

**MINISTRY of EDUCATION And SCIENCES of  
REPUBLIC KAZAKHSTAN  
KAZAKH NATIONAL PEDAGOGICAL UNIVERSITY  
after ABAY**

**Physics and Mathematics faculty**

**EDUCATIONAL AND METHODICAL COMPLEX FOR  
STUDENTS OF THE DISCIPLINE**

**«Database and information system»**

**for the students of the specialty  
5B11100 – «Computer science»**

**Almaty, 2012**

### **The author**

**Khalikova G.Z.**, candidate of pedagogical science., docent, professor of Kazakh national pedagogical university named after Abai

Educational and methodical complex for students of the discipline « Database and information system»

for the students of the specialty 5B11100 – «Computer science». – Almaty:KazNPU named after Abay, 2012. – p.114

Educational and methodical complex for students of the discipline is formed on the basis of:

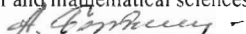
- State general compulsory standard of Higher professional education on the specialty 050111 – «Computer science»;
- Schooling plan on specialty 050111 – «Computer science», approved « \_\_\_\_ » \_\_\_\_.

## Contents

| №  | Title of the document                                    | Page |
|----|--|------|
| 1. | Syllabus discipline for students                         | 4    |
| 2. | Thesis of the lectures                                   | 7    |
| 3. | Laboratory sessions                                      | 62   |
| 4. | Themes for tutorial lessons                              | 106  |
| 5. | Self-control   | 107  |
| 6. | Tasks for self-control and preparing for the examination | 110  |
| 7. | Literature   | 112  |
| 8. | Glossary   | 113  |

## 1. Syllabus discipline for students

### 1. Information about discipline

|   |  |  |                             |
|---|--|--|-----------------------------|
| Title of discipline<br>«Database and information<br>system» | discipline code  | Quantity of credits<br><b>3</b>                                  | Course - 2,<br>semester - 3 |
| Title of speciality<br><b>Computer Science</b>              | Speciality code<br><b>5B011100</b>   | Chair<br><b>Computer Science<br/>and applied<br/>mathematics</b> | Faculty<br><b>FMF</b>       |
| Form of the education<br><b>Day department</b>              |  | Teaching Language<br><b>English</b>                              |                             |
| Lecture   | Time and room – according to the schedule  |  |                             |
| Laboratory sessions   |  |  |                             |
| Independent work<br>student with<br>supervisor              |  |  |                             |
| Consultation according to the schedule                      |  |  |                             |
| Rating control schedule                                     |  |  |                             |
| Professor Khalikova G.Z.                                    | Job telephone: 2611576, Email: <a href="mailto:xgulira@rambler.ru">xgulira@rambler.ru</a>  |  |                             |
|   | Chair holder of computer science and applied mathematics,<br>doctor of the physical and mathematical sciences, professor<br>Berdyshev A.S.  |  |                             |

### 2. Description of the discipline

Databases lie at the heart of modern commercial application development. Their use extends beyond this to many applications and environments where large amounts of data are stored for efficient update and retrieval. Their principles and fundamental techniques are being extended today to the Web and to novel kinds of data, like XML.

**The purpose of this course** is to provide an introduction to the principles of database systems; database design, covering the entity relationship model; the basic principle of a database, information systems, the database management system conceptual concept formation; mathematical models describing the database, interpret the principle of database design also consider the basic technology which intended for databases.

#### **The main objectives:**

to know the values of the main constituent databases and database management system, show implemented methods the corresponding values in the real system programs, know the theoretical foundations of database and direct the process of designing.

#### **Student should know:**

- database design, covering the entity relationship model;
- to cover the relational data model, relational algebra and SQL;
- to show how database requirements are captured using entity-relationship models, and the definition of databases and their queries in the relational model;
- Relational algebra and calculus, as are the normal forms for expressing relationships subject to dependencies;
- The SQL query language;

- The primary data structures and strategies for efficient querying of a database.

At the end of the course the student will be **able** to:

- Understand the relational model, and know how to translate requirements captured in an Entity-Relationship diagram into a relational schema;
- Be able to reason about dependencies in a relational schema;
- Understand normal form schemas, and the decomposition process by which normal forms are obtained;
- Use relational query languages such as SQL;
- Understand equivalences within relational algebra and their use in query optimization;
- Understand indexing and its role in query plans;
- Be able to create database-driven web interfaces.

### 3. Prior requisite of discipline:

- informatics,
- programming language,
- discrete mathematics;
- Information and communication technologies in education.

### 4. Post requisite of discipline:

- computer systems and telecommunications;
- computer architecture;
- theory of language programming and translation methods;
- Theoretical bases of protection of information.

## 5 Calendar-themed background.

| №  | Name those disciplines  | week | Classroom       |                         |   | TWS(hour) | In all(hour.) |
|----|---|------|-----------------|-------------------------|---|-----------|---------------|
|    |   |      | lecture (hour.) | sem./lab./classes(hour) |   |           |               |
|    | 1. Database and Information system  |      |                 |                         |   |           |               |
| 1  | Introduction to database. Characteristics of Databases terminology.   | 1    | 2               | 1                       | 3 | 3         | 9             |
| 2  | Information systems architecture<br>File server system. Database server. Local information systems. Remoted information systems | 2    | 2               | 1                       | 3 | 3         | 9             |
| 3  | Database management system  | 3    | 2               | 1                       | 3 | 3         | 9             |
| 4  | Modeling database   | 4    | 4               | 1                       | 5 | 5         | 15            |
| 5  | Data model  | 5,6  | 4               | 1                       | 5 | 5         | 15            |
| 6  | Relational model. Relational algebra, relational account. Normal form   | 7,8  | 2               | 1                       | 3 | 3         | 9             |
| 7  | Information systems in networks   | 9    | 2               | 1                       | 3 | 3         | 9             |
| 8  | Usage database  | 10   | 2               | 1                       | 3 | 3         | 9             |
|    | 2. The modern technology to create database   |      |                 |                         |   |           |               |
| 9  | Analyze the current technology which materialize database. Standards and languages. (MS SQL)                                    | 11   | 4               | 2                       | 7 | 7         | 20            |
| 10 | MS Access Database Management system  | 12   | 2               | 2                       | 3 | 3         | 10            |

|    |  |           |    |    |    |    |     |
|----|--|-----------|----|----|----|----|-----|
| 11 | Object-oriented approach. Create databases in the Delphi environment | 13,14, 15 | 4  | 3  | 7  | 7  | 21  |
|    | Total:   |           | 30 | 15 | 45 | 45 | 135 |

**laboratory sessions**

| №№<br>п.п. | Theme and content of laboratory classes   | Equipment for<br>laboratory classes     | Hours |
|------------|---|---|-------|
| 1          | Create a table in the MS Access Database management system. Establish a link between the tables         | MS Access Database<br>management system | 2     |
| 2          | Create query in the MS Access Database management system  |   | 3     |
| 3          | Create forms in the MS Access Database management system  |   | 3     |
| 4          | create a report in the MS Access Database management system   |   | 3     |
| 5          | create a macros and sheets in the MS Access Database management system                                  |   | 3     |
| 6          | Create databases in the Delphi environment  | Delphi 7.0                              | 2     |
| 7          | Create table. Create the table structure by programming. Table component                                |   | 2     |
| 8          | Way of removing the table in the form   |   | 2     |
| 9          | Create query in Delphi environment. SQL language  |   | 3     |
| 10         | Create a query by using SQL Builder   |   | 2     |
| 11         | Create report in Delphi environment. Create a simple report   |   | 2     |
| 12         | Create report in Delphi environment. To show auxiliary and serial informations use TQRSysData component |   | 3     |
|            | Total:  |   | 30    |

## 6. literature

### The main literature:

1. C.J. Date. An Introduction to Database Systems. 8th Edition. Addison-Wesley, 2000.-1072 p.
2. G. Hansen, Hansen, J. // Databases: Design and Management. Translation, Moscow 1999.
3. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г.// Базы данных. Учебное пособие, 2002.
4. Халыкова К.З. //Мәліметтер қоры және банкі. Оқу құралы. 2004 ж. -160 б.
5. Stojanovic, T.A. (2007) Guidelines for Implementing Local Information Systems at the Coast.COREPOINT and Cardiff University, Cardiff. <http://corepoint.ucc.ie/Cpages/outputs.htm>

### Additional literature:

1. Джеймс Р. Грофф, Пол Н. Вайнберг SQL - полное руководство. Перевод, Киев 1999г.
2. Наумова М.А. Системы управления базами данных и знаний. Высшая школа, 1992г.
3. Диго С.М. Проектирование баз данных. Финансы и статистика, 1990г.
4. Мейс С. Использование языка SQL обеспечивает возможность объединения баз данных.В мире компьютеров, № 2, 1988.

5. Дж.Мартин Организация баз данных в вычислительных системах. М, Наука 1980г.
6. Фаронов В.В. Программирование баз данных на Delphi 7. Учебный курс, 2005.
7. Золотова С.И. Практикум по Access. Финансы и статистика, Москва, 2000.

## 7. Assessment criteria

| №            | Term constituents                        | Mark (max points) | Quantity | Subtotal   |
|--------------|--|-------------------|----------|------------|
| 1            | Attendance, work at the lesson, homework | 2                 | 15       | 30         |
| 2            | Laboratory session                       | 30                | 2        | 60         |
| 3            | Colloquium                               | 20                | 2        | 40         |
| 3            | Tasks for self-control                   | 35                | 2        | 70         |
| 4            | Examination                              | 100               | 1        | 100        |
| <b>Total</b> |  |                   |          | <b>300</b> |

## 8. Teacher's requirements

- Not to be late for sessions
- Not to deal with other deals
- To disconnect cellular telephone
- Not to miss sessions without good excuse
- To retake missed sessions
- Actively participate in the study process
- To be support feedback constructively
- To observe the time and be obligatory

## 2. Thesis of the lectures

### Lesson1.

*Theme of lecture:* Introduction to database. Characteristics of Databases terminology.

*Plan:*

*Description the importance of data and databases.*

*Database concept*

*Understanding Database Terminology*

A **database** is an organized collection of data. The data are typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies).

The term *database* is correctly applied to the data and their supporting data structures, and not to the database management system (DBMS). The database data collection with DBMS is called a database system.

The term *database system* implies that the data are managed to some level of quality (measured in terms of accuracy, availability, usability, and resilience) and this in turn often implies the use of a general-purpose database management system (DBMS). A general-purpose DBMS is typically a complex software system that meets many usage requirements to properly maintain its databases which are often large and complex. The utilization of databases is now so widespread that virtually every technology and product relies on databases and DBMSs for its development and commercialization, or even may have DBMS software embedded in it. Also, organizations and companies, from small to large, depend heavily on databases for their operations.

Well known DBMSs include FoxPro, IBM DB2, Lintier, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, PostgreSQL and SQLite. A database is not generally portable across different DBMS, but different DBMSs can inter-operate to some degree by using standards like SQL and ODBC together to support a single application built over more than one database. A DBMS also needs to provide effective run-time execution to properly support (e.g., in terms of performance, availability, and security) as many database end-users as needed.

A way to classify databases involves the type of their contents, for example: bibliographic, document-text, statistical, or multimedia objects. Another way is by their application area, for example: accounting, music compositions, movies, banking, manufacturing, or insurance.

### Database concept

The database concept has evolved since the 1960s to ease increasing difficulties in designing, building, and maintaining complex information systems (typically with many concurrent end-users, and with a large amount of diverse data). It has evolved together with database management systems which enable the effective handling of databases. Though the terms database and DBMS define different entities, they are inseparable: a database's properties are determined by its supporting DBMS. The Oxford English dictionary cites<sup>[citation needed]</sup> a 1962 technical report as the first to use the term "data-base." With the progress in technology in the areas of processors, computer memory, computer storage and computer networks, the sizes, capabilities, and performance of databases and their respective DBMSs have grown in orders of magnitudes. For decades it has been unlikely that a complex information system can be built effectively without a proper database supported by a DBMS. The utilization of databases is now spread to such a wide degree that virtually every technology and



product relies on databases and DBMSs for its development and commercialization, or even may have such embedded in it. Also, organizations and companies, from small to large, heavily depend on databases for their operations.

No widely accepted exact definition exists for DBMS. However, a system needs to provide considerable functionality to qualify as a DBMS. Accordingly its supported data collection needs to meet respective usability requirements (broadly defined by the requirements below) to qualify as a database. Thus, a database and its supporting DBMS are defined here by a set of general requirements listed below. Virtually all existing mature DBMS products meet these requirements to a great extent, while less mature either meet them or converge to meet them.

### **Understanding Database Terminology**

A computer cannot process data unless it is organized in special ways; into characters, fields, records, files and databases.

A character is the most basic element of data that can be observed and manipulated. Behind it are the invisible data elements we call bits and bytes, referring to physical storage elements used by the computer hardware. A character is a single symbol such as a digit, letter, or other special character (e.g., \$, #, and ?).

#### *Field*

A field contains an item of data; that is, a character, or group of characters that are related. For instance, a grouping of related text characters such as "John Smith" makes up a name in the name field. Let's look at another example. Suppose a political action group advocating gun control in Pennsylvania is compiling the names and addresses of potential supporters for their new mailing list. For each person, they must identify the name, address, city, state, zip code and telephone number. A field would be established for each type of information in the list. The name field would contain all of the letters of the first and last name. The zip code field would hold all of the digits of a person's zip code, and so on. In summary, a field may contain an attribute (e.g., employee salary) or the name of an entity (e.g., person, place, or event).

#### *Record*

A record is composed of a group of related fields. As another way of saying it, a record contains a collection of attributes related to an entity such as a person or product. Looking at the list of potential gun control supporters, the name, address, zip code and telephone number of a single individual would constitute a record. A payroll record would contain the name, address, social security number, and title of each employee.

#### *Database File*

As we move up the ladder, a database file is defined as a collection of related records. A database file is sometimes called a table. A file may be composed of a complete list of individuals on a mailing list, including their addresses and telephone numbers. Files are frequently categorized by the purpose or application for which

they are intended. Some common examples include mailing lists, quality control files, inventory files, or document files. Files may also be classified by the degree of permanence they have. Transition files are only temporary, while master files are much more long-lived.

#### *Database*

Organizations and individuals use databases to bring independent sources of data together and store them electronically. Thus, a database is composed of related files that are consolidated, organized and stored together. One collection of related files might pertain to employee information. Another collection of related files might contain sports statistics.

Organizations and individuals may have and use many different databases, depending on the nature of the work involved. For example, a library database might consist of several related, but separate, databases including book titles and author names, book description, books on order, books checked out, and similar sets of information. Most organizations have product information databases, customer databases, and human resource databases that contain information about employees, salaries, home address, stock purchase plans, and tax deduction information. In each case, the data stored in a database is independent from the application programs which use and process the data.

#### *Data Management System*

Data management systems are used to access and manipulate data in a database. A database management system is a software package that enables users to edit, link, and update files as needs dictate. Database management systems will be discussed in greater detail in another lesson.

#### *Key*

In order to track and analyze data effectively, each record requires a unique identifier or what is called a key. The key must be completely unique to a particular record just as each individual has a unique social security number assigned to them. In fact, social security numbers are often used as keys in large databases. You might think that the name field would be a good choice for a key in a mailing list. However, this would not be a good choice because some people might have the same name. A key must be identified or assigned to each record for computerized information processing to function correctly. An existing field may be used if the entries are entirely unique, such as a social security or telephone number. In most cases, a new field will be developed to hold a key, such as a customer number or product number.

## **Lesson 2.**

*Theme of lecture:* Information systems architecture

#### *Plan:*

*File server system.*

*Database server.*

*Local information systems.*

*Remoted information systems*

Today's database professionals face several options when considering architectures in which to employ to address the various needs of their employers and/or clients. The following text will provide an overview of three main categories of database architectures and their sub-categories, as well as offer some insight into the benefits of each.

### **Application Logic**

Database architectures can be distinguished by examining the way application logic is distributed throughout the system. Application logic consists of three components: Presentation Logic, Processing Logic, and Storage Logic.

The presentation logic component is responsible for formatting and presenting data on the user's screen. The processing logic component handles data processing logic, business rules logic, and data management logic. Finally, the storage logic component is responsible for the storage and retrieval from actual devices such as a hard drive or RAM.

By determining which tier(s) these components are processed on we can get a good idea of what type of architecture and subtype we are dealing with.

#### *One Tier Architectures*

Imagine a person on a desktop computer who uses Microsoft Access to load up a list of personal addresses and phone numbers that he or she has saved in MS Windows' "My Documents" folder. This is an example of a one-tier database architecture. The program (Microsoft Access) runs on the user's local machine, and references a file that is stored on that machine's hard drive, thus using a single physical resource to access and process information.

Another example of a one-tier architecture is a file server architecture. In this scenario, a workgroup database is stored in a shared location on a single machine. Workgroup members use a software package such as Microsoft Access to load the data and then process it on their local machine. In this case, the data may be shared among different users, but all of the processing occurs on the local machine. Essentially, the file-server is just an extra hard drive from which to retrieve files.

Yet another way one-tier architectures have appeared is in that of mainframe computing. In this outdated system, large machines provide directly connected unintelligent terminals with the means necessary to access, view and manipulate data. Even though this is considered a client-server system, since all of the processing power (for both data and applications) occurs on a single machine, we have a one-tier architecture.

One-tier architectures can be beneficial when we are dealing with data that is relevant to a single user (or small number of users) and we have a relatively small amount of data. They are somewhat inexpensive to deploy and maintain.

#### *Two Tier Client/Server Architectures*

A two-tier architecture is one that is familiar to many of today's computer users. A common implementation of this type of system is that of a Microsoft Windows based client program that accesses a server database such as Oracle or SQL Server.

Users interact through a GUI (Graphical User Interface) to communicate with the database server across a network via SQL (Structured Query Language).

In two-tier architectures it is important to note that two configurations exist. A thin-client (fat-server) configuration exists when most of the processing occurs on the server tier. Conversely, a fat-client (thin-server) configuration exists when most of the processing occurs on the client machine.

Another example of a two-tier architecture can be seen in web-based database applications. In this case, users interact with the database through applications that are hosted on a web-server and displayed through a web-browser such as Internet Explorer. The web server processes the web application, which can be written in a language such as PHP or ASP. The web app connects to a database server to pass along SQL statements which in turn are used to access, view, and modify data. The DB server then passes back the requested data which is then formatted by the web server for the user.

Although this appears to be a three-tier system because of the number of machines required to complete the process, it is not. The web-server does not normally house any of the business rules and therefore should be considered part of the client tier in partnership with the web-browser.

Two-tier architectures can prove to be beneficial when we have a relatively small number of users on the system (100-150) and we desire an increased level of scalability.

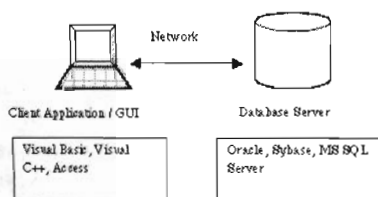


Figure 1. Two-Tier Client-Server Architecture

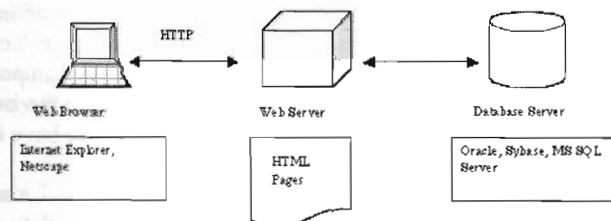


Figure 2. Web-Based, Two-Tier Client-Server Architecture  
N-Tier Client/Server Architectures

Most n-tier database architectures exist in a three-tier configuration. In this architecture the client/server model expands to include a middle tier (business tier),

which is an application server that houses the business logic. This middle tier relieves the client application(s) and database server of some of their processing duties by translating client calls into database queries and translating data from the database into client data in return. Consequently, the client and server never talk directly to one-another.

A variation of the n-tier architecture is the web-based n-tier application. These systems combine the scalability benefits of n-tier client/server systems with the rich user interface of web-based systems.

Because the middle tier in a three-tier architecture contains the business logic, there is greatly increased scalability and isolation of the business logic, as well as added flexibility in the choice of database vendors.

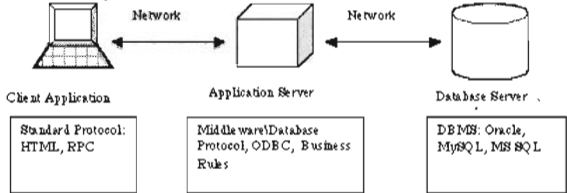


Figure 3. Three-Tier Client-Server Architecture

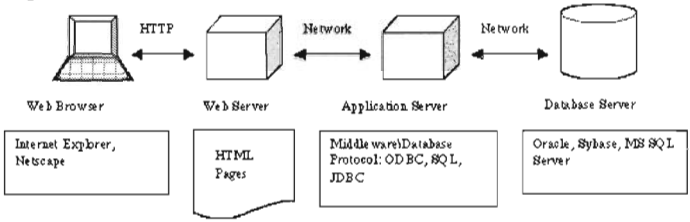


Figure 4. Web-Based, Three-Tier Client Server Architecture

**File server**

In computing, a file server is a computer attached to a network that has the primary purpose of providing a location for shared disk access, i.e. shared storage of computer files (such as documents, sound files, photographs, movies, images, databases, etc.) that can be accessed by the workstations that are attached to the same computer network. The term server highlights the role of the machine in the client-server scheme, where the clients are the workstations using the storage. A file server is not intended to perform computational tasks, and does not run programs on behalf of its clients. It is designed primarily to enable the storage and retrieval of data while the computation is carried out by the workstations.

File servers are commonly found in schools and offices, where users use a LAN to connect their client computers.

**Types of file servers**

A file server may be dedicated or non-dedicated. A dedicated server is designed specifically for use as a file server, with workstations attached for reading and writing files and databases.

File servers may also be categorized by the method of access: Internet file servers are frequently accessed by File Transfer Protocol (FTP) or by HTTP (but are different from web servers, that often provide dynamic web content in addition to static files). Servers on a LAN are usually accessed by SMB/CIFS protocol (Windows and Unix-like) or NFS protocol (Unix-like systems).

Database servers, that provide access to a shared database via a database device driver, are not regarded as file servers.

### **Design of file servers**

In modern businesses the design of file servers is complicated by competing demands for storage space, access speed, recoverability, ease of administration, security, and budget. This is further complicated by a constantly changing environment, where new hardware and technology rapidly obsolesces old equipment, and yet must seamlessly come online in a fashion compatible with the older machinery. To manage throughput, peak loads, and response time, vendors may utilize queuing theory[1] to model how the combination of hardware and software will respond over various levels of demand. Servers may also employ dynamic load balancing scheme to distribute requests across various pieces of hardware.

The primary piece of hardware equipment for servers over the last couple of decades has proven to be the hard disk drive. Although other forms of storage are viable (such as magnetic tape and solid-state drives) disk drives have continued to offer the best fit for cost, performance, and capacity.

### **Storage**

Since the crucial function of a file server is storage, technology has been developed to operate multiple disk drives together as a team, forming a disk array. A disk array typically has cache (temporary memory storage that is faster than the magnetic disks), as well as advanced functions like RAID and storage virtualization. Typically disk arrays increase level of availability by using redundant components other than RAID, such as power supplies. Disk arrays may be consolidated or virtualized in a [storage area network] (SAN).

### **Network-attached storage**

Network-attached storage (NAS) is file-level computer data storage connected to a computer network providing data access to heterogeneous clients. NAS devices specifically are distinguished from file servers generally in a NAS being a computer appliance – a specialized computer built from the ground up for serving files – rather than a general purpose computer being used for serving files (possibly with other functions). In discussions of NASs, the term "file server" generally stands for a contrasting term, referring to general purpose computers only.

As of 2010 NAS devices are gaining popularity, offering a convenient method for sharing files between multiple computers.[2] Potential benefits of network-

attached storage, compared to non-dedicated file servers, include faster data access, easier administration, and simple configuration.[3]

NAS systems are networked appliances containing one or more hard drives, often arranged into logical, redundant storage containers or RAID arrays. Network Attached Storage removes the responsibility of file serving from other servers on the network. They typically provide access to files using network file sharing protocols such as NFS, SMB/CIFS (Server Message Block/Common Internet File System), or AFP.

