Κεφάλαιο 6

SQL

Στο κεφάλαιο αυτό παρουσιάζεται η δομημένη γλώσσα ερωτοαποκρίσεων (Structured Query Language, SQL) που χρησιμοποιείται για τη διαχείριση των δεδομένων της βάσης. Η διαχείριση αυτή περιλαμβάνει τόσο τη δημιουργία και μεταβολή των πινάκων της εφαρμογής, όσο και την καταχώρηση και ανάκτηση δεδομένων με βάση συγκεκριμένα κριτήρια επιλογής.

Όπως έχει ήδη αναφερθεί, μια από τις βασικές λειτουργίες ενός σχεσιακού συστήματος διαχείρισης βάσεων δεδομένων, είναι **η διαχείριση του πληροφοριακού περιεχομένου που είναι αποθηκευμένο στη βάση**, και οργανωμένο σε πίνακες οι οποίοι συσχετίζονται μεταξύ τους. Οι πιο σημαντικές μορφές αυτής της διαχείρισης είναι **η εισαγωγή, μεταβολή και διαγραφή των εγγραφών των πινάκων της βάσης** – διεργασίες οι οποίες οδηγούν στη μεταβολή του πληροφοριακού περιεχομένου – αλλά και **η επιλογή και εμφάνιση των εγγραφών των πινάκων με βάση κάποια κριτήρια αναζήτησης**. Πριν βέβαια πραγματοποιηθούν όλες οι παραπάνω ενέργειες, θα πρέπει να δημιουργήσουμε τους πίνακες της βάσης και να ορίσουμε τις συσχετίσεις που υφίστανται ανάμεσά στα πεδία τους, με βάση το λογικό σχεδιασμό του συστήματος που έχουμε περιγράψει σε προηγούμενο κεφάλαιο.

Αυτές δύο βασικές ομάδες λειτουργιών – **η δημιουργία του σχεσιακού σχήματος και η διαχείριση του πληροφοριακού περιεχομένου που περιέχεται σε αυτό** – πραγματοποιούνται από κατάλληλα εργαλεία που είναι ειδικά σχεδιασμένα για αυτό το σκοπό. Πιο συγκεκριμένα, η διαχείριση των πινάκων της βάσης, πραγματοποιείται χρησιμοποιώντας τη **γλώσσα ορισμού δεδομένων (Data Definition Language, DDL)**, ενώ για τη διαχείριση των δεδομένων της εφαρμογής, χρησιμοποιείται η **γλώσσα χειρισμού δεδομένων (Data Manipulation Language, DML)**. Αυτές οι δύο γλώσσες συναντώνται σε όλα τα μοντέρνα συστήματα διαχείρισης βάσεων δεδομένων και αποτελούν τμήμα μιας πιο γενικευμένης γλώσσας η οποία ονομάζεται **δομημένη γλώσσα ερωτοαποκρίσεων (Structured Query Language, SQL)** και έχει καθιερωθεί ως το διεθνές πρότυπο διαχείρισης των δεδομένων μιας εφαρμογής.

Υπάρχουν πολλές παραλλαγές της γλώσσας **SQL** που κυκλοφορούν στην αγορά, οι οποίες ωστόσο χαρακτηρίζονται από την ίδια δομή και την ίδια φιλοσοφία.. Έτσι, μια τυπική γλώσσα **SQL**, θα περιλαμβάνει τις επόμενες δομικές μονάδες:

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

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

Ορισμός όψεων της βάσης (View Definition): Επιτρέπει τη δημιουργία όψεων της βάσης δεδομένων οι οποίες όπως θα δούμε στη συνέχεια, ορίζονται ως **εικονικοί πίνακες (virtual tables)** οι οποίοι περιέχουν δεδομένα από έναν ή περισσότερους πίνακες της βάσης.

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

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

Η αναλυτική περιγραφή της γλώσσας **SQL** αποτελεί αντικείμενο των σελίδων που ακολουθούν.

Η ΓΛΩΣΣΑ ΟΡΙΣΜΟΥ ΔΕΔΟΜΕΝΩΝ

Η **γλώσσα ορισμού δεδομένων (Data Definition Language, DDL)**, επιτρέπει τη διαχείριση **πινάκων (tables), όψεων (views) και δεικτών (indices)** σε μια βάση δεδομένων. Αυτή η διαχείριση περιλαμβάνει **τον ορισμό και τη μεταβολή της δομής αυτών των αντικειμένων, καθώς επίσης και τη διαγραφή τους**. Από τα τρία αυτά αντικείμενα, **οι πίνακες είναι το δομικό χαρακτηριστικό μιας σχεσιακής βάσης δεδομένων, καθώς περιέχουν τα δεδομένα που καταχωρούνται σε αυτή**, ενώ **οι όψεις**, όπως έχουμε δει, **προκύπτουν από έναν ή περισσότερους πίνακες και συσχετίζουν δεδομένα που είναι αποθηκευμένα σε αυτούς**. Τέλος **οι δείκτες είναι ειδικές δομές δεδομένων, οι οποίες επιταχύνουν τη διαδικασία της αναζήτησης πληροφοριών από τη βάση**. Αναλυτική περιγραφή των όψεων και των δεικτών, θα δοθεί στη συνέχεια. Στις επόμενες σελίδες παρουσιάζονται οι εντολές της γλώσσας ορισμού δεδομένων με τις οποίες μπορούμε να διαχειριστούμε τους πίνακες, τις όψεις και τους δείκτες μιας σχεσιακής βάσης δεδομένων.

(α) Διαχείριση πινάκων

Η διαχείριση των πινάκων από τις εντολές της γλώσσας ορισμού δεδομένων, περιλαμβάνει **τη δημιουργία και διαγραφή πινάκων από τη βάση δεδομένων καθώς και την προσθήκη πεδίων στους πίνακες, μετά τη δημιουργία τους**. Οι εντο-

λές που πραγματοποιούν αυτές τις διαδικασίες, είναι οι **CREATE TABLE, DROP TABLE** και **ALTER TABLE**.

Η εντολή **CREATE TABLE** χρησιμοποιείται για τη δημιουργία ενός καινούριου πίνακα στη βάση δεδομένων. Η εντολή δέχεται ως παράμετρο **το όνομα του πίνακα που θέλουμε να δημιουργήσουμε και τη δομή αυτού του πίνακα, που περιλαμβάνει το σύνολο των πεδίων που θα περιέχονται σε αυτόν**. Για κάθε πεδίο καθορίζουμε **το όνομά του και τον τύπο του** ενώ προαιρετικά μπορούμε να θέσουμε και κάποιους **περιορισμούς** όσον αφορά το σύνολο και τα χαρακτηριστικά των τιμών του.

Οι πιο συνηθισμένοι τύποι δεδομένων που μπορούμε να χρησιμοποιήσουμε για τα πεδία των πινάκων, είναι **αριθμητικοί τύποι και συμβολοσειρέ**ς. Ένα πεδίο στο οποίο θα καταχωρηθεί κάποιος αριθμός, μπορεί να είναι **μικρός** (**SMALLINT** με μέγιστη τιμή **32767**) ή **μεγάλος (INTEGER) ακέραιος** ή **πραγματικός αριθμός (FLOAT)**, ενώ εάν θέλουμε να καταχωρήσουμε επ' ακριβώς τη μορφή του πραγματικού αριθμού, μπορούμε να χρησιμοποιήσουμε τον τύπο **DECIMAL (i, j)** όπου **i** είναι το συνολικό πλήθος των ψηφίων του αριθμού, και **j** το πλήθος των ψηφίων μετά την υποδιαστολή. Εάν σε ένα πεδίο πρόκειται να καταχωρηθεί **συμβολοσειρά**, μπορούμε να χρησιμοποιήσουμε τον τύπο **CHAR (n)** ή **VARCHAR (n)** όπου **n** είναι το πλήθος των χαρακτήρων που θα χρησιμοποιηθούνε. Η διαφορά ανάμεσα σε αυτούς τους δύο τύπους, είναι πως στον τύπο **CHAR** χρησιμοποιούνται ακριβώς **n** χαρακτήρες, ενώ στον τύπο **VARCHAR** το πλήθος των χαρακτήρων είναι μεταβλητό, αλλά σε κάθε περίπτωση, όχι μεγαλύτερο από n χαρακτήρες. Στα μοντέρνα συστήματα διαχείρισης βάσεων δεδομένων, υπάρχει και ένας τύπος δεδομένων **DATE** τον οποίο χρησιμοποιούμε όταν θέλουμε να καταχωρήσουμε ημερομηνίες.

Τέλος, οι περιορισμοί τους οποίους μπορούμε να θέσουμε για κάθε πεδίο, αφορούν κυρίως **το δικαίωμα καταχώρησης ή όχι της τιμής NULL**, σε αυτό το πεδίο. Έτσι, για όσα πεδία δεν επιτρέπεται η απόδοσης τιμής **NULL** σε αυτά, χρησιμοποιούμε κατά την κλήση της εντολής, τον κανόνα «**NOT NULL**».

Χρησιμοποιώντας την εντολή **CREATE TABLE**, μπορούμε να κατασκευάσουμε όλους τους πίνακες της βάσης δεδομένων της εταιρείας, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
CREATE TABLE EMPLOYEE  (

        FNAME           VARCHAR (15)    NOT NULL,
        MINIT           CHAR (1),
        LNAME           VARCHAR (15)    NOT NULL,
        SSN             CHAR (9)        NOT NULL,
        BDATE           DATE,
        ADDRESS         VARCHAR (30),
        SEX             VARCHAR (30),
        SALARY          INTEGER,
        SUPERSSN        CHAR (9),
        DNO             INTEGER      );

CREATE TABLE DEPARTMENT  (

        DNAME           VARCHAR (15)    NOT NULL,
        DNUMBER         INTEGER         NOT NULL,
        MGRSSN          CHAR (9),
        MGRDATE         DATE    );

CREATE TABLE DEPT_LOCATIONS (

        DNUMBER         INTEGER         NOT NULL,
        DLOCATION       VARCHAR (15)    NOT NULL   );

CREATE TABLE PROJECT  (

        PNAME           VARCHAR (15)    NOT NULL,
        PNUMBER         INTEGER         NOT NULL,
        PLOCATION       VARCHAR (15),
        DNUM            INTEGER         NOT NULL   );

CREATE TABLE WORKS_ON  (

        ESSN            CHAR (9)        NOT NULL,
        PNO             INTEGER         NOT NULL,
        HOURS           DECIMAL (3,1)   NOT NULL   );
```

CREATE TABLE DEPENDENT (

ESSN	**CHAR (9)**	**NOT NULL,**
DEP_NAME	**VARCHAR (15)**	**NOT NULL,**
SEX	**CHAR (1),**	
BDATE	**CHAR (9),**	
RELATION	**VARCHAR (8));**	

Δεν είναι δύσκολο να διαπιστώσει κανείς, πως στην παραπάνω σύνταξη της εντολής **CREATE TABLE** δεν έχουμε τη δυνατότητα να καθορίσουμε **ποιο είναι το πρωτεύον κλειδί του πίνακα**. Αυτό είναι κάτι που ποικίλλει από γλώσσα σε γλώσσα, καθώς σε άλλες εκδόσεις της **SQL**, είναι δυνατός ο καθορισμός των κλειδιών των πινάκων, κατά τη φάση της δημιουργίας τους, με την **CREATE TABLE**. Η παράμετρος που καθορίζει το πρωτεύον κλειδί σε αυτές τις περιπτώσεις, είναι η **UNIQUE** η οποία χρησιμοποιείται ως κανόνας μαζί με τον κανόνα **NOT NULL**. Έτσι για να δείξουμε πως το πεδίο **SSN** είναι πρωτεύον κλειδί του πίνακα **EMPLOYEE** μπορούμε να το δηλώσουμε στην **CREATE TABLE** ως

SSN CHAR (9) NOT NULL UNIQUE

Χρησιμοποιώντας την παράμετρο **UNIQUE** μπορούμε να ορίσουμε μόνο απλά και όχι σύνθετα πρωτεύοντα κλειδιά. Για να ορίσουμε ένα σύνθετο κλειδί, θα πρέπει όπως θα δούμε στη συνέχεια, να δημιουργήσουμε ένα **δείκτη** χρησιμοποιώντας την εντολή **CREATE UNIQUE INDEX**.

Ας σημειωθεί επιπλέον, πως οι πίνακες που δημιουργούνται με την εντολή **CREATE TABLE**, είναι **οι βασικοί πίνακες της βάσης (base tables)**, διότι **τα δεδομένα που περιέχουν, αποθηκεύονται σε αρχεία του συστήματος**. Αντίθετα, οι **όψεις**, είναι **εικονικοί πίνακες (virtual tables)**, που μπορεί να αποθηκεύονται και αυτοί σε αρχείο, χωρίς όμως κάτι τέτοιο να είναι υποχρεωτικό. Τέλος, θα πρέπει να αναφερθεί, πως **η σειρά με την οποία αποθηκεύονται οι τιμές των πεδίων στους πίνακες, είναι ίδια με τη σειρά με την οποία έχουν δηλωθεί τα πεδία των πινάκων κατά την κλήση της εντολής**. Αυτό σημαίνει, πως στον πίνακα **EMPLOYEE**, η **ημερομηνία γέννησης (BDATE)** για τον κάθε εργαζόμενο, αποθηκεύεται μετά από τον κωδικό **SSN**, διότι το πεδίο **BDATE** στην εντολή **CREATE TABLE** έχει δηλωθεί μετά το πεδίο **SSN** στον πίνακα **EMPLOYEE**.

Οι άλλες δύο εντολές που περιλαμβάνονται στη γλώσσα ορισμού δεδομένων, είναι η **DROP TABLE** και η **ALTER TABLE**. Η εντολή **DROP TABLE** χρησιμοποιείται **για τη διαγραφή ενός πίνακα από τη βάση δεδομένων**, ενέργεια, η οποία διαγράφει όχι μόνο τα δεδομένα του πίνακα, αλλά και τον ίδιο τον πίνακα. Παράδειγμα χρήσης της εντολής είναι η κλήση της με τη μορφή

DROP TABLE DEPENDENTS

η οποία διαγράφει τον πίνακα **DEPENDENTS** μαζί με όλα τα δεδομένα που περιέχει, από τη βάση δεδομένων της εταιρείας.

Τέλος, η εντολή **ALTER TABLE** επιτρέπει **την τροποποίηση της δομής του πίνακα μετά τη δημιουργία του**, και πιο συγκεκριμένα, **την προσθήκη νέων πεδίων σε αυτόν** – αν και σε ορισμένες περιπτώσεις επιτρέπεται τόσο η **διαγραφή** όσο και η **μεταβολή** του τύπου δεδομένων για κάποιο από τα πεδία του πίνακα. Για να προσθέσουμε ένα νέο πεδίο σε κάποιο πίνακα, θα πρέπει να καθορίσουμε το όνομά του και τον τύπο του. Για παράδειγμα η εντολή

ALTER TABLE EMPLOYEE ADD JOB VARCHAR (12)

προσθέτει στον πίνακα **EMPLOYEE** το πεδίο **JOB** που είναι **συμβολοσειρά** με μέγιστο μήκος ίσο με **12 χαρακτήρες**.

(β) Διαχείριση όψεων

Η **όψη (view)**, στην ορολογία της **SQL**, ορίζεται ως **ένας απλός πίνακας, ο οποίος προκύπτει από το συνδυασμό των πεδίων ενός ή περισσοτέρων πινάκων**. Αυτοί οι πίνακες, με τη σειρά τους, μπορεί να είναι είτε **βασικοί πίνακες** τα δεδομένα των οποίων αποθηκεύονται στα αρχεία του συστήματος, **ή άλλες όψεις**, οι οποίες θεωρούνται **εικονικοί πίνακες** που τους χρησιμοποιούμε χωρίς να αποθηκεύουμε τα περιεχόμενά τους.

Δημιουργώντας μια όψη, μπορούμε να έχουμε εύκολη προσπέλαση σε πεδία τα οποία χρησιμοποιούμε πολύ συχνά. Ας υποθέσουμε για παράδειγμα πως **για κάποιο υπάλληλο χρησιμοποιούμε συνέχεια το όνομά του, καθώς και τα ονόματα των projects στα οποία εργάζεται**. Αυτά τα δεδομένα βρίσκονται αποθηκευμένα σε τρεις διαφορετικούς πίνακες – τους πίνακες **EMPLOYEE, PROJECT** και **WORKS_ON** – και αυτό σημαίνει πως κάθε φορά που θέλουμε να τα χρησιμοποιήσουμε θα πρέπει να καταφύγουμε στις πράξεις της σύζευξης αυτών των πινάκων. Δημιουργώντας όμως μια όψη, η οποία θα προκύπτει από αυτές τις συζεύξεις των τριών πινάκων, μπορούμε να χρησιμοποιούμε αυτά τα πεδία πάρα πολύ εύκολα, θεωρώντας τα ως πεδία τα οποία ανήκουν στον ίδιο πίνακα. Οι βασικοί πίνακες από τους οποίους δημιουργείται μια όψη, αναφέρονται και ως **πίνακες ορισμού της όψης (defining tables)**.

Για να δημιουργήσουμε μια όψη, χρησιμοποιούμε την εντολή **CREATE VIEW**. Η εντολή δέχεται ως όρισμα **το όνομα της όψης που πρόκειται να δημιουργήσουμε**. Επίσης, εάν το επιθυμούμε, μπορούμε να καθορίσουμε και τα ονόματα των πεδίων αυτής της όψης – στην αντίθετη περίπτωση, θα χρησιμοποιηθούν τα ονόματα των πεδίων των πινάκων ορισμού της. Τέλος επειδή τα πεδία της όψης θα καθοριστούν εφαρμόζοντας ένα **ερώτημα (query)** επί των βασικών πινάκων της βάσης, θα πρέπει να περάσουμε ως παράμετρο στην εντολή και ένα τέτοιο ερώτημα. Ένα παράδειγμα χρήστης της **CREATE VIEW** παρουσιάζεται στη συνέχεια:

```
CREATE VIEW     WORKS_ON1
AS              SELECT FNAME, LNAME, PNAME, HOURS
                FROM EMPLOYEE, PROJECT, WORKS_ON
                WHERE SSN = ESSN AND PNO = PNUMBER
```

Η παραπάνω εντολή δημιουργεί μια καινούρια όψη που φέρει το όνομα **WORKS_ON1** και περιέχει τα πεδία **FNAME, LNAME, PNAME** και **HOURS** των πινάκων **EMPLOYEE, PROJECT** και **WORKS_ON**. Έχοντας δημιουργήσει τώρα αυτή την όψη, μπορούμε να τη χρησιμοποιήσουμε σαν ένα απλό πίνακα εφαρμόζοντας πάνω του όλες τις πράξεις που έχουν να κάνουν με τη διαχείριση πινάκων. Για παράδειγμα, προκειμένου να βρούμε τα ονοματεπώνυμα των υπαλλήλων που εργάζονται σε όλα τα **PROJECTS** περισσότερες από **10 ώρες την εβδομάδα**, δεν χρειάζεται να συνδυάσουμε τους πίνακες **EMPLOYEE, PROJECT** και **WORKS_ON**, αλλά μπορούμε απλά να γράψουμε

```
SELECT    FNAME, LNAME
FROM      WORKS_ON1
WHERE     HOURS > 10
```

Ένα δεύτερο παράδειγμα χρήσης της **CREATE VIEW** παρουσιάζεται στη συνέχεια:

```
CREATE VIEW    DEPT_INFO (DNAME, EMP_NO, AVG_SAL)
AS             SELECT DNAME, COUNT (*), SUM (SALARY)
               FROM DEPARTMENT, EMPLOYEE
               WHERE DNUMBER, DNO
               GROUP BY DNAME
```

Αυτή η εντολή δημιουργεί μια καινούρια όψη με το όνομα **DEPT_INFO** που για κάθε τμήμα περιέχει το όνομά του, το πλήθος των υπαλλήλων που εργάζονται σε αυτό, καθώς και το μέσο όρο των μισθών τους. Στην προκειμένη περίπτωση, **τα ονόματα των πεδίων της όψης, δεν είναι τα ίδια με τα ονόματα των αντίστοιχων πεδίων του πίνακα, αλλά τα καθορίζουμε επ' ακριβώς κατά την κλήση της εντολής**. Περισσότερες λεπτομέρειες όσον αφορά τη χρήση της εντολής **SELECT** που χρησιμοποιείται στα δύο παραπάνω παραδείγματα, θα δοθεί στις επόμενες σελίδες.

Τέλος, η **διαγραφή** μιας όψης γίνεται με την εντολή **DROP VIEW**. Παράδειγμα χρήσης της εντολής αυτής, είναι η κλήση της με τη μορφή

```
DROP VIEW WORKS_ON1
```

η οποία διαγράφει την όψη **WORKS_ON1** από τη βάση δεδομένων της εταιρείας.

(γ) Διαχείριση δεικτών

Η έννοια του **δείκτη (index)** σε όλα τα συστήματα διαχείρισης βάσεων δεδομένων, είναι στενά συνυφασμένη με την έννοια της **αναζήτησης πληροφορίας**. Ο δείκτης δεν είναι τίποτε άλλο από **μια δομή δεδομένων που αποθηκεύεται σε ειδικά αρχεία της βάσης (index files) και έχει ως στόχο να επιταχύνει τη διαδικασία αναζήτησης πληροφορίας από τους πίνακες της βάσης**. Οι δείκτες συνήθως ορίζονται για συγκεκριμένα πεδία αυτών των πινάκων **(indexing fields)** και για κάθε τιμή των εν λόγω πεδίων, αποθηκεύουν ένα σύνολο από **pointers** που δείχνουν όλες τις περιοχές της συσκευής δευτερεύουσας αποθήκευσης στις οποίες βρίσκονται αποθηκευμένα, τα δεδομένα της κάθε εγγραφής. Τα αρχεία των δεικτών είναι επομένως αρ-

κετά μικρότερα σε μέγεθος σε σχέση με τα αρχεία δεδομένων, και η διαδικασία αναζήτησης της πληροφορίας δια της χρήσης τους, λαμβάνει χώρα δια της εφαρμογής γνωστών τεχνικών αναζήτησης, όπως είναι για παράδειγμα η **δυαδική αναζήτηση (binary search)**.

Η δημιουργία ενός **δείκτη** στη γλώσσα **SQL**, γίνεται χρησιμοποιώντας την εντολή **CREATE INDEX**, και περιλαμβάνει **τον καθορισμό του ονόματος του δείκτη, και του πεδίου του πίνακα για το οποίο θα δημιουργηθεί**. Για παράδειγμα, προκειμένου να δημιουργήσουμε ένα δείκτη για το πεδίο **LNAME** του πίνακα **EMPLOYEE**, θα καλέσουμε την εντολή **CREATE INDEX** με τη μορφή

```
CREATE INDEX    LNAME_INDEX
ON              EMPLOYEE (LNAME)
```

Η ταξινόμηση των τιμών του πεδίου πάνω στο οποίο θα δημιουργηθεί ο δείκτης, μπορεί να είναι είτε **αύξουσα (ascending)** είτε **φθίνουσα (descending)**, και καθορίζεται από το χρήστη, αναγράφοντας τις παραμέτρους **ASC** και **DESC** αντίστοιχα, αμέσως μετά το όνομα του πεδίου. Εάν δεν χρησιμοποιήσουμε αυτή την παράμετρο, όπως στο παραπάνω παράδειγμα, ο δείκτης χρησιμοποιεί την αύξουσα ταξινόμηση για τη διάταξη των τιμών του πεδίου. Η δημιουργία ενός δείκτη για ένα συνδυασμό πεδίων είναι επίσης δυνατή, και μάλιστα έχουμε τη δυνατότητα να καθορίσουμε διαφορετικό είδος ταξινόμησης για κάθε απλό πεδίο. Έτσι η εντολή

```
CREATE INDEX    NAMES_INDEX
ON              EMPLOYEE (LNAME ASC, FNAME DESC, MINIT)
```

δημιουργεί ένα **δείκτη** πάνω στα πεδία **LNAME, MINIT** και **FNAME**, εκ των οποίων το **LNAME** ταξινομείται **κατά αύξουσα διάταξη** ενώ το **FNAME** ταξινομείται **κατά φθίνουσα διάταξη**. Τέλος όταν δεν έχουμε τη δυνατότητα να ορίσουμε κατά το στάδιο δημιουργίας των πινάκων το πρωτεύον κλειδί του πίνακα, μπορούμε να το κάνουμε δημιουργώντας ένα **INDEX** για αυτό το πεδίο, και ταυτόχρονα δηλώνοντας πως οι τιμές που καταχωρούνται σε αυτό, θα πρέπει να είναι μοναδικές:

```
CREATE UNIQUE INDEX    SSN_INDEX
ON                     EMPLOYEE (SSN)
```

Ας σημειωθεί πως ο καθορισμός του πρωτεύοντος κλειδιού του πίνακα με τον τρόπο αυτό, θα πρέπει να γίνει πριν την καταχώρηση εγγραφών σε αυτόν, διότι εάν υπάρχουν ήδη εγγραφές που έχουν τις ίδιες τιμές στο πεδίο που θέλουμε να κάνουμε πρωτεύον κλειδί, η παραπάνω διαδικασία θα αποτύχει.

Η διαγραφή ενός δείκτη από τη βάση δεδομένων, γίνεται χρησιμοποιώντας την εντολή **DROP_INDEX**. Παράδειγμα χρήσης αυτής της εντολής είναι η κλήση της με τη μορφή **DROP INDEX LNAME_INDEX** η οποία διαγράφει το δείκτη **LNAMES_INDEX**, από τη βάση δεδομένων της εταιρείας.

Η ΓΛΩΣΣΑ ΧΕΙΡΙΣΜΟΥ ΔΕΔΟΜΕΝΩΝ

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

ΕΙΣΑΓΩΓΗ ΕΓΓΡΑΦΩΝ ΣΕ ΠΙΝΑΚΑ ΤΗΣ ΒΑΣΗΣ – Η ΕΝΤΟΛΗ INSERT

Η εισαγωγή μιας νέας εγγραφής σε κάποιον από τους πίνακες της βάσης δεδομένων, γίνεται δια της χρήσης της εντολής **INSERT**, η οποία στη γενική περίπτωση, καλείται με τη μορφή

INSERT INTO <TABLE NAME>
VALUES (<v_1>, <v_2>, …… , <v_n>)

όπου **<TABLE NAME>** είναι το όνομα του πίνακα στον οποίο θα καταχωρηθεί η νέα εγγραφή, και **<v_1>, <v_2>, …… , <v_n>**, είναι οι τιμές των πεδίων της εγγραφής που θέλουμε να καταχωρήσουμε. Έτσι η εντολή

INSERT INTO EMPLOYEE
VALUES ('Richard', 'K', 'Marini', '653298653', '30-Dec-52',
'98 Oak Forest, Katy, TX', 'M', 37000, '987654321', 4)

καταχωρεί στον πίνακα **EMPLOYEE** ένα νέο υπάλληλο με όνομα **Richard K. Marini**, που έχει κωδικό SSN **653298653**, ημερομηνία γέννησης **30 Δεκεμβρίου 1952**, Διεύθυνση **98 Oak Forest, Katy, TX**, φύλλο **άρρεν**, μισθό **37000**, ο οποίος εποπτεύεται από τον **MANAGER** με κωδικό SSN **987654321**, και έχει τοποθετηθεί στο τμήμα με κωδικό αριθμό **4**.

Από το παραπάνω παράδειγμα, διαπιστώνουμε πως οι τιμές των πεδίων στην εντολή **INSERT**, θα πρέπει να αναγραφούν με τη σειρά με την οποία δηλώθηκαν τα πεδία του πίνακα στην εντολή **CREATE TABLE**, έτσι ώστε **η κάθε τιμή να καταχωρηθεί στη σωστή στήλη του πίνακα**, και να διασφαλισθεί η συμβατότητα ανάμεσα στον τύπο δεδομένων του πεδίου, και στον τύπο δεδομένων των καταχωρημένων τιμών. Εάν για κάποιο πεδίο δεν επιθυμούμε να καταχωρήσουμε κάποια τιμή, τότε η αντίστοιχη τιμή στην εγγραφή που θα καταχωρήσουμε θα είναι ίση με **NULL.** Εναλλακτικά, μπορούμε να παραλείψουμε τις τιμές **NULL**, για κάποια πεδία του πίνακα. Στην περίπτωση αυτή όμως θα πρέπει να καθορίσουμε επ' ακριβώς σε ποιες στήλες του πίνακα θα καταχωρηθούν οι τιμές που καθορίζουμε. Έτσι η εντολή

INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', '653298653')

υποδηλώνει ξεκάθαρα, πως η νέα εγγραφή που θα καταχωρήσουμε περιλαμβάνει μόνο **το όνομα, το επώνυμο και τον κωδικό SSN για τον νέο EMPLOYEE**. Θα πρέπει ωστόσο η σειρά με την οποία αναγράφονται οι τιμές της νέας εγγραφής, να είναι και πάλι ίδια με τη σειρά με την οποία αναγράφονται τα ονόματα των πεδίων, δίπλα από το όνομα του πίνακα, έτσι ώστε η κάθε τιμή να καταχωρηθεί στο σωστό πεδίο.

Ας σημειωθεί πως στη γενική περίπτωση η γλώσσα ορισμού δεδομένων, **δεν ελέγχει εάν η εγγραφή που πρόκειται να καταχωρηθεί ικανοποιεί τους κανόνες ακεραιότητας (integrity constraints) που έχουμε καθορίσει κατά το λογικό σχεδιασμό του σχήματος της βάσης**. Έτσι, η παραπάνω εγγραφή θα προστεθεί στον πίνακα **EMPLOYEE**, ακόμη και εάν ο κωδικό του τμήματος στον οποίο πρόκειται να καταχωρηθεί ο νέος υπάλληλος, αναφέρεται σε ένα ανύπαρκτο τμήμα. Είναι λοιπόν καθήκον του χρήστη να εξετάσει εάν πληρούνται οι κανόνες ακεραιότητας κάθε φορά που επιχειρεί να προσθέσει μια νέα εγγραφή. Αντίθετα η εντολή **INSERT** δεν θα εκτελεστεί εάν καταχωρήσουμε τιμή **NULL** σε κάποια από τα πεδία του πίνακα, για τα οποία στην **CREATE TABLE** έχουμε καθορίσει τον κανόνα «**NOT NULL**» - στην περίπτωση αυτή, η διαδικασία καταχώρησης της νέας εγγραφής, θα αποτύχει.

Τέλος, έχουμε τη δυνατότητα να συνδυάσουμε τις εντολές **INSERT** και **SELECT** προκειμένου να καταχωρήσουμε σε ένα πίνακα, περισσότερες από μια εγγραφές ταυτόχρονα. Για παράδειγμα η εντολή

```
INSERT INTO     DEPTS_INFO (DNAME, EMP_NO, TOTAL_SAL)
                SELECT DNAME, COUNT (*), SUM (SALARY)
                FROM DEPARTMENT, EMPLOYEE
                WHERE DNUMBER = DNO
                GROUP BY DNAME
```

καταχωρεί στον πίνακα **DEPTS_INFO** ένα πλήθος εγγραφών, που για κάθε τμήμα της εταιρείας, περιλαμβάνει **το όνομα του τμήματος, το πλήθος των υπαλλήλων που εργάζονται σε αυτό, και το σύνολο των μισθών τους**. Είναι προφανές πως για να εκτελεστεί η παραπάνω εντολή, ο πίνακας **DEPTS_INFO** θα πρέπει να έχει δημιουργηθεί σε προηγούμενο στάδιο της διαδικασίας, και η δομή του θα πρέπει να είναι συμβατή με τη δομή του πίνακα που επιστρέφει η **SELECT** έτσι ώστε να είναι δυνατή η προσθήκη σε αυτόν των νέων εγγραφών.

ΔΙΑΓΡΑΦΗ ΕΓΓΡΑΦΩΝ ΑΠΟ ΠΙΝΑΚΑ ΤΗΣ ΒΑΣΗΣ
Η ΕΝΤΟΛΗ DELETE

Η εντολή **DELETE**, χρησιμοποιείται για τη διαγραφή εγγραφών από κάποιο πίνακα. Οι εγγραφές διαγράφονται **μόνο από ένα πίνακα κάθε φορά** και καθορίζονται δια της χρήσης της πρότασης **WHERE** η οποία χρησιμοποιείται με τον ίδιο τρόπο που χρησιμοποιείται και στη **SELECT**. Εάν δεν χρησιμοποιηθεί η πρόταση **WHERE**, η εντολή **DELETE** διαγράφει όλες τις εγγραφές του πίνακα πάνω στον οποίο εφαρμόζεται. Αυτό βέβαια δεν σημαίνει πως λαμβάνει χώρα και διαγραφή του πίνακα από τη βάση – για να απομακρυνθεί και ο ίδιος ο πίνακας θα πρέπει να χρησιμοποιηθεί και η εντολή **DROP TABLE**.

Παραδείγματα χρήσης της εντολής **DELETE** παρουσιάζονται στη συνέχεια:

```
DELETE FROM    EMPLOYEE
WHERE          LNAME = 'Brown'

DELETE FROM    EMPLOYEE
WHERE          SSN = '123456789'

DELETE FROM    EMPLOYEE
WHERE          DNO IN    ( SELECT DNUMBER
                           FROM DEPARTMENT
                           WHERE DNAME = 'Research' )
```

Στα δύο πρώτα παραδείγματα η εντολή **DELETE** διαγράφει μια εγγραφή κάθε φορά – αυτή που αναφέρεται στον **Brown**, και αυτή που αναφέρεται στον υπάλληλο με κωδικό **SSN = 123456789**. Στο τρίτο παράδειγμα οι εγγραφές που πρόκειται να διαγραφούν, καθορίζονται από μια πρόταση **SELECT** η οποία επιστρέφει τον κωδικό του τμήματος με όνομα **DEPARTMENT**. Επομένως, η παραπάνω εντολή, διαγράφει **όλους τους υπαλλήλους της εταιρείας, που ανήκουν στο Research Department**.

Τέλος, η εντολή **DELETE FROM EMPLOYEE** διαγράφει **όλες τις εγγραφές του πίνακα EMPLOYEE, δηλαδή όλους τους υπαλλήλους της εταιρείας**.

ΤΡΟΠΟΠΟΙΗΣΗ ΤΟΥ ΠΕΡΙΕΧΟΜΕΝΟΥ ΤΩΝ ΕΓΓΡΑΦΩΝ ΤΩΝ ΠΙΝΑΚΩΝ ΤΗΣ ΒΑΣΗΣ – Η ΕΝΤΟΛΗ UPDATE

Η εντολή **UPDATE**, χρησιμοποιείται για την τροποποίηση των τιμών των πεδίων επιλεγμένων εγγραφών κάποιου από τους πίνακες της βάσης. Αυτές οι εγγραφές, καθορίζονται όπως και στην εντολή **DELETE**, από μια πρόταση **WHERE**, που καθορίζει **τα χαρακτηριστικά των εγγραφών που θέλουμε να τροποποιήσουμε**. Τα πεδία των οποίων οι τιμές θα μεταβληθούν, καθώς επίσης και οι νέες τιμές που θα καταχωρήσουμε σε αυτά, καθορίζονται, χρησιμοποιώντας την πρόταση **SET**. Έτσι, η εντολή

```
UPDATE    PROJECT
SET       PLOCATION = 'Bellaire', DNUM = 5
WHERE     PNUMBER = 10
```

τροποποιεί τις τιμές των πεδίων της εγγραφής του πίνακα **PROJECT** για την οποία το πεδίο **PNUMBER** έχει την τιμή **10**, και αποδίδει στο πεδίο **PLOCATION** την τιμή **Bellaire**, και στο πεδίο **DNUM**, την τιμή **5**.

Τέλος, όπως και στην εντολή **DELETE**, μπορούμε να τροποποιήσουμε τις τιμές πολλών εγγραφών ταυτόχρονα, εάν συνδυάσουμε την εντολή **UPDATE** με την εντολή **SELECT**. Έτσι η εντολή

```
UPDATE    EMPLOYEE
SET       SALARY = SALARY * 1.1
WHERE     DNO IN   ( SELECT DNUMBER
                     FROM DEPARTMENT
                     WHERE DNAME = 'Research' )
```

τροποποιεί τους **μισθούς** όλων των **υπαλλήλων** που εργάζονται στο **Research Department**, προκαλώντας **αύξηση της τιμής τους κατά 10%**.

ΕΠΙΛΟΓΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΟΥΣ ΠΙΝΑΚΕΣ ΤΗΣ ΒΑΣΗΣ Η ΕΝΤΟΛΗ SELECT

Εκτός από την ομάδα διαδικασιών η οποία τροποποιεί το περιεχόμενο των πινάκων της βάσης, υπάρχουν και άλλες διαδικασίες, οι οποίες δε μεταβάλλουν τα δεδομένα των πινάκων, αλλά **αναζητούν και εμφανίζουν δεδομένα, τα οποία πληρούν κάποια συγκεκριμένα κριτήρια επιλογής**. Αυτού του είδους οι διαδικασίες ονομάζονται **ερωτήματα (queries)** και πραγματοποιούνται δια της χρήσης μιας εκ των εντολών της **γλώσσας χειρισμού δεδομένων (Data Manipulation Language, DML)**, που φέρει το όνομα **SELECT**.

Η χρήση της εντολής **SELECT** ακολουθεί τη σύνταξη

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>
```

όπου ο κατάλογος **<attribute list>** περιέχει **τα ονόματα των πεδίων που θέλουμε να ανακτήσουμε**, ο κατάλογος **<table list>** περιέχει **τα ονόματα των πινάκων που θα χρησιμοποιηθούνε στη δημιουργία του ερωτήματος**, ενώ **<condition>**, είναι **το κριτήριο αναζήτησης, που θα πρέπει να ικανοποιείται από τα δεδομένα των πινάκων**, προκειμένου αυτά να εμφανιστούν στο τελικό αποτέλεσμα.

Ένα παράδειγμα της εντολής **SELECT** είναι η κλήση της με τη μορφή

```
SELECT    BDATE, ADDRESS
FROM      EMPLOYEE
WHERE     FNAME='John' AND MINIT='B' AND LNAME='Smith'
```

Η εντολή αυτή εφαρμόζεται πάνω στον πίνακα **EMPLOYEE** και επιστρέφει **την ημερομηνία γέννησης και τη διεύθυνση του υπαλλήλου, που φέρει το όνομα John B. Smith**.

