



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Purchasing and Selling Computer Parts
Visual Basic

Graduation Project
COM- 400

Student Name: Ahmad Moh`d Eid(20020934)

Supervisor: Assist.Prof Dr. Firudin Muradov

Nicosia - 2007

ACKNOWLEDGEMENT

First of all I would like to thanks ALLAH for guiding me through my study.

More over I feel proud to pay my special regards to my project adviser "Assist.Prof Dr. Firudin Muradov". He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project. He has Devine place in my heart and I am less than the half without his help. I am really thankful to my teacher.

Also, I want to pay special regards to my parents who are enduring these all expenses and supporting me in all events. I am nothing without their prayers. They also encouraged me in crises. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting. They may get peaceful life in Heaven. At the end I am again thankful to those all persons who helped me or even encouraged me to complete me, my project. My all efforts to complete this project might be fruitful.

To the best of my knowledge, I want to honor those all people who have supported me or helped me in my project. I also pay my special thanks to my all friends who have helped me in my project and gave me their precious time to complete my project.

ABSTRACT

This project is a package program to store the company daily transaction concerning purchasing and selling computer parts.

The project uses Microsoft Visual Basic 6.0, Microsoft Access XP for creating Data Base and some SQL Queries to manage database.

The aim of this project is to help the user to manage the data storage and use it when it is needed.

Customers pass by in computer shops to search for the best quality computer parts and prices, some customers pay in cash and some pay by using credit cards. So, the selling operation must be managed and controlled daily. On the other hand the company must hire employees with full information about them in which they can help the customers in finding the computer parts and writing full information about the product sells with the customer information.

This program helps user to manage the whole information in general and save it in database so they can use it later when it's needed.

Table of Contents

ACKNOWLEDGEMENT	i
ABSTRACT	ii
CONTENTS	iii
INTRODUCTION	iii
1. DATABASE	1
1.1 Overview	1
1.2 History	2
1.3 Database models	4
1.3.1 Flat model	4
1.3.2 Hierarchical model	5
1.3.3 Network model	5
1.3.4 Relational model	6
1.3.4.1 Relational operations	7
1.3.5 Dimensional model	8
1.3.6 Object database models	9
1.4 Database internals	10
1.4.1 Indexing	10
1.4.2 Transactions and concurrency	10
1.4.3 Replication	11
1.5 Applications of database	11
2. SQL	13
2.1 What is SQL	13
2.2 History	13
2.3 Standardization	14
2.4 Scope	15
2.5 Reasons For Lack Of Portability	16
2.6 SQL keywords	17
2.6.1 Data retrieval	17
2.6.2 Data manipulation	19

2.6.3 Data transaction	20
2.6.4 Data definition	20
2.6.5 Data control	21
2.7 Criticisms of SQL	23
2.8 Alternatives to SQL	25
3. MICROSOFT ACCESS	27
3.1 Over View	27
3.2 History	28
3.3 Uses	28
3.4 Features	29
3.5 Development	30
4. VISUAL BASIC	32
4.1 Over View	32
4.2 Derivative languages	32
4.3 Language features	34
4.4 Controversy	36
4.4.1 Weaknesses	36
4.4.1.1 Performance	36
4.4.1.2 Error Handling	37
4.4.1.3 Simplicity	37
4.4.2 Strengths	38
4.4.2.1 Debugging	38
4.4.2.2 Simplicity	38
4.4.3 Programming constructs not present in Visual Basic	39
4.4.4 Characteristics present in Visual Basic	40
4.5 Evolution of Visual Basic	41
4.6 Timeline of Visual Basic (VB1 to VB6)	42
5. DESCRIPTION OF THE SOFTWARE	44
5.1 Introduction	44
5.2 Description of the forms	48
CONCLUSION	59

REFERENCES

60

APPENDIX

61

INTRODUCTION

This project describes database system concepts and a simple computer shop program which named purchasing and selling computer parts. The application part uses Access and SQL quires. The program is written by Microsoft Visual Basic 6.0, and Microsoft Access XP to create database. Also it uses some SQL codes.

The project contains five chapters.

CHAPTER 1: describes information about the database in general, database models, relational database operations and brands.

CHAPTER 2: describes basics of SQL. Its history, keywords and some of the commands of it.

CHAPTER 3: describes Microsoft Access and its features.

CHAPTER 4: presents information about Visual Basic Language Features and the basic concepts of Visual Basic.

CHAPTER 5: describes the program execution and basics about purchasing and selling computer parts,

CHAPTER ONE

DATABASE

1.1 Overview

The term database originated within the computer industry. Although its meaning has been broadened by popular use, even to include non-electronic databases, this article takes a more technical perspective. A possible definition is that a database is a collection of records stored in a computer in a systematic way, so that a computer program can consult it to answer questions. The items retrieved in answer to queries become information that can be used to make decisions. The computer program used to manage and query a database is known as a database management system (DBMS). The properties and design of database systems are included in the study of information science.

The central concept of a database is that of a collection of records, or pieces of knowledge. Typically, for a given database, there is a structural description of the type of facts held in that database: this description is known as a schema. The schema describes the objects that are represented in the database, and the relationships among them. There are a number of different ways of organizing a schema, that is, of modeling the database structure: these are known as database models (or data models). The model in most common use today is the relational model, which in layman's terms represents all information in the form of multiple related tables each consisting of rows and columns (the true definition uses mathematical terminology). This model represents relationships by the use of values common to more than one table. Other models such as the hierarchical model and the network model use a more explicit representation of relationships.

Strictly speaking, the term database refers to the collection of related records, and the software should be referred to as the database management system or DBMS. When the context is unambiguous, however, many database administrators and programmers use the term database to cover both meanings.

Many professionals would consider a collection of data to constitute a database only if it has certain properties: for example, if the data is managed to ensure its integrity and quality, if it allows shared access by a community of users, if it has a schema, or if it supports a query language. However, there is no agreed definition of these properties.

Database management systems are usually categorized according to the data model that they support: relational, object-relational, network, and so on. The data model will tend to determine the query languages that are available to access the database. A great deal of the internal engineering of a DBMS, however, is independent of the data model, and is concerned with managing factors such as performance, concurrency, integrity, and recovery from hardware failures. In these areas there are large differences between products.

1.2 History

The earliest known use of the term 'data base' was in June 1963, when the System Development Corporation sponsored a symposium under the title Development and Management of a Computer-centered Data Base. Database as a single word became common in Europe in the early 1970s and by the end of the decade it was being used in major American newspapers. (Databank, a comparable term, had been used in the Washington Post newspaper as early as 1966.)

The first database management systems were developed in the 1960s. A pioneer in the field was Charles Bachman. Bachman's early papers show that his aim was to make more effective use of the new direct access storage devices becoming available: until then, data processing had been based on punched cards and magnetic tape, so that serial processing was the dominant activity. Two key data models arose at this time: CODASYL developed the network model based on Bachman's ideas, and (apparently independently) the hierarchical model was used in a system developed by North American Rockwell, later adopted by IBM as the cornerstone of their IMS product.

The relational model was proposed by E. F. Codd in 1970. He criticized existing models for confusing the abstract description of information structure with descriptions of physical access mechanisms. For a long while, however, the relational model remained of academic interest only. While CODASYL systems and IMS were conceived as practical engineering solutions taking account of the technology as it existed at the time, the relational model took a much more theoretical perspective, arguing (correctly) that hardware and software technology would catch up in time. Among the first implementations were Michael Stonebraker's Ingres at Berkeley, and the System R project at IBM. Both of these were research prototypes, announced during 1976. The first commercial products, Oracle and DB2, did not appear until around 1980. The first successful database product for microcomputers was dBASE for the CP/M and PC-DOS/MS-DOS operating systems.

During the 1980s, research activity focused on distributed database systems and database machines, but these developments had little effect on the market. Another important theoretical idea was the Functional Data Model, but apart from some specialized applications in genetics, molecular biology, and fraud investigation, the world took little notice.

In the 1990s, attention shifted to object-oriented databases. These had some success in fields where it was necessary to handle more complex data than relational systems could easily cope with, such as spatial databases, engineering data (including software engineering repositories), and multimedia data. Some of these ideas were adopted by the relational vendors, who integrated new features into their products as a result.

In the 2000s, the fashionable area for innovation is the XML database. As with object databases, this has spawned a new collection of startup companies, but at the same time the key ideas are being integrated into the established relational products. XML databases aim to remove the traditional divide between documents and data, allowing all of an organization's information resources to be held in one place, whether they are highly structured or not

1.3 Database models

Various techniques are used to model data structure. Most database systems are built around one particular data model, although it is increasingly common for products to offer support for more than one model. For any one logical model various physical implementations may be possible, and most products will offer the user some level of control in tuning the physical implementation, since the choices that are made have a significant effect on performance. An example of this is the relational model: all serious implementations of the relational model allow the creation of indexes which provide fast access to rows in a table if the values of certain columns are known.

A data model is not just a way of structuring data: it also defines a set of operations that can be performed on the data. The relational models, for example, define operations such as select, project, and join. Although these operations may not be explicit in a particular query language, they provide the foundation on which a query language is built.

1.3.1 Flat model

This may not strictly qualify as a data model, as defined above. 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 that might be used as a part of a system security database. Each row would have the specific password associated with an individual 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, incidentally, a basis of the spreadsheet.

1.3.2 Hierarchical model

In a hierarchical model, data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list. Hierarchical structures were widely used in the early mainframe database management systems, such as the Information Management System (IMS) by IBM, and now describe the structure of XML documents. This structure allows one 1: N relationship between two types of data. This structure is very efficient to describe many relationships in the real world; recipes, table of contents, ordering of paragraphs/verses, any nested and sorted information. However, the hierarchical structure is inefficient for certain database operations when a full path (as opposed to upward link and sort field) is not also included for each record.

1.3.3 Network model

The network model (defined by the CODASYL specification) organizes data using two fundamental constructs, called records and sets. Records contain fields (which may be organized hierarchically, as in the programming language COBOL). Sets (not to be confused with mathematical sets) define one-to-many relationships between records: one owner, many members. A record may be an owner in any number of sets, and a member in any number of sets.

The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.

Although it is not an essential feature of the model, network databases generally implement the set relationships by means of pointers that directly address the location of a record on disk. This gives excellent retrieval performance, at the expense of operations such as database loading and reorganization.

1.3.4 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.

The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model defined by Codd. The data structures in these products are tables, rather than relations: the main differences being that tables can contain duplicate rows, and that the rows (and columns) can be treated as being ordered. The same criticism applies to the SQL language which is the primary interface to these products. There has been considerable controversy, mainly due to Codd himself, as to whether it is correct to describe SQL implementations as "relational": but the fact is that the world does so, and the following description uses the term in its popular sense.

A relational database contains multiple tables, each similar to the one in the "flat" database model. Relationships between tables are not defined explicitly; instead, keys are used to match up rows of data in different tables. A key is a collection of one or more columns in one table whose values match corresponding columns in other tables: for example, an Employee table may contain a column named Location which contains a value that matches the key of a Location table. Any column can be a key, or multiple columns can be grouped together into a single key. It is not necessary to define all the keys in advance; a column can be used as a key even if it was not 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 a row; this is defined as the table's primary key.

A key that has an external, real-world meaning (such as a person's name, a book's ISBN, or a car's serial number) is sometimes called a "natural" key. If no natural key is suitable (think of the many people named Brown), 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 cannot 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.)

1.3.4.1 Relational operations

Users (or programs) request data from a relational database by sending it a query that is written in a special language, usually a dialect of SQL. Although SQL was originally intended for end-users, it is much more common for SQL queries to be embedded into software that provides an easier user interface. Many web sites, such as Wikipedia, perform SQL queries when generating pages.

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 are combined into one, by doing a join. Conceptually, this is done by taking all possible combinations of rows (the Cartesian 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.

There are a number of relational operations in addition to join. These include project (the process of eliminating some of the columns), restrict (the process of eliminating some of the rows), union (a way of combining two tables with similar structures), difference (which lists the rows in one table that are not found in the other), intersect (which lists the rows found in both tables), and product (mentioned above, which combines each row of one table with each row of the other). Depending on which other sources you consult, there are a number of other operators - many of which can be defined in terms of those listed above. These include semi-join, outer operators such as outer join and outer union, and various forms of division. Then there are operators to rename columns, and summarizing or aggregating operators, and if you permit relation values as attributes (RVA - relation-valued attribute), then

operators such as group and ungroup. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.

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.

1.3.5 Dimensional model

The dimensional model is a specialized adaptation of the relational model used to represent data in data warehouses in a way that data can be easily summarized using OLAP queries. In the dimensional model, a database consists of a single large table of facts that are described using dimensions and measures. A dimension provides the context of a fact (such as who participated, when and where it happened, and its type) and is used in queries to group related facts together. Dimensions tend to be discrete and are often hierarchical; for example, the location might include the building, state, and country. A measure is a quantity describing the fact, such as revenue. It's important that measures can be meaningfully aggregated - for example, the revenue from different locations can be added together.

In an OLAP query, dimensions are chosen and the facts are grouped and added together to create a summary.

The dimensional model is often implemented on top of the relational model using a star schema, consisting of one table containing the facts and surrounding tables containing the dimensions. Particularly complicated dimensions might be represented using multiple tables, resulting in a snowflake schema.

A data warehouse can contain multiple star schemas that share dimension tables, allowing them to be used together. Coming up with a standard set of dimensions is an important part of dimensional modeling.

1.3.6 Object database models

In recent years, the object-oriented paradigm has been applied to database technology, creating a new programming model known as object databases. These databases attempt to bring the database world and the application programming world closer together, in particular by ensuring that the database uses the same type system as the application program. This aims to avoid the overhead (sometimes referred to as the impedance mismatch) of converting information between its representation in the database (for example as rows in tables) and its representation in the application program (typically as objects). At the same time object databases attempt to introduce the key ideas of object programming, such as encapsulation and polymorphism, into the world of databases.

A variety of these ways have been tried for storing objects in a database. Some products have approached the problem from the application programming end, by making the objects manipulated by the program persistent. This also typically requires the addition of some kind of query language, since conventional programming languages do not have the ability to find objects based on their information content. Others have attacked the problem from the database end, by defining an object-oriented data model for the database, and defining a database programming language that allows full programming capabilities as well as traditional query facilities.

Object databases suffered because of a lack of standardization: although standards were defined by ODMG, they were never implemented well enough to ensure interoperability between products. Nevertheless, object databases have been used successfully in many applications: usually specialized applications such as engineering databases or molecular biology databases rather than mainstream commercial data processing. However, object database ideas were picked up by the relational vendors and influenced extensions made to these products and indeed to the SQL language.