### **Security**

File servers generally offer some form of system security to limit access to files to specific users or groups. In large organizations, this is a task usually delegated to what is known as directory services such as openLDAP, Novell's eDirectory or Microsoft's Active Directory.

These servers work within the hierarchical computing environment which treat users, computers, applications and files as distinct but related entities on the network and grant access based on user or group credentials. In many cases, the directory service spans many file servers, potentially hundreds for large organizations. In the past, and in smaller organizations, authentication can take place directly to the server itself.

### **File Transfer Protocol**

File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host or to another host over a TCP-based network, such as the Internet.

FTP is built on a client-server architecture and uses separate control and data connections between the client and the server.[1] FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that hides (encrypts) the username and password, and encrypts the content, FTP is often secured with SSL/TLS ("FTPS"). SSH File Transfer Protocol ("SFTP") is sometimes also used instead.

The first FTP client applications were command-line applications developed before operating systems had graphical user interfaces, and are still shipped with most Windows, Unix, and Linux operating systems[2][3]. Dozens of FTP clients and automation utilities have since been developed for desktops, servers, mobile devices, and hardware, and FTP has been incorporated into hundreds of productivity applications, such as web page editors

### **Communication and data transfer**

The server responds over the control connection with three-digit status codes in ASCII with an optional text message. For example "200" (or "200 OK") means that the last command was successful. The numbers represent the code for the response and the optional text represents a human-readable explanation or request (e.g. <Need account for storing file>).[1] An ongoing transfer of file data over the data connection can be aborted using an interrupt message sent over the control connection.

Illustration of starting a passive connection using port 21

FTP may run in active or passive mode, which determines how the data connection is established.[5] In active mode, the client creates a TCP control connection to the server and sends the server the client's IP address and an arbitrary client port number, and then waits until the server initiates the data connection over TCP to that client IP address and client port number.[6] In situations where the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In this mode, the client uses the control connection to send a PASV command to the server and then receives a server IP address and server port number from the server,[6][5] which the client then uses to open a data connection from an arbitrary client port to the server IP address and server port number received.[4] Both modes were updated in September 1998 to support IPv6. Further changes were introduced to the passive mode at that time, updating it to extended passive mode.[7]

### **Lesson 3.**

*Theme of lecture:* **Database management system**

*Plan:*

*An Introduction to Database Management Systems*

*Define the term database management system (DBMS).*

*Describe the basic purpose and functions of a DBMS.*

*Discuss the advantages and disadvantages of DBMSs*

A database is a collection of related files that are usually integrated, linked or cross-referenced to one another. The advantage of a database is that data and records contained in different files can be easily organized and retrieved using specialized database management software called a database management system (DBMS) or database manager.

A database management system is a set of software programs that allows users to create, edit and update data in database files, and store and retrieve data from those database files. Data in a database can be added, deleted, changed, sorted or searched all using a DBMS. If you were an employee in a large organization, the information about you would likely be stored in different files that are linked together. One file about you would pertain to your skills and abilities, another file to your income tax status, another to your home and office address and telephone number, and another to your annual performance ratings. By cross-referencing these files, someone could change a person's address in one file and it would automatically be reflected in all the other files. DBMSs are commonly used to manage:

### **DBMSs and File Management Systems**

Computerized file management systems (sometimes called file managers) are not considered true database management systems because files cannot be easily linked to each other. However, they can serve as useful data management functions by providing a system for storing information in files. For example, a file management system might be used to store a mailing list or a personal address book.



When files need to be linked, a relational database should be created using database application software such as Oracle, Microsoft Access, IBM DB2, or FileMaker Pro.

#### The Advantages of a DBMS

**Improved availability:** One of the principle advantages of a DBMS is that the same information can be made available to different users.

**Minimized redundancy:** The data in a DBMS is more concise because, as a general rule, the information in it appears just once. This reduces data redundancy, or in other words, the need to repeat the same data over and over again. Minimizing redundancy can therefore significantly reduce the cost of storing information on hard drives and other storage devices. In contrast, data fields are commonly repeated in multiple files when a file management system is used.

**Accuracy:** Accurate, consistent, and up-to-date data is a sign of data integrity. DBMSs foster data integrity because updates and changes to the data only have to be made in one place. The chances of making a mistake are higher if you are required to change the same data in several different places than if you only have to make the change in one place.

**Program and file consistency:** Using a database management system, file formats and system programs are standardized. This makes the data files easier to maintain because the same rules and guidelines apply across all types of data. The level of consistency across files and programs also makes it easier to manage data when multiple programmers are involved.

**User-friendly:** Data is easier to access and manipulate with a DBMS than without it. In most cases, DBMSs also reduce the reliance of individual users on computer specialists to meet their data needs.

**Improved security:** As stated earlier, DBMSs allow multiple users to access the same data resources. This capability is generally viewed as a benefit, but there are potential risks for the organization. Some sources of information should be protected or secured and only viewed by select individuals. Through the use of passwords, database management systems can be used to restrict data access to only those who should see it.

#### The Disadvantages of a DBMS

There are basically two major downsides to using DBMSs. One of these is cost, and the other the threat to data security.

**Cost:** Implementing a DBMS system can be expensive and time-consuming, especially in large organizations. Training requirements alone can be quite costly.

**Security:** Even with safeguards in place, it may be possible for some unauthorized users to access the database. In general, database access is an all or nothing proposition. Once an unauthorized user gets into the database, they have access to all the files, not just a few. Depending on the nature of the data involved, these breaches in security can also pose a threat to individual privacy. Steps should also be taken to regularly make backup copies of the database files and store them because of the possibility of fires and earthquakes that might destroy the system.

In this lesson, a database management system is defined, as well as its purposes and functions. One of the most powerful aspects of a DBMS is the ability to organize and retrieve data from different, but related, files. However, using databases and DBMSs has its advantages and disadvantages. As you proceed with your career, you should be aware of the tradeoffs that accompany using these computerized tools. The tradeoffs we have discussed so far include such things as the redundancy, accuracy, accessibility, and user-friendliness of data in a DBMS. Being educated about the strengths and weaknesses of DBMSs will allow you to make more effective decisions about how to organize and use data.

#### **Lesson 4.**

*Theme of lecture:* **Modeling database. Database design**

*Plan:*

*1 How are Data Models Used in Practice?*

*Conceptual data models*

*Logical data models (LDMs).*

*Physical data models (PDMs).*

#### **Database design**

Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).[1]

The process of doing database design generally consists of a number of steps which will be carried out by the database designer. Usually, the designer must:

Determine the relationships between the different data elements.

Superimpose a logical structure upon the data on the basis of these relationships  
ER Diagram (Entity-relationship model)

A sample Entity-relationship diagram

Database designs also include ER (Entity-relationship model) diagrams. An ER diagram is a diagram that helps to design databases in an efficient way.

Attributes in ER diagrams are usually modeled as an oval with the name of the attribute, linked to the entity or relationship that contains the attribute.

Within the relational model the final step can generally be broken down into two further steps, that of determining the grouping of information within the system, generally determining what are the basic objects about which information is being stored, and then determining the relationships between these groups of information, or objects. This step is not necessary with an Object database.

### **The Design Process**

1. Determine the purpose of the database - This helps prepare for the remaining steps.
2. Find and organize the information required - Gather all of the types of information to record in the database, such as product name and order number.
3. Divide the information into tables - Divide information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.
4. Turn information items into columns - Decide what information needs to be stored in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.
5. Specify primary keys - Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.
6. Set up the table relationships - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. Refine the design - Analyze the design for errors. Create tables and add a few records of sample data. Check if results come from the tables as expected. Make adjustments to the design, as needed.
8. Apply the normalization rules - Apply the data normalization rules to see if tables are structured correctly. Make adjustments to the tables

### *Determining data to be stored*

In a majority of cases, a person who is doing the design of a database is a person with expertise in the area of database design, rather than expertise in the domain from which the data to be stored is drawn e.g. financial information, biological information etc. Therefore the data to be stored in the database must be determined in cooperation

with a person who does have expertise in that domain, and who is aware of what data must be stored within the system.

This process is one which is generally considered part of requirements analysis, and requires skill on the part of the database designer to elicit the needed information from those with the domain knowledge. This is because those with the necessary domain knowledge frequently cannot express clearly what their system requirements for the database are as they are unaccustomed to thinking in terms of the discrete data elements which must be stored. Data to be stored can be determined by Requirement Specification

### **Normalization**

In the field of relational database design, normalization is a systematic way of ensuring that a database structure is suitable for general-purpose querying and free of certain undesirable characteristics—insertion, update, and deletion anomalies—that could lead to a loss of data integrity.

A standard piece of database design guidance is that the designer should create a fully normalized design; selective denormalization can subsequently be performed, but only for performance reasons. However, some modeling disciplines, such as the dimensional modeling approach to data warehouse design, explicitly recommend non-normalized designs, i.e. designs that in large part do not adhere to 3NF.

Normalization consists of normal forms that are 1NF, 2NF, 3NF, BOYCE-CODD NF (3.5NF), 4NF and 5NF.

Although methodology issues are covered later, we need to discuss how data models can be used in practice to better understand them. You are likely to see three basic styles of data model:

Conceptual data models. These models, sometimes called domain models, are typically used to explore domain concepts with project stakeholders. On Agile teams high-level conceptual models are often created as part of your initial requirements envisioning efforts as they are used to explore the high-level static business structures and concepts. On traditional teams conceptual data models are often created as the precursor to LDMs or as alternatives to LDMs.

Logical data models (LDMs). LDMs are used to explore the domain concepts, and their relationships, of your problem domain. This could be done for the scope of a single project or for your entire enterprise. LDMs depict the logical entity types, typically referred to simply as entity types, the data attributes describing those entities, and the relationships between the entities. LDMs are rarely used on Agile projects although often are on traditional projects (where they rarely seem to add much value in practice).

Physical data models (PDMs). PDMs are used to design the internal schema of a database, depicting the data tables, the data columns of those tables, and the relationships between the tables.

### **Table 1. Data Modeling - Conceptual, Logical and Physical Data Models**

| Feature              | Conceptual | Logical | Physical |
|----------------------|------------|---------|----------|
| Entity Names         | ✓          | ✓       |          |
| Entity Relationships | ✓          | ✓       |          |
| Attributes           |            | ✓       |          |
| Primary Keys         |            | ✓       | ✓        |
| Foreign Keys         |            | ✓       | ✓        |
| Table Names          |            |         | ✓        |
| Column Names         |            |         | ✓        |
| Column Data Types    |            |         | ✓        |

Although LDMs and PDMs sound very similar, and they in fact are, the level of detail that they model can be significantly different. This is because the goals for each diagram is different – you can use an LDM to explore domain concepts with your stakeholders and the PDM to define your database design. Figure 1 presents a simple LDM and Figure 2 a simple PDM, both modeling the concept of customers and addresses as well as the relationship between them. Both diagrams apply the Barker notation, summarized below. Notice how the PDM shows greater detail, including an associative table required to implement the association as well as the keys needed to maintain the relationships. More on these concepts later. PDMs should also reflect your organization’s database naming standards, in this case an abbreviation of the entity name is appended to each column name and an abbreviation for “Number” was consistently introduced. A PDM should also indicate the data types for the columns, such as integer and char(5). Although Figure 2 does not show them, lookup tables (also called reference tables or description tables) for how the address is used as well as for states and countries are implied by the attributes ADDR\_USAGE\_CODE, STATE\_CODE, and COUNTRY\_CODE.

A simple logical data model.

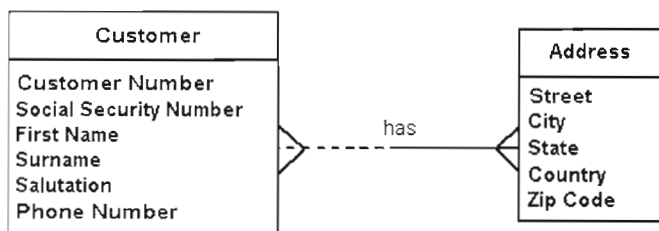


Figure 5. A simple physical data model.

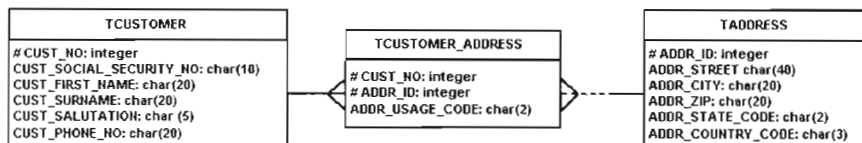


Figure 6.

An important observation about Figures 1 and 2 is that I'm not slavishly following Barker's approach to naming relationships. For example, between Customer and Address there really should be two names "Each CUSTOMER may be located in one or more ADDRESSES" and "Each ADDRESS may be the site of one or more CUSTOMERS". Although these names explicitly define the relationship I personally think that they're visual noise that clutter the diagram. I prefer simple names such as "has" and then trust my readers to interpret the name in each direction. I'll only add more information where it's needed, in this case I think that it isn't. However, a significant advantage of describing the names the way that Barker suggests is that it's a good test to see if you actually understand the relationship – if you can't name it then you likely don't understand it.

Data models can be used effectively at both the enterprise level and on projects. Enterprise architects will often create one or more high-level LDMs that depict the data structures that support your enterprise, models typically referred to as enterprise data models or enterprise information models. An enterprise data model is one of several views that your organization's enterprise architects may choose to maintain and support – other views may explore your network/hardware infrastructure, your organization structure, your software infrastructure, and your business processes (to name a few). Enterprise data models provide information that a project team can use both as a set of constraints as well as important insights into the structure of their system.

Project teams will typically create LDMs as a primary analysis artifact when their implementation environment is predominantly procedural in nature, for example they are using structured COBOL as an implementation language. LDMs are also a good choice when a project is data-oriented in nature, perhaps a data warehouse or reporting system is being developed (having said that, experience seems to show that

usage-centered approaches appear to work even better). However LDMs are often a poor choice when a project team is using object-oriented or component-based technologies because the developers would rather work with UML diagrams or when the project is not data-oriented in nature. As Agile Modeling advises, apply the right artifact(s) for the job. Or, as your grandfather likely advised you, use the right tool for the job. It's important to note that traditional approaches to Master Data Management (MDM) will often motivate the creation and maintenance of detailed LDMs, an effort that is rarely justifiable in practice when you consider the total cost of ownership (TCO) when calculating the return on investment (ROI) of those sorts of efforts.

2. *What About Conceptual Models?*

Halpin (2001) points out that many data professionals prefer to create an Object-Role Model (ORM), an example is depicted in Figure 3, instead of an LDM for a conceptual model. The advantage is that the notation is very simple, something your project stakeholders can quickly grasp, although the disadvantage is that the models become large very quickly. ORM's enable you to first explore actual data examples instead of simply jumping to a potentially incorrect abstraction – for example Figure 3 examines the relationship between customers and addresses in detail. For more information about ORM, visit [www.orm.net](http://www.orm.net).

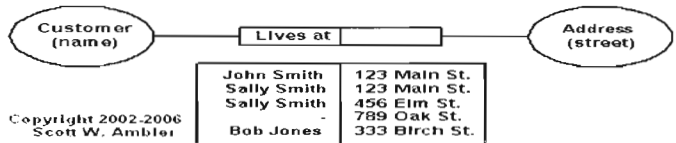


Figure 7. A simple Object-Role Model.

2.3. Common Data Modeling Notations

Figure 4 presents a summary of the syntax of four common data modeling notations: Information Engineering (IE), Barker, IDEF1X, and the Unified Modeling Language (UML). This diagram isn't meant to be comprehensive, instead its goal is to provide a basic overview. Furthermore, for the sake of brevity I wasn't able to depict the highly-detailed approach to relationship naming that Barker suggests. Although I provide a brief description of each notation in Table 1 I highly suggest David Hay's paper *A Comparison of Data Modeling Techniques* as he goes into greater detail than I do.

The IE notation (Finkelstein 1989) is simple and easy to read, and is well suited for high-level logical and enterprise data modeling. The only drawback of this notation, arguably an advantage, is that it does not support the identification of attributes of an entity. The assumption is that the attributes will be modeled with another diagram or simply described in the supporting documentation.

Lessons 5-6.

## *Theme of lecture:* **Data model**

### *Plan:*

*Hierarchical Model;*

*Network Model;*

*Relational Model;*

*Object-Oriented Model;*

*Other Database models.*

Database architecture is the fundamental schema, or physical layout, of a database. Rooted in mathematical/structural theory, the most common database architectures are: Hierarchical, Networked, Object-Oriented and Relational.

### *Hierarchical Model*

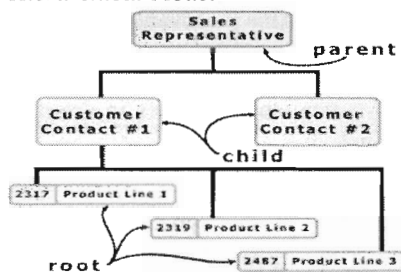


Figure 8.

The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths. For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee



segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

### Network Model

The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model. The basic data modeling construct in the network model is the set construct. A set consists of an owner record type, a set name, and a member record type. A member record type can have that role in more than one set, hence the multiparent concept is supported. An owner record type can also be a member or owner in another set. The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pairwise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1:M relationship, although 1:1 is permitted. The CODASYL network model is based on mathematical set theory.

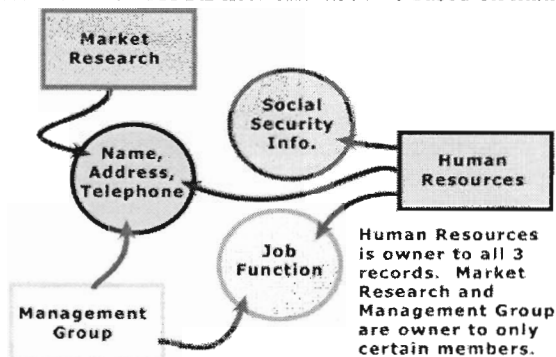


Figure 9. Types of DBMS: Network Databases

### Relational Model

(RDBMS - relational database management system) A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

Values Are Atomic  
 Each Row is Unique  
 Column Values Are of the Same Kind  
 The Sequence of Columns is Insignificant  
 The Sequence of Rows is Insignificant  
 Each Column Has a Unique Name

|   | First Name | Last Name | Social Security No. |
|---|------------|-----------|---------------------|
| 1 | John       | Smith     | 010-22-9432         |
| 2 | John       | Smith     | 003-63-0037         |
| 3 | John       | Smith     | 003-63-0037         |
| 4 | John       | Smith     | 003-63-0037         |
| 5 | Sally      | Smith     | 003-63-0037         |
| 6 | Steve      | Smith     | 003-63-0037         |

|   | Date of Birth | Social Security No. |
|---|---------------|---------------------|
| 1 | 6/12/82       | 010-22-9432         |
| 2 | 5/9/40        | 003-63-0037         |
| 3 | 5/9/40        | 003-63-0037         |
| 4 | 5/9/40        | 003-63-0037         |
| 5 | 5/9/40        | 003-63-0037         |
| 6 | 5/9/40        | 003-63-0037         |

|   | Address             | Social Security No. |
|---|---------------------|---------------------|
| 1 | 321 Byberry Road    | 010-22-9432         |
| 2 | 268 Monroe Avenue   | 003-63-0037         |
| 3 | 8120 Venshire Drive | 020-45-9326         |
| 4 | 207 Congress Drive  | 289-56-4321         |
| 5 | 1519 Ashbury Lane   | 420-54-2933         |

Figure 10. Types of DBMS: Relational Databases

Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retrieval time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

### Object/Relational Model

Object/relational database management systems (ORDBMSs) add new object storage capabilities to the relational systems at the core of modern information systems. These new facilities integrate management of traditional fielded data, complex objects such as time-series and geospatial data and diverse binary media such as audio, video, images, and applets. By encapsulating methods with data structures, an ORDBMS server can execute complex analytical and data manipulation operations to search and transform multimedia and other complex objects.

As an evolutionary technology, the object/relational (OR) approach has inherited the robust transaction- and performance-management features of its relational

ancestor and the flexibility of its object-oriented cousin. Database designers can work with familiar tabular structures and data definition languages (DDLs) while assimilating new object-management possibilities. Query and procedural languages and call interfaces in ORDBMSs are familiar: SQL3, vendor procedural languages, and ODBC, JDBC, and proprietary call interfaces are all extensions of RDBMS languages and interfaces. And the leading vendors are, of course, quite well known: IBM, Informix, and Oracle.

### **Object-Oriented Model**

Object DBMSs add database functionality to object programming languages. They bring much more than persistent storage of programming language objects. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability, while retaining native language compatibility. A major benefit of this approach is the unification of the application and database development into a seamless data model and language environment. As a result, applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.

According to Rao (1994), "The object-oriented database (OODB) paradigm is the combination of object-oriented programming language (OOP) systems and persistent systems. The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs."

In contrast to a relational DBMS where a complex data structure must be flattened out to fit into tables or joined together from those tables to form the in-memory structure, object DBMSs have no performance overhead to store or retrieve a web or hierarchy of interrelated objects. This one-to-one mapping of object programming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object DBMSs better suited to support applications such as financial portfolio risk analysis systems, telecommunications service applications, world wide web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data.

### **Semistructured Model**

In semistructured data model, the information that is normally associated with a schema is contained within the data, which is sometimes called "self-describing". In such database there is no clear separation between the data and the schema, and the degree to which it is structured depends on the application. In some forms of semistructured data there is no separate schema, in others it exists but only places loose constraints on the data. Semi-structured data is naturally modelled in terms of graphs which contain labels which give semantics to its underlying structure. Such databases subsume the modelling power of recent extensions of flat relational

databases, to nested databases which allow the nesting (or encapsulation) of entities, and to object databases which, in addition, allow cyclic references between objects.

Semistructured data has recently emerged as an important topic of study for a variety of reasons. First, there are data sources such as the Web, which we would like to treat as databases but which cannot be constrained by a schema. Second, it may be desirable to have an extremely flexible format for data exchange between disparate databases. Third, even when dealing with structured data, it may be helpful to view it as semistructured for the purposes of browsing.

## Lesson 7.

### *Theme of lecture: Relational Algebra and Relational Calculus*

#### *Plan:*

*Relational Algebra*

*Relational Calculus*

### **Relational Algebra**

Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input and result into a new relation as an output. These operations are divided into two groups. One group consists of operations developed specifically for relational databases such as select, project, rename, join, and division. The other group includes the set-oriented operations such as union, intersection, difference, and cartesian product. Some of the queries, which are mathematical in nature, cannot be expressed using the basic operations of relational algebra. For such queries, additional operations like aggregate functions and grouping are used such as finding sum, average, etc. This section discusses these operations in detail.

#### *Formal Relational Query Languages*

- ❖ Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
  - *Relational Algebra*: More operational (procedural), very useful for representing execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it: Non-operational, *declarative*.
- ❖ A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed.
  - The schema for the *result* of a given query is also fixed! - determined by definition of query language constructs.
- ❖ Positional vs. named-field notation:

- Positional notation easier for formal definitions, named-field notation more readable.
- Both used in SQL

### Relational Algebra

#### ❖ Basic operations:

- Selection ( $\sigma$ ) Selects a subset of rows from relation.
- Projection ( $\pi$ ) Deletes unwanted columns from relation.
- Cross-product ( $\times$ ) Allows us to combine two relations.
- Set-difference ( $-$ ) Tuples in reln. 1, but not in reln. 2.
- Union ( $\cup$ ) Tuples in reln. 1 and in reln. 2.

#### ❖ Additional operations:

- Intersection, join, division, renaming: Not essential, but (very!) useful.

Since each operation returns a relation, operations can be composed: algebra is “closed”.