Εάν τα δεδομένα που περιλαμβάνονται στο ερώτημα ανήκουν σε περισσότερους από έναν πίνακες, τότε η εντολή **SELECT** θα εφαρμοσθεί πάνω σε όλους αυτούς τους πίνακες. Επιπλέον, στην προκειμένη περίπτωση, εκτός από τις επιθυμητές τιμές των πεδίων που θέλουμε να εμφανίσουμε, θα πρέπει στην πρόταση **WHERE**, να συμπεριλάβουμε και τη **συνθήκη σύζευξης (join condition)** που καθορίζει ποιες στήλες των δύο πινάκων συσχετίζονται μεταξύ τους. Ας πάρουμε για παράδειγμα την εντολή

```
SELECT    FNAME, LNAME, ADDRESS
FROM      EMPLOYEE
WHERE     DNAME = 'Research' AND DNUMBER = DNO
```

η οποία εμφανίζει το ονοματεπώνυμο και τη διεύθυνση όλων των υπαλλήλων που εργάζονται στο **Research Department**. Στο ερώτημα αυτό, τα πεδία που θα ανακτήσουμε ανήκουν στον πίνακα **EMPLOYEE**, αλλά το πεδίο **DNAME** που περιλαμβάνεται στην πρόταση **WHERE**, ανήκει στον πίνακα **DEPARTMENT**. Αυτό σημαίνει πως στη σύνταξη της εντολής, θα εμφανιστούν και οι δύο πίνακες. Λόγω της εμφάνισης περισσότερων από ένα πινάκων, δεν αρκεί μόνο ο καθορισμός του κριτηρίου επιλογής, αλλά θα πρέπει να συμπεριλάβουμε στο εν λόγω ερώτημα και τα κοινά πεδία των δύο πινάκων τα οποία συσχετίζονται μεταξύ τους. Για το λόγο αυτό, στην πρόταση **WHERE**, εκτός από το κριτήριο επιλογής **DNAME = 'Research'**, καθορίζουμε και τη συνθήκη σύζευξης που είναι η **DNUMBER = DNO**.

Οι δύο αυτές συνθήκες, θα πρέπει να ικανοποιούνται ταυτόχρονα – και αυτό πραγματοποιείται, δια της χρήσης του λογικού τελεστή **AND**. Η χρήση των άλλων λογικών τελεστών **OR** και **NOT** είναι επίσης δυνατή, ενώ στη συνθήκη καθορισμού των κριτηρίων αναζήτησης μπορούμε να χρησιμοποιήσουμε και όλους τους τελεστές σύγκρισης που είχαμε αναλύσει στο προηγούμενο κεφάλαιο. Έτσι για να βρούμε όλους τους υπαλλήλους που έχουν μισθό μεγαλύτερο από **30000**, θα πρέπει στην πρόταση **WHERE** να συμπεριλάβουμε τη συνθήκη **SALARY > 30000**.

Σε περιπτώσεις κατά της οποίες υπάρχουν πεδία που ανήκουν σε διαφορετικούς πίνακες, αλλά φέρουν το ίδιο όνομα, το ερώτημα χαρακτηρίζεται από κάποιο βαθμό ασάφειας, όσον αφορά τον πίνακα στον οποίο ανήκει το κάθε πεδίο. Για να αναδείξουμε αυτή την ασάφεια, ας υποθέσουμε πως στον πίνακα **EMPLOYEE**, το πεδίο που περιέχει τον κωδικό του τμήματος στον οποίο εργάζεται, δεν ονομάζεται **DNO**, αλλά **DNUMBER**, έχει δηλαδή το ίδιο όνομα με το αντίστοιχο πεδίο του πίνακα **DEPARTMENT**. Στην περίπτωση αυτή η πρόταση **WHERE** του προηγούμενου ερωτήματος θα λάβει τη μορφή

```
WHERE   DNAME = 'Research' AND DNUMBER = DNUMBER
```

η οποία ασφαλώς περιέχει μεγάλο ποσοστό απροσδιοριστίας, καθώς δεν αποσαφηνίζει σε ποιο πίνακα ανήκει το κάθε ένα από τα δύο πεδία **DNUMBER**. Για να απομακρύνουμε αυτή την ασάφεια, θα πρέπει να αναφερθούμε στα πεδία των πινάκων, χρησιμοποιώντας ένα πιο πλήρη τρόπο σύνταξης που έχει τη μορφή

```
TABLE_NAME.ATTRIBUTE_NAME
```

περιλαμβάνει δηλαδή, τον καθορισμό, όχι μόνο του ονόματος του πεδίου, αλλά και το όνομα του πίνακα στον οποίο ανήκει το εν λόγω πεδίο. Έτσι η ασάφεια που χαρακτηρίζει το προηγούμενο ερώτημα, αίρεται, εάν το επαναδιατυπώσουμε στη μορφή

```
SELECT    FNAME, LNAME, ADDRESS
FROM      EMPLOYEE
WHERE     DNAME = 'Research' AND
          DEPARTMENT.DNUMBER = EMPLOYEE.DNUMBER
```

Μια ακόμη ιδιάζουσα περίπτωση που αξίζει να αναφερθεί, αφορά την εμφάνιση της ασάφειας με τον τρόπο που την ορίσαμε παραπάνω, όχι όσον αφορά τα πεδία των πινάκων, αλλά όσον αφορά τους ίδιους τους πίνακες. Χαρακτηριστικό παράδειγμα αυτής της περίπτωσης είναι οι **αναδρομικές συσχετίσεις (recursive relationships)** που υφίστανται ανάμεσα σε δύο αντίγραφα του ίδιου πίνακα. Αυτό π.χ. συμβαίνει στην περίπτωση κατά την οποία θέλουμε **για κάθε υπάλληλο, να ανακτήσουμε το ονοματεπώνυμό του, καθώς και το ονοματεπώνυμο του MANAGER που τον εποπτεύει**. Επειδή όμως και ο **MANAGER** είναι και ο ίδιος ένας υπάλληλος, και επομένως τα προσωπικά του στοιχεία βρίσκονται και αυτά αποθηκευμένα στον πίνακα **EMPLOYEE**, θα πρέπει στη σύνταξη του ερωτήματος, να χρησιμοποιήσουμε δύο φορές τον πίνακα **EMPLOYEE**, με αποτέλεσμα την εμφάνιση απροσδιοριστίας όσον αφορά τον ακριβή ρόλο του κάθε αντιγράφου του πίνακα κατά την εκτέλεση του παραπάνω ερωτήματος.

Η άρση της απροσδιοριστίας στην περίπτωση αυτή, λαμβάνει χώρα δια της χρήσης για κάθε αντίγραφο του πίνακα και ενός διαφορετικού ονόματος. Αυτό το όνομα δεν επηρεάζει τις ιδιότητες του πίνακα, αλλά πρόκειται απλά για ένα **ψευδώνυμο (alias)**. Έτσι, το παραπάνω ερώτημα θα διατυπωθεί με τη μορφή

```
SELECT    E.FNAME, E.LNAME, S.FNAME, F.LNAME
FROM      EMPLOYEE  E  S
WHERE     E.SUPERSSN = S.SSN
```

στην οποία έχουμε ορίσει για κάθε αντίγραφο του πίνακα **EMPLOYEE** και ένα διαφορετικό ψευδώνυμο. Πιο συγκεκριμένα, το ψευδώνυμο **E**, αναφέρεται στον πίνακα **EMPLOYEE** με το ρόλο του **υπαλλήλου**, ενώ το ψευδώνυμο **S**, αναφέρεται στον πίνακα **EMPLOYEE** με το ρόλο του **επόπτη**. Στην περίπτωση αυτή αίρεται κάθε μορφή απροσδιοριστίας όσον αφορά τον τρόπο ερμηνείας του παραπάνω ερωτήματος, καθώς το κάθε αντίγραφο του πίνακα παίζει πλέον το δικό του ρόλο ο οποίος είναι διακριτός και πλήρως ορισμένος.

Στην περίπτωση κατά την οποία καλέσουμε την εντολή **SELECT** χωρίς την πρόταση **WHERE**, το αποτέλεσμα που θα λάβουμε θα περιλαμβάνει όλες τις εγγραφές του πίνακα, καθώς δεν καθορίζουμε κάποιο συγκεκριμένο κριτήριο αναζήτησης. Για παράδειγμα, εάν γράψουμε

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
```

θα λάβουμε τα ονοματεπώνυμα όλων των υπαλλήλων της εταιρείας, ενώ για να λάβουμε όλα τα πεδία της κάθε εγγραφής, θα χρησιμοποιήσουμε τον τελεστή «*». Έτσι η εντολή

```
SELECT    *
FROM      EMPLOYEE
WHERE     DNO = 5
```

θα επιστρέψει όλα τα πεδία των υπαλλήλων της εταιρείας, για τους οποίους ο κωδικός του τμήματος στο οποίο απασχολούνται έχει την τιμή **5**.

Η παράμετρος **DISTINCT**: Σε αντίθεση με τη **σχεσιακή άλγεβρα** όπου οι πίνακες διαπραγματεύονται **ως σύνολα από πλειάδες**, η γλώσσα **SQL** δεν ακολουθεί αυτή την προσέγγιση, και δεν διαγράφει από μόνη της τις **διπλοεγγραφές** που ενδέχεται να εμφανιστούν στο τελικό αποτέλεσμα. Εάν για παράδειγμα χρησιμοποιήσουμε την εντολή

```
SELECT    SALARY
FROM      EMPLOYEE
```

για να ανακτήσουμε τους μισθούς των υπαλλήλων της εταιρείας, υπάρχει πιθανότητα στο τελικό αποτέλεσμα να εμφανίζεται η ίδια τιμή περισσότερες από μια φορές – στην περίπτωση κατά την οποία υπάρχουν πολλοί υπάλληλοι που παίρνουν τον ίδιο μισθό. Εάν όμως θέλουμε να κρατήσουμε μόνο μια φορά την κάθε τιμή θα καλέσουμε την εντολή με τη μορφή

```
SELECT    DISTINCT SALARY
FROM      EMPLOYEE
```

Στον επόμενο πίνακα παρουσιάζουμε το αποτέλεσμα των παραπάνω εντολών, προκειμένου να γίνει καλύτερα κατανοητός, ο τρόπος λειτουργίας της **DISTINCT**.

SALARY	SALARY
30000	30000
40000	40000
25000	25000
43000	43000
38000	38000
25000	55000
25000	
55000	

Στον παραπάνω πίνακα η αριστερή στήλη περιέχει το αποτέλεσμα της εντολής χωρίς τη χρήση της παραμέτρου **DISTINCT**, η οποία χρησιμοποιείται στη δεύτερη περίπτωση. Όπως φαίνεται από τα περιεχόμενα των πινάκων, η χρήση της **DISTINCT** απομακρύνει τις τιμές που εμφανίζονται περισσότερες από μιας φορά, με αποτέλεσμα το σύνολο των τιμών που επιστρέφεται από τη **SELECT** να είναι πλέον **διακριτό**, και να περιλαμβάνει μόνο διαφορετικές τιμές.

Η παράμετρος **UNION**: σε ορισμένες εκδόσεις της **SQL** υποστηρίζεται η εφαρμογή επί των πινάκων της βάσης **των πράξεων που εφαρμόζονται ανάμεσα σε σύνολα**. Στην περίπτωση αυτή, οι πίνακες διαπραγματεύονται **ως σύνολα από πλειάδες**, με αποτέλεσμα **τη διαγραφή των διπλοεγγραφών εάν αυτές εμφανιστούν**. Υπενθυμίζουμε από τη θεωρία της σχεσιακής άλγεβρας πως για να είναι δυνατή η εφαρμογή των πράξεων από τη θεωρία συνόλων πάνω στους πίνακες της βάσης – οι πράξεις αυτές είναι **η ένωση (UNION), η τομή (INTERSECTION)** και **η διαφορά (DIFFERENCE)** – θα πρέπει οι πίνακες να είναι **συμβατοί ως προς την ένωση (union compatible)**, δηλαδή να έχουν **το ίδιο πλήθος πεδίων**, και τα αντίστοιχα πεδία των πινάκων να έχουν **το ίδιο σύνολο τιμών**.

Ως παράδειγμα εφαρμογής της ένωσης δύο πινάκων, θα επαναδιατυπώσουμε σε γλώσσα **SQL** ένα από τα παραδείγματα του προηγούμενου κεφαλαίου και πιο συγκεκριμένα εκείνο που ζητούσε να ανακτηθούν οι κωδικοί των **PROJECTS** στα οποία δουλεύει ο υπάλληλος με το επώνυμο **Smith**, τόσο ως **εργαζόμενος** όσο και ως **MANAGER** κάποιου τμήματος. Για να απαντήσουμε σε αυτό το ερώτημα, θα πρέπει να βρούμε τους κωδικούς των **PROJECTS** στα οποία ο **Smith** συμμετέχει ως **εργαζόμενος** και τους κωδικούς των **PROJECTS** στα οποία ο **Smith** συμμετέχει ως **MANAGER**, οπότε, το τελικό αποτέλεσμα θα αποτελείται από την ένωση των δύο συνόλων. Έτσι, το ερώτημα αυτό σε γλώσσα **SQL** θα διατυπωθεί ως εξής:

```
SELECT    PNUMBER
FROM      PROJECT, DEPARTMENT, EMPLOYEE
WHERE     DNUM = DNUMBER AND MGRSSN = SSN AND
          LNAME = 'Smith'
UNION
SELECT    PNUMBER
FROM      PROJECT, WORKS_ON, EMPLOYEE
WHERE     PNUMBER = PNO AND ESSN = SSN AND
          LNAME = 'Smith'
```

Σύνολα τιμών και φωλιασμένα ερωτήματα (nested queries): Σε ορισμένες περιπτώσεις, είναι δυνατή η χρήση στην πρόταση **WHERE** ενός **συνόλου (set)**, από το οποίο θα πρέπει να παίρνει τιμές, κάποιο από τα πεδία του πίνακα. Εάν για παράδειγμα ζητούμε τους **κωδικούς SSN** των υπαλλήλων που εργάζονται στα **PROJECTS** με κωδικό **1**, **2** ή **3**, μπορούμε να γράψουμε το ερώτημα με τη μορφή

```
SELECT    DISTINCT ESSN
FROM      WORKS_ON
WHERE     PNUMBER = 1 OR PNUMBER = 2 OR
          PNUMBER = 3
```

Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε τη σύνταξη

```
SELECT    DISTINCT ESSN
FROM      WORKS_ON
          WHERE PNUMBER IN (1, 2, 3)
```

Στην παραπάνω σύνταξη **ο τελεστής σύγκρισης (comparison operator) IN**, συγκρίνει την τιμή του πεδίου πάνω στο οποίο εφαρμόζεται με όλες τις τιμές του συνόλου, και επιστρέφει την τιμή **TRUE**, εάν διαπιστώσει ταύτιση με κάποια από αυτές τις τιμές.

Είναι προφανές πως στη γενική περίπτωση είναι πρακτικά δύσκολο να χρησιμοποιήσουμε ένα σύνολο τιμών με τον τρόπο που το κάναμε παραπάνω. Αυτό συμβαίνει διότι **αφ' ενός δεν μπορούμε να γνωρίζουμε τις τιμές που θα περιλαμβάνονται σε αυτό**, έτσι ώστε να τις παραθέσουμε τη μία μετά την άλλη, και αφ' ετέρου, **αυτές οι τιμές ενδέχεται να αλλάξουν στο μέλλον, καθώς η βάση χαρακτηρίζεται από συνεχή εισαγωγή, διαγραφή και τροποποίηση εγγραφών**. Για το λόγο αυτό,

δεν καταχωρούμε απευθείας αυτές τις τιμές, αλλά **τις ανακτούμε με μία πρόταση SELECT** την οποία διαβιβάζουμε ως παράμετρο στον τελεστή σύγκρισης **IN**. Με τον τρόπο αυτό καλούμε την εντολή **SELECT** μέσα από κάποια άλλη **SELECT**, κάτι που έχει ως αποτέλεσμα τη δημιουργία των επονομαζόμενων **φωλιασμένων ερωτημάτων (nested queries)**. Η εντολή **SELECT** που χρησιμοποιείται μαζί με τον τελεστή σύγκρισης **IN** στην πρόταση **WHERE**, ονομάζεται **εσωτερικό ερώτημα (inner query)**, ενώ η **SELECT** μέσα από την οποία καλείται το εσωτερικό ερώτημα, ονομάζεται **εξωτερικό ερώτημα (outer query)**. Χρησιμοποιώντας φωλιασμένα ερωτήματα, το παραπάνω παράδειγμα που ζητούσε τους κωδικούς των **PROJECTS** στα οποία ο **Smith** συμμετέχει τόσο ως **εργαζόμενος**, όσο και ως **MANAGER**, μπορεί να επαναδιατυπωθεί ως εξής:

```
SELECT    DISTINCT PNAME
FROM      PROJECT
WHERE     PNUMBER IN      (
                SELECT    PNUMBER
                FROM      PROJECT, DEPARTMENT
                          EMPLOYEE
                WHERE     DNUM = DNUMBER AND
                          MGRSSN = SSN AND
                          LNAME = 'Smith'    )
          OR
          PNUMBER IN      (
                SELECT    PNO
                FROM      WORKS_ON, EMPLOYEE
                WHERE     ESSN = SSN AND
                          LNAME = 'Smith'    )
```

Εκείνο που κάνουμε στην παραπάνω εντολή είναι να βρούμε το σύνολο των κωδικών των **PROJECTS** στα οποία ο **Smith** συμμετέχει τόσο ως **εργαζόμενος** όσο και ως **MANAGER**, και στη συνέχεια χρησιμοποιώντας τον λογικό τελεστή **OR** να ενώσουμε στην ουσία αυτά τα σύνολα. Επειδή δεν είναι δυνατόν να γνωρίζουμε σε κάθε χρονική στιγμή τις τιμές αυτών των συνόλων, τις ανακτούμε με δύο φωλιασμένες **SELECT**, έτσι ώστε σε κάθε περίπτωση τα σύνολα αυτά, να περιέχουν τις σωστές τιμές.

Μιλώντας γενικά, μπορούμε να έχουμε όσες φωλιασμένες **SELECT** θέλουμε, τη μία μέσα στην άλλη. Σε περιπτώσεις δε, κατά τις οποίες υπάρχει κάποια **ασάφεια** όσον αφορά τα ονόματα των πεδίων ή των πινάκων που περιλαμβάνονται στο ερώτημα, μπορούμε να χρησιμοποιήσουμε **ψευδώνυμα** (**aliases**) για αυτούς τους πίνακες, με τον τρόπο που έχουμε περιγράψει στα προηγούμενα παραδείγματα. Ας θεωρήσουμε π.χ. το ερώτημα

```
SELECT    E.FNAME , E.LNAME
FROM      EMPLOYEE E
WHERE     E.SSN IN  (
                SELECT    ESSN
                FROM      DEPENDENT
                WHERE     ESSN = E.SSN AND
                          E.FNAME = DEPENDENT_NAME
                          AND  SEX = E.SEX        )
```

το οποίο επιστρέφει **το όνομα και το επώνυμο κάθε υπαλλήλου, που έχει ένα προστατευόμενο μέλος με το ίδιο όνομα και το ίδιο φύλλο με το δικό του**. Επειδή ο πίνακας **EMPLOYEE** εμφανίζεται τόσο στο **εσωτερικό** όσο και στο **εξωτερικό** ερώτημα, θα πρέπει αφ ενός μεν, να δημιουργήσουμε ένα ψευδώνυμό του, **E**, έτσι ώστε να καταστήσουμε **διακριτό** το ρόλο των δύο αντιγράφων του, αφ ετέρου δε, να χρησιμοποιήσουμε τον πλήρη τρόπο σύνταξης κάθε φορά που χρησιμοποιούμε κάποιο πεδίο, έτσι ώστε να ξέρουμε σε ποιο πίνακα αναφέρεται.

Ολοκληρώνουμε την περιγραφή των φωλιασμένων **SELECT** στη γλώσσα **SQL**, με την περιγραφή των παραμέτρων **EXISTS** και **NOT EXISTS**, οι οποίες χρησιμοποιούνται για να ελέγξουν **εάν κάποιο φωλιασμένο ερώτημα έχει επιστρέψει πλειάδες ή όχι**. Εάν η εσωτερική **SELECT** έχει επιστρέψει πλειάδες οι οποίες θα χρησιμοποιηθούνε από το εξωτερικό ερώτημα, η παράμετρος **EXISTS** θα επιστρέψει την τιμή **TRUE**, ενώ στην αντίθετη περίπτωση θα επιστρέψει την τιμή **FALSE**. Εντελώς ανάλογος είναι και ο ρόλος της **NOT EXISTS** η οποία για τις δύο παραπάνω περιπτώσεις θα επιστρέψει τις τιμές **FALSE** και **TRUE** αντίστοιχα.

Για να κατανοήσουμε καλύτερα τη λειτουργία αυτών των δύο παραμέτρων, ας θεωρήσουμε το επόμενο ερώτημα : **έστω ότι θέλουμε να βρούμε τα ονοματεπώνυμα όλων των υπαλλήλων της εταιρείας που δεν έχουν προστατευόμενα μέλη**. Η εντολή **SELECT** που θα απαντήσει σε αυτό το ερώτημα, έχει τη μορφή

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     NOT EXISTS      (
                SELECT    *
                FROM      DEPENDENT
                WHERE     SSN = ESSN        )
```

Σε αυτό το ερώτημα, η εσωτερική **SELECT**, επιστρέφει όλες τις πλειάδες του πίνακα **DEPENDENT** που συσχετίζονται με κάποιο από τους υπαλλήλους της εταιρείας. Εάν αυτή η **SELECT** δεν επιστρέψει καμία πλειάδα, αυτό σημαίνει πως **ο υπάλληλος αυτός δεν έχει προστατευόμενα μέλη**, και επομένως ικανοποιεί τη συνθήκη για αυτό το ερώτημα – με τον τρόπο αυτό μπορούμε τελικά να ανακτήσουμε όλους τους υπαλλήλους που χαρακτηρίζονται από αυτή την ιδιότητα. Εντελώς ανάλογη είναι και η χρήση της παραμέτρου **EXISTS**.

Αθροιστικές συναρτήσεις (aggregate functions) : όπως έχει ήδη αναφερθεί στο προηγούμενο κεφάλαιο, σε ένα σχεσιακό σχήμα βάσεων δεδομένων, υπάρχει η δυνατότητα να εφαρμόσουμε πάνω στα δεδομένα των πινάκων, ένα σύνολο συναρτή-

σεων οι οποίες επιστρέφουν κάποιες τιμές. Στη γλώσσα **SQL** οι συναρτήσεις αυτές είναι η **COUNT** η οποία επιστρέφει **το πλήθος των εγγραφών ενός πίνακα** ή **το πλήθος κάποιων τιμών** ανάλογα με τον τρόπο με τον οποίο καλείται, η **MIN** και η **MAX** οι οποίες επιστρέφουν **την ελάχιστη και τη μέγιστη τιμή κάποιου συνόλου**, η **SUM** που επιστρέφει **το άθροισμα κάποιων τιμών**, και η **AVG** που επιστρέφει **το μέσο όρο τους**. Στις επόμενες παραγράφους παρουσιάζουμε κάποια παραδείγματα που θα μας επιτρέψουν να κατανοήσουμε τον τρόπο με τον οποίο χρησιμοποιούνται αυτές οι συναρτήσεις μέσα από τη γλώσσα χειρισμού δεδομένων.

1) Να **βρεθεί το άθροισμα των μισθών** όλων των **υπαλλήλων** που δουλεύουν **στο Research Department**, καθώς επίσης **ο μεγαλύτερος και ο μικρότερος μισθός και ο μέσος όρος τους**

```
SELECT    SUM (SALARY), MAX (SALARY), MIN (SALARY),
          AVG (SALARY)
FROM      EMPLOYEE, DEPARTMENT
WHERE     DNO = DNUMBER AND DNAME = "Research".
```

2) Να βρεθεί **το πλήθος των υπαλλήλων** που δουλεύουν στο **Research Department**

```
SELECT    COUNT (*)
FROM      EMPLOYEE, DEPARTMENT
WHERE     DNO = DNUMBER AND DNAME = "Research".
```

3) Να ανακτηθεί **το όνομα και το επώνυμο των υπαλλήλων της εταιρείας** που έχουν περισσότερα από **δύο προστατευόμενα μέλη**.

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     (    SELECT    COUNT (*)
               FROM      DEPENDENT
               WHERE     SSN=ESSN    ) ≥ 2
```

Οι παράμετροι **GROUP BY** και **HAVING**: Σε ορισμένες περιπτώσεις, επιθυμούμε την εφαρμογή των αθροιστικών συναρτήσεων, όχι σε όλες τις πλειάδες του πίνακα**, αλλά μόνο σε ένα σύνολο από αυτές**, οι οποίες έχουν ένα κοινό χαρακτηριστικό, Ας υποθέσουμε για παράδειγμα πως **για κάθε τμήμα της εταιρείας ζητούμε να ανακτήσουμε το πλήθος των υπαλλήλων που εργάζονται σε αυτό, καθώς και το μέσο όρο των μισθών τους**. Στην περίπτωση αυτή είναι προφανές πως **θα πρέπει να ομαδοποιήσουμε τις εγγραφές των υπαλλήλων με κριτήριο τον κωδικό του τμήματος στο οποίο απασχολούνται**, και στη συνέχεια να εφαρμόσουμε τις αθροιστικές συναρτήσεις, σε κάθε μια από αυτές τις ομάδες. Το σύνολο των πεδίων ως προς τα οποία θα λάβει χώρα αυτή η ομαδοποίηση **(grouping attributes)** πρέπει υποχρεωτικά να είναι ανάμεσα στα πεδία που θα επιστρέψει η **SELECT** έτσι ώστε να γνωρίζουμε σε ποιο πεδίο αναφέρεται η κάθε μια από τις εγγραφές που θα εμφανιστούν στο τελικό αποτέλεσμα.

Χαρακτηριστικό παράδειγμα χρήσης της παραμέτρου **GROUP BY** είναι το ερώτημα

```
SELECT    DNO, COUNT (*), AVG (SALARY)
FROM      EMPLOYEE
GROUP BY  DNO
```

που για κάθε τμήμα της εταιρείας, επιστρέφει **τον κωδικό του, το πλήθος των υπαλλήλων που εργάζονται σε αυτό, και το μέσο όρο των μισθών τους**.

Είναι σημαντικό να αναφερθεί στο σημείο αυτό, πως σε περιπτώσεις κατά τις οποίες τα δεδομένα που θέλουμε να ανακτήσουμε ανήκουν σε δύο ή περισσότερους πίνακες, η παράμετρος **GROUP BY** θα κληθεί μετά την πράξη της σύζευξης των δύο πινάκων. Ας θεωρήσουμε για παράδειγμα το ερώτημα

```
SELECT    PNUMBER, PNAME, COUNT (*)
FROM      PROJECTS, WORKS_ON
WHERE     PNUMBER = PNO
GROUP BY  PNUMBER, PNAME
```

που για κάθε **PROJECT** της εταιρείας, επιστρέφει **τον κωδικό του, το όνομά του, καθώς και το πλήθος των υπαλλήλων που εργάζονται σε αυτό**. Επειδή στην περίπτωση αυτή θα λάβει χώρα συσχέτιση πληροφορίας από δύο διαφορετικούς πίνακες, θα πρέπει να χρησιμοποιηθεί η πράξη της **σύζευξης (join)** που στην προκειμένη περίπτωση θα συσχετίσει το πεδίο **PNUMBER** του πίνακα **PROJECT**, με το πεδίο **PNO** του πίνακα **WORKS_ON**. Έτσι, η ομαδοποίηση των εγγραφών με βάση **τον κωδικό και το όνομα του PROJECT**, θα πρέπει να εφαρμοσθεί πάνω στο αποτέλεσμα της σύζευξης των δύο πινάκων, κάτι που σημαίνει πως η παράμετρος **GROUP BY** θα χρησιμοποιηθεί τελευταία.

Επιπλέον, είναι πιθανό **να μην επιθυμούμε την εφαρμογή των αθροιστικών συναρτήσεων, πάνω σε όλες τις ομάδες εγγραφών που οποίες προκύπτουν δια της χρήσης της παραμέτρου GROUP BY, αλλά μόνο πάνω σε εκείνες που πληρούν κάποια συγκεκριμένη συνθήκη**. Στην περίπτωση κατά την οποία είμαστε υποχρεωμένοι να διαλέξουμε αυτές τις ομάδες, μπορούμε να χρησιμοποιήσουμε την παράμετρο **HAVING** συνοδευόμενη από κάποια **συνθήκη επιλογής**. Ας υποθέσουμε για παράδειγμα ότι ζητούμε το όνομα και τον κωδικό του **PROJECT** καθώς και το πλήθος των υπαλλήλων που εργάζονται σε αυτό, όχι όμως για όλα τα **PROJECTS**, αλλά **μόνο για εκείνα στα οποία εργάζονται περισσότεροι από δύο υπάλληλοι**. Στην περίπτωση αυτή θα επαναδιατυπώσουμε το προηγούμενο ερώτημα, ως εξής :

```
SELECT    PNUMBER, PNAME, COUNT (*)
FROM      PROJECTS, WORKS_ON
WHERE     PNUMBER = PNO
GROUP BY  PNUMBER, PNAME
HAVING    COUNT (*) > 2.
```

Η διαφορά ανάμεσα στα τελευταία δύο ερωτήματα, έχει να κάνει **με το πλήθος των ομάδων εγγραφών που θα περιλαμβάνοντα στο τελικό αποτέλεσμα**. Και

στις δύο περιπτώσεις, θα λάβει χώρα **η σύζευξη ανάμεσα στους πίνακες, καθώς και η ομαδοποίηση των πλειάδων του ενδιάμεσου πίνακα που θα προκύψει από τη αυτή τη σύζευξη, με κριτήριο τον κωδικό και το όνομα του PROJECT**. Η πρόταση **SELECT** όμως δεν θα επιστρέψει όλες τις εγγραφές αλλά μόνο εκείνες που αντιστοιχούν σε πλήθος υπαλλήλων μεγαλύτερο του **2**. Επομένως η παράμετρος **HAVING** επιτρέπει την επιλογή συγκεκριμένων ομάδων εγγραφών.

Τέλος, ιδιαίτερη προσοχή απαιτείται σε περιπτώσεις κατά τις οποίες υπάρχουν **δύο συνθήκες επιλογής**, μια στην πρόταση **SELECT**, και μια στην παράμετρο **HAVING**. Έστω για παράδειγμα ότι για κάθε τμήμα, **ζητούμε το πλήθος των υπαλλήλων που εργάζονται σε αυτό και έχουν μισθό μεγαλύτερο από 40000, αλλά την πληροφορία αυτή τη θέλουμε μόνο για εκείνα τα τμήματα στα οποία εργάζονται περισσότεροι από 5 υπάλληλοι**. Στην περίπτωση αυτή έχουμε δύο συνθήκες – **τη συνθήκη SALARY > 40000** που αναφέρεται στους μισθούς των υπαλλήλων, και τη συνθήκη **COUNT (*) > 5** – που αναφέρεται στο πλήθος των υπαλλήλων που δουλεύουν σε κάθε τμήμα. Η σωστή απάντηση σε αυτό το ερώτημα έχει τη μορφή

```
SELECT    DNAME, COUNT (*)
FROM      DEPARTMENT, EMPLOYEE
WHERE     DNUMBER = DNO  AND
          SALARY > 40000    AND
          DNO  IN  (
                SELECT    DNO
                FROM              EMPLOYEE
                GROUP BY  DNO
                HAVING    COUNT (*) > 5  )
GROUP BY  DNAME
```

Συγκρίσεις ανάμεσα σε συμβολοσειρές, αριθμητικοί τελεστές, και ταξινόμηση εγγραφών: Σε ορισμένα συστήματα διαχείρισης βάσεων δεδομένων, υπάρχει η δυνατότητα να καθορίσουμε κριτήρια αναζήτησης στην πρόταση **WHERE** που να περιλαμβάνουν **συγκρίσεις ανάμεσα σε συμβολοσειρές**. Μια **συμβολοσειρά** ορίζεται **ως ένα σύνολο χαρακτήρων** – όπως είναι για παράδειγμα ένα επώνυμο ή μια διεύθυνση. Σε μια εντολή **SELECT**, η πρόταση **WHERE** επιτρέπει τη σύγκριση ανάμεσα σε τμήματα συμβολοσειρών δια της χρήσης της παραμέτρου **LIKE**. Η **LIKE** δέχεται ως όρισμα ένα τμήμα συμβολοσειράς, και το διαβιβάζει στην πρόταση **WHERE** η οποία συγκρίνει αυτό το τμήμα με την τιμή του αντίστοιχου πεδίου, και επιστρέφει μόνο εκείνες τις εγγραφές, οι οποίες ικανοποιούν τη συνθήκη επιλογής.

Επειδή η **LIKE** συγκρίνει **τμήματα συμβολοσειρών**, θα πρέπει με κάποιο τρόπο να καθορίσουμε, **σε ποιο τμήμα της συμβολοσειράς αναφέρεται το όρισμά της**. Για να το κάνουμε αυτό, μπορούμε να χρησιμοποιήσουμε τα ειδικά σύμβολα «**%**» και «_» εκ των οποίων το πρώτο αναφέρεται **σε ένα αυθαίρετο πλήθος χαρακτήρων**, ενώ το δεύτερο **σε ένα απλό χαρακτήρα**. Εάν για παράδειγμα θέλουμε να ανακτήσουμε όλους τους υπαλλήλους της εταιρείας των οποίων η διεύθυνση περιλαμβάνει τη λέξη **Houston**, μπορούμε να χρησιμοποιήσουμε την εντολή

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
WHERE     ADDRESS LIKE '%HOUSTON%'
```

Η εντολή αυτή επιστρέφει όλους τους υπαλλήλους της εταιρείας, η διεύθυνση των οποίων περιέχει τη συμβολοσειρά «**HOUSTON**». Τα σύμβολα **«%»** που υπάρχουν πριν και μετά τη συμβολοσειρά, υποδηλώνουν πως πριν και μετά τη λέξη **HOUSTON** μπορεί να βρίσκεται ένα αυθαίρετο πλήθος χαρακτήρων – το μόνο πράγμα που ενδιαφέρει την εντολή είναι η ύπαρξη ή όχι της συμβολοσειράς **HOUSTON** μέσα στην τιμή της διεύθυνσης για κάθε υπάλληλο.

Σε περιπτώσεις κατά τις οποίες θέλουμε να αναφερθούμε σε συγκεκριμένο χαρακτήρα μέσα στη συμβολοσειρά, μπορούμε να χρησιμοποιήσουμε το σύμβολο «_». Ας υποθέσουμε για παράδειγμα πως θέλουμε να ανακτήσουμε **όλους τους εργαζόμενους που έχουν γεννηθεί τη δεκαετία του 1950**. Εάν λάβουμε υπ΄ όψιν το φορμαλισμό που χρησιμοποιούμε για να καταχωρήσουμε τις ημερομηνίες – μια τυπική ημερομηνία είναι π.χ. η **12/6/1957** – δεν είναι δύσκολο να διαπιστώσουμε πως οι εργαζόμενοι που έχουν γεννηθεί το παραπάνω χρονικό διάστημα, **θα έχουν ως όγδοο ψηφίο στην τιμή της ημερομηνίας γέννησής τους, τον αριθμό 5**. Επομένως μπορούμε να επαναδιατυπώσουμε το παραπάνω ερώτημα, ζητώντας από τη βάση να μας επιστρέψει όλους τους εργαζόμενους, **η ημερομηνία γέννησης των οποίων, έχει οποιαδήποτε τιμή σε όλα τα ψηφία της, με εξαίρεση το όγδοο ψηφίο, το οποίο θα πρέπει υποχρεωτικά να έχει την τιμή 5**. Η εντολή που περιγράφει αυτό το ερώτημα, έχει τη μορφή

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     BDATE LIKE '_ _ _ _ _ _ _ 5 _'
```

Με άλλα λόγια το σύμβολο «_» αντικαθιστά **ένα χαρακτήρα κάθε φορά**, ενώ το σύμβολο «**%**», αντικαθιστά **ένα σύνολο χαρακτήρων, αυθαίρετου μεγέθους**.

Ένα άλλο ενδιαφέρον χαρακτηριστικό που συναντάται σε πολλές εκδόσεις της γλώσσας χειρισμού δεδομένων, είναι η **δυνατότητα χρήσης αριθμητικών τελεστών, μέσα στην πρόταση SELECT**. Τυπικό παράδειγμα μιας τέτοιας περίπτωσης είναι το ερώτημα

```
SELECT    FNAME, LNAME, 1.1*SALARY
FROM      EMPLOYEE, WORKS_ON, PROJECT
WHERE     SSN = ESSN AND PNO = PNUMBER AND
          PNAME = 'ProductX'
```

που για όλους τους υπαλλήλους που εργάζονται στο **PROJECT** με όνομα **ProductX**, επιστρέφει **το όνομά τους, το επώνυμό τους, και το μισθό τους αυξημένο κατά 10%**. Με τον ίδιο τρόπο μπορούμε να χρησιμοποιήσουμε και τους αριθμητικούς τελεστές «+», «–» και «/».

Τέλος, εάν το επιθυμούμε, μπορούμε να εμφανίσουμε τις εγγραφές των πινάκων που επιστρέφονται από ένα ερώτημα, όχι με τη σειρά με την οποία είναι καταχω-

ρημένες στους πίνακες της βάσης, αλλά **ταξινομημένες, κατά αύξουσα ή κατά φθίνουσα ταξινόμηση**. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε την παράμετρο **ORDER BY** συνοδευόμενη από τα πεδία ως προς τα οποία θέλουμε να λάβει χώρα αυτή η ταξινόμηση. Εάν δεν καθορίσουμε κάποιο είδος ταξινόμησης, τα αποτελέσματα του ερωτήματος θα ταξινομηθούν κατά **αύξουσα σειρά**. Έτσι η εντολή

```
SELECT    DNAME, LNAME, FNAME, PNAME
FROM      DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE     DNUMBER = DNO AND SSN = ESSN AND
          PNO = PNUMBER
ORDER BY  DNAME, LNAME, FNAME
```

θα επιστρέψει **τα ονόματα των υπαλλήλων, των τμημάτων στα οποία εργάζονται, και των έργων στα οποία απασχολούνται, ταξινομημένων κατά αύξουσα σειρά**, ως προς **το όνομα του τμήματος, το επώνυμο του υπαλλήλου, και το όνομα του υπαλλήλου**. Αντίθετα εάν χρησιμοποιήσουμε την **ORDER BY** με τη μορφή

```
ORDER BY  DNAME DESC, LNAME ASC, FNAME ASC
```

τότε η ταξινόμηση ως προς **το όνομα του τμήματος** θα γίνει κατά **φθίνουσα σειρά**, ενώ η ταξινόμηση ως προς **το επώνυμο και το όνομα του υπαλλήλου**, θα γίνει κατά **αύξουσα σειρά**.