ

7.1 Εσωτερικές ενώσεις

Εσωτερική ένωση ονομάζεται μία ένωση που δίνει συνδυασμούς γραμμών που ταυτίζονται από δύο διαφορετικούς πίνακες. Στην ενότητα αυτή περιγράφεται ο τρόπος με τον οποίο μπορούμε να ανακτήσουμε πληροφορίες από περισσότερους από έναν πίνακες χρησιμοποιώντας τις εσωτερικές ενώσεις. **Ότι ισχύει για τα ερωτήματα με έναν πίνακα ισχύει και για την περίπτωση που θα αναφερόμαστε σε περισσότερους πίνακες**. Αυτό που αλλάζει είναι ότι όλοι οι πίνακες από τους οποίους θα ανακτήσουμε στοιχεία θα πρέπει να αναφέρονται μετά τον όρο FORM και να χωρίζονται με ένα κόμμα μεταξύ τους.

Το πιο απλό ερώτημα μπορεί να γραφεί βάζοντας μετά τη FORM δύο πίνακες. Για παράδειγμα η παρακάτω εντολή θα επιστρέφει όλες τις εγγραφές των δύο πινάκων students και lessons. Το αποτέλεσμα είναι γνωστό ως καρτεσιανό γινόμενο των δύο πινάκων γιατί επιστρέφει όλους τους πιθανούς συνδυασμούς των εγγραφών των δύο πινάκων.

```
SELECT * FROM Country, City;                    (7.1)
```

Στην πραγματικότητα όμως στο παραπάνω παράδειγμα κάθε χώρα έχει συγκεκριμένες πόλεις. Για να ανακτηθεί λοιπόν ένα λογικό αποτέλεσμα θα πρέπει να γίνει ταύτιση των εγγραφών μέσω μίας συνθήκης στον όρο WHERE. Στο επόμενο παράδειγμα ορίζουμε τα ονόματα των πεδίων με κοινές τιμές ανάμεσα στους δύο πίνακες. Η εντολή (7.2) θα επιστρέφει δύο στήλες όπου στη πρώτη θα αναγράφονται τα ονόματα των χωρών και στη δεύτερη τα ονόματα των πόλεων που ανήκουν σε κάθε χώρα.

```
SELECT Country.Name, City.Name
FROM Country, City
WHERE Code = CountryCode;                       (7.2)
```

Το πρώτυπο SQL2 υποστηρίζει και μία νέα μορφή ένωσης πινάκων με τη χρήση των όρων INNER JOIN … ON. Το παράδειγμα 7.3 είναι ισοδύναμο με το 7.2.

```
SELECT Country.Name, City.Name
FROM Country INNER JOIN City
ON Code = CountryCode;                                    (7.3)
```

Σε αυτή την περίπτωση δεν υπάρχει διαφορά σε σχέση με τη σειρά δήλωσης των πινάκων μετά τους όρους FROM και JOIN.

Παρατηρούμε ότι μετά το ON γράφουμε μία έκφραση που δηλώνει τη σχέση μεταξύ των πινάκων.Αν το πεδίο που θα συσχέτιζε τους δύο πίνακες είχε ίδιο όνομα και σοτυς δύο π.χ. Code τότε θα μπορούσαμε αντί του όρου ON να χρησιμοποιήσουμε τον όρο USING(πεδιο) όπως στο παράδειγμα που ακολουθεί

```
SELECT Country.Name, City.Name
FROM Country INNER JOIN City
USING (Code);                                             (7.4)
```

Συνώνυμα του INNER JOIN είναι και τα **JOIN** και **CROSS JOIN.**

Παρατήρηση: Αν δύο πίνακες έχουν στήλες με το ίδιο όνομα τότε αυτές τις στήλες πρέπει να τις καλούμε με πρόθεμα τον πίνακα που βρίσκονται.