### Projection

- ❖ Deletes attributes that are not in projection list.
- ❖ Schema of result contains exactly the fields in the projection list, with the same names that they had in the input relation.
- ❖ Projection operator has to eliminate duplicates! Why?
  - *Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it (by DISTINCT). Why not?*

### Selection

- ❖ Selects rows that satisfy selection condition.
- ❖ No duplicates in result! Why?
- ❖ Schema of result identical to schema of input relation.
- ❖ What is Operator composition?
- ❖ Selection is distributive over binary operators
- ❖ Selection is commutative

Table 2. Union, Intersection, Set-Difference

| sid | sname  | rating | age  |
|-----|--------|--------|------|
| 22  | dustin | 7      | 45.0 |
| 31  | lubber | 8      | 55.5 |
| 58  | rusty  | 10     | 35.0 |
| 44  | guppy  | 5      | 35.0 |
| 28  | yuppy  | 9      | 35.0 |

$$S1 \cup S2$$

Table 3.

| sid | sname  | rating | age  |
|-----|--------|--------|------|
| 22  | dustin | 7      | 45.0 |

$$S1 - S2$$

Table 4.

| sid | sname  | rating | age  |
|-----|--------|--------|------|
| 31  | lubber | 8      | 55.5 |
| 58  | rusty  | 10     | 35.0 |

$$S1 \cap S2$$

*Cross-Product (Cartesian Product)*

- ❖ Each row of S1 is paired with each row of R1.
- ❖ *Result schema* has one field per field of S1 and R1, with field names 'inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

Table 5.

| (sid) | sname  | rating | age  | (sid) | bid | day      |
|-------|--------|--------|------|-------|-----|----------|
| 22    | dustin | 7      | 45.0 | 22    | 101 | 10/10/96 |
| 22    | dustin | 7      | 45.0 | 58    | 103 | 11/12/96 |
| 31    | lubber | 8      | 55.5 | 22    | 101 | 10/10/96 |
| 31    | lubber | 8      | 55.5 | 58    | 103 | 11/12/96 |
| 58    | rusty  | 10     | 35.0 | 22    | 101 | 10/10/96 |
| 58    | rusty  | 10     | 35.0 | 58    | 103 | 11/12/96 |

- Renaming operator:

$$\rho (C (1 \rightarrow sid_1, 5 \rightarrow sid_2), S1 \times R1)$$

*Joins: used to combine relations*

Condition Join:  $R \bowtie_c S = \sigma_c (R \times S)$

Table 6.

| (sid) | sname  | rating | age  | (sid) | bid | day      |
|-------|--------|--------|------|-------|-----|----------|
| 22    | dustin | 7      | 45.0 | 58    | 103 | 11/12/96 |
| 31    | lubber | 8      | 55.5 | 58    | 103 | 11/12/96 |

$$S1 \bowtie_c S1.sid < R1.sid \quad R1$$

- ❖ Result schema same as that of cross-product.
- ❖ Fewer tuples than cross-product, might be able to compute more efficiently
- ❖ Sometimes called a theta-join.

*Division*

- ❖ Not supported as a primitive operator, but useful for expressing queries like:  
Find sailors who have reserved all boats.

- ❖ Let  $A$  have 2 fields,  $x$  and  $y$ ;  $B$  have only field  $y$ :
  - $A/B =$
  - i.e.,  $A/B$  contains all  $x$  tuples (sailors) such that for every  $y$  tuple (boat) in  $B$ , there is an  $xy$  tuple in  $A$ .
  - Or: If the set of  $y$  values (boats) associated with an  $x$  value (sailor) in  $A$  contains all  $y$  values in  $B$ , the  $x$  value is in  $A/B$ .
- ❖ In general,  $x$  and  $y$  can be any lists of fields;  $y$  is the list of fields in  $B$ , and  $x$  is the list of fields of  $A$ .

Table 7.

Examples of Division  $A/B$

| sno | pno |      |      |      |
|-----|-----|------|------|------|
| s1  | p1  | pno  | pno  | pno  |
| s1  | p2  | p2   | p2   | p1   |
| s1  | p3  | B1   | p4   | p2   |
| s1  | p4  |      | B2   | p4   |
| s2  | p1  |      |      | B3   |
| s2  | p2  | sno  |      |      |
| s3  | p2  | s1   | sno  |      |
| s4  | p2  | s2   | s1   | sno  |
| s4  | p4  | s3   | s4   | s1   |
|     |     | s4   |      |      |
|     |     | A/B1 | A/B2 | A/B3 |

## Relational Calculus

- Comes in two flavors: Tuple relational calculus (TRC) and Domain relational calculus (DRC).
- Calculus has *variables*, *constants*, *comparison ops*, *logical connectives* and *quantifiers*.
  - TRC: Variables range over (i.e., get bound to) *tuples*.
  - DRC: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to true.

# Domain Relational Calculus

- ❖ *Query* has the form:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle) \}$$

- ❖ *Answer* includes all tuples  $\langle x_1, x_2, \dots, x_n \rangle$  that make the formula  $p(\langle x_1, x_2, \dots, x_n \rangle)$  be true.
- ❖ *Formula* is recursively defined, starting with simple atomic formulas (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the logical connectives.

## DRC Formulas

- ❖ *Atomic formula*:  $x_1, x_2, \dots, x_n \in Rname$ , or  $X op Y$ , or  $X op constant$ 
  - $op$  is one of  $<, >, =, \leq, \geq, \neq$
- ❖ *Formula*:
  - an atomic formula, or
  - $\neg p, p \wedge q, p \vee q$ , where  $p$  and  $q$  are formulas, or
  - $\exists X (p(X))$ , where  $X$  is a domain variable or
  - $\forall X (p(X))$ , where  $X$  is a domain variable.
- ❖ The use of quantifiers  $\exists$  and  $\forall$  is said to bind  $X$ .

### Free and Bound Variables

- ❖ The use of quantifiers  $\exists X$  and  $\forall X$  in a formula is said to bind  $X$ .
  - A variable that is not bound is free.
- ❖ Let us revisit the definition of a query:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle) \}$$

- ❖ There is an important restriction: the variables  $x_1, \dots, x_n$  that appear to the left of  $\{ \}$  must be the *only* free variables in the formula  $p(\dots)$ .

### Summary of Relational Algebra

The relational model has rigorously defined query languages that are simple and powerful.

Relational algebra is more operational; useful as internal representation for query evaluation plans.

Several ways of expressing a given query; a query optimizer should choose the most efficient version.

### Summary of Relational Calculus



Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)

Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.

## **Lesson 8.**

*Theme of lecture: Entity–relationship model*

### *Plan*

*Entity – Relationship model (ER model for short)*

*The building blocks: entities, relationships, and attributes*

*Entity–relationship diagrams don't show single entities or single instances of relations.*

In software engineering, an Entity – Relationship model (ER model for short) is an abstract way to describe a database. It usually starts with a relational database, which stores data in tables. Some of the data in these tables point to data in other tables - for instance, your entry in the database could point to several entries for each of the phone numbers that are yours. The ER model would say that you are an entity, and each phone number is an entity, and the relationship between you and the phone numbers is 'has a phone number'. Diagrams created to design these entities and relationships are called entity–relationship diagrams or ER diagrams.

Using the three schema approach to software engineering, there are three levels of ER models that may be developed. The conceptual data model is the highest level ER model in that it contains the least granular detail but establishes the overall scope of what is to be included within the model set. The conceptual ER model normally defines master reference data entities that are commonly used by the organization. Developing an enterprise-wide conceptual ER model is useful to support documenting the data architecture for an organization.

A conceptual ER model may be used as the foundation for one or more logical data models. The purpose of the conceptual ER model is then to establish structural metadata commonality for the master data entities between the set of logical ER models. The conceptual data model may be used to form commonality relationships between ER models as a basis for data model integration.

A logical ER model does not require a conceptual ER model especially if the scope of the logical ER model is to develop a single disparate information system. The logical ER model contains more detail than the conceptual ER model. In addition to master data entities, operational and transactional data entities are now defined. The details of each data entity are developed and the entity relationships between these data entities are established. The logical ER model is however developed independent of technology into which it will be implemented.

One or more physical ER models may be developed from each logical ER model. The physical ER model is normally developed be instantiated as a database.

Therefore, each physical ER model must contain enough detail to produce a database and each physical ER model is technology dependent since each database management system is somewhat different.

The physical model is normally forward engineered to instantiate the structural metadata into a database management system as relational database objects such as database tables, database indexes such as unique key indexes, and database constraints such as a foreign key constraint or a commonality constraint. The ER model is also normally used to design modifications to the relational database objects and to maintain the structural metadata of the database.

The first stage of information system design uses these models during the requirements analysis to describe information needs or the type of information that is to be stored in a database. The data modeling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain area of interest. In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model; this in turn is mapped to a physical model during physical design. Note that sometimes, both of these phases are referred to as "physical design".

The building blocks: entities, relationships, and attributes

Two related entities

An entity with an attribute

A relationship with an attribute

Primary key

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of a domain. When we speak of an entity, we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world [4].

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type. Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term.

Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem.

A relationship captures how entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: owns relationship between a company and a computer, supervises relationship between an employee and a department, a performs relationship between an artist and a song, a proved relationship between a mathematician and a theorem.

The model's linguistic aspect described above is utilized in the declarative database query language ERROL, which mimics natural language, constructs. ERROL's semantics and implementation are based on Reshaped relational algebra (RRA), a relational algebra which is adapted to the entity-relationship model and captures its linguistic aspect.

Entities and relationships can both have attributes. Examples: an employee entity might have a Social Security Number (SSN) attribute; the proved relationship may have a date attribute.



Figure 11.



Figure 12.



Figure 13.

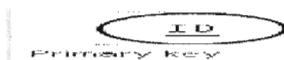


Figure 14.

Every entity (unless it is a weak entity) must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.

Entity-relationship diagrams don't show single entities or single instances of relations. Rather, they show entity sets and relationship sets. Example: a particular song is an entity. The collection of all songs in a database is an entity set. The eaten relationship between a child and her lunch is a single relationship. The set of all such child-lunch relationships in a database is a relationship set. In other words, a relationship set corresponds to a relation in mathematics, while a relationship corresponds to a member of the relation.

## Lesson 9.

### *Theme of lecture: Database Normalization Basics*

#### *Plan*

*Normalization is the process of efficiently organizing data in a database.*

#### *The Normal Forms*

#### *First Normal Form*

If you've been working with databases for a while, chances are you've heard the term normalization. Perhaps someone's asked you "Is that database normalized?" or "Is that in BCNF?" All too often, the reply is "Uh, yeah." Normalization is often brushed aside as a luxury that only academics have time for. However, knowing the principles of normalization and applying them to your daily database design tasks really isn't all that complicated and it could drastically improve the performance of your DBMS.

In this lecture, we'll introduce the concept of normalization and take a brief look at the most common normal forms. Future articles will provide in-depth explorations of the normalization process.

#### *What is Normalization?*

Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

#### *The Normal Forms*

The database community has developed a series of guidelines for ensuring that databases are normalized. These are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or 1NF) through five (fifth normal form or 5NF). In practical applications, you'll often see 1NF, 2NF, and 3NF along with the occasional 4NF. Fifth normal form is very rarely seen and won't be discussed in this lecture.

Before we begin our discussion of the normal forms, it's important to point out that they are guidelines and guidelines only. Occasionally, it becomes necessary to stray from them to meet practical business requirements. However, when variations take place, it's extremely important to evaluate any possible ramifications they could have on your system and account for possible inconsistencies. That said, let's explore the normal forms.

#### *First Normal Form (1NF)*

First normal form (1NF) sets the very basic rules for an organized database: Eliminate duplicative columns from the same table.

Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Eliminate duplicative columns from the same table.

Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

For more details, read Putting your Database in First Normal Form

Second Normal Form (2NF)

Second normal form (2NF) further addresses the concept of removing duplicative data: Meet all the requirements of the first normal form.

Remove subsets of data that apply to multiple rows of a table and place them in separate tables.

Create relationships between these new tables and their predecessors through the use of foreign keys.

Meet all the requirements of the first normal form.

Remove subsets of data that apply to multiple rows of a table and place them in separate tables.

Create relationships between these new tables and their predecessors through the use of foreign keys.

For more details, read Putting your Database in Second Normal Form

Third Normal Form (3NF)

Third normal form (3NF) goes one large step further: Meet all the requirements of the second normal form.

Remove columns that are not dependent upon the primary key.

Meet all the requirements of the second normal form.

Remove columns that are not dependent upon the primary key.

For more details, read Putting your Database in Third Normal Form

Boyce-Codd Normal Form (BCNF or 3.5NF)

The Boyce-Codd Normal Form, also referred to as the "third and half (3.5) normal form", adds one more requirement: Meet all the requirements of the third normal form.

Every determinant must be a candidate key.

Meet all the requirements of the third normal form.

Every determinant must be a candidate key.

For more details, read Putting your Database in Boyce Codd Normal Form

Fourth Normal Form (4NF)

Finally, fourth normal form (4NF) has one additional requirement: Meet all the requirements of the third normal form.

A relation is in 4NF if it has no multi-valued dependencies.

Meet all the requirements of the third normal form.

A relation is in 4NF if it has no multi-valued dependencies.

Remember, these normalization guidelines are cumulative. For a database to be in 2NF, it must first fulfill all the criteria of a 1NF database.

While database normalization is often a good idea, it's not an absolute requirement. In fact, there are some cases where deliberately violating the rules of normalization is a good practice. For more on this topic, read *Should I Normalize My Database?*

## **Lesson 10.**

*Theme of lecture:* **Network Information System (NIS/YP)**

*Plan*

*Terms/Processes*

*Choosing a NIS Domain Name*

*Physical Server Requirements*

*NIS Servers*

NIS, which stands for Network Information Services, was developed by Sun Microsystems to centralize administration of UNIX® (originally SunOS™) systems. It has now essentially become an industry standard; all major UNIX like systems (Solaris™, HP-UX, AIX®, Linux, NetBSD, OpenBSD, FreeBSD, etc) support NIS.

NIS was formerly known as Yellow Pages, but because of trademark issues, Sun changed the name. The old term (and yp) is still often seen and used.

It is a RPC-based client/server system that allows a group of machines within an NIS domain to share a common set of configuration files. This permits a system administrator to set up NIS client systems with only minimal configuration data and add, remove or modify configuration data from a single location.

It is similar to the Windows NT® domain system; although the internal implementation of the two are not at all similar, the basic functionality can be compared.

*Terms/Processes*

There are several terms and several important user processes that you will come across when attempting to implement NIS on FreeBSD, whether you are trying to create an NIS server or act as an NIS client:

Table 8.

| Term           | Description  |
|----------------|--|
| NIS domainname | An NIS master server and all of its clients (including its slave servers) have a NIS domainname. Similar to an Windows NT domain name, the NIS domainname does not have anything to do with DNS  |
| Rpcbind        | Must be running in order to enable RPC (Remote Procedure Call, a network protocol used by NIS). If rpcbind is not running, it will be impossible to run an NIS server, or to act as an NIS client  |
| Ypbind         | "Binds" an NIS client to its NIS server. It will take the NIS domainname from the system, and using RPC, connect to the server. ypbind is the core of client-server communication in an NIS environment; if ypbind dies on a client machine, it will not be able to access the NIS server  |
| Ypserv         | Should only be running on NIS servers; this is the NIS server process itself. If ypserv(8) dies, then the server will no longer be able to respond to NIS requests (hopefully, there is a slave server to take over for it). There are some implementations of NIS (but not the FreeBSD one), that do not try to reconnect to another server if the server it used before dies. Often, the only thing that helps in this case is to restart the server process (or even the whole server) or the ypbind process on the client. |
| rpc.yppasswdd  | Another process that should only be running on NIS master servers; this is a daemon that will allow NIS clients to change their NIS passwords. If this daemon is not running, users will have to login to the NIS master server and change their passwords there   |

### *How Does It Work?*

There are three types of hosts in an NIS environment: master servers, slave servers, and clients. Servers act as a central repository for host configuration information. Master servers hold the authoritative copy of this information, while slave servers mirror this information for redundancy. Clients rely on the servers to provide this information to them.

Information in many files can be shared in this manner. The master.passwd, group, and hosts files are commonly shared via NIS. Whenever a process on a client needs information that would normally be found in these files locally, it makes a query to the NIS server that it is bound to instead.

### *Machine Types*

A NIS master server. This server, analogous to a Windows NT primary domain controller, maintains the files used by all of the NIS clients. The passwd, group, and other various files used by the NIS clients live on the master server.

*Note: It is possible for one machine to be an NIS master server for more than one NIS domain. However, this will not be covered in this introduction, which assumes a relatively small-scale NIS environment.*

NIS slave servers. Similar to the Windows NT backup domain controllers, NIS slave servers maintain copies of the NIS master's data files. NIS slave servers provide the redundancy, which is needed in important environments. They also help to balance the load of the master server: NIS Clients always attach to the NIS server whose response they get first, and this includes slave-server-replies.

NIS clients. NIS clients, like most Windows NT workstations, authenticate against the NIS server (or the Windows NT domain controller in the Windows NT workstations case) to log on.

### *Using NIS/YP*

This section will deal with setting up a sample NIS environment.

### Planning

Let us assume that you are the administrator of a small university lab. This lab, which consists of 15 FreeBSD machines, currently has no centralized point of administration; each machine has its own /etc/passwd and /etc/master.passwd. These files are kept in sync with each other only through manual intervention; currently, when you add a user to the lab, you must run adduser on all 15 machines. Clearly, this has to change, so you have decided to convert the lab to use NIS, using two of the machines as servers.

Therefore, the configuration of the lab now looks something like:

Table 9.

| Machine name | IP address    | Machine role          |
|--------------|---------------|-----------------------|
| Ellington    | 10.0.0.2      | NIS master            |
| Coltrane     | 10.0.0.3      | NIS slave             |
| Basie        | 10.0.0.4      | Faculty workstation   |
| Bird         | 10.0.0.5      | Client machine        |
| cli[1-11]    | 10.0.0.[6-17] | Other client machines |

If you are setting up a NIS scheme for the first time, it is a good idea to think through how you want to go about it. No matter what the size of your network, there are a few decisions that need to be made.

### **Choosing a NIS Domain Name**

This might not be the "domainname" that you are used to. It is more accurately called the "NIS domainname". When a client broadcasts its requests for info, it



includes the name of the NIS domain that it is part of. This is how multiple servers on one network can tell which server should answer which request. Think of the NIS domainname as the name for a group of hosts that are related in some way.

Some organizations choose to use their Internet domainname for their NIS domainname. This is not recommended as it can cause confusion when trying to debug network problems. The NIS domainname should be unique within your network and it is helpful if it describes the group of machines it represents. For example, the Art department at Acme Inc. might be in the "acme-art" NIS domain. For this example, assume you have chosen the name test-domain.

However, some operating systems (notably SunOS) use their NIS domain name as their Internet domain name. If one or more machines on your network have this restriction, you must use the Internet domain name as your NIS domain name.

### **Physical Server Requirements**

There are several things to keep in mind when choosing a machine to use as a NIS server. One of the unfortunate things about NIS is the level of dependency the clients have on the server. If a client cannot contact the server for its NIS domain, very often the machine becomes unusable. The lack of user and group information causes most systems to temporarily freeze up. With this in mind you should make sure to choose a machine that will not be prone to being rebooted regularly, or one that might be used for development. The NIS server should ideally be a stand alone machine whose sole purpose in life is to be an NIS server. If you have a network that is not very heavily used, it is acceptable to put the NIS server on a machine running other services, just keep in mind that if the NIS server becomes unavailable, it will affect all of your NIS clients adversely.

### **NIS Servers**

The canonical copies of all NIS information are stored on a single machine called the NIS master server. The databases used to store the information are called NIS maps. In FreeBSD, these maps are stored in /var/yp/[domainname] where [domainname] is the name of the NIS domain being served. A single NIS server can support several domains at once, therefore it is possible to have several such directories, one for each supported domain. Each domain will have its own independent set of maps.

NIS master and slave servers handle all NIS requests with the ypserv daemon. ypserv is responsible for receiving incoming requests from NIS clients, translating the requested domain and map name to a path to the corresponding database file and transmitting data from the database back to the client.

## **Lesson 11.**

*Theme of lecture: Usage Database. Database security*

*Plan*

*Access control*

*Data security*

*Database audit*

Database security deals with all various aspects of protecting the database content, its owners, and its users. It ranges from protection from intentional unauthorized database uses to unintentional database accesses by unauthorized entities (e.g., a person or a computer program).

Database security concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/administrative and physical. Database security is a specialist topic within the broader realms of computer security, information security and risk management.

Security risks to database systems include, for example:

Unauthorized or unintended activity or misuse by authorized database users, database administrators, or network/systems managers, or by unauthorized users or hackers (e.g. inappropriate access to sensitive data, metadata or functions within databases, or inappropriate changes to the database programs, structures or security configurations);

Malware infections causing incidents such as unauthorized access, leakage or disclosure of personal or proprietary data, deletion of or damage to the data or programs, interruption or denial of authorized access to the database, attacks on other systems and the unanticipated failure of database services;

Overloads, performance constraints and capacity issues resulting in the inability of authorized users to use databases as intended;

Physical damage to database servers caused by computer room fires or floods, overheating, lightning, accidental liquid spills, static discharge, electronic breakdowns/equipment failures and obsolescence;

Design flaws and programming bugs in databases and the associated programs and systems, creating various security vulnerabilities (e.g. unauthorized privilege escalation), data loss/corruption, performance degradation etc.;

Data corruption and/or loss caused by the entry of invalid data or commands, mistakes in database or system administration processes, sabotage/criminal damage etc.

Many layers and types of information security control are appropriate to databases, including:

- Access control

- Auditing
- Authentication
- Encryption
- Integrity controls
- Backups
- Application security

Traditionally databases have been largely secured against hackers through network security measures such as firewalls, and network-based intrusion detection systems. While network security controls remain valuable in this regard, securing the database systems themselves, and the programs/functions and data within them, has arguably become more critical as networks are increasingly opened to wider access, in particular access from the Internet. Furthermore, system, program, function and data access controls, along with the associated user identification, authentication and rights management functions, have always been important to limit and in some cases log the activities of authorized users and administrators. In other words, these are complementary approaches to database security, working from both the outside-in and the inside-out as it were.

Many organizations develop their own "baseline" security standards and designs detailing basic security control measures for their database systems. These may reflect general information security requirements or obligations imposed by corporate information security policies and applicable laws and regulations (e.g. concerning privacy, financial management and reporting systems), along with generally-accepted good database security practices (such as appropriate hardening of the underlying systems) and perhaps security recommendations from the relevant database system and software vendors. The security designs for specific database systems typically specify further security administration and management functions (such as administration and reporting of user access rights, log management and analysis, database replication/synchronization and backups) along with various business-driven information security controls within the database programs and functions (e.g. data entry validation and audit trails). Furthermore, various security-related activities (manual controls) are normally incorporated into the procedures, guidelines etc. relating to the design, development, configuration, use, management and maintenance of databases.

The following are major areas of database security (among many others).

#### **Access control**

Database access control deals with controlling who (a person or a certain computer program) is allowed to access what information in the database. The information may comprise specific database objects (e.g., record types, specific records, data structures), certain computations over certain objects (e.g., query types, or specific queries), or utilizing specific access paths to the former (e.g., using specific indexes or other data structures to access information).

Access control refers to exerting control over who can interact with a resource. Often but not always, this involves an authority, who does the controlling. The resource can be a given building, group of buildings, or computer-based information system. But it can also refer to a restroom.

Access control is, in reality, an everyday phenomenon. A lock on a car door is essentially a form of access control. A PIN on an ATM system at a bank is another means of access control. The possession of access control is of prime importance when persons seek to secure important, confidential, or sensitive information and equipment.

Item control or electronic key management is an area within (and possibly integrated with) an access control system which concerns the managing of possession and location of small assets or physical (mechanical) keys

Database access controls are set by special authorized (by the database owner) personnel that uses dedicated protected security DBMS interfaces.

Physical access by a person may be allowed depending on payment, authorization, etc. Also there may be one-way traffic of people. These can be enforced by personnel such as a border guard, a doorman, a ticket checker, etc., or with a device such as a turnstile. There may be fences to avoid circumventing this access control. An alternative of access control in the strict sense (physically controlling access itself) is a system of checking authorized presence, see e.g. Ticket controller (transportation). A variant is exit control, e.g. of a shop (checkout) or a country.

In physical security, the term access control refers to the practice of restricting entrance to a property, a building, or a room to authorized persons. Physical access control can be achieved by a human (a guard, bouncer, or receptionist), through mechanical means such as locks and keys, or through technological means such as access control systems like the mantrap. Within these environments, physical key management may also be employed as a means of further managing and monitoring access to mechanically keyed areas or access to certain small assets.[

Physical access control is a matter of who, where, and when. An access control system determines who is allowed to enter or exit, where they are allowed to enter or exit, and when they are allowed to enter or exit. Historically this was partially accomplished through keys and locks. When a door is locked only someone with a key can enter through the door depending on how the lock is configured. Mechanical locks and keys do not allow restriction of the key holder to specific times or dates. Mechanical locks and keys do not provide records of the key used on any specific door and the keys can be easily copied or transferred to an unauthorized person. When a mechanical key is lost or the key holder is no longer authorized to use the protected area, the locks must be re-keyed.

Electronic access control uses computers to solve the limitations of mechanical locks and keys. A wide range of credentials can be used to replace mechanical keys. The electronic access control system grants access based on the credential presented. When access is granted, the door is unlocked for a predetermined time and the

transaction is recorded. When access is refused, the door remains locked and the attempted access is recorded. The system will also monitor the door and alarm if the door is forced open or held open too long after being unlocked.

### **Data security**

The definition of data security varies and may overlap with other database security aspects. Broadly it deals with protecting specific chunks of data, both physically (i.e., from corruption, or destruction, or removal; e.g., see Physical security), or the interpretation of them, or parts of them to meaningful information (e.g., by looking at the strings of bits that they comprise, concluding specific valid credit-card numbers; e.g., see Data encryption).

#### *Data Security Technologies*

##### *Disk Encryption*

Disk encryption refers to encryption technology that encrypts data on a hard disk drive. Disk encryption typically takes form in either software (see disk encryption software) or hardware (see disk encryption hardware). Disk encryption is often referred to as on-the-fly encryption ("OTFE") or transparent encryption.

##### *Hardware based Mechanisms for Protecting Data*

Software based security solutions encrypt the data to prevent data from being stolen. However, a malicious program or a hacker may corrupt the data in order to make it unrecoverable or unusable. Similarly, encrypted operating systems can be corrupted by a malicious program or a hacker, making the system unusable. Hardware-based security solutions can prevent read and write access to data and hence offers very strong protection against tampering and unauthorized access.

Hardware based or assisted computer security offers an alternative to software-only computer security. Security tokens such as those using PKCS#11 may be more secure due to the physical access required in order to be compromised. Access is enabled only when the token is connected and correct PIN is entered (see two factor authentication). However, dongles can be used by anyone who can gain physical access to it. Newer technologies in hardware based security solves this problem offering fool proof security for data.

**Working of Hardware based security:** A hardware device allows a user to login, logout and to set different privilege levels by doing manual actions. The device uses biometric technology to prevent malicious users from logging in, logging out, and changing privilege levels. The current state of a user of the device is read by controllers in peripheral devices such as harddisks. Illegal access by a malicious user or a malicious program is interrupted based on the current state of a user by harddisk and DVD controllers making illegal access to data impossible. Hardware based access control is more secure than protection provided by the operating systems as operating systems are vulnerable to malicious attacks by viruses and hackers. The data on harddisks can be corrupted after a malicious access is obtained. With hardware based protection, software cannot manipulate the user privilege levels, it is impossible for a hacker or a malicious program to gain access to secure data protected by hardware or perform unauthorized privileged operations. The hardware

protects the operating system image and file system privileges from being tampered. Therefore, a completely secure system can be created using a combination of hardware based security and secure system administration policies.

In cryptography, encryption is the process of encoding messages (or information) in such a way that eavesdroppers or hackers cannot read it, but that authorized parties. In an encryption scheme, the message or information (referred to as plaintext) is encrypted using an encryption algorithm, turning it into an unreadable ciphertext (ibid.). This is usually done with the use of an encryption key, which specifies how the message is to be encoded. Any adversary that can see the ciphertext, should not be able to determine anything about the original message. An authorized party, however, is able to decode the ciphertext using a decryption algorithm, that usually requires a secret decryption key, that adversaries do not have access to. For technical reasons, an encryption scheme usually needs a key-generation algorithm, to randomly produce keys.

There are two basic types of encryption schemes private-key encryption and public-key encryption. In private-key schemes, the encryption and decryption keys are the same. Thus communicating parties must agree on a secret key before they wish to communicate. By contrast, in public-key schemes, the encryption key is public: that is, anyone (friend or foe) has access to the encryption key, and can encrypt messages. However only the receiving party has access to the decryption key and thus is the only one capable of reading the encrypted messages. Public-key encryption is a relatively recent invention: historically, all encryption schemes have been private-key schemes.

Encryption has long been used by militaries and governments to facilitate secret communication. It is now commonly used in protecting information within many kinds of civilian systems. For example, the Computer Security Institute reported that in 2007, 71% of companies surveyed utilized encryption for some of their data in transit, and 53% utilized encryption for some of their data in storage. Encryption can be used to protect data "at rest", such as files on computers and storage devices (e.g. USB flash drives). In recent years there have been numerous reports of confidential data such as customers' personal records being exposed through loss or theft of laptops or backup drives. Encrypting such files at rest helps protect them should physical security measures fail. Digital rights management systems which prevent unauthorized use or reproduction of copyrighted material and protect software against reverse engineering (see also copy protection) are another somewhat different example of using encryption on data at rest.

Encryption is also used to protect data in transit, for example data being transferred via networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in transit being intercepted in recent years. Encrypting data in transit also helps to secure it as it is often difficult to physically secure all access to networks.

Encryption, by itself, can protect the confidentiality of messages, but other techniques are still needed to protect the integrity and authenticity of a message; for example, verification of a message authentication code (MAC) or a digital signature. Standards and cryptographic software and hardware to perform encryption are widely available, but successfully using encryption to ensure security may be a challenging problem. A single slip-up in system design or execution can allow successful attacks. Sometimes an adversary can obtain unencrypted information without directly undoing the encryption. One of the earliest public key encryption applications was called Pretty Good Privacy (PGP). It was written in 1991 by Phil Zimmermann and was purchased by Symantec in 2010.

Digital signature and encryption must be applied at message creation time (i.e. on the same device it has been composed) to avoid tampering. Otherwise any node between the sender and the encryption agent could potentially tamper it. It should be noted that encrypting at the time of creation only adds security if the encryption device itself has not been tampered with.

### **Database audit**

Database audit primarily involves monitoring that no security breach, in all aspects, has happened. If security breach is discovered then all possible corrective actions are taken.

## **Lesson 12.**

*Theme of lecture:* **Analyze the current technology which materialize database. Standards and languages. (MS SQL)**

### *Plan*

#### *SQL Fundamentals*

#### *Basic commands of SQL:*

*CREATE*

*USE*

*ALTER*

*DROP*

#### *Data Manipulation Language*

*INSERT*

*SELECT*

*UPDATE*

*DELETE*

## **SQL Fundamentals**

### **Introduction**

The Structured Query Language (SQL) comprises one of the fundamental building blocks of modern database architecture. SQL defines the methods used to create and manipulate relational databases on all major platforms. At first glance, the language may seem intimidating and complex but it's really not all that bad.

We'll take a brief look at some of the main commands used to create and modify databases and provide a few examples for illustrative purposes and explain the theory behind them.

By the way, the correct pronunciation of SQL is a contentious issue within the database community. In their SQL standard, the American National Standards Institute declared that the official pronunciation is "es queue el." However, many database professionals have taken to the slang pronunciation "sequel." The choice is yours.

SQL comes in many flavors. Oracle databases utilize their proprietary PL/SQL. Microsoft SQL Server makes use of Transact-SQL. However, all of these variations are based upon the industry standard ANSI SQL. In our tutorial series, we'll stick to ANSI-compliant SQL commands that will work on any modern relational database system.

SQL commands can be divided into two main sublanguages. The Data Definition Language (DDL) contains the commands used to create and destroy databases and database objects. After the database structure is defined with DDL, database administrators and users can utilize the Data Manipulation Language to insert, retrieve and modify the data contained within it.

The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

## ***CREATE***

Installing a database management system (DBMS) on a computer allows you to create and manage many independent databases. For example, you may want to maintain a database of customer contacts for your sales department and a personnel database for your HR department. The CREATE command can be used to establish each of these databases on your platform. For example, the command:

*CREATE DATABASE* employees

creates an empty database named "employees" on your DBMS. After creating the database, your next step is to create tables that will contain data. Another variant of the CREATE command can be used for this purpose. The command:

*CREATE TABLE* personal\_info (first\_name char(20) not null, last\_name char(20) not null, employee\_id int not null)

establishes a table titled "personal\_info" in the current database. In our example, the table contains three attributes: first\_name, last\_name and employee\_id.

## ***USE***

The USE command allows you to specify the database you wish to work with within your DBMS. For example, if we're currently working in the sales database and want to issue some commands that will affect the employees database, we would preface them with the following SQL command:



## *USE employees*

It's important to always be conscious of the database you are working in before issuing SQL commands that manipulate data.

### **ALTER**

Once you've created a table within a database, you may wish to modify the definition of it. The ALTER command allows you to make changes to the structure of a table without deleting and recreating it. Take a look at the following command:

```
ALTER TABLE personal_info  
ADD salary money null
```

This example adds a new attribute to the personal\_info table -- an employee's salary. The "money" argument specifies that an employee's salary will be stored using a dollars and cents format. Finally, the "null" keyword tells the database that it's OK for this field to contain no value for any given employee.

### **DROP**

The final command of the Data Definition Language, DROP, allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal\_info table that we created, we'd use the following command:

```
DROP TABLE personal_info
```

Similarly, the command below would be used to remove the entire employees database:

```
DROP DATABASE employees
```

Use this command with care! Remember that the DROP command removes entire data structures from your database. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

That's the Data Definition Language in a nutshell. In the next section of this article, we'll take a look at how the Data Manipulation Language is used to manipulate the information contained within a database.

### **Data Manipulation Language**

The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

### **INSERT**

The INSERT command in SQL is used to add records to an existing table. Returning to the personal\_info example from the previous section, let's imagine that

our HR department needs to add a new employee to their database. They could use a command similar to the one shown below:

```
INSERT INTO personal_info  
values('bart','simpson',12345,$45000)
```

Note that there are four values specified for the record. These correspond to the table attributes in the order they were defined: first\_name, last\_name, employee\_id, and salary.

## **SELECT**

The SELECT command is the most commonly used command in SQL. It allows database users to retrieve the specific information they desire from an operational database. Let's take a look at a few examples, again using the personal\_info table from our employees database.

The command shown below retrieves all of the information contained within the personal\_info table. Note that the asterisk is used as a wildcard in SQL. This literally means "Select everything from the personal\_info table."

```
SELECT *  
FROM personal_info
```

Alternatively, users may want to limit the attributes that are retrieved from the database. For example, the Human Resources department may require a list of the last names of all employees in the company. The following SQL command would retrieve only that information:

```
SELECT last_name  
FROM personal_info
```

Finally, the WHERE clause can be used to limit the records that are retrieved to those that meet specified criteria. The CEO might be interested in reviewing the personnel records of all highly paid employees. The following command retrieves all of the data contained within personal\_info for records that have a salary value greater than \$50,000:

```
SELECT *  
FROM personal_info  
WHERE salary > $50000
```

## **UPDATE**

The UPDATE command can be used to modify information contained within a table, either in bulk or individually. Each year, our company gives all employees a 3% cost-of-living increase in their salary. The following SQL command could be used to quickly apply this to all of the employees stored in the database:

```
UPDATE personal_info  
SET salary = salary * 1.03
```

On the other hand, our new employee Bart Simpson has demonstrated performance above and beyond the call of duty. Management wishes to recognize his

stellar accomplishments with a \$5,000 raise. The WHERE clause could be used to single out Bart for this raise:

```
UPDATE personal_info  
SET salary = salary + $5000  
WHERE employee_id = 12345
```

#### **DELETE**

Finally, let's take a look at the DELETE command. You'll find that the syntax of this command is similar to that of the other DML commands. Unfortunately, our latest corporate earnings report didn't quite meet expectations and poor Bart has been laid off. The DELETE command with a WHERE clause can be used to remove his record from the personal\_info table:

```
DELETE FROM personal_info  
WHERE employee_id = 12345
```

JOIN Statements

Now that you've learned the basics of SQL, it's time to move on to one of the most powerful concepts the language has to offer – the JOIN statement. Quite simply, these statements allow you to combine data in multiple tables to quickly and efficiently process large quantities of data. These statements are where the true power of a database resides.

We'll first explore the use of a basic JOIN operation to combine data from two tables. In future installments, we'll explore the use of outer and inner joins to achieve added power.

We'll continue with our example using the PERSONAL\_INFO table, but first we'll need to add an additional table to the mix. Let's assume we have a table called DISCIPLINARY\_ACTION that was created with the following statement:

```
CREATE TABLE disciplinary_action (action_id int not null, employee_id int  
not null, comments char(500))
```

This table contains the results of disciplinary actions on company employees. You'll notice that it doesn't contain any information about the employee other than the employee number. It's then easy to imagine many scenarios where we might want to combine information from the DISCIPLINARY\_ACTION and PERSONAL\_INFO tables.

Assume we've been tasked with creating a report that lists the disciplinary actions taken against all employees with a salary greater than \$40,000. The use of a JOIN operation in this case is quite straightforward. We can retrieve this information using the following command:

```

SELECT          personal_info.first_name,          personal_info.last_name,
disciplinary_action.comments
FROM personal_info, disciplinary_action
WHERE personal_info.employee_id = disciplinary_action.employee_id
AND personal_info.salary > 40000

```

As you can see, we simply specified the two tables that we wished to join in the FROM clause and then included a statement in the WHERE clause to limit the results to records that had matching employee IDs and met our criteria of a salary greater than \$40,000.

### **Lesson 13.**

*Theme of lecture: MS Access Database Management system*

*Plan*

*Tables*

*Forms*

*Reports*

*Queries*

*Macros*

*Modules*

*Data types*

*Table relationships*

*Keys*

*Benefits of using relationships*

A computerized database is a container of objects. One database can contain more than one table. For example, an inventory tracking system that uses three tables is not three databases, but one database that contains three tables. Unless it has been specifically designed to use data or code from another source, an Access database stores its tables in a single file, along with other objects, such as forms, reports, macros, and modules. Databases created in the Access 2007 format have the file extension .accdb, and databases created in earlier Access formats have the file extension .mdb. You can use Access 2007 to create files in earlier file formats (for example, Access 2000 and Access 2002-2003).

Using Access, you can:

Add new data to a database, such as a new item in an inventory

Edit existing data in the database, such as changing the current location of an item

Delete information, perhaps if an item is sold or discarded

Organize and view the data in different ways

Share the data with others via reports, e-mail messages, an intranet , or the Internet

The following sections are short descriptions of the parts of a typical Access database:

- Tables;
- Forms;
- Reports;
- Queries;
- Macros;
- Modules.

## **Tables**

A database table is similar in appearance to a spreadsheet, in that data is stored in rows and columns. As a result, it is usually quite easy to import a spreadsheet into a database table. The main difference between storing your data in a spreadsheet and storing it in a database is in how the data is organized.

To get the most flexibility out of a database, the data needs to be organized into tables so that redundancies don't occur. For example, if you're storing information about employees, each employee should only need to be entered once in a table that is set up just to hold employee data. Data about products will be stored in its own table, and data about branch offices will be stored in another table. This process is called normalization.

Each row in a table is referred to as a record. Records are where the individual pieces of information are stored. Each record consists of one or more fields. Fields correspond to the columns in the table. For example, you might have a table named "Employees" where each record (row) contains information about a different employee, and each field (column) contains a different type of information, such as first name, last name, address, and so on. Fields must be designated as a certain data type, whether it's text, date or time, number, or some other type.

Another way to describe records and fields is to visualize a library's old-style card catalog. Each card in the cabinet corresponds to a record in the database. Each piece of information on an individual card (author, title, and so on) corresponds to a field in the database.

## **Forms**

Forms are sometimes referred to as "data entry screens." They are the interfaces you use to work with your data, and they often contain command buttons that perform various commands. You can create a database without using forms by simply editing your data in the table datasheets. However, most database users prefer to use forms for viewing, entering, and editing data in the tables.

Forms provide an easy-to-use format for working with the data, and you can also add functional elements, such as command buttons, to them. You can program the buttons to determine which data appears on the form, open other forms or reports, or perform a variety of other tasks. For example, you might have a form named "Customer Form" in which you work with customer data. The customer form might

have a button which opens an order form where you can enter a new order for that customer.

Forms also allow you to control how other users interact with the data in the database. For example, you can create a form that shows only certain fields and allows only certain operations to be performed. This helps protect data and to ensure that the data is entered properly.

## **Reports**

Reports are what you use to summarize and present data in the tables. A report usually answers a specific question, such as "How much money did we receive from each customer this year?" or "What cities are our customers located in?" Each report can be formatted to present the information in the most readable way possible.

A report can be run at any time, and will always reflect the current data in the database. Reports are generally formatted to be printed out, but they can also be viewed on the screen, exported to another program, or sent as e-mail message.

## **Queries**

Queries are the real workhorses in a database, and can perform many different functions. Their most common function is to retrieve specific data from the tables. The data you want to see is usually spread across several tables, and queries allow you to view it in a single datasheet. Also, since you usually don't want to see all the records at once, queries let you add criteria to "filter" the data down to just the records you want. Queries often serve as the record source for forms and reports.

Certain queries are "updateable," meaning you can edit the data in the underlying tables via the query datasheet. If you are working in an updateable query, remember that your changes are actually being made in the tables, not just in the query datasheet.

Queries come in two basic varieties: select queries and action queries. A select query simply retrieves the data and makes it available for use. You can view the results of the query on the screen, print it out, or copy it to the clipboard. Or, you can use the output of the query as the record source for a form or report.

An action query, as the name implies, performs a task with the data. Action queries can be used to create new tables, add data to existing tables, update data, or delete data.

## **Macros**

Macros in Access can be thought of as a simplified programming language which you can use to add functionality to your database. For example, you can attach a macro to a command button on a form so that the macro runs whenever the button is clicked. Macros contain actions that perform tasks, such as opening a report, running a query, or closing the database. Most database operations that you do manually can be automated by using macros, so they can be great time-saving devices.

## Modules

Modules, like macros, are objects you can use to add functionality to your database. Whereas you create macros in Access by choosing from a list of macro actions, you write modules in the Visual Basic for Applications (VBA) programming language. A module is a collection of declarations, statements, and procedures that are stored together as a unit. A module can be either a class module or a standard module. Class modules are attached to forms or reports, and usually contain procedures that are specific to the form or report they're attached to. Standard modules contain general procedures that aren't associated with any other object. Standard modules are listed under Modules in the Navigation Pane, whereas class modules are not.

## Data types

Every field has a data type. A field's data type indicates the kind of data that the field stores, such as large amounts of text or attached files.

Table 10.

| Field Name  | Data Type  | Description |
|-------------|------------|-------------|
| Comments    | Memo       |             |
| Attachments | Attachment |             |

Field Properties

A data type is a field property, but it differs from other field properties as follows:

You set a field's data type in the table design grid, not in the Field Properties pane.

A field's data type determines what other properties the field has.

You must set a field's data type when you create the field.

Note You can create a new field in Access by entering data in a new column in Datasheet view. When you create a field by entering data in Datasheet view, Access automatically assigns a data type for the field, based on the value that you enter. If no other data type is implied by your input, Access sets the data type to Text. If needed, you can change the data type by using the Ribbon, part of the new Microsoft Office Fluent user interface.

The following table shows how automatic data type detection works in Datasheet view.

| Office Access 2007<br>creates a field with a data<br>type of:   |              |
|---|--------------|
| <b>If you enter:</b>  |              |
| John  | Text         |
| http://www.contoso.com  | Hyperlink    |
| You can use any valid Internet protocol prefix. For example, http://, https://, and mailto: are valid prefixes. |              |
| 1   | Number, Long |
| 50,000  | Integer      |
|   | Number, Long |

|   |                |
|---|----------------|
|   | Integer        |
| 50,000.99   | Number, Double |
| 50000.389   | Number, Double |
| 12/67   |                |
| The date and time formats recognized are those of your user locale. | Date/Time      |
| December 31, 2006   | Date/Time      |
| 10:50:23  | Date/Time      |
| 10:50 am  | Date/Time      |
| 17:50   | Date/Time      |
| \$12.50   | Currency       |
| The currency symbol recognized is that of your user locale.         |                |
| 21.75   | Number, Double |
| 123.00%   | Number, Double |
| 3.46E+03  | Number, Double |

### Table relationships

Although each table stores data about a different subject, tables in a database usually store data about subjects that are related to each other. For example, a database might contain:

A customers table that lists your company's customers and their addresses.

A products table that lists the products that you sell, including prices and pictures for each item.

An orders table that tracks customer orders.

Because you store data about different subjects in separate tables, you need some way to tie the data together so that you can easily combine related data from those separate tables. To connect the data stored in different tables, you create relationships. A relationship is a logical connection between two tables that specifies fields that the tables have in common.

### Keys

Fields that are part of a table relationship are called keys. A key usually consists of one field, but may consist of more than one field. There are two kinds of keys:

#### Primary key

A table can have only one primary key. A primary key consists of one or more fields that uniquely identify each record that you store in the table. Often, there is a unique identification number, such as an ID number, a serial number, or a code, that serves as a primary key. For example, you might have a Customers table where each customer has a unique customer ID number. The customer ID field is the primary key of the Customers table. When a primary key contains more than one field, it is usually composed of pre-existing fields that, taken together, provide unique values. For example, you might use a combination of last name, first name, and birth date as the primary key for a table about people.

#### Foreign key

A table can also have one or more foreign keys. A foreign key contains values that correspond to values in the primary key of another table. For example, you might



have an Orders table in which each order has a customer ID number that corresponds to a record in a Customers table. The customer ID field is a foreign key of the Orders table.

The correspondence of values between key fields forms the basis of a table relationship. You use a table relationship to combine data from related tables. For example, suppose that you have a Customers table and an Orders table. In your Customers table, each record is identified by the primary key field, ID.

To associate each order with a customer, you add a foreign key field to the Orders table that corresponds to the ID field of the Customers table, and then create a relationship between the two keys. When you add a record to the Orders table, you use a value for customer ID that comes from the Customers table. Whenever you want to view any information about an order's customer, you use the relationship to identify which data from the Customers table corresponds to which records in the Orders table.

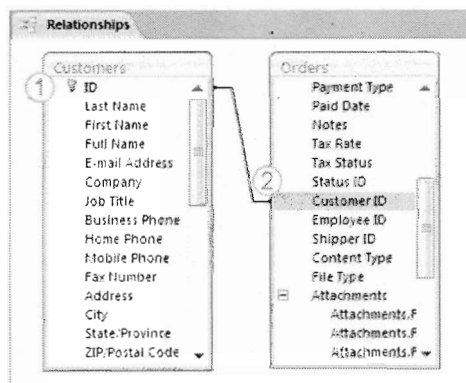


Figure 14. Benefits of using relationships

Keeping data separated in related tables produces the following benefits:

**Consistency** Because each item of data is recorded only once, in one table, there is less opportunity for ambiguity or inconsistency. For example, you store a customer's name only once, in a table about customers, rather than storing it repeatedly (and potentially inconsistently) in a table that contains order data.

**Efficiency** Recording data in only one place means you use less disk space. Moreover, smaller tables tend to provide data more quickly than larger tables. Finally, if you don't use separate tables for separate subjects, you will introduce null values (the absence of data) and redundancy into your tables, both of which can waste space and impede performance.

**Comprehensibility** The design of a database is easier to understand if the subjects are properly separated into tables.

Tip Plan your tables with relationships in mind. You can use the Lookup Wizard to create a foreign key field if the table that contains the corresponding primary key already exists. The Lookup Wizard creates the relationship for you.

## **Lessons 14-15.**

*Theme of lecture:* **Object-oriented approach. Create databases in the Delphi environment**

*Plan*

*Overview of Delphi's database features and capabilities*

*Delphi Database architecture*

Delphi enables you to create robust database applications quickly and easily. Delphi database applications can work directly with desktop databases like Paradox, dBASE, the Local InterBase Server, and ODBC data sources. The Delphi Client/Server edition also works with remote database servers such as Oracle, Sybase, Microsoft SQL Server, Informix, InterBase, and ODBC data sources. Delphi client applications can be scaled easily between mission critical network-based client/server databases, and local databases on a single machine.

Building a database application is similar to building any other Delphi application. This lecture assumes you understand the basic application development techniques covered in the Delphi User's Guide, including:

- Creating and managing projects;
- Creating forms and managing units;
- Working with components, properties, and events;
- Writing simple Object Pascal source code.

You also need to have a working knowledge of the Database Management System (DBMS) your Delphi database applications access, whether it is a desktop database such as dBASE or Paradox, or an SQL server.

## **Overview of Delphi's database features and capabilities**

A Delphi database application is built using Delphi database development tools, Delphi data-access components, and data-aware GUI components. A database application uses Delphi components to communicate with the Borland Database Engine (BDE), which in turn communicates with databases. The following figure illustrates the relationship of Delphi tools and Delphi database applications to the BDE and data sources:

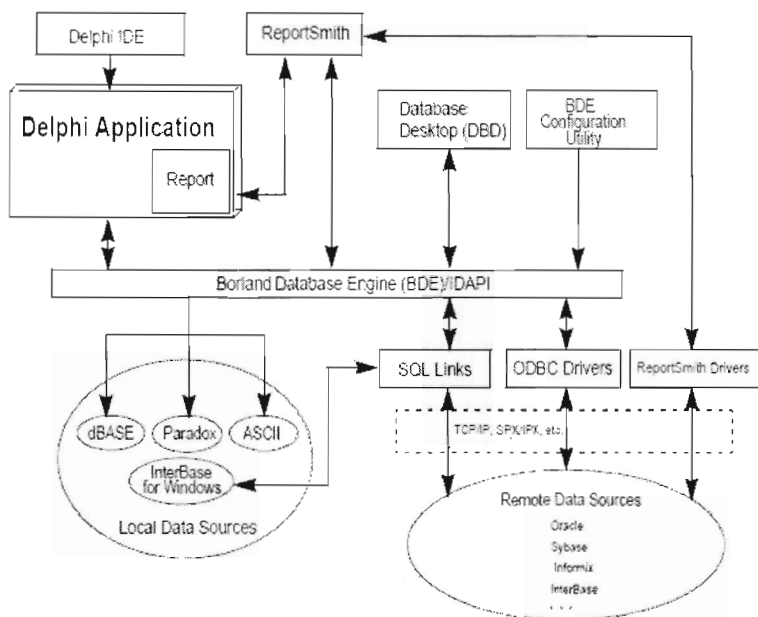


Figure 15. Delphi Database architecture

The following table summarizes Delphi's database features.

**Table 1.1** Database features summary

Table 11.

| Tool                    | Purpose  |
|-------------------------|--|
| Data Access components  | Access databases, tables, stored procedures, and custom component editors.                                   |
| Data Control components | Provide user interface to database tables.   |
| Database Desktop (DBD)  | Create, index, and query Paradox and dBASE tables, and SQL databases. Access and edit data from all sources. |
| ReportSmith             | Create, view, and print reports.   |

|                               |   |
|-------------------------------|---|
| Borland Database Engine (BDE) | Access data from file-based Paradox and dBASE tables, and from local InterBase server databases.  |
| BDE Configuration Utility     | Create and manage database connection Aliases used by the BDE.  |
| Local InterBase Server        | Provides a single-user, multi-instance desktop SQL server for building and testing Delphi applications, before scaling them up to a production database, such as Oracle, Sybase, Informix, or InterBase on a remote server. |
| InterBase SQL Link            | Native driver that connect Delphi applications to the Local InterBase Server.   |

These features enable you to build database applications with live connections to Paradox and dBASE tables, and the Local InterBase Server through the BDE. In many cases, you can create simple data access applications with these components and their properties without writing a line of code.

The BDE is built into Delphi components so you can create database applications without needing to know anything about the BDE. The Delphi installation program installs drivers and sets up configuration for Paradox, dBASE, and the Local InterBase Server, so you can begin working with tables native to these systems immediately. The BDE Configuration Utility enables you to tailor database connections and manage database aliases.

Advanced BDE features are available to programmers who need more functionality. These features include local SQL, which is a subset of the industry-standard SQL that enables you to issue SQL statements against Paradox and dBASE tables; low-level API function calls for direct engine access; and ODBC support for communication with other ODBC-compliant databases, such as Access and Btrieve.

Delphi includes Borland ReportSmith, so you can embed database report creation, viewing, and printing capabilities in Delphi database applications. Delphi also includes the Database Desktop (DBD), a tool that enables you to create, index, and query desktop and SQL databases, and to copy data from one source to another.

The Local InterBase Server is a single-user, multi-instance, 16-bit, ANSI SQL-compliant, Windows-based version of Borland's 32-bit InterBase SQL server that is available for Novell NetWare, Windows NT, and Unix.

SQL Links provide Delphi applications with SQL access to data residing on remote servers, including Sybase, Microsoft SQL Server, Oracle, and Informix.

When an SQL Link driver is installed, SQL statements are passed directly to the server for parsing and execution.

Table 11.1 Data Access Components

| Component   | Purpose  |
|-------------|--|
| TDataSource | Acts as a conduit between a TTable, TQuery, TStoredProc component and data-aware components, such as TDBGrid.  |
| TTable      | Retrieves data from a database table via the BDE and supplies it to one or more data-aware components through a TDataSource component. Sends data received from a component to a database via the BDE.                                       |
| TQuery      | Uses SQL statements to retrieve data from a database table via the BDE and supplies it to one or more data-aware components through a TDataSource component, or uses SQL statements to send data from a component to a database via the BDE. |
| TStoredProc | Enables an application to access server stored procedures. Sends data received from a component to a database via the BDE.   |
| TDatabase   | Sets up a persistent connection to a database, especially a remote database requiring a user login and password.   |
| TBatchMove  | Copies a table structure or its data. Can be used to move entire tables from one database format to another.   |
| TReport     | Enables printing and viewing of database reports through ReportSmith.  |

When building a database application, you place data access components on a form, then assign them properties that specify the database, table, and records to access. They provide the connection between a data source and Data Control components. At run time, after an application is built and compiled, data access objects are not visible, but are “under the hood,” where they manage data access.

Four data access components deserve special mention. Most forms provide a link to a database with a TTable or TQuery component (or through a user-defined component based on the normally hidden abstract class, TDataSet, of which TTable and TQuery are descendants). Other forms provide a link to a database with TStoredProc, also a descendant of TDataSet. In turn, all forms must provide a TDataSource component to link a TTable, TQuery, or TStoredProc component to data control components that provide the visible user interface to the data.

TTable, TQuery, (and TStoredProc, when it returns a result set) contain a collection of TField components. Each TField corresponds to a column or field in the table or query. TFields are created.

- Automatically, when TTable, TQuery, or TStoredProc are activated.
- At design time, using the Fields editor.

The TQuery component provides a tool for data access using SQL statements, such as a SELECT statement, to specify a set of records and a subset of columns from a table. TQuery is useful for building local SQL queries against Paradox and dBASE data, and for building client/server applications that run against SQL servers.

Every dataset that supplies a data control component must have at least one TDataSource component. TDataSource acts as a bridge between one TTable, TQuery, or TStoredProc component and one or more data control components that provide a visible user interface to data.

TTable and TQuery can establish connections to a database through the BDE, but they cannot display database information on a form. Data Control components

provide the visible user interface to data, but are unaware of the structure of the table from which they receive (and to which they send) data. A TDataSource component bridges the gap.

### 3. Laboratory sessions

#### Laboratory session №1

***The theme: Create a table in the MS Access 2007 Database Mangement system. Building Table Relationships***

***The aim of lesson:*** Explain the meaning of Table relationships in MS Access and develop the skills and abilities to create table; get to know Keys, Primary key, Foreign key and establish relationships between the tables in an Access 2007 database; increase students' interests in creating and programming database system.

#### ***Content of lesson***

##### **Table relationships**

Although each table stores data about a different subject, tables in a database usually store data about subjects that are related to each other. For example, a database might contain:

A customers table that lists your company's customers and their addresses.

A products table that lists the products that you sell, including prices and pictures for each item.

An orders table that tracks customer orders.

Because you store data about different subjects in separate tables, you need some way to tie the data together so that you can easily combine related data from those separate tables. To connect the data stored in different tables, you create relationships. A relationship is a logical connection between two tables that specifies fields that the tables have in common.

By now, you have set up the tables you need for your database, and created fields for those different tables. Relationships provide Access 2007 with the means to bring that information together for you when you need it.

This lesson explains how to establish relationships between the tables in an Access 2007 database. You will learn how to read and manipulate the relationship map. You will also learn about Primary and Foreign Keys, relationship types, and about referential integrity.

##### ***Establishing Relationships***

To establish a relationship between tables:

Click the Relationships command in the Show/Hide group on the Database Tools tab in the Ribbon.

NOTE: Tables must be closed in order to establish relationships.

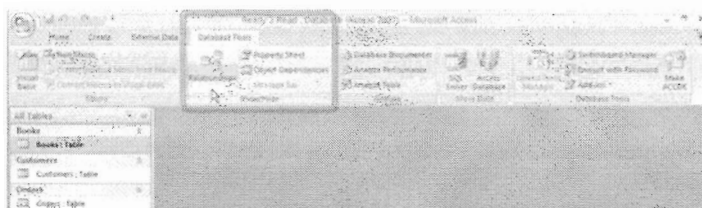


Figure 1.1 *Relationships Command*

When the Show Table dialog box appears:

Select each table name and click Add for the tables you want to relate.

When you are done, Close the Show Table dialog box



Figure 1.2 Show Table Dialog Box

You should now see a relationship map that contains all the tables that were selected.

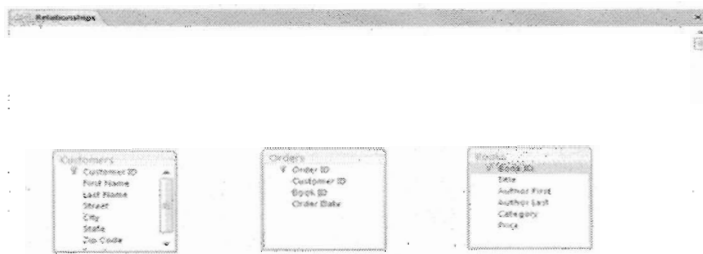


Figure 1.3 *Relationship Map*

### Moving Tables in the Relationship Map

To move a table that appears in the relationship map:

Place your mouse over the table you want to move.

Hold down the left mouse button, and drag the table to a new location.

Release the mouse button to drop the table in its new place.

### *Understanding the Relationship Map*

The relationship map lists all of the tables that were selected to relate, and all of the fields that were set up for that table previously. Notice that the first field has a key icon next to it. This is the Primary Key for the table.

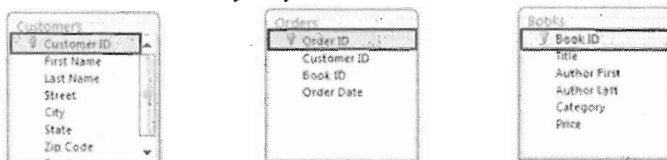


Figure 1.4 Primary Keys

### *Primary and Foreign Keys*

A Primary Key is the first field in each table of the database. You may recall that this field auto numbers by default, so that every record in the table has its own unique number to identify it. Access uses this number to quickly pull information together for you when you run queries or reports, which are covered later.

In the example above, the primary key for the Customers table is Customer ID, the primary key for the Orders table is Order ID, and the primary key for the Books table is Book ID.

A Foreign Key is a field that is the Primary Field in its own table, but that shows up in another table. If you look closely at the Orders table, the fields Customer ID and Book ID appear there, as well as in their own respective tables. These fields are the primary key in their own tables, but in the Orders table, they are considered Foreign Keys.

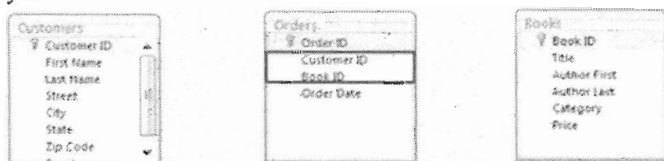


Figure 1.5 Foreign Keys

### *Relating Tables*

There are a few ways to establish relationships between tables:

- Using the Edit Relationships command located on the Design tab of the Ribbon
- Using the Drag and Drop method



Both methods give you the same end result, but the Drag and Drop method is much easier and saves you several steps.

*Relating Tables with the Drag and Drop Method*

It is easy to relate tables from the relationship map:

Select a field name from one table by holding down the left mouse button.

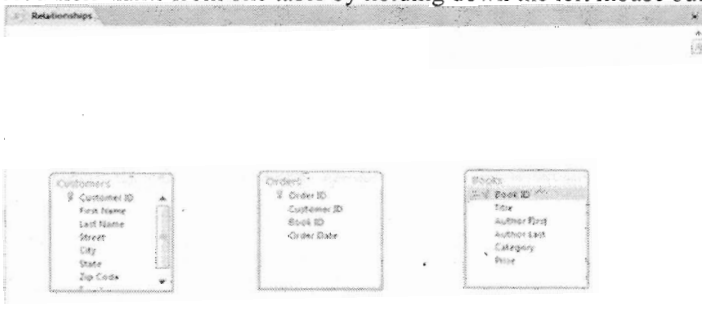


Figure 1.6 Relationship Map

Drag the field name from the one table to the other table in the desired relationship. Drop the first field name onto the field name that you want to relate by releasing the left mouse button. In the example above, we selected the Book ID field from the Books table, and dragged and dropped it on the Book ID field in the Orders table. The Edit Relationships dialog box appears.

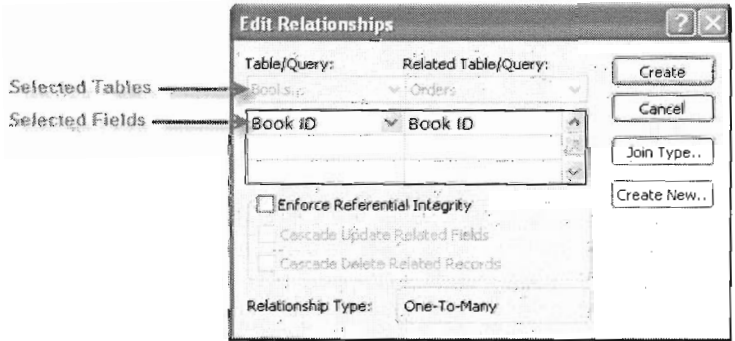


Figure 1.7 Edit Relationships Dialog Box

Select the Enforce Referential Integrity option. This option is explained in detail below. Click Create.

## *Understanding Types of Relationships*

Access 2007 allows for several different types of relationships. These include:

- One to One
- One to Many
- Many to Many

The relationship type you will come across most frequently, and the one created in our bookstore scenario, is the One to Many relationship.

### *One to Many*

The One to Many relationship means that data for that field will show up a single time in one table, but many times in the related table. For example, let's look at one of the book titles in our bookstore. The Book ID for that book should appear only once in the Books table, because that table lists every title that we stock. But it will probably appear many times in the Orders table, because we hope that it gets ordered by many people many times.

The symbols for the One to Many relationship look like this:

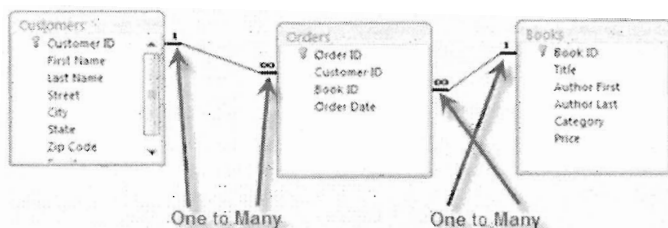


Figure 1.8 One to Many Relationships

### *Enforcing Referential Integrity*

In the Edit Relationships dialog box, an option to Enforce Referential Integrity appears. You should click Enforce Referential Integrity to make sure that we NEVER have an order for a book that doesn't appear in our Books table. Selecting this option tells Access to check for these things when someone is working with your data records.

### *Editing Existing Relationships*

Access 2007 allows you to edit relationships that already exist. This can be done using the Edit Relationships command on the Ribbon. However, a much simpler way



Access asks you which tables you want to ask questions about. First off, let's just take a look at the *Book* table. We can add the *Author* table later.

The real power in Access is the ability to easily deal with multiple tables at once, but one step at a time.

Click on *Book*, and click the *Add* button. The window stays open, so click the *Close* button.

Access presents you with the query design page.

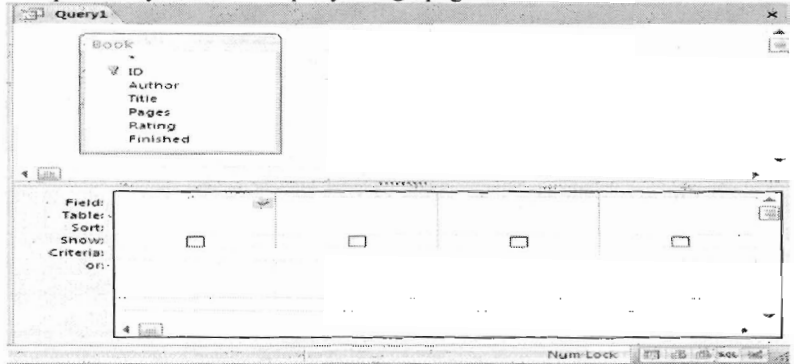


Figure 2.3

You can make some adjustments to the way the layout looks by dragging the central divider up or down, and there are shortcuts at the bottom right, in the status bar, that let you change the type of view you are using. More about those later.

The upper portion of the screen contains all of the included tables, with a list of the fields. The lower portion is where the questions are asked.

First, you need to choose which of the fields in the table you want to either ask questions about, or wish to include in the answer. To choose, double-click the field, or drag it to the grid below.

For our example we want to choose *Author*, *Title* & *Rating*.

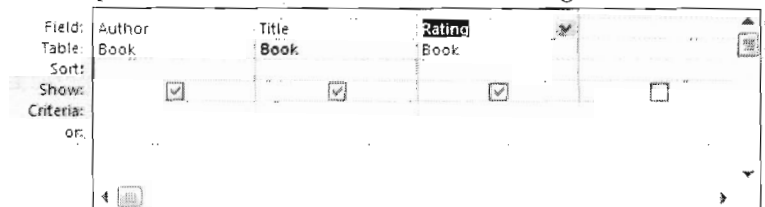


Figure 2.4

Once you have the fields in the grid, there are a lot of choices to make. They work line by line.

We have already chosen the *fields*, and the *tables* are added automatically. The next thing is the *sort*. To sort the books by rating for instance, click in the sort box for that column, and change the setting to *Ascending* or *Descending*.

You can sort by multiple columns. The priority is from left to right, so if you wanted to sort by *Rating* and then *Title*, you would need to rearrange the columns. You can just select by the grey bar at the top and drag them around.

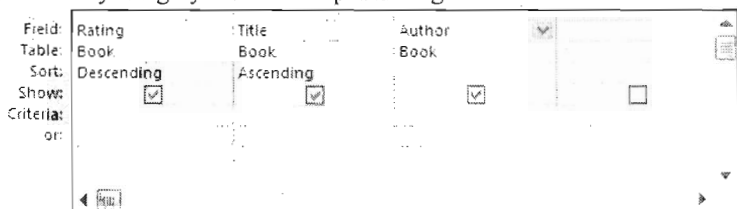


Figure 2.5

The *Criteria* row is a little more complex, but it's very easy to use once you get used to it. Criteria are specifications for which records (rows) from the table to show. And for the technical types reading, these are generally what is known as *AND* criteria. That is, *all* of the criteria need to be met. If instead you wish to use *OR* criteria (that means that *any* of the criteria can be met) then put the criteria on different rows. You can use as many rows as you wish from the one labelled *Criteria* downwards.

In our case, we want to only see books where the Title starts with “S”, and the rating is better than 2. The “S” criteria also includes what is known as a wild card. That is, the title needs to start the letter S, but anything at all is permitted after that.

Numeric criteria are allowed to be defined as limits, rather than specific values, so in the case we can use the “>” operator.

We could spend the whole day talking about criteria and wildcards, but let's move on.

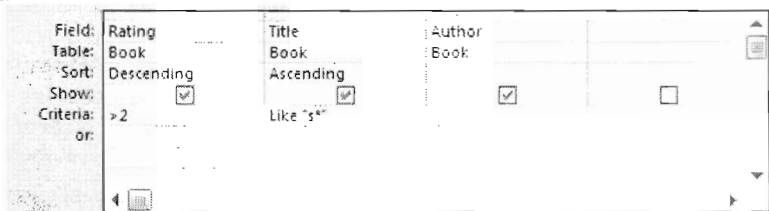


Figure 2.6 Criteria dialog window

Now that we have defined the question we wish to ask, we can pose it to Access, and view the answer. Click the View button in the ribbon or the datasheet view button in the status bar. You can flick back and forth between design and datasheet to make further changes to the query.

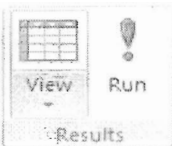


Figure 2.7

| Rating | Title                      | Author   |
|--------|----------------------------|----------|
| 4      | Seahorse in the Sky        | Cooper   |
| 4      | Star Beast                 | Heinlein |
| 4      | Stranger in a strange land | Heinlein |

Figure 2.8 Query table

It's important to note that as a general rule, the datasheet view from a query is live. That is, if you make changes to the query results then you make changes to the table data.

Finally, you can save the query for later. There is some confusion with this at times. Saving the query saves the question, not the answer. So that means that next time you run the query, if the data in the table has changed, then the answer might also change. There some other options to grab a snapshot of the data later on if necessary.

Click the *Save* button in the quick toolbar at the top left of the Access window. Remember that queries are saved along with the tables inside the one Access file on your hard drive.

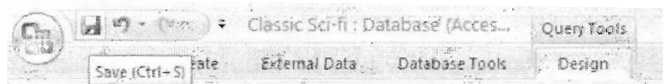


Figure 2.9 Save button

You often need to connect tables together in queries. For instance in this case, we could add the *Author* table so that we can make use of the information in it for sorting or further criteria.

As it happens, the lookup that we set up for the *Author* table means that we already have access to the *Author's* last name, but let's just pretend we wanted to sort the output by the author's first name instead. After all, these guys (or at least the few who are still alive) are friendly enough. Let's call them Isaac and Robert, right? Oh, hold on. Those two are dead.

To make this work, add the *Author* table to the query.

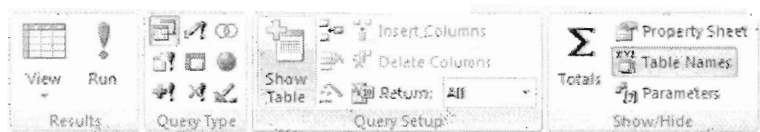


Figure 2.10 Query page

While in Design View, click the *Show Table* button and add the *Author* table to the grid.

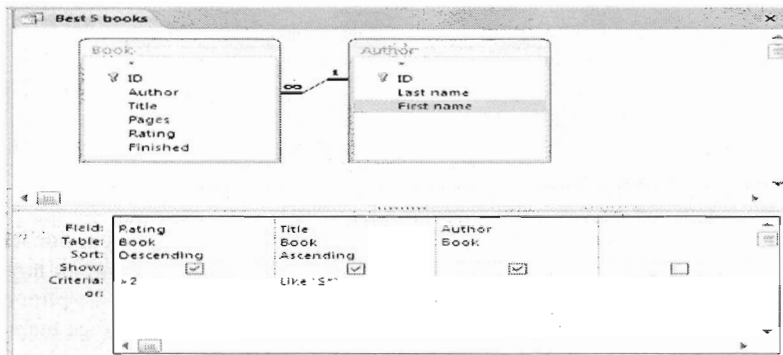


Figure 2.11. Table relationships

Because of the lookup that was set up, Access already knows how the tables are related, so you don't need to worry about that. Drag the *First Name* field down into the criteria block, then drag it off to the left so you can sort it as a priority.

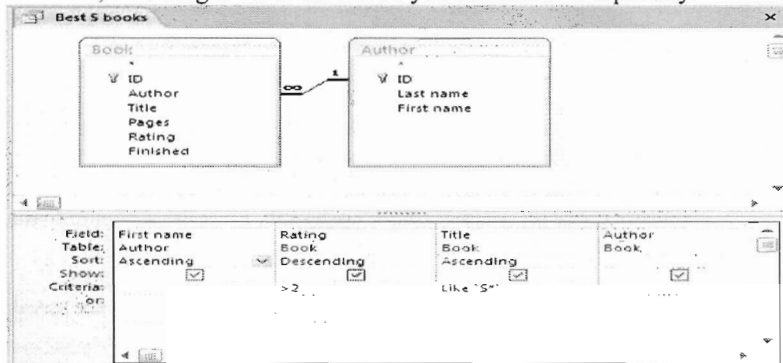


Figure 2.12. Click the Datasheet View button to see the difference.

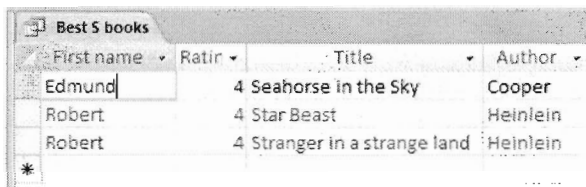


Figure 2.13 Query table as outcomes



Figure 2.14

### A Microsoft Access Tutorial on Query types

The query we just built, the default type in Access, is called a *Select* query. It's essentially a view of the answer to a question. The other types do a number of

specific things that might be useful later. I won't go into too much detail here, but some pointers might help.

Most of these other queries are what is known as *Action* queries. That is because they actually change data in tables. No changes are made until you click the *Run* button (the Datasheet view only previews the results) and you will be warned that changes are about to be made.

#### **Update**

An *update* query is used to make changes to the table data in one hit, rather than dealing with the records one by one. For instance, perhaps an author might change his name, or admit to having written a stack of books under a nom-de-plume. An update query would let you select the appropriate records and then change them all at once.

#### **Make Table**

A *Make Table* query works the same way as an Update, but puts the results in a new table. This might be useful where for some reason you need to maintain both sets of data separately.

#### **Append**

An *Append* query lets you select records from one table and add them to the end of another. The most common use for this is for archiving records from a main table to a secondary one.

#### **Delete**

A Delete query is extremely useful, but care needs to be taken with using it. This query lets you select some records from a table, and then delete them.

#### **Other**

The other types of query (Union, Cross-tab, Pass-through and Data Definition) are for advanced use, and I won't cover these here.

That's it for now, until I'm back with a post on Access Forms.

Let me know how it goes with queries, and whether there are any difficulties I can help with in the comments.

### **Laboratory session №3**

#### **The theme of lesson: Create forms in the MS Access Database Management system**

*The aim of lesson:* Explain the meaning of Forms in MS Access and develop the skills and abilities to create a different type of form, to learn use different components in the database; get to know type of forms in an Access 2007 database; increase students' interests in creating and programming database system.

#### ***Content of lesson***

#### **Open Your Access Database**





Figure 3.1 The main page

First, you'll need to start Microsoft Access and open the database that will house your new form.

In this example, we'll use a simple database I've developed to track running activity. It contains two tables: one that keeps track of the routes that I normally run and another that tracks each run. We'll create a new form that allows the entry of new runs and modification of existing runs.

### Select the Table for your Form

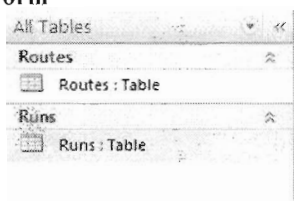


Figure 3.2 List of tables

Before you begin the form creation process, it's easiest if you pre-select the table that you'd like to base your form upon. Using the "All Tables" pane on the left side of the screen, locate the appropriate table and double-click on it. <BR><BR> In our example, we'll build a form based upon the Runs table, so we select it, as shown in the figure above.

### Select Create Form from the Access Ribbon

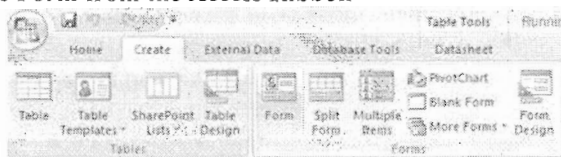
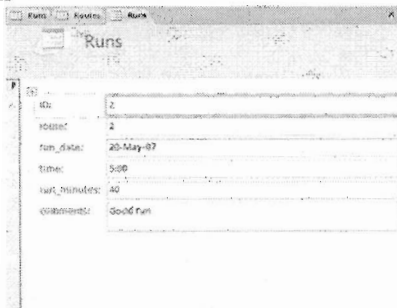


Figure 3.3. Form create window

Next, select the Create tab on the Access Ribbon and choose the Create Form button, as shown in the image above.

### View the Basic Form



|              |           |
|--------------|-----------|
| ID:          | 2         |
| IDREF:       | 2         |
| run_date:    | 20-May-07 |
| time:        | 5:00      |
| run_minutes: | 40        |
| comments:    | Good run  |

Figure 3.4 A simple Form window

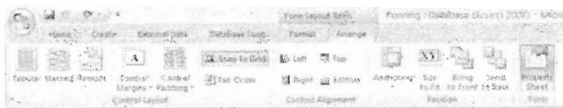


Figure 3.5. The form page

Access will now present you with a basic form based upon the table you selected. If you're looking for a quick and dirty form, this may be good enough for you. If that's the case, go ahead and skip to the last step of this tutorial on Using Your Form. Otherwise, read on as we explore changing the form layout and formatting.

### Arrange Your Form Layout

After your form is created, you'll be placed immediately into Layout View, where you can change the arrangement of your form. If, for some reason, you're not in Layout View, choose it from the drop-down box underneath the Office button.

From this view, you'll have access to the Form Layout Tools section of the Ribbon. Choose the Format tab and you'll see the icons shown in the image above.

While in Layout View, you can rearrange fields on your form by dragging and dropping them to their desired location. If you want to completely remove a field, right-click on it and choose the Delete menu item.

Explore the icons on the Arrange tab and experiment with the various layout options. When you're done, move on to the next step.

## Use Your Form

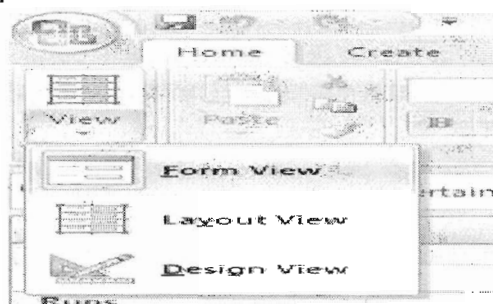
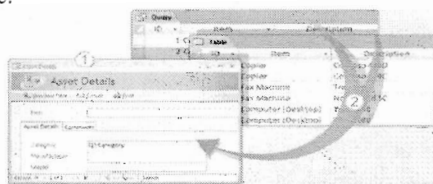


Figure 3.6. Form view window

You've put a lot of time and energy into making your form match your needs. Now it's time for your reward! Let's explore using your form. To use your form, you first need to switch into Form View. Click the drop-down arrow on the Views section of the Ribbon, as shown in the figure above. Select Form View and you'll be ready to use your form!

Once you're in Form View, you can navigate through the records in your table by using the Record arrow icons at the bottom of the screen or entering a number into the "1 of x" textbox. You can edit data as you view it, if you like. You can also create a new record by either clicking the icon at the bottom of the screen with a triangle and star or simply using the next record icon to navigate past the last record in the table.



Your new asset-tracking database is already saving time and money, but some of your coworkers don't like using Datasheet view to enter data. They find that a grid of columns and rows isn't that easy to use. Not a problem: Forms to the rescue.

A form is a screen that allows you to enter, change, and view the data in a database. Think of forms as windows into your data that help users understand and work with that data.

Let's take a quick look at what goes into a form:

1. Forms are made up of controls, such as text boxes, buttons, document tabs, and drop-down lists, grouped in a way that makes them easy to use and helps you get work done.

2. The controls in the form are usually bound, or connected, to the tables or queries in your database — but not always. For example, a control that displays your



way they did for the tables, also. The following picture shows the navigation buttons for a form.

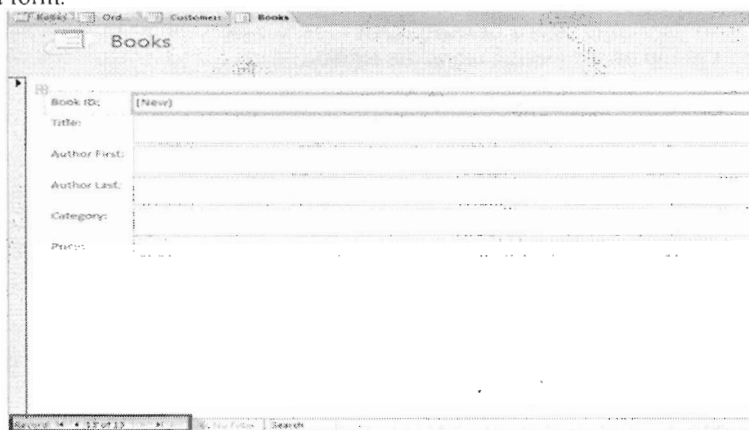


Figure 3.8 Form Navigation Buttons

### *To Add a Record using a Form*

To add a record to the database using a form:

Navigate to a new record, either by using the New Record navigation button, or the New command in the Records group on the Ribbon.

Then, simply add the new data.

Your data must be entered using an acceptable format. The acceptable formats were established when the field properties were set.

Finally, you must save the record.

Save by using either the Save command on the Ribbon, or by progressing to another record using the New (Blank) Record navigation button. Moving to a new record saves the most recently entered record. However, it may be necessary to refresh the table in Datasheet View to see the newest record.

### *To Edit Records using a Form*

Just like in a table, the database user can edit records from a form using the Find and Replace command. This command works exactly the same way for a form as it does in a table.

### *Creating a Drop Down List*

Using a drop down list on a form can increase the integrity of the data in the database, because drop down lists force the form user to select one of the pre-set options in the list to populate the field. These types of form controls are relatively easy to set up using the Combo Box.

## To Create a Drop Down List using a Combo Box Control

With the form opened in Design View, select the Combo Box command in the Controls group on the Design tab in the Ribbon

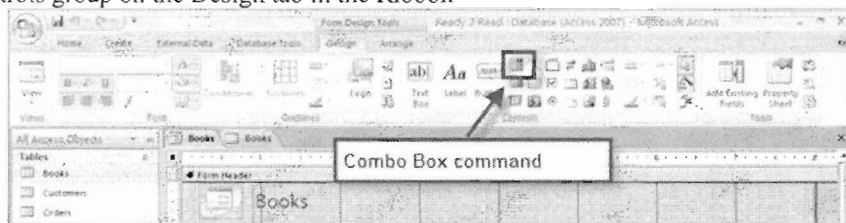


Figure 3.9 Combo Box Command

Drag and drop the Combo Box sizing tool to create the Combo Box where desired on the form.

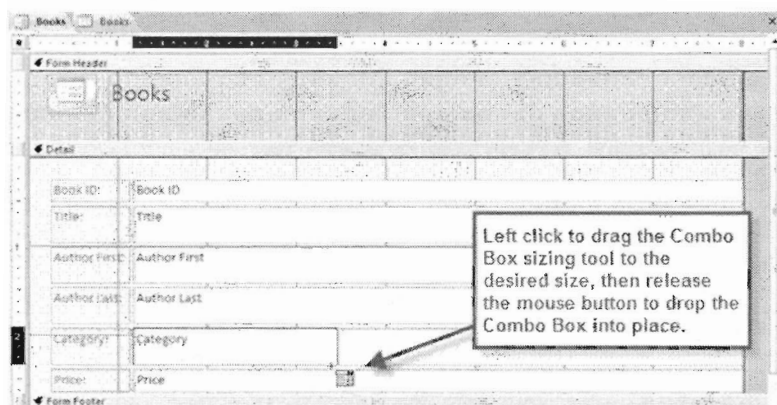


Figure 3.10 Combo Box Sizing Tool

The Combo Box Wizard appears.

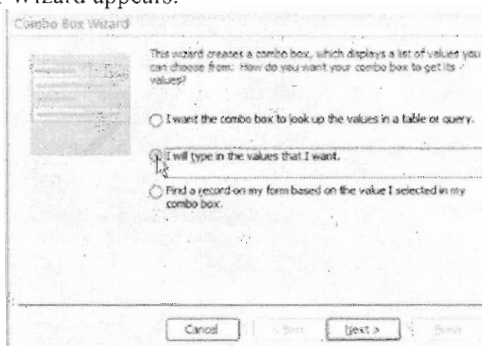


Figure 3.11 Combo Box Wizard

Choose the desired option from the Wizard, and click Next.

Because the middle option was selected in the example above, the Wizard progresses to the next step, which asks for the values to be typed into a small table.

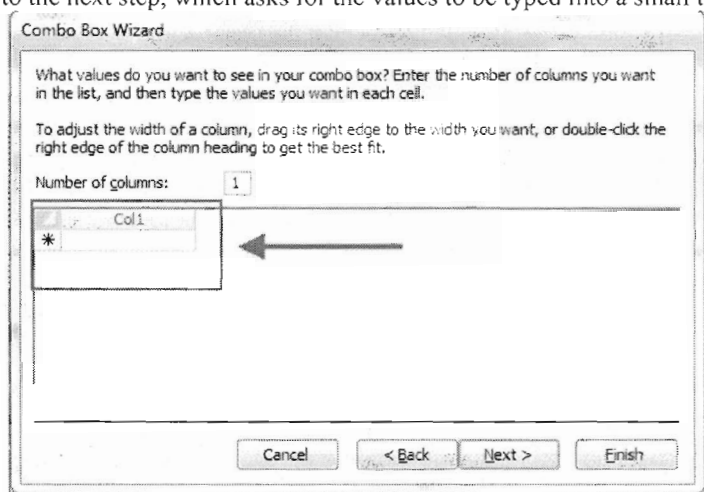


Figure 3.12 Combo Box Wizard

Next, the Wizard asks what to do with the entered values. Access can either remember the values for later use, or can populate a field with the entered values. Use the drop down list to select which field Access should use to store the values.

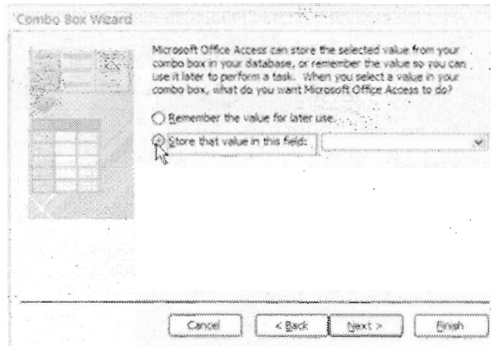


Figure 3.13 Combo Box Wizard

Once the desired option has been selected, click Next.

Finally, the Wizard gives the Combo Box a generic name, which may be meaningful to you later if there is ever a need to adjust the properties of this or another Combo Box. If you choose, give the Combo Box a name and click Finish.

Whatever name is entered will appear as a label on the form. This label may be deleted, if desired.

Switch to Form View to see how the Combo Box works. The Category drop down list appears on our Books form, as seen below.





Figure 3.14. Books Form Drop Down Category List

### **Laboratory session №4**

The theme of lesson: **Create a report in the MS Access Database**

#### **Management system**

*The aim of lesson:* Explain the meaning of reports in MS Access and develop the skills and abilities to create a different type of reports, to learn To Create a Report Based on a Table using the Report Command, Create a Report Based on a Query, to Group Items on a Report, to Format a Report in Layout View; get to know type of reports in an Access 2007 database; increase students' interests in creating and programming database system.

#### ***Content of lesson***

Now that you know how to use queries to analyze the data in a database, it is time to find out how to create a report that will make the data meaningful to someone else. This lesson will show you how to create a report using the Report command. It will also show you how to use grouping options and query limits to make the report easier to read, as well as identify several report formatting and layout options that can be set in Layout View. Finally, you will see how to use Print Preview and how to save the report.

#### *Using Reports To Make Data Meaningful to Others*

As you know, queries make the data in a database meaningful to you. Sometimes, though, you need to share that data with someone else. A report is an effective way to present your data using an attractive layout. The text can be formatted in an Access report like it can be in Word documents.

Microsoft Access 2007 offers tools that allow you to create and format a report. The Report Wizard walks you through the steps of creating a report. The Report command, however, is much easier to use, and all of the formatting options are still available to you in Layout View once the report is created. With these tools, you can create a report based on a table or on a query.

#### *Creating a Report Based on a Table*

One of the easiest ways to create a report is using a table as the source of the report. For example, in our bookstore scenario we have a table that lists all of the books in our inventory. We want to create a Book Price List report that lists all of the details for each book in our store's inventory. The Report command makes this incredibly easy, as it automatically includes every field in the source table in the report.

#### **To Create a Report Based on a Table using the Report Command**

To create a report based on a table using the Report command:

Choose the table you wish to use as the source of your report. To do that, you can either open the table, or just highlight the table name in the Navigation Pane. In our example, we used the open Books table to create the report.

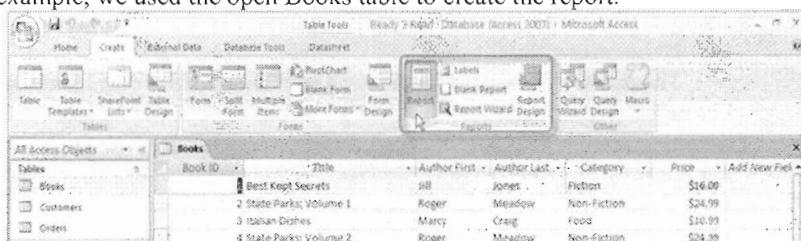


Figure 4.1 Report from Books Table

Select the Report command on the Create tab in the Ribbon, as seen above.

The report is automatically generated and includes every field in the table in order of their appearance in the table. This can be seen in the example below, which was created from the table above.

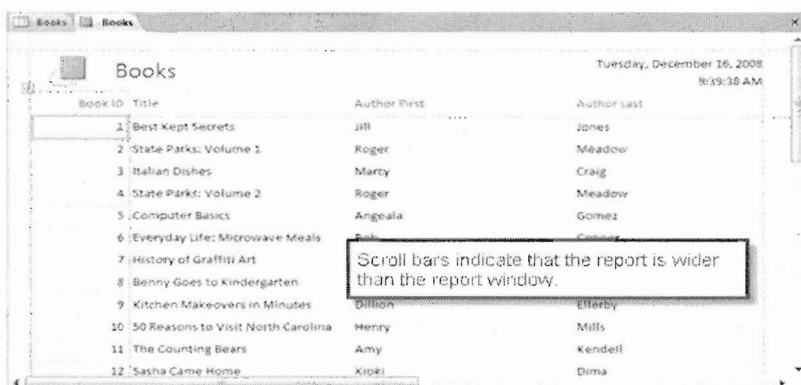


Figure 4.2 Book Price List

The layout and formatting of the report can be manipulated in Layout View.

### Creating a Report Based on a Query

Access 2007 can create a report using a query as the source, as well. The process for creating a report based on a query is identical to the process for creating a report based on a table that was outlined on the previous page. And just like when making a report from a table, every field and record that appears in the query results will appear on the report.

To Limit the Number of Records in a Report

It is possible to limit the number of records in a report, provided that the report was based on a query. The limit is set in the query itself, using the query design screen.

To limit the records returned in a query:

Open the query in Design View

Use the Return option in the Query Setup command group to set the number of records you want to see in the query results and the final report.

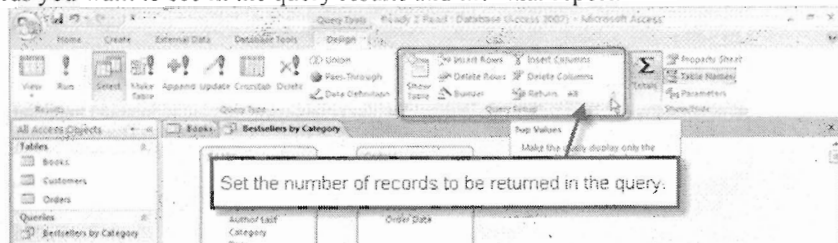


Figure 4.3 Return Limit

- Click Run! to make sure the query results look like you want the report to look.
- Create the report using the Report command on the Create tab
- Format the report as desired.

## Grouping Items on a Report

Grouping items on a report can make it much more readable. Microsoft Access 2007 offers a quick and easy way to add grouping to a report.

### To Add Grouping to a Report

To add a level of grouping to a report:

With the report open, select the Group & Sort command from the Grouping & Totals command group on the Format tab in the Ribbon.

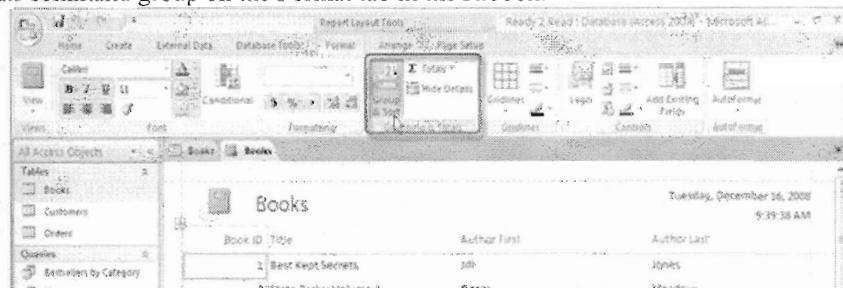


Figure 4.4 Grouping Command for Reports

This opens a Group, Sort, and Total dialog box in the lower portion of the window.

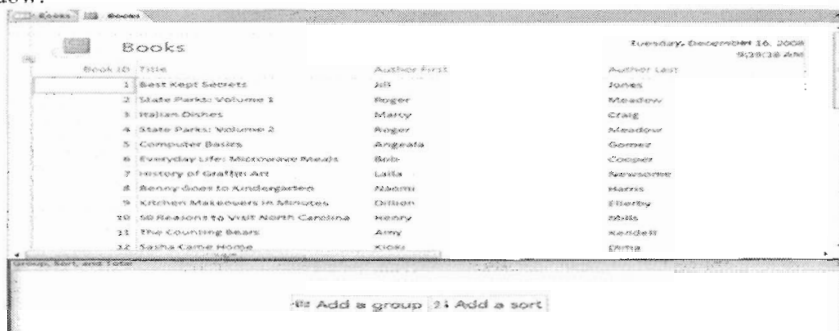


Figure 4.5 Group, Sort, and Total Dialog Box

In the Group, Sort, and Total dialog box, select Add a group.

Select the field you wish to group by from the drop down list. We chose to group our list by Category.

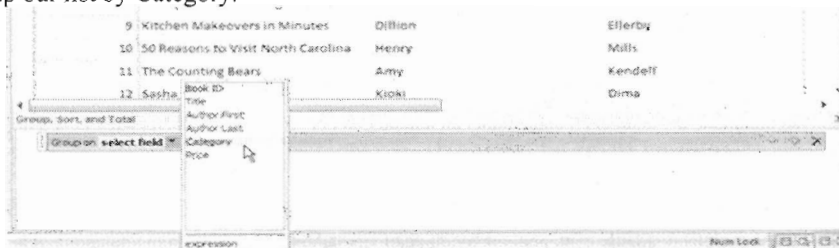


Figure 4.6 Grouping Drop Down List

When you release the mouse button, the report will now appear with items grouped. Our report is grouped on Category now, as seen below.

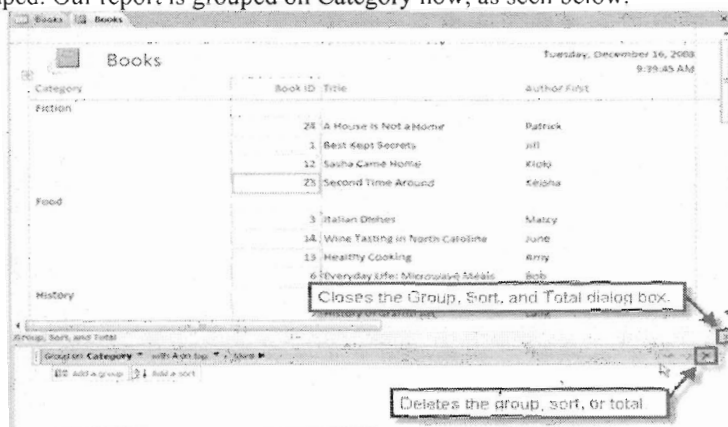


Figure 4.7 Books Price List Grouped on Category

The Group, Sort, and Total dialog box will remain open until you close it.

## Formatting a Report in Layout View

Access opens the created report in Layout View, so that you can easily make modifications. In Layout View, you can change the look of your report in many different ways, including:

- deleting columns and other report elements
- moving and resizing columns
- adding a logo
- changing the title and other text on the report headings
- applying a report style with AutoFormat
- modifying the page layout

### To Delete a Column or Other Report Element

To delete a column or other report element:

Highlight the element by clicking on it.

Hit the Delete button on your keyboard.

### To Move a Column or Other Report Element

To move a column or other report element:

Highlight the element by clicking on it.

Drag and drop the element to a new location on the report.

### To Re-size a Column or Other Report Element

To re-size a column or other report element:



When the highlight appears, type in the new title.

### *To Modify Text in Report Headings*

If you don't like the standard font face and size that Access used to create your report, you can modify them using common Microsoft Office text formatting commands. You can modify the size, font face, font color, alignment, and much more. They all work basically the same way:

Highlight the text you want to change

Select the formatting option you wish from the lists that appear when you click on a command.

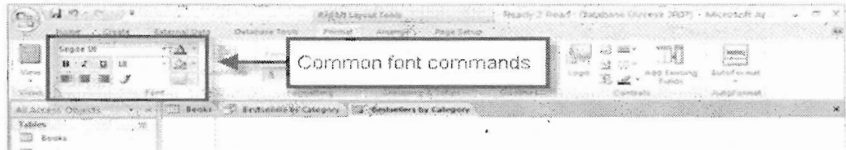


Figure 4.11 Text Commands for Report

The change appears when you release the mouse button.

### *To Apply an AutoFormat Style*

Like with forms, Microsoft Access 2007 offers a variety of report styles in the AutoFormat command. To apply a style:

Click on the AutoFormat command in the Ribbon.



Figure 4.12 AutoFormats for Report

Select a format from the drop down list. The change is applied instantly.

### *To Change the Page Layout*

When a report is created, it opens in Layout View, like the one in the picture below. The dotted lines are showing where the edge of the page will be in Report View.

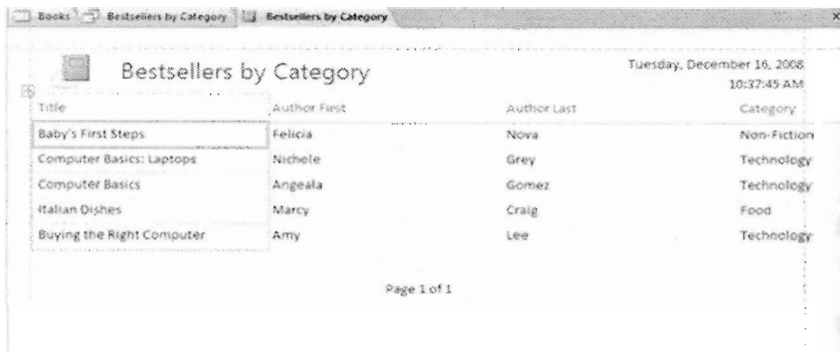


Figure 4.13 Report in Layout View

To change the page layout options:  
Switch to Print Preview using the Views command in the Ribbon.

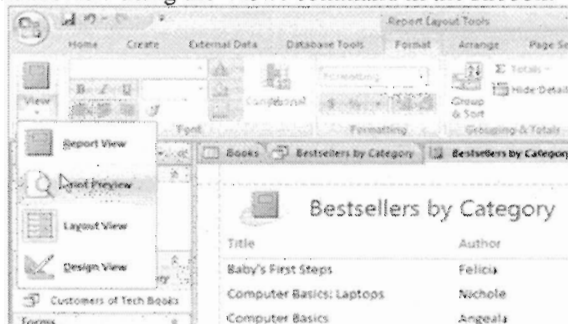


Figure 4.14 Print Preview Command

Select the layout option you wish to alter from the Page Layout command group on the Ribbon.

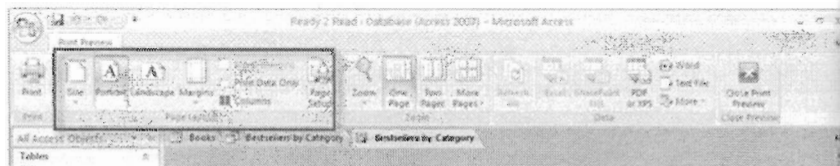


Figure 4.15 Page Layout Options for Report

All of the standard Microsoft page layout options are available, including: Page Layout Option      Description



Margins To set the margins for narrow, wide, or normal  
Orientation To select either a Portrait or Landscape orientation  
Size To set the paper size

## Saving a Report

When you have created and modified a report and try to close it, Microsoft Access 2007 will prompt you to name and save the report. If you do not ever need this report again, you need not save it. However, if you think you may want to publish it again, it is best to save.

To Save a Report

As with all Access objects, to save a report:

Right click on the report tab.

Choose Save from the list that appears.

When the Save as dialog box opens, give the report a name.

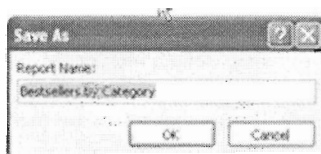


Figure 4.16 Save as Dialog Box

Click OK.

## Laboratory session №5

The theme of lesson: **Create a macros in the MS Access Database**

### Management system

*The aim of lesson:* Explain the meaning of macros in MS Access and develop the skills and abilities to create a different type of macroses; get to know type of reports in an Access 2007 database; increase students' interests in creating and programming database system.

### Content of lesson

#### Macros

In Microsoft Office Access 2007, you can define a macro to execute just about any task you would otherwise initiate with the keyboard or the mouse. This article introduces you to the unique power of macros in Office Access 2007 — their ability to automate responses to many types of events without forcing you to learn a

programming language. The event might be a change in the data, the opening or closing of a form or a report, or even a change of focus from one control to another.

#### *Uses of macros*

Macros are particularly useful for building small, personal applications or for prototyping larger ones. Office Access 2007 provides various types of macro actions that you can use to automate your application. With macros, you can:

- Open any table, query, form, or report in any available view or close any open table, query, form, or report.

- Open a report in Print Preview or Report view or send a report directly to the printer.

- Send the output data from a report to a Rich Text Format (.rtf) file, a Windows Notepad (.txt) file, or a Snapshot (.snp) format file. You can then open the file in Microsoft Word or Notepad.

- Execute a select query or an action query. You can base the parameters of a query on the values of controls in any open form.

- Include conditions that test values in a database, a form, or a report and use the results of a test to determine what action runs next.

- Execute other macros or execute Visual Basic functions. You can halt the current macro or all macros, cancel the event that triggered the macro, or quit the application.

- Trap errors caused during execution of macro actions, evaluate the error, and execute alternate actions.

- Set the value of any form or report control or set selected properties of forms and form controls.

- Emulate keyboard actions and supply input to system dialog boxes.

- Refresh the values in forms, list box controls, and combo box controls.

- Apply a filter to, go to any record in, or search for data in a form's underlying table or query.

- Execute any of the commands on any of the Access Ribbons.

- Move and size, minimize, maximize, or restore any window within the Access workspace when you work in multiple-document interface mode.

- Change the focus to a window or to any control within a window or select a page of a report to display in Print Preview.

- Display informative messages and sound a beep to draw attention to your messages. You can also disable certain warning messages when executing action queries.

- Rename any object in your database, make another copy of a selected object in your database, or copy an object to another Access database.

- Delete objects in your database or save an open object.

- Import, export, or attach other database tables or import or export spreadsheet or text files.

- Start an application and exchange data with the application using Dynamic Data Exchange (DDE) or the Clipboard. You can send data from a table, query, form, or

report to an output file and then open that file in the appropriate application. You can also send keystrokes to the target application.

Consider some of the other possibilities for macros. For example, you can make moving from one task to another easier by using command buttons that open and position forms and set values. You can create very complex editing routines that validate data entered in forms, including checking data in other tables. You can even check something like the customer name entered in an order form and open another form so that the user can enter detailed data if no record exists for that customer.

### *Working with the Macro design window*

When creating a macro, begin by opening the database with which you are working.

On the Create tab, in the Other group, click the arrow on the New Object button, and click Macro from the list of three options. (The top half of the New Object button displays the last type of new object created —Macro, Module, or Class Module. If you see the Macro icon in the top half of the New Object button, you can also click that button to begin creating a new macro.) Access opens a new Macro window similar to the one shown in Figure 1. In the upper part of the Macro window, you define your new macro; and in the lower part, you enter settings, called arguments, for the actions you've selected for your macro. The upper part shows at least two columns, Action and Comment. You can view all five columns shown in Figure 1 by clicking the Macro Names, Conditions, and Arguments buttons in the Show/Hide group on the Design tab.

You can cause the Macro Name and Condition columns to appear automatically for any new macro by selecting the Names Column and Conditions Column check boxes under Show In Macro Design in the Display section of the Advanced category in the Access Options dialog box.



Figure 5.1 A new Macro window displays columns where you can define your macro.

Notice that the area at the lower right displays a brief help message. The message changes depending on where the insertion point is located in the upper part of the window.

You can always press F1 to open a context-sensitive Help topic.

In the Action column, you can specify any one of the 70 macro actions provided by Office Access 2007. If you click any box in the Action column, an arrow appears at the right side of the box. Click this arrow to open a list of the macro actions, as shown in Figure 5.2



Figure 5.2 The list of macro actions displays 70 actions you can use in Office Access 2007.

The Macro Builder has been redesigned in Access 2010 to make it even easier to create, modify, and share Access Macros. Watch a video or try Office 2010!

## Saving your macro

You must save a macro before you can run it. Click the Save button on the Quick Access Toolbar, or click the Microsoft Office Button and then click Save. When you do so, Access opens the dialog box shown in Figure 3. Enter the name for this macro, and click OK to save it.

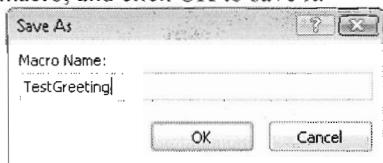


Figure 5.3. Enter a name for your macro in the Save As dialog box.

## Testing your macro

You can run some macros directly from the Navigation Pane or from the Macro window because they don't depend on controls on an open form or report. If your

macro does depend on a form or a report, you must link the macro to the appropriate event and run it that way. However you run your macro, Access provides a way to test it by allowing you to single step through the macro actions.

To activate single stepping, right-click the macro you want to test in the Navigation Pane, and then click Design View on the shortcut menu. This opens the macro in the Macro window. Click the Single Step button in the Tools group on the Design tab. Now when you run your macro, Access opens the Macro Single Step dialog box before executing each action in your macro. In this dialog box, you'll see the macro name, the action, and the action arguments.

Try this procedure with the macro you just created. Open the Macro window, click the Single Step button, and then click the Run button in the Tools group on the Design tab. The Macro Single Step dialog box opens, as shown in Figure 4. The Macro Single Step dialog box also shows you the result of testing your condition.

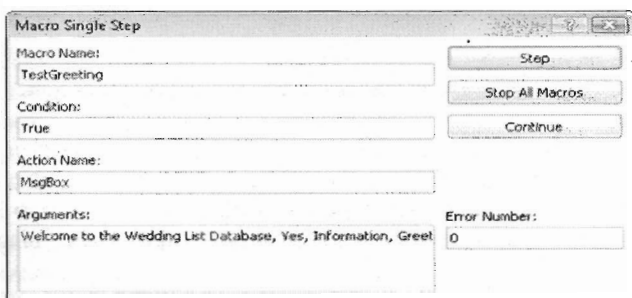


Figure 5.4. The Macro Single Step dialog box allows you to test each action in your macro.

If you click the Step button in the dialog box, the action you see in the dialog box will run, and you'll see the dialog box opened by your MsgBox action with the message you created, as shown in Figure 5. Click the OK button in the message box to dismiss it. If your macro had more than one action defined, you would have returned to the Macro Single Step dialog box, which would have shown you the next action. In this case, your macro has only one action, so Access returns you to the Macro window.



Figure 5.5 Access displays the dialog box you created by using the MsgBox action in your macro.

If Access encounters an error in any macro during the normal execution of your application, Access first displays a dialog box explaining the error it found. You then see an Action Failed dialog box, which is similar to the Macro Single Step dialog box, containing information about the action that caused the problem. At this point, you can click only the Stop All Macros button. You can then edit your macro to fix the problem.

Before going further, you might want to return to the Macro window and click the Single Step button again so it's no longer selected. Otherwise you'll continue to single step through every macro you run until you exit and restart Access or click Continue in one of the Single Step dialog boxes.

As you dig deeper into macros you'll find out how to include multiple actions and define condition checking so that different actions are performed depending on the values in your forms or reports.

### **Laboratory session №6,7**

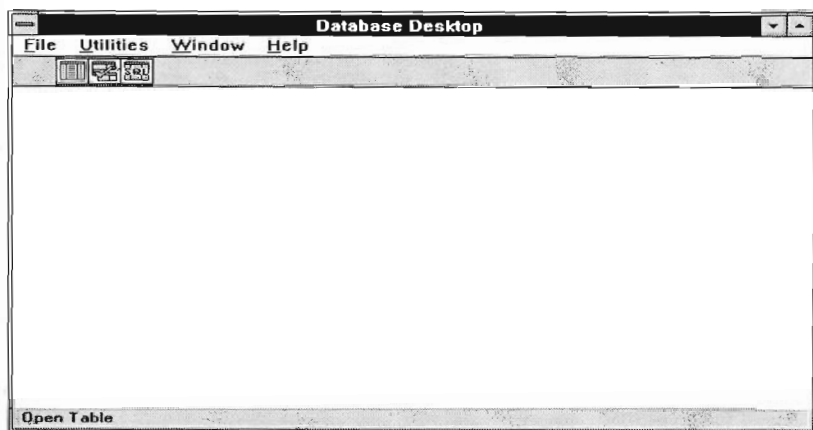
The theme of lesson: **Create databases in the Delphi environment.**

*The aim of lesson:* Explain the meaning of databases in the Delphi environment and develop the skills and abilities to create an alias and database; get to know type of components; increase students' interests in creating and programming database system.

### ***Content of lesson***

Database Desktop provides an easy way to create, restructure, and query tables to help you develop database applications with Delphi. You can use Database Desktop either as a standalone application on a single computer running Windows or as a multiuser application on a network.

To start Database Desktop, double-click the Database Desktop icon in the Delphi program group or choose File|Run in the Program Manager and run DBD.EXE. Database Desktop has several command-line options that let you control its configuration. The first time you start Database Desktop, the Database Desktop window opens. All Database Desktop windows are opened in and contained by this window.



The Database Desktop application window

Files you open in Database Desktop appear in their own type of windows. Tables appear in Table windows, queries appear in Query windows, and SQL statements appear in the SQL Editor.

In Database Desktop you work with three types of files: QBE queries, .SQL files, and tables. Other files are created automatically by Database Desktop. For a list of file extensions used by Database Desktop, search for "file-name extensions" in the keyword list in Database Desktop Help, and choose the topic "Types of Files."

#### *Opening files*

To open a QBE query, SQL statement, or table, follow these steps:

- 1 Choose File|Open.
- 2 Choose the type of file to open—QBE query, SQL statement, or table.
- 3 Specify the file to open.

The Select File dialog box appears. For detailed information on the Select File dialog box, search for "Select File dialog box" in the keyword list in Database Desktop Help.

*Note* To access tables stored on a network, you must specify the location of the network control file. You do this by running the BDE Configuration Utility; double-click the BDE Configuration Utility icon in the Delphi program group.

Configuration Utility for more details.

#### *Setting up a working directory*

The working directory is where Database Desktop looks first for files. The Working Directory setting controls what files are listed in File|Open and File|Save dialog boxes. So, for example, if you want to open C:\DBD\SAMPLES\BOOKORD.DB, make C:\DBD\SAMPLES your working directory so that you see BOOKORD.DB when you choose File|Open|Table.

To specify a working directory, choose File|Working Directory, then type the path to the directory. SQL You cannot set your working directory to an alias on a remote server.

### Setting up a private directory

You should store temporary tables, such as Answer, in a private directory so they do not get overwritten by other users or applications. Choose File|Private Directory to establish a private directory.

Files stored in your private directory are listed in File|Open and File|Save dialog boxes, preceded by :PRIV: . Private directory files are visible and available to you, but not to other network users.

### Aliases

You can assign an alias as a shorthand for a directory using the Alias Manager dialog box. For example, if you have a collection of tables and queries in one directory (called C:\DBD\PROJECTS\CUSTLIST), you can specify the alias :MYWORK: rather than type the entire path. Using aliases, you can avoid typing long path names, and you can use the Path list in File|Open and File|Save dialog boxes to list files in any directory for which you have defined an alias. To create an alias, choose File|Aliases. For information on creating, changing, or removing an alias, search for "aliases" in the keyword list in Database Desktop Help.

### Creating tables

This section describes tables and discusses how to create and restructure Paradox, dBASE, and SQL tables in Database Desktop. A database is an organized collection of information or data. An address book is an example of a database. It organizes data about people into specific categories: names, phone numbers, and addresses. In a relational database, the data is organized into tables. Each row of a table contains information about a particular item; this is called a record. Each column contains one piece of the information that makes up a record; this is called a field.

| Table : CUSTOMER.DB |          |              |      |                      |              |       |       |
|---------------------|----------|--------------|------|----------------------|--------------|-------|-------|
| CUSTOMER            | Cust ID  | Last Name    | Init | Street               | City         | State | Zip   |
| 1                   | 1,386.00 | Aberdeen     | F    | 45 Utah Street       | Washington   | DC    | 20032 |
| 2                   | 1,388.00 | Svenvald     | I    | Gouvenment House     | Reykjavik    |       |       |
| 3                   | 1,784.00 | McDougal     | L    | 4950 Pullman Ave NE  | Seattle      | WA    | 98105 |
| 4                   | 2,177.00 | Bonnefemme   | S    | 128 University Drive | Stanford     | CA    | 94323 |
| 5                   | 2,579.00 | Chavez       | L    | Cypress Drive        | Palm Springs | FL    | 32938 |
| 6                   | 2,779.00 | Fahd         | S    | The Palace           | Riyadh       |       |       |
| 7                   | 3,128.00 | Elsbeth, III | R    | 1 Hanover Square     | London       |       |       |
| 8                   | 3,266.00 | Hanover      | A    | 15 State Street      | Dallas       | TX    | 75043 |
| 9                   | 3,271.00 | Massey       | C    | 29 Aragona Drive     | Oxon Hill    | MD    | 29902 |
| 10                  | 3,771.00 | Montaigne    | L    | 30 Tauton Drive      | Bellevue     | WA    | 98004 |
| 11                  | 4,277.00 | Matthews     | R    | P. O. Box 20336      | Albuquerque  | NM    | 87234 |

To create a new table,

1 Choose File|New|Table. Or right-click the Open Table tool bar button, and choose New. The Table Type dialog box appears.



2 If you want a table type other than Paradox for Windows, click the arrow next to the list box and select from the drop-down list.

3 Choose OK. The Create Table dialog box appears. This dialog box may have a slightly different appearance for different table types, but it will function the same. For a step-by-step description of creating a table, search for "creating tables" in the keyword list in Database Desktop Help, and choose the topic "Creating a New Table."

#### *Adding, deleting, and rearranging fields*

You can add, delete, and rearrange fields in the Field Roster.

For detailed information on changing fields, search for "fields" in the keyword list in Database Desktop Help."

#### *Specifying field type*

To specify the field type in the Create Table dialog box,

1 Select the Type column of the field you want.

2 Type the symbol (or name, for SQL tables) for the field type or select from the dropdown list. You can use the list in two ways:

- Right-click the Type column again and click to select the field type.
- Press Spacebar to see the list, then choose the field type.

For information on field types and sizes, search for "field types" in the keyword list in Database Desktop Help, and choose the topic for the type of table you are using (Paradox, dBASE, or SQL).

### **Laboratory session №7**

The theme of lesson: **Create a table. Create the table structure by programming. Table component.**

*The aim of lesson:* Explain the meaning of databases in the Delphi environment and develop the skills and abilities to create a tables; get to know type of components; increase students' interests in creating and programming database system.

#### ***Content of lesson***

MASTAPP aliases. All TTable and TQuery components used in example code in this chapter set their DatabaseName property to DBDEMOS. In contrast, the complete demo in the MASTAPP directory does the following to facilitate porting:

1 The main form (MAIN.PAS) has a TDatabase component with its AliasName property set to DBDEMOS and DatabaseName property set to MAST.

2 All datasets on all forms have their Database properties set to MAST. Now all forms can use a different BDE alias simply by changing the main form's TDatabase component's AliasName property.

#### *Building a single-table form*

The steps in this section show how to use the Database Form Expert to build a singletable form. Of course, anything the expert does, you can do by hand, but the expert saves a lot of time.

| Property            | Value    | Remarks   |
|---------------------|----------|---|
| <i>Active</i>       | False    | When <i>Active</i> is False, data-aware controls do not display data at design time. To make controls display data at design time, set <i>Active</i> to True. |
| <i>DataBaseName</i> | MAST     | MAST is an alias that points to where the table resides. Use aliases, not hard-coded paths, to make applications portable and easy to upsize.                 |
| <i>Name</i>         | Form1    | Use the Object Inspector to change names.   |
| <i>TableName</i>    | PARTS.DB | Tells the component which table to link to.   |

What to do:

1 Choose Help|Database Form Expert to open the Form Expert.

2 Specify a table, fields, and field layout as shown in the following figure. The Form Expert creates the form.

3 Press F9 to run the form. Click the navigator control buttons to move through the records in the table.

The form contains one *TDataSource* component. The expert links it to the *TTable* component by setting the properties listed in the following table. A *TDataSource* component acts as a bridge between one dataset component (*TTable* in this case) and one or more data-aware controls that provide a visible interface to data.

Important *TDataSource* properties for a single-table form (continued)

| Property        | Value          | Remarks  |
|-----------------|----------------|--|
| <i>AutoEdit</i> | True (default) | When <i>AutoEdit</i> is True, Delphi puts the <i>TDataSource</i> into Edit state automatically when the user changes a value in a linked control. To make a <i>TDataSource</i> read-only, or to control when to enter Edit state, set <i>AutoEdit</i> to False. <i>TTable</i> component linked to a table <i>TDataSource</i> linked to the <i>TTable</i> |
| <i>DataSet</i>  | Table1         | Specifies which <i>TTable</i> (or <i>TQuery</i> ) is supplying the data. <i>Name DataSource1</i> Use the Object Inspector to change names.   |
| <i>DataSet</i>  | Table1         | Specifies which <i>TTable</i> (or <i>TQuery</i> ) is supplying the data. <i>Name DataSource1</i> Use the Object Inspector to change names.   |

Important *TDBGrid* properties for a single-table form

| Property | Value | Remarks |
|----------|-------|---------|
|----------|-------|---------|

|                   |                    |   |
|-------------------|--------------------|---|
| <i>DataSource</i> | <i>DataSource1</i> | Links the <i>DBGrid</i> control to a <i>TDataSource</i> component, which supplies the data. |
|-------------------|--------------------|---|

The expert does more than create a form and components. It also generates a line of code to open the table at run time in case you do not activate the table at design time. For example, the expert creates *TTable* components with the *Active* property set to *False*.

That's why the various *TDBEdit* controls aren't displaying data. You could set *Active* to *True*, and the controls would display data from the first record. Instead, the Form Expert generates the following code to open the table at run time.

```
procedure TEditPartsForm.FormCreate(Sender: TObject);
begin
  Table1.Open;
end;
```

This code is hooked to the form's *OnCreate* event, so Delphi executes it before creating the form. As a result, the table is opened before the form is displayed.

## Laboratory session №8

### The theme of lesson: **Building a master-detail form**

*The aim of lesson:* Explain the meaning of databases forms in the Delphi environment and develop the skills and abilities to create a forms; get to know type of components; increase students' interests in creating and programming database system.

### *Content of lesson*

The steps in this section show how to use the Database Form Expert to build a form containing two tables: a master table and a detail table, linked one-to-many. This form is the basis for the CUSTORD.DFM form in MASTAPP. The master table is CUSTOMER.DB and the detail table is ORDERS.DB. You can access both tables using the MAST alias. The expert links these tables and creates components to display data for one customer at a time, and for each customer, to display many orders.

#### *What to do*

When you use the Database Form Expert, building a master-detail form is much like building a single-table form.

- 1 Choose Help|Database Form Expert to open the Form Expert.
- 2 In the first panel, specify a master-detail form that uses *TTable* objects.
- 3 In subsequent panels, specify the master table (CUSTOMER.DB), fields (use them all), and field layout (grid).

- 4 Specify the detail table (ORDERS.DB), fields (all), and field layout (grid).
- 5 Specify fields to link the master and detail tables as shown in the following figure, then tell the expert to create the form.

### *How it works*

The expert builds a master-detail form much as it builds a single-table form. It creates TTable components and TDataSource components for the master table and the detail table and links them to the underlying data by setting properties.

The expert creates controls to display the data from each table, and sets properties to link them to the corresponding TDataSource component. The expert also creates a TDBNavigator control linked to the master table.

### *Building a one-many-many form*

This section describes how to build a form that displays data from three tables linked one-many-many. For example, one customer may place many orders, and each order may have many items. Use the Database Form Expert to create a master-detail form linking the Customer table to the Orders table as described on page 24. Then, to display and link the Items table, place components and set properties by hand. You can also use the techniques described here to build a master-detail form from scratch.

### *What to do*

- 1 Choose Help|Database Form Expert to open the Form Expert.
- 2 In the first panel, specify a master-detail form that uses TTable objects.  
In subsequent panels, specify the master table (CUSTOMER.DB), fields (use them all), and field layout (horizontal).
- 4 Specify the detail table (ORDERS.DB), fields (all), and field layout (horizontal).
- 5 Specify fields to link the master and detail tables: choose CustNo for the IndexFieldNames property and link the CustNo fields in each table.
- 6 Tell the expert to create the form.
- 7 Move components around to make room at the bottom of the form. You may have to change the Align property of some controls from alClient to alNone.
- 8 Place a TTable component, a TDataSource component, and a TDBGrid component as shown in the following figure. (The TTable and TDataSource are on the Data Access components page; the TDBGrid is on the Data Controls page.) These components represent the third table in the one-many-many link. In this example it's the Items table.
- 9 To create the link, set properties of these new components as shown in the following table.

## Laboratory session №9

The theme of lesson: **Create query in Delphi environment. SQL language**

*The aim of lesson:* Explain the meaning of databases in the Delphi environment and develop the skills and abilities to create a tables; get to know type of components; increase students' interests in creating and programming database system.

### *Content of lesson*

TQuery is a dataset component. In addition, TQuery enables Delphi applications to issue SQL statements to a database engine (either the BDE or a server SQL engine). The SQL statements can be either static or dynamic, that is, they can be set at design time or include parameters that vary at run time.

#### *When to use TQuery*

For simple database operations, TTable is often sufficient and provides portable database access through the BDE. However, TQuery provides additional capabilities that TTable does not.

Use TQuery for:

- Multi-table queries (joins).
- Complex queries that require sub-SELECTs.
- Operations that require explicit SQL syntax.

TTable does not use SQL syntax;

TQuery uses SQL, which provides powerful relational capabilities but may increase an application's overall complexity. Also, use of nonstandard (server-specific) SQL syntax may decrease an application's portability among servers; for more information, see Chapter 6, "Building a client/server application."

#### *How to use TQuery*

To access a database, set the DatabaseName property to a defined BDE alias, a directory path for desktop database files, or a file name for a server database. If the application has a TDatabase component, DatabaseName can also be set to a local alias that it defines.

To issue SQL statements with a TQuery component:

- Assign the TQuery component's SQL property the text of the SQL statement. You can do this:
  - At design time, by editing the TQuery's SQL property in the Object Inspector, choosing the SQL property, and entering the SQL statements in the String List Editor dialog box. With Delphi Client/Server, you can also use the Visual Query Builder to construct SQL syntax.
  - At run time, by closing any current query with Close, clearing the SQL property with Clear, and then specifying the SQL text with the Add method.
- Execute the statement with the TQuery component's Open or ExecSQL method.

#### *Use*

Open for SELECT statements. Use ExecSQL for all other SQL statements. The differences between Open and ExecSQL are discussed in a subsequent section.

- To use a dynamic SQL statement, use the Prepare method, provide parameters and then call Open or ExecSQL. Prepare is not required, but will improve performance for dynamic queries executed multiple times.

#### **Creating the query text**

You can enter the SQL text for a TQuery at design time by double-clicking on the SQL property in the Object Inspector, or choosing the ellipsis button. The String List Editor opens, enabling you to enter an SQL statement.

Choose OK to assign the text you enter to the SQL property of the query. Choose Load to include text from a file or Save to save the text to a file.

To specify SQL text at run time, an application should first close the query with Close and clear the SQL property with Clear. For example,

*Query1.Close; {This closes the query}*

*Query1.SQL.Clear; {This clears the contents of the SQL property}*

It is always safe to call Close—if the query is already closed, the call will have no effect. Use the SQL property's Add method to add the SQL statements to it. For example,

*Query1.SQL.Add("SELECT \* FROM COUNTRY");*

*Query1.SQL.Add("WHERE NAME = 'ARGENTINA'");*

An application should always call Clear before specifying an SQL statement. Otherwise, Add will simply append the statements to the existing one.

Note The SQL property may contain only one complete SQL statement at a time. In general, multiple statements are not allowed. Some servers support multiple statement "batch" syntax; if the server supports this, then such statements are allowed.

You can also use the LoadFromFile method to assign the text in an SQL script file to the SQL property. For example,

*Query1.SQL.LoadFromFile('C:\MYQUERY.TXT');*

### **Laboratory session №10**

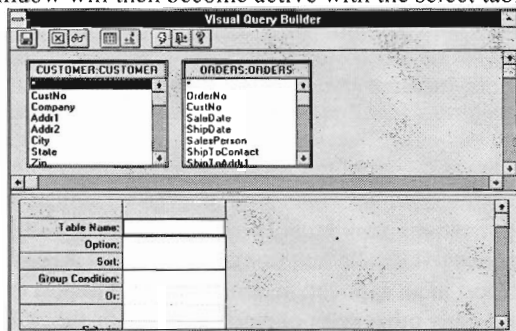
#### **The theme of lesson: Create a query by using SQL Builder**

*The aim of lesson:* Explain the meaning of databases in the Delphi environment and develop the skills and abilities to create a query **by using SQL Builder**; get to know type of query components; increase students' interests in creating and programming database system.

#### **Content of lesson**

Delphi Client/Server includes a Visual Query Builder that enables you to construct SQL

SELECT statements visually. To invoke the Visual Query Builder, right click on a TQuery component and select Run Visual Query Builder. A dialog box prompts you to select the database to work with; select the desired database and choose OK. Another dialog box will prompt you to enter the tables you want to query; select the desired tables, choosing Add after each, and then choose Close. The Visual Query Builder window will then become active with the select tables.



For information on how to use the Visual Query Builder, refer to its online Help. After

you have you constructed a query and exited the Visual Query Builder, the SQL statement you constructed will be entered in the SQL property of the selected TQuery component.

Executing a query

At design time, you can execute a query by changing its Active property in the Object Inspector to True. The results of the query will be displayed in any data controls connected to the Query component (through a data source).

At run time, an application can execute a query with either the Open or the ExecSQL methods. Use Open for SQL statements that return a result set (SELECT statements). Use

ExecSQL for all other SQL statements (INSERT, UPDATE, DELETE, and so on). For example,

**Query1.Open;** {Returns a result set}

If the SQL statement does not return a cursor and a result set from the database, use ExecSQL instead of Open. For example,

**Query1.ExecSQL;** {Does not return a result set}

If you don't know at design time whether a query will return a result set, use a try...except block with Open in the try part and ExecSQL in the except part.

*The UniDirectional property*

Use the UniDirectional property to optimize access to a database table through a TQuery component. If you set UniDirectional to True, you can iterate through a table

more quickly, but you will only be able to move in a forward direction. UniDirectional is False by default.

### Laboratory session №11

#### The theme of lesson: **Create report in Delphi environment. Create a simple report**

*The aim of lesson:* Explain the meaning of databases in the Delphi environment and develop the skills and abilities to create a **simple report**; get to know type of report components; increase students' interests in creating and programming database system.

#### *Content of lesson*

Delphi applications can include reports created with ReportSmith with the TReport component. TReport appears on the Data Access component page. To incorporate a report in an application, simply add a TReport component to the desired form as you would any other component. Then specify the name of the report (created with ReportSmith) and other report parameters with properties of the component.

Designing reports with ReportSmith is described in Creating Reports. You can invoke ReportSmith at design time by double-clicking on a TReport component, or on the ReportSmith icon in the Delphi program group. Specify the name of an existing report in the ReportName property and the directory in the ReportDir property. To load ReportSmith Runtime and print the specified report, use the Run method (i.e., Report1.Run). The report prints on the default printer defined in ReportSmith. Preview is a Boolean property that specifies whether to print the report or just display it: If set to True, Run will display the report onscreen only; if set to False, Run will print the report.

The AutoUnload property specifies whether to automatically unload the ReportSmith Runtime executable after a report is run. Generally, if an application runs one report at a time, AutoUnload should be True. If an application is going to run a series of reports, then AutoUnload should be False.

The InitialValues property is of type TStringList and specifies the report variables to use with the report. Each line specifies a report variable as follows:

REPORTVAR = value

Some important methods of TReport are listed in the following table:

| Method          | Purpose   |
|-----------------|---|
| <i>Run</i>      | Run a report.   |
| <i>RunMacro</i> | Send a macro command to ReportSmith.                                    |
| <i>Connect</i>  | Preconnect the report to a database, so it does not prompt for login. e |



|                     |   |
|---------------------|---|
| <i>SetVariable</i>  | Change a specific report variable.                              |
| <i>ReCalcReport</i> | Run a report again. Use this when report variables have changed |

What to do: printing reports

This example show how to print a ReportSmith report that lists MAST customers.

1 Choose File|New Form to open the Browse Gallery, then choose Blank form and click OK to create a blank form.

2 Place a TReport component anywhere in the form. (The TReport component is on the Data Access components page.)

3 Use the Object Inspector to set the TReport component's properties as shown in the following table.

4 Place a button control anywhere in the form.

5 Double-click the button to open a code window, then write code to handle its OnClick event, as follows.

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
```

```
    Report1.Run;
```

```
end;
```

6 Press F9 to run the form.

7 Click the button to run the report. If the TReport component's Preview property is set to True, Delphi displays the report onscreen; if Preview is False, Delphi sends the report to the printer.

#### *How it works*

The Run method of TReport opens the run-time version of ReportSmith, which prints or displays a report as specified by the TReport component's properties. When you're designing a form, you can double-click a TReport component to open the full version of ReportSmith and build a report.

## **5. Themes for Tutorial lessons**

### **Tutorial №1,2**

#### **Theme: Create a Database system**

1. Open the database and establish a relationship between the Books table and the Orders table using the Drag and Drop method.
2. Establish a relationship between the Customers table and the Orders table using the
3. Edit Relationships command on the Ribbon.
4. Edit an established relationship by double-clicking the link.
5. Explore the options and settings in the Edit Relationships dialog box.
6. Move the tables around in the relationship map.
7. Benefits of using relationships

#### **Tutorial №3,4,5**

##### **Theme: Creating a Form**

1. Scroll through the customer records using the Customers form.
2. Create a basic Books form using the Form command.
3. Use the new Books form to enter and save the following data:

Title: The Secret Streets of Savannah

Author: Amy Little

Category: Travel

Price: \$34.99

Title: Cars and Trucks

Author: Jonathon Bradley

Category: Kids

Price: \$14.99

Using the Combo Box command, create a drop down list on the Books form for the following Categories:

Fiction

Non-Fiction

Kids

History

Technology

Home & Garden

Travel

Food

4. Add another record to the database using the Books form with the drop down Category selector.
5. Use the Find and Replace command to change the price of all books that are \$14.99 to be \$16.99.

#### **4. Self-control**

##### **Self-control №1,2**

##### **Theme: Create a Database system**

Create a database in the MS Access, which consists of 3-4 tables and create two users, privileges distribute according to roles in a database. Enter data into tables, with 3-5 lines.

Tasks examples:

1. The University. Tables: students, teachers, subjects. Roles: The student — can borrow, but can't make change; the Teacher — can look through and make changes into a database.
2. Shop. Tables: buyers, sellers, the goods, purchases (connects buyers with the goods and the sellers who have sold the goods). Roles: The buyer — sees the goods and the purchases; Seller.
3. Bank. Tables: clients, contracts (between the client and the operator, on concrete type of a contribution), types of deposits. Roles: Client, Operator.
4. Library. Tables: library tickets, books, orders of books (comparison of books and library tickets). Roles: Librarian, Reader.
5. Cellular operator. Tables: clients, records of conversations (record about the client, conversation time, a tariff), customer accounts. Roles: Client, Operator.
6. Real estate agency. Tables: real estate, client, agents, lease contract. Roles: Client, Agent.
7. School. Tables: pupils, teachers, estimates. Roles: Pupils, Teachers.
8. Car-care center. Tables: clients, cars, masters. Roles: Client, Master.
9. Railway cash desk. Tables: routes, trains, tickets. Roles: The cashier, the Manager — has full authority.
10. Support service. Tables: objects, employees, demands for works. Roles: Manager, Technician.

##### **Self-control №3,4,5**

##### **Theme: Creating a Query**

Expand earlier made database to 5 — 6 tables. Prepare some difficult inquiries. Think, what inquiries — most often used.

Tasks examples:

1. University. Tables: groups, chairs, books in subjects. Query: to find books, in subjects which it passes.

2. Shop. Tables: commodity groups, departments. Query: on departments to find buyers by departments.
3. Bank. Tables: bank offices, operators. Query: to find clients that with whom the chosen operator concluded contracts.
4. Library. Tables: authors, genres. Query: to define favorite genres of the chosen reader.
5. Cellular operator. Tables: tariffs, areas (where the operator serves). Query: to display the conversations, which area was chosen.
6. Real estate agency. Tables: departments of agents, rent prices. Query: how many rent pays that a chosen client.
7. School. Tables: classes, subjects. Query: to find pupils by marks.
8. Car-care center. Tables: spare parts, malfunctions. Query: Which spare parts are required for chosen client.
9. Railway cash desk. Tables: clients, orders. Query: to display bookings by a known train.
10. Support service. Tables: malfunctions, expendables. Query: to display the expendables that used by chosen object.

### **Self-control №6,7,8**

#### **Theme: Create a form**

Index fields so that it led to increase of speed of a database (at large volumes of data). Explain the decision with reason.

1. Create representations for query which could be useful for real work of an educational database.
2. University. Representation a form: to find books, in subjects which it passes.
3. Shop. Representation: on to find buyers by departments.
4. Bank. Representation: to find clients with whom the chosen operator concluded contracts.
5. Library. Representation: to define favourite genres of the chosen reader.
6. Real estate agency. Representation: how many rent pays that a chosen client.
7. School. Representation: to find pupils by marks.
8. Car-care center. Representation: Which spare parts are required for chosen client.
9. Railway cash desk. Representation: to display bookings by a known train.
10. Support service. Representation: to display the expendables that used by chosen object.

## Self-control №9,10,11

**Theme: Design a Database in the environment Delphi which you carried out on the previous Self-control lessons.**

Tasks examples:

1. The University. Tables: students, teachers, subjects. Roles: The student — can borrow, but can't make change; the Teacher — can look through and make changes into a database.
2. Shop. Tables: buyers, sellers, the goods, purchases (connects buyers with the goods and the sellers who have sold the goods). Roles: The buyer — sees the goods and the purchases; Seller.
3. Bank. Tables: clients, contracts (between the client and the operator, on concrete type of a contribution), types of deposits. Roles: Client, Operator.
4. Library. Tables: library tickets, books, orders of books (comparison of books and library tickets). Roles: Librarian, Reader.
5. Cellular operator. Tables: clients, records of conversations (record about the client, conversation time, a tariff), customer accounts. Roles: Client, Operator.
6. Real estate agency. Tables: real estate, client, agents, lease contract. Roles: Client, Agent.
7. School. Tables: pupils, teachers, estimates. Roles: Pupils, Teachers.
8. Car-care center. Tables: clients, cars, masters. Roles: Client, Master.
9. Railway cash desk. Tables: routes, trains, tickets. Roles: The cashier, the Manager — has full authority.
10. Support service. Tables: objects, employees, demands for works. Roles: Manager, Technician.

### **Tutorial №6,7,8**

**Theme: Create a Form, using constructor**

1. Open the Orders Form in Layout View and change the form by:
2. Adding a picture using the Logo command.
3. Giving the form a new Title.
4. Modifying some of the text on the form.
5. Applying one of the AutoFormat options.
6. Moving or resizing an object on the form.
7. Applying a border to an object on the form.

### **Tutorial №9,10,11**

**Theme: Create a Query**

1. Plan a query to find out which customers order a certain category of books.
2. Use the Query Design command to set up the query.
3. Run the query and view your results.
4. Save the query.
5. Modify the query to hide a field.

### **Tutorial №12,13,14,15**

**Theme: Create a Report**

1. Create a report based on a table.
2. Create a report based on query.
3. Modify the layout of a report by:
4. Resizing or moving columns
5. Deleting report elements
6. Giving the report a new title
7. Applying an AutoFormat style to the report
8. Use groups, sorts, or totals in a report

### **6 Control questions**

**1 Microsoft Access 2007 is a:**

- A. word processing software package
- B. slideshow presentation software package
- C. relational database management software package \*
- D. spreadsheet software package

**2 The Ribbon is Microsoft Access 2007's new menu system.**

- A. True \*
- B. False

3 To open Access 2007 database objects, you must use the:

- A. Navigation Pane\*
- B. Quick Access Toolbar
- C. Ribbon
- D. None of the above

4 A table is a collection of:

- A. Forms
- B. Queries
- C. Records\*
- D. Reports

5 When designing a database in Access, what is the first thing you need to determine in order to begin?

- A. The number of tables needed
- B. The number of fields needed
- C. The name of the tables
- D. The purpose of the database \*

6 When adding fields to a table, which "view" would you work in?

- A. Datasheet view
- B. Design view
- C. Print view
- D. Both A and B \*

7 In a relationship map, you have a primary key in each table. What is a primary key?

- A. A field that is the primary field in its own table, but shows up in another table
- B. The second field in each table of a database
- C. The first field in each table of the database \*
- D. The last field in each table of the database

8 Data validation is the process by which Access tests the data that is being entered into the database, to make sure it is in an acceptable or valid format.

- A. True \*
- B. False

9 When you are entering or editing a record, a pencil icon appears in the active record row.

- A. True \*
- B. False

10 To encourage database users to save records, you can include a \_\_\_\_\_ on the data entry form.

- A. Toggle Button
- B. Action Button \*
- C. Combo Box
- D. List Box

11 You add an AutoFormat to a form or report in \_\_\_\_\_ view.

- A. Datasheet View
- B. Design View
- C. Print Preview
- D. Layout View \*

12 Access allows you to use or include a sort function in:

- A. Queries
- B. Reports

- C. Tables
- D. All of the above \*

13 An ascending sort lists records:

- A. A to Z \*
- B. Z to A
- C. Highest to Lowest
- D. None of the above

14 When you set up a filter, you are telling Access to look for records that match a specific \_\_\_\_\_ that you set.

- A. number limit
- B. criteria \*
- C. total
- D. command

15 An advanced filter is a small query run on \_\_\_\_\_.

- A. several tables at once
- B. only one table \*
- C. several reports at once
- D. only one report

16 You can NOT modify a query once it has been saved.

- A. True
- B. False

17 To include an addition function to a query in the query design view:

- A. Use the Function command
- B. Use the Addition command
- C. Use the Totals command \*
- D. Use the Sort command

18 To limit the number of records that appear in a report, you must:

- A. set the Limit option in Layout View
- B. set Page Layout options
- C. delete records from your tables
- D. set the Return option in the query design \*

#### 7 Literature

##### The main literature:

1. C.J. Date. An Introduction to Database Systems. 8th Edition. Addison-Wesley, 2000.-1072 p.
2. G. Hansen, Hansen, J. // Databases: Design and Management. Translation, Moscow 1999.
3. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г.// Базы данных. Учебное пособие, 2002.
4. Халыкова К.З. //Маліметтер коры және банкі. Оқу құралы. 2004 ж. -160 б.
5. Stojanovic, T.A. (2007) Guidelines for Implementing Local Information Systems at the Coast.COREPOINT and Cardiff University, Cardiff. <http://corepoint.ucc.ie/Cpages/outputs.htm>

##### Additional literature:

1. Джеймс Р. Грофф, Пол Н. Вайнберг SQL - полное руководство. Перевод, Киев 1999г.
2. Наумова М.А. Системы управления базами данных и знаний. Высшая школа, 1992г.
3. Диго С.М. Проектирование баз данных. Финансы и статистика, 1990г.



4. Мейс С. Использование языка SQL обеспечивает возможность объединения баз данных. В мире компьютеров, № 2, 1988.
5. Дж.Мартин Организация баз данных в вычислительных системах. М, Наука 1980г.
6. Фаронов В.В. Программирование баз данных на Delphi 7. Учебный курс, 2005.
7. Золотова С.И. Практикум по Access. Финансы и статистика, Москва, 2000.
8. <http://www.freebsd.org/doc/handbook/network-servers.html>

## 8 Glossary

A **database** is an organized collection of data.

The term **database system** implies that the data are managed to some level of quality (measured in terms of accuracy, availability, usability, and resilience)

A **field** contains an item of data; that is, a character, or group of characters that are related.

A **record** is composed of a group of related fields.

A **database file** is sometimes called a table. A file may be composed of a complete list of individuals on a mailing list, including their addresses and telephone numbers.

**Data management systems** are used to access and manipulate data in a database. A database management system is a software package that enables users to edit, link, and update files as needs dictate.

In order to track and analyze data effectively, each record requires a unique identifier or what is called a **key**. The key must be completely unique to a particular record just as each individual has a unique social security number assigned to them.

**Database architectures** can be distinguished by examining the way application logic is distributed throughout the system. Application logic consists of three components: Presentation Logic, Processing Logic, and Storage Logic.

In computing, a **file server** is a computer attached to a network that has the primary purpose of providing a location for shared disk access, i.e. shared storage of computer files (such as documents, sound files, photographs, movies, images, databases, etc.) that can be accessed by the workstations that are attached to the same computer network.

Since the crucial function of a file server is **storage**, technology has been developed to operate multiple disk drives together as a team, forming a disk array.

File servers generally offer some form of system **security** to limit access to files to specific users or groups. In large organizations, this is a task usually delegated to what is known as directory services such as openLDAP, Novell's eDirectory or Microsoft's Active Directory

A **database management system** is a set of software programs that allows users to create, edit and update data in database files, and store and retrieve data from those database files. Data in a database can be added, deleted, changed, sorted or searched all using a DBMS.

**Database design** is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database.

Database designs also include ER (Entity-relationship model) diagrams. An ER diagram is a diagram that helps to design databases in an efficient way.

Attributes in **ER diagrams** are usually modeled as an oval with the name of the attribute, linked to the entity or relationship that contains the attribute.