1.4 Database internals

1.4.1 Indexing

All of these kinds of database can take advantage of indexing to increase their speed, and this technology has advanced tremendously since its early uses in the 1960s and 1970s. 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 indexes can be created or dropped without changing existing applications making use of it. The database chooses between many different strategies based on which one it estimates will run the fastest.

Relational DBMSs utilize 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 analyzing 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

1.4.2 Transactions and concurrency

In addition to their data model, most practical databases ("transactional databases") attempt to enforce a database transaction model that has desirable data integrity properties. Ideally, the database software should enforce the ACID rules, summarized here:

Atomicity: Either all the tasks in a transaction must be done, or none of them. The transaction must be completed, or else it must be undone (rolled back).

Consistency: Every transaction must preserve the integrity constraints — the declared consistency rules — of the database. It cannot place the data in a contradictory state.

Isolation: Two simultaneous transactions cannot interfere with one another. Intermediate results within a transaction are not visible to other transactions.

Durability: Completed transactions cannot be aborted later or their results discarded. They must persist through (for instance) restarts of the DBMS after crashes

In practice, many DBMS's allow most of these rules to be selectively relaxed for better performance.

Concurrency control is a method used to ensure that transactions are executed in a safe manner and follow the ACID rules. The DBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions.

1.4.3 Replication

Replication of databases is closely related to transactions. If a database can log its individual actions, it is possible to create a duplicate of the data in real time. The duplicate can be used to improve performance or availability of the whole database system. Common replication concepts include:

Master/Slave Replication: All write requests are performed on the master and then replicated to the slaves

Quorum: The result of Read and Write requests is calculated by querying a "majority" of replicas.

Multimaster: Two or more replicas sync each other via a transaction identifier.

1.5 Applications of database

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. Software database drivers are

available for most database platforms so that application software can use a common application programming interface (API) to retrieve the information stored in a database. Two commonly used database APIs are JDBC and ODBC. A database is also a place where you can store data and then arrange that data easily and efficiently.

CHAPTER TWO

SQL

2.1 What Is SQL

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database. This tutorial will provide you with the instruction on the basics of each of these commands as well as allow you to put them to practice using the SQL Interpreter.

2.2 History

An influential 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, although drafts of it were circulated internally within IBM in 1969. Codd's model became widely accepted as the definitive model for relational database management systems (RDBMS or RDMS).

During the 1970s, a group at IBM's San Jose research center developed a database system "System R" based upon, but not strictly faithful to, 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 because the word 'SEQUEL' was held as a trademark by the Hawker-Siddeley aircraft company of the UK. Although SQL was influenced by Codd's work, Donald D. Chamberlin and

Raymond F. Boyce at IBM were the authors of the SEQUEL language design. Their concepts were published to increase interest in SQL.

The first non-commercial, relational, non-SQL database, Ingres, was developed in 1974 at U.C. Berkeley.

In 1978, methodical testing commenced at customer test sites. Demonstrating both the usefulness and practicality of the system, this testing proved to be a success for IBM. As a result, IBM began to develop commercial products based on their System R prototype that implemented SQL, including the System/38 (announced in 1978 and commercially available in August 1979), SQL/DS (introduced in 1981), and DB2 (in 1983).

At the same time Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Chamberlin and Boyce and developed their own version of a RDBMS for the Navy, CIA and others. In the summer of 1979 Relational Software, Inc. introduced Oracle V2 (Version2) for VAX computers as the first commercially available implementation of SQL. Oracle is often incorrectly cited as beating IBM to market by two years, when in fact they only beat IBM's release of the System/38 by a few weeks. Considerable public interest then developed; soon many other vendors developed versions, and Oracle's future was ensured.

2.3 Standardization

SQL was adopted as a standard by 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 /ɛs kju ɛl/, although many English-speaking database professionals still pronounce it as sequel.

The SQL standard has gone through a number of revisions:

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89		Minor revision.
1992	SQL-92	SQL2	Major revision (ISO 9075).
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.)
2003	SQL:2003		Introduced XML-related features, window functions, standardized sequences and columns with auto-generated values (including identity-columns).

The SQL standard is not freely available. SQL: 2003 may be purchased from ISO or ANSI. A late draft is available as a zip archive from Whitemarsh Information Systems Corporation. The zip archive contains a number of PDF files that define the parts of the SQL: 2003 specification.

2.4 Scope

SQL is defined by both ANSI and ISO. There are many extensions to and variations on the version of the language. Many of these extensions are of a proprietary nature, such as Oracle Corporation's PL/SQL, IBM's SQL PL (SQL Procedural Language) and Sybase / Microsoft's Transact-SQL. Commercial implementations commonly omit support for basic features of the standard, such as the `DATE` or `TIME` data types, preferring some variant of their own. SQL code can rarely be ported between database systems without major modifications, in contrast to ANSI C or ANSI Fortran, which can usually be ported from platform to platform without major structural changes.

SQL is designed for a specific, limited purpose — querying data contained in a relational database. As such, it is a set-based, declarative computer language rather than an imperative language such as C or BASIC which, being general-purpose, are designed to solve a much broader set of problems. Language extensions such as PL/SQL bridge this gap to some extent by adding procedural elements, such as flow-of-control constructs. Another approach is to allow programming language code to be embedded in and interact with the database. For example, Oracle and others include Java in the database, while PostgreSQL allows functions to be written in a wide variety of languages, including Perl, Tcl, and C.

SQL contrasts with the more powerful database-oriented fourth-generation programming languages such as Focus or SAS in its relative functional simplicity and simpler command set. This greatly reduces the degree of difficulty involved in maintaining 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.

2.5 Reasons For Lack Of Portability

There are several reasons for this lack of portability between database systems:

The complexity and size of the SQL standard means that most databases do not implement the entire standard.

The standard does not specify database behavior in several important areas (e.g. indexes), leaving it up to implementations of the standard to decide how to behave.

The SQL standard precisely specifies the syntax that a conforming database system must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to areas of ambiguity.

Many database vendors have large existing customer bases; where the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.

Some believe the lack of compatibility between database systems is intentional in order to ensure vendor lock-in.

2.6 SQL keywords

SQL keywords fall into several groups.

2.6.1 Data retrieval

The most frequently used operation in transactional databases is the data retrieval operation. When restricted to data retrieval commands, SQL acts as a declarative language.

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 Data Manipulation Language 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 efficient 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 from which tables the data is to be taken, 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. WHERE is evaluated before the 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 of the "combined rows" (combined rows are produced when the query has a GROUP BY keyword or when the SELECT part

contains aggregates), are to be retrieved. HAVING acts much like a WHERE, but it operates on the results of the GROUP BY and hence can use aggregate functions.

ORDER BY is used to identify which columns are used to sort the resulting data.

Data retrieval is very often combined with data projection; usually it isn't the verbatim data stored in primitive data types that a user is looking for or a query is written to serve. Often the data needs to be expressed differently from how it's stored. SQL allows a wide variety of formulas included in the select list to project data. A common example would be:

```
SELECT UnitCost * Quantity As TotalCost FROM Orders
```

Example 1:

```
SELECT * FROM books
```

```
WHERE price > 100.00 and price < 150.00
```

```
ORDER BY title
```

This is an example that could be used to get a list of expensive books. It retrieves the records from the books table that have a price field which is greater than 100.00 and less than 150.00. The result is sorted alphabetically by book title. The asterisk (*) means to show all columns of the books table. Alternatively, specific columns could be named.

Example 2:

```
SELECT books.title, count(*) AS Authors
```

```
FROM books
```

```
JOIN book_authors
```

```
ON books.book_number = book_authors.book_number
```

```
GROUP BY books.title
```

Example 2 shows both the use of multiple tables in a join, and aggregation (grouping). This example shows how many authors there are per book. Example output may resemble:

Title	Authors
SQL Examples and Guide	3
The Joy of SQL	1
How to use Wikipedia	2
Pitfalls of SQL	1
How SQL Saved my Dog	1

2.6.2 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.

`MERGE` is used to combine the data of multiple tables. It is something of a combination of the `INSERT` and `UPDATE` elements. It is defined in the SQL:2003 standard; prior to that, some databases provided similar functionality via different syntax, sometimes called an "upsert".

`TRUNCATE` deletes all data from a table (non-standard, but common SQL command).

`DELETE` removes zero or more existing rows from a table.

Example:

```
INSERT INTO my_table (field1, field2, field3) VALUES ('test', 'N', NULL);
```

```
UPDATE my_table SET field1 = 'updated value' WHERE field2 = 'N';
```

```
DELETE FROM my_table WHERE field2 = 'N';
```

2.6.3 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.

Example:

```
BEGIN WORK;
```

```
UPDATE inventory SET quantity = quantity - 3 WHERE item = 'pants';
```

```
COMMIT;
```

2.6.4 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.

Example:

```
CREATE TABLE my_table (  
  
my_field1 INT,  
  
my_field2 VARCHAR (50),  
  
my_field3 DATE NOT NULL,  
  
PRIMARY KEY (my_field1, my_field2) )
```

All DDL statements are auto commit so while dropping a table need to have close look at its future needs.

2.6.5 Data control

The third group of SQL keywords is the Data Control Language (DCL). DCL handles the authorization aspects of data and permits the user to control who has access to see or manipulate data within the database.

Its two main keywords are:

GRANT — authorizes one or more users to perform an operation or a set of operations on an object.

REVOKE — removes or restricts the capability of a user to perform an operation or a set of operations.

Example:

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

Other

ANSI-standard SQL supports as a single line comment identifier (some extensions also support curly brackets or C-style `/* comments */` for multi-line comments).

Example:

```
SELECT * FROM inventory -- Retrieve everything from inventory table
```

Some SQL servers allow User Defined Functions

Database systems using SQL

Comparison of relational database management systems

Comparison of truly relational database management systems

Comparison of object-relational database management systems

Comparison of SQL syntax

List of relational database management systems

List of object-relational database management systems

List of hierarchical database management systems

2.7 Criticisms of SQL

Technically, SQL is a declarative computer language for use with "SQL databases". Theorists and some practitioners note that many of the original SQL features were inspired by, but in violation of, the relational model for database management and its tuple calculus realization. Recent extensions to SQL achieved relational completeness, but have worsened the violations, as documented in The Third Manifesto.

In addition, there are also some criticisms about the practical use of SQL:

Implementations are inconsistent and, usually, incompatible between vendors. In particular date and time syntax, string concatenation, nulls, and comparison case sensitivity often vary from vendor-to-vendor.

The language makes it too easy to do a Cartesian join, which results in "run-away" result sets when `WHERE` clauses are mistyped. Cartesian joins are so rarely used in practice that requiring an explicit `CARTESIAN` keyword may be warranted.

A similar and more common problem is that of Exploding joins; this is something between what the user desired and a full-blown Cartesian join. What happens is that part of the relationship or criteria has not been defined, and the database engine returns all possible combinations of records that satisfy the ill-defined query criteria. Most often this happens in systems which use compound keys that are not respected in the offending query; for example maybe one out of three keys will be matched, resulting in too much information, but still less than a Cartesian join would produce.

SQL's set theory techniques and operations usually cannot also apply to column lists. Thus, column lists cannot be computed dynamically.

The difference between value-to-column assignment in `UPDATE` and `INSERT` can result in confusion and added work for automated SQL code generation modules.

It does not provide a standard way, or at least a commonly-supported way, to split large commands into multiple smaller ones that reference each other by name. This tends to result in "run-on SQL sentences" and may force one into a deep hierarchical

nesting when a graph-like (reference-by-name) approach may be more appropriate and better repetition-factoring. (Views, and stored procedures can help with this, but often require special database privileges and are not really meant for single-query usage.) Here is an illustration for a "duplication finder" query:

Sample Table "codeTable"

	locat	code	descript
--	-------	------	----------

10	AA	Foo Bar
----	----	---------

20	AA	Foo Baar
----	----	----------

30	AA	Foo Bar
----	----	---------

10	BB	Glab Zab
----	----	----------

20	BB	Glab Zab
----	----	----------

```
select *
```

```
from codeTable
```

```
where locat not in (30, 50)
```

```
and code not in
```

```
(
```

```
select code
```

```
from
```

```
(select code, descript --(gets unique code-and-descript combos)
```

```
from codeTable
```

```
where locat not in (30, 50)
```

group by code, descript)

group by code

having count(*) > 1)

order by code, locat

Here we have a table of codes in which we want to find and study typos in the descriptions that are supposed to repeat for each location. (Perhaps the repetition is bad normalization, but sometimes one has to deal with such data from clunky old systems.) For example, the second row in the sample data has the typo "Baar".

In this case we want to ignore codes from location 30 and 50 because we know they are not being used right now and thus we don't care to inspect them. To do it properly, we have to apply the filter in two different places. I see this kind of thing in a good many queries. There may be ways around such, but they are not obvious and not general-purpose solutions to such.

2.8 Alternatives to SQL

A distinction should be made between alternatives to relational query languages and alternatives to SQL. The lists below are proposed alternatives to SQL, but are still (nominally) relational. See navigational database for alternatives to relational.

IBM Business System 12 (IBM BS12)

Tutorial D

TQL - Luca Cardelli (May not be relational)

Top's Query Language - A draft language influenced by IBM BS12. Tentatively renamed to SMEQL to avoid confusion with similar projects called TQL.

Hibernate Query Language [2] (HQL) - A Java-based tool that uses modified SQL

Quel introduced in 1974 by the U.C. Berkeley Ingres project.

Object Query Language - Object Data Management Group.

Datalog

CHAPTER THREE

MICROSOFT ACCESS

3.1 Overview

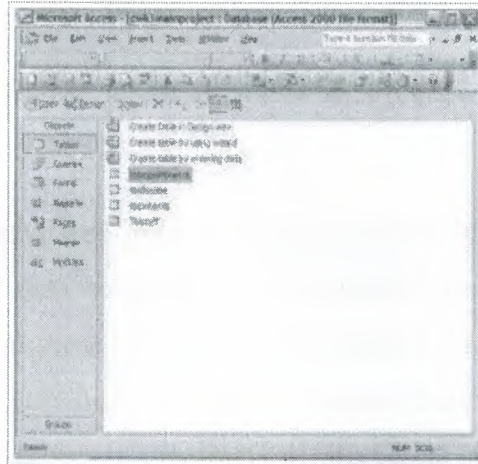


Figure 3.1: MICROSOFT ACCESS

relational Microsoft Access (current full name Microsoft Office Access) is a Microsoft Office , packaged with Microsoft from database management system with a Microsoft Jet Database Engine which combines the relational Professional .graphical user interface

, Microsoft SQL Server stored in Access/Jet, dataMicrosoft Access can use data and software developers-compliant data container. Skilled ODBC, or any Oracle programmers. Relatively unskilled application software use it to develop architects and non-programmer "power users" can use it to build simple applications. It supports (OO) techniques but falls short of being a fully OO development object-oriented some tool.

Microsoft Access was also the name of a communications program from and other programs. This Microsoft ProCommMicrosoft, meant to compete with Access proved a failure and was dropped. Years later Microsoft reused the name for its database software.

3.2 History

Microsoft specified the minimum operating system for Version 1.1 as Microsoft Windows v3.0 with 4 MB of RAM. 6 MB RAM was recommended along with a minimum of 8 MB of available hard disk space (14 MB hard disk space recommended).

The product was shipped on seven 1.44 MB diskettes. The manual shows a 1993 copyright date.

The software worked well with very large records sets but testing showed some circumstances caused data corruption. For example, file sizes over 700 MB were problematic. (Note that most hard disks were smaller than 700 MB at the time this was in wide use). The Getting Started manual warns about a number of circumstances where obsolete device drivers or incorrect configurations can cause data loss.

MS-Access's initial codename was Cirrus. This was developed before Visual Basic and the forms engine was called Ruby. Bill Gates saw the prototypes and decided that the Basic language component should be co-developed as a separate expandable application. This project was called Thunder. The two projects were developed separately as the underlying forms engines were incompatible with each other; however, these were merged together again after VBA.

3.3 Uses

Access is used by small businesses, within departments of large corporations, and hobby programmers to create ad hoc customized desktop systems for handling the creation and manipulation of data. Access can also be used as the database for basic web based applications hosted on Microsoft's Internet Information Services and utilizing Microsoft Active Server Pages ASP. More complex web applications may require tools like PHP/MySQL or ASP/Microsoft SQL Server.

Some professional application developers use Access for rapid application development, especially for the creation of prototypes and standalone applications that serve as tools for on-the-road salesmen. Access does not scale well if data access

is via a network, so applications that are used by more than a handful of people tend to rely on a Client-Server based solution such as Oracle, DB2, Microsoft SQL Server, Windows Share Point Services, PostgreSQL, MySQL, Alpha Five, MaxDB, or FileMaker. However, an Access "front end" (the forms, reports, queries and VB code) can be used against a host of database back ends, including JET (file-based database engine, used in Access by default), Microsoft SQL Server, Oracle, and any other ODBC-compliant product.

Many developers who use Microsoft Access use the Leszynski naming convention, though this is not universal; it is a programming convention, not a DBMS-enforced rule.

3.4 Features

One of the benefits of Access from a programmer's perspective is its relative compatibility with SQL—queries may be viewed and edited as SQL statements, and SQL statements can be used directly in Macros and VBA Modules to manipulate Access tables.

Users may mix and use both VBA and "Macros" for programming forms and logic and offers object-oriented possibilities.

MSDE (Microsoft SQL Server Desktop Engine) 2000, a mini-version of MS SQL Server 2000, is included with the developer edition of Office XP and may be used with Access as an alternative to the Jet Database Engine.

Unlike a complete RDBMS, the Jet Engine lacks database triggers and stored procedures. Starting in MS Access 2000 (Jet 4.0), there is a syntax that allows creating queries with parameters, in a way that looks like creating stored procedures, but these procedures are limited to one statement per procedure. Microsoft Access does allow forms to contain code that is triggered as changes are made to the underlying table (as long as the modifications are done only with that form), and it is common to use pass-through queries and other techniques in Access to run stored procedures in RDBMSs that support these.

In ADP files (supported in MS Access 2000 and later), the database-related features are entirely different, because this type of file connects to a MSDE or Microsoft SQL Server, instead of using the Jet Engine. Thus, it supports the creation of nearly all objects in the underlying server (tables with constraints and triggers, views, stored procedures and UDF-s). However, only forms, reports, macros and modules are stored in the ADP file (the other objects are stored in the back-end database).

3.5 Development

Access allows relatively quick development because all database tables, queries, forms, and reports are stored in the database. For query development, Access utilizes the Query Design Grid, a graphical user interface that allows users to create queries without knowledge of the SQL programming language. In the Query Design Grid, users can "show" the source tables of the query and select the fields they want returned by clicking and dragging them into the grid. Joins can be created by clicking and dragging fields in tables to fields in other tables. Access allows users to view and manipulate the SQL code if desired.

The programming language available in Access is, as in other products of the Microsoft Office suite, Microsoft Visual Basic for Applications. Two database access libraries of COM components are provided: the legacy Data Access Objects (DAO), only available with Access, and the new ActiveX Data Objects (ADO).

Microsoft Access can be applied to small projects but scales poorly to larger projects involving multiple concurrent users because it is a desktop application, not a true client-server database. When a Microsoft Access database is shared by multiple concurrent users, processing speed suffers. The effect is dramatic when there are more than a few users or if the processing demands of any of the users are high. Access includes an Upsizing Wizard that allows users to upsize their database to Microsoft SQL Server if they want to move to a true client-server database.

Since all database queries, forms, and reports are stored in the database, and in keeping with the ideals of the relational model, there is no possibility of making a physically structured hierarchy with them.

One design technique is to divide an Access application between data and programs. One database should contain only tables and relationships, while another would have all programs, forms, reports and queries, and links to the first database tables. Unfortunately, Access allows no relative paths when linking, so the development environment should have the same path as the production environment (though it is possible to write a "dynamic-linker" routine in VBA that can search out a certain back-end file by searching through the directory tree, if it can't find it in the current path). This technique also allows the developer to divide the application among different files, so some structure is possible.

CHAPTER FOUR

VISUAL BASIC

4.1 Over View

Visual Basic (VB) is an event driven programming language and associated development environment from Microsoft. VB has been replaced by Visual Basic .NET. The older version of VB was derived heavily from BASIC and enables the rapid application development (RAD) of graphical user interface (GUI) applications, access to databases using DAO, RDO, or ADO, and creation of ActiveX controls and objects.

A programmer can put together an application using the components provided with Visual Basic itself. Programs written in Visual Basic can also use the Windows API, but doing so requires external function declarations.

In business programming, Visual Basic has one of the largest user bases. According to some sources, as of 2003, 52% of software developers used Visual Basic, making it the most popular programming language at that time. Another point of view was provided by the research done by Evans Data that found that 43% of Visual Basic developers planned to move to other languages. Visual Basic currently competes with PHP and C++ as the third most popular programming language behind Java and C.

4.2 Derivative languages

Microsoft has developed derivatives of Visual Basic for use in scripting. It is derived heavily from BASIC and host applications, and has replaced the original Visual Basic language with a .NET platform version:

- Visual Basic for Applications (VBA) is included in many Microsoft applications (Microsoft Office), and also in several third-party products like WordPerfect Office 2002 and ESRI ArcGIS. There are small inconsistencies in the

way VBA is implemented in different applications, but it is largely the same language as VB6.

- Here is an example of the language:

To find the area of a circle

```
Private Sub Command1_Click ()
```

```
pi = 3.14159265358979323846264
```

```
r = Val(Radius.Text)
```

```
a = pi * r *r
```

```
area.Text = Str$(a)
```

```
End Sub
```

Another Example to write the words "Hello world" on the form:

```
Private Sub Form1_Load()
```

```
Print "Hello World"
```

```
End Sub
```

- VBScript is the default language for Active Server Pages and can be used in Windows scripting and client-side web page scripting. Although it resembles VB in syntax, it is a separate language and it is executed by the Windows Script Host as opposed to the VB runtime. These differences can affect the performance of an ASP web site (namely inefficient string concatenation and absence of short-cut evaluation). ASP and VBScript must not be confused with ASP.NET which uses Visual Basic.Net or any other language that targets the .NET Common Language Runtime.
- Visual Basic .NET is Microsoft's designated successor to Visual Basic 6.0, and is part of Microsoft's .NET platform. The VB.NET programming language is a true object-oriented language that compiles and runs on the .NET Framework.

VB.NET is a totally new tool from the ground up, not backwards compatible with VB6. For this reason, it was suggested by Bill Vaughn, and wholeheartedly embraced by the user community, that it ought to have been given an alternative name. Visual Fred (or VFred for short) was the consensus choice. VB.NET ships with a rudimentary utility to convert legacy VB6 code, although the inefficient nature of the resulting code (due to major differences between the two languages) often leads programmers to prefer manual conversion instead. Indeed, automated conversion is seen as a fantasy.

Many users have found that automated conversion of anything more than trivial VB6 programs is essentially impossible, with many TODO's marking incompatible sections. A rewrite does take care of this, but a complete rewrite of a complex program is often not practical for several reasons. First, a small company considering a rewrite must usually choose between spending its budget on new features and maintenance, or on conversion of a static program, which in itself adds no value. Second, a rewrite in a new language means an extensive testing cycle, again an expense with no corresponding market value. As a result, the migration path has not as often been from VB6 to VB.NET, but rather to other languages and platforms such as Java, C# and Delphi.

4.3 Language features

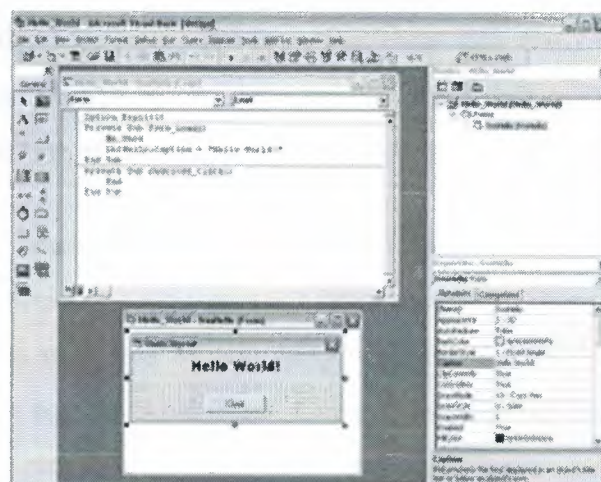


Figure 4.1 A typical session in Microsoft Visual Basic 6

Visual Basic was designed to be easy to learn and use. The language not only allows programmers to easily create simple GUI applications, but also has the flexibility to develop fairly complex applications as well. Programming in VB is a combination of visually arranging components or controls on a form, specifying attributes and actions of those components, and writing additional lines of code for more functionality. Since default attributes and actions are defined for the components, a simple program can be created without the programmer having to write many lines of code. Performance problems were experienced by earlier versions, but with faster computers and native code compilation this has become less of an issue.

Although programs can be compiled into native code executables from version 5 onwards, they still require the presence of runtime libraries of approximately 2 MB in size. This runtime is included by default in Windows 2000 and later, but for earlier versions of Windows it must be distributed together with the executable.

Forms are created using drag and drop techniques. A tool is used to place controls (e.g., text boxes, buttons, etc.) on the form (window). Controls have attributes and event handlers associated with them. Default values are provided when the control is created, but may be changed by the programmer. Many attribute values can be modified during run time based on user actions or changes in the environment, providing a dynamic application. For example, code can be inserted into the form resize event handler to reposition a control so that it remains centered on the form, expands to fill up the form, etc. By inserting code into the event handler for a keypress in a text box, the program can automatically translate the case of the text being entered, or even prevent certain characters from being inserted.

Visual Basic can create executables (EXE), ActiveX controls, DLL files, but is primarily used to develop Windows applications and to interface web database systems. Dialog boxes with less functionality (e.g., no maximize/minimize control) can be used to provide pop-up capabilities. Controls provide the basic functionality of the application, while programmers can insert additional logic within the appropriate event handlers. For example, a drop-down combination box will automatically display its list and allow the user to select any element. An event handler is called when an item is selected, which can then execute additional code created by the programmer to

perform some action based on which element was selected, such as populating a related list.

Alternatively, a Visual Basic component can have no user interface, and instead provide ActiveX objects to other programs via Component Object Model (COM). This allows for server-side processing or an add-in module.

The language is garbage collected using reference counting, has a large library of utility objects, and has basic object oriented support. Since the more common components are included in the default project template, the programmer seldom needs to specify additional libraries. Unlike many other programming languages, Visual Basic is generally not case sensitive, although it will transform keywords into a standard case configuration and force the case of variable names to conform to the case of the entry within the symbol table entry. String comparisons are case sensitive by default, but can be made case insensitive if so desired.

4.4 Controversy

Visual Basic is seen as a controversial language; many programmers have strong feelings regarding the quality of Visual Basic and its ability to compete with newer languages. It was designed to be a simple language. In the interest of convenience and rapid development, some features like explicit variable declaration are turned off by default, something that can be easily changed. This leads to some programmers praising Visual Basic for how simple it is to use, but can also lead to frustration when programmers encounter problems that the features would have detected. For instance, in Visual Basic a common mistake is to incorrectly type the name of a variable, creating a new variable with a slightly different name.

4.4.1 Weaknesses

4.4.1.1 Performance

Early versions of Visual Basic were not competitive at performing computationally intensive tasks because they were interpreted, and not compiled to machine code. Although this roadblock was removed with VB5 (which compiles to

the same intermediate language and uses the same back end as Visual C++, some features of the language design still introduce overhead which can be avoided in languages like Delphi or C++. These are more likely to be encountered in code involving objects, methods, and properties than in strictly numerical code.

4.4.1.2 Error Handling

Visual Basic does not have exception handling with the same capabilities of C++ or Java, but the On Error facility does provide nonlocal error handling with features similar to Windows Structured Exception Handling, including the ability to resume after an error (a feature that is not provided by either of the other two languages, although of dubious utility in production code).

4.4.1.3 Simplicity

Many critics of Visual Basic explain that the simple nature of Visual Basic is harmful in the long run. Many people have learned VB on their own without learning good programming practices. Even when VB is learned in a formal classroom, the student may not be introduced to many fundamental programming techniques and constructs, since much of the functionality is contained within the individual components and not visible to the programmer. Since it is possible to learn how to use VB without learning standard programming practices, this often leads to unintelligible code and workarounds. Second, having many of the checks and warnings that a compiler implements turned off by default may lead to difficulties in finding bugs. Experienced programmers working in VB tend to turn such checks on.

Many of the criticisms fired at Visual Basic are in fact criticisms of its ancestor, BASIC. A famous formulation by Edsger Dijkstra states, "It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration (Dijkstra was no less scathing about FORTRAN, PL/I, COBOL and APL).

4.4.2 Strengths

4.4.2.1 Debugging

Visual Basic has a comprehensive set of debugging tools comparable to those available in the Visual C++ products of the same time period. Features include breakpoints, the ability to watch variables and modify watched variables while paused, the ability to modify the point of execution, and the ability to make modifications to code while paused, often not requiring a program restart. Arbitrary code could be executed in the "immediate window", an online interpreter, a very powerful feature. In some cases, these features were more capable than their counterparts in Visual C++—for instance, edit and continue in VC was inspired by the VB feature, and there has never been a VC equivalent of the immediate window.

Furthermore, since VB5 it has been possible to generate debug symbols for a native executable and step into VB code in external debuggers, like the Microsoft debugger or the VC debugger, although the implementation of VB objects makes it difficult to debug code that uses them heavily.

4.4.2.2 Simplicity

While some detractors argue that the simplicity of Visual Basic is a weakness, many proponents of Visual Basic explain that the simple nature of Visual Basic is its main strength, allowing very rapid application development to experienced Visual Basic coders and a very slight learning curve for programmers coming from other languages. Additionally, Visual Basic applications can easily be integrated with databases, a common requirement. For example, by using controls that are bound to a database, it is possible to write a VB application that maintains information within the database without writing any lines of VB code.

Visual Basic is also a conglomerate of language features and syntax, with less consistency, but more tolerance, than many modern programming languages. Many language features like GoSub, On Error, and declaring the type of a variable by the last character in the name (i.e. str\$) are legacies from Visual Basic's BASIC roots, and are included for backward-compatibility. The syntax of VB is different from most

other languages, which can lead to confusion for new VB programmers. For example, the statement `"Dim a, b, c As Integer"` declares `"c"` as integer, but `"a"` and `"b"` are declared as Variant. Another source of confusion for new programmers is the different use of parentheses for arguments of functions and subroutines. Other characteristics include the entry of keyword, variable and subroutine names that are not case sensitive, and an underscore `"_"` must be used for a statement to span multiple lines.

Some Visual Basic programmers perceive these as strengths needed to avoid case-sensitive compiler errors, and accidentally omitting line-termination characters some languages require (usually semicolons). For example, the ability to enter variable and subroutine names in any case, coupled with the IDE's automatic correction to the case used in the declaration, can be used to the programmer's advantage: by declaring all names in mixed case, but entering them in lower case elsewhere, allows the programmer to type faster and to detect typos when a token remains in lower case.

The language continues to attract much praise and criticism, and it continues to cater to a large base of users and developers. The language is well suited for certain kinds of GUI applications (e.g., front end to a database), but less suited for others (e.g., compute-bound programs). Its simplicity and ease of use explain its popularity as a tool for solving business problems — most business stakeholders do not care about technical elegance and effectiveness, and concentrate instead on the cost effectiveness of Visual Basic.

4.4.3 Programming constructs not present in Visual Basic.

Many of these features are implemented in Microsoft's replacement for Visual Basic 6 and prior, VB.NET.

- **Inheritance.** Visual Basic versions 5 and 6 are not quite object oriented languages as they do not include implementation inheritance. VB5 and 6 do, however, include specification of interfaces. That is, a single class can have as many distinct interfaces as the programmer desires. Visual Basic provides a specific syntax for access to attributes called Property methods, and this is often implemented using

getters and setters in C++ or Java. Python has an equivalent notation to VB6's property Let and Get.

- Threading support (can be done by using external Windows functions).
- C++ or Java exception handling. Error handling is controlled by an "On Error" statement, which provides similar functionality to Windows Structured Exception Handling.
- Typecasting. VB instead has conversion functions.
- Equivalents to C-style pointers are very limited.
- Visual Basic is limited to unsigned 8-bit integers and signed integers of 16 and 32 bits. Many other languages provide wider range of signed and unsigned integers.
- 32-bit Visual Basic is internally limited to UTF-16 strings, although it provides conversion functions to other formats (16-bit Visual Basic is internally limited to ASCII strings).
- Visual Basic doesn't allow constant variables to contain an array. Therefore extra processing is required to emulate this.

While Visual Basic does not naturally support these features, programmers can construct work-arounds to give their programs similar functionality if they desire.

4.4.4 Characteristics present in Visual Basic

Visual Basic has the following uncommon traits:

- Boolean constant True has numeric value -1. In most other languages, True is mapped to numeric value 1. This is because the Boolean data type is stored as a 16-bit signed integer. In this construct -1 evaluates to 16 binary 1s (the Boolean value True), and 0 as 16 0s (the Boolean value False). This is apparent when performing a Not operation on a 16 bit signed integer value 0 which will return the integer value -1, in other words True = Not False. This inherent functionality becomes especially useful when performing logical operations on the individual bits of an integer such as And, Or, Xor and Not.
- Logical and bitwise operators are unified. This is unlike all the C-derived languages (such as Java or Perl), which have separate logical and bitwise operators.

- Variable array base. Arrays are declared by specifying the upper and lower bounds in a way similar to Pascal and Fortran. It is also possible to use the Option Base statement to set the default lower bound. Use of the Option Base statement can lead to confusion when reading Visual Basic code and is best avoided by always explicitly specifying the lower bound of the array. This lower bound is not limited to 0 or 1, because it can also be set by declaration. In this way, both the lower and upper bounds are programmable. In more subscript-limited languages, the lower bound of the array is not variable. This uncommon trait does not exist in Visual Basic .NET and VBScript.
- Relatively strong integration with the Windows operating system and the Component Object Model.
- Banker's rounding as the default behavior when converting real numbers to integers.
- Integers are automatically promoted to reals in expressions involving the normal division operator (/) so that division of an odd integer by an even integer produces the intuitively correct result. There is a specific integer divide operator (\) which does truncate.
- By default, if a variable has not been declared or if no type declaration character is specified, the variable is of type Variant. However this can be changed with Deftype statements such as DefInt, DefBool, DefVar, DefObj, DefStr. There are 12 Deftype statements in total offered by Visual Basic 6.0.

4.5 Evolution of Visual Basic

VB 1.0 was introduced in 1991. The approach for connecting the programming language to the graphical user interface is derived from a prototype developed by Alan Cooper called Tripod. Microsoft contracted with Cooper and his associates to develop Tripod into a programmable shell for Windows 3.0, under the code name Ruby (no relation to the Ruby programming language).

Tripod did not include a programming language at all, and Ruby contained only a rudimentary command processor sufficient for its role as a Windows shell. Microsoft decided to use the simple Program Manager shell for Windows 3.0 instead of Ruby, and combine Ruby with the Basic language to create Visual Basic.

Ruby provided the "visual" part of Visual Basic—the form designer and editing tools—along with the ability to load dynamic link libraries containing additional controls (then called "gizmos"). Ruby's extensible gizmos later became the VBX interface.

4.6 Timeline of Visual Basic (VB1 to VB6)

- Project 'Thunder' was initiated.
- Visual Basic 1.0 (May 1991) was released for Windows at the Comdex/Windows World trade show in Atlanta, Georgia.



Figure 4.2 Visual Basic for MS-DOS

- Visual Basic 1.0 for DOS was released in September 1992. The language itself was not quite compatible with Visual Basic for Windows, as it was actually the next version of Microsoft's DOS-based BASIC compilers, QuickBASIC and BASIC Professional Development System. The interface was textual, using extended ASCII characters to simulate the appearance of a GUI.
- Visual Basic 2.0 was released in November 1992. The programming environment was easier to use, and its speed was improved. Notably, forms became instantiable objects, thus laying the foundational concepts of class modules as were later offered in VB4.
- Visual Basic 3.0 was released in the summer of 1993 and came in Standard and Professional versions. VB3 included version 1.1 of the Microsoft Jet Database Engine that could read and write Jet (or Access) 1.x databases.

- Visual Basic 4.0 (August 1995) was the first version that could create 32-bit as well as 16-bit Windows programs. It also introduced the ability to write non-GUI classes in Visual Basic.
- With version 5.0 (February 1997), Microsoft released Visual Basic exclusively for 32-bit versions of Windows. Programmers who preferred to write 16-bit programs were able to import programs written in Visual Basic 4.0 to Visual Basic 5.0, and Visual Basic 5.0 programs can easily be converted with Visual Basic 4.0. Visual Basic 5.0 also introduced the ability to create custom user controls, as well as the ability to compile to native Windows executable code, speeding up calculation-intensive code execution.
- Visual Basic 6.0 (Mid 1998) improved in a number of areas, including the ability to create web-based applications. VB6 is currently scheduled to enter Microsoft's "non-supported phase" starting March 2008.
- Mainstream Support for Microsoft Visual Basic 6.0 ended on March 31, 2005. Extended support will end in March 2008. In response, the Visual Basic user community expressed its grave concern and lobbied users to sign a petition to keep the product alive. Microsoft has so far refused to change their position on the matter. Ironically, around this time, it was exposed that Microsoft's new anti-spyware offering, Microsoft AntiSpyware, was coded in Visual Basic 6.0. Windows Defender Beta 2 was rewritten as C++/CLI code, as mentioned in Paul Thurrott's review of this product.

CHAPTER FIVE

DESCRIPTION OF THE SOFTWARE

5.1 Introduction

Microsoft Visual Basic 6.0 has many special tools to create a project. After opening the Visual Basic, a form is displayed to ask the user what kind of project is to be selected. Click Standard Exe then (open) button.

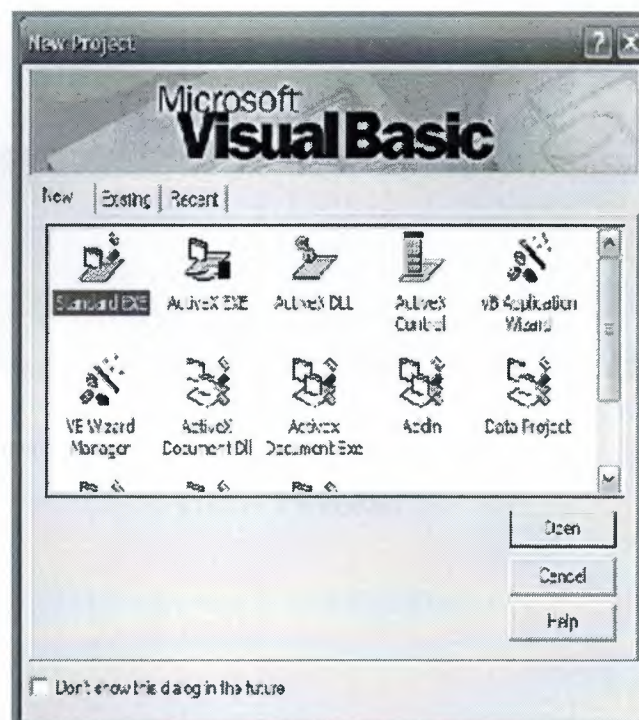


Figure 5.1 Starting a New Project

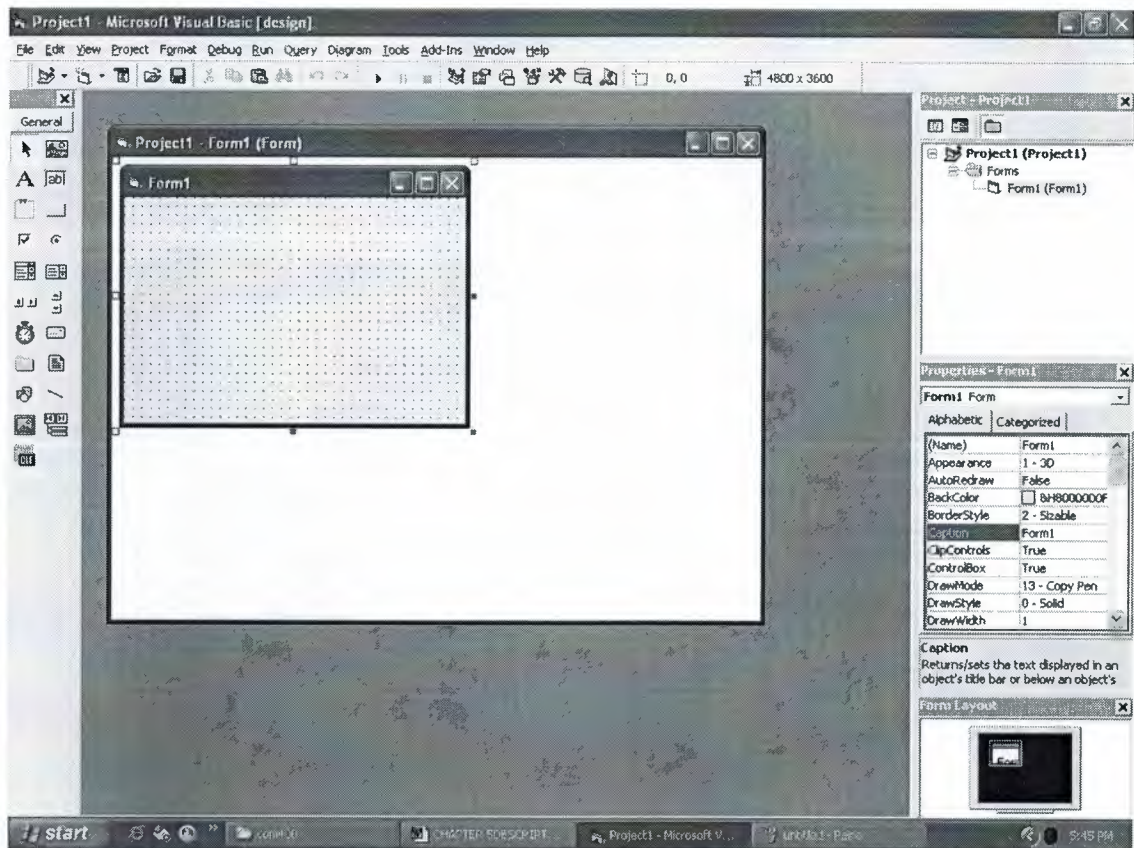


Figure 5.2 A new form will appear and by double clicking this form you can start writing VB codes.

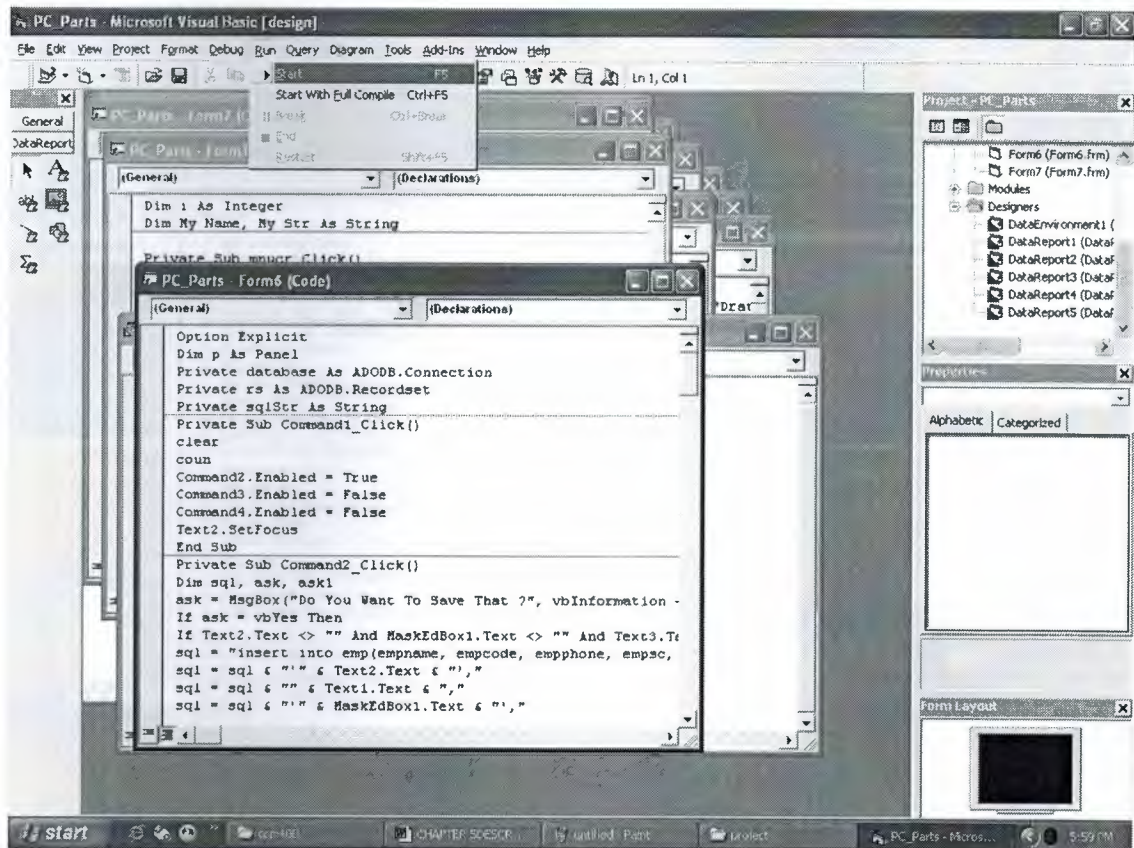


Figure 5.3 After finishing with writing the codes click the start button from the Visual Basic menu or easily press F5 button from Keyboard. The Program will start running unless there are errors on the code.

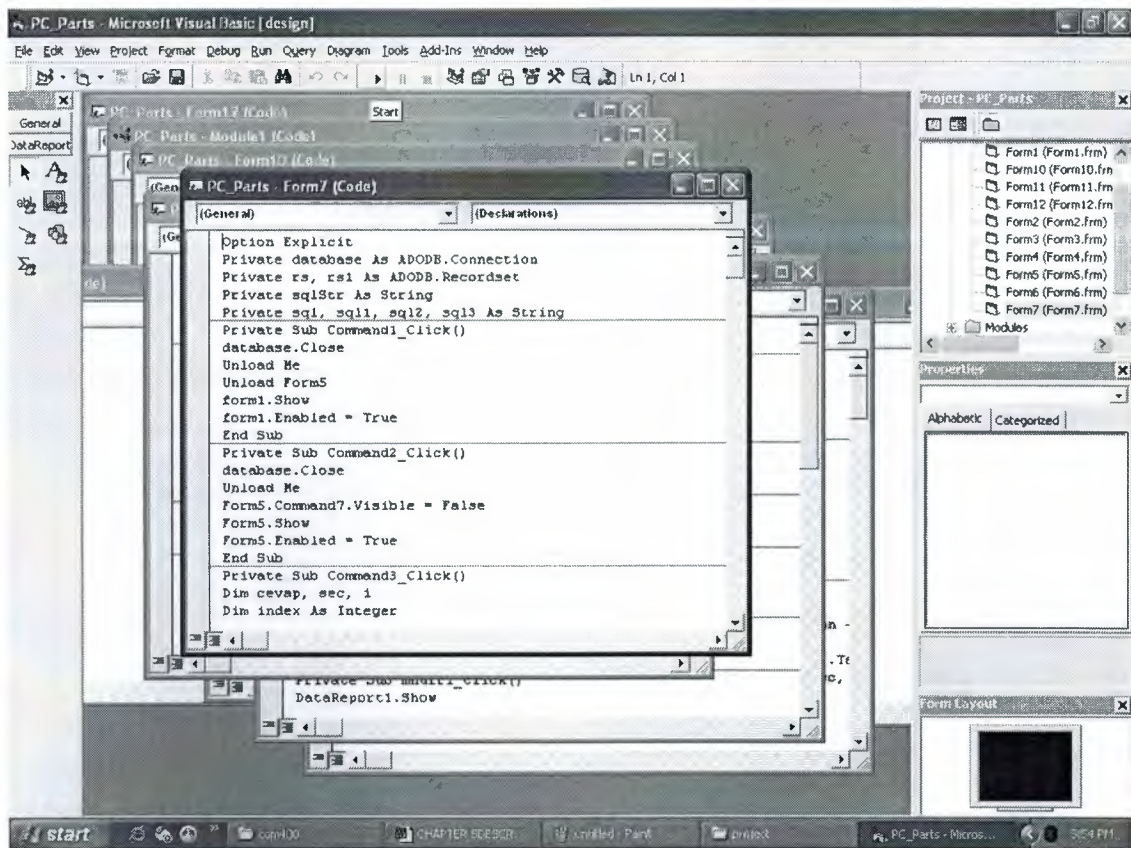


Figure 5.4 After testing the project if every thing works fine click the Visual Basic Menu File and then Save Project button, all the changes will be saved.

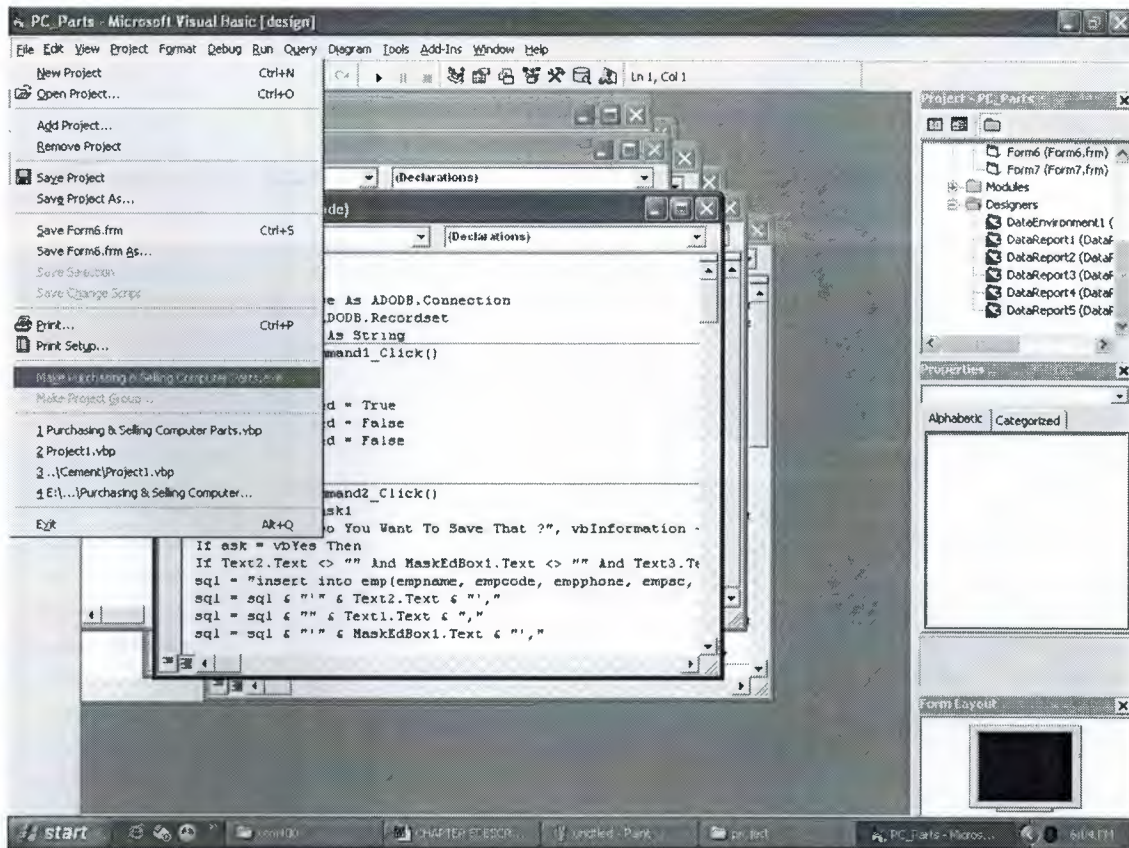


Figure 5.5 Saving a project and Making Executable file of the project is very easy. By clicking File Menu of Visual Basic then select make EXE selection here.

5.2 Description of the forms

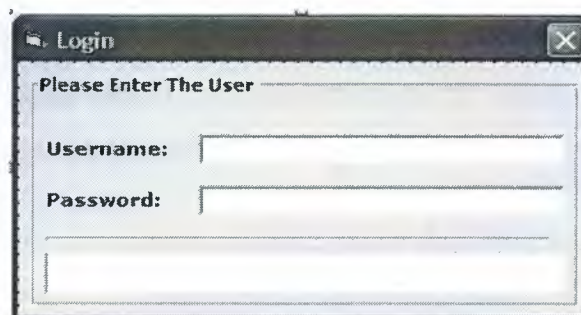


Figure 5.6 Security

In this form users should enter their users name and password to start the program.

If the user typed a wrong username or password a message will be displayed saying invalid username or password, otherwise the user will enter the program successfully.



Figure 5.7 Main Menu

This is the main menu form, in this form there are 8 commands and each command gives different information when you press any one of them.

For the firm records command it will lead you to the firm form, the customer records command will lead you to customer form, employee records will lead you to employee form, account revenue command will lead you to account form, report command will lead you to reports form, user management command will lead you to user management form, about command will lead you to about form, and the exit command will lead you out of the program.

Firm Records

FIRMS RECORDS

Firm Code : 7

Firm Name :

Firm Phone : 0() - -

Firm Fax : 0() - -

Firm Tax No : - - - -

Firm City :

Firm Address :

Firm Reg Date : 4/28/2007

Search For Firm:

Find

Stock Min. Level

New Save Update Delete MainMenu

7:08 PM 4/28/2007

Figure 5.8 Firm record

This is a firm record form, here we can add different firm records with their stocks. In this form we can search, save, update and delete firms according to the firm code. Once a new firm is added also their stocks are added in the database.

Stock Records

STOCKS RECORDS

Firm Code : 2

Firm Name : maher balgasy

Stock Code : 2

Stock Name : kyeboard

Stock Buying Date : 12/28/2006

Stock Unit : 23

Stock Buy Price : 12

Stock Prof Perc :% 1 Tax %

Stock Sell Price : 12.12

Min Stock Lev: 4

Find 2

New Save Edit Delete FirmMenu

MainMenu

Updated Information

New Stock Name : kyeboard

New Stock Unit : 23

New Buy Price : 12

New Prof Perc : 1

New Tax :

New Min Stock Lev : 4

Cancel Save

6:38 PM 4/29/2007

Figure 5.9 Stock record

This is a stock record, here stocks are separated according to their firms. We can search, add, save, delete and edit stocks of computer products. This form shows us the product of the stocks that are available.

Employee Records

Employee Records

Employee Code : 9

Employee Name :

Employee Phone : 0() - -

Employee Security No : - - - -

Employee City :

Employee Address :

Employee Reg Date : 4 /28/2007 ▼

Employee Salary :

Load Picture

Search For Employee:

Find

New Save Update Delete MainMenu

7:14 PM 4/28/2007

Figure 5.10 Employee record

This is an Employee record which shows us all the information needed about an employee, we might want to hire a new employee or we might want search, delete or update information about him or her. So it's so useful and saves time. This employee application form displays for us all the information we need to know about the person.

Customer Records

CUSTOMERS RECORDS

Customer Code : 8

Customer Name :

Customer Phone : 0() ..

Customer Tax No :

Customer City :

Customer Address :

Cus Reg Date : 4/28/2007

Find

New Save Update Delete MainMenu

7:13 PM 4/28/2007

Figure 5.11 Customer record

This is a customer form which shows us the details of a customer who is willing to buy a stock from the shop. Here we can add, search, delete and update the customer information.

Invoice Process

INVOICE

Customer Information

Customer Code: Invoice No:
 Customer Name: Date:
 Phone Number:
 Customer Address: Payment Type: ☒ Cash ☐ Credit

Stock Details

Stock Code	Stock Name	Stock Amount	Selling Price	Min Level	Amount
<input type="text" value="1"/>	<input type="text" value="cd rooom"/>	<input type="text" value="97"/>	<input type="text" value="14.1984"/>	<input type="text" value="2"/>	<input type="text"/>

Stock Code	Stock Name	Quantity	Unit Price	Total

Remove From List Save SUB TOTAL Include Tax:

Customer Menu Main Menu Sold By:

Figure 5.12 Invoice

The invoice form for selling stocks to the customers contains two types of payments,

Cash or credit. This invoice will give us information about stock unit, name, amount, price and minimum level .At the end the sub total Include tax is calculated.

Account Revenue Process

CREDIT

Date : 4/27/2007

Please Select The Criteria :

☐ Invoice No + Credit Amount Find
 ☐ Customer No + Credit Amount Refresh

LIST OF INVOICE BY CUSTOMER CODE					
	Invoice	Customer	Invoice Date	Employee Name	Invoice Amount
▶	11	1	1/4/2007	khadejeh	14.0592
	12	1	1/4/2007	saed	14.0592
	18	1	1/7/2007	fateh	8.3968
	22	2	1/8/2007	mosa	3.8108

Total 4 Record Exist

MainMenu

Figure 5.13 Account revenue

This credit form shows us all the list of credit payments.

This form shows us what the company owes, once the customer pays his or her debits. The company will register the credit form in his or her invoice form.

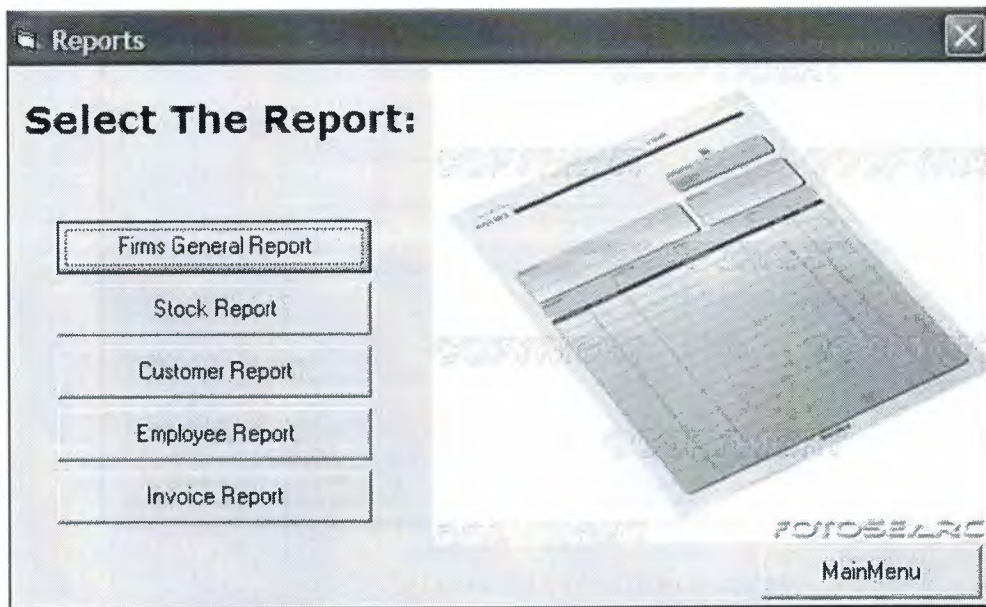


Figure 5.14 Repot

This is a report form, which summaries the whole adding and selling events.

Here we have five reports, first the firm general report that show us a general information about the firms and their stock, second we'll have the stock report which shows us the stock code, name, quantity and the minimum level, third we'll have the customer report which shows us the customer code, name, phone, address and the registration date. Fourth we'll have the employee report which shows us the employee code, name, phone and salary, at the end we'll have the invoice report which shows us the invoice, customer and stock code, also the quantity price and the total invoice.

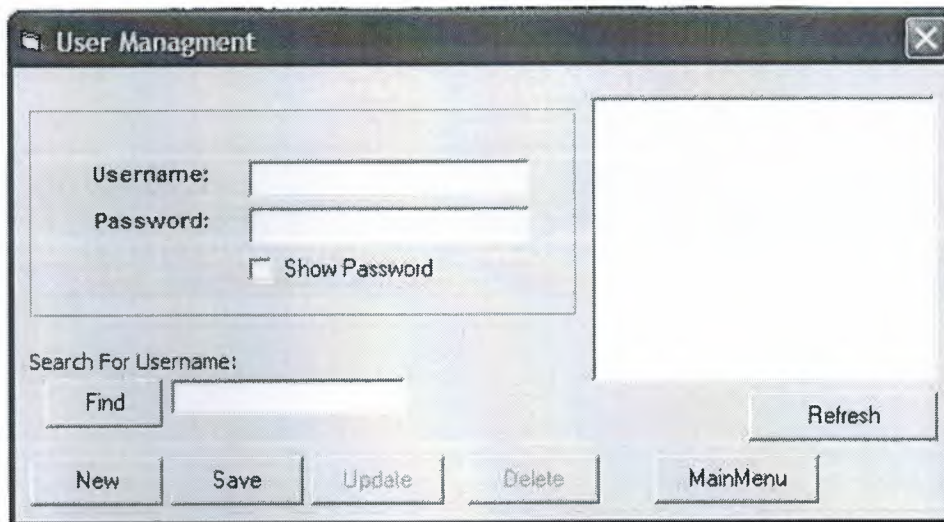
A screenshot of a 'User Management' application window. The window has a title bar with the text 'User Management' and a close button. Inside, there is a form with two input fields for 'Username:' and 'Password:', followed by a checkbox labeled 'Show Password'. Below these is a 'Search For Username:' label, a 'Find' button, and another input field. To the right of the search section is a large empty rectangular box. At the bottom of the window, there are five buttons: 'New', 'Save', 'Update', 'Delete', and 'MainMenu'. A 'Refresh' button is located to the right of the search input field.

Figure 5.15 User management

This is an administration management form, in this form you can change the administer names and password, also you can add a new administer or search, update and delete them from the administration form by the database.

The screenshot shows a software window titled "Stock Records Min. Levels" with a close button (X) in the top right corner. Below the title bar is a header area with the text "STOCK RECORDS". The main content area displays a table with the caption "LIST OF MINIMUM LEVEL GREATER THAN AVAILABLE STOCK UNIT BY STOCKCODE". The table has five columns: "StockCod", "StockName", "FirmCod", "StockUnit", and "Stock Mi". Below the table, there is a large, empty rectangular area. At the bottom of the window, there are three buttons: "Record Not Exist", "FirmMenu", and "MainMenu".

LIST OF MINIMUM LEVEL GREATER THAN AVAILABLE STOCK UNIT BY STOCKCODE				
StockCod	StockName	FirmCod	StockUnit	Stock Mi

Figure 5.16 stock min levels

This form informs me how many stocks are available in the stock room.

When I run out of item this message will appear on my screen to inform me that I am running out of the minimum stock level.

CONCLUSION



Database management has evolved from a specialized computer application to a central component of a modern computing environment. As such, knowledge about database systems has become an essential part of an education in computer science. Our purpose in this project is to present the fundamental concepts of database management. These concepts include aspects of database design, and database-system implementation.

The software described in the project is prepared by Visual Basic 6.0 and Microsoft Access XP. Both of them are powerful tools to create and operate sophisticated data organizations.

Visual Basic 6.0 has many tools to help programmer. However, there are new editions of Visual Basic, the version 6.0 is preferred, to show that this version is also sufficient to make good software.

This project contains many examples of using database (SQL) quires like deleting, updating and inserting information's in the data.

REFERENCE

- [1] <http://www.Wikipedia.com>
- [2] <http://www.sqlcourse.com/table.html>
- [3] <http://www.howstuffworks.com>
- [4] <http://www.Computerhope.com>
- [5] <http://www.google.com>
- [6] Sams Teach Yourself Microsoft Access 2000 in 21 Days
- [7] <http://www.yahoo.com>
- [8] Visual Basic 6 Black Book
- [9] Sam's Teach Yourself MySQL in 21 Days
- [10] www.VisualBasic.com

APPENDIX

SOURCE CODE LISTINGS

The user name form

```
Dim sqlstr As String
Dim database As ADODB.Connection
Dim rs As Recordset

Private Sub StartProgress()
    Dim i As Integer
    Frame3.Visible = True
    X.Width = 1
    For i = 1 To 6255
        X.Width = X.Width + 1
        DoEvents
    Next

    Frame3.Visible = False
End Sub

Private Sub Command1_Click()
    conn
    sqlstr = "select * from users where username='" & Text1.Text & "' and password='"
    & Text2.Text & "'"
    Set rs = database.Execute(sqlstr)

    StartProgress

    If Not rs.EOF Then
        If rs![UserName] = Text1.Text And rs![Password] = Text2.Text Then
            Unload Me
        Else
            MsgBox "Error: Invalid Username or Password!!!", vbCritical
            Text1.SetFocus
        End If
    Else
        MsgBox "Error: Invalid Username or Password!!!", vbCritical
        Text1.SetFocus
    End If
End Sub

Private Sub conn()
    Set database = New ADODB.Connection
    database.CursorLocation = adUseServer
```

```

    sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
"\system.mdb"
    database.Open sqlstr
End Sub

```

The main menu form

```

Dim i As Integer
Dim My_Name, My_Str As String
Dim IsMoving As Boolean

```

```

Private Sub Command1_Click(Index As Integer)
    Select Case Index
        Case Is = 0: Form2.Show vbModal
        Case Is = 1: Form5.Show vbModal
        Case Is = 2: Form6.Show vbModal
        Case Is = 3: Form9.Show vbModal
        Case Is = 4: Form10.Show vbModal
        Case Is = 5: Call Form_QueryUnload(0, 0)
        Case Is = 6: Form14.Show vbModal
        Case Is = 7: Form8.Show vbModal
    End Select
End Sub

```

```

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If MsgBox("Are you sure you want to exit?", vbYesNo + vbQuestion, "Exit") = vbYes
Then End
End Sub

```

```

Private Sub Timer1_Timer()
    Dim sString As String

    My_Name = Left(Label1.Caption, 1)
    My_Str = Right(Label1.Caption, Len(Label1.Caption) - 1)

    Label1.Caption = My_Str + My_Name

    sString = "This Program Is Created By Ahmad Moh`d Eid..."

    If Timer1.Tag = 0 Then
        Me.Caption = sString
        Timer1.Tag = 1
    ElseIf Timer1.Tag < Len(sString) Then
        Me.Caption = Right(sString, Len(sString) - Timer1.Tag)
        Timer1.Tag = Timer1.Tag + 1
    End If
End Sub

```

```

ElseIf Timer1.Tag = Len(sString) Then
    Me.Caption = sString
    Timer1.Tag = 0

End If
End Sub

```

The firm record form

Option Explicit

```

Private database As ADODB.Connection
Private rs As ADODB.Recordset
Private sqlstr As String

```

```

Dim p As Panel

```

```

Private Sub cmdNewfrm_Click()
    Call clear
    Call coun

```

```

    cmdStkEnt.Visible = False
    cmdfrmSav.Enabled = True
    cmdfrmUpd.Enabled = False
    cmdfrmDel.Enabled = False

```

```

    txtFrmName.SetFocus
End Sub

```

```

Private Sub cmdStkMinlv1_Click()
    Form12.Show vbModal
End Sub

```

```

Private Sub cmdfrmSav_Click()
    Dim sql, rs1, rs11

```

```

    rs1 = MsgBox("Do You Want To Save That ?", vbInformation + vbYesNo, "Save")

```

```

    If rs1 = vbYes Then
        If txtFrmName.Text <> "" And txtFrmPhn.Text <> ""
            And txtFrmFax.Text <> "" And txtFrmTax.Text <> ""
            And txtFrmCity.Text <> "" And txtFrmAdd.Text <> "" Then

```

```

            sql = "insert into firms(firmname, firmcode, firmphone, firmfax, firmtaxno,
firmcity, firmadres, firmregdate) values("
            sql = sql & """" & txtFrmName.Text & """, "

```



```

sql = sql & "" & txtFrmCode.Text & ","
sql = sql & "" & txtFrmPhn.Text & ","
sql = sql & "" & txtFrmFax.Text & ","
sql = sql & "" & txtFrmTax.Text & ","
sql = sql & "" & txtFrmCity & ","
sql = sql & "" & txtFrmAdd & ","
sql = sql & "" & DtPckfrmRegDat.Value & ""

database.Execute (sql)

Dim i As Integer

StartProgress

rslt1 = MsgBox("Firm Information Save Successful! ", , "Saved")

Private Sub cmdfrmMain_Click()
database.Close
Unload Me
End Sub
Private Sub cmdfrmFnd_Click()
Dim find As Integer
conn

If Text7.Text = "" Then
MsgBox "Error: Please select an employee first!", vbCritical
Exit Sub
End If

find = Val(Text7.Text)
sqlstr = "select * from firms where firmcode=" & find & ""
Set rs = database.Execute(sqlstr)
If rs.EOF Then
MsgBox ("The Wanted Firm is Not Available!")
Else
Dim i As Integer

StartProgress

txtFrmCode.Text = rs![firmcode]
txtFrmName.Text = rs![firmname]
txtFrmPhn.Text = rs![firmphone]
txtFrmFax.Text = rs![firmfax]
txtFrmTax.Text = rs![firmtaxno]
txtFrmCity.Text = rs![firmcity]

```

```

txtFrmAdd.Text = rs![firmadres]
DtPckfrmRegDat.Value = rs![firmregdate]
txtFrmName.SetFocus
cmdfrmSav.Enabled = False
cmdfrmUpd.Enabled = True
cmdfrmDel.Enabled = True
cmdStkEnt.Visible = True
End If
rs.Close
End Sub
Private Sub cmdStkEnt_Click()
Form4.Text1.Text = txtFrmCode.Text
Form4.Text2.Text = txtFrmName.Text
Form4.Show vbModal
End Sub

Private Sub Form_Load()
coun
With StatusBar1.Panels
    Set p = .Add(, , sbrTime)
    Set p = .Add(, , sbrDate)
End With
DtPckfrmRegDat.Value = Date
End Sub
Private Sub clear()
txtFrmCode.Text = ""
txtFrmName.Text = ""
txtFrmCity.Text = ""
txtFrmAdd.Text = ""
txtFrmPhn.Mask = ""
txtFrmPhn.Text = ""
txtFrmPhn.Mask = "0(999)999-99-99"
txtFrmFax.Mask = ""
txtFrmFax.Text = ""
txtFrmFax.Mask = "0(999)999-99-99"
txtFrmTax.Mask = ""
txtFrmTax.Text = ""
txtFrmTax.Mask = "999-999-999-999-999"
DtPckfrmRegDat.Value = Date
End Sub
Private Sub coun()
Dim Count, Count1
conn
Set rs = New ADODB.Recordset
Count = "select * from Firms"
Set rs = database.Execute(Count)

```

```

If rs.EOF Then
    cmdfrmFnd.Enabled = False
    txtFrmCode.Text = 1
Else
    Count1 = "select max(firmcode) as cis from firms"
    Set rs = database.Execute(Count1)
    txtFrmCode.Text = rs![cis] + 1
End If
rs.Close
End Sub
Public Sub conn()
Set database = New ADODB.Connection
    database.CursorLocation = adUseServer
    sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
"\system.mdb"
    database.Open sqlstr
End Sub
Private Sub StartProgress()

    Dim i As Integer
    Frame1.Visible = True
    X.Width = 1
    For i = 1 To 6255
        X.Width = X.Width + 1
        DoEvents
    Next

    Frame1.Visible = False

End Sub

```

The stock record form

```

Option Explicit
Private database As ADODB.Connection
Private rs As ADODB.Recordset
Private sqlstr As String
Dim p As Panel
Dim ah As Boolean
Dim a, b, d, e, g, h, f, l As Double
Dim m, n, decrip, sql1
Private Sub Command1_Click()
clear
coun
Command2.Enabled = True
Command3.Enabled = False

```



```

Command4.Enabled = False
Text4.SetFocus
End Sub
Private Sub Command2_Click()
If care Then Exit Sub
If ah = True Then
Dim sql, sql1, rslt, rslt1
rslt = MsgBox("Do You Want To Save That ?", vbInformation + vbYesNo, "Save")
If rslt = vbYes Then
If Text4.Text <> "" And Text5.Text <> "" And Text6.Text <> "" And Text8.Text <> ""
And Text10.Text <> "" And Text11.Text <> "" Then
sql = "insert into stocks(firmname, firmcode, stockcode, stockname, stockminl, stockbd,
stockunit, stockbp, stockpperc, stocksellp) values("
sql = sql & "" & Text2.Text & ","
sql = sql & "" & Text1.Text & ","
sql = sql & "" & Text3.Text & ","
sql = sql & "" & Text4.Text & ","
sql = sql & "" & Text11 & ","
sql = sql & "" & Label11.Caption & ","
sql = sql & "" & a & ","
sql = sql & "" & b & ","
sql = sql & "" & h & ","
sql = sql & "" & Text9 & ")")
database.Execute (sql)
m = Text5.Text
l = Text6.Text
n = m * l
decrip = Text3 + "," + "no" + "st"
sql1 = "insert into account(accdate, expense, revenue, description, expcode) values("
sql1 = sql1 & "" & Label11.Caption & ","
sql1 = sql1 & "" & n & ","
sql1 = sql1 & "" & 0 & ","
sql1 = sql1 & "" & decrip & ","
sql1 = sql1 & "" & Text3.Text & ")")
database.Execute (sql1)
Dim i As Integer

StartProgress

rslt1 = MsgBox("stock Information Save Successful! ", , "Saved")
Command2.Enabled = False
Command3.Enabled = True
Command4.Enabled = True
Command6.Enabled = True
Else
rslt1 = MsgBox("Please Fill The Other Texts!")

```

```

Command3.Enabled = False
Command4.Enabled = False
Text4.SetFocus
End If
End If
End If
End Sub
Private Sub Command6_Click()
Dim find As Integer
conn

    If Text7.Text = "" Then
        MsgBox "Error: Please select an employee first!", vbCritical
        Exit Sub
    End If

    find = Val(Text7.Text)
    sqlstr = "select * from stocks where stockcode=" & find & ""
    Set rs = database.Execute(sqlstr)
    If rs.EOF Then
        MsgBox ("The Wanted Stock is Not Available!")
    Else
        Dim i As Integer

        StartProgress

        Text1.Text = rs![firmcode]
        Text2.Text = rs![firmname]
        Text3.Text = rs![stockcode]
        Text4.Text = rs![stockname]
        Text5.Text = rs![stockunit]
        Text6.Text = rs![stockbp]
        Text8.Text = rs![stockpperc]
        Text9.Text = rs![stocksellp]
        Label11.Caption = rs![stockbd]
        Text11.Text = rs![stockminl]
        Command2.Enabled = False
        Command3.Enabled = True
        Command4.Enabled = True
    End If
    rs.Close
End Sub

Private Sub Form_Load()
    coun
    With StatusBar1.Panels

```

```

        Set p = .Add(, , sbrTime)
        Set p = .Add(, , sbrDate)
    End With
    Label11.Caption = Date
End Sub
Private Sub Form_Unload(Cancel As Integer)
    Unload Me
End Sub

Private Sub Text10_Change()
    On Error Resume Next
    a = Text5.Text
    b = Text6.Text
    h = Text8.Text
    d = Val(a) * Val(b)
    e = ((d * Val(h)) / 100) + d
    f = e / a
    g = ((f * Val(Text10.Text)) / 100) + f
    Text9.Text = g
End Sub
Private Sub coun()
    Dim Count, Count1
    conn

    Set rs = New ADODB.Recordset
    Count = "select * from stocks"

    Set rs = database.Execute(Count)

    If rs.EOF Then
        Command6.Enabled = False
        Text3.Text = 1
    Else
        Count1 = "select max(stockcode) as cis from stocks"
        Set rs = database.Execute(Count1)
        Text3.Text = rs![cis] + 1
    End If
    rs.Close
End Sub
Public Sub conn()
    Set database = New ADODB.Connection
    database.CursorLocation = adUseClient
    sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
    "\system.mdb"
    database.Open sqlstr
End Sub

```



```

Private Sub clear()
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text8.Text = ""
Text9.Text = ""
Text10.Text = ""
Text11.Text = ""
End Sub

```

```

Private Sub StartProgress()

```

```

    Dim i As Integer
    Frame2.Visible = True
    X.Width = 1
    For i = 1 To 6255
        X.Width = X.Width + 1
        DoEvents
    Next

```

```

    Frame2.Visible = False

```

```

End Sub

```

Stock record Min level

```

Option Explicit
Dim RST As New ADODB.Recordset
Private database As ADODB.Connection
Private rs As ADODB.Recordset
Private sqlstr As String
Private Sub Command1_Click()
Unload Me
If Not Form2.Visible Then Form2.Show
End Sub
Private Sub Command2_Click()
Unload Me
If Not Form1.Visible Then Form1.Show
End Sub

```

```

Private Sub Form_Load()
com
End Sub
Private Sub com()
On Error Resume Next
conne

```

```

Set RST = database.Execute("Select stockcode, stockname,firmcode, stockunit,stockminl
from stocks where stockminl >= stockunit")
Set DataGrid1.DataSource = RST
DataGrid1.Caption = "LIST OF MINIMUM LEVEL GREATER THAN AVAILABLE
STOCK UNIT BY STOCKCODE"
DataGrid1.Columns(0).Caption = "StockCode"
DataGrid1.Columns(1).Caption = "StockName"
DataGrid1.Columns(2).Caption = "FirmCode"
DataGrid1.Columns(3).Caption = "StockUnit"
DataGrid1.Columns(4).Caption = "Stock Min.Level"
DataGrid1.Columns(4).Alignment = dbgRight
Select Case RST.RecordCount
Case Is > 1
Label1.Caption = "Total " & Trim(Str(RST.RecordCount)) & " Min. Level >= Stock
Quantity Record Exist"
Case Is = 1
Label1.Caption = "Total " & Trim(Str(RST.RecordCount)) & " Min. Level >= Stock
Quantity Record Exist"
Case Is = 0
Label1.Caption = "Record Not Exist"
End Select
End Sub
Private Sub conne()
Set database = New ADODB.Connection
    database.CursorLocation = adUseClient
    sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
"\system.mdb"
    database.Open sqlstr
End Sub
Private Sub Form_Unload(Cancel As Integer)
Unload Me
End Sub

```

Employee record form

```

Option Explicit
Dim p As Panel
Private database As ADODB.Connection
Private rs As ADODB.Recordset
Private sqlstr As String

Private Sub cmdLdPic_Click()
On Error GoTo ErrH
    With CDG
        .CancelError = True
        .DialogTitle = "Select Student Picture"
    End With

```

```

.Flags = cdlOFNFileMustExist
.Filter = "JPEG Files(*.jpg)|*.jpg;*.jpeg|Bitmap Images(*.bmp)|*.bmp|GIF
Files(*.gif)|*.gif|All Files(*.*)|*.*"
.ShowOpen

If Not .FileName = vbNullString Then
    If CheckFile(.FileName) Then _
        PicEmp.Picture = LoadPicture(.FileName)
        PicEmp.Tag = .FileName
        PicEmp.ToolTipText = "Picture Loaded: (" & .FileName & ")"
    Else
        MsgBox "Error: Picture Is Not Loaded!!!", vbCritical, "Error: Access Error"
    End If
End With
Exit Sub

```

```

ErrH:
End Sub

```

```

Public Function CheckFile(FileName$) As Boolean
    On Error GoTo FileNotFound
    Dim X&
    X = FileLen(FileName)
    If X >= 0 Then CheckFile = True: Exit Function

```

```

FileNotFound:
    If Err.Number = 53 Then CheckFile = False: Exit Function
End Function

```

```

Private Sub Command1_Click()
    clear
    coun
    Command2.Enabled = True
    Command3.Enabled = False
    Command4.Enabled = False
    Text2.SetFocus
End Sub

```

```

Private Sub SetEmpPic(DoSet As Boolean, EmpNum As String, objPic As PictureBox)
    If DoSet Then
        Dim EmpPicFile As String
        EmpPicFile = App.Path & "\Employers\" & EmpNum & ".jpg"

        If CheckFile(EmpPicFile) = True Then
            Set objPic.Picture = LoadPicture(EmpPicFile)
        Else

```



```

        Set objPic.Picture = Nothing
        MsgBox "Could Not Load Employers Picture!!!" & vbCrLf & "File Access
Error", vbCritical, "Error: File Error"
    End If

```

```

Else
    objPic.Picture = Nothing
End If
End Sub

```

```

Private Sub Command6_Click()
    Dim find As Integer
    conn

    If Text6.Text = "" Then
        MsgBox "Error: Please select an employee first!", vbCritical
        Exit Sub
    End If

```

```

    find = Val(Text6.Text)
    sqlstr = "select * from emp where empcode=" & find & ""

```

```

    Set rs = database.Execute(sqlstr)
    If rs.EOF Then
        MsgBox ("The Wanted Employee is Not Available!")
        Call SetEmpPic(False, Text1.Text, PicEmp)
    Else

```

```

StartProgress

```

```

    Text1.Text = rs![EmpCode]
    Text2.Text = rs![empname]

```

```

    Call SetEmpPic(True, Text1.Text, PicEmp)

```

```

    MaskedTextBox1.Text = rs![empphone]
    MaskedTextBox3.Text = rs![empsc]

```

```

    Text3.Text = rs![empcity]
    Text4.Text = rs![empadres]
    Text5.Text = rs![empsal]

```

```

    DTPicker1.Value = rs![emprd]

```

```

    Text2.SetFocus

```

```

        Command2.Enabled = False
        Command3.Enabled = True
        Command4.Enabled = True
    End If
    rs.Close
End Sub

Private Sub Form_Load()
    coun
    With StatusBar1.Panels
        Set p = .Add(, , sbrTime)
        Set p = .Add(, , sbrDate)
    End With
    DTPicker1.Value = Date
End Sub

Private Sub clear()
    Text1.Text = ""
    Text2.Text = ""
    Text3.Text = ""
    Text4.Text = ""
    Text5.Text = ""
    MaskedTextBox1.Mask = ""
    MaskedTextBox1.Text = ""
    MaskedTextBox1.Mask = "0(999)999-99-99"
    MaskedTextBox3.Mask = ""
    MaskedTextBox3.Text = ""
    MaskedTextBox3.Mask = "999-999-999-999-999"
    DTPicker1.Value = Date
End Sub

Private Sub coun()
    Dim Count, Count1
    conn
    Set rs = New ADODB.Recordset
    Count = "select * from emp"
    Set rs = database.Execute(Count)
    If rs.EOF Then
        Command6.Enabled = False
        Text1.Text = 1
    Else
        Count1 = "select max(empcode) as cis from emp"
        Set rs = database.Execute(Count1)
        Text1.Text = rs![cis] + 1
    End If
    rs.Close
End Sub

Public Sub conn()

```

```

Set database = New ADODB.Connection
database.CursorLocation = adUseClient
sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
"\system.mdb"
database.Open sqlstr
End Sub

```

```

Private Sub Text5_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then
    KeyAscii = 0
    SendKeys "{Tab}"
ElseIf InStr(("1234567890" & vbBack & ""), Chr(KeyAscii)) = 0 Then
    KeyAscii = 0
End If
End Sub

```

```

Private Sub StartProgress()

    Dim i As Integer
    Frame1.Visible = True
    X.Width = 1
    For i = 1 To 6255
        X.Width = X.Width + 1
        DoEvents
    Next

```

```

    Frame1.Visible = False

```

```

End Sub

```

Customer record form

```

Option Explicit
Dim p As Panel
Private database As ADODB.Connection
Private rs As ADODB.Recordset
Private sqlstr As String
Private Sub Command1_Click()
clear
coun
Command2.Enabled = True
Command3.Enabled = False
Command4.Enabled = False
Command7.Visible = False
Text2.SetFocus
End Sub

```



```

Private Sub Command2_Click()
Dim sql, rslt, rslt1
rslt = MsgBox("Do You Want To Save That ?", vbInformation + vbYesNo, "Save")
If rslt = vbYes Then
If Text2.Text <> "" And MaskedTextBox1.Text <> "" And Text3.Text <> "" And
MaskedTextBox3.Text <> "" And Text3.Text <> "" And Text4.Text <> "" Then
sql = "insert into cus(cusname, cuscode, cusphone, custaxno, cuscity, cusadres, cusrd)
values("
sql = sql & "" & Text2.Text & ","
sql = sql & "" & Text1.Text & ","
sql = sql & "" & MaskedTextBox1.Text & ","
sql = sql & "" & MaskedTextBox3.Text & ","
sql = sql & "" & Text3 & ","
sql = sql & "" & Text4 & ","
sql = sql & "" & DTPicker1.Value & """)
database.Execute (sql)

```

StartProgress

```

rslt1 = MsgBox("Customer Information Save Successful! ", , "Saved")
Command2.Enabled = False
Command3.Enabled = True
Command4.Enabled = True
Command7.Visible = True
Command6.Enabled = True
Else
rslt1 = MsgBox("Please Fill The Other Texts!", vbCritical, "Customer")
Command6.Enabled = True
Command3.Enabled = False
Command4.Enabled = False
Text2.SetFocus
End If
End If
End Sub

```

```

Private Sub Command6_Click()
Dim find As Integer
conn

```

```

If Text6.Text = "" Then
MsgBox "Error: Please select an employee first!", vbCritical
Exit Sub
End If

```

```

find = Val(Text6.Text)
sqlstr = "select * from cus where cuscode=" & find & ""

```

```

Set rs = database.Execute(sqlstr)
If rs.EOF Then
MsgBox ("The Wanted Customer is Not Available!")
Else

```

StartProgress

```

Text1.Text = rs![cuscode]
Text2.Text = rs![cusname]
MaskedTextBox1.Text = rs![cusphone]
MaskedTextBox3.Text = rs![custaxno]
Text3.Text = rs![cuscit]
Text4.Text = rs![cusadres]
DTPicker1.Value = rs![cusrd]
Text2.SetFocus
Command7.Visible = True
Command2.Enabled = False
Command3.Enabled = True
Command4.Enabled = True
End If
rs.Close
End Sub
Private Sub Command7_Click()
Dim X$
X = "select empname from emp"
Set rs = database.Execute(X)

If rs.EOF Then
MsgBox "Error: Employee table is empty," & vbNewLine & _
"Please add one employee at least before continuing.", vbCritical

Exit Sub
Else
Form7.Text4.Text = Text1.Text
Form7.Text5.Text = Text2.Text
Form7.Text7.Text = MaskedTextBox1.Text
Form7.Text6.Text = Text4.Text
Form7.Show vbModal
End If
End Sub
Private Sub Form_Load()
coun
With StatusBar1.Panels
Set p = .Add(, , sbrTime)
Set p = .Add(, , sbrDate)

```

```

End With
DTPicker1.Value = Date
End Sub
Private Sub clear()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
MaskedTextBox1.Mask = ""
MaskedTextBox1.Text = ""
MaskedTextBox1.Mask = "0(999)999-99-99"
MaskedTextBox3.Mask = ""
MaskedTextBox3.Text = ""
MaskedTextBox3.Mask = "999-999-999-999-999"
DTPicker1.Value = Date
End Sub
Private Sub coun()
Dim Count, Count1
conn
Set rs = New ADODB.Recordset
Count = "select * from cus"
Set rs = database.Execute(Count)
If rs.EOF Then
    Command6.Enabled = False
    Text1.Text = 1
Else
    Count1 = "select max(cuscode) as cis from cus"
    Set rs = database.Execute(Count1)
    Text1.Text = rs![cis] + 1
End If
rs.Close
End Sub
Public Sub conn()
Set database = New ADODB.Connection
database.CursorLocation = adUseClient
sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
"\system.mdb"
database.Open sqlstr
End Sub

Private Sub StartProgress()

    Dim i As Integer
    Frame1.Visible = True
    X.Width = 1
    For i = 1 To 6255

```



```

    X.Width = X.Width + 1
    DoEvents
Next

```

```

    Frame1.Visible = False
End Sub

```

The invoice form

```

Option Explicit
Private database As ADODB.Connection
Private rs, RS1, RS3 As ADODB.Recordset
Private sqlstr As String
Private sql, sql1, sql2, sql3 As String

```

```

Private Sub Combo2_Click()
    Text8.Text = List2.List(Combo2.ListIndex)
    Text8.SetFocus
    SendKeys "{enter}"
End Sub

```

```

Private Sub coun()
    Dim Count, Count1
    conn
    Set rs = New ADODB.Recordset
    Count = "select incode from invoice2"
    Set rs = database.Execute(Count)
    If rs.EOF Then
        Label9.Caption = 1
    Else
        Count1 = "select max(incode) as cis from invoice2"
        Set rs = database.Execute(Count1)
        Label9.Caption = rs![cis] + 1
    End If
    Count = "select empname from emp"
    Set rs = database.Execute(Count)
    If rs.EOF Then
        MsgBox "Error: Employee table is empty," & vbNewLine & _
            "Please add one employee at least before continuing.", vbCritical
        Call Command1_Click
        Exit Sub
    Else
        Count1 = "select empname from emp"
        Set rs = database.Execute(Count1)
        While Not rs.EOF
            Combo1.AddItem rs.Fields![empname]
            rs.MoveNext
        End While
    End If

```

```

DoEvents
Wend
End If
rs.Close
End Sub
Private Sub Command4_Click()
    Dim k, i, rslt, toplam, stk, stk2, sq, sq1, sq2, decrip

    If Combo1.Text = "Select Employee" Then
        rslt = MsgBox("Please select the employee name", vbInformation, "Invoice")
    ElseIf List1(0).ListCount = 0 Then
        rslt = MsgBox("Please select one stock at least", vbInformation, "Invoice")
    Else
        rslt = MsgBox("Do You Want to Save The Invoice ?", vbYesNo + vbQuestion,
"Invoice Saving")
        If rslt = vbYes Then
            conn
            For i = 0 To List1(0).ListCount - 1
                stk = Val(List1(0).List(i))
                stk2 = List1(1).List(i)

                sq = Val(List1(2).List(i))
                sq1 = Val(List1(3).List(i))
                sq2 = Val(List1(4).List(i))

                sql1 = "insert into invoice1(incod, cuscode, stockcode, sname, quan, uprice,
totp) values ("
                sql1 = sql1 & "" & Label9 & ","
                sql1 = sql1 & "" & Text4.Text & ","
                sql1 = sql1 & "" & stk & ","
                sql1 = sql1 & "" & stk2 & ","
                sql1 = sql1 & "" & sq & ","
                sql1 = sql1 & "" & sq1 & ","
                sql1 = sql1 & "" & sq2 & ")"

                database.Execute (sql1)
                sql2 = "update stocks set stockunit=stockunit-" & sq & " where stockcode=" &
stk & ""
                database.Execute (sql2)
            Next

            If Option1 = True Then
                toplam = 0
                decrip = Label9 + "," + "no" + "Invoice"
            End If
        End If
    End If
End Sub

```

```

        sql3 = "insert into account(accdate, expense, revenue, description, expcode)
values('" & Label12 & "', " & 0 & ", " & Text2.Text & ", " & decrip & "', " & toplam & "
)"

```

```

        database.Execute (sql3)

```

```

ElseIf Option2 = True Then
    toplam = Text2.Text

```

```

End If

```

```

Private Sub Form_Activate()
On Error Resume Next
Dim MySql$
Text8.SetFocus
conn
MySql = "select stockname, stockcode from stocks"
Set RS3 = database.Execute(MySql)

```

```

Combo2.clear

```

```

Do While Not RS3.EOF
    Combo2.AddItem RS3![stockname]
    List2.AddItem RS3![stockcode]
    DoEvents
    RS3.MoveNext
    DoEvents

```

```

Loop
End Sub
Private Sub Form_Load()
Option1 = True
Label12 = Date
coun
End Sub

```

```

Private Sub List1_Click(Index As Integer)
Dim secind, topin, j
On Error Resume Next
secind = List1(Index).ListIndex
topin = List1(Index).TopIndex
For j = 0 To 4
    List1(j).ListIndex = secind
    List1(j).TopIndex = topin
Next
End Sub

```

```

Private Sub Text11_KeyPress(KeyAscii As Integer)
Dim ans
If KeyAscii = 13 Then
KeyAscii = 0
If Val(Text11.Text) > Val(Text9.Text) Then
ans = MsgBox("Stock not enough to sell this amount / Available stock is =" & Text9 & "
unit", vbCritical, "Invoice")
Else
If Val(Text11.Text) <= 0 Then
MsgBox "Error: Quantity Error, At least 1 item should be sold!!!", vbCritical
Exit Sub
End If
List1(0).AddItem Text8.Text
List1(1).AddItem Combo2.Text
List1(2).AddItem Text11.Text
List1(3).AddItem Text10.Text
List1(4).AddItem (Val(Text10.Text) * Val(Text11.Text))
Text11.Enabled = False
ClearAll
Text8.Text = ""
Text8.SetFocus
Dim i, a, b
For i = 0 To List1(4).ListCount
a = Val(List1(4).List(i))
b = b + a
Next
Text2.Text = b
End If
ElseIf InStr(("1234567890" & vbBack & ""), Chr(KeyAscii)) = 0 Then
KeyAscii = 0
End If
End Sub
Private Sub conn()
Set database = New ADODB.Connection
database.CursorLocation = adUseClient
sqlstr = "provider=Microsoft.jet.oledb.3.51; Data Source=" & App.Path &
\system.mdb"
database.Open sqlstr
End Sub
Public Sub ClearAll()
Text8.Text = ""
Combo2.Text = ""
Text9.Text = ""
Text10.Text = ""
Text11.Text = ""
End Sub

```


The account revenue form

```
Option Explicit
Dim rs, RS1, RS2, RS3 As New ADODB.Recordset
Private DB As ADODB.Connection
Private RST As ADODB.Recordset
Private ConnStr As String
Dim sqlstr, textq, a, decip, b, c, rslt

Private Sub Command1_Click()
If Option1.Value = True Then
On Error Resume Next
If Text1.Text = "" Then
MsgBox "Please Enter The Selected Criteria!", vbCritical, "Account"
Frame1.Visible = False
Text1.SetFocus
Else
conne
sqlstr = "select incode, cuscode, invdate, empname, subtot from invoice2 where incode = " & Text1.Text & " and subtot>0 "
Set rs = DB.Execute(sqlstr)
If rs.EOF Then
MsgBox "Wanted Invoice Does Not Exist!", vbCritical, "Account"
Label2.Visible = False
Label3.Visible = False
Frame1.Visible = False
Else
textq = "select sum(subtot) as com from invoice2 where incode=" & Text1.Text & ""
Set RS1 = DB.Execute(textq)
a = rs![cuscode]
Set DG.DataSource = rs
Call SetGridData
Label3.Caption = RS1![com]
Label2.Visible = True
Label3.Visible = True
Frame1.Visible = True
End If
End If
End If
If Option2 = True Then
On Error Resume Next
If Text1.Text = "" Then
MsgBox "Please Enter The Selected Criteria!", vbCritical, "Account"
Frame1.Visible = False
Text1.SetFocus
Else
```

```

conne
sqlstr = "select incode, cuscode, invdate, empname, subtotal from invoice2 where cuscode
= " & Text1.Text & " and Subtot>0 "
Set rs = DB.Execute(sqlstr)
If rs.EOF Then
MsgBox "Wanted Customer No Not Exist!", vbCritical, "Account"
Label2.Visible = False
Label3.Visible = False
Frame1.Visible = False
Else
textq = "select sum(subtot) as com from invoice2 where cuscode=" & Text1.Text & ""
Set RS1 = DB.Execute(textq)
a = rs![cuscode]
Call SetGridData
Label3.Caption = RS1![com]
Label2.Visible = True
Label3.Visible = True
Frame1.Visible = True
End If
End If
End If
Text1.Text = ""
Text1.SetFocus
End Sub

```

```

Private Sub Form_Load()
Label12.Caption = Date
comtot
Command1.Enabled = False
End Sub
Private Sub comtot()
On Error Resume Next
conne
Set rs = DB.Execute("Select incode, cuscode, invdate, empname, subtotal from invoice2
where subtotal >0")
Set DG.DataSource = rs
Call SetGridData
Select Case rs.RecordCount
Case Is > 1
Label8.Caption = "Total " & Trim(Str(rs.RecordCount)) & " Record Exist"
Case Is = 1
Label8.Caption = "Total " & Trim(Str(rs.RecordCount)) & " Record Exist"
Case Is = 0
Label8.Caption = "Record Not Exist"
End Select
End Sub

```

```

Private Sub Option2_Click()
Label5.Visible = True
Label6.Visible = False
Text1.SetFocus
Command1.Enabled = True
End Sub
Private Sub SetGridData()
Set DG.DataSource = rs
DG.Refresh
DG.Caption = "LIST OF INVOICE BY CUSTOMER CODE"
DG.Columns(0).Caption = "Invoice No"
DG.Columns(1).Caption = "Customer No"
DG.Columns(2).Caption = "Invoice Date"
DG.Columns(3).Caption = "Employee Name"
DG.Columns(4).Caption = "Invoice Amount"
DG.Columns(4).Alignment = dbgRight
End Sub

```

The user management form

```

Private database As ADODB.Connection
Private rs As ADODB.Recordset
Private sqlstr As String

Private Sub Check1_Click()
    txtpass.PasswordChar = IIf(Check1.Value = vbChecked, "", "*")
End Sub

Private Sub cmdNewfrm_Click()
On Error Resume Next
    Call DoClear
    Call coun

    cmdfrmSav.Enabled = True
    cmdfrmUpd.Enabled = False
    cmdfrmDel.Enabled = False

    txtusrn.SetFocus
End Sub

Private Sub cmdfrmSav_Click()
Dim sql, rs!t, rs!t1

    rs!t = MsgBox("Do You Want To Save That ?", vbInformation + vbYesNo, "Save")

    If rs!t = vbYes Then

```

```

If txtusrn.Text <> "" And txtpass.Text <> "" Then

    sql = "insert into users(username, password) values('" & txtusrn.Text & "', '" &
txtpass.Text & "')"

    database.Execute (sql)

    StartProgress

    rslt1 = MsgBox("User was saved successfully! ", , "Saved")

    cmdfrmSav.Enabled = False
    cmdfrmUpd.Enabled = True
    cmdfrmDel.Enabled = True
    cmdfrmFnd.Enabled = True

Else
    rslt1 = MsgBox("Please Fill All Texts!")

    cmdfrmUpd.Enabled = False
    cmdfrmDel.Enabled = False
    cmdfrmFnd.Enabled = True
    txtusrn.SetFocus
End If
End If
End Sub

Private Sub cmdfrmUpd_Click()
Dim rslt As String
If txtusrn.Text <> "" And txtpass.Text <> "" Then
rslt = MsgBox("Do You Want To Update User Information?", vbCritical + vbYesNo,
"Update")
If rslt = vbYes Then
StartProgress
conn
sqlstr = "update users set username='" & txtusrn.Text & "', password='" & txtpass.Text &
"' where username='" & txtusrn.Text & "'"
database.Execute (sqlstr)
MsgBox ("User Information Updated!")
End If
Else
MsgBox ("Please Find User First!")
End If
cmdfrmUpd.Enabled = False
cmdfrmDel.Enabled = False
End Sub

```



```

Private Sub cmdfrmFnd_Click()
On Error Resume Next
Dim find As String
conn

If Text7.Text = "" Then
    MsgBox "Error: Please select an employee first!", vbCritical
    Exit Sub
End If

```

```

find = Text7.Text
sqlstr = "select * from users where username='" & find & "'"
Set rs = database.Execute(sqlstr)
If rs.EOF Then
    MsgBox ("The Wanted User Was NotFound!")
Else
    Dim i As Integer

    StartProgress

    txtusrn.Text = rs![UserName]
    txtpass.Text = rs![Password]

    txtusrn.SetFocus

    cmdfrmSav.Enabled = False
    cmdfrmUpd.Enabled = True
    cmdfrmDel.Enabled = True
End If
rs.Close
End Sub

```

```

Private Sub Command1_Click()
    List1.clear
    sqlstr = "select username from users"
    Set rs = database.Execute(sqlstr)

    DoEvents

    If Not rs.EOF Then
        While Not rs.EOF
            List1.AddItem rs.Fields![UserName]
            rs.MoveNext
            DoEvents
        Wend
    End If

```

```

    End If
End Sub

Private Sub StartProgress()

    Dim i As Integer
    Frame2.Visible = True
    X.Width = 1
    For i = 1 To 6975
        X.Width = X.Width + 1
        DoEvents
    Next

    Frame2.Visible = False

```

```
End Sub
```

```

Private Sub List1_Click()
    Text7.Text = List1.List(List1.ListIndex)
    Call cmdfrmFnd_Click
End Sub

```

The reports form

Option Explicit

```

Private Sub Command1_Click(Index As Integer)
    StartProgress
    Select Case Index
        Case Is = 0: DataReport1.Show vbModal
        Case Is = 1: DataReport2.Show vbModal
        Case Is = 2: DataReport3.Show vbModal
        Case Is = 3: DataReport4.Show vbModal
        Case Is = 4: DataReport5.Show vbModal
    End Select
End Sub

```

```

Private Sub Command2_Click()
    Unload Me
End Sub

```

```

Private Sub StartProgress()

    Dim i As Integer
    Frame1.Visible = True
    X.Width = 1

```

```

For i = 1 To 6255
    X.Width = X.Width + 1
    DoEvents
Next
    Frame1.Visible = False
    DoEvents
End Sub
The about form
Option Explicit
Private Sub Timer1_Timer()
    Image1.Top = Image1.Top - 10

    If Image1.Top <= -12360 Then
        Timer1.Enabled = False
    End If
End Sub

```