7.1.1 Παραδείγματα εσωτερικών ενώσεων

Παρακάτω ακολουθούν ορισμένα παραδείγματα ένωσης πινάκων:

1. Αλφαβητική προβολή των πόλεων των χωρών της Αφρικής

```
SELECT City.Name
FROM Country, City
WHERE Code = CountryCode
AND Continent=”Africa”
ORDER BY City.Name;                                       (7.5)
+----------+
| Name     |
+----------+
| Aba      |
| Abeokuta |
| Abidjan  |
| Abuja    |
| Accra    |
| Alger    |
```

.....

2. Προβολή των πόλεων της Ελλάδας και του πληθυσμού τους ταξινομημένες κατά φθίνουσα σειρά πληθυσμού

```
SELECT City.Name, City.Population
FROM Country, City
WHERE Code = CountryCode
AND Country.Name=”Greece”
ORDER BY City.Population Desc;                          (7.6)
```

```
+--------------+------------+
| Name         | Population |
+--------------+------------+
| Athenai      |     772072 |
| Thessaloniki |     383967 |
| Pireus       |     182671 |
| Patras       |     153344 |
| Peristerion  |     137288 |
| Herakleion   |     116178 |
| Kallithea    |     114233 |
| Larisa       |     113090 |
+--------------+------------+
```

3. Προβολή του πληθυσμού της Ελλάδας που ζει σε πόλεις .

```
SELECT SUM(City.Population) AS Plythismos
FROM Country INNER JOIN City ON Code = CountryCode
WHERE Country.Name=”Greece”;                            (7.7)
```

```
+------------+
| Plythismos |
+------------+
|    1972843 |
+------------+
```

4. Προβολή των χωρών που έχουν καταχωρημένες στη βάση πάνω από 15 πόλεις

```
SELECT Country.Name, COUNT(*)
FROM Country, City
WHERE CountryCode=Code
Group BY Country.Name
HAVING COUNT(*)>15;                                     (7.8)
```

```
+--------------------------------------+----------+
| Name                                 | COUNT(*) |
+--------------------------------------+----------+
| Algeria                              |       18 |
| Argentina                            |       57 |
| Bangladesh                           |       24 |
| Belarus                              |       16 |
```

```
| Brazil                                  |      250 |
.....................................................................
```

7.2 Εξωτερικές ενώσεις

Οι εξωτερικές ενώσεις επιστρέφουν τις ίδιες εγγραφές με τις εσωτερικές και επιπλέον εκείνες τις εγγραφές που δεν έχουν ταυτόσιμες τιμές στο πεδίο με το οποίο γίνεται η ένωση είτε από τον ένα πίνακα (LEFT JOIN, RIGHT JOIN) είται και από τους δύο (FULL OUTER JOIN).

Έστω ότι οι πίνακες `People` και City έχουν τις παρακάτω εγγραφές:

```
    People                  Cities
+------+------+        +------------+--------+
| Code | Name |        | PeopleCode | City   |
+------+------+        +------------+--------+
|    1 | Anna |        |          2 | Kavala |
|    2 | John |        |          3 | Drama  |
+------+------+        +------------+--------+
```

Μία εσωτερική ένωση μπορεί να συσχετίσει τους ανθρώπους από τον πίνακα People με τις πόλεις που κατοικούν από τον πίνακα Cities μέσω μίας ένωσης που βασίζεται στα πεδία Code και.PeopleCode. Δεν μπορεί όμως να εμφανίσει τα ονόματα των ανθρώπων που δεν σχετίζονται με καμία πόλη από τον πίνακα Cities δηλαδή αυτούς για τους οποίους δεν υπάρχουν καταχωρημένες εγγραφές στον πίνακα Cities.

Έτσι ενώ μία εσωτερική ένωση των δύο πινάκων θα έδινε το αποτέλεσμα

```
SELECT Code, Name, City
FROM People INNER JOIN Cities
ON Code= PeopleCode;                                    (7.9)
+------+------+--------+
| Code | Name | City   |
+------+------+--------+
|    2 | John | Kavala |
+------+------+--------+
```

Μία εξωτερική ένωση θα μας έδειχνε ότι και η εσωτερική και επιπλέον τις ανατιστοιχίες του ενός πίνακα σε σχέση με τον άλλο. Έτσι στο παράδειγμα 7.10 έχουμε μία εξωτερική ένωση με LEFT JOIN η οποία μας δείχνει πλέον των ταυτόσιμων εγγραφών, όλες τις εγγραφές του αριστερού πίνακα (People) που δεν έχουν σχετιζόμενες εγγραφές στον πίνακα Cities

```
SELECT Code, Name, City
FROM People LEFT JOIN Cities
ON Code= PeopleCode;                                    (7.10)
+------+------+--------+
| Code | Name | City   |
+------+------+--------+
|    1 | Anna | NULL   |
|    2 | John | Kavala |
+------+------+--------+
```

Αν διατυπώναμε το ίδιο ερώτημα αλλά με RIGHT JOIN τότε σε αυτή την περίπτωση θα εμφανιζώταν όλες ταυτόσιμες εγγραφές των δύο πινάκων και επιπλέον οι εγγραφές του δεξιού πίνακα (Cities) που δεν σχετίζονται με καμία εγγραφή του αριστερού πίνακα (People).

```
SELECT Code, Name, City
FROM People RIGHT JOIN Cities
ON Code= PeopleCode;                                    (7.11)
+------+------+--------+
| Code | Name | City   |
+------+------+--------+
|    2 | John | Kavala |
|    3 | NULL | Drama  |
+------+------+--------+
```

Μπορούμε να μετατρέψουμε μία RIGHT JOIN σε LEFT JOIN ή το αντίστροφο αλλάζοντας τη σειρά δήλωσης των δύο πινάκων. Για παράδειγμα αν θέλουμε να μετατρέψουμε την εντολή 7.11 σε LEFT JOIN γράφουμε

```
SELECT Code, Name, City
FROM Cities LEFT JOIN People
ON Code= PeopleCode;                                    (7.12)
```

7.3 Ενημέρωση δεδομένων σε πολλούς πίνακες

Οι εντολές UPDATE και DELETE οι οποίες είδαμε σε προηγούμενα κεφάλαια είνα δυνατό να επεξεργαστούν δεδομένα σε περισσότερους από έναν πίνακες.

7.3.1 Η εντολή UPDATE για πολλούς πίνακες

Όσον αφορά την UPDATE θα πρέπει αμέσως μετά την αναγραφή του όρου να δίνουμε τους πίνακες που εμπλέκονται στην ενημέρωση οι οποίοι όπως και στην εντολή SELECT θα χωρίζονται από κόμμα. Στον όρο WHERE θα πρέπει να αναγράφουμε την συνθήκη

συσχέτισης των δύο πινάκων και τέλος στον όρο SET αναγράφουμε οποιεσδήποτε στήλες των δύο πινάκων στις οποίες θέλουμε να εφαρμόσουμε εκχώριση τιμών.

Για παράδειγμα η επόμενη εντολή αντιγράφει τη τιμή του πεδίου name από τον pinaka2 στον pinaka1 στις εγγραφές που ταυτίζονται:

```
UPDATE pinaka1, pinaka2
SET pinaka1.name=pinaka2.name
WHERE pinaka1.id = pinaka2.id;                    (7.13)
```

Έστω ότι έχουμε τους παρακάτω δύο πίνακες

```
        Pelates                          Daneia
+----+------+------------+           +-----------+--------+
| id | name | occupation |           | pelatesID | xreos  |
+----+------+------------+           +-----------+--------+
|  1 | Anna | Student    |           |         1 |    850 |
|  2 | John | Teacher    |           |         2 |    260 |
.................................................  ..........................................
```

Η επόμενη εντολή μειόνει το χρέος των φοιτητών κατά 5%. Για να γίνει αυτό ελένχεται με συνθήκη στην WHERE αν το πεδίο occupation έχει τιμή student και εφόσων η συνθήκη είναι αληθής ενημερώνεται το χρέος στον πίνακα Daneia..

```
UPDATE Pelates, Daneia
SET xreos=xreos*0.95
WHERE id = pelatesID
AND occupation = "Student";                       (7.14)
```

7.3.2 Η εντολή DELETE για πολλούς πίνακες

Αντίστοιχα με την εντολή DELETE μπορούμε να διαγράφουμε εγγραφές από περισσότερους πίνακες ή να διαγράφουμε εγγραφές ενός πίνακα βασιζόμενοι σε συνθήκη με πεδία άλλου πίνακα. Όσον αφορά την σύνταξη της ενολής θα πρέπει μετά τον όρο FROM να αναφέουμε όλους τους πίνακες που εμπλέκονται, τους οποίους χωρίζουμε με κόμμα ενώ μετά τον όρο DELETE αναγράφουμε όλους τους πίνακες από τους οποίους θέλουμε να διαγράψουμε εγγραφές. Επίσης όπως και με την UPDATE θα πρέπει να υπάρχει και ένας όρος WHERE όπου θα αναγράφεται η συνθήκη συσχέτισης των δύο πινάκων.

Για παράδειγμα η ακόλουθη εντολή διαγράφει τις εγγραφές των πινάκων που παρουσιάστηκαν στην προηγούμενη παράγραφο όπου το χρέος είναι μηδενικό.

```
DELETE Pelates, Daneia
FROM Pelates, Daneia
WHERE id = pelatesID
AND xreos = 0;                                    (7.15)
```

Ενώ η εντολή 7.16 διαγράφει τα χρέη των φοιτητών από τον πίνακα Daneia όχι όμως και τις εγγραφές με τα ονόματα και το επάγγελμά τους.

```
DELETE Daneia
FROM Pelates, Daneia
WHERE id = pelatesID
AND occupation = "Student";                       (7.16)
```

ΚΕΦΑΛΑΙΟ 8

ΥΠΟΕΡΩΤΗΜΑΤΑ

Ένα υποερώτημα είναι μία εντολή SELECT που τοποθετείται εντός παρενθέσεων μέσα σε άλλη εντολή SQL και σκοπό έχει να μας διευκολύνει στην εύρεση κάποιων δεδομένων. Πολλές φορές είναι δυνατό κάποια ερωτήματα να διατυπωθούν διαφορετικά χωρίς υποερώτημα συνήθως με κάποια ένωση πινάκων. Ο τρόπος με τον οποίο θα επιλέξουμε να διατυπώσουμε ένα ερώτημα από ένα σημείο και μετά είναι θέμα δικής μας επιλογής.

8.1 Απλά Υποερωτήματα

Ας δούμε μερικά παραδείγματα ερωτημάτων προκειμένου να γίνει κατανοητή η χρήση και η χρησιμότητα τους. Για τα παραδείγματα θα χρησιμοποιήσουμε τους πίνακες της ΒΔ world που παρουσιάζονται στην αρχή των σημειώσεων.

Έστω ότι θέλουμε να εμφανίσουμε αλφαβητικά τις πόλεις τις Ελλάδας και τον πληθυσμό τους. Απάντηση σε αυτή την ερώτηση δίνει η εντολή 7.6 που παρουσιάστηκε στο προηγούμενο κεφάλαιο η οποία καταφεύγει στην ένωση πινάκων για να επιτύχει το σωστό αποτέλεσμα. Ισοδύναμα μπορούμε να γράψουμε την εντολή 8.1 όπου γίνεται χρήση υποερωτήματος

```
SELECT Name, Population
FROM City
WHERE CountryCode = (SELECT Code
                     FROM Country
                     WHERE Name="Greece")
ORDER BY Population DESC;                    (8.1)
```

Παρατηρούμε ότι δεν χρειάστηκε να βάλουμε πρόθεμα το όνομα του πίνακα πριν από το Name και το Population γιατί η κάθε εντολή SELECT λειτουργέι αυτόνομα οπότε η εξωτερική εντολή θα κοιτάξει για αυτά τα πεδία στον πίνακα City ενώ το υποερώτημα θα κοιτάξει στον πίνακα Country.

Η προηγούμενη εντολή μπορούσε όπως είδαμε να διατυπωθεί και με άλλο τρόπο χωρίς την χρήση υποερωτήματος. Υπάρχουν όμως και περιπτώσεις όπου η χρήση υποερωτήματος κρίνεται απαραίτητη. Έστω ότι θέλουμε να βρούμε σε ποια χώρα βρίσκεται η πολυπληθέστερη πόλη στον κόσμο. Σε αυτή την περίπτωση πρέπει με ένα ερώτημα να βρούμε πρώτα τον πληθυσμό της πολυπληθέστερης πόλης με την χρήση ενός υποερωτήματος και έπειτα να ζητήσουμε να εκτυπωθεί η χώρα η οποία σχετίζεται με την συγκεκριμένη εγγραφή.

```
SELECT Country.Name
FROM City, Country
WHERE Code = CountryCode
AND City.Population = (SELECT MAX(Population)
                       FROM City);                (8.2)
```

Υποερωτήματα μπορούμε να χρησιμοποιήσουμε και στην φράση HAVING. Για παράδειγμα η επόμενη εντολή εμφανίζει τις ηπείρους οι οποίες έχουν πάνω από τους διπλάσιους κατοίκους από τις Η.Π.Α.

```
SELECT Continent
FROM Country
GROUP BY Continent
HAVING SUM(Population)>2*(SELECT Population
                          FROM Country
                          WHERE Name="USA");      (8.3)
```

Παρατήρηση: Μία εντολή ένωσης πινάκων μπορεί πάντα να εκφραστεί με υποερώτημα. Ένα υποερώτημα μπορεί συχνά αλλά όχι πάντοτε να εκφραστεί με ένωση πινάκων.

8.2 Σχετιζόμενα υποερωτήματα

Στα παραδείγματα που είδαμε έως τώρα το εξωτερικό ερώτημα με το υποερώτημα ήταν ανεξάρτητα μεταξύ τους. Σε κάποιες περιπτώσεις όμως το υποερώτημα πρέπει να περιέχει αναφορές στις τιμές του εξωτερικού ερωτήματος και δε μπορεί να χρησιμοποιηθεί ανεξάρτητα από αυτό . Τότε ονομάζεται σχετιζόμενο υποερώτημα.

Στο επόμενο ερώτημα θέλουμε να εμφανίσουμε ποια χώρα σε κάθε ήπειρο έχει τον μεγαλύτερο πληθυσμό.

```
SELECT Continent, Name, Population
```

```
FROM Country c
WHERE Population =(SELECT MAX(Population)
                   FROM Country c2
                   WHERE c.Continent = c2.Continent
                   AND Population>0);              (8.4)
```

Η τιμή της στήλης Continent του εξωτερικού ερωτήματος θα χρησιμοποιείται κάθε φορά για τον καθορισμό της ηπείρου στο εσωτερικό ερώτημα. Αυτό γίνεται όπως βλέπουμε με αναφορά του εσωτερικού ερωτήματος στο ψευδώνυμο του πίνακα Country του εξωτερικού ερωτήματος.

Θα μπορούσαμε αντί να χρησιμοποιήσουμε ψευδώνυμο στον πίνακα να χρησιμοποιούσαμε ψευδώνυμο στο πεδίο Continent. Η επόμενη εντολή είναι ισοδύναμη με την εντολή 8.4

```
SELECT Continent con, Name, Population
FROM Country
WHERE Population =(SELECT MAX(Population)
                   FROM Country
                   WHERE con = Continent
                   AND Population>0);              (8.5)
```

8.3 Υποερωτήματα με τον τελεστή IN

Όταν το υποερώτημα επιστρέφει πολλές τιμές τότε μας ενδιαφέρει συνήθως να εξετάσουμε αν το πεδίο του εξωτερικού ερωτήματος παίρνει κάποια από τις επιστρεφόμενες τιμές του υποερωτήματος . Για τις περιπτώσεις αυτές χρησιμοποιούμε τον τελεστή IN. Αυτά τα ερωτήματα έχουν την γενική μορφή

```
WHERE συνθήκη [NOT] IN (υποερώτημα)              (8.6)
```

Στο επόμενο παράδειγμα εμφανίζονται οι χώρες οι οποίες έχουν πόλεις με πληθυσμό μεγαλύτερο των 10.000.000.

```
SELECT Name
FROM Country
WHERE Code IN (SELECT DISTINCT CountryCode
               FROM City
               WHERE Population>10000000);        (8.7)
```

8.4 Υποερωτήματα με τους τελεστές ANY και ALL

Οι συγκριτικοί τελεστές (<,>, <=, >=, <> κλπ) που εισάγουν μία ερώτηση μπορούν να τροποποιηθούν με τις λέξεις κλειδιά ANY και ALL. Συγκεκριμένα το >ALL σημαίνει μεγαλύτερο από όλες τις τιμές ενώ το >ANY σημαίνει μεγαλύτερο από κάποια τιμή.

Ας δούμε πρώτα την παρακάτω εντολή

```
SELECT Continent, AVG(Population)
FROM Country
GROUP BY Continent;                                    (8.8)
+---------------+-----------------+
| Continent     | AVG(Population) |
+---------------+-----------------+
| Asia          |   72647562.7451 |
| Europe        |   15871186.9565 |
| North America |   13053864.8649 |
| Africa        |   13525431.0345 |
| Oceania       |    1085755.3571 |
| Antarctica    |          0.0000 |
| South America |   24698571.4286 |
+---------------+-----------------+
```

Στο επόμενο παράδειγμα βρίσκουμε τις χώρες οι οποίες έχουν πληθυσμό μεγαλύτερο από τον μέσο πληθυσμό χώρας κάθε ηπείρου. Δηλαδή μεγαλύτερο από το μέσο πληθυσμό χώρας της Ασίας που είναι ο μεγαλύτερος μέσος πληθυσμός όλων των ηπείρων

```
SELECT Name, Population
FROM Country
WHERE Population >ALL (SELECT AVG(Population)
                       FROM Country
                       GROUP BY Continent);
ORDER BY Name;                                         (8.9)
```

Παρατήρηση: Το πεδίο Continent έχει αφαιρεθεί από το υποερώτημα για τον λόγο ότι ένα υποερώτημα ποσότητας πρέπει να παράγει μόνο μία στήλη τιμών με της οποίας τις τιμές θα γίνεται και η σύγκριση. Ειδάλλως δεν είναι δυνατό να καταλάβει η MySQL ποια στήλη να χρησιμοποιήσει για την σύγκριση.

Οι συγκρίσεις που χρησιμοποιούν την εντολή ANY θα είναι αληθής (TRUE) αν η συνθήκη σύγκρισης επιστρέφει TRUE τουλάχιστον με μία τιμή από τις εγγραφές που επιστρέφει το υποερώτημα. Αν δηλαδή στο προηγούμενο παράδειγμα είχαμε χρησιμοποιήσει τον όρο

ANY θα εμφάνιζε όλες τις χώρες οι οποίες έχουν πληθυσμό μεγαλύτερο από έστω και ένα μέσο πληθυσμό ηπείρου.

Το επόμενο παράδειγμα βρίσκει τις χώρες της Αφρικής και για κάθε μία ελέγχει αν βρίσκεται μεταξύ των χωρών όπου ομίλιουνται τα Αγγλικά. Εφόσον συμβαίνει αυτό η συνθήκη είναι αληθής και εκτυπώνεται το όνομα της χώρας.

```
SELECT Name
FROM Country
WHERE Continent = 'Africa'
AND Code = ANY (SELECT CountryCode
                FROM CountryLanguage
                WHERE Language = 'English');
ORDER BY Name;                                          (8.10)
+--------------+
| Name         |
+--------------+
| Lesotho      |
| Saint Helena |
| Seychelles   |
| South Africa |
| Zimbabwe     |
+--------------+
```

Παρατήρηση: το =ANY είναι ισοδύναμο με το IN. Το <>ANY και το !=ANY όμως είναι διαφορετικά από το NOT IN. Το NOT IN σημαίνει ότι η τιμή που συγκρίνουμε δεν είναι ίση με καμία από τις τιμές της λίστας ενώ το !=ANY σημαίνει η τιμή που ψάχνουμε πρέπει να μην είναι ίση τουλάχιστον με μία από τις τιμές της λίστας.

8.5 Υποερωτήματα με τον τελεστή EXISTS

Ο όρος EXISTS ελέγχει αν το υποερώτημα επέστρεψε ή όχι κάποιες εγγραφές. Αν επέστρεψε έστω και μία εγγραφή η συνθήκη είναι αληθής ενώ σε αντίθετη περίπτωση ψευδής. Για παράδειγμα η επόμενη εντολή εμφανίζει τις ηπείρους όπου έστω και μία χώρα τους ομιλάει την Ισπανική

```
SELECT Distinct Continent
FROM Country c
WHERE EXISTS (SELECT *
              FROM CountryLanguage
              WHERE CountryCode= c.Code
              AND Language = 'Spanish');
```

```
ORDER BY Name;                                                    (8.11)
+---------------+
| Continent     |
+---------------+
| Europe        |
| North America |
| South America |
+---------------+
```

ΚΕΦΑΛΑΙΟ 9

ΟΨΕΙΣ

Οι όψεις είναι ένας εναλλακτικός τρόπος για να βλέπουμε τα δεδομένα ενός ή περισσοτέρων πινάκων. Προέρχονται από έναν ή περισσότερους πίνακες αλλά και από άλλες όψεις. Οι πίνακες από τους οποίους προέρχονται τα δεδομένα μίας όψης ονομάζονται πίνακες βάσης ή υποκείμενοι πίνακες. Ουσιαστικά οι όψεις είναι ερωτήματα που αποθηκεύονται στη βάση δεδομένων. Δεν δημιουργούνται ξεχωριστά αντίγραφα δεδομένων αλλά τα δεδομένα τα οποία βλέπουμε είναι αποθηκευμένα στους υποκείμενους πίνακες. Όταν τροποποιούμε τα δεδομένα σε μία όψη ουσιαστικά αλλάζουμε τα δεδομένα στους υποκείμενους πίνακες. Αλλά και αν αλλάξουμε τα δεδομένα στους υποκείμενους πίνακες αυτόματα οι αλλαγές εμφανίζονται και στις όψεις που προέρχονται από αυτούς.

9.1 Δημιουργία όψεων

Για να δημιουργήσουμε μία όψη χρησιμοποιούμε την εντολή CREATE VIEW με την ακόλουθη γενική σύνταξη.

```
CREATE VIEW όνομα_όψης [λίστα στήλών]
AS SELECT ..... ;        (9.1)
```

Για παράδειγμα η επόμενη εντολή δημιουργεί μία όψη με το όνομα EuroPop η οποία εμφανίζει τις χώρες της Ευρώπης και τον πληθυσμό τους.

```
CREATE VIEW EuroPop AS
SELECT Name, Population
FROM Country
WHERE Continent = 'Europe';                    (9.2)
```

Για να ανακτήσουμε τα δεδομένα της όψης γράφουμε

```
SELECT * FROM EuroPop                          (9.3)
```

Τα ονόματα των πεδίων στις όψεις πρέπει να είναι μοναδικά. Αν προσπαθήσουμε να εμφανήσουμε τις χώρες με τις πόλεις τους με την επόμενη εντολή θα δημιουργηθεί σφάλμα γιατί δύο στήλες της όψης θα έχουν το όνομα Name..

```
CREATE VIEW CountryCities AS
SELECT Country.Name, City.Name
FROM Country, City
WHERE Code=CountryCode;
```

Λάθος (9.4)

Σε αυτή την περίπτωση θα πρέπει να μετονομάσουμε τουλάχιστον μία στήλη. Η επόμενη εντολή δίνει νέα ονόματα και στις δύο στήλες της όψης.

```
CREATE VIEW CountryCities (Xora, Poli) AS
SELECT Country.Name, City.Name
FROM Country, City
WHERE Code=CountryCode;
```

Σωστ (9.5)

Μπορούμε να έχουμε και όψεις με ομαδοποίηση πινάκων. Ακολουθεί σχετικό παράδειγμα όπου εμφανίζει τον μέγιστο πληθυσμό χώρας κάθε ηπείρου.

```
CREATE VIEW ContinentMaxPop (Continent, MaxPopulation) AS
SELECT Continent, MAX(Population)
FROM Country
GROUP BY Continent;
```

(9.6)

Παρατήρηση: Θα πρέπει κάθε φορά που έχουμε μία συγκεντρωτική συνάρτηση στα πεδία μίας όψης να ονομάζουμε ρητά τα ονόματα των πεδίων της όψης.

9.2 Ενημέρωση όψεων

Οι όψεις όπως έχουμε αναφέρει είναι εικονικοί πίνακες. Δεν έχουν αποθηκευμένα δεδομένα αλλά προβάλλουν τα δεδομένα των υποκείμενων πινάκων. Ωστόσο κάτω από ορισμένες συνθήκες είναι δυνατή η ενημέρωση των δεδομένων τους. Η βασική συνθήκη για να είναι δυνατή η ενημέρωση των δεδομένων μίας όψης είναι ότι θα πρέπει οι γραμμές της όψης να έχουν μία μονοσήμαντη σχέση με τις γραμμές του υποκείμενου πίνακα. Πιο πρακτικά θα πρέπει μεταξύ άλλων να ισχύουν τα εξής

- Η όψη να έχει προέλθει μόνο από ένα πίνακα
- Στο ερώτημα δεν θα πρέπει να υπάρχουν οι όροι DISTINCT, GROUP BY, HAVING
- Δεν επιτρέπεται η SELECT να περιέχει υπολογιζόμενα πεδία ή συναρτήσεις

Θα δημιουργήσουμε μία όψη με τα ονόματα και τον αρχηγό του κράτους.

```
CREATE VIEW CountryHead (Xora, Arxigos) AS
SELECT Country.Name, HeadOfState
FROM Country;                                    (9.7)
```

Έπειτα θα εισάγουμε μία νέα εγγραφή:

```
INSERT INTO CountryHead (Xora, Arxigos)
VALUES ('Skopia', 'Gligorof');                   (9.8)
```

Και στη συνέχεια θα εισάγουμε το όνομα του νέου προέδρου των Σκοπίων

```
UPDATE CountryHead
SET Arxigos = 'Tservenkofski'
WHERE Xora='Skopia'                              (9.9)
```

Τέλος θα διαγράψουμε την εγγραφή που αναφέρεται στα Σκόπια

```
DELETE FROM CountryHead
WHERE Xora=''Skopia';                            (9.10)
```

9.3 Τροποποίηση όψεων

Μπορούμε να αλλάξουμε τον ορισμό μίας όψης με χρήση της εντολής ALTER VIEW. Η ALTER VIEW συντάσεται όπως ακριβώς και η CREATE VIEW και η μόνη διαφορά είναι ότι αναφέτεται σε υπάρχουσα όψη. Η ALTER VIEW αντικαθιστά τον παλαιό ορισμό της όψης με τον νέο που αναφέρεται μετά τον όρο AS. Για παράδειγμα μπορούμε να αλλάξουμε την όψη του παραδείγματος 9.7 ώστε να μας δείχνει μόνο τις χώρες της Ευρώπης.

```
ALTER VIEW CountryHead (Xora, Arxigos) AS
SELECT Country.Name, HeadOfState
FROM Country
WHERE Continent = 'Europe';                      (9.11)
```

9.4 Διαγραφή όψεων

Η διαγραφή των όψεων που έχουμε δημιουργήσει γίνεται με την εντολή DROP VIEW κατά τα πρότυπα της DROP TABLE που έχουμε δει σε προηγούμενο κεφάλαιο. Η σύνταξή της είναι η ακόλουθη.

DROP VIEW [IF EXISTS] onoma_opsis [, onoma_opsis] …; (9.12)

Για να διαγράψουμε τις όψεις των παραδειγμάτων 9.6 και 9.7 γράφουμε

```
DROP VIEW IF EXISTS ContinentMaxPop, CountryHead; (9.13)
```