



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer

STOCK CONTROL SYSTEM

Using Visual Basic Programming

**Graduation Project
Com 400**

Student: Yasir Makki.

Supervisor: Mr. Umit Ilhan.

Nicosia - 2001

ACKNOWLEDGMENTS

First I want to thank Mr. Umit Ilhan to be my supervisor. Under his guidance, I successfully overcome many difficulties and learn a lot about Visual Basic Programming. In each discussion, he explained my questions patiently, and I felt my quick progress from his advises. I also want to thank Mr. Tayseer Alshanabla to be my advisor. He always helps me a lot either in my study or my life.

Special thanks to Mekki and Ahmed. With their kind help, and perform computational problems. Thanks to Faculty of Engineering for having such a good computational environment.

I also want to thank my friends in NEU, with them make my 3 years in NEU full fun.

Finally, I want to thank my family, especially my parents. Without their endless support and love for me, I would never achieve my current position. I wish my mother lives happily always, and my father in the heaven be proud of me.



ABSTRACT

The purpose of this thesis is to define the Stock Control system Database. As all of the data described in the database are derived from data captured by project.

It is useful to review the Data Element Dictionary. This document contains definitions and file description for each of the data elements to be collected as part of the project. In addition, a high level overview of the design of the system, and structure of the various records and fields to be submitted to hospital system are provided.

Purpose of this Thesis

This project defines the Stock Control System database files.

Audience for this Thesis

The intended audience for this project includes the follow:

- (1) Codes - any codes that are responsible for creating and maintaining the data elements and file description specified in this project.
- (2) Screens - those individuals who wish to view the data collected and processed as part of the Stock Control Project from a "summary" or "subtotaled" point of view. The database files are used by the company staff to verify that the underlying unitary data reported as part of the project are valid and consistent from term-to-term and year-to-year.

Scope of this thesis

The central role of this project is to provide information concerning the Stock Control System Database files. This document provides an overview of the database design as well as detailed specifications for each of the database files and data elements that comprise them.

Also, This document is both intended, to give a detailed explanation of the sources of the data used to create the files, and a detailed explanation as to how the files can be used. The source for all files is the data collected and processed as part of the Stock Control System project.

TABLE OF CONTENT

ACKNOWLEDGEMENT	i
ABSTRACT	ii
INTRODUCTION	iii
1. DATA MODEL	1
1.1. Conceptual Data Model	1
1.2. The Logical Data Model	2
1.3. Physical Data Model	2
1.4. Normalization	7
1.5. Populating Entity Attributes	8
1.6. Aggregating Entity Attributes	9
1.7. Relationship Types	14
2. LOGICAL DATABASE DESIGN	17
2.1. Database Design Methodologies	17
2.2. Entity –Attributes- Relation Versus Object- Relation Approaches	18
2.3. Rules to Guide Logical Database Design	20
2.4. Relationship Characteristics	22
3. CREATING A DATABASE IN MICROSOFT ACCESS	25
4. VISUAL BASIC	26
4.1. The major types of errors	30
4.2. Managing Data within the tables	31
4.3. The three important challenges that face every Database Programmer writing multi-user application	33
4.4. ODBC Data Access via the ODBC API	35
5. STOCK CONTROL SYSTEM	38
5.1. Tables and Relations	38
5.2. Screens and Layouts	39
5.3. Code	47
CONCLUSION	72
REFERENCES	73

INTRODUCTION

The success of a database is completely dependent on the logical database design. Even if we buy expensive and fast hardware and software, the quality of the database design will dictate whether a project will succeed. In a way, it is the Achilles heel of a project.

A good database design does the following:

1. Provides minimum search time when locating specific record.
2. Stores data in the most efficient manner possible to keep the database from growing too large.
3. Makes data updates as easy as possible.
4. Is flexible enough to allow inclusion of new functions required of the program.

The database design process can be divided into six steps:

1. Requirement analysis.
2. Conceptual database design.
3. Logical database design.
4. Schema refinements.
5. Physical database design.
6. Security database design.

What is logical database design?

It is the phase in the system development life cycle concerned with constructing tables and their columns. During this phase decisions are made about which piece of data should be stored and how those pieces should be arranged logically in tables. This phase precedes physical database design where the emphasis is on how the data is really stored on disk. Physical database design deals with issues such as storage and performance.

This one-day workshop discusses all the aspects of logical database design. Structured techniques for developing a logical design are discussed. Additionally,

many guidelines, tips, and tricks are given. And logical database design is looked at from a transactional and from a data warehouse environment. Because the use of data warehouse is different, different rules apply.

Two opposing techniques exist to perform logical database design. The bottom-up approach, which is based on normalization, is the oldest and most well known one. One start with placing all columns in one wide table and then these tables are decomposed into more well structured tables. The decomposition is based on rules called the normal forms. The second technique, the top-down approach, uses as a starting point information models where semantic and object-oriented concepts, such as subtypes and aggregates, are used. These concepts are translated into tables and columns using an algorithm. Both approaches have their advantages and disadvantages en is, therefore, discussed thoroughly.

Although some of the rules described in thirds workshop do stem from relational theory, such as the normal forms, the emphasis will be on practical issues. The workshop is a culmination of many years of experience of designing large operational databases and data warehouses.

The aim of logical design is to construct a logical schema that correctly and efficiently represents all of the information described by an entity relationship schema produced during the conceptual design phase. This is not just a simple translation from one model to another for two reasons, first, there not a close correspondence between the models involved because not all the constructs of the entity-relationship model can be translated naturally into the relational model. For example, while an entity can easily be represented by a relation. There are various options for the generalizations. Secondly, the aim of conceptual design is to represent the data accurately and naturally from a high-level, computer-independent point of view. Logical design is instead the basis for the actual implementation of the application, and must take into account, as far as possible, the performance of the final product. The schema must there for be destructed in such away as to make the execution of the projected operations as efficient as possible. In sum, we must plan a task that is not only a translation (from the conceptual model to the logical) but also reorganization. Since the organization can for the most part be dealt with independently of the logical model, it is helpful to divide the logical design into two steps:

- Restricting of the Entity-Relationship schema, which is independent of the chosen logical model and is based on criteria for the optimization of the schema and the simplification of the following step.
- Translation into the logical model, which refers to a specific logical model (In our case, the relational model) and can include a further optimization, based on the features of the logical model itself.

The input for the first step is the conceptual schema produced in the preceding phase and the estimated database load. In terms of the amount of data, and the operational requirements. The result obtained is a restricted Entity-Relationship schema.

DATA MODELS

Levels of abstraction usually categorize data models:

- Conceptual
- Logical
- Physical

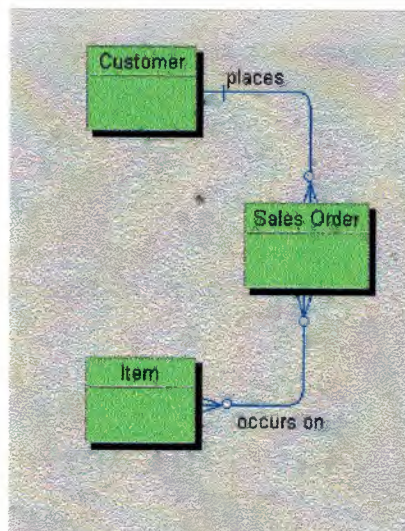
These have no agreed formal definitions. Professional data modelers understand the approximate scope of each.

Conceptual Data Model:

A conceptual data model shows data through business eyes.

It suppresses technical details to emphasize:

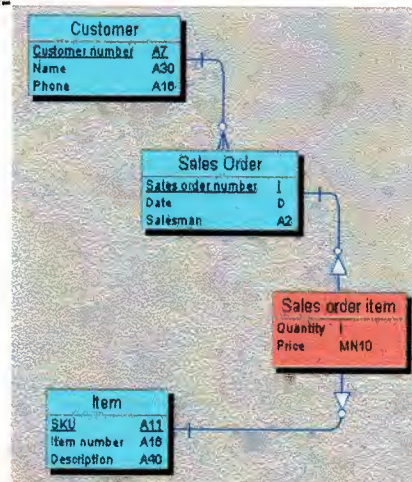
- All entities, which have business meaning.
- Important relationships (including many-to-many).
- A few significant attributes in the entities.
- A few identifiers or candidate keys.



The Logical Data Model

- Is a generic relational schema (in at least 1NF) which -

- Replaces many-to-many relationships with associative entities
- Defines a full population of entity attributes
- May Use non-physical entities for domains and subtypes
- Establishes entity identifiers
- Has *no* specifics for any RDBMS or configuration



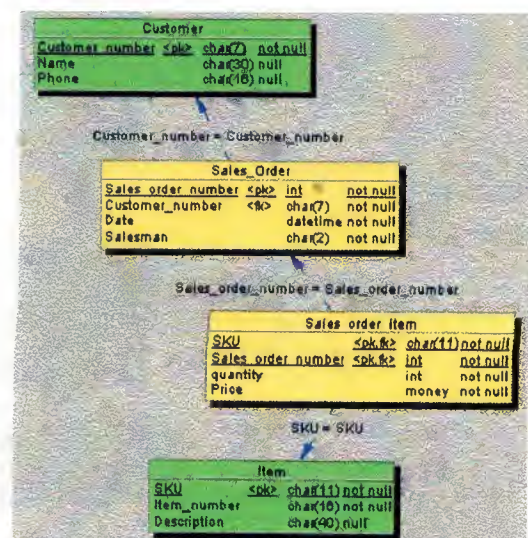
Propagation of foreign keys may be explicit or implied in a logical data model. As long as the resulting physical schema includes the necessary foreign key columns and joins, the representation of foreign keys in the logical model is a matter of convenience and taste.

Replacing many-to-many relationships with associative entities is necessary to model 1st normal form, support internal attributes and secondary relationships, and enable alternate identifiers.

Physical Data Model

A physical data model is a database design for:

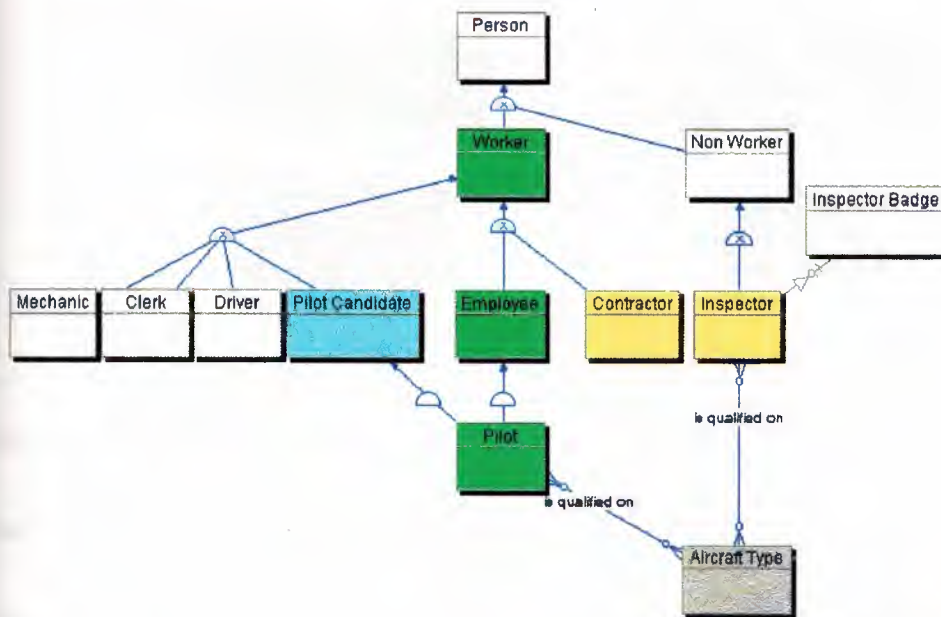
- One DBMS product
- One site configuration



A physical data model may include:

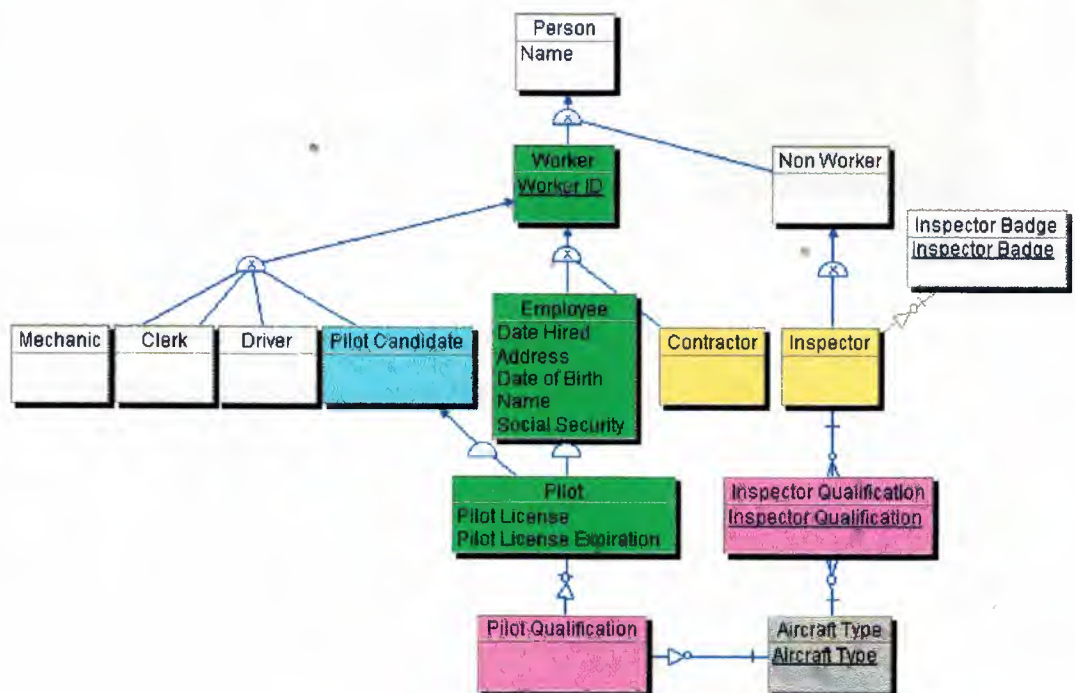
- Referential integrity
- Indexes
- Views
- Alternate keys and other constraints
- Table spaces and physical storage objects

Conceptual Data Model - An Example:

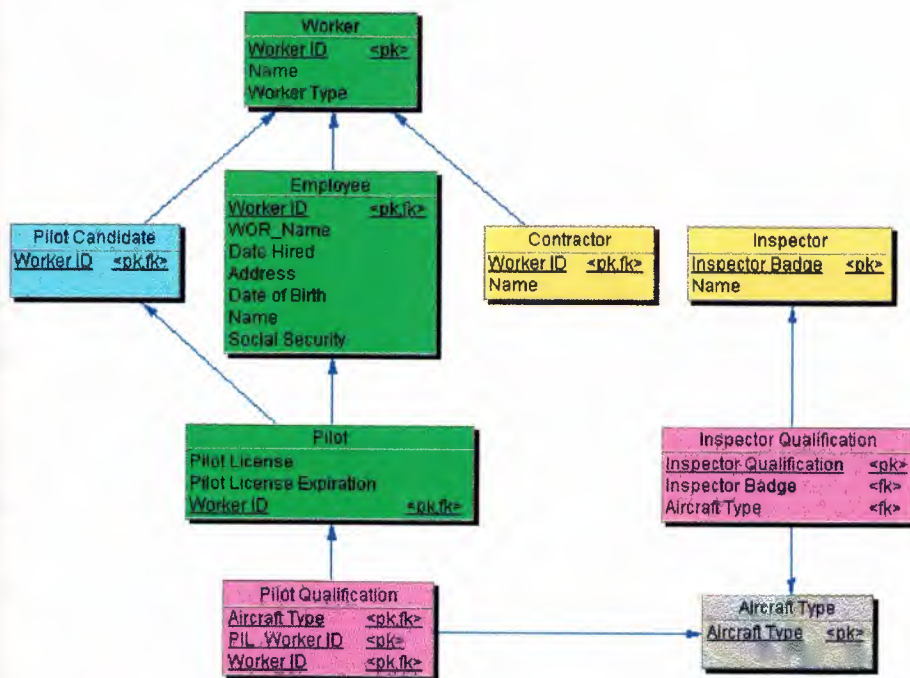


Logical Data
Model - Same

Example:



Physical Data Model - Same Example:



What is an Entity?

Person, place, or thing? (Too specific)

Any thing in which the business has an interest? (Too vague)

A synonym for a relational table? (Misses the point)

An information container in 1st normal form which:

- Records a fixed set of attributes
- Holds 0 to n occurrences
- Is a relational abstraction of some real-world concept
- May or may not map to a physical table in the database



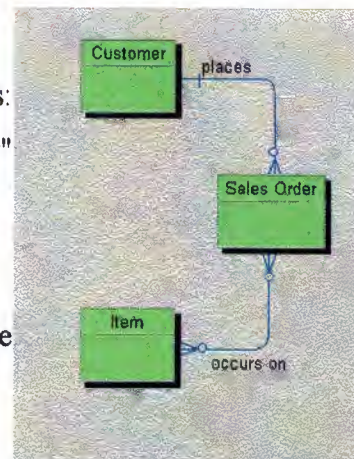
What is a Relationship?

A connection, association, or rule among entities:

"Customer places Sales Order"

"Item occurs on Sales Order"

In a conceptual model, it is sufficient to state or draw the relationship.



A logical model defines specific means of joining two entities via implied or expressed foreign keys.

Relationships can be classified into a few relationship types.

Foreign Keys (FK):

A foreign key is a function, not a fact!

A foreign key is the result of relationship and identity.

If the relationship changes, so does the foreign key.

The "child" entity gets foreign key attributes to match the identifier of its "parent" entity.

State		
state name	state code	row id
CK	CK	CK
Alabama	AL	1
California	CA	2
New York	NY	3
Tennessee	TN	4

Disaster	
row id	disaster year
FK	
2	1979
2	1982
2	1995

If the parent identifier changes, so does the child's foreign key.

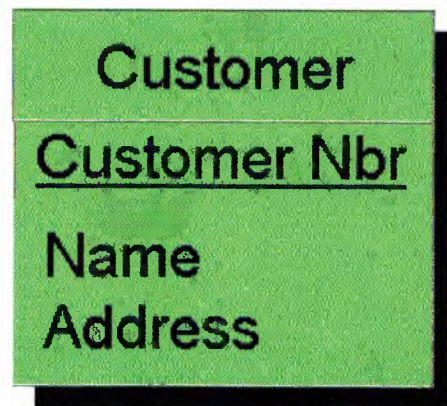
What is an Entity Identifier?

The identifier of an entity is some set of attributes whose combined value is unique for all instances of that entity.

Thus an entity's identifier is one of its (possibly several) candidate keys.

If an entity has more than one candidate key, the choice of one to be the identifier is an arbitrary convenience for RDBMS operation.

While an entity identifier is not absolutely mandatory, it is hard to think of a useful entity without one.



Evolving the Logical Model:

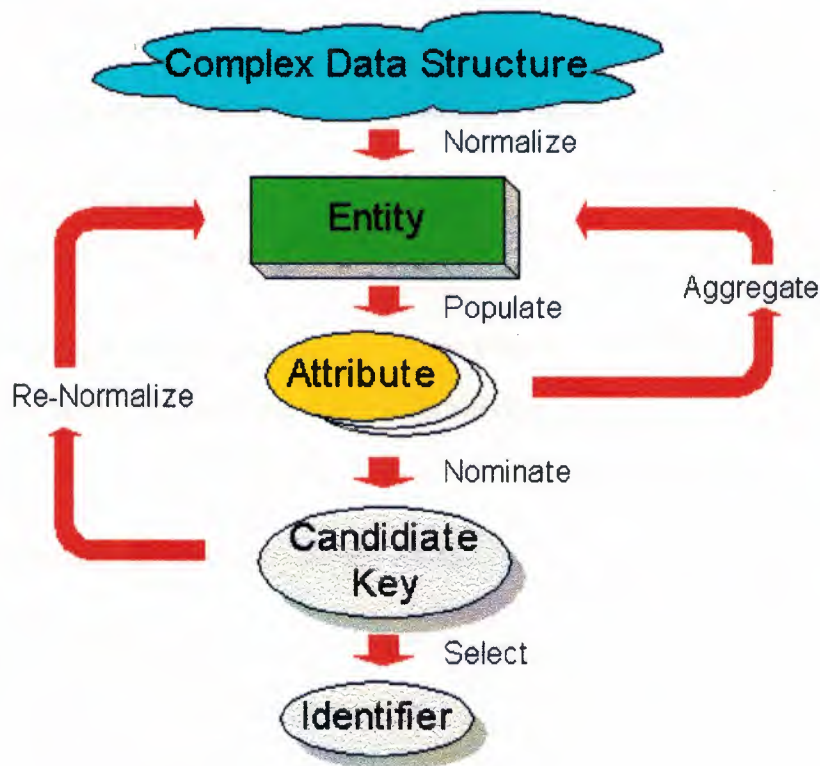
Normalize structures

Populate attributes

Aggregate data items into new entities

Nominate candidate keys

Re-Normalize on the new candidate keys



Normalization:

Normal forms are the property that we can use to evaluate the quality of relational database. We will see that when a relation does not satisfy a normal form, then it presents redundancies and produces undesirable behaviour during update operations. This principle can be used to carry out quality analysis on relational databases and so constitutes a useful tool for database design. For the schemas that do not satisfy the normal form, we can provide a procedure called Normalization. Normalization allows the non-normalized schemas to be transformed in to new schemas for which the satisfaction of a normal form is guaranteed.

Normalization theory constitutes a useful verification tool, which indicates amendments, it has been developed in the context of the relational model and for this reason, it provides analysis and design techniques for the out come of logical design.

Normalization can also be used on the Entity-Relationship schemas and during the quality analysis of the conceptual design.

Normalizing to a Logical Model:

Every raindrop, every snowflake, every hailstone has a single speck of dust at the core.

Every logical entity has a single idea at its core.

The essence of normalization is one entity = one idea:

- A customer is a person or organization that buys from us.
- A service order holds one customer request for service.

Examine complex data structures for hidden entities in:

- Nouns - tangible or intangible
- Adjectives whose value is one of a known list ...
female | male; green | yellow | red; 6' | 8' | 10'
- Embedded ideas, which can exist on their own

Populating Attributes:

For each entity, ask, "What properties does this thing have - even if nothing else exists around it?"

- A person has age - even the last person on earth.
- A building has height - even if it is abandoned.
- A song has a key, even if it is unsung.

Ask of each property, "Does this entity have only one of these?"

- A person is of one age.
- A building is of one height.
- A song may be written in one key but sung in another!

An attribute is a property of an entity, which depends solely on its entity - nothing else - and can have only one value at a time

Domains:

6 feet 1/4 inches + 51.6 years =?

A domain (in *relational* usage) is a set of values and operations that may be used to populate and operate on one or more columns.

The values may be specified by list or formula.

While is not yet any theoretical or practical way to limit the operations applied to a domain, this example shows the need.

Aggregating Entity Attributes:

Sometimes you can reveal entities by looking in the data dictionary for homeless attributes:

- "To whom does atomic weight belong?

And what about the year in which an element was discovered?"

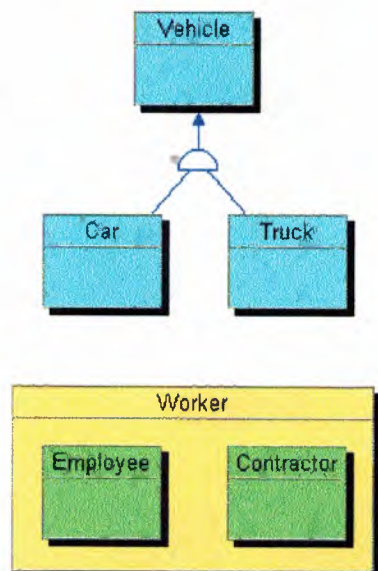
- Aggregate atomic weight and year into anew entity called Atomic Element.

Crosscheck data elements captured in the data dictionary from data flow diagrams, use cases, or other analysis.

All data structures and data elements discovered in analysis must be accounted for in the logical model.

Looking for Hierarchies

Entities often occur in hierarchies - family trees related by inheritance.



This is sub-typing or specialization and generalization - the same as building OO class structures.

Each child entity inherits all attributes and relationships from its We define properties at their highest level in the hierarchy to avoid redundancy.

In a conceptual model, we ignore how inheritance operates.

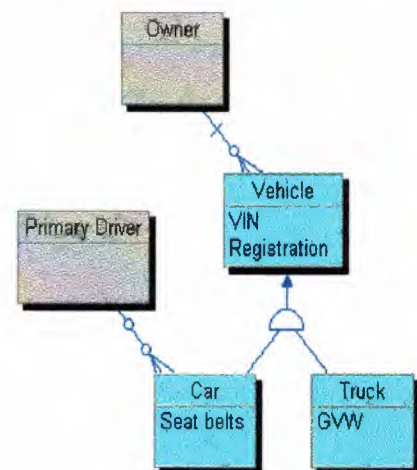
Later we want to specify how super-and sub-types map from the logical model to physical structures parent.

Generalization:

Hierarchies let us locate attributes and relationships at the appropriate level.

All vehicles have:

- VIN and Registration
- Owner



The attributes and relationship are *generalized* To all vehicles.

Specialization:

- Only cars have primary drivers and seat belts.
- Only trucks have gross vehicle weight.

These attributes and relationships are *specialized* to the child level.

Specialization of Relationships:

Optional parent relationships usually hide a need for specialization. Ask your self:

- Is the relationship *sometimes* true for *each* instance?



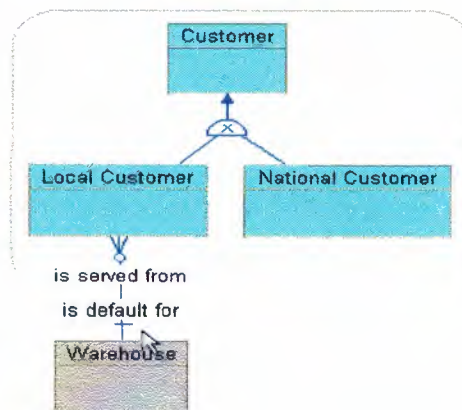
Then it is correctly modeled as optional.

- Is the relationship *always* true for *some* instances?

Then it requires some customers must have a default warehouse.

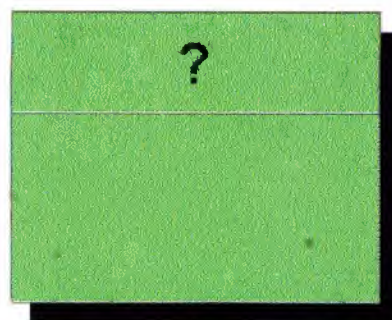
For others it does not exist (in this model).

By splitting the Customer entity into two sub-types, we can model the relationship to Warehouse precisely.



Populating a Conceptual Data Model:

- Diagram entities
- Diagram relationships
- Look for hierarchies:
 - Sub-types / super-types
 - Specialization /generalization
 - Class hierarchies



Diagramming Entities:

The entity symbol is a rectangle with the name at the top.

The entity name must be descriptive and meaningful. An unambiguous text definition is important.

The entity graphic symbol may differ slightly by CASE tool or author but the shape is unimportant.

The entity name is how people will refer to the entity. In the conceptual model an entity name should not be limited by RDBMS product limits - this is a generic name for the business. When physical DBMS object names are assigned, be sure to take into account the naming limits and reserved words of your target.

For example, what is a customer?

- Someone who has purchased?
- Any organization or person who may purchase?

Will this entity definition be clear two years later to a new team?

An Entity represents

- A tangible thing, a real-world event, or any intangible concept: "Product", "Sales visit", "Customer class discount"
- A class of things, not any one instance. "Person" has instances of "Tom" and "Simone".

Entities are *not* –

- Independent or Dependent. Those terms apply only to the identification choice you make.
- Fundamental, Attributive, or Associative. Classifications have meaning only in a model context of entities *and* relationships.

Diagramming Relationships:

The relationship symbol is a line between two entities. Define a relationship with:

- Predicate statements in one or both directions
- Unambiguous text description
- (A name is not important)
- cordiality symbols at each end

Will the relationship be clear two years later to a new team?

Relationships are -

- Unambiguous, immutable expressions of business rules.
- Binary or unary in IE, SSADM, IDEF1X and OO methods.
- Logical objects. Relationships can be reattached, with their properties intact, to different entities.

Relationships are *not* –

- Identifying or Non-Identifying. Those apply only to entity identification.
- Information containers. If you sense a need for information about a "relationship" then it is an entity!
- DBMS objects. Relationships only define joins between entities.

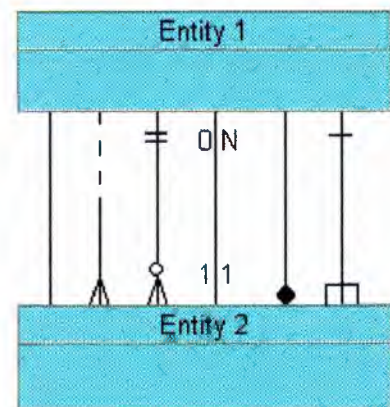
Relationship Notations:

There are many notation styles for relationships.

There are no standards for relationship style.

Different styles are read in different directions

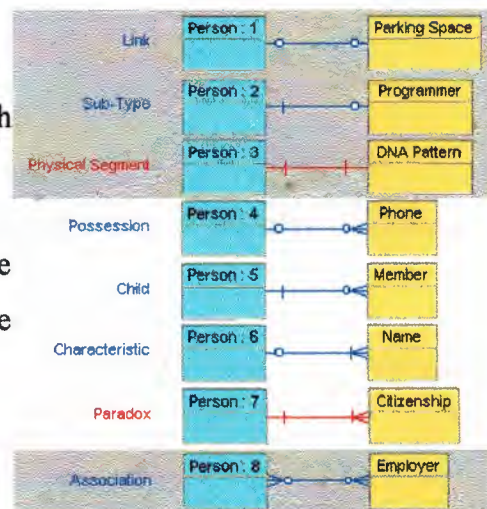
But they all express the same information!



Relationship Cardinality:

Cardinality specifies the number of *instances* which may be involved in each entity of a relationship.

Most methods show the *Boolean abstract*, not the absolute number, because this determines the relationship type...



Relationship Types:

Relationships are grouped by their cardinality:

- One-to-Many is the *only* relational form.
- >99% of logical model

Relationship Types:

One-to-One is a special case of One-to-Many;
<1% of a logical model

		minimum of	maximum of		
Customer	places	0	n	Sales Order	
Sales Order	is placed by	1	1	Customer	

		at least 1	possibly n		
Customer	places	No	Yes	Sales Order	
Sales Order	is placed by	Yes	No	Customer	

Populating Attributes:

For each entity, ask "What properties does this thing have - even if nothing else exists around it?"

- A person has age - even the last person on earth.
- A building has height - even if it is abandoned.
- A song has a key, even if it is unsung.

Ask of each property, "Does this entity have only one of these?"

- A person is of one age.
- A building is of one height.
- A song may be written in one key but sung in another!

An attribute is a property of an entity which depends solely on its entity - nothing else - and can have only one value at a time.

What are Candidate Keys?

A candidate key is any set of one table's columns whose combined value is unique throughout that table.

Code	Name	Admission
NV	Nevada	36
TN	Tennessee	16
MO	Missouri	24
PA	Pennsylvania	1
HI	Hawaii	50
IN	Indiana	19

- In the U.S. each state has a unique code - one candidate key.
- Each state name is also unique - another candidate key.
- And so is the order of admission to the union.

Since both code and name are unique, code and name together are also unique.

That's another candidate key - seven with all the combinations.

Why are Candidate Keys?

- As a *candidate* for selection as the one identifier or *primary* key. A candidate key usually holds the core idea inside an entity:
- This state table is *about* states, which are known by their names.

A candidate key *always* expresses a business rule of uniqueness:

- Every state has a unique state code for mailing.

A table or entity with no candidate key is probably not normalized. and almost certainly not useful in an information system.

Testing Candidate Keys

A candidate key is unique. Is that enough?

Social security number is unique. Were you born with one?

A candidate key's value must exist. It cannot be null.

Your driver's license number is unique. Can it change?

The value of a candidate key must be stable. It's value cannot change outside the control of the system.

The value of a candidate key is unique, extant, and stable.

Re-Normalize on the Candidate Key:

After at least one candidate key has been noted, every attribute and relationship of the entity must be tested -

- Does this property depend solely and completely on the candidate key?

If not, move the property (normalize it) to the entity where it depends solely and completely on the candidate key.

Repeat these steps -

- Normalization
- Population
- Candidate keys
- Re-normalization

- until every object in the data dictionary is consumed and every entity is normalized to at least 1NF.

1st Normal Form:

For a given table, every row must have the same columns. To remove embedded lists:

Separate children from parents, or -

Provide a series of columns to hold all of any parent's children.

But the latter method has drawbacks:

Multiple columns hold the children

More children require disruptive database redesign

Ugly nulls where any parent has less than the maximum.

Not normalized			
Mother	1st child	2nd child	3rd child
Sarah	Sally	Mark	Ashley
Mother	1st child		
Oblied	Raoul		

1st normal form		
Mother	Mother	Child
Sarah	Sarah	Sally
Oblied	Sarah	Mark
	Sarah	Ashley
	Oblied	Raoul

Alternate 1st normal form			
Mother	1st child	2nd child	3rd child
Sarah	Sally	Mark	Ashley
Oblied	Raoul	(nul)	(nul)

LOGICAL DATABASE DESIGN

Logical database design uses several rules or concepts which are reasonably well understood and accepted. Disagreement arises in formulating a particular methodology—the place to start and the sequence of steps to follow in applying those rules. After a brief discussion of database design methodologies, this section presents several concepts, principles, or rules which are generally recognized and applied regardless of the particular methodology used.

Database Design Methodologies

A *database design methodology* specifies a sequence of steps to follow in developing a “good” database design—one that meets user needs for information and that satisfies performance constraints. Each step consists of the application of a set of techniques or rules that may be formalized to varying degrees and embodied in software tools. A methodology should be (1) usable in a wide variety of design situations and (2) reproducible in different designers. The second objective implies that the methodology be teachable, and that those trained in applying the methodology would arrive at the same end result. This is not evident in the present state of the art. Logical database design remains very much an art.

Theory and Pry [1982] outline a database design methodology consisting of four steps;

1. *User Information Requirements*—involving the users in analyzing organizational needs, setting the scope of interest, investigating what people do (organizational tasks; usage patterns), and determining the data elements needed to perform those tasks,
2. *Conceptual Design*—developing a high-level diagrammatic representation of a logical data structure; a structure which includes object domains, events, entities, attributes, and relationships: a structure which seeks to model the users’ world.
3. *Implementation Design*—refining the conceptual design, checking for satisfaction of user needs and for consistency, and adjusting it to meet processing and performance constraints in a particular computer and DBMS environment.
4. *Physical Design*—developing record storage designs, clustering, and establishing access paths.

The techniques and rules in the steps of a methodology are applied iteratively in the process of unfolding, growing, and refining a database design. For a starting point, some suggest applying the methodology to individual user application areas or local views. Different user views may contain related complementary parts or overlapping pans. Multiple local views are then consolidated into a global logical structure or conceptual schema. The process of consolidation seeks to resolve inconsistencies, and to integrate related pieces. Even within a local view, there may be redundant, overlapping, and inconsistent pads. The rules of a methodology are intended to assist the designer in asking the right questions and representing the data structure in a coherent and consistent way regardless of the scope of the design activity, and regardless of whether it begins with individual local views or a global perspective. The product of the design activity will grow as it unfolds over the area of interest; and it will be refined as the rules are applied to focus attention on particular aspects of an infinitely complex reality and to resolve ambiguities and inconsistencies in the developing database structure.

Entity-Attribute-Relation versus Object-Relation Approaches

Perhaps the most significant difference among methodologies or approaches to logical database design is found in the point at which data items are clustered or grouped into records. The top division of the taxonomy of data structures presented in Chapter 4 reflects this division. The number of basic constructs distinguishes the two approaches: Those which presume an early clustering are often called “Entity Attribute-Relation” or “E-A-R” approaches; the alternative is called the “Object Relation” or “O-R” approach.

Historically data processing has always worked with records. Programming languages such as COBOL and FORTRAN cluster data items into records. The formation of records as a contiguous set of data items is necessary for efficient data processing. A record is the unit of access for getting data in and out of programs. Data is moved to and from secondary storage in blocks of records. Earlier data processing systems forced a “unit record” view, that is, all data for an application had to reside in a single sequence of records (this reflected the technology of the day, which used what was called “unit record equipment”). Even today, with DBMSs supporting a multfile data structure, data exists in the form of records in most organizations. Users are very familiar and comfortable with a record-oriented view of their data. Most designers today use an E-A-R approach to logical database design.

The major problem with the E-A-R approach to logical database design is that it allows the relationships among data items within a record to be hidden. It does not force the designer to explicitly consider and define inter record structures. This accounts for the recent emphasis in the literature on record decomposition and normalization based on an analysis of functional and multivalued dependencies. These techniques are all aimed at uncovering and making explicit the relationships among individual data items within records.

The end result of repeatedly applying record decomposition rules is irreducible varies—at which point there exists at *most one* non-identifier data item within each record. By then the designer will have considered all inter item relationships.

At the implementation or physical level, data items must be clustered into records for efficient data processing. Even at the logical level, it is still relevant and useful to think of attributes which cluster around and describe entities, whether the attribute items are considered as part of entity records or as individual object domains. It is relatively unimportant whether the design activity starts with records which are decomposed to analyze inter item relationships, or starts with object domains which are clustered to form records. In practice a designer will do both. It is important that certain rules and concepts be applied in the design process. Early formation of records is dangerous only if it inhibits the designer from properly analyzing intrarecord relationships among data items, and from considering alternative groupings of items into records -Ideally, the formation of records should be part of the implementation phase of database design since it is done primarily for system convenience and processing efficiency. In fact, it is desirable to have software tools to perform the clustering, leaving the designer to concentrate on defining the individual data objects, relationships, and performance factors and constraints.

In a strict application of the object-relation approach to logical design, all object domains are treated equally. In the E-A-R approach, attention is initially focused on entities, then on the attributes of those entities, which may turn out to be other entities. In fact, the distinction between attributes and entities is often confusing and arbitrary. Again, regardless of the approach taken, it is important for the designer to focus attention on the more important parts of the users' world being modeled in the data structure. This is automatically done in the E-A-R approach but can also be done in the O-R approach. The designer needs a high level of abstraction when developing a data structure and may start out by representing the main entities as boxes labeled with a name only .

Even though there is no widespread acceptance of any particular design methodology, there is general recognition of many underlying rules and concepts used in logical database design. They relate to conceptual design and part of implementation design.

A good database designer will generally know these rules and apply them, often intuitively, wherever they are relevant in the process of developing, checking, and refining a database design.

The following rules are presented here in a reasonably logical order, but there is no implication that they should be applied in any strict sequence. There is also no implication that these rules are sufficient or complete for the database design task. While progress is being made in formalizing the principles and process of database design, it still depends heavily on human intelligence and experience. Even experienced designers can arrive at different database designs, which purport to model the same user environment.

ENTITY: *Clearly identify the entities to be represented in the database.*

An entity is any object (person, place, thing), event, or abstract concept within the scope of interest about which data is collected. An entity is the object of decisions and actions within an organization. Entities are the pivotal elements in a data structure and must be well defined. Start out by focusing on the main entities, gradually expanding the logical data structure view to include related entities. When looking at an existing database, clearly define the primary entity, which is described in each file (record type).

INCLUSION: *Specify the criteria for including (or excluding) entity instances from a defined class of entities.*

The ENTITY rule names a class of entities and the INCLUSION rule specifies the conditions for membership in that class. For example, does the EMPLOYEE entity class include managers, job applicants, rejected job applicants, those fired or laid off, those who quit, or employees on definite or indefinite leave? Consideration of these other "EMPLOYEES" may suggest broadening the name of the entity class, or it may give rise to another entity class. Narrowing (sub setting) or broadening the definition of the entity class represents movement along the generalization hierarchy.

ATTRIBUTE: *Identify the attributes of each entity.*

Initially focus on the major attributes of each entity. Some will be clear and obvious, some will seem to be artificial, and some may also relate to other entities. Include all attributes, which assist in understanding the nature of the entity being described. Include at least one attribute from each set of similar attributes.

ATTRIBUTE CHARACTERISTICS: *Define the characteristics of each attribute.*

Clearly define the characteristics of each attribute. Initially focus on name, type, (size), existence, uniqueness, and some indication of the nature of the value set. When describing an existing database, specify any encoding of data item values. Description of other characteristics can be deferred until later in the database design process. Eventually plan to describe one attribute per page in the final database documentation.

DERIVED ATTRIBUTE: identify and define derived attributes.

The values of an attribute may be derived from the values of other attributes in the database. Specify the derivation rule, which may be an expression for a derived item or a statistical calculation across instances of an entity type or a repeating group.

IDENTIFIER: Designate the attribute(s), which uniquely identify entity instances in each entity class.

An entity identifier may be a single attribute (EMPNO) or multiple attributes (UNIT and JOECODE for POSITION). There may be multiple identifiers for the same entity (EMPNO and SOCIAL SECURITY NUMBER). Indicate if the identifier is not guaranteed to be unique. The identifier can be a good clue to understanding the nature of the entity described in an existing file.

RELATIONSHIP: identify the primary relationships between entities.

RELATIONSHIP CHARACTERISTICS:

Define the characteristics of interentity relationships, particularly exclusivity and exhaustibility (or dependency).

Exclusivity refers to whether instances of one entity type can be related to at most one or more than one instance of another entity type. Since it is defined in both directions there are four possibilities: 1:1, 1:Many, Many: 1, Many: Many. Exhaust stability (also called dependency or personality) specifies whether or not an instance of one entity type must be related to an instance of another entity type. Indicate if there is some condition on the dependency of a relationship. Also indicate if there is some minimum or maximum cardinality on the "many" side of a relationship. (See section 6.3.3 for more detail on these characteristics.)

FOREIGN IDENTIFIER: indicate the basis for each relationship by including, as an attribute in one entity type, the identifier from each related entity type.'

Every relationship is based upon common domain(s) in the related entity records. At the logical level, it is necessary to include the identifier of a related entity as a foreign identifier. In the storage structure, if the common domain is not explicitly stored in a related record, then some form of physical pointer is necessary to represent the relationship.

DERIVED RELATIONSHIPS: Suppress derived relationships.

The logical database design should not include relationships, which can be derived from other relationships. For example, it is reasonable to think of organizational units as possessing a pool of skills. Furthermore, such information can be retrieved from the database. However, such a relationship should not be defined since it is derived from the ORGANIZATION-EMPLOYEE relationship and the EMPLOYEE-SKILL relationship. An organizational unit only possesses skills because it has employees who possess skills.

REPEATING GROUP: *isolate any multivalued data item or repeating group of data items within a record.*

This rule ensures that a record only contains atomic (single-valued) data items, thus allowing only flat files. This is also called first normal form. The real importance of this rule

is to force the designer to explicitly recognize a “something-to-many” relationship and possibly a new entity type. If a repeating group of data items becomes a new entity record type, the identifier of its parent record must propagate down into the new record. If the relationship was actually many-to-many, the propagated identifier becomes part of the identifier of the new record; if the relationship was one-to-many, the propagated identifier becomes a foreign identifier in the new record (but not part of the identifier). Multi-valued data items or nested repeating groups of data items may be included in the storage structure of a record (as they are in a hierarchical data structure).

PARTIAL DEPENDENCY: *Each attribute must be dependent upon the whole record (entity) identifier.*

An attribute that is dependent upon only part of the identifier should be removed from the record, and placed in a record where that part of the identifier is the whole identifier. Suppose we had a record with the following data items: EMPNO, SKILLCODE, SKILL DESCRIPTION, and PROFICIENCY. The identifier would have to be the first two data items jointly since PROFICIENCY relates to both of them together. However, DESCRIPTION relates only to the SKILLCODE and, therefore, should not be in this record. A record with no partial dependencies is said to be in second normal form.

TRANSITIVE DEPENDENCY: *Each attribute within a record must be directly dependent upon the entity identifier.*

Any attribute, which is not directly dependent upon the record identifier, should be removed from the record, and related directly to the object on which it is functionally dependent. For example, if the EMPLOYEE record contained UNIT and BOSS, and the employee was moved to another organizational unit, it would not be sufficient to update the employee's UNIT—the BOSS data item would also have to be changed. The update anomaly results because BOSS is directly dependent upon UNIT and not EMPNO. BOSS does not belong in the EMPLOYEE record *even if processing is faster and easier*; it belongs in the ORGANIZATIONAL UNIT record. A record with no partial or transitive dependencies is said to be in third normal form. Restated: An attribute should be dependent upon the identifier, the whole identifier, and nothing but the identifier.

Application of the previous three rules to arrive at third normal form requires an examination of every attribute in a record. A record not in third normal form produces undesirable update anomalies. To identify these anomalies, the designer can ask: If a given attribute is updated, what other attributes must change, or if another attribute is updated, what effect will it have on the given attribute?

NAMING: *Assign names to entities, attributes, and relationships using a consistent, well-defined naming convention.*

When describing an existing database, watch for naming inconsistencies— different names for the same object, or the same name used to refer to different objects.

STORAGE & ACCESS: *Suppress any consideration of physical storage structures and access mechanisms in describing the logical structure of the data.*

This includes any stored ordering on the records in a file, and whether or not a data item is indexed. Do not be concerned with questions of how to find or access a particular record in a file, perhaps along a relationship. Remember, all relationships are inherently bi-directional.

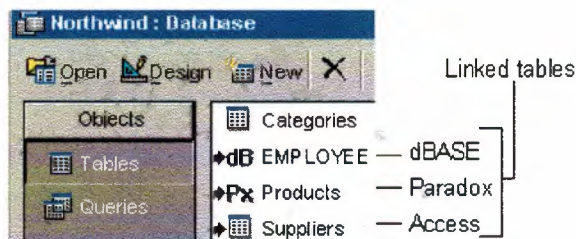
Creating a database in Microsoft Access

Microsoft Access provides two methods to create a database. You can create a blank database and then add the tables, forms, reports, and other objects later — this is the most flexible method, but it requires you to define each database element separately. Or you can use a Database Wizard to create in one operation the required tables, forms, and reports for the type of database you choose — this is the easiest way to start creating your database. Either way, you can modify and extend your database at any time after it has been created.

Linking data

In an Access database, linking data enables you to read and in most cases, update data in the external data source without importing. The external data source's format is not altered so that you can continue to use the file with the program that originally created it, but you can add, delete, or edit its data by using Microsoft Access as well. You can link a table only in an Access database, not an Access project.

Microsoft Access uses different icons to represent linked tables and tables that are stored in the current database. If you delete the icon for a linked table, you delete the link to the table, not the external table itself.



VISUAL BASIC

First thing to know about Visual Basic in the relational database model, how to use the Visual Basic database objects to access and update existing databases, and how to use the Visdata program to create and maintain databases. We also take a look at the design and code data entry forms (including use of the Visual Basic bound data controls), and how to create input validation routines at the keystroke, field, and form levels. Lastly, We see how to use the Visual Basic Crystal Reports Pro report writer to design simple reports, and how to use the Crystal Reports control to run those reports from within your Visual Basic programs.

Taking in mind that Microsoft Access is the best way to make the relations and build up the main structure that the Visual Basic depend upon.

There are certain requirements for the use of visual Basic that affects some of the main questions to be asked :

How to use the Structured Query Language (SQL) to extract data from existing databases.

What the Microsoft Jet engine is, and how you can use Visual Basic code to create and maintain data access objects.

How to create data entry forms with Visual Basic code.

How to use the Microsoft graph control to create graphs and charts of your data.

How to use data-bound list boxes, data-bound combo boxes, and data-bound grids to create advanced data entry forms.

How to make applications more solid with error trapping.

We are going to start working with the data entry:

Use the data control to bind a form to a database and data table by setting the DatabaseName and RecordSource properties.

Use the Text box bound input control to bind an input box on the form to a data table and data field by setting the DataSource and DataField properties

Combine standard command buttons and the AddNew and Delete methods to provide Add and Delete record functionality to a data entry form

A relational database is a collection of related data.

The three key building blocks of relational databases are data fields, data records, and data tables.

The two types of database relationships are one-to-one (which uses qualifier fields) and one-to-many (which uses pointer fields).

There are two types of key (or index) fields: primary and foreign, there are 1basic data field types recognized by Visual Basic 5.

Working with SQL.

Create basic SQL statements that select data from existing tables. the most fundamental form of the SQL statement is the `SELECT_FROM` clause. This clause is used to select one or more columns from a table and display the results of in a result set, or view.

Optional clauses that you can add to the `SELECT_FROM` clause:

The `WHERE` clause: Used to limit the rows in the result set using logical comparisons (for example, `WHERE Table.Name = "SMITH"`) and to link two tables in a single, nonupdatable, view (for example, `WHERE Table1.Name = Table2.Name`).

The `ORDER BY` clause: Used to control the order in which the result set is displayed (for example, `ORDER BY Name ASC`).

The `GROUP BY` clause: Used to create a subtotal result set based on a break column (for example, `GROUP BY Name`).

The `HAVING` clause: Used only with the `GROUP BY` clause, the `HAVING` clause acts as a `WHERE` clause for the `GROUP BY` subtotal clause (for example, `GROUP BY Name HAVING SUM(SalesTotal)>1000`).

The `INNER JOIN` clause: Used to join two tables together into a single, updatable result set.

The `INNER JOIN` returns rows that have a corresponding match in both tables.

The `LEFT JOIN` and `RIGHT JOIN`: Used to join two tables into a single, updatable result set.

The `LEFT JOIN` includes all records from the first (left-hand) table and all rows from the second table that have a corresponding match. The `RIGHT JOIN` works in reverse.

The `UNION` clause: Used to combine two or more complete SQL queries into a single result set (for example, `SELECT * FROM Table1 UNION SELECT * FROM Table2`).

The `TRANSFORM_PIVOT` clause: Used to create a cross-tab query as a result set (for example, `TRANSFORM SUM(MonthlySales) FROM SalesTable GROUP BY Product PIVOT Month`).

Additional SQL keywords that you can use to control the contents of the result set:

`BETWEEN_AND`

`DISTINCT` and `DISTINCTROW`

`AS TOP n` and `TOP n PERCENT`

`AVG`, `COUNT`, `MAX`, `MIN`, and `SUM` Microsof Jet data-access.

The features and functions of Visual Basic Microsoft Jet data-access objects and ODBCDirect access objects. These objects are used within Visual Basic code to create and maintain workspaces, databases, tables, fields, indexes, queries, and relations. the properties, methods, and collections of each object. Use Visual Basic code to inspect the values in the properties, and how to use the methods to perform basic database operations.

Using Visual Basic Code.

Writing data entry forms using Visual Basic code.: record search routines, the creation of a procedure library to handle all data entry processes, and creating a working data entry form for the CompanyMaster project.

Perform single-record searches using the three search methods:

Move for browsing the dataset

Seek for indexed table objects

Find for non-table objects (Dynasets and Snapshots)

Creat an OLE Server library to handle adding, editing, deleting, reading, writing, and locating records in datasets. These routines were written as a generic DLL that can be inserted into all Visual Basic programs you write in the future.

You used the new library to add a new form to the CompanyMaster database project. This new form reads a dataset and allows the user to update and browse the table. This new data entry form was built using fewer than 30 lines of Visual Basic code.

Also, you build a graph ActiveX DLL object library that you can use to display virtually any dataset in a variety of graph formats. This library lets you save the graph to disk, send the graph to the printer, or copy the graph to the Windows Clipboard for placement in other Windows programs through the Paste Special operation.

While building the graph library, you declare and use enumerated constants to improve the readability of your Visual Basic code.

Finally, you use the new graph library to add graphs to the CompanyMaster project.

Load and use three of the data-bound controls that are shipped with Visual Basic 5:

The data-bound list box

The data-bound combo box

The data-bound grid

You link these new controls to Recordsets using the Visual Basic 5 data controls and how to use these links to update related tables.

There are several of the important Visual Basic 5 events associated with the data grid. These events let you create user-friendly data entry routines using just a data control and the data grid.

Finally, you must draw upon your knowledge of data grids, SQL, and form layout to design and implement a data entry subform. This form shows a master table at the top, and a related list table at the bottom of the form in a data-bound grid.

How to create, alter, and delete database table structures using DDL (Data Definition Language) SQL keywords are one of the major topics of Visual Basic. Using DDL statements to build tables, create indexes, and establish relationships is an excellent way to automatically document table layouts. Maintaining database structures using the following DDL keywords:

`CREATE TABLE` enables you to create entirely new tables in your existing database.

`DROP TABLE` enables you to completely remove a table, including any data that is already in the table.

`ALTER TABLE` enables you to `ADD` a new column or `DROP` an existing column from the table without losing existing data in the other columns.

`CREATE INDEX` and `DROP INDEX` enable you to create indexes that can enforce data integrity or speed data access.

The `CONSTRAINT` clause can be added to the `CREATE TABLE` or `ALTER TABLE` statement to define relationships between tables using the `FOREIGN KEY` clause.

Errors

You can create creating your own error-handling routines for Visual Basic applications. An error handler has three basic parts:

The `On Error Goto` statement

The body of the error handler code

The error handler exit

An error handler has four possible exits:

`Resume`: Re-executes the code that caused the error.

Resume Next: Continues processing at the line immediately following the line that caused the error.

Resume label: Continues processing at the location identified by the label.

EXIT or END: EXIT ends processing for the current routine and END exits the program completely.

Use the `Err.Raise` method to flag errors without resorting to modal dialog boxes

The major types of errors that you are likely to encounter in your program:

General file errors: These include errors such as File not Found and Invalid Path. Errors of this type can usually be fixed by the user and then the original procedure re-attempted. Use `Resume` as an exit for these types of errors.

Database errors: These include errors related to data entry mistakes, integrity violations, and multiuser-related errors, such as locked records. Errors of this type are best handled by allowing the user to correct the data and attempt the operation again. If you use the Visual Basic data control, you do not have to write error handlers--the data control handles them for you. For operations that do not use the data control, you need to write your own error-handling routines.

Physical media errors: These errors relate to device problems, such as unresponsive printers, downed communications ports, and so on. Sometimes users can fix the problems and continue (for example, refilling the paper tray of the printer). Other times, users cannot fix the problem without first exiting the program. It is a good idea to give users the option of exiting the program safely when errors of these types are reported.

Program code errors: These errors occur because of problems within the Visual Basic code itself. Examples of program code errors include Object variable not Set and For loop not initialized. Usually the user cannot do anything to fix errors of this type. It is best to encourage the user to report the error to the system administrator and then exit the program safely.

You can declare a global error handler or a local error handler. The advantage of the global error handler is that it allows you to create a single module that handles all expected errors. The disadvantage is that, because of the way Visual Basic keeps track of running routines, you are not able to resume processing at the point the error occurs once you arrive at the global error handler. The advantage of the local error handler is that you are always able to use `Resume`, `Resume Next`, or `Resume label` to continue processing at the point the error

occurs. The disadvantage of the local error handler is that you need to add error-handling code to every routine in your program.

Finally, you must create an error handler object library that combines local error trapping with global error messages and responses. The error handler object library also contains modules to keep track of the procedures currently running at the time of the error, a process for printing procedure stack dumps to the screen and to a file, and a process that creates an error log on file for later review.

When you develop database applications for multiple users or multiple sites. You must advance SQL language for manipulating records within existing databases (DML). There are five rules of data normalization that is applied to improve the speed, accuracy, and integrity of your databases.

Visual Basic database locking schemes for the database, table, and page level ,the advantages and limitations of adding cascading updates and deletes to your database relationship definitions. Using Visual Basic keywords `BeginTrans`, `CommitTrans`, and `Rollback` to improve database integrity and processing speed during mass updates.

Remote Data Control and the Remote Data Objects. We use these tools to attach to RDBMSs and they have properties, methods, and events of these useful tools.

write data entry forms that use the ODBC API calls to link directly with the ODBC interface to access data in registered ODBC data sources and install the ODBC Administrator, which are used to create new ODBC data sources for your ODBC-enabled Visual Basic programs.

Updating Databases with SQL.

To add, delete, and edit data within tables using the DML (Data Manipulation Language) SQL keywords by using DML statements you can quickly create test data for tables and load default values into startup tables. DML statements--such as `Append` queries, `Make Table` queries, and `Delete` queries--can outperform equivalent Visual Basic code versions of the same operations.

Managing data within the tables using the following DML keywords:

The `INSERT INTO` statement can be used to add new rows to the table using the `VALUES` clause.

You can create an `Append` query by using the `INSERT INTO...FROM` syntax to copy data from one table to another. You can also copy data from one database to another using the `IN` clause on an `INSERT INTO...FROM` statement.

You can create new tables by copying the structure and some of the data using the `SELECT INTO` statement. This statement can incorporate `WHERE`, `ORDER BY`, `GROUP BY`, and `HAVING` clauses to limit the scope of the data used to populate the new table you create.

You can use the `DELETE FROM` clause to remove one or more records from an existing table. You can even create customized views of the database using the `JOIN` clause, and remove only records that are the result of a `JOIN` statement.

Database Normalization to improve database integrity and access speed using the five rules of data normalization:

Rule 1: Eliminate repeating groups. If you have a set of fields that have the same name followed by a number (Skill1, Skill2, Skill3, and so forth), remove these repeating groups, create a new table for the repeating data, and relate it to the key field in the first table.

Rule 2: Eliminate redundant data. Don't store the same data in two different locations. This can lead to update and delete errors. If equivalent data elements are entered in two fields, remove the second data element, create a new master table with the element and its partner as a key field, and then place the key field as a relationship in the locations that formerly held both data elements.

Rule 3: Eliminate columns not dependent on keys. If you have data elements that are not directly related to the primary key of the table, these elements should be removed to their own data table. Only store data elements that are directly related to the primary key of the table. This particularly includes derived data or other calculations.

Rule 4: Isolate independent multiple relationships. Use this rule to improve database design when you are dealing with more than one one-to-many relationship in the database. Before you add a new field to a table, ask yourself whether this field is really dependent upon the other fields in the table. If not, create a new table with the independent data.

Rule 5: Isolate related multiple relationships. Use this rule to improve database design when you are dealing with more than one many-to-many relationship in the database. If you have database rules that require multiple references to the same field or sets of fields, isolate the fields into smaller tables and construct one or more link tables that contain the required constraints that enforce database integrity.

Multiuser Considerations

The three important challenges that face every database programmer writing multiuser applications:

Database locking schemes

Using cascading updates and deletes to maintain database integrity .

Using database transactions to provide commit/rollback options for major updates to your database.

There are three levels of locking available to Visual Basic programs:

The database level--You can use the Exclusive property of the data control or the second parameter of the `OpenDatabase` method to lock the entire database. Use this option when you need to perform work that affects multiple database objects (such as tables, queries, indexes, relations, and so on).

The table level--You can set the Options property of the data control to 3 or the third parameter of the `OpenRecordset` method to `dbDenyRead+dbDenyWrite` in order to lock the entire table for your use only. Use this option when you need to perform work that affects multiple records in a single table (for example, increasing the sales price on all items in the inventory table).

The page level--Microsoft Jet automatically performs page-level locking whenever you use the data control to edit and save a record, or whenever you use Visual Basic code to perform the `Edit/AddNew` and `Update/CancelUpdate` methods. You can use the `LockEdits` property of the `Recordset` to set the page locking to pessimistic (to perform locking at edit time) or optimistic (to perform locking only at update time).

Use Visual Basic to enforce referential integrity and automatically perform cascading updates or deletes to related records. You learned that there are times when it is not advisable to establish cascading

We can use database transactions to protect your database during extended, multitable operations by the `BeginTrans`, `CommitTrans`, and `Rollback` methods of the workspace object.

Using the Remote Data Control and the Remote Data Objects there are two alternate methods for accessing remote data by using the Remote Data control to create simple data

entry forms with data-bound controls use the Remote Data Objects to create Visual Basic 5.0 programs that can access data from a remote RDBMS.

Along with the details of the Remote Data Control and the Remote Data objects Some of the basics of remote data access in general:

Cursor drivers--These are the tools that manage the location of the Recordset pointer in a dataset. you can use client-side or server-side cursor drivers with RDC/RDO connections.

Dataset types-- there are a number of dataset types available to you when you connect to remote data sources including ForwardOnly--ReadOnly sets, Static sets, Keysets, and Dynamic sets.

Lock types-- there are several different lock types you can use when accessing data from your remote data source. You can use ConcurrentLock sets that perform locks as soon as you receive the data rows, or you can use several versions of optimistic locking that only attempt to lock the rows when you update them.

Details of the following Microsoft Remote Data Objects:

The rdoEngine object--This is the top-level data engine used to access remote data.

The rdoEnvironment object--This is the RDO equivalent of the Microsoft Jet Workspace object.

The rdoConnection object--This is the RDO equivalent of the Microsoft Jet Database object.

The rdoResultset object--This is the RDO equivalent of the Microsoft Jet Recordset object.

The rdoTable object--This is the RDO version of the Microsoft Jet Table object.

The rdoColumn object--This is the RDO version of the Microsoft Jet Field object.

The rdoQuery object--This is the RDO version of the Microsoft Jet QueryDef object.

The rdoParameters object--This is a special collection of query parameters for the rdoQuery object.

ODBC Data Access via the ODBC API

Using the Open Database Connectivity (ODBC) API to directly link your Visual Basic program to target data sources through the ODBC interface. The ODBC interface is generally faster than Microsoft Jