

NEAR EAST UNIVERSITY



Faculty of Engineering

Department of Computer Engineering

DATABASE SYSTEM DESIGN

**GRADUATION PROJECT
COM – 400**

Student: Murat BİNGÜL (20010593)

Supervisor: Assist Prof. Dr. Adel Amirjanov

Nicosia - 2005

ACKNOWLEDGMENT

First of all I would like to thank Assit. Prof. Dr. Adil Amirjanov for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

I would like to thanks all of my friends who helped me to overcome my project especially Mehmet Gögebakan

Finally, I would like to thank my family, especially my parents. Their love and guidance saw me through doubtful times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together.

They have made countless sacrifices for my betterment. I can't repay them, but I do hope that their endless efforts will bear fruit and that I may lead them, myself and all who surround me to a better future.

Abstract

In this project development of software on car gallery has been considered. This software compiled with Microsoft Access Programing which is a database program included Microsoft Office Packet.

Nowadays, computers are used almost every area of the business and life, with computers and softwares they increase the speed of our calculations, transactions ...etc

Our application is designed for car galleries which cover all their needs. In this program we are able to hold costumer records, car records, our accounting transactions and we can give reports on our transactions.

With this program maintenance of a car gallery will be easy ,faster and reliable.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENT	I
ABSTRACT	II
TABLE OF CONTENTS	III
LIST OF FIGURES	VI
INTRODUCTION	1
CHAPTER ONE INTRODUCTION TO DATABASE MANAGEMENT	
SYSTEMS DATABASE	2
1.1 Overview.....	2
1.2 Database Models.....	3
1.2.1 Flat Model.....	3
1.2.2 Network Model.....	3
1.2.3 Relational model.....	4
1.3 Why we use a Relational Database Design.....	5
1.3.1 Relationships Between Tables.....	5
1.3.2 One-to-one relationships.....	5
1.3.3 One-to-many relationships.....	6
1.4 Relational Operations.....	8
1.4.1 Implementations and indexing.....	9
1.5 Applications of databases.....	10
1.6 Data modeling.....	10
1.6.1 Database normalization	11
1.6.2 Primary key.....	11
1.6.3 Foreign Key.....	12
1.6.4 Compound Key.....	13

CHAPTER TWO: STRUCTURED QUERY LANGUAGE (SQL)	14
2.1 History.....	14
2.2 Description of SQL.....	15
2.3 SQL keywords.....	16
2.3.1 Data retrieval.....	16
2.3.2 Data manipulation.....	17
2.3.3 Data transaction.....	17
2.3.4 Data definition.....	17
 CHAPTER THREE : MICROSOFT ACCESS DATABASE SYSTEM	 19
3.1 Introductory Microsoft Access.....	19
3.2 Working with Tables.....	19
3.3 Creating Tables Manually.....	21
3.3.1 Data Types.....	21
3.3.2 AutoNumber fields.....	23
3.3.3 Text fields.....	23
3.3.4 Number fields.....	24
3.3.5 Memo, OLE Object, Date/Time, and Yes/No fields.....	25
3.3.6 Primary Key.....	25
3.3.7 Working with Forms.....	26
3.3.8 Switchboard Forms.....	27
3.3.8.1 Subforms.....	28
3.3.8.2 Parameter queries.....	31
3.4 Creating a Query.....	31
3.5 Security.....	36
3.5.1 Setting a database password.....	36

CHAPTER FOUR: CAR GALLERY DATABASE DESIGN

4.1 Creating Tables for Car Gallery.....	38
4.2 Construction relationship between tables.....	44
4.3 Creating forms for our application.....	45
4.4 Sql Operations for Car Gallery.....	49
4.5 Creating report for Car Gallery.....	52
4.6 Applying Switchboard Manager.....	55
4.7 Applying security password to database.....	58
CONCLUSION.....	60
REFERENCES.....	61

LIST OF FIGURE

Figure 3.1 Create table in design view.....	19
Figure 3.2 Fields of table.....	20
Figure 3.3 Table wizard.....	20
Figure 3.4 Form wizard.....	29
Figure 3.5 Mytable.....	30
Figure 3.6 Simple query wizard.....	32
Figure 3.7 Query view.....	33
Figure 3.8 Report wizard view.....	34
Figure 3.9 Report sorting view.....	34
Figure 3.10 Report format view.....	35
Figure 4.1 Table of customer.....	38
Figure-4.2 Creating Tables.....	39
Figure 4.3 Sale table in design view.....	40
Figure 4.4 Details of the Sale table is shown.....	41
Figure 4.5 Structure of stock table.....	42
Figure 4.6 Color table.....	43
Figure 4.7 Producer table.....	43
Figure 4.8 Structure of relationships among tables.....	44
Figure 4.9 Form wizard screen.....	45
Figure 4.10 Next step of form wizard for layout of the form.....	46
Figure 4.11 Next step of form wizard for style design.....	46
Figure 4.12 Modifying the form.....	47
Figure-4.13 Toolbox	47
Figure-4.14 Command button wizard screen.....	48
Figure-4.15 Next step of button wizard screen.....	48
Figure-4.16 View of the customer form.....	49
Figure 4.17 Query is design view.....	49
Figure-4.18 Show table screen.....	50
Figure-4.19 Query design view.....	50
Figure 4.20 Query in sql code view.....	51
Figure-4.21 Structure of total query in sql code view.....	51

Figure-4.22 Create report design view.....	52
Figure 4.23 Report wizard next step view.....	53
Figure 4.24 Selecting fields to order customer table for report.....	53
Figure-4.25 Screens shows that hot to give style to a report.....	54
Figure-4.26 Shows that modifying existent report.....	55
Figure 4.27 Switchboard design.....	55
Figure 4.28 Next steps of switchboard design.....	56
Figure 4.29 Shows that hot to connect all forms to switchboard.....	57
Figure 4.30 Open a file in exclusive mode.....	58
Figure 4.31 Screen for password for the database.....	59

INTRODUCTION

A database is a means of collecting and organizing information. You can create simple ones that contain a list of your business contacts, for example, or you can build a full-featured data management system that you can use to manage a business. Microsoft Access gives you the tools to build just about any kind of database you need.

Access is a popular development platform in large measure because it is part of the Microsoft Office suite. Many clients want their Access systems to interoperate with the rest of Office, and they want systems that are transparent and easy to maintain without developer assistance.

Once you have created a database and added information to it, you can use Access to create forms or reports. With these, you can retrieve the information you've put into the database. Forms and reports can be anything from invoices to accounts receivable summaries to mailing labels to inventory reports and more. With the program's built-in samples, templates, and wizards, it's easy for a beginner to get started. Access 2000 is straightforward enough to allow a beginner to create quick and simple, but very useful databases. As your skills grow, you can add more features that will expand your database's usefulness to you or your business. You will very quickly see just how easy it is, as you begin to create your own database.

In the project development of database system was considered for car gallery with Microsoft access programming.

In Chapter one : discussed about databases, data model, type of databases, relationship, primary and foreign, compound key in detail.

In Chapter two: general SQL structure and its most used keywords presented.

In Chapter three: introductory remarks on Microsoft access programming is given. Creating and modifying of Tables, queries, Report, forms are explained. How to use wizard is illustrated with figures.

In Chapter four, general structure of our application is given. Implementation of the program is given step by step with explanation.

CHAPTER ONE : INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

DATABASE

1.1 Overview

A database is a collection of information stored in a computer in a systematic way, such that a computer program can consult it to answer questions. The software used to manage and query a database is known as a database management system (DBMS). The properties of database systems are studied in information science.

At the core of the concept of a database is the idea of a collection of facts, or pieces of knowledge. Facts may be structured in a number of ways, known as data models. For instance, one model is to associate each fact with a record representing an entity (such as a person), and to arrange these entities into trees or hierarchies the hierarchical data model. Another model is to arrange facts into sets of values which satisfy logical predicates the relational model.

(The terms database and database management system are sometimes interchanged by students. In the professional argot, a database is always the collection of facts, not the software program.)

Database management systems range from the extremely simple to the highly complex. Differences among DBMSes include whether they are capable of ensuring the integrity of the data; whether they may be used by many users at once; and what sorts of conclusions they can be programmed to compute from a set of data.

The first database management systems were developed in the 1960s. A pioneer in the field was Charles Bachman. Two key data models arose at this time: the network model followed by the hierarchical model. These were later usurped by the relational model, which was contemporary with the so-called flat model designed for very small tasks. Another contemporary of the relational model is the object-oriented database (OODB).

While the relational model is based on set theory, one proposed modification suggests fuzzy set theory as an alternative.

1.2 Database Models

Various techniques are used to model data structure. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementations may be possible. An example of this is the relational model: In larger systems the physical implementation often has indexes which point to the data this is similar to some aspects of common implementations of the network model. But in small relational databases the data is often stored in a set of files, one per table, in a flat, unindexed structure. There is some confusion below and elsewhere in this article as to logical data model vs its physical implementation.[7]

1.2.1 Flat Model

The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password might be used as a part of a system security database. Each row would have the specific password associated with a specific user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is the basis of the spreadsheet.[7]

1.2.2 Network model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.[7]

1.2.3 Relational model

The relational model was introduced in an academic paper by E. F. Codd in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few, obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allows tables to be defined that allow duplicate rows – an extension (or violation) of the relational model. In common English usage, a DBMS is called relational if it supports relational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, non-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't originally intended to be one.

A key that can be used to uniquely identify a row in a table is called a unique key.

Typically one of the unique keys is the preferred way to refer to row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no natural key is suitable, an arbitrary key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that can't break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example,

records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

1.3 Why we use a Relational Database Design

Thus, maintaining a simple, so-called flat database consisting of a single table does not require much knowledge of database theory. On the other hand, most databases worth maintaining are quite a bit more complicated than that. Real- life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full- fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database!

1.3.1 Relationships Between Tables

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many, and many to many.

1.3.2 One-to-one relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their trucks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support security. Access applies user-level security at the table level.

Therefore, if a subset of the fields in a table requires security, placing them in a separate table lets your application restrict access to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to those fields.[4]

1.3.3 One-to-many relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a many-to-many relationship as well.

Redundancy

The main problems associated with using a single table to maintain a database stem from the issue of unnecessary repetition of data, that is, redundancy. Some repetition of data is always necessary, but the idea is to remove as much unnecessary repetition as possible. The redundancy in the library flat table (Table 1.1) is obvious. For instance, the name and phone number of Big House publishers is repeated in the table. In an effort to remove as much redundancy as possible from a database, a database designer must split the data into multiple tables. Here is one possibility for the library_flat example, which splits the original database into four separate tables.

- a books table, shown in Table 1.2, in which each book has its own record
- An authors table, shown in Table 1.3, in which each author has his or her own record
- a publishers table, shown in Table 1.4, in which each publisher has its own record
- book/author table, shown in Table 1.5, the purpose of which we will explain a bit later

Table 1.1. The LIBRARY_FLAT Sample Database

ISBN	Title	AuID	AuName	AuPhone	PubID	PubName	PubPhone	Price
1-1111-1111-	C++	4	Roman	444-444-	1	Big House	123-456-	\$29.95

Table 1.2. The BOOKS Table from the LIBRARY_FLAT Database

ISBN	Title	PubID	Price
0-555-55555-9	Macbeth	2	\$12.00
0-91-335678-7	Faerie Queene	1	\$15.00
0-99-999999-9	Emma	1	\$20.00

Table 1.3. The AUTHORS Table from the LIBRARY_FLAT Database

AuID	AuName	AuPhone
1	Austen	111-111-1111
12	Grumpy	321-321-0000
3	Homer	333-333-3333

Table 1.4. The PUBLISHERS Table from the LIBRARY_FLAT Database

PubID	PubName	PubPhone
1	Big House	123-456-7890
2	Alpha Press	999-999-9999
3	Small House	714-000-0000

Table 1.5. The BOOK/AUTHOR Table from the LIBRARY_FLAT Database

ISBN	AuID
0-103-45678-9	3
0-11-345678-9	2
0-12-333433-3	8

Note that now the name and phone number of Big House appears only once in the database (in the publishers table), as does Shakespeare's phone number (in the authors table). Of course, there are still some duplicated data in the database. For instance, the PubID information appears in more than one place in these tables. As mentioned earlier, we

can not eliminate all duplicate data and still maintain the relationships between the data. To get a feel for the reduction in duplicate data achieved by the four-table approach, imagine (as is reasonable) that the database also includes the address of each publisher. Then Table 1.1 would need a new column containing many addresses many of which are duplicates. On the other hand, the four-table database needs only one new column in the publishers table, adding a total of three distinct addresses. To drive the difference home, consider the 16-million-book database of the Library of Congress. Suppose the database contains books from 10,000 different publishers. A publisher's address column in a flat database design would contain 16 million addresses, whereas a multitable approach would require only 10,000 addresses. Now, if the average address is 50 characters long, then the multitable approach would save $(16,000,000 - 10,000) * 50 = 799$ million characters

Assuming that each character takes 2 bytes (in the Unicode that is used internally by Microsoft Access), the single-table approach wastes about 1.6 gigabytes of space, just for the address field! Indeed, the issue of redundancy alone is quite enough to convince a database designer to avoid the flat database approach. However, there are several other problems with flat databases, which we now discuss.

1.4 Relational Operations

You request data from a relational database by sending it a query that's written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it's much more common for SQL queries to be embedded into software that provides an easier user interface.

In response to a query, the database returns a result set, which is just a list of rows containing the answers. The simplest query is just to return all the rows from a table, but more often, the rows are filtered in some way to return just the answer wanted.

Often, data from multiple tables gets combined into one, by doing a "join". Conceptually, this is done by taking all possible combinations of rows (the "cross-product"), and then

filtering out everything except the answer. In practice, relational database management systems rewrite ("optimize") queries to perform faster, using a variety of techniques.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for decades. This has made the idea and implementation of relational databases very popular with businesses.

1.4.1 Implementations and indexing

All of these kinds of database can take advantage of indexing to increase their speed. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Various methods of indexing are commonly used; b-trees, hashes, and linked lists are all common indexing techniques.

Relational DBMSs have the advantage that indices can be created or dropped without changing existing applications, because applications don't use the indices directly. Instead, the database software decides on behalf of the application which indices to use. The database chooses between many different strategies based on which one it estimates will run the fastest.

Relational DBMSs utilise many different algorithms to compute the result of an SQL statement. The RDBMs will produce a plan of how to execute the query, which is generated by analysing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are Nested Loops Join, Sort-Merge Join and Hash Join.

1.5 Applications of databases

Databases are used in many applications, spanning virtually the entire range of computer software. Databases are the preferred method of storage for large multiuser applications, where coordination between many users is needed. Even individual users find them convenient, though, and many electronic mail programs and personal organizers are based on standard database technology.

A **database management system (DBMS)** is a computer program (or more typically, a suite of them) designed to manage a database, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMS use include accounting, human resources and customer support systems. Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

DBMS's are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system

1.6 Data modeling

In information system design, **data modeling** is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "data analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together) than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

In the early phases of a software development project, emphasis will be on the design of a conceptual data model. This can be detailed into a logical data model sometimes called a functional data model. In later stages, this model may be translated into physical data model.

1.6.1 Database normalization

database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMSs lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case denormalization is sometimes used to improve performance, at the cost of reduced consistency.

1.6.2 Primary key

In database design, a **primary key** is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions), and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a **primary key** is a candidate key chosen as the main method of uniquely identifying a tuple in a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a tuple in a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others [7].

In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning that its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity, by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

1.6.3 Foreign Key

A **foreign key** (FK) is a field in a database record that points to a key field of another database record in another table. Usually a foreign key in one table refers to the primary key (PK) of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail need not include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book - or even the book itself. The ISBN is the primary-key of the book, and it is used as a foreign-key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

1.6.4 Compound Key

In database design, a **compound key** (also called a **composite key**) is a key that consists on 2 or more attributes.

No restriction is applied to the attributes regarding their (initial) ownership within the data model. This means that any one, none, or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute

CHAPTER TWO: STRUCTURED QUERY LANGUAGE (SQL)

2.1 History

Structured Query Language (SQL) is the most popular computer language used to create, modify and retrieve data from relational database management systems. The language has evolved beyond its original purpose to support object-relational database management systems.

A seminal paper, "A Relational Model of Data for Large Shared Data Banks, by Dr. Edgar F. Codd, was published in June, 1970 in the Association for Computing Machinery (acm) journal, Communications of the acm. Codd's model became widely accepted as the definitive model for relational database management systems (rdbms).

During the 1970s, a group at IBM's San Jose research center developed a database system "System R" based upon Codd's model. **Structured English Query Language** ("sequel") was designed to manipulate and retrieve data stored in System R. The acronym sequel was later condensed to **SQL** due to a trademark dispute (the word 'sequel' was held as a trademark by the Hawker-Siddeley aircraft company of the uk).

In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL and, soon, many vendors developed dialects of it.

SQL was adopted as a standard by the ANSI (American National Standards Institute) in 1986 and ISO (International Organization for Standardization) in 1987. ANSI has declared that the official pronunciation for SQL is "es queue el", although many English-speaking database professionals still pronounce it as sequel.

2.2 Description of SQL

SQL allows the specification of queries in a high-level, declarative manner. For example, to select rows from a database, the user need only specify the criteria that they want to search by; the details of performing the search operation efficiently is left up to the database system, and is invisible to the user.

Compared to general-purpose programming languages, this structure allows the user/programmer to be less familiar with the technical details of the data and how they are stored, and relatively more familiar with the information contained in the data. This blurs the line between user and programmer, appealing to individuals who fall more into the 'business' or 'research' area and less in the 'information technology' area. The original vision for SQL was to allow non-technical users to write their own database queries. While this has been realized to some extent, the complexity of querying an advanced database system using SQL can still require a significant learning curve.

SQL contrasts with the more powerful database-oriented fourth-generation programming languages such as Focus or sas, however, in its relative functional simplicity and simpler command set. This greatly reduces the degree of difficulty involved in maintaining the worst SQL source code, but it also makes programming such questions as 'Who had the top ten scores?' more difficult, leading to the development of procedural extensions, discussed above. However, it also makes it possible for SQL source code to be produced (and optimized) by software, leading to the development of a number of natural language database query languages, as well as 'drag and drop' database programming packages with 'object oriented' interfaces. Often these allow the resultant SQL source code to be examined, for educational purposes, further enhancement, or to be used in a different environment

2.3 SQL keywords

SQL keywords fall into several groups.

2.3.1 Data retrieval

The most frequently used operation in transactional databases is the data retrieval operation.

- **SELECT** is used to retrieve zero or more rows from one or more tables in a database. In most applications, **SELECT** is the most commonly used DML command. In specifying a **SELECT** query, the user specifies a description of the desired result set, but they do not specify what physical operations must be executed to produce that result set. Translating the query into an optimal query plan is left to the database system, more specifically to the query optimizer.
- Commonly available keywords related to **SELECT** include:
 - **FROM** is used to indicate which tables the data is to be taken from, as well as how the tables join to each other.
 - **WHERE** is used to identify which rows to be retrieved, or applied to **GROUP BY**.
 - **GROUP BY** is used to combine rows with related values into elements of a smaller set of rows.
 - **HAVING** is used to identify which rows, following a **GROUP BY**, are to be retrieved.
 - **ORDER BY** is used to identify which columns are used to sort the resulting data.

2.3.2 Data manipulation

First there are the standard Data Manipulation Language (DML) elements. DML is the subset of the language used to add, update and delete data.

- INSERT is used to add zero or more rows (formally tuples) to an existing table.
- UPDATE is used to modify the values of a set of existing table rows.
- DELETE removes zero or more existing rows from a table.

2.3.3 Data transaction:

Transaction, if available, can be used to wrap around the DML operations.

BEGIN WORK (or START TRANSACTION, depending on SQL dialect) can be used to mark the start of a database transaction, which either completes completely or not at all.

COMMIT causes all data changes in a transaction to be made permanent.

ROLLBACK causes all data changes since the last COMMIT or ROLLBACK to be discarded, so that the state of the data is "rolled back" to the way it was prior to those changes being requested.

COMMIT and ROLLBACK interact with areas such as transaction control and locking. Strictly, both terminate any open transaction and release any locks held on data. In the absence of a BEGIN WORK or similar statement, the semantics of SQL are implementation-dependent.[3]

2.3.4 Data definition

The second group of keywords is the Data Definition Language (DDL). DDL allows the user to define new tables and associated elements. Most commercial SQL databases have proprietary extensions in their DDL, which allow control over nonstandard features of the database system.

The most basic items of DDL are the CREATE and DROP commands. CREATE causes an object (a table, for example) to be created within the database. DROP causes an existing object within the database to be deleted, usually irretrievably. Some database systems also have an ALTER command, which permits the user to modify an existing object in various ways for example, adding a column to an existing table.

CHAPTER THREE: MICROSOFT ACCESS DATABASE SYSTEM

3.1 Introductory Microsoft Access

- Starting Microsoft Access
- Opening Access
- Click **Start**, select **All Programs** and click on **Microsoft Access**
- Creating a database
- Click **File, New** or click the new icon on the standard toolbar
- Select **Blank Database** from the Task Pane menu
- Type a name for database in the File Name window
- Click **Create**
- Closing a database
- Click **File, Close**
- Opening a database
- Click **File, Open** or click the open icon on the standard toolbar
- Browse to where the database is saved
- Click the name of the database
- Click **Open**

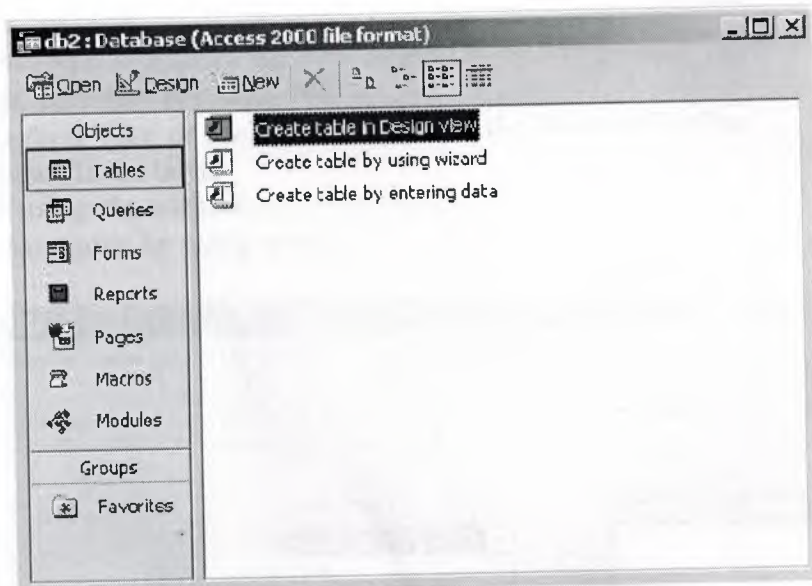
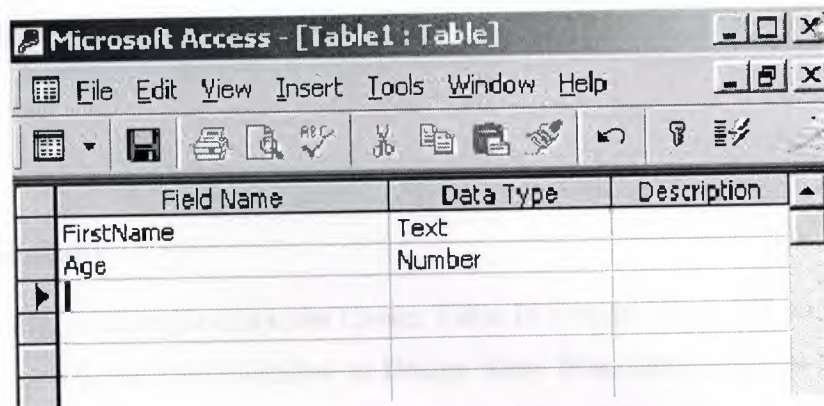


Figure 3.1 create table in design view

3.2 Working with Tables

- A table is a collection of data about a specific topic, such as people, products, or companies.
- Creating a Table
- From the main database window, click **Tables** under Objects on the left menu
- **Creating a table without assistance:**
- Double-click **Create table in design view**
- In the Field Name column type the name of data field (i.e. FirstName)

- In the Data Type column select the type of data to be entered in the field (i.e. Text, Number, etc.)
- Text, Number, etc.)
- Complete steps b and c for all other data fields (i.e. LastName, Address, etc.)



Field Name	Data Type	Description
FirstName	Text	
Age	Number	

Figure 3.2 fields of table

- Save the table. Click **File, Save** or click the save icon on the Standard Toolbar.
 - Type the name for the table.
 - Click **Ok**.
 - If you do not want a primary key, click No.
 - Click **View, Datasheet View** or click the view icon on the Standard Toolbar to begin entering data in the table.
- Creating a table using the wizard:**
- Double-click **Create table by using wizard**

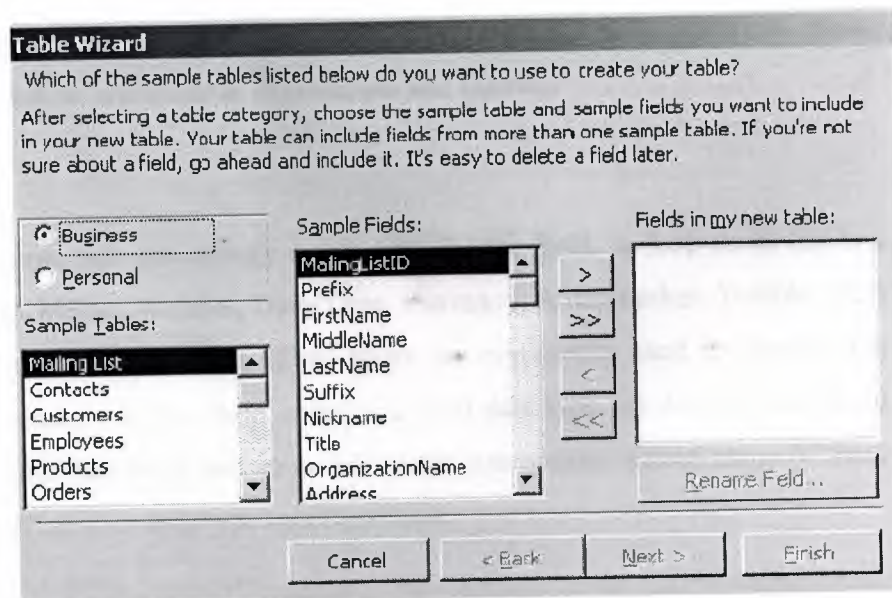


Table Wizard

Which of the sample tables listed below do you want to use to create your table?

After selecting a table category, choose the sample table and sample fields you want to include in your new table. Your table can include fields from more than one sample table. If you're not sure about a field, go ahead and include it. It's easy to delete a field later.

☒ Business
☐ Personal

Sample Tables:

- Mailing List
- Contacts
- Customers
- Employees
- Products
- Orders

Sample Fields:

- MailingListID
- Prefix
- FirstName
- MiddleName
- LastName
- Suffix
- Nickname
- Title
- OrganizationName
- Address

Fields in my new table:

Rename Field...

Cancel < Back Next > Finish

Figure 3.3 table wizard

- Select the type of table (Business or Personal)
- Choose a table from the **Sample Tables** list
- Select a data field to include in the table
- Click the single arrow
- Repeat steps d and e for all other data fields you wish to include in the table
- Click **Finish**
- Enter data into table

3.3 Creating Tables Manually

To create a table manually, you double-click the Create Table In Design View icon in the Database window to open a blank Table window in Design view. From here, you can add fields, a primary key, and an index. (Primary keys are indexes with special property settings.)

To add a field, type the field's name in a blank Field Name column in the Design view window. Field names follow normal Visual Basic for Applications (VBA) naming conventions. They can be up to 64 characters long, and the characters can be letters, numbers, spaces, and special character except the period, the exclamation mark, square brackets, and the grave accent character (`). Also, you cannot start a field name with a space or a control character (ASCII values 0 through 31). While you can include internal spaces in field names, they must be bracketed in expressions and queries.

3.3.1 Data Types

In the Data Type column, you can specify a data type for the field. A drop-down list box offers 10 options: Text, Memo, Number, Date/Time, Currency, AutoNumber, Yes/No, OLE Object, Hyperlink, and Lookup wizard. (Data types are commonly used to identify the information a field contains. A Text field contains a Text data type, an AutoNumber field contains an AutoNumber data type, and so on.) You can use options within many of these data types to further refine your data type specifications.

Variable Data Types

<i>Name</i>	<i>Number of Bytes</i>	<i>Range</i>
Byte	1	0 through 255
Boolean	2	True or false
Integer	2	-32,768 through 32,767
Long	4	-2,147,483,648 through 2,147,483,647
Single	4	-3.402823E38 through -1.401298E-45 for negative values 1.40129E-45 through 3.402823E38 for positive values
Double	8	-1.79769313486232E308 through -14.94065645841247E-324 for negative values 4.94065645841247E-324 through 1.79769313486232E308 for positive values
Currency	8	-922,337,203,685,477.5808 through 922,337,203,685,477.5807
Date	8	January 1, 100 through December 31, 9999
Object	4	A reference to an object (see <i>Set</i> statement in the online help)
String (fixed)	Length	Up to approximately 64,000 characters
String (variable)	10 + length	Up to approximately 2 billion characters
Variant (with numbers)	16	Same as Double
Variant (with characters)	22 + length	Same as String (variable)
User-defined	Depends on elements	Sum of the constituent elements for the custom data type

NEAR EAST UNIVERSITY



Faculty of Engineering

Department of Computer Engineering

DATABASE SYSTEM DESIGN

**GRADUATION PROJECT
COM – 400**

Student: Murat BİNGÜL (20010593)

Supervisor: Assist Prof. Dr. Adel Amirjanov

Nicosia - 2005

ACKNOWLEDGMENT

First of all I would like to thank Assit. Prof. Dr. Adil Amirjanov for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

I would like to thank all of my friends who helped me to overcome my project especially Mehmet Gögebakan

Finally, I would like to thank my family, especially my parents. Their love and guidance saw me through doubtful times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together.

They have made countless sacrifices for my betterment. I can't repay them, but I do hope that their endless efforts will bear fruit and that I may lead them, myself and all who surround me to a better future.

Abstract

In this project development of software on car gallery has been considered. This software compiled with Microsoft Access Programing which is a database program included Microsoft Office Packet.

Nowadays, computers are used almost every area of the business and life, with computers and softwares they increase the speed of our calculations, transactions ...etc

Our application is designed for car galleries which cover all their needs. In this program we are able to hold costumer records, car records, our accounting transactions and we can give reports on our transactions.

With this program maintenance of a car gallery will be easy ,faster and reliable.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENT	I
ABSTRACT	II
TABLE OF CONTENTS	III
LIST OF FIGURES	VI
INTRODUCTION	1
CHAPTER ONE INTRODUCTION TO DATABASE MANAGEMENT	
SYSTEMS DATABASE	2
1.1 Overview.....	2
1.2 Database Models.....	3
1.2.1 Flat Model.....	3
1.2.2 Network Model.....	3
1.2.3 Relational model.....	4
1.3 Why we use a Relational Database Design.....	5
1.3.1 Relationships Between Tables.....	5
1.3.2 One-to-one relationships.....	5
1.3.3 One-to-many relationships.....	6
1.4 Relational Operations.....	8
1.4.1 Implementations and indexing.....	9
1.5 Applications of databases.....	10
1.6 Data modeling.....	10
1.6.1 Database normalization	11
1.6.2 Primary key.....	11
1.6.3 Foreign Key.....	12
1.6.4 Compound Key.....	13

CHAPTER TWO: STRUCTURED QUERY LANGUAGE (SQL)	14
2.1 History.....	14
2.2 Description of SQL.....	15
2.3 SQL keywords.....	16
2.3.1 Data retrieval.....	16
2.3.2 Data manipulation.....	17
2.3.3 Data transaction.....	17
2.3.4 Data definition.....	17
 CHAPTER THREE : MICROSOFT ACCESS DATABASE SYSTEM	 19
3.1 Introductory Microsoft Access.....	19
3.2 Working with Tables.....	19
3.3 Creating Tables Manually.....	21
3.3.1 Data Types.....	21
3.3.2 AutoNumber fields.....	23
3.3.3 Text fields.....	23
3.3.4 Number fields.....	24
3.3.5 Memo, OLE Object, Date/Time, and Yes/No fields.....	25
3.3.6 Primary Key.....	25
3.3.7 Working with Forms.....	26
3.3.8 Switchboard Forms.....	27
3.3.8.1 Subforms.....	28
3.3.8.2 Parameter queries.....	31
3.4 Creating a Query.....	31
3.5 Security.....	36
3.5.1 Setting a database password.....	36

CHAPTER FOUR: CAR GALLERY DATABASE DESIGN

4.1 Creating Tables for Car Gallery.....	38
4.2 Construction relationship between tables.....	44
4.3 Creating forms for our application.....	45
4.4 Sql Operations for Car Gallery.....	49
4.5 Creating report for Car Gallery.....	52
4.6 Applying Switchboard Manager.....	55
4.7 Applying security password to database.....	58
CONCLUSION.....	60
REFERENCES.....	61

LIST OF FIGURE

Figure 3.1 Create table in design view.....	19
Figure 3.2 Fields of table.....	20
Figure 3.3 Table wizard.....	20
Figure 3.4 Form wizard.....	29
Figure 3.5 Mytable.....	30
Figure 3.6 Simple query wizard.....	32
Figure 3.7 Query view.....	33
Figure 3.8 Report wizard view.....	34
Figure 3.9 Report sorting view.....	34
Figure 3.10 Report format view.....	35
Figure 4.1 Table of customer.....	38
Figure-4.2 Creating Tables.....	39
Figure 4.3 Sale table in design view.....	40
Figure 4.4 Details of the Sale table is shown.....	41
Figure 4.5 Structure of stock table.....	42
Figure 4.6 Color table.....	43
Figure 4.7 Producer table.....	43
Figure 4.8 Structure of relationships among tables.....	44
Figure 4.9 Form wizard screen.....	45
Figure 4.10 Next step of form wizard for layout of the form.....	46
Figure 4.11 Next step of form wizard for style design.....	46
Figure 4.12 Modifying the form.....	47
Figure-4.13 Toolbox	47
Figure-4.14 Command button wizard screen.....	48
Figure-4.15 Next step of button wizard screen.....	48
Figure-4.16 View of the customer form.....	49
Figure 4.17 Query is design view.....	49
Figure-4.18 Show table screen.....	50
Figure-4.19 Query design view.....	50
Figure 4.20 Query in sql code view.....	51
Figure-4.21 Structure of total query in sql code view.....	51

Figure-4.22 Create report design view.....	52
Figure 4.23 Report wizard next step view.....	53
Figure 4.24 Selecting fields to order customer table for report.....	53
Figure-4.25 Screens shows that hot to give style to a report.....	54
Figure-4.26 Shows that modifying existent report.....	55
Figure 4.27 Switchboard design.....	55
Figure 4.28 Next steps of switchboard design.....	56
Figure 4.29 Shows that hot to connect all forms to switchboard.....	57
Figure 4.30 Open a file in exclusive mode.....	58
Figure 4.31 Screen for password for the database.....	59

INTRODUCTION

A database is a means of collecting and organizing information. You can create simple ones that contain a list of your business contacts, for example, or you can build a full-featured data management system that you can use to manage a business. Microsoft Access gives you the tools to build just about any kind of database you need.

Access is a popular development platform in large measure because it is part of the Microsoft Office suite. Many clients want their Access systems to interoperate with the rest of Office, and they want systems that are transparent and easy to maintain without developer assistance.

Once you have created a database and added information to it, you can use Access to create forms or reports. With these, you can retrieve the information you've put into the database. Forms and reports can be anything from invoices to accounts receivable summaries to mailing labels to inventory reports and more. With the program's built-in samples, templates, and wizards, it's easy for a beginner to get started. Access 2000 is straightforward enough to allow a beginner to create quick and simple, but very useful databases. As your skills grow, you can add more features that will expand your database's usefulness to you or your business. You will very quickly see just how easy it is, as you begin to create your own database.

In the project development of database system was considered for car gallery with Microsoft access programming.

In Chapter one : discussed about databases, data model, type of databases, relationship, primary and foreign, compound key in detail.

In Chapter two: general SQL structure and its most used keywords presented.

In Chapter three: introductory remarks on Microsoft access programming is given. Creating and modifying of Tables, queries, Report, forms are explained. How to use wizard is illustrated with figures.

In Chapter four, general structure of our application is given. Implementation of the program is given step by step with explanation.

CHAPTER ONE : INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

DATABASE

1.1 Overview

A database is a collection of information stored in a computer in a systematic way, such that a computer program can consult it to answer questions. The software used to manage and query a database is known as a database management system (DBMS). The properties of database systems are studied in information science.

At the core of the concept of a database is the idea of a collection of facts, or pieces of knowledge. Facts may be structured in a number of ways, known as data models. For instance, one model is to associate each fact with a record representing an entity (such as a person), and to arrange these entities into trees or hierarchies the hierarchical data model. Another model is to arrange facts into sets of values which satisfy logical predicates the relational model.

(The terms database and database management system are sometimes interchanged by students. In the professional argot, a database is always the collection of facts, not the software program.)

Database management systems range from the extremely simple to the highly complex. Differences among DBMSes include whether they are capable of ensuring the integrity of the data; whether they may be used by many users at once; and what sorts of conclusions they can be programmed to compute from a set of data.

The first database management systems were developed in the 1960s. A pioneer in the field was Charles Bachman. Two key data models arose at this time: the network model followed by the hierarchical model. These were later usurped by the relational model, which was contemporary with the so-called flat model designed for very small tasks. Another contemporary of the relational model is the object-oriented database (OODB).

While the relational model is based on set theory, one proposed modification suggests fuzzy set theory as an alternative.

1.2 Database Models

Various techniques are used to model data structure. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementations may be possible. An example of this is the relational model: In larger systems the physical implementation often has indexes which point to the data this is similar to some aspects of common implementations of the network model. But in small relational databases the data is often stored in a set of files, one per table, in a flat, unindexed structure. There is some confusion below and elsewhere in this article as to logical data model vs its physical implementation.[7]

1.2.1 Flat Model

The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password might be used as a part of a system security database. Each row would have the specific password associated with a specific user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is the basis of the spreadsheet.[7]

1.2.2 Network model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.[7]

1.2.3 Relational model

The relational model was introduced in an academic paper by E. F. Codd in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few, obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allows tables to be defined that allow duplicate rows — an extension (or violation) of the relational model. In common English usage, a DBMS is called relational if it supports relational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, non-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't originally intended to be one.

A key that can be used to uniquely identify a row in a table is called a unique key.

Typically one of the unique keys is the preferred way to refer to row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no natural key is suitable, an arbitrary key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that can't break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example,

records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

1.3 Why we use a Relational Database Design

Thus, maintaining a simple, so-called flat database consisting of a single table does not require much knowledge of database theory. On the other hand, most databases worth maintaining are quite a bit more complicated than that. Real- life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full- fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database!

1.3.1 Relationships Between Tables

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many, and many to many.

1.3.2 One-to-one relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their trucks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support security. Access applies user-level security at the table level.

Therefore, if a subset of the fields in a table requires security, placing them in a separate table lets your application restrict access to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to those fields.[4]

1.3.3 One-to-many relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a many-to-many relationship as well.

Redundancy

The main problems associated with using a single table to maintain a database stem from the issue of unnecessary repetition of data, that is, redundancy. Some repetition of data is always necessary, but the idea is to remove as much unnecessary repetition as possible. The redundancy in the library flat table (Table 1.1) is obvious. For instance, the name and phone number of Big House publishers is repeated in the table. In an effort to remove as much redundancy as possible from a database, a database designer must split the data into multiple tables. Here is one possibility for the library_flat example, which splits the original database into four separate tables.

- a books table, shown in Table 1.2, in which each book has its own record
- An authors table, shown in Table 1.3, in which each author has his or her own record
- a publishers table, shown in Table 1.4, in which each publisher has its own record
- book/author table, shown in Table 1.5, the purpose of which we will explain a bit later

Table 1.1. The LIBRARY_FLAT Sample Database

ISBN	Title	AuID	AuName	AuPhone	PubID	PubName	PubPhone	Price
1-1111-1111-	C++	4	Roman	444-444-	1	Big House	123-456-	\$29.95

Table 1.2. The BOOKS Table from the LIBRARY_FLAT Database

ISBN	Title	PubID	Price
0-555-55555-9	Macbeth	2	\$12.00
0-91-335678-7	Faerie Queene	1	\$15.00
0-99-999999-9	Emma	1	\$20.00

Table 1.3. The AUTHORS Table from the LIBRARY_FLAT Database

AuID	AuName	AuPhone
1	Austen	111-111-1111
12	Grumpy	321-321-0000
3	Homer	333-333-3333

Table 1.4. The PUBLISHERS Table from the LIBRARY_FLAT Database

PubID	PubName	PubPhone
1	Big House	123-456-7890
2	Alpha Press	999-999-9999
3	Small House	714-000-0000

Table 1.5. The BOOK/AUTHOR Table from the LIBRARY_FLAT Database

ISBN	AuID
0-103-45678-9	3
0-11-345678-9	2
0-12-333433-3	8

Note that now the name and phone number of Big House appears only once in the database (in the publishers table), as does Shakespeare's phone number (in the authors table). Of course, there are still some duplicated data in the database. For instance, the PubID information appears in more than one place in these tables. As mentioned earlier, we

can not eliminate all duplicate data and still maintain the relationships between the data. To get a feel for the reduction in duplicate data achieved by the four-table approach, imagine (as is reasonable) that the database also includes the address of each publisher. Then Table 1.1 would need a new column containing many addresses many of which are duplicates. On the other hand, the four-table database needs only one new column in the publishers table, adding a total of three distinct addresses. To drive the difference home, consider the 16-million-book database of the Library of Congress. Suppose the database contains books from 10,000 different publishers. A publisher's address column in a flat database design would contain 16 million addresses, whereas a multitable approach would require only 10,000 addresses. Now, if the average address is 50 characters long, then the multitable approach would save $(16,000,000 - 10,000) * 50 = 799$ million characters.

Assuming that each character takes 2 bytes (in the Unicode that is used internally by Microsoft Access), the single-table approach wastes about 1.6 gigabytes of space, just for the address field! Indeed, the issue of redundancy alone is quite enough to convince a database designer to avoid the flat database approach. However, there are several other problems with flat databases, which we now discuss.

1.4 Relational Operations

You request data from a relational database by sending it a query that's written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it's much more common for SQL queries to be embedded into software that provides an easier user interface.

In response to a query, the database returns a result set, which is just a list of rows containing the answers. The simplest query is just to return all the rows from a table, but more often, the rows are filtered in some way to return just the answer wanted.

Often, data from multiple tables gets combined into one, by doing a "join". Conceptually, this is done by taking all possible combinations of rows (the "cross-product"), and then

filtering out everything except the answer. In practice, relational database management systems rewrite ("optimize") queries to perform faster, using a variety of techniques.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for decades. This has made the idea and implementation of relational databases very popular with businesses.

1.4.1 Implementations and indexing

All of these kinds of database can take advantage of indexing to increase their speed. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Various methods of indexing are commonly used; b-trees, hashes, and linked lists are all common indexing techniques.

Relational DBMSs have the advantage that indices can be created or dropped without changing existing applications, because applications don't use the indices directly. Instead, the database software decides on behalf of the application which indices to use. The database chooses between many different strategies based on which one it estimates will run the fastest.

Relational DBMSs utilise many different algorithms to compute the result of an SQL statement. The RDBMs will produce a plan of how to execute the query, which is generated by analysing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are Nested Loops Join, Sort-Merge Join and Hash Join.

1.5 Applications of databases

Databases are used in many applications, spanning virtually the entire range of computer software. Databases are the preferred method of storage for large multiuser applications, where coordination between many users is needed. Even individual users find them convenient, though, and many electronic mail programs and personal organizers are based on standard database technology.

A **database management system (DBMS)** is a computer program (or more typically, a suite of them) designed to manage a database, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMS use include accounting, human resources and customer support systems. Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

DBMS's are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system

1.6 Data modeling

In information system design, **data modeling** is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "data analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together) than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

In the early phases of a software development project, emphasis will be on the design of a conceptual data model. This can be detailed into a logical data model sometimes called a functional data model. In later stages, this model may be translated into physical data model.

1.6.1 Database normalization

database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMSs lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case denormalization is sometimes used to improve performance, at the cost of reduced consistency.

1.6.2 Primary key

In database design, a **primary key** is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions), and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a **primary key** is a candidate key chosen as the main method of uniquely identifying a tuple in a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a tuple in a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others [7].

In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning that its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity, by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

1.6.3 Foreign Key

A **foreign key** (FK) is a field in a database record that points to a key field of another database record in another table. Usually a foreign key in one table refers to the primary key (PK) of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail need not include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book - or even the book itself. The ISBN is the primary-key of the book, and it is used as a foreign-key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

1.6.4 Compound Key

In database design, a **compound key** (also called a **composite key**) is a key that consists on 2 or more attributes.

No restriction is applied to the attributes regarding their (initial) ownership within the data model. This means that any one, none, or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute

CHAPTER TWO: STRUCTURED QUERY LANGUAGE (SQL)

2.1 History

Structured Query Language (SQL) is the most popular computer language used to create, modify and retrieve data from relational database management systems. The language has evolved beyond its original purpose to support object-relational database management systems.

A seminal paper, "A Relational Model of Data for Large Shared Data Banks, by Dr. Edgar F. Codd, was published in June, 1970 in the Association for Computing Machinery (acm) journal, Communications of the acm. Codd's model became widely accepted as the definitive model for relational database management systems (rdbms).

During the 1970s, a group at IBM's San Jose research center developed a database system "System R" based upon Codd's model. **Structured English Query Language** ("sequel") was designed to manipulate and retrieve data stored in System R. The acronym sequel was later condensed to **SQL** due to a trademark dispute (the word 'sequel' was held as a trademark by the Hawker-Siddeley aircraft company of the uk).

In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL and, soon, many vendors developed dialects of it.

SQL was adopted as a standard by the ANSI (American National Standards Institute) in 1986 and ISO (International Organization for Standardization) in 1987. ANSI has declared that the official pronunciation for SQL is "es queue el", although many English-speaking database professionals still pronounce it as sequel.

2.2 Description of SQL

SQL allows the specification of queries in a high-level, declarative manner. For example, to select rows from a database, the user need only specify the criteria that they want to search by; the details of performing the search operation efficiently is left up to the database system, and is invisible to the user.

Compared to general-purpose programming languages, this structure allows the user/programmer to be less familiar with the technical details of the data and how they are stored, and relatively more familiar with the information contained in the data. This blurs the line between user and programmer, appealing to individuals who fall more into the 'business' or 'research' area and less in the 'information technology' area. The original vision for SQL was to allow non-technical users to write their own database queries. While this has been realized to some extent, the complexity of querying an advanced database system using SQL can still require a significant learning curve.

SQL contrasts with the more powerful database-oriented fourth-generation programming languages such as Focus or sas, however, in its relative functional simplicity and simpler command set. This greatly reduces the degree of difficulty involved in maintaining the worst SQL source code, but it also makes programming such questions as 'Who had the top ten scores?' more difficult, leading to the development of procedural extensions, discussed above. However, it also makes it possible for SQL source code to be produced (and optimized) by software, leading to the development of a number of natural language database query languages, as well as 'drag and drop' database programming packages with 'object oriented' interfaces. Often these allow the resultant SQL source code to be examined, for educational purposes, further enhancement, or to be used in a different environment

2.3 SQL keywords

SQL keywords fall into several groups.

2.3.1 Data retrieval

The most frequently used operation in transactional databases is the data retrieval operation.

- **SELECT** is used to retrieve zero or more rows from one or more tables in a database. In most applications, **SELECT** is the most commonly used DML command. In specifying a **SELECT** query, the user specifies a description of the desired result set, but they do not specify what physical operations must be executed to produce that result set. Translating the query into an optimal query plan is left to the database system, more specifically to the query optimizer.
- Commonly available keywords related to **SELECT** include:
 - **FROM** is used to indicate which tables the data is to be taken from, as well as how the tables join to each other.
 - **WHERE** is used to identify which rows to be retrieved, or applied to **GROUP BY**.
 - **GROUP BY** is used to combine rows with related values into elements of a smaller set of rows.
 - **HAVING** is used to identify which rows, following a **GROUP BY**, are to be retrieved.
 - **ORDER BY** is used to identify which columns are used to sort the resulting data.

2.3.2 Data manipulation

First there are the standard Data Manipulation Language (DML) elements. DML is the subset of the language used to add, update and delete data.

- INSERT is used to add zero or more rows (formally tuples) to an existing table.
- UPDATE is used to modify the values of a set of existing table rows.
- DELETE removes zero or more existing rows from a table.

2.3.3 Data transaction:

Transaction, if available, can be used to wrap around the DML operations.

BEGIN WORK (or START TRANSACTION, depending on SQL dialect) can be used to mark the start of a database transaction, which either completes completely or not at all.

COMMIT causes all data changes in a transaction to be made permanent.

ROLLBACK causes all data changes since the last COMMIT or ROLLBACK to be discarded, so that the state of the data is "rolled back" to the way it was prior to those changes being requested.

COMMIT and ROLLBACK interact with areas such as transaction control and locking. Strictly, both terminate any open transaction and release any locks held on data. In the absence of a BEGIN WORK or similar statement, the semantics of SQL are implementation-dependent.[3]

2.3.4 Data definition

The second group of keywords is the Data Definition Language (DDL). DDL allows the user to define new tables and associated elements. Most commercial SQL databases have proprietary extensions in their DDL, which allow control over nonstandard features of the database system.

The most basic items of DDL are the CREATE and DROP commands. CREATE causes an object (a table, for example) to be created within the database. DROP causes an existing object within the database to be deleted, usually irretrievably. Some database systems also have an ALTER command, which permits the user to modify an existing object in various ways for example, adding a column to an existing table.

CHAPTER THREE: MICROSOFT ACCESS DATABASE SYSTEM

3.1 Introductory Microsoft Access

- Starting Microsoft Access
- Opening Access
- Click **Start**, select **All Programs** and click on **Microsoft Access**
- Creating a database
- Click **File, New** or click the new icon on the standard toolbar
- Select **Blank Database** from the Task Pane menu
- Type a name for database in the File Name window
- Click **Create**
- Closing a database
- Click **File, Close**
- Opening a database
- Click **File, Open** or click the open icon on the standard toolbar
- Browse to where the database is saved
- Click the name of the database
- Click **Open**

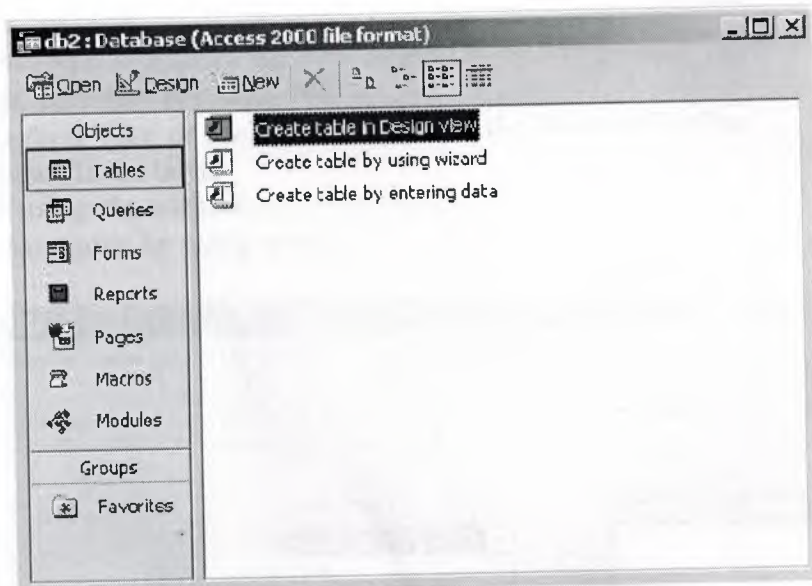
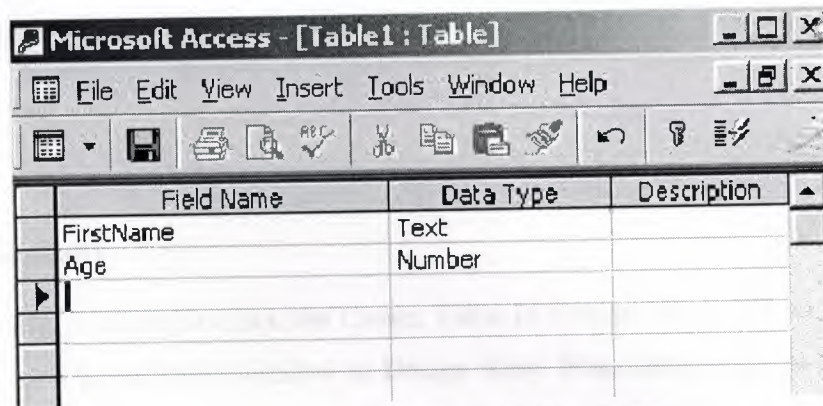


Figure 3.1 create table in design view

3.2 Working with Tables

- A table is a collection of data about a specific topic, such as people, products, or companies.
- Creating a Table
- From the main database window, click **Tables** under Objects on the left menu
- **Creating a table without assistance:**
- Double-click **Create table in design view**
- In the Field Name column type the name of data field (i.e. FirstName)

- In the Data Type column select the type of data to be entered in the field (i.e. Text, Number, etc.)
- Complete steps b and c for all other data fields (i.e. LastName, Address, etc.)

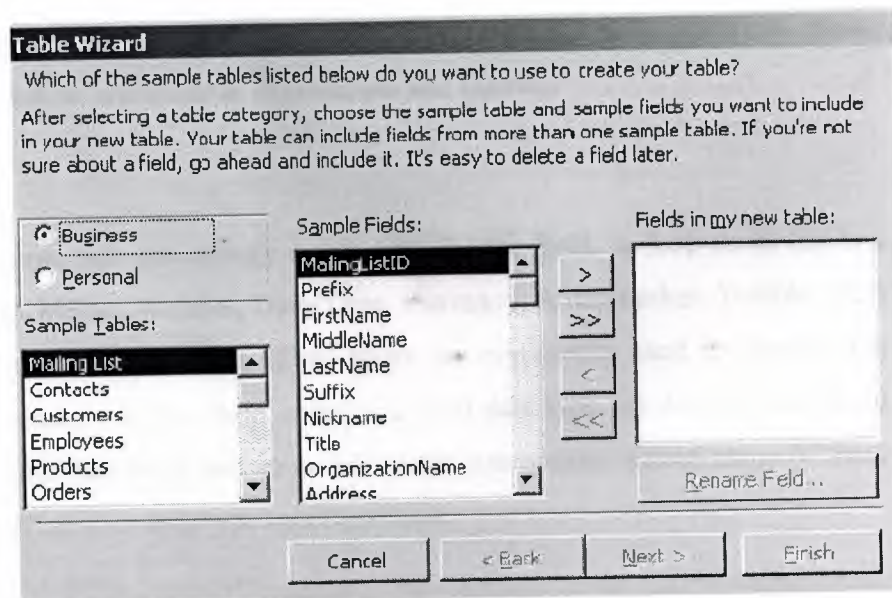


The screenshot shows the Microsoft Access interface with a table named 'Table1'. The table has three columns: 'Field Name', 'Data Type', and 'Description'. The first row contains 'FirstName' with a 'Text' data type. The second row contains 'Age' with a 'Number' data type. There are several empty rows below.

Field Name	Data Type	Description
FirstName	Text	
Age	Number	

Figure 3.2 fields of table

- Save the table. Click **File, Save** or click the save icon on the Standard Toolbar.
 - Type the name for the table.
 - Click **Ok**.
 - If you do not want a primary key, click No.
 - Click **View, Datasheet View** or click the view icon on the Standard Toolbar to begin entering data in the table.
- Creating a table using the wizard:**
- Double-click **Create table by using wizard**



The screenshot shows the 'Table Wizard' dialog box. It asks 'Which of the sample tables listed below do you want to use to create your table?'. Below this, it says 'After selecting a table category, choose the sample table and sample fields you want to include in your new table. Your table can include fields from more than one sample table. If you're not sure about a field, go ahead and include it. It's easy to delete a field later.'

On the left, there are two radio buttons: 'Business' (selected) and 'Personal'. Below them is a list of 'Sample Tables': 'Mailing List', 'Contacts', 'Customers', 'Employees', 'Products', and 'Orders'. 'Mailing List' is selected.

In the center, there is a list of 'Sample Fields': 'MailingListID', 'Prefix', 'FirstName', 'MiddleName', 'LastName', 'Suffix', 'Nickname', 'Title', 'OrganizationName', and 'Address'. 'MailingListID' is selected.

On the right, there is a box labeled 'Fields in my new table:' which is currently empty. There are buttons '>', '>>', '<', and '<<' to move fields between the lists.

At the bottom, there are buttons: 'Cancel', '< Back', 'Next >', and 'Finish'. There is also a 'Rename Field...' button next to the 'Fields in my new table:' box.

Figure 3.3 table wizard

- Select the type of table (Business or Personal)
- Choose a table from the **Sample Tables** list
- Select a data field to include in the table
- Click the single arrow
- Repeat steps d and e for all other data fields you wish to include in the table
- Click **Finish**
- Enter data into table

3.3 Creating Tables Manually

To create a table manually, you double-click the Create Table In Design View icon in the Database window to open a blank Table window in Design view. From here, you can add fields, a primary key, and an index. (Primary keys are indexes with special property settings.)

To add a field, type the field's name in a blank Field Name column in the Design view window. Field names follow normal Visual Basic for Applications (VBA) naming conventions. They can be up to 64 characters long, and the characters can be letters, numbers, spaces, and special character except the period, the exclamation mark, square brackets, and the grave accent character ('). Also, you cannot start a field name with a space or a control character (ASCII values 0 through 31). While you can include internal spaces in field names, they must be bracketed in expressions and queries.

3.3.1 Data Types

In the Data Type column, you can specify a data type for the field. A drop-down list box offers 10 options: Text, Memo, Number, Date/Time, Currency, AutoNumber, Yes/No, OLE Object, Hyperlink, and Lookup wizard. (Data types are commonly used to identify the information a field contains. A Text field contains a Text data type, an AutoNumber field contains an AutoNumber data type, and so on.) You can use options within many of these data types to further refine your data type specifications.

Variable Data Types

<i>Name</i>	<i>Number of Bytes</i>	<i>Range</i>
Byte	1	0 through 255
Boolean	2	True or false
Integer	2	-32,768 through 32,767
Long	4	-2,147,483,648 through 2,147,483,647
Single	4	-3.402823E38 through -1.401298E-45 for negative values 1.40129E-45 through 3.402823E38 for positive values
Double	8	-1.79769313486232E308 through -14.94065645841247E-324 for negative values 4.94065645841247E-324 through 1.79769313486232E308 for positive values
Currency	8	-922,337,203,685,477.5808 through 922,337,203,685,477.5807
Date	8	January 1, 100 through December 31, 9999
Object	4	A reference to an object (see <i>Set</i> statement in the online help)
String (fixed)	Length	Up to approximately 64,000 characters
String (variable)	10 + length	Up to approximately 2 billion characters
Variant (with numbers)	16	Same as Double
Variant (with characters)	22 + length	Same as String (variable)
User-defined	Depends on elements	Sum of the constituent elements for the custom data type

3.3.2 AutoNumber fields

AutoNumber field types frequently serve as the primary key for a table. Access automatically assigns a new value to the field when you add a record to the table. This field is not manually updateable, so its values are ideal for uniquely marking a row within a table.

Access automatically sets the value of AutoNumber field types. To cause an AutoNumber field to increment sequentially, select Increment (the default value) from the New Values drop-down list box on the General page at the bottom left of the Table window. To indicate that an AutoNumber field should have a randomly assigned value, select Random from the New Values drop-down list box.

You can use the General and Lookup pages to select other properties that affect the field, such as whether the user must enter a value into the field or whether the field has a default value. Field properties set at the table level propagate through to forms and reports. Table field properties can also simplify the code written for forms and reports. Maintaining data properties at the table level also means that properties are changed in a single place rather than within each form and report that uses a field.

Access 2000 is the first version of Access to enable programmatic control over the initial value and step size of AutoNumber field types. You can use the ALTER TABLE and ALTER COLUMN keywords in Jet SQL to update the next start. The start and step properties of this data type let you programmatically modify the next AutoNumber value and the step size for subsequent values.[2]

3.3.3 Text fields

You use Text fields to hold string entries that contain up to 255 characters. The Text data type can store items such as contact information and numerical values that do not require computation (for example, social security numbers, telephone numbers, and parts numbers). You can also use Text fields in a table to persist computed string values. You

can index primary keys for fast sorts and retrieval based on last name or another Text field type.

3.3.4 Number fields

Number fields are different from Text fields because they can assume a variety of subtypes, ranging from a single byte (Byte subtype) to 16 bytes (Replication ID subtype). The other data subtypes between these extremes include Integer, Long Integer, Single, Double, and Decimal. With the exception of the Byte and Replication ID subtypes. The Byte subtype is similar to the Boolean variable data type. Both types can store Boolean values, but the Byte subtype requires just 1 byte of storage while the Boolean data type requires 2 bytes. The Replication ID data type is not available as a variable data type. Its primary use is in replication, but it serves as a unique identifier. Its length and method of creation make it a more secure way to ensure uniqueness than an AutoNumber field.

The Decimal subtype facilitates the elimination of rounding errors while still accommodating large numbers using Precision and Scale properties. These properties control the number of digits on either side of the decimal point. Precision, which represents the total number of digits that can be stored in the field, can range from 1 through 28. Scale, which indicates the number of digits to the right of the decimal that can be stored in the field, can range from 0 through the value in the Precision property. Because of the Scale property, the Decimal data subtype can store more digits after the decimal point without rounding errors than other Number data subtypes can.

The CurrencyBalance field uses the Currency data type, CurrencyFloat uses the Number data type with the Double subtype, and CurrencyBalanceDec uses the Number data type with the Decimal subtype. The CurrencyBalanceDec field has a Scale property setting of 6, which indicates that the field can store six digits to the right of the decimal point. This is more digits than the Currency data type can precisely represent its limit is four digits after the decimal. The Double data subtype can represent a number with four, five, or six places after the decimal, but it does not perform this task with integer precision. The first row in Persons displays the value 1.0001 in Currency, Double, and Decimal data formats. The

second row expresses 1.00001 in the same three formats. Notice that in Datasheet view the Currency format initially shows 1.00001 as 1.0000 since it is limited to four places after the decimal. The Double and Decimal representations appear identical.

3.3.5 Memo, OLE Object, Date/Time, and Yes/No fields

Other data types included the Memo data type, which holds very large text data strings that can exceed the 255-character limit of the Text data type. A single Memo data type can grow to 64 KB. You can access and write back its contents in 64-KB blocks using the GetChunk and AppendChunk methods. Jet 4 supports indexing the first 255 characters of a Memo field. This is particularly useful for Hyperlink data types that depend on the Memo data type.

OLE Object is another large data type. It works with objects in their binary format, such as a Microsoft Excel workbook or a Microsoft Word document.

Date/Time data types can represent either dates or times. Date values are stored to the left of the decimal point; time values are stored to the right of the decimal point. The Yes/No data type is the smallest. It is always in one of two states either Yes/No, True/False, or On/Off. It occupies a single byte of storage.[2]

3.3.6 Primary Key

A Primary Key is a field or combination of fields that uniquely identify each record in a table. Primary Key features: no two records in a table can have the same value in the primary key field. Records are automatically sorted based on the primary key. Primary Keys perform the following functions:

- Prevent duplicate values.
- Maintains the record order.
- Creates a primary index: indexes are used to improve the speed of queries, reports, and locating records.
- Facilitates relationships to other normalized data in the database. The primary table to be joined must have a primary key field.

NOTE: Memo, Yes/No, OLE object fields, and Hyperlink fields can not be Primary fields.

Entering data

- Double-click on the table name
- Click in the field you wish to enter data
- Press Tab to move to the next field or press Enter to move to the next record

Sorting data

- Click in the data field you wish to sort by
- Click a sort icon on the Standard Toolbar or
- Click Records, Sort
- Select the sort order (Ascending or Descending)

Deleting records

- Click in the record you wish to delete
- Click Edit, Delete Record

Saving a table

- Click File, Save or click the save icon on the Standard Toolbar

Closing a table

- Click File, Close

3.3.7 Working with Forms

While the Access Project user interface delivers extraordinary functionality with remote data sources, you can automate and simplify processes by developing custom programmatic solutions.

Opening a form

When you open a form with the Access Project interface, the form populates a local copy of the remote data in the client workstation. This local copy is a snapshot, at a point in time, of the remote data for the form. When you open the form programmatically, you must create the local copy of the remote data. One advantage to programmatically opening a form is that you can dynamically assign values to the local cache of the remote data. Your application can do this because the recordset that you assign to the form with visual basic for applications overrides the Record Source setting on the form's property sheet.

The procedure below constructs a recordset for a form before opening it. It starts by setting a reference to a new recordset instance: It assigns `adUseClient` to the recordset's `CursorLocation` property to establish the location of the form's data. (Recall that the form gets the data from the local cache on the workstation, not from the remote server.) Next, it opens the recordset with a SQL statement that extracts data from the remote source into the local copy. A commented line shows a SQL statement that can override the form's default record source. After making the local copy of the remote data, the procedure opens the form and assigns the local copy to the form's `Recordset` property. This new property possesses the functionality of the `RecordsetClone` property; in addition, changes to the recordset appear on the form automatically-the `RecordsetClone` property provides a read-only copy of a form's recordset. Like the `RecordsetClone` property, a form's `Recordset` property is available only programmatically.

3.3.8 Switchboard Forms

Switchboard forms are a common way to facilitate navigation in an application. It is common for switchboard forms to contain several command buttons that users can click to open another form. This section offers two approaches to using switchboard forms: one based on hyperlinks and the other based on visual basic for applications procedures

3.3.8.1 Subforms

A subform, one of the most popular ways of displaying data in Access, is a form embedded within a main form. The main form holds general information about an object (such as an order or a patient name). One or more hierarchically related details (such as order line items or patient visits) appear in one or more subforms on the main form. At least one common field must tie the record source of the main form and each subform together. The common field enables the subform to show only records that match the current record in the main form. When the user moves to a new record on the main form, the subform displays a new set of records that tie uniquely to the new record in the main form.

. To create a subform, open the main form in Design view, make sure that the Control Wizards button on the Toolbox is depressed, and then drag a table, query, or form from the Database window and drop it on the main form. The subform appears as a control on the main form. To synchronize the main form and the subform, you must designate at least one common field. Select the subform container and set its Link Child and Link Master properties to the common field.

A main form can have multiple subforms. The only requirement is that the record source for each subform share at least one common field with the record source for the main form. If you define relationships between tables and queries in the Relationships window or by using the properties of a subdatasheet, you can create a main form with an embedded subform as easily as you create a simple bound form. In the Database window, select the table or query on which the main form will be based, and then click the New Object: AutoForm button. The AutoForm wizard will build a main form with an embedded subform. The subform uses the information in the Relationships windows or the subdatasheet. You can manually drag other tables, queries, or forms to the main form in Design view to create additional subform

Creating a form

- From the main database window, click Forms under Objects on the left menu
- Click Create form by using wizard

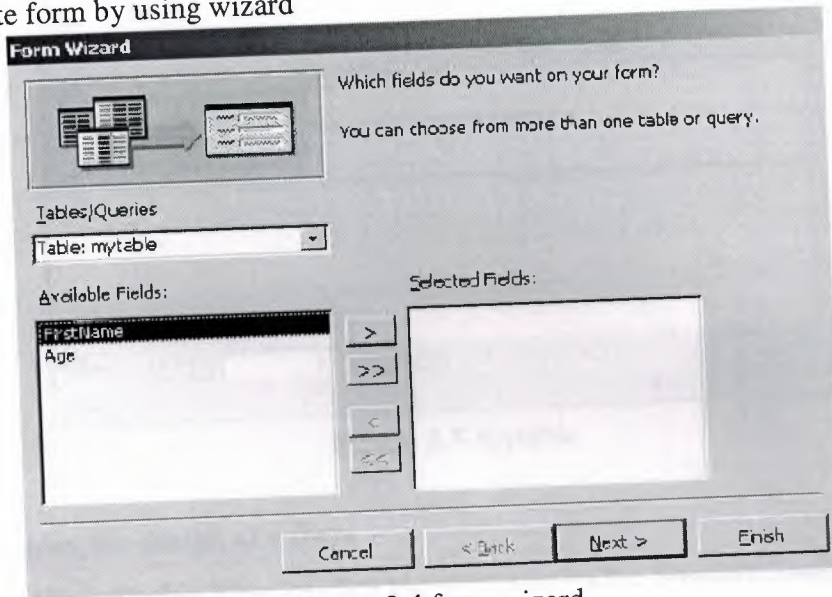


Figure 3.4 form wizard

- In the **Tables/Queries** drop-down list, select the table or query you wish to create the form from
- In the **Available Fields** window, select a data field to include in the form and click the single arrow or
- Click the double arrow to include all data fields in the form
- Click **Next**
- Select a layout for the form, then click **Next**
- Select a style for the form, then click **Next**
- Type a name for the form
- Click **Finish**

Entering data in a form

- Click in a data field and enter the data
- Click tab to move to the next data field
- To move between records, use the arrows on the record menu at the bottom of the form
- To insert a new record, click the arrow with the star (*) beside it

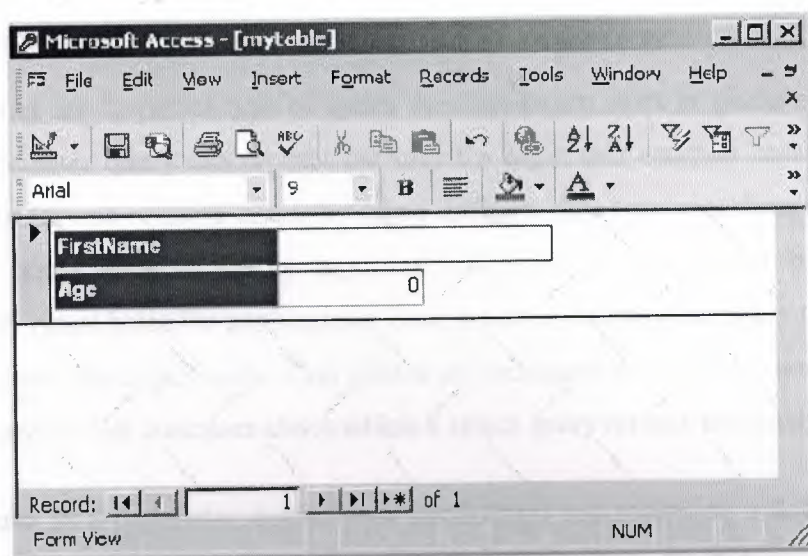


Figure 3.5 Mytable

Changing the design of a form

- Click **View, Design View** or click the view icon on the Standard Toolbar
- To change the style: click **Format, AutoFormat**
- Select the new format, then click **OK**
- To move form objects: position the mouse over the object. When the hand appears, click and hold the mouse down then move the object to its new location.
- To change the color of an object: right-click on the object and select a new fill or font, color.

Saving a form

- Click **File, Save** or click the save icon on the Standard Toolbar

Closing a form

- Click **File, Close**

Working with Queries

You use queries to view, change, and analyze data in different ways. You can also use them as a source of records for forms and reports. There are five types of queries: Select, Parameter, Crosstab, Action, and SQL.[1]

3.3.8.2 Parameter queries

Parameter queries are a special type of query that can return rows or perform actions. At run time, a parameter query can prompt the user for input that controls how it performs. You can prompt for one or more inputs by using different data type specifications. You tell the parameter query what to do by inputting values to its prompts or by setting its parameters with visual basic for applications code before executing the query to control the return set or action that it performs. This allows the designation of a customer ID value at run time to determine the customer about which a select query returns information.

As an alternative to a parameter query, your application can reference a Sql string for a select or action query with string variables. Before executing the Sql statement in an ActiveX Data Objects (ADO) command, assign the string variables specific values. This can provide more flexible results than a parameter query since you can actually alter whole clauses in the Sql statement for a query. For certain cases, parameter queries offset these benefits with data typing and built-in prompts for values. In addition, parameter queries eliminate the need to refine string concatenation statements as you refine your query's Sql statement.

3.4 Creating a Query

- From the main database window, click **Queries** under Objects on the left menu
- Double-click **Create query by using wizard**

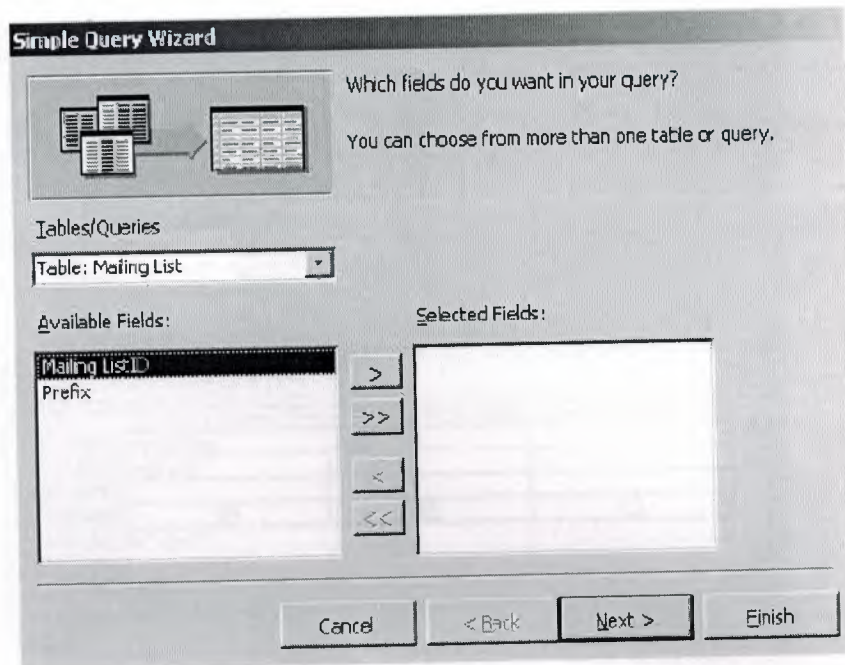


Figure 3.6 simple query wizard

- In the **Tables/Queries** drop-down list, select the table or query you wish to create the query from
- In the **Available Fields** window, select a data field to include in the query and click the single arrow or
- Click the double arrow to include all data fields in the query
- Click **Next**
- Click **Next**
- Type a name for the query, then click **Finish**

Restricting query data

- Click **View**, **Design** view or click the view icon on the Standard Toolbar

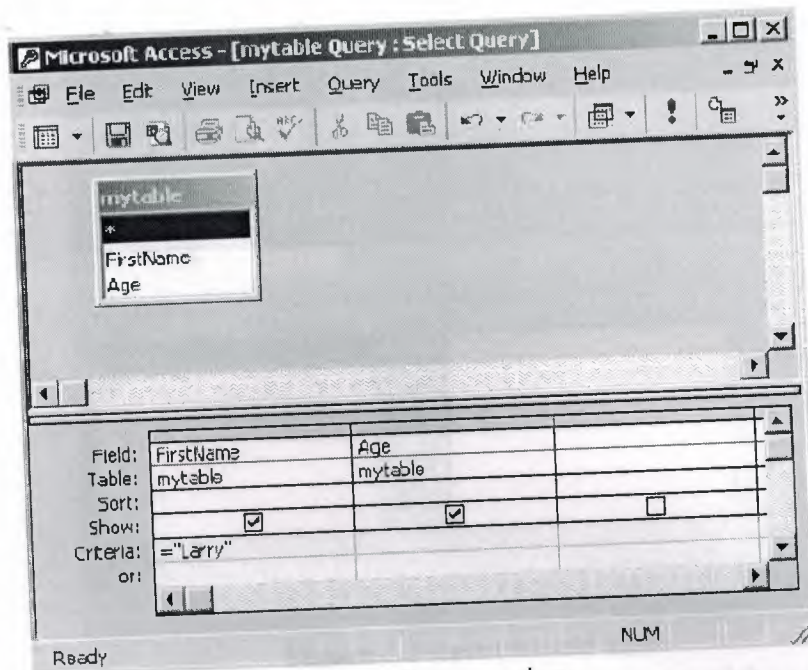


Figure 3.7 Query view

- Click in the criteria box of the field you wish to restrict
- Enter the criteria (i.e. to view records where the first name is Larry, type = "Larry" in the criteria of the FirstName field.)
- Click **View, Datasheet view** or click the view icon on the Standard Toolbar to view the query

Saving a query

- Click File, Save or click the save icon on the Standard Toolbar

Closing a query

- Click File, Close

Working with Reports

You use reports to view, organize, and summarize data in a printable format.

Creating a report

- From the main database window, click Reports under Objects on the left menu
- Double-click Create report by using wizard

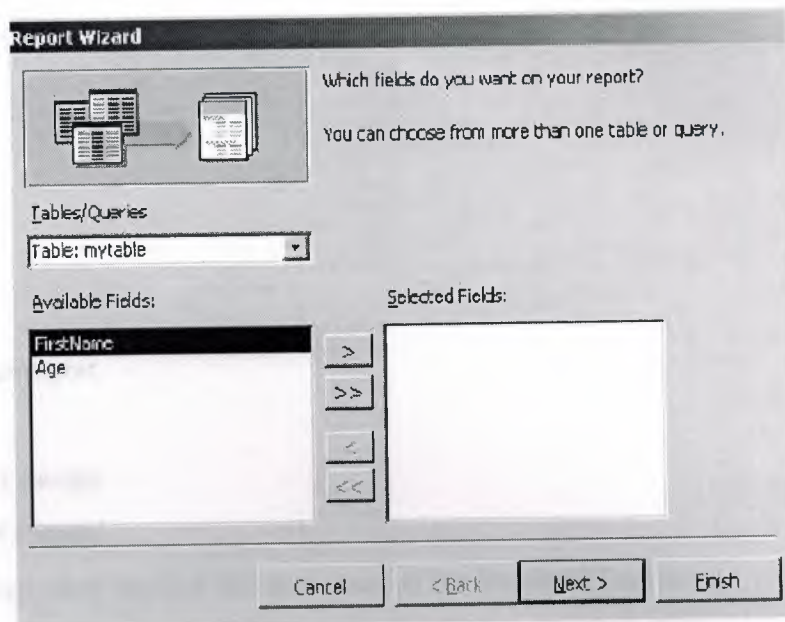


Figure 3.8 report wizard view

- In the **Tables/Queries** drop-down list, select the table or query you wish to create the report from
- In the **Available Fields** window, select a data field to include in the report and click the single arrow or
- Click the double arrow to include all data fields in the report
- Click **Next**
- Click **Next**

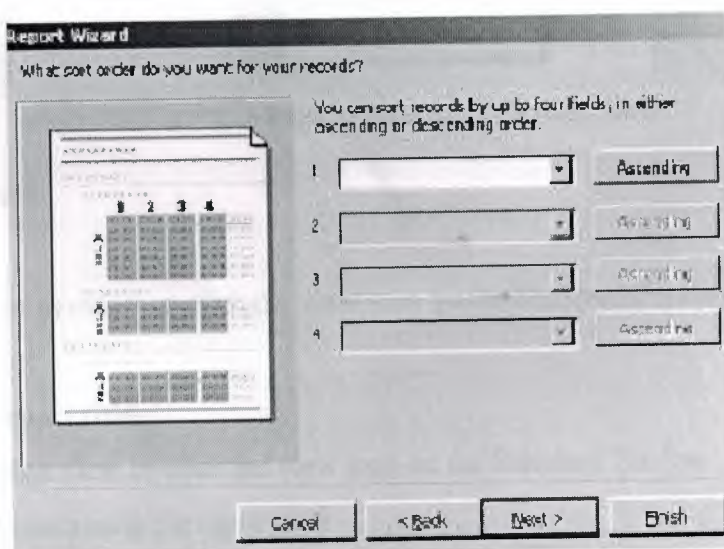


Figure 3.9 _report sorting view

- In the drop-down list, select the data field you wish to sort by
- Click **Next**
- Select a layout
- Click **Next**
- Select a style
- Click **Next**
- Type a title for the report
- Click **Finish**

Changing the report design

Updating the report format

- Click **View, Design view** or click the view icon on the Standard Toolbar
- Click **Format, AutoFormat**

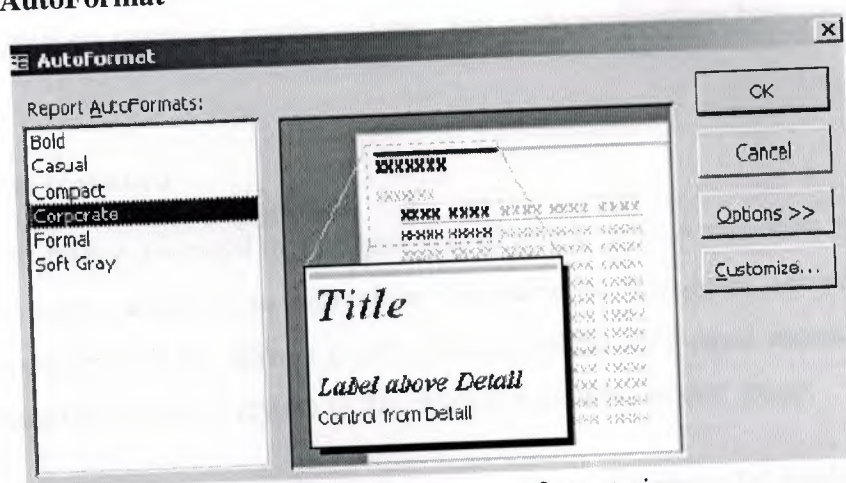


Figure 3.10 report format view

- Select a new format
- Click **Ok**
- Click **View, Print preview** or click the view icon on the Standard Toolbar to return to the report

Changing the report title

- Click **View, Design view** or click the view icon on the Standard Toolbar
- Click in the box containing the report title
- Type the new title for the report

- Click **View, Print preview** or click the view *icon* on the Standard Toolbar to return to the report

Saving a report

- Click **File, Save** or click the save icon on the Standard Toolbar

Printing a report

- Click **File, Print** or click the print icon on the Standard Toolbar

Closing a report

- Click **File, Close**

3.5 Security

Access offers a rich array of security features to support the needs of different types of Access applications. Most multi-user Access applications can benefit from user-level security, which lets developers designate groups of users. But some applications have more specialized needs

3.5.1 Setting a database password

You can require users to enter a password to gain unrestricted access to all Access data and database objects. Passwords are easy to administer compared to user-level security. Password security is appropriate if you have a group whose members need equal access to all elements of a database file but not everyone in the office is a member of that group.

You cannot use a password-protected file as a member in a replica set because Jet database replication cannot synchronize with a password-protected file. You should also be careful about linking to database files with password protection because anyone who can access the file that links the protected file has unrestricted access to the protected file. Furthermore, Access stores an encrypted version of the password along with other information about the linked file. Finally, if someone changes the password for a linked file, Access prompts for the new password the next time another database file links to it.

To assign and remove a database password, you need exclusive access to the file. Take the following steps:

1. Open a file by choosing Open Exclusive from the Open button in the Open dialog box to assign a password to a file.
2. Choose Security-Set Database Password from the Tools menu.
3. In the Set Database Password dialog box, enter your password of choice in the Password and Verify text boxes and then click OK. The next time a user opens the file, the application will ask for the password.
4. After opening a database exclusively, choose Tools-Security-Unset Database Password. Remove the password by typing the password in the Unset Database Password dialog box. This removes the initial prompt for a password before a database is made available.[3]

CHAPTER FOUR: CAR GALLERY DATABASE DESIGN

4.1 Creating Tables for Car Gallery

Firstly I went to Cangar and meet with the manager of Cangar. I collect some information about what the program should be like. After documenting the requirements for the program, I start to design the database. First, I create the “customers” table that is shown in figure-4.1

Field Name	Data Type	Description
customer_id	AutoNumber	
first_name	Text	
last_name	Text	
address1	Text	
address2	Text	
address3	Text	
city	Text	
postal_code	Text	
e_mail	Text	
home_phone	Text	
mobile_phone	Text	
fax_number	Text	
birth_date	Date/Time	
note	Text	

Field Properties	
General	Lookup
Field Size	30
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

Figure 4.1 Table of customer

The “customer” table is for keeping the customer information. Each customer has a unique “customer_id”. The customer information is entered once and can be used many times.

NEAR EAST UNIVERSITY



Faculty of Engineering

Department of Computer Engineering

DATABASE SYSTEM DESIGN

**GRADUATION PROJECT
COM – 400**

Student: Murat BİNGÜL (20010593)

Supervisor: Assist Prof. Dr. Adel Amirjanov

Nicosia - 2005

ACKNOWLEDGMENT

First of all I would like to thank Assit. Prof. Dr. Adil Amirjanov for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

I would like to thanks all of my friends who helped me to overcome my project especially Mehmet Gögebakan

Finally, I would like to thank my family, especially my parents. Their love and guidance saw me through doubtful times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together.

They have made countless sacrifices for my betterment. I can't repay them, but I do hope that their endless efforts will bear fruit and that I may lead them, myself and all who surround me to a better future.

Abstract

In this project development of software on car gallery has been considered. This software compiled with Microsoft Access Programing which is a database program included Microsoft Office Packet.

Nowadays, computers are used almost every area of the business and life, with computers and softwares they increase the speed of our calculations, transactions ...etc

Our application is designed for car galleries which cover all their needs. In this program we are able to hold costumer records, car records, our accounting transactions and we can give reports on our transactions.

With this program maintenance of a car gallery will be easy ,faster and reliable.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENT	I
ABSTRACT	II
TABLE OF CONTENTS	III
LIST OF FIGURES	VI
INTRODUCTION	1
CHAPTER ONE INTRODUCTION TO DATABASE MANAGEMENT	
SYSTEMS DATABASE	2
1.1 Overview.....	2
1.2 Database Models.....	3
1.2.1 Flat Model.....	3
1.2.2 Network Model.....	3
1.2.3 Relational model.....	4
1.3 Why we use a Relational Database Design.....	5
1.3.1 Relationships Between Tables.....	5
1.3.2 One-to-one relationships.....	5
1.3.3 One-to-many relationships.....	6
1.4 Relational Operations.....	8
1.4.1 Implementations and indexing.....	9
1.5 Applications of databases.....	10
1.6 Data modeling.....	10
1.6.1 Database normalization	11
1.6.2 Primary key.....	11
1.6.3 Foreign Key.....	12
1.6.4 Compound Key.....	13

CHAPTER TWO: STRUCTURED QUERY LANGUAGE (SQL)	14
2.1 History.....	14
2.2 Description of SQL.....	15
2.3 SQL keywords.....	16
2.3.1 Data retrieval.....	16
2.3.2 Data manipulation.....	17
2.3.3 Data transaction.....	17
2.3.4 Data definition.....	17
 CHAPTER THREE : MICROSOFT ACCESS DATABASE SYSTEM	 19
3.1 Introductory Microsoft Access.....	19
3.2 Working with Tables.....	19
3.3 Creating Tables Manually.....	21
3.3.1 Data Types.....	21
3.3.2 AutoNumber fields.....	23
3.3.3 Text fields.....	23
3.3.4 Number fields.....	24
3.3.5 Memo, OLE Object, Date/Time, and Yes/No fields.....	25
3.3.6 Primary Key.....	25
3.3.7 Working with Forms.....	26
3.3.8 Switchboard Forms.....	27
3.3.8.1 Subforms.....	28
3.3.8.2 Parameter queries.....	31
3.4 Creating a Query.....	31
3.5 Security.....	36
3.5.1 Setting a database password.....	36

CHAPTER FOUR: CAR GALLERY DATABASE DESIGN

4.1 Creating Tables for Car Gallery.....	38
4.2 Construction relationship between tables.....	44
4.3 Creating forms for our application.....	45
4.4 Sql Operations for Car Gallery.....	49
4.5 Creating report for Car Gallery.....	52
4.6 Applying Switchboard Manager.....	55
4.7 Applying security password to database.....	58
CONCLUSION.....	60
REFERENCES.....	61

LIST OF FIGURE

Figure 3.1 Create table in design view.....	19
Figure 3.2 Fields of table.....	20
Figure 3.3 Table wizard.....	20
Figure 3.4 Form wizard.....	29
Figure 3.5 Mytable.....	30
Figure 3.6 Simple query wizard.....	32
Figure 3.7 Query view.....	33
Figure 3.8 Report wizard view.....	34
Figure 3.9 Report sorting view.....	34
Figure 3.10 Report format view.....	35
Figure 4.1 Table of customer.....	38
Figure-4.2 Creating Tables.....	39
Figure 4.3 Sale table in design view.....	40
Figure 4.4 Details of the Sale table is shown.....	41
Figure 4.5 Structure of stock table.....	42
Figure 4.6 Color table.....	43
Figure 4.7 Producer table.....	43
Figure 4.8 Structure of relationships among tables.....	44
Figure 4.9 Form wizard screen.....	45
Figure 4.10 Next step of form wizard for layout of the form.....	46
Figure 4.11 Next step of form wizard for style design.....	46
Figure 4.12 Modifying the form.....	47
Figure-4.13 Toolbox	47
Figure-4.14 Command button wizard screen.....	48
Figure-4.15 Next step of button wizard screen.....	48
Figure-4.16 View of the customer form.....	49
Figure 4.17 Query is design view.....	49
Figure-4.18 Show table screen.....	50
Figure-4.19 Query design view.....	50
Figure 4.20 Query in sql code view.....	51
Figure-4.21 Structure of total query in sql code view.....	51

Figure-4.22 Create report design view.....	52
Figure 4.23 Report wizard next step view.....	53
Figure 4.24 Selecting fields to order customer table for report.....	53
Figure-4.25 Screens shows that hot to give style to a report.....	54
Figure-4.26 Shows that modifying existent report.....	55
Figure 4.27 Switchboard design.....	55
Figure 4.28 Next steps of switchboard design.....	56
Figure 4.29 Shows that hot to connect all forms to switchboard.....	57
Figure 4.30 Open a file in exclusive mode.....	58
Figure 4.31 Screen for password for the database.....	59

INTRODUCTION

A database is a means of collecting and organizing information. You can create simple ones that contain a list of your business contacts, for example, or you can build a full-featured data management system that you can use to manage a business. Microsoft Access gives you the tools to build just about any kind of database you need.

Access is a popular development platform in large measure because it is part of the Microsoft Office suite. Many clients want their Access systems to interoperate with the rest of Office, and they want systems that are transparent and easy to maintain without developer assistance.

Once you have created a database and added information to it, you can use Access to create forms or reports. With these, you can retrieve the information you've put into the database. Forms and reports can be anything from invoices to accounts receivable summaries to mailing labels to inventory reports and more. With the program's built-in samples, templates, and wizards, it's easy for a beginner to get started. Access 2000 is straightforward enough to allow a beginner to create quick and simple, but very useful databases. As your skills grow, you can add more features that will expand your database's usefulness to you or your business. You will very quickly see just how easy it is, as you begin to create your own database.

In the project development of database system was considered for car gallery with Microsoft access programming.

In Chapter one : discussed about databases, data model, type of databases, relationship, primary and foreign, compound key in detail.

In Chapter two: general SQL structure and its most used keywords presented.

In Chapter three: introductory remarks on Microsoft access programming is given. Creating and modifying of Tables, queries, Report, forms are explained. How to use wizard is illustrated with figures.

In Chapter four, general structure of our application is given. Implementation of the program is given step by step with explanation.

CHAPTER ONE : INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

DATABASE

1.1 Overview

A database is a collection of information stored in a computer in a systematic way, such that a computer program can consult it to answer questions. The software used to manage and query a database is known as a database management system (DBMS). The properties of database systems are studied in information science.

At the core of the concept of a database is the idea of a collection of facts, or pieces of knowledge. Facts may be structured in a number of ways, known as data models. For instance, one model is to associate each fact with a record representing an entity (such as a person), and to arrange these entities into trees or hierarchies the hierarchical data model. Another model is to arrange facts into sets of values which satisfy logical predicates the relational model.

(The terms database and database management system are sometimes interchanged by students. In the professional argot, a database is always the collection of facts, not the software program.)

Database management systems range from the extremely simple to the highly complex. Differences among DBMSes include whether they are capable of ensuring the integrity of the data; whether they may be used by many users at once; and what sorts of conclusions they can be programmed to compute from a set of data.

The first database management systems were developed in the 1960s. A pioneer in the field was Charles Bachman. Two key data models arose at this time: the network model followed by the hierarchical model. These were later usurped by the relational model, which was contemporary with the so-called flat model designed for very small tasks. Another contemporary of the relational model is the object-oriented database (OODB).

While the relational model is based on set theory, one proposed modification suggests fuzzy set theory as an alternative.

1.2 Database Models

Various techniques are used to model data structure. Certain models are more easily implemented by some types of database management systems than others. For any one logical model various physical implementations may be possible. An example of this is the relational model: In larger systems the physical implementation often has indexes which point to the data this is similar to some aspects of common implementations of the network model. But in small relational databases the data is often stored in a set of files, one per table, in a flat, unindexed structure. There is some confusion below and elsewhere in this article as to logical data model vs its physical implementation.[7]

1.2.1 Flat Model

The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another. For instance, columns for name and password might be used as a part of a system security database. Each row would have the specific password associated with a specific user. Columns of the table often have a type associated with them, defining them as character data, date or time information, integers, or floating point numbers. This model is the basis of the spreadsheet.[7]

1.2.2 Network model

The network model allows multiple datasets to be used together through the use of pointers (or references). Some columns contain pointers to different tables instead of data. Thus, the tables are related by references, which can be viewed as a network structure. A particular subset of the network model, the hierarchical model, limits the relationships to a tree structure, instead of the more general directed graph structure implied by the full network model.[7]

1.2.3 Relational model

The relational model was introduced in an academic paper by E. F. Codd in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

Although the basic idea of a relational database has been very popular, relatively few people understand the mathematical definition and only a few, obscure DBMSs implement it completely and without extension. Oracle, for example, can be used in a purely relational way, but it also allows tables to be defined that allow duplicate rows – an extension (or violation) of the relational model. In common English usage, a DBMS is called relational if it supports relational operations, regardless of whether it enforces strict adherence to the relational model. The following is an informal, non-technical explanation of how "relational" database management systems commonly work.

A relational database contains multiple tables, each similar to the one in the "flat" database model. However, unlike network databases, the tables are not linked by pointers. Instead, keys are used to match up rows of data in different tables. A key is just one or more columns in one table that correspond to columns in other tables. Any column can be a key, or multiple columns can be grouped together into a single key. Unlike pointers, it's not necessary to define all the keys in advance; a column can be used as a key even if it wasn't originally intended to be one.

A key that can be used to uniquely identify a row in a table is called a unique key.

Typically one of the unique keys is the preferred way to refer to row; this is defined as the table's primary key.

When a key consists of data that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number), it's called a "natural" key. If no natural key is suitable, an arbitrary key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that can't break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example,

records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

1.3 Why we use a Relational Database Design

Thus, maintaining a simple, so-called flat database consisting of a single table does not require much knowledge of database theory. On the other hand, most databases worth maintaining are quite a bit more complicated than that. Real- life databases often have hundreds of thousands or even millions of records, with data that are very intricately related. This is where using a full- fledged relational database program becomes essential. Consider, for example, the Library of Congress, which has over 16 million books in its collection. For reasons that will become apparent soon, a single table simply will not do for this database!

1.3.1 Relationships Between Tables

When you create tables for an application, you should also consider the relationships between them. These relationships give a relational database much of its power. There are three types of relationships between tables: one-to-one, one-to-many, and many to many.

1.3.2 One-to-one relationships

In a one-to-one relationship, each record in one table corresponds to a single record in a second table. This relationship is not very common, but it can offer several benefits. First, you can put the fields from both tables into a single, combined table. One reason for using two tables is that each field is a property of a separate entity, such as owner operators and their trucks. Each operator can operate just one truck at a time, but the fields for the operator and truck tables refer to different entities.

A one-to-one relationship can also reduce the time needed to open a large table by placing some of the table's columns in a second, separate table. This approach makes particular sense when a table has some fields that are used infrequently. Finally, a one-to-one relationship can support security. Access applies user-level security at the table level.

Therefore, if a subset of the fields in a table requires security, placing them in a separate table lets your application restrict access to certain fields. Your application can link the restricted table back to the main table via a one-to-one relationship so that people with proper permissions can edit, delete, and add new records to those fields.[4]

1.3.3 One-to-many relationships

A one-to-many relationship, in which a row from one table corresponds to one or more rows from a second table, is more common. This kind of relationship can form the basis for a many-to-many relationship as well.

Redundancy

The main problems associated with using a single table to maintain a database stem from the issue of unnecessary repetition of data, that is, redundancy. Some repetition of data is always necessary, but the idea is to remove as much unnecessary repetition as possible. The redundancy in the library flat table (Table 1.1) is obvious. For instance, the name and phone number of Big House publishers is repeated in the table. In an effort to remove as much redundancy as possible from a database, a database designer must split the data into multiple tables. Here is one possibility for the library_flat example, which splits the original database into four separate tables.

- a books table, shown in Table 1.2, in which each book has its own record
- An authors table, shown in Table 1.3, in which each author has his or her own record
- a publishers table, shown in Table 1.4, in which each publisher has its own record
- book/author table, shown in Table 1.5, the purpose of which we will explain a bit later

Table 1.1. The LIBRARY_FLAT Sample Database

ISBN	Title	AuID	AuName	AuPhone	PubID	PubName	PubPhone	Price
1-1111-1111-	C++	4	Roman	444-444-	1	Big House	123-456-	\$29.95

Table 1.2. The BOOKS Table from the LIBRARY_FLAT Database

ISBN	Title	PubID	Price
0-555-55555-9	Macbeth	2	\$12.00
0-91-335678-7	Faerie Queene	1	\$15.00
0-99-999999-9	Emma	1	\$20.00

Table 1.3. The AUTHORS Table from the LIBRARY_FLAT Database

AuID	AuName	AuPhone
1	Austen	111-111-1111
12	Grumpy	321-321-0000
3	Homer	333-333-3333

Table 1.4. The PUBLISHERS Table from the LIBRARY_FLAT Database

PubID	PubName	PubPhone
1	Big House	123-456-7890
2	Alpha Press	999-999-9999
3	Small House	714-000-0000

Table 1.5. The BOOK/AUTHOR Table from the LIBRARY_FLAT Database

ISBN	AuID
0-103-45678-9	3
0-11-345678-9	2
0-12-333433-3	8

Note that now the name and phone number of Big House appears only once in the database (in the publishers table), as does Shakespeare's phone number (in the authors table). Of course, there are still some duplicated data in the database. For instance, the PubID information appears in more than one place in these tables. As mentioned earlier, we

can not eliminate all duplicate data and still maintain the relationships between the data. To get a feel for the reduction in duplicate data achieved by the four-table approach, imagine (as is reasonable) that the database also includes the address of each publisher. Then Table 1.1 would need a new column containing many addresses many of which are duplicates. On the other hand, the four-table database needs only one new column in the publishers table, adding a total of three distinct addresses. To drive the difference home, consider the 16-million-book database of the Library of Congress. Suppose the database contains books from 10,000 different publishers. A publisher's address column in a flat database design would contain 16 million addresses, whereas a multitable approach would require only 10,000 addresses. Now, if the average address is 50 characters long, then the multitable approach would save $(16,000,000 - 10,000) * 50 = 799$ million characters.

Assuming that each character takes 2 bytes (in the Unicode that is used internally by Microsoft Access), the single-table approach wastes about 1.6 gigabytes of space, just for the address field! Indeed, the issue of redundancy alone is quite enough to convince a database designer to avoid the flat database approach. However, there are several other problems with flat databases, which we now discuss.

1.4 Relational Operations

You request data from a relational database by sending it a query that's written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it's much more common for SQL queries to be embedded into software that provides an easier user interface.

In response to a query, the database returns a result set, which is just a list of rows containing the answers. The simplest query is just to return all the rows from a table, but more often, the rows are filtered in some way to return just the answer wanted.

Often, data from multiple tables gets combined into one, by doing a "join". Conceptually, this is done by taking all possible combinations of rows (the "cross-product"), and then

filtering out everything except the answer. In practice, relational database management systems rewrite ("optimize") queries to perform faster, using a variety of techniques.

The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for decades. This has made the idea and implementation of relational databases very popular with businesses.

1.4.1 Implementations and indexing

All of these kinds of database can take advantage of indexing to increase their speed. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Various methods of indexing are commonly used; b-trees, hashes, and linked lists are all common indexing techniques.

Relational DBMSs have the advantage that indices can be created or dropped without changing existing applications, because applications don't use the indices directly. Instead, the database software decides on behalf of the application which indices to use. The database chooses between many different strategies based on which one it estimates will run the fastest.

Relational DBMSs utilise many different algorithms to compute the result of an SQL statement. The RDBMs will produce a plan of how to execute the query, which is generated by analysing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are Nested Loops Join, Sort-Merge Join and Hash Join.

1.5 Applications of databases

Databases are used in many applications, spanning virtually the entire range of computer software. Databases are the preferred method of storage for large multiuser applications, where coordination between many users is needed. Even individual users find them convenient, though, and many electronic mail programs and personal organizers are based on standard database technology.

A **database management system (DBMS)** is a computer program (or more typically, a suite of them) designed to manage a database, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMS use include accounting, human resources and customer support systems. Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

DBMS's are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system

1.6 Data modeling

In information system design, **data modeling** is the analysis and design of the information in the system, concentrating on the logical entities and the logical dependencies between these entities. Data modeling is an abstraction activity in that the details of the values of individual data observations are ignored in favor of the structure, relationships, names and formats of the data of interest, although a list of valid values is frequently recorded. It is by the data model that definitions of what the data means is related to the data structures.

While a common term for this activity is "data analysis" the activity actually has more in common with the ideas and methods of synthesis (putting things together) than it does in the original meaning of the term analysis (taking things apart). This is because the activity strives to bring the data structures of interest together in a cohesive, inseparable, whole by eliminating unnecessary data redundancies and relating data structures by relationships.

In the early phases of a software development project, emphasis will be on the design of a conceptual data model. This can be detailed into a logical data model sometimes called a functional data model. In later stages, this model may be translated into physical data model.

1.6.1 Database normalization

database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

However, many relational DBMSs lack sufficient separation between the logical database design and the physical implementation of the data store, such that queries against a fully normalized database often perform poorly. In this case denormalization is sometimes used to improve performance, at the cost of reduced consistency.

1.6.2 Primary key

In database design, a **primary key** is a value that can be used to identify a particular row in a table. Attributes are associated with it. Examples are names in a telephone book (to look up telephone numbers), words in a dictionary (to look up definitions), and Dewey Decimal Numbers (to look up books in a library).

In the relational model of data, a **primary key** is a candidate key chosen as the main method of uniquely identifying a tuple in a relation. Practical telephone books, dictionaries and libraries can not use names, words or Dewey Decimal System numbers as candidate keys because they do not uniquely identify telephone numbers, word definitions or books. In some design situations it is impossible to find a natural key that uniquely identifies a tuple in a relation. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others [7].

In addition to the requirement that the primary key be a candidate key, there are several other factors which may make a particular choice of key better than others for a given relation.

The primary key should generally be short to minimize the amount of data that needs to be stored by other relations that reference it. A compound key is usually not appropriate. (However, this is a design consideration, and some database management systems may be better than others in this regard.)

The primary key should be immutable, meaning that its value should not be changed during the course of normal operations of the database. (Recall that a primary key is the means of uniquely identifying a tuple, and that identity, by definition, never changes.) This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen.

1.6.3 Foreign Key

A **foreign key** (FK) is a field in a database record that points to a key field of another database record in another table. Usually a foreign key in one table refers to the primary key (PK) of the other table. This way references can be made to link information together and it is an essential part of database normalization.

For example, a person sending an e-mail need not include the entire text of a book in the e-mail. Instead, they can include the ISBN of the book, and interested persons can then use the number to get information about the book - or even the book itself. The ISBN is the primary-key of the book, and it is used as a foreign-key in the e-mail.

Note that using a foreign key often assumes its existence as a primary key somewhere else. Improper foreign key/primary key relationships are the source of many database problems.

1.6.4 Compound Key

In database design, a **compound key** (also called a **composite key**) is a key that consists on 2 or more attributes.

No restriction is applied to the attributes regarding their (initial) ownership within the data model. This means that any one, none, or all, of the multiple attributes within the compound key can be foreign keys. Indeed, a foreign key may, itself, be a compound key.

Compound keys almost always originate from attributive or associative entities (tables) within the model, but this is not an absolute

CHAPTER TWO: STRUCTURED QUERY LANGUAGE (SQL)

2.1 History

Structured Query Language (SQL) is the most popular computer language used to create, modify and retrieve data from relational database management systems. The language has evolved beyond its original purpose to support object-relational database management systems.

A seminal paper, "A Relational Model of Data for Large Shared Data Banks, by Dr. Edgar F. Codd, was published in June, 1970 in the Association for Computing Machinery (acm) journal, Communications of the acm. Codd's model became widely accepted as the definitive model for relational database management systems (rdbms).

During the 1970s, a group at IBM's San Jose research center developed a database system "System R" based upon Codd's model. **Structured English Query Language** ("sequel") was designed to manipulate and retrieve data stored in System R. The acronym sequel was later condensed to **SQL** due to a trademark dispute (the word 'sequel' was held as a trademark by the Hawker-Siddeley aircraft company of the uk).

In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL and, soon, many vendors developed dialects of it.

SQL was adopted as a standard by the ANSI (American National Standards Institute) in 1986 and ISO (International Organization for Standardization) in 1987. ANSI has declared that the official pronunciation for SQL is "es queue el", although many English-speaking database professionals still pronounce it as sequel.

2.2 Description of SQL

SQL allows the specification of queries in a high-level, declarative manner. For example, to select rows from a database, the user need only specify the criteria that they want to search by; the details of performing the search operation efficiently is left up to the database system, and is invisible to the user.

Compared to general-purpose programming languages, this structure allows the user/programmer to be less familiar with the technical details of the data and how they are stored, and relatively more familiar with the information contained in the data. This blurs the line between user and programmer, appealing to individuals who fall more into the 'business' or 'research' area and less in the 'information technology' area. The original vision for SQL was to allow non-technical users to write their own database queries. While this has been realized to some extent, the complexity of querying an advanced database system using SQL can still require a significant learning curve.

SQL contrasts with the more powerful database-oriented fourth-generation programming languages such as Focus or sas, however, in its relative functional simplicity and simpler command set. This greatly reduces the degree of difficulty involved in maintaining the worst SQL source code, but it also makes programming such questions as 'Who had the top ten scores?' more difficult, leading to the development of procedural extensions, discussed above. However, it also makes it possible for SQL source code to be produced (and optimized) by software, leading to the development of a number of natural language database query languages, as well as 'drag and drop' database programming packages with 'object oriented' interfaces. Often these allow the resultant SQL source code to be examined, for educational purposes, further enhancement, or to be used in a different environment

2.3 SQL keywords

SQL keywords fall into several groups.

2.3.1 Data retrieval

The most frequently used operation in transactional databases is the data retrieval operation.

- **SELECT** is used to retrieve zero or more rows from one or more tables in a database. In most applications, **SELECT** is the most commonly used DML command. In specifying a **SELECT** query, the user specifies a description of the desired result set, but they do not specify what physical operations must be executed to produce that result set. Translating the query into an optimal query plan is left to the database system, more specifically to the query optimizer.
- Commonly available keywords related to **SELECT** include:
 - **FROM** is used to indicate which tables the data is to be taken from, as well as how the tables join to each other.
 - **WHERE** is used to identify which rows to be retrieved, or applied to **GROUP BY**.
 - **GROUP BY** is used to combine rows with related values into elements of a smaller set of rows.
 - **HAVING** is used to identify which rows, following a **GROUP BY**, are to be retrieved.
 - **ORDER BY** is used to identify which columns are used to sort the resulting data.

2.3.2 Data manipulation

First there are the standard Data Manipulation Language (DML) elements. DML is the subset of the language used to add, update and delete data.

- INSERT is used to add zero or more rows (formally tuples) to an existing table.
- UPDATE is used to modify the values of a set of existing table rows.
- DELETE removes zero or more existing rows from a table.

2.3.3 Data transaction:

Transaction, if available, can be used to wrap around the DML operations.

BEGIN WORK (or START TRANSACTION, depending on SQL dialect) can be used to mark the start of a database transaction, which either completes completely or not at all.

COMMIT causes all data changes in a transaction to be made permanent.

ROLLBACK causes all data changes since the last COMMIT or ROLLBACK to be discarded, so that the state of the data is "rolled back" to the way it was prior to those changes being requested.

COMMIT and ROLLBACK interact with areas such as transaction control and locking. Strictly, both terminate any open transaction and release any locks held on data. In the absence of a BEGIN WORK or similar statement, the semantics of SQL are implementation-dependent.[3]

2.3.4 Data definition

The second group of keywords is the Data Definition Language (DDL). DDL allows the user to define new tables and associated elements. Most commercial SQL databases have proprietary extensions in their DDL, which allow control over nonstandard features of the database system.

The most basic items of DDL are the CREATE and DROP commands. CREATE causes an object (a table, for example) to be created within the database. DROP causes an existing object within the database to be deleted, usually irretrievably. Some database systems also have an ALTER command, which permits the user to modify an existing object in various ways for example, adding a column to an existing table.

CHAPTER THREE: MICROSOFT ACCESS DATABASE SYSTEM

3.1 Introductory Microsoft Access

- Starting Microsoft Access
- Opening Access
- Click **Start**, select **All Programs** and click on **Microsoft Access**
- Creating a database
 - Click **File, New** or click the new icon on the standard toolbar
 - Select **Blank Database** from the Task Pane menu
 - Type a name for database in the File Name window
 - Click **Create**
- Closing a database
 - Click **File, Close**
- Opening a database
 - Click **File, Open** or click the open icon on the standard toolbar
 - Browse to where the database is saved
 - Click the name of the database
 - Click **Open**

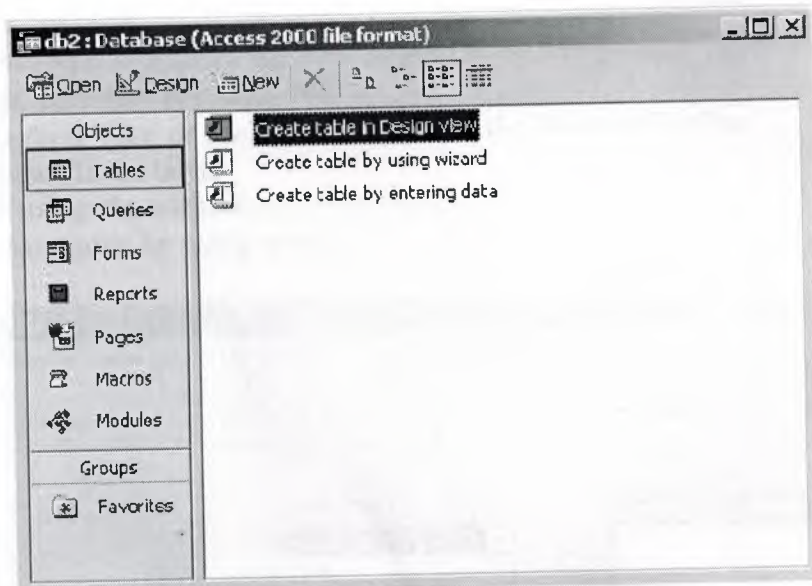
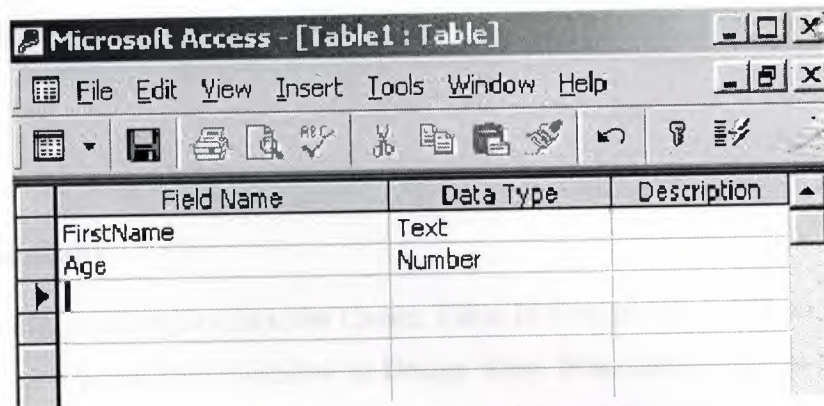


Figure 3.1 create table in design view

3.2 Working with Tables

- A table is a collection of data about a specific topic, such as people, products, or companies.
- Creating a Table
 - From the main database window, click **Tables** under Objects on the left menu
- Creating a table without assistance:
 - Double-click **Create table in design view**
 - In the Field Name column type the name of data field (i.e. FirstName)

- In the Data Type column select the type of data to be entered in the field (i.e. Text, Number, etc.)
- Text, Number, etc.)
- Complete steps b and c for all other data fields (i.e. LastName, Address, etc.)



Field Name	Data Type	Description
FirstName	Text	
Age	Number	

Figure 3.2 fields of table

- Save the table. Click **File, Save** or click the save icon on the Standard Toolbar.
 - Type the name for the table.
 - Click **Ok**.
 - If you do not want a primary key, click No.
 - Click **View, Datasheet View** or click the view icon on the Standard Toolbar to begin entering data in the table.
- Creating a table using the wizard:**
- Double-click **Create table by using wizard**

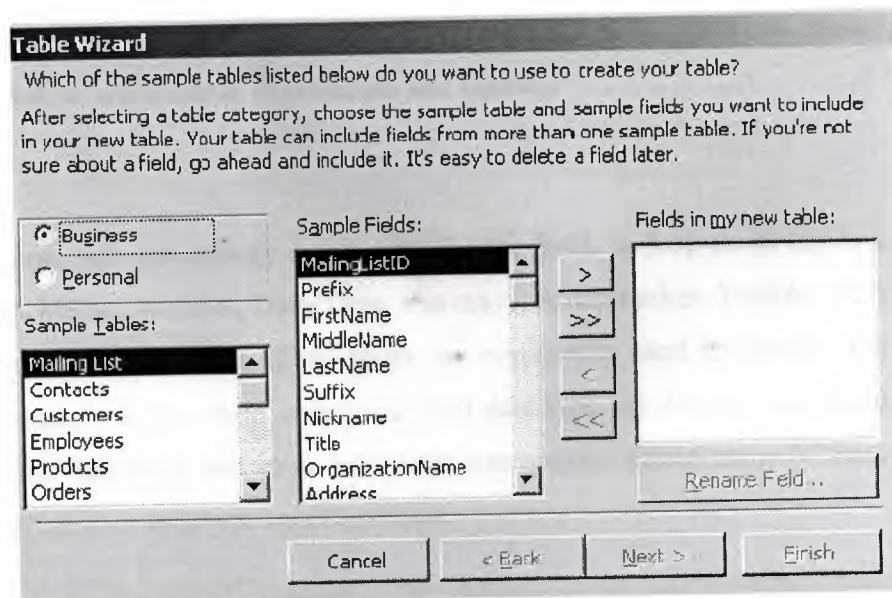


Table Wizard

Which of the sample tables listed below do you want to use to create your table?

After selecting a table category, choose the sample table and sample fields you want to include in your new table. Your table can include fields from more than one sample table. If you're not sure about a field, go ahead and include it. It's easy to delete a field later.

☒ Business
☐ Personal

Sample Tables:

- Mailing List
- Contacts
- Customers
- Employees
- Products
- Orders

Sample Fields:

- MailingListID
- Prefix
- FirstName
- MiddleName
- LastName
- Suffix
- Nickname
- Title
- OrganizationName
- Address

Fields in my new table:

Rename Field...

Cancel < Back Next > Finish

Figure 3.3 table wizard

- Select the type of table (Business or Personal)
- Choose a table from the **Sample Tables** list
- Select a data field to include in the table
- Click the single arrow
- Repeat steps d and e for all other data fields you wish to include in the table
- Click **Finish**
- Enter data into table

3.3 Creating Tables Manually

To create a table manually, you double-click the Create Table In Design View icon in the Database window to open a blank Table window in Design view. From here, you can add fields, a primary key, and an index. (Primary keys are indexes with special property settings.)

To add a field, type the field's name in a blank Field Name column in the Design view window. Field names follow normal Visual Basic for Applications (VBA) naming conventions. They can be up to 64 characters long, and the characters can be letters, numbers, spaces, and special character except the period, the exclamation mark, square brackets, and the grave accent character (`). Also, you cannot start a field name with a space or a control character (ASCII values 0 through 31). While you can include internal spaces in field names, they must be bracketed in expressions and queries.

3.3.1 Data Types

In the Data Type column, you can specify a data type for the field. A drop-down list box offers 10 options: Text, Memo, Number, Date/Time, Currency, AutoNumber, Yes/No, OLE Object, Hyperlink, and Lookup wizard. (Data types are commonly used to identify the information a field contains. A Text field contains a Text data type, an AutoNumber field contains an AutoNumber data type, and so on.) You can use options within many of these data types to further refine your data type specifications.

Variable Data Types

<i>Name</i>	<i>Number of Bytes</i>	<i>Range</i>
Byte	1	0 through 255
Boolean	2	True or false
Integer	2	-32,768 through 32,767
Long	4	-2,147,483,648 through 2,147,483,647
Single	4	-3.402823E38 through -1.401298E-45 for negative values 1.40129E-45 through 3.402823E38 for positive values
Double	8	-1.79769313486232E308 through -14.94065645841247E-324 for negative values 4.94065645841247E-324 through 1.79769313486232E308 for positive values
Currency	8	-922,337,203,685,477.5808 through 922,337,203,685,477.5807
Date	8	January 1, 100 through December 31, 9999
Object	4	A reference to an object (see <i>Set</i> statement in the online help)
String (fixed)	Length	Up to approximately 64,000 characters
String (variable)	10 + length	Up to approximately 2 billion characters
Variant (with numbers)	16	Same as Double
Variant (with characters)	22 + length	Same as String (variable)
User-defined	Depends on elements	Sum of the constituent elements for the custom data type

3.3.2 AutoNumber fields

AutoNumber field types frequently serve as the primary key for a table. Access automatically assigns a new value to the field when you add a record to the table. This field is not manually updateable, so its values are ideal for uniquely marking a row within a table.

Access automatically sets the value of AutoNumber field types. To cause an AutoNumber field to increment sequentially, select Increment (the default value) from the New Values drop-down list box on the General page at the bottom left of the Table window. To indicate that an AutoNumber field should have a randomly assigned value, select Random from the New Values drop-down list box.

You can use the General and Lookup pages to select other properties that affect the field, such as whether the user must enter a value into the field or whether the field has a default value. Field properties set at the table level propagate through to forms and reports. Table field properties can also simplify the code written for forms and reports. Maintaining data properties at the table level also means that properties are changed in a single place rather than within each form and report that uses a field.

Access 2000 is the first version of Access to enable programmatic control over the initial value and step size of AutoNumber field types. You can use the ALTER TABLE and ALTER COLUMN keywords in Jet SQL to update the next start. The start and step properties of this data type let you programmatically modify the next AutoNumber value and the step size for subsequent values.[2]

3.3.3 Text fields

You use Text fields to hold string entries that contain up to 255 characters. The Text data type can store items such as contact information and numerical values that do not require computation (for example, social security numbers, telephone numbers, and parts numbers). You can also use Text fields in a table to persist computed string values. You

can index primary keys for fast sorts and retrieval based on last name or another Text field type.

3.3.4 Number fields

Number fields are different from Text fields because they can assume a variety of subtypes, ranging from a single byte (Byte subtype) to 16 bytes (Replication ID subtype). The other data subtypes between these extremes include Integer, Long Integer, Single, Double, and Decimal. With the exception of the Byte and Replication ID subtypes. The Byte subtype is similar to the Boolean variable data type. Both types can store Boolean values, but the Byte subtype requires just 1 byte of storage while the Boolean data type requires 2 bytes. The Replication ID data type is not available as a variable data type. Its primary use is in replication, but it serves as a unique identifier. Its length and method of creation make it a more secure way to ensure uniqueness than an AutoNumber field.

The Decimal subtype facilitates the elimination of rounding errors while still accommodating large numbers using Precision and Scale properties. These properties control the number of digits on either side of the decimal point. Precision, which represents the total number of digits that can be stored in the field, can range from 1 through 28. Scale, which indicates the number of digits to the right of the decimal that can be stored in the field, can range from 0 through the value in the Precision property. Because of the Scale property, the Decimal data subtype can store more digits after the decimal point without rounding errors than other Number data subtypes can.

The CurrencyBalance field uses the Currency data type, CurrencyFloat uses the Number data type with the Double subtype, and CurrencyBalanceDec uses the Number data type with the Decimal subtype. The CurrencyBalanceDec field has a Scale property setting of 6, which indicates that the field can store six digits to the right of the decimal point. This is more digits than the Currency data type can precisely represent its limit is four digits after the decimal. The Double data subtype can represent a number with four, five, or six places after the decimal, but it does not perform this task with integer precision. The first row in Persons displays the value 1.0001 in Currency, Double, and Decimal data formats. The

second row expresses 1.00001 in the same three formats. Notice that in Datasheet view the Currency format initially shows 1.00001 as 1.0000 since it is limited to four places after the decimal. The Double and Decimal representations appear identical.

3.3.5 Memo, OLE Object, Date/Time, and Yes/No fields

Other data types included the Memo data type, which holds very large text data strings that can exceed the 255-character limit of the Text data type. A single Memo data type can grow to 64 KB. You can access and write back its contents in 64-KB blocks using the GetChunk and AppendChunk methods. Jet 4 supports indexing the first 255 characters of a Memo field. This is particularly useful for Hyperlink data types that depend on the Memo data type.

OLE Object is another large data type. It works with objects in their binary format, such as a Microsoft Excel workbook or a Microsoft Word document.

Date/Time data types can represent either dates or times. Date values are stored to the left of the decimal point; time values are stored to the right of the decimal point. The Yes/No data type is the smallest. It is always in one of two states either Yes/No, True/False, or On/Off. It occupies a single byte of storage.[2]

3.3.6 Primary Key

A Primary Key is a field or combination of fields that uniquely identify each record in a table. Primary Key features: no two records in a table can have the same value in the primary key field. Records are automatically sorted based on the primary key. Primary Keys perform the following functions:

- Prevent duplicate values.
- Maintains the record order.
- Creates a primary index: indexes are used to improve the speed of queries, reports, and locating records.
- Facilitates relationships to other normalized data in the database. The primary table to be joined must have a primary key field.

NOTE: Memo, Yes/No, OLE object fields, and Hyperlink fields can not be Primary fields.

Entering data

- Double-click on the table name
- Click in the field you wish to enter data
- Press Tab to move to the next field or press Enter to move to the next record

Sorting data

- Click in the data field you wish to sort by
- Click a sort icon on the Standard Toolbar or
- Click Records, Sort
- Select the sort order (Ascending or Descending)

Deleting records

- Click in the record you wish to delete
- Click Edit, Delete Record

Saving a table

- Click File, Save or click the save icon on the Standard Toolbar

Closing a table

- Click File, Close

3.3.7 Working with Forms

While the Access Project user interface delivers extraordinary functionality with remote data sources, you can automate and simplify processes by developing custom programmatic solutions.

Opening a form

When you open a form with the Access Project interface, the form populates a local copy of the remote data in the client workstation. This local copy is a snapshot, at a point in time, of the remote data for the form. When you open the form programmatically, you must create the local copy of the remote data. One advantage to programmatically opening a form is that you can dynamically assign values to the local cache of the remote data. Your application can do this because the recordset that you assign to the form with visual basic for applications overrides the Record Source setting on the form's property sheet.

The procedure below constructs a recordset for a form before opening it. It starts by setting a reference to a new recordset instance: It assigns `adUseClient` to the recordset's `CursorLocation` property to establish the location of the form's data. (Recall that the form gets the data from the local cache on the workstation, not from the remote server.) Next, it opens the recordset with a SQL statement that extracts data from the remote source into the local copy. A commented line shows a SQL statement that can override the form's default record source. After making the local copy of the remote data, the procedure opens the form and assigns the local copy to the form's `Recordset` property. This new property possesses the functionality of the `RecordsetClone` property; in addition, changes to the recordset appear on the form automatically-the `RecordsetClone` property provides a read-only copy of a form's recordset. Like the `RecordsetClone` property, a form's `Recordset` property is available only programmatically.

3.3.8 Switchboard Forms

Switchboard forms are a common way to facilitate navigation in an application. It is common for switchboard forms to contain several command buttons that users can click to open another form. This section offers two approaches to using switchboard forms: one based on hyperlinks and the other based on visual basic for applications procedures

3.3.8.1 Subforms

A subform, one of the most popular ways of displaying data in Access, is a form embedded within a main form. The main form holds general information about an object (such as an order or a patient name). One or more hierarchically related details (such as order line items or patient visits) appear in one or more subforms on the main form. At least one common field must tie the record source of the main form and each subform together. The common field enables the subform to show only records that match the current record in the main form. When the user moves to a new record on the main form, the subform displays a new set of records that tie uniquely to the new record in the main form.

. To create a subform, open the main form in Design view, make sure that the Control Wizards button on the Toolbox is depressed, and then drag a table, query, or form from the Database window and drop it on the main form. The subform appears as a control on the main form. To synchronize the main form and the subform, you must designate at least one common field. Select the subform container and set its Link Child and Link Master properties to the common field.

A main form can have multiple subforms. The only requirement is that the record source for each subform share at least one common field with the record source for the main form. If you define relationships between tables and queries in the Relationships window or by using the properties of a subdatasheet, you can create a main form with an embedded subform as easily as you create a simple bound form. In the Database window, select the table or query on which the main form will be based, and then click the New Object: AutoForm button. The AutoForm wizard will build a main form with an embedded subform. The subform uses the information in the Relationships windows or the subdatasheet. You can manually drag other tables, queries, or forms to the main form in Design view to create additional subform

Creating a form

- From the main database window, click Forms under Objects on the left menu
- Click Create form by using wizard

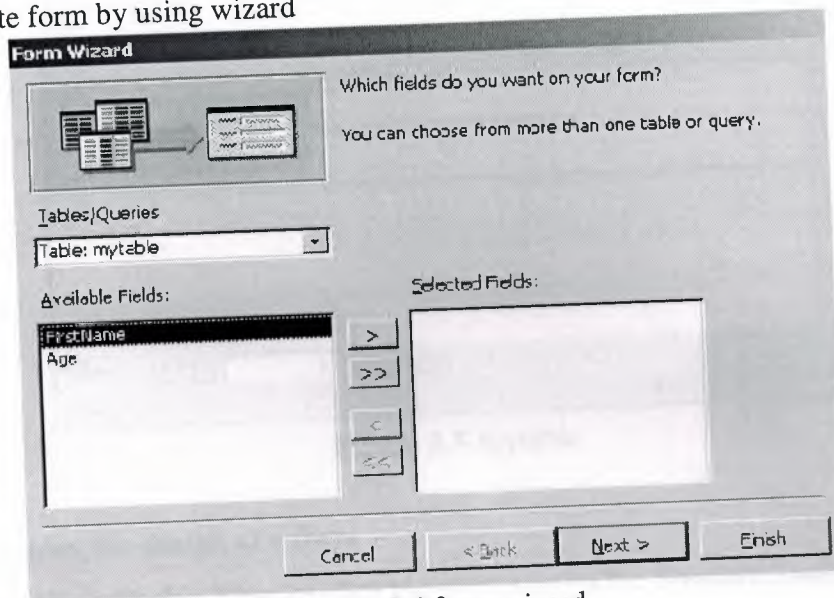


Figure 3.4 form wizard

- In the **Tables/Queries** drop-down list, select the table or query you wish to create the form from
- In the **Available Fields** window, select a data field to include in the form and click the single arrow or
- Click the double arrow to include all data fields in the form
- Click **Next**
- Select a layout for the form, then click **Next**
- Select a style for the form, then click **Next**
- Type a name for the form
- Click **Finish**

Entering data in a form

- Click in a data field and enter the data
- Click tab to move to the next data field
- To move between records, use the arrows on the record menu at the bottom of the form
- To insert a new record, click the arrow with the star (*) beside it

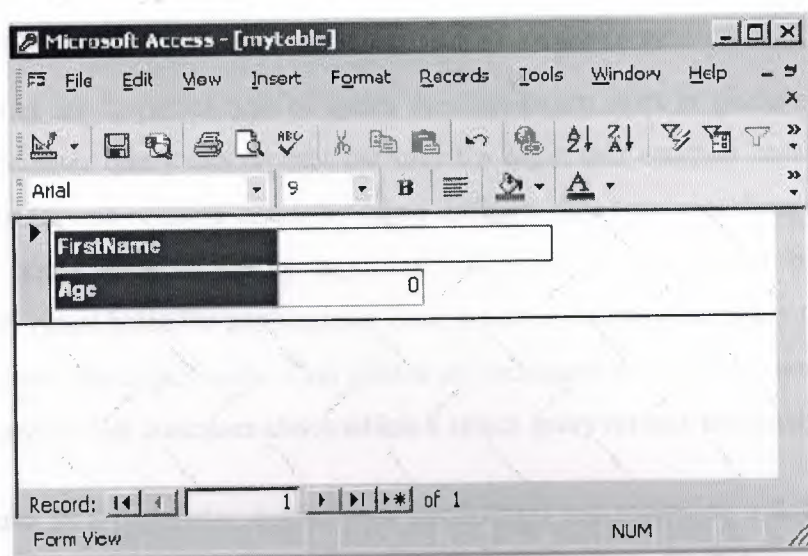


Figure 3.5 Mytable

Changing the design of a form

- Click **View, Design View** or click the view icon on the Standard Toolbar
- To change the style: click **Format, AutoFormat**
- Select the new format, then click **OK**
- To move form objects: position the mouse over the object. When the hand appears, click and hold the mouse down then move the object to its new location.
- To change the color of an object: right-click on the object and select a new fill or font, color.

Saving a form

- Click **File, Save** or click the save icon on the Standard Toolbar

Closing a form

- Click **File, Close**

Working with Queries

You use queries to view, change, and analyze data in different ways. You can also use them as a source of records for forms and reports. There are five types of queries: Select, Parameter, Crosstab, Action, and SQL.[1]

3.3.8.2 Parameter queries

Parameter queries are a special type of query that can return rows or perform actions. At run time, a parameter query can prompt the user for input that controls how it performs. You can prompt for one or more inputs by using different data type specifications. You tell the parameter query what to do by inputting values to its prompts or by setting its parameters with visual basic for applications code before executing the query to control the return set or action that it performs. This allows the designation of a customer ID value at run time to determine the customer about which a select query returns information.

As an alternative to a parameter query, your application can reference a Sql string for a select or action query with string variables. Before executing the Sql statement in an ActiveX Data Objects (ADO) command, assign the string variables specific values. This can provide more flexible results than a parameter query since you can actually alter whole clauses in the Sql statement for a query. For certain cases, parameter queries offset these benefits with data typing and built-in prompts for values. In addition, parameter queries eliminate the need to refine string concatenation statements as you refine your query's Sql statement.

3.4 Creating a Query

- From the main database window, click **Queries** under Objects on the left menu
- Double-click **Create query by using wizard**

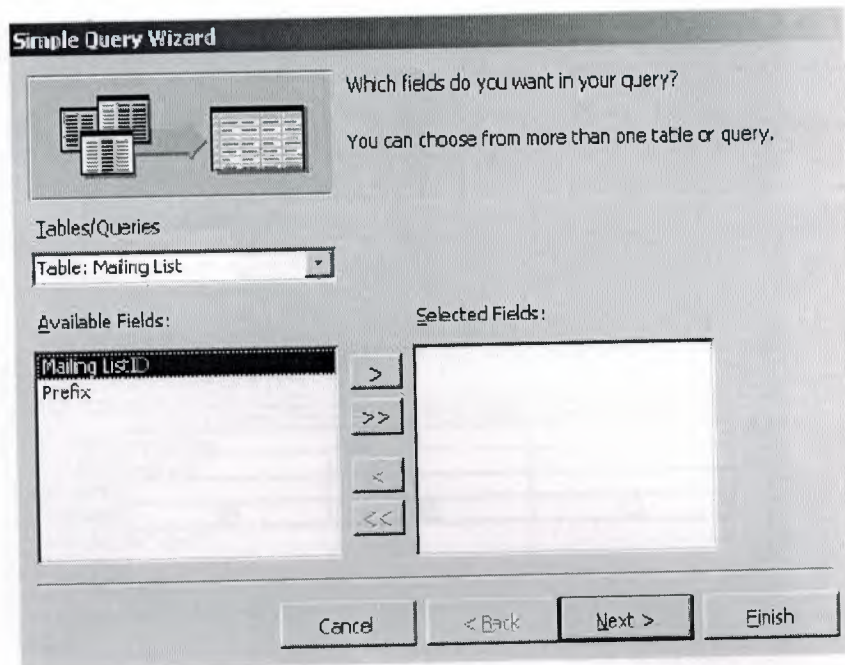


Figure 3.6 simple query wizard

- In the **Tables/Queries** drop-down list, select the table or query you wish to create the query from
- In the **Available Fields** window, select a data field to include in the query and click the single arrow or
- Click the double arrow to include all data fields in the query
- Click **Next**
- Click **Next**
- Type a name for the query, then click **Finish**

Restricting query data

- Click **View**, **Design** view or click the view icon on the Standard Toolbar

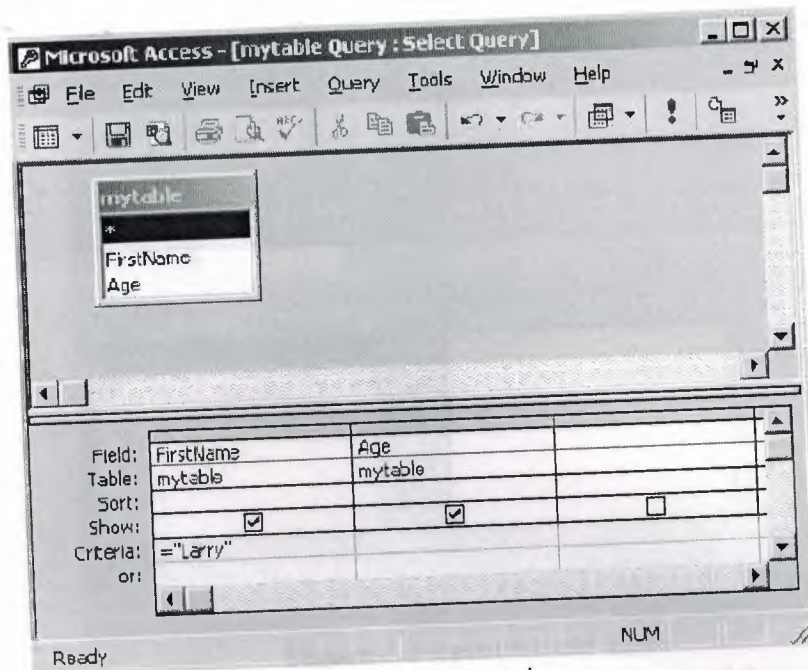


Figure 3.7 Query view

- Click in the criteria box of the field you wish to restrict
- Enter the criteria (i.e. to view records where the first name is Larry, type = "Larry" in the criteria of the FirstName field.)
- Click **View, Datasheet view** or click the view icon on the Standard Toolbar to view the query

Saving a query

- Click File, Save or click the save icon on the Standard Toolbar

Closing a query

- Click File, Close

Working with Reports

You use reports to view, organize, and summarize data in a printable format.

Creating a report

- From the main database window, click Reports under Objects on the left menu
- Double-click Create report by using wizard

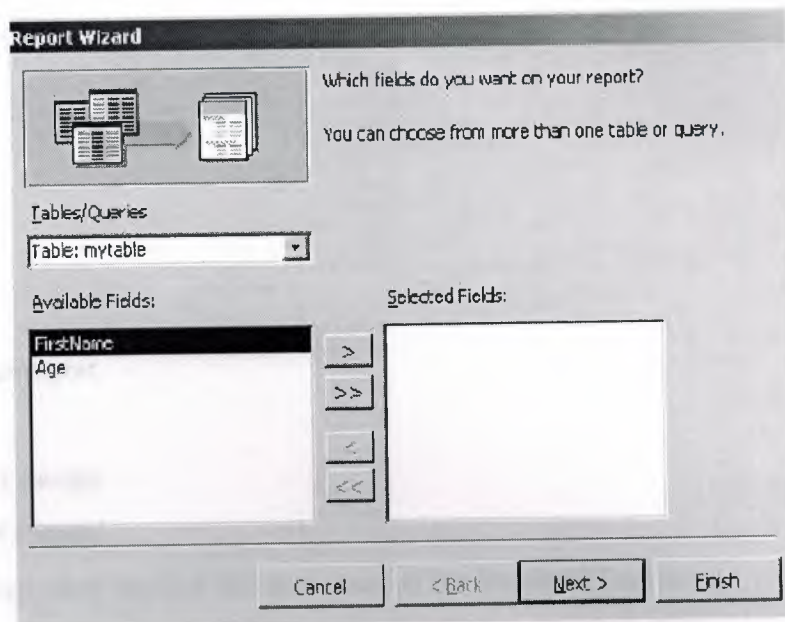


Figure 3.8 report wizard view

- In the **Tables/Queries** drop-down list, select the table or query you wish to create the report from
- In the **Available Fields** window, select a data field to include in the report and click the single arrow or
- Click the double arrow to include all data fields in the report
- Click **Next**
- Click **Next**

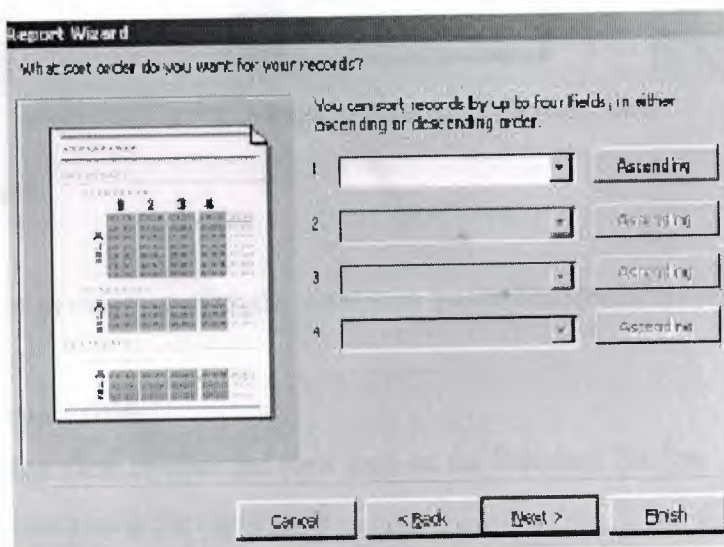


Figure 3.9 _report sorting view

- In the drop-down list, select the data field you wish to sort by
- Click **Next**
- Select a layout
- Click **Next**
- Select a style
- Click **Next**
- Type a title for the report
- Click **Finish**

Changing the report design

Updating the report format

- Click **View, Design view** or click the view icon on the Standard Toolbar
- Click **Format, AutoFormat**

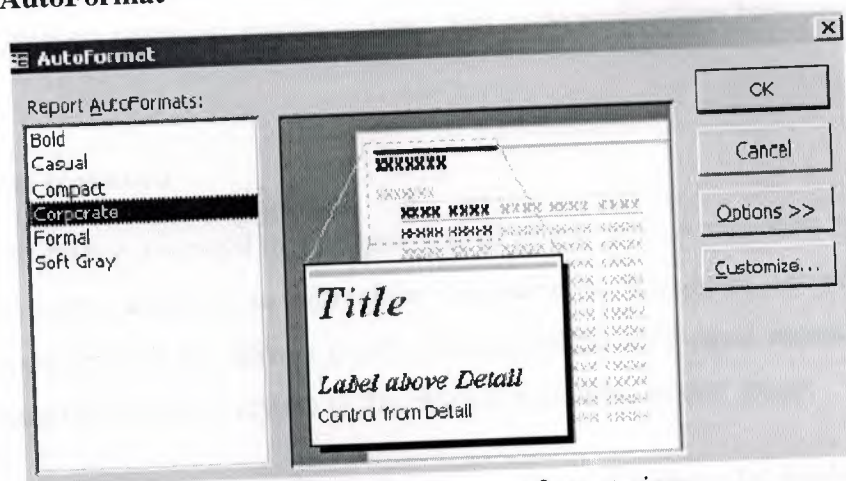


Figure 3.10 report format view

- Select a new format
- Click **Ok**
- Click **View, Print preview** or click the view icon on the Standard Toolbar to return to the report

Changing the report title

- Click **View, Design view** or click the view icon on the Standard Toolbar
- Click in the box containing the report title
- Type the new title for the report

- Click **View, Print preview** or click the view *icon* on the Standard Toolbar to return to the report

Saving a report

- Click **File, Save** or click the save icon on the Standard Toolbar

Printing a report

- Click **File, Print** or click the print icon on the Standard Toolbar

Closing a report

- Click **File, Close**

3.5 Security

Access offers a rich array of security features to support the needs of different types of Access applications. Most multi-user Access applications can benefit from user-level security, which lets developers designate groups of users. But some applications have more specialized needs

3.5.1 Setting a database password

You can require users to enter a password to gain unrestricted access to all Access data and database objects. Passwords are easy to administer compared to user-level security. Password security is appropriate if you have a group whose members need equal access to all elements of a database file but not everyone in the office is a member of that group.

You cannot use a password-protected file as a member in a replica set because Jet database replication cannot synchronize with a password-protected file. You should also be careful about linking to database files with password protection because anyone who can access the file that links the protected file has unrestricted access to the protected file. Furthermore, Access stores an encrypted version of the password along with other information about the linked file. Finally, if someone changes the password for a linked file, Access prompts for the new password the next time another database file links to it.

To assign and remove a database password, you need exclusive access to the file. Take the following steps:

1. Open a file by choosing Open Exclusive from the Open button in the Open dialog box to assign a password to a file.
2. Choose Security-Set Database Password from the Tools menu.
3. In the Set Database Password dialog box, enter your password of choice in the Password and Verify text boxes and then click OK. The next time a user opens the file, the application will ask for the password.
4. After opening a database exclusively, choose Tools-Security-Unset Database Password. Remove the password by typing the password in the Unset Database Password dialog box. This removes the initial prompt for a password before a database is made available.[3]

CHAPTER FOUR: CAR GALLERY DATABASE DESIGN

4.1 Creating Tables for Car Gallery

Firstly I went to Cangar and meet with the manager of Cangar. I collect some information about what the program should be like. After documenting the requirements for the program, I start to design the database. First, I create the “customers” table that is shown in figure-4.1

Field Name	Data Type	Description
customer_id	AutoNumber	
first_name	Text	
last_name	Text	
address1	Text	
address2	Text	
address3	Text	
city	Text	
postal_code	Text	
e_mail	Text	
home_phone	Text	
mobile_phone	Text	
fax_number	Text	
birth_date	Date/Time	
note	Text	

Field Properties	
General	Lookup
Field Size	30
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

Figure 4.1 Table of customer

The “customer” table is for keeping the customer information. Each customer has a unique “customer_id”. The customer information is entered once and can be used many times.

CHAPTER FOUR: CAR GALLERY DATABASE DESIGN

4.1 Creating Tables for Car Gallery

Firstly I went to Cangar and meet with the manager of Cangar. I collect some information about what the program should be like. After documenting the requirements for the program, I start to design the database. First, I create the “costumers” table that is shown in figure-4.1

Field Name	Data Type	Description
customer_id	AutoNumber	
first_name	Text	
last_name	Text	
address1	Text	
address2	Text	
address3	Text	
city	Text	
postal_code	Text	
e_mail	Text	
home_phone	Text	
mobile_phone	Text	
fax_number	Text	
birth_date	Date/Time	
note	Text	

Field Properties	
General	Lookup
Field Size	30
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

Figure 4.1 Table of customer

The “customer” table is for keeping the customer information. Each customer has a unique “customer_id”. The customer information is entered once and can be used many times.

Then, I create “properties” table as shown in figure-4.2

Field Name	Data Type	Description
model	Text	
producer_id	Text	
motor	Text	
cylinder	Text	
hp	Text	
tork	Text	
gearbox	Text	
trans	Text	
speed	Text	
fuel_cons	Text	
baggage	Text	
weight	Text	
w/l/h	Text	
picture	OLE Object	

Field Properties	
General	
Field Size	50
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

Figure-4.2 Creating Tables

The purpose of the “properties” table is to keep the general information specified with the brand and its model. When a new model of a brand comes in to the market, the general information of that car is entered to the database. This process is done just once for every brand and its model.

For this table, I can't use a single primary key. Because as usual, one brand can have more than one models. So, composite key should be used. This means, there can be many brands with different models.

Then I create sale table. It can be seen in Figure-4.3

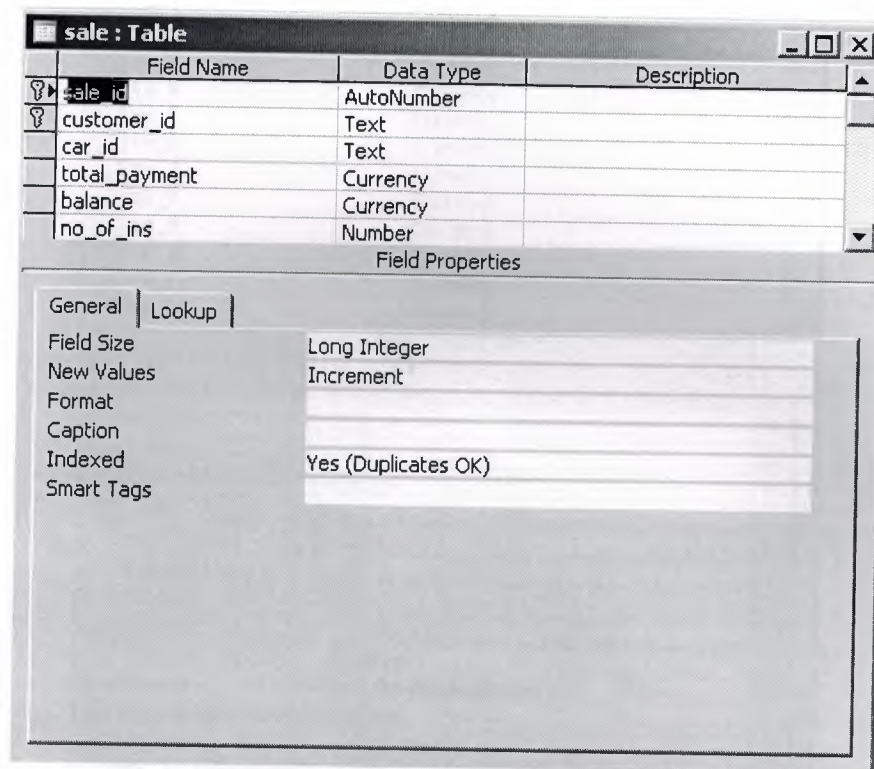


Figure 4.3 Sale table in design view

This table is used for keeping the general information of a sale. Each sale is defined with a "sale_id" and it includes the information of whom the sale is realized with customer_id field, Which car is sold, the price of the saled car, balance of the customer, and the number of instalments.

For this table, one primary key can not be used. Because one customer can buy more than one car. That is why I use composite key.

Then I create sale_detail table. It can be seen in figure-4.4

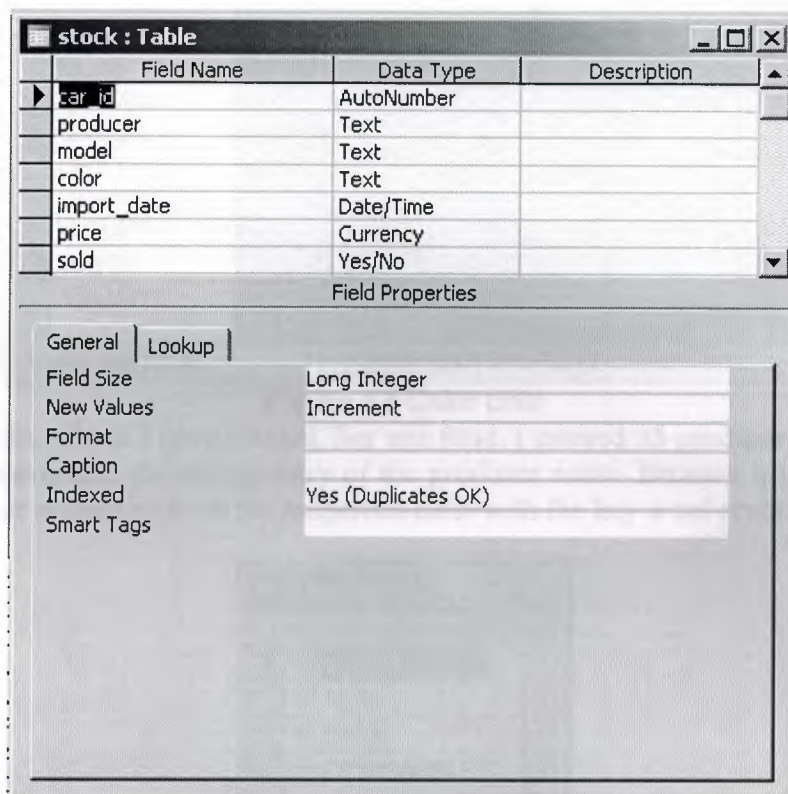
Field Name	Data Type	Description
sale_id	Text	
cash_paid	Currency	
ins_1	Currency	
ins_2	Currency	
ins_3	Currency	
ins_4	Currency	
ins_5	Currency	
ins_6	Currency	
ins_7	Currency	
ins_8	Currency	
ins_9	Currency	
ins_10	Currency	
ins_11	Currency	
ins_12	Currency	

Field Properties	
General	
Field Size	10
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

Figure 4.4 Details of the Sale table is shown.

This table keeps the payment detail information about a specific sales. Like the amounth of each instalmants. Sale_id is set as primary key.

Then I create stock table. It can be seen in Figure 4.5



Field Name	Data Type	Description
car_id	AutoNumber	
producer	Text	
model	Text	
color	Text	
import_date	Date/Time	
price	Currency	
sold	Yes/No	

Field Properties	
General	
Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (Duplicates OK)
Smart Tags	

Figure 4.5 Structure of stock table

This table is keeping the information of the stock. Actually the field “car_id”, which is set as primary key, is the chassis number of the car. The producer and the model of the car is also kept in the table because these fields are conected as foreigne key with referance to the properties table. By this way the genral information about the specific car can be reached. I also want to add a “sold” field to the table. The purpose of this field is to keep the information if the car is sold or not. This enables me to keep the information of the specific car even after it is sold.

Then I create the color table (figure-4.6). Colors table has one field and helps to reduce the memory usage.

color
BLACK
BLUE
BROWN
GREEN
GREY
ORANGE
VIOLET
WHITE
YELLOW

Figure 4.6 Color table

Producers tables, which I have created, has one field. I entered 35 producer names. This is actually for preventing the wrong entry of the producer name. Because it is important for me. I search car properties from the properties table with the key-word producer name.

producer
ALFA ROMEO
AUDI
BMW
CEHEVROLET
CHRYSLER
CITROEN
DACIA
DAIHATSU
FIAT
FORD
HONDA
HYUNDAI
ISUZU
JEEP
KIA
LADA
LAND ROVER
MAZDA
MERCEDES

Figure 4.7 Producer table

4.2 Construction relationship between tables

I established relationships among tables shown in Figure 4.8

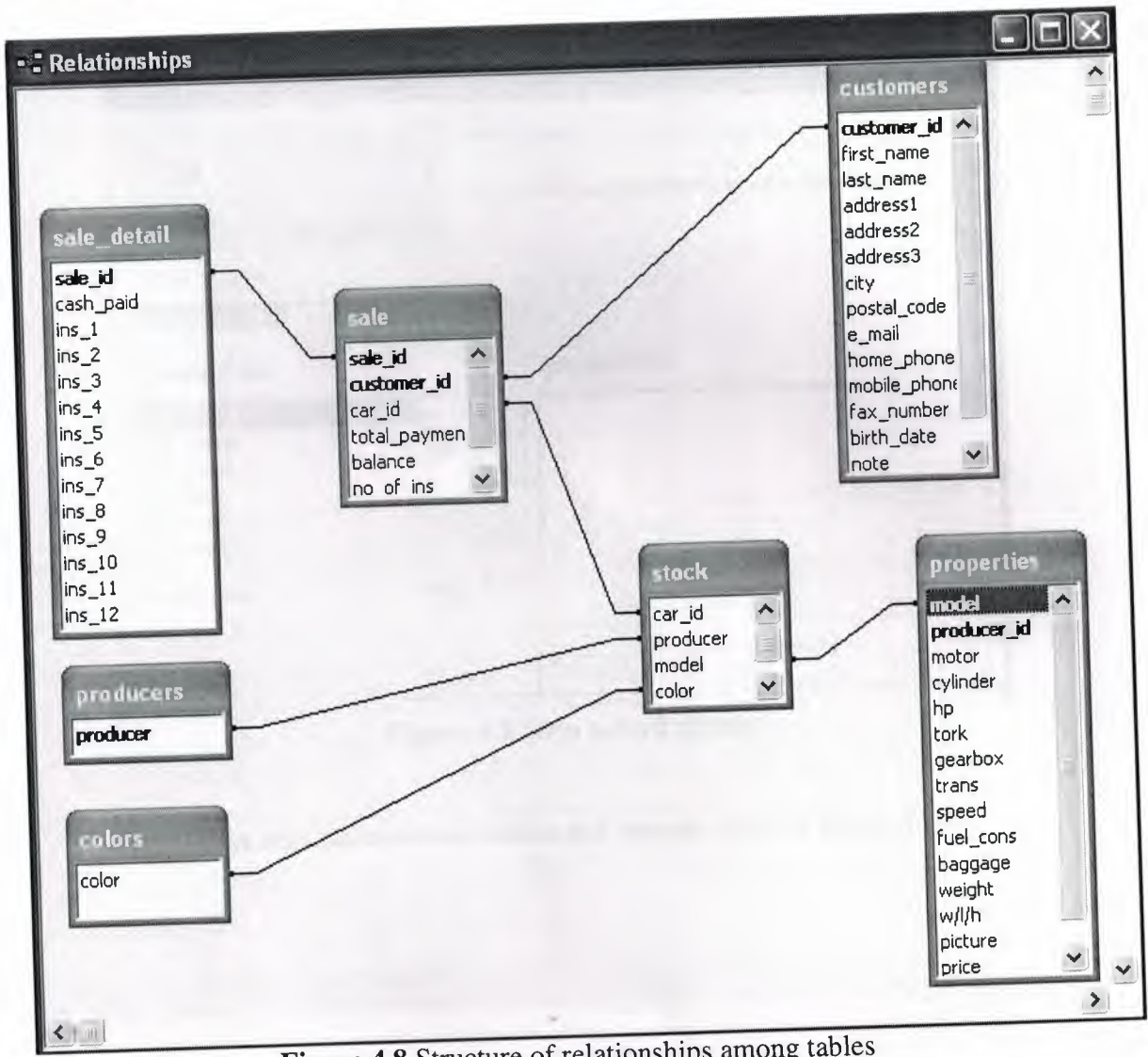


Figure 4.8 Structure of relationships among tables

4.3 Creating forms for our application

I selected form wizard to create customers form I selected all fields from available fields and click next button (in Figure 4.9)

Form Wizard

Which fields do you want on your form?
You can choose from more than one table or query.

Tables/Queries
Table: customers

Available Fields:

- customer_id
- first_name
- last_name
- address1
- address2
- address3
- city
- postal_code

Selected Fields:

Cancel < Back Next > Finish

Figure 4.9 form wizard screen

I have selected columnar button and pressed next (in Figure 4.10)

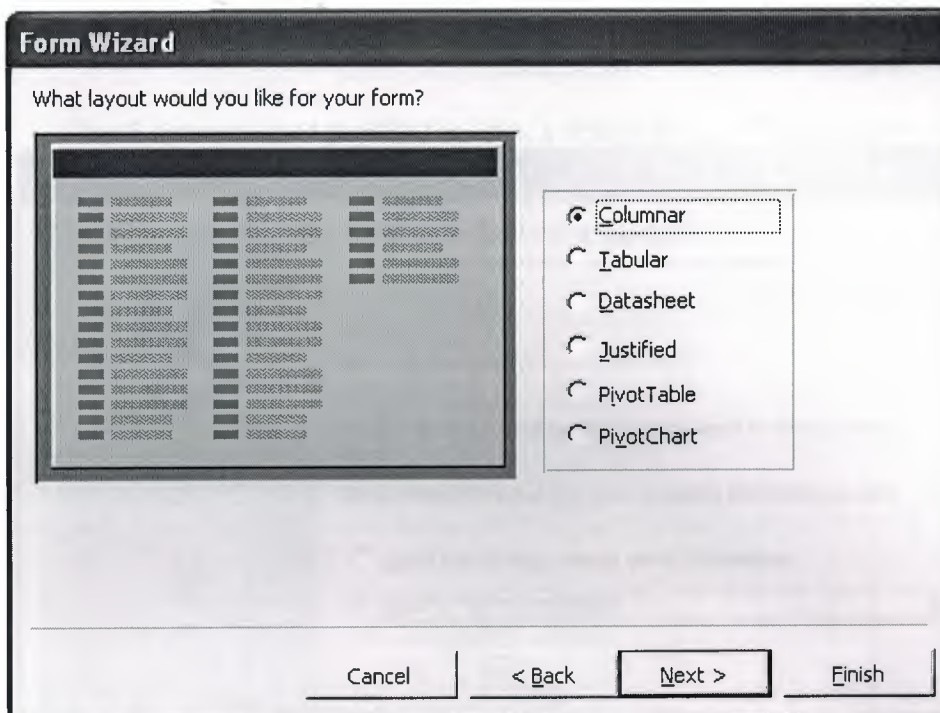


Figure 4.10 Next step of form wizard for layout of the form

Then I have select standard style (in figure 4.11)

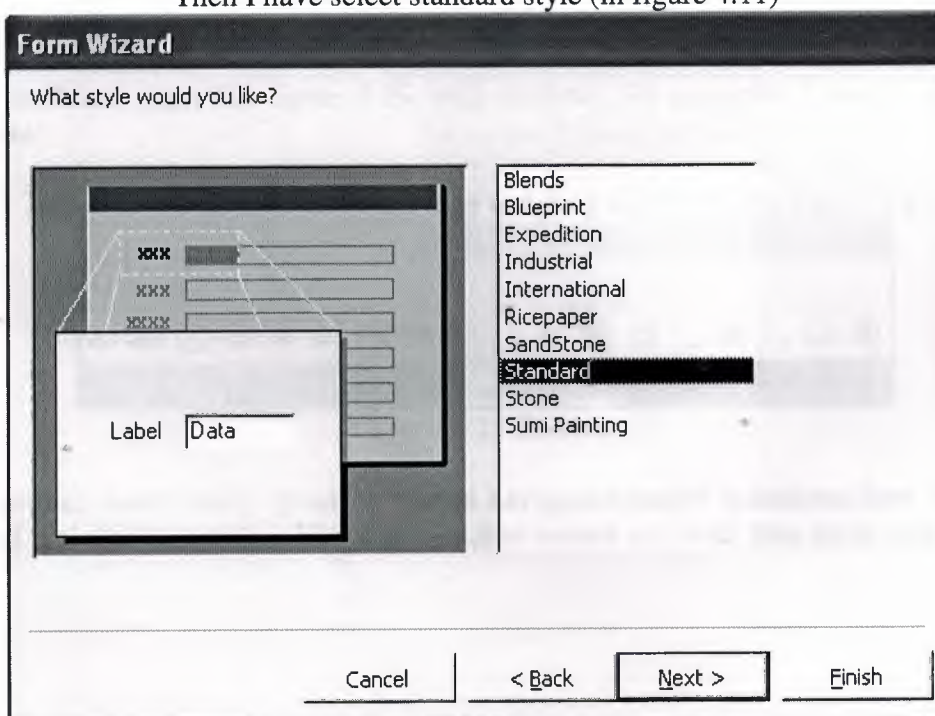


Figure 4.11 Next step of form wizard for style design

Then I have selected modify the form's design option (Figure-4.12)

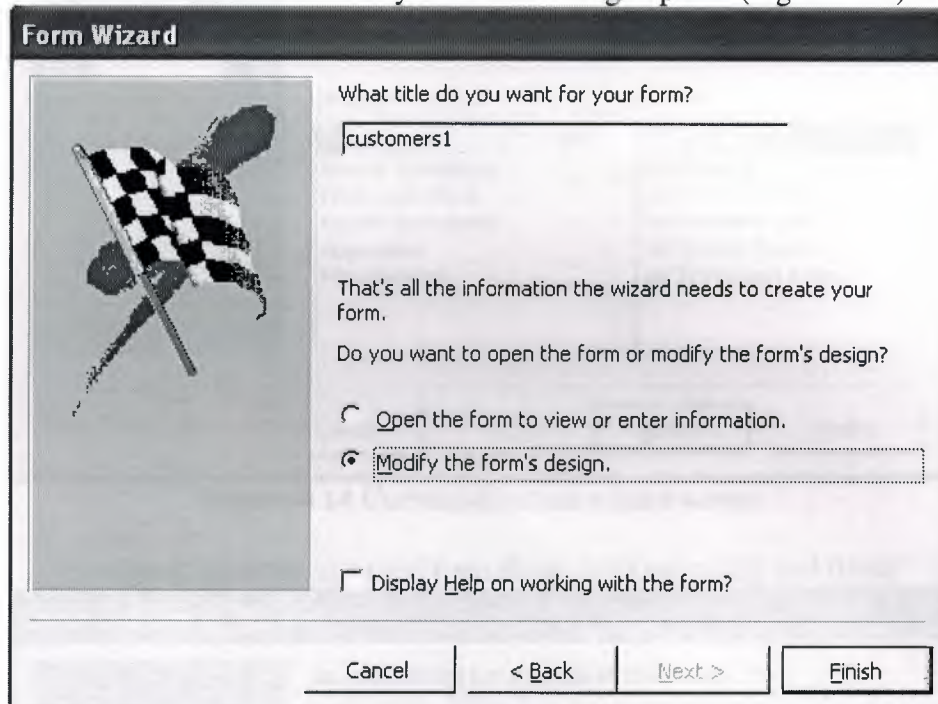


Figure 4.12 Modifying the form

Then I modified form like figure-4.16 with toolbox , for example I want to add button onto form

I selected command button and drop and drag onto form (in Figure-4.13)

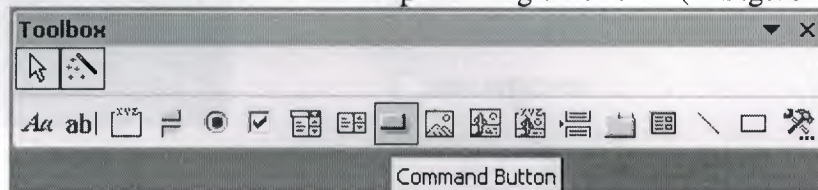


Figure-4.13 Toolbox

Then you can select many option as record navigation,record operations,form operations etc. and assigned to actions like find next,find record ect. And then click next (Figure-4.14)

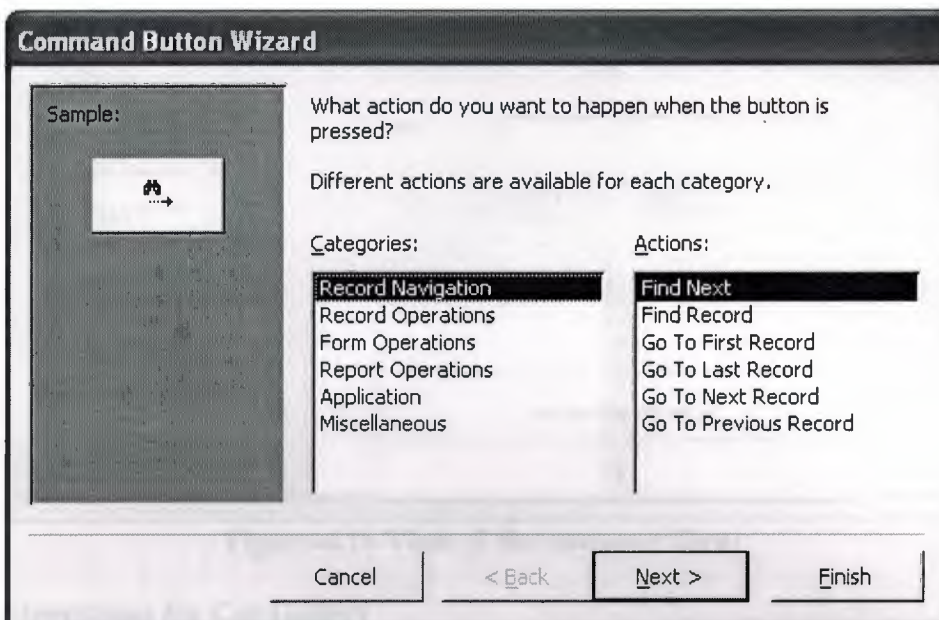


Figure-4.14 Command button wizard screen

I assigned a picture to button from figure 4.15 next click and finish

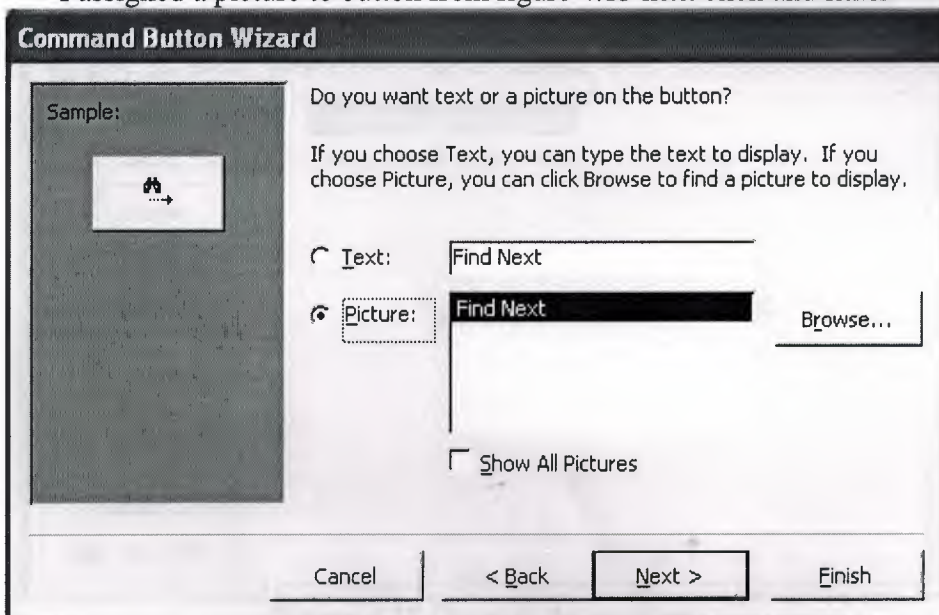


Figure-4.15 Next step of button wizard screen

I have designed costumers form and I had dropped many button as mentioned above

customer_id: 141

first_name: MURAT

last_name: BINGUL

address1: TURGUT

address2: TEMELLI CAD

address3: MALATYA

city: MALATYA

postal_code: 44200

birth_date:

note:

MURAT KEMAL	BINGUL TURGUT
-------------	---------------

Find Record as customer_id

Figure-4.16 View of the customer form

4.4 Sql Operations for Car Gallery

I created query with "create query in design view" options as you see figure-4.17

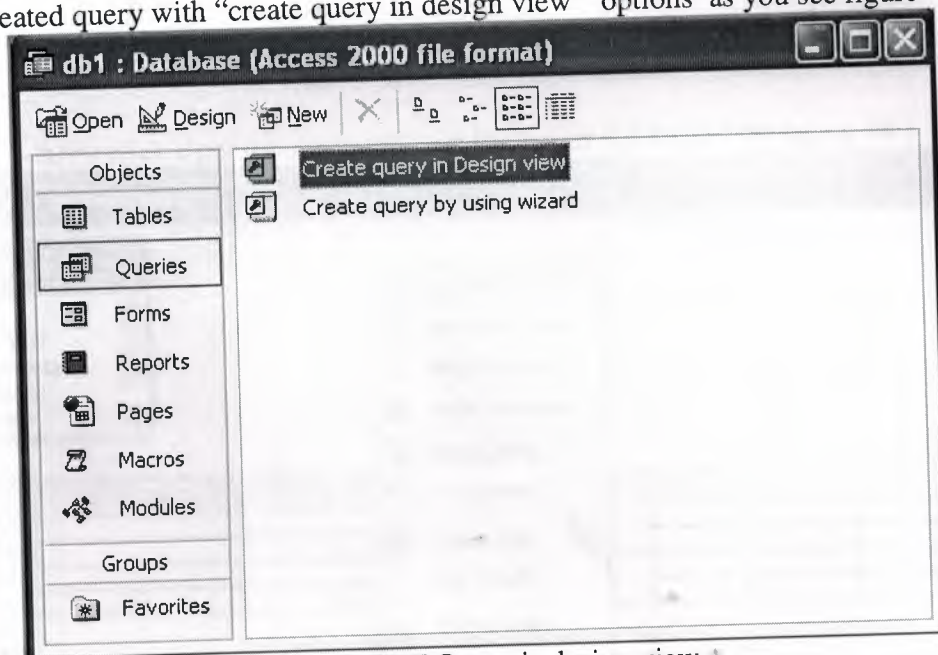


Figure 4.17 Query is design view

after click add button then selected stock table from figure-4.18 and click close

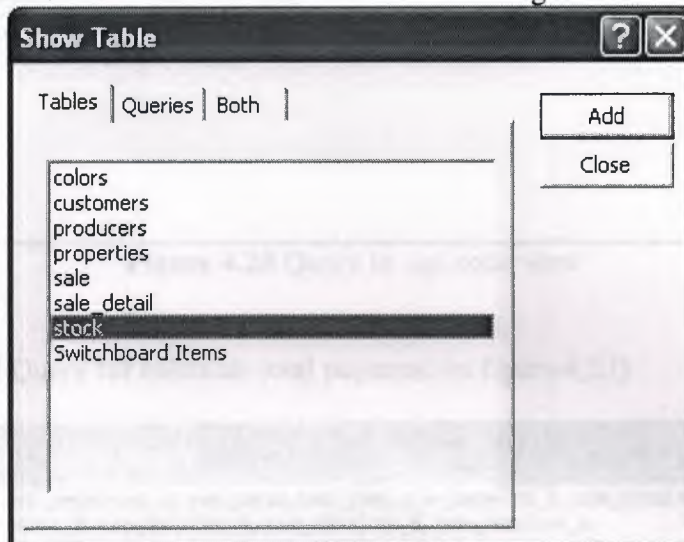


Figure-4.18 Show table screen

After click right mouse button and I selected Sql View (Figure-4.19)

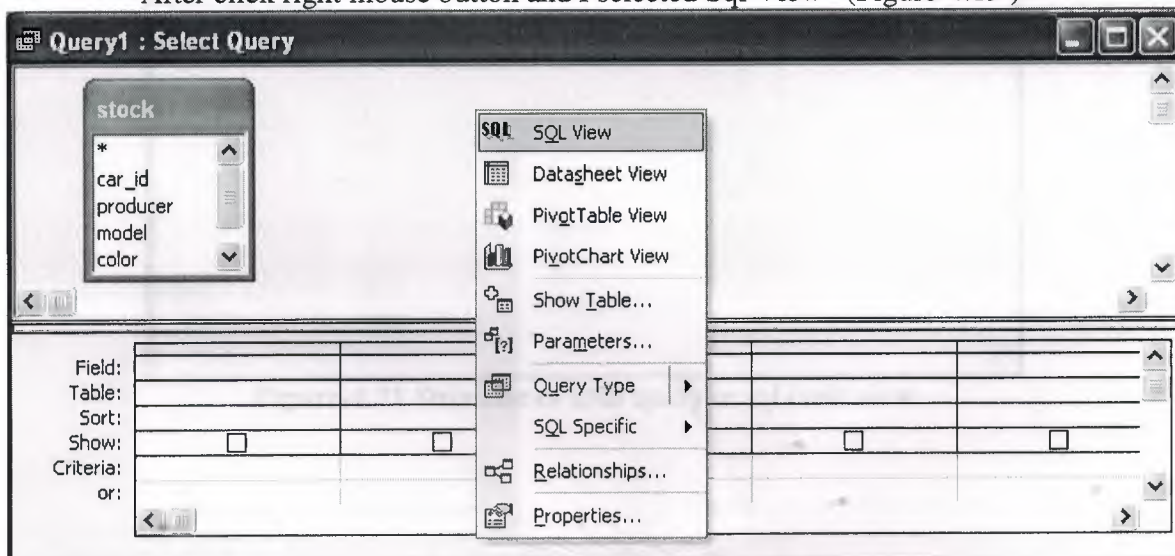
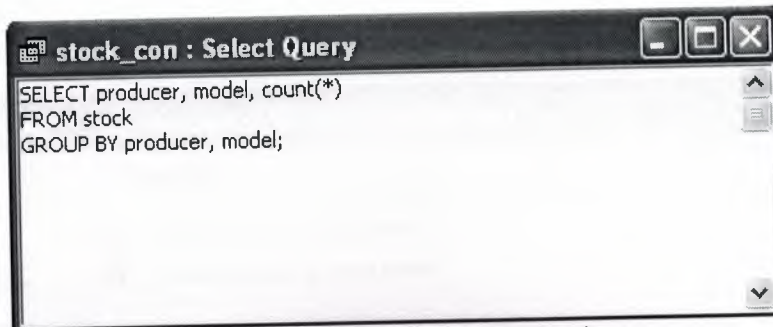


Figure-4.19 Query design view

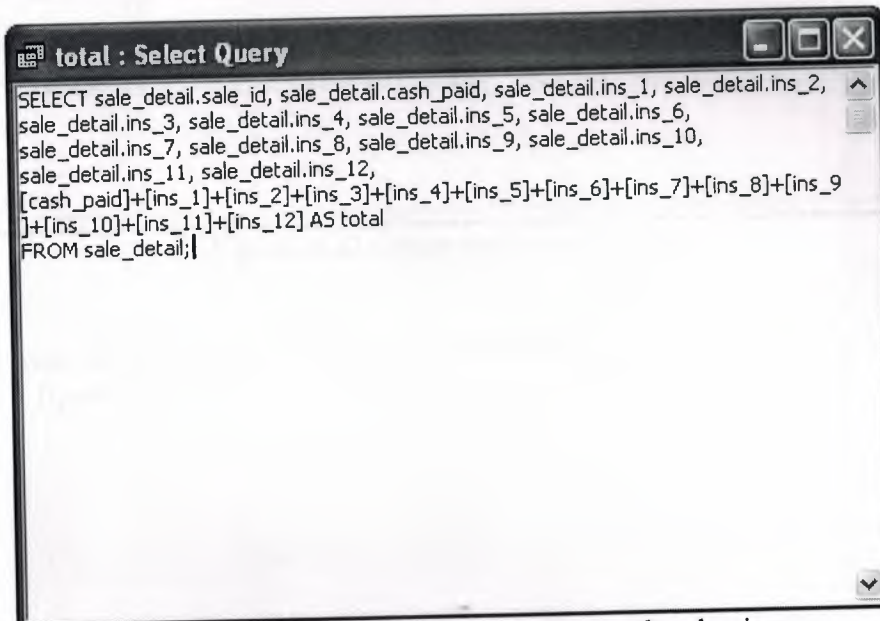
Then I wrote sql commands for counting stock of cars (in Figure 4.20)
After created query form as mentioned



```
stock_con : Select Query
SELECT producer, model, count(*)
FROM stock
GROUP BY producer, model;
```

Figure 4.20 Query in sql code view

Then I wrote total Query for calculate total payment (in figure-4.21)



```
total : Select Query
SELECT sale_detail.sale_id, sale_detail.cash_paid, sale_detail.ins_1, sale_detail.ins_2,
sale_detail.ins_3, sale_detail.ins_4, sale_detail.ins_5, sale_detail.ins_6,
sale_detail.ins_7, sale_detail.ins_8, sale_detail.ins_9, sale_detail.ins_10,
sale_detail.ins_11, sale_detail.ins_12,
[cash_paid]+[ins_1]+[ins_2]+[ins_3]+[ins_4]+[ins_5]+[ins_6]+[ins_7]+[ins_8]+[ins_9]
]+[ins_10]+[ins_11]+[ins_12] AS total
FROM sale_detail;
```

Figure-4.21 Structure of total query in sql code view

4.5 Creating report for Car Gallery

I used costumers table and prepared costumer reports with “create report by using wizard” in figure-4.22

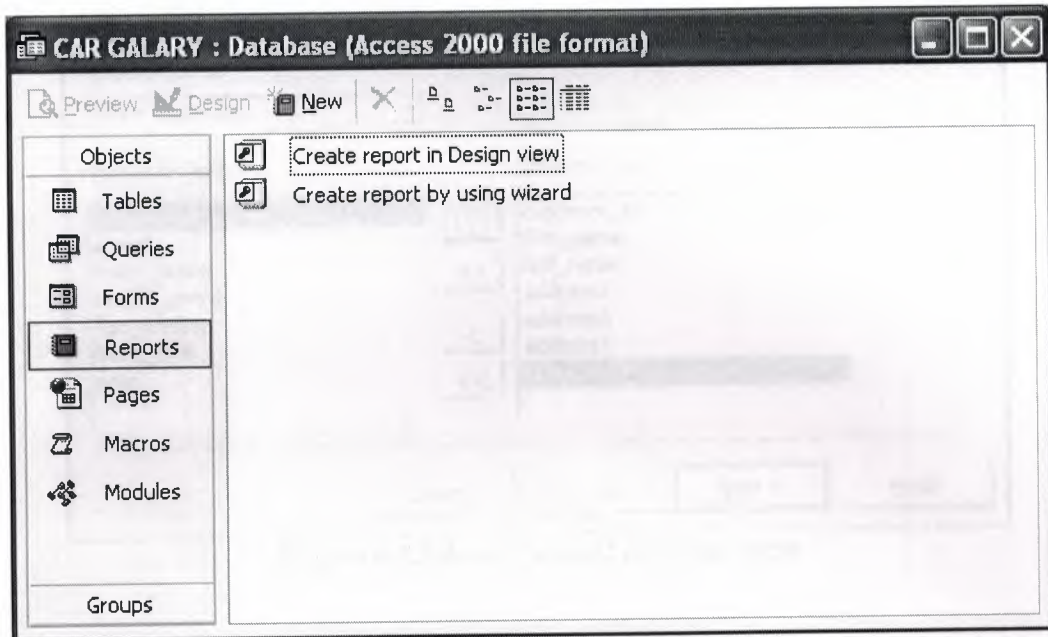


Figure-4.22 Create report design view

I selected some necessary fields from Available fields to cross box with ">" buttons and click next (figure-4.23)

Report Wizard

Which fields do you want on your report?

You can choose from more than one table or query.

Tables/Queries
Table: customers

Available Fields:

- postal_code
- e_mail
- home_phone
- mobile_phone
- fax_number
- birth_date
- note

Selected Fields:

- customer_id
- first_name
- last_name
- address1
- address2
- address3
- city

Cancel < Back Next > Finish

Figure 4.23 Report wizard next step view

After I arrange to fields as ascending and click next (figure-4.24)

Report Wizard

What sort order do you want for your records?

You can sort records by up to four fields, in either ascending or descending order.

1 customer_id Ascending

2 first_name Ascending

3 last_name Ascending

4 Ascending

Cancel < Back Next > Finish

Figure 4.24 Selecting fields to order customer table for report

I choosed corporate style from in figure-4.25 and click next

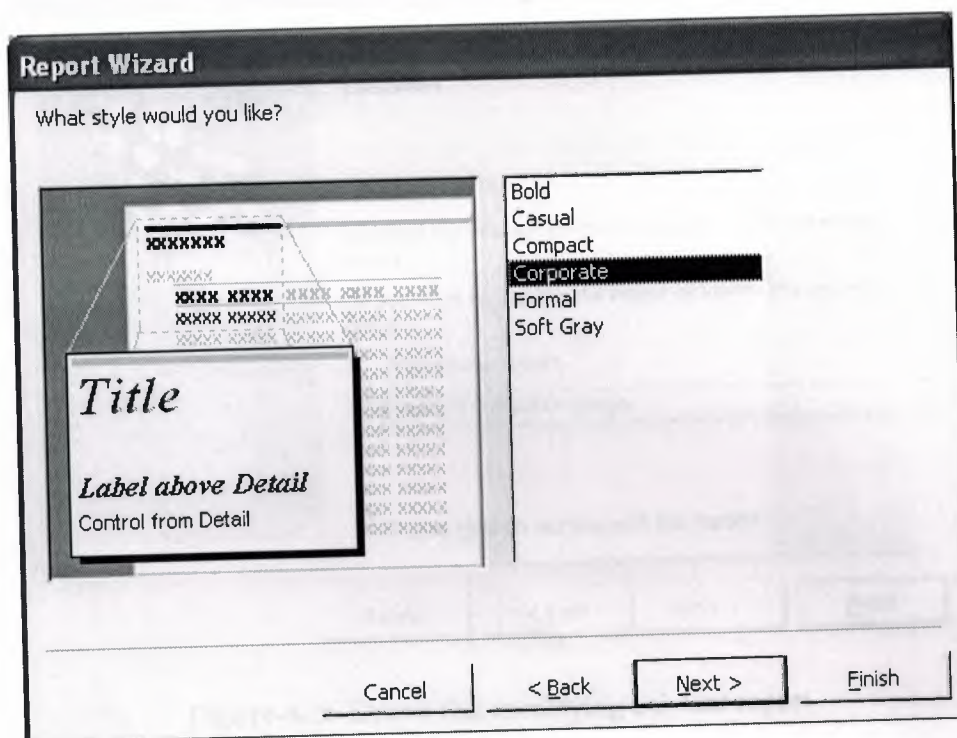


Figure-4.25 Screens shows that hot to give style to a report

I selected "modify the report's design" for make modify and click finish button (Figure-4.26)

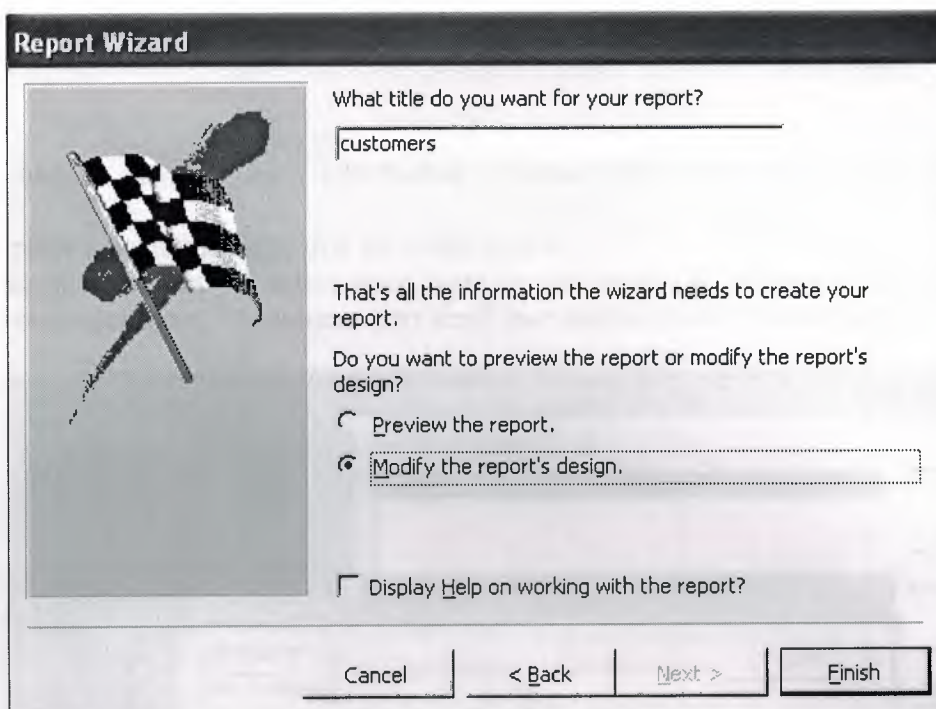


Figure-4.26 Shows that modifying existent report

4.6 Applying Switchboard Manager

The following steps shows that how to put password to our database
Database Utilities selected and switchboard manager function pressed

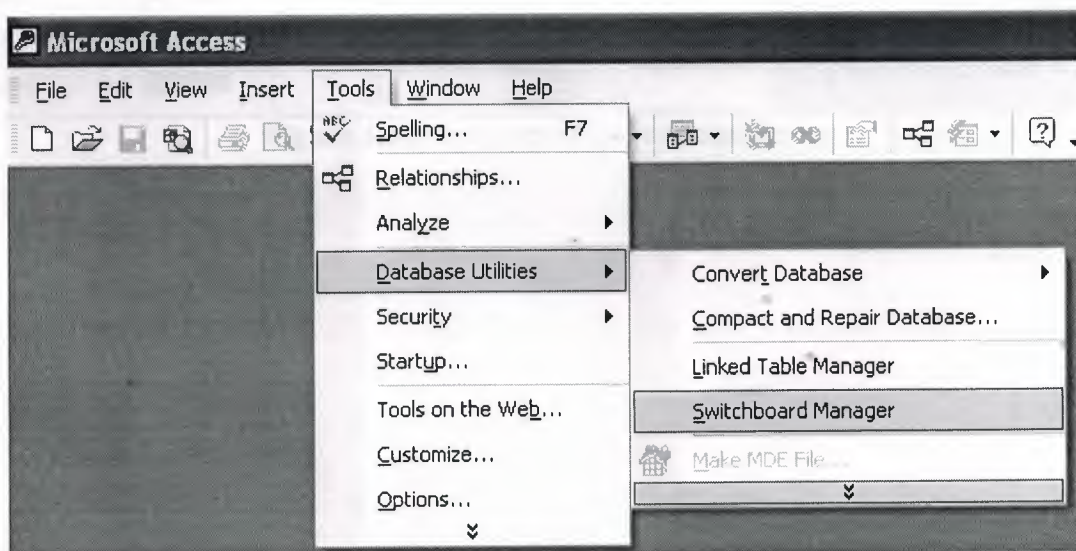


Figure 4.27 Switchboard design

on switchboard manager form edit button selected. After that edit switchboard page appears then we press new button, after that we come to new in command scroolbox we select open form in edit mode, in following scroll box we selected customer form. 'Costomers' text scroll box shown in switchboard table

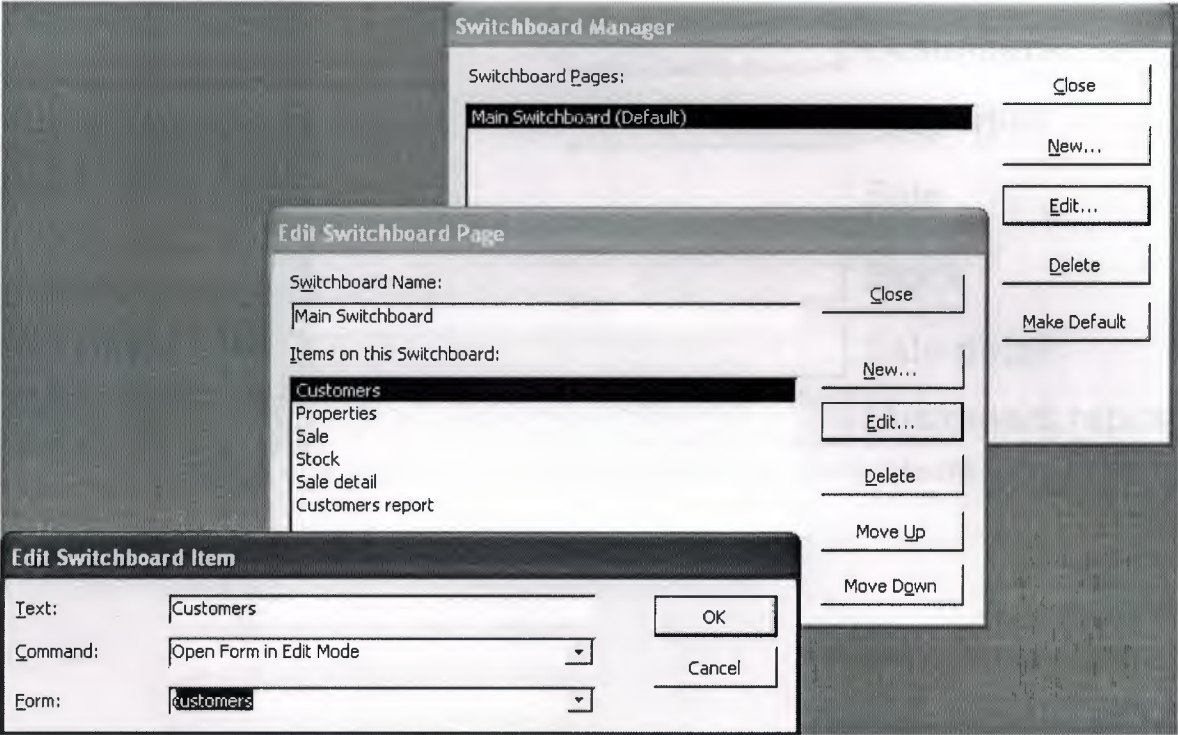


Figure 4.28 Next steps of switchboard design

Figure 4.29 all forms connected to this form

The image shows a software interface titled 'Main Switchboard' in the top-left corner. The main content area is titled 'CAR GALARY' and contains a list of six forms, each with a rectangular icon to its left. The forms are listed vertically: 'Customers', 'Properties', 'Sale', 'Stock', 'Sale detail', and 'Customers report'. The 'Customers' form icon is highlighted with a dotted border, while the others have solid borders. The background of the window is dark, and the text is white.

Form Name
Customers
Properties
Sale
Stock
Sale detail
Customers report

Figure 4.29 shows that hot to connect all forms to switchboard

4.7 Applying security password to database

Figure 4.30 We select the name of the our database to put password, it is necessary to open this database in open exclusive mode

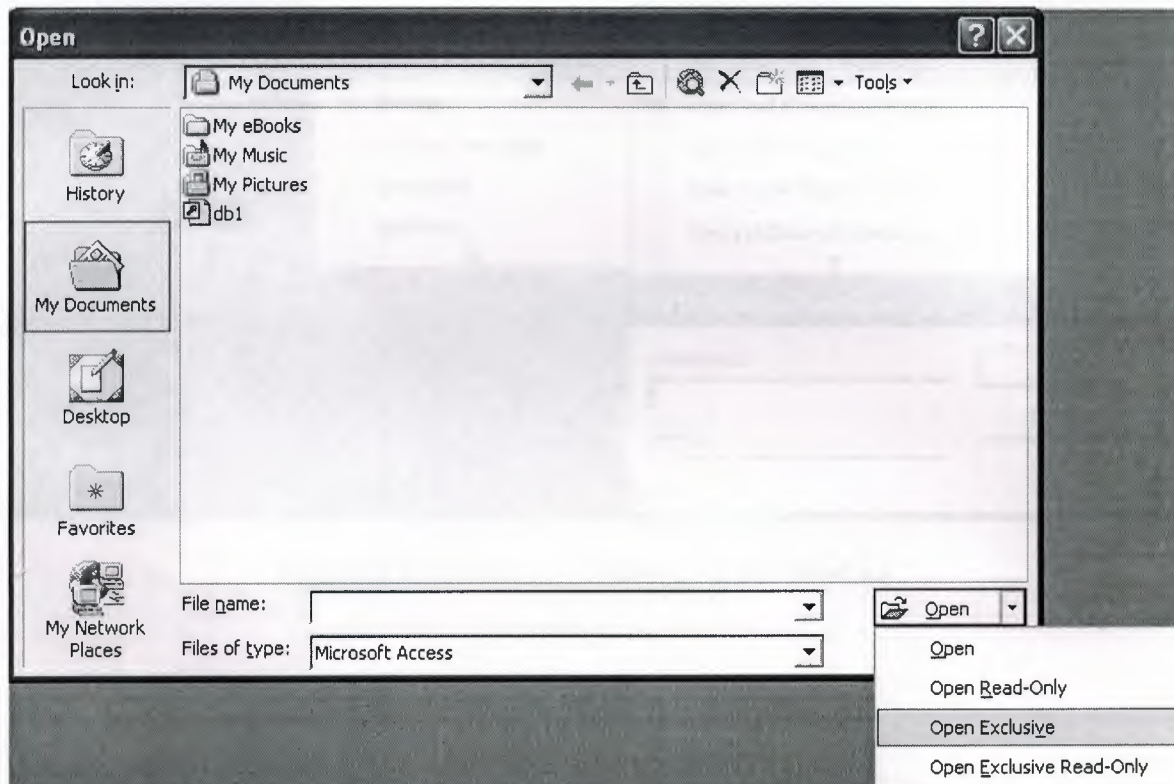


Figure 4.30 Open a file in exclusive mode

in figure 4.31 shows last step for creating password for our database security mode selects and set database password pressed then following screen appears we enter a password then verify it

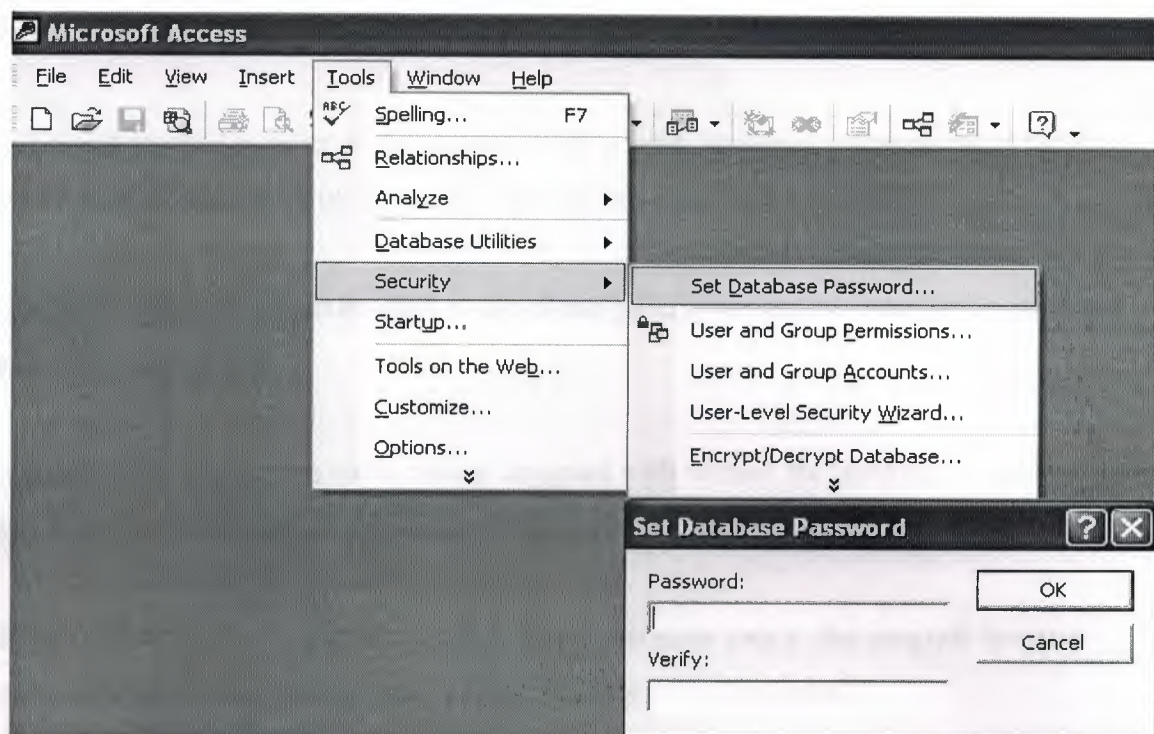


Figure 4.31 Screen for password for the database

CONCLUSION

Database is a powerful tool on a computer that organizes information. Its power lies in its ability to make information accessible in a variety of ways almost instantaneously.

This project stocked number of any model of car is held, customer information and sales record as well as properties of the cars with pictures.

This program is able to maintain small range of car gallery, but can be extended to maintain large range of car gallery.

Difficulties of this project are some menus designed with wizard. Because of its rapid manipulation. But if you forget some part you have to create the form again.

While you use this program you should care when you enter data to the program because the program does not check whether entered data is suitable for the fields.

In access programming it is difficult to work with buttons. When we press a button, in SQL during execution we come across some hardness.

After every time you enter data, you have to refresh the program. This is difficulty of access programming.

REFERENCES

Bob Villareal Access 2002 Programming By Example, Que cooperation press , july 2002.

Prof. Dr. Mithat Uysal (1999). Development Of The Access Database systems Istanbul.
a Press.

Ihsan Karagülle ; Zeydin Pala (1999). Microsoft Access 6.0 Pro. Istanbul. Türkmen Press.

Microsoft Access Programming Tutorials Site. Retrieved June 3,2002 From World Wide
b:<http://www.programmingtutorials.com>.

Lee, Thomas and Davies, Joseph, Microsoft Windows 2000 Access Programming
chnical Reference. Microsoft Press, 2000.

www.Google.com

www.Wikipedia.org

www.Microsoft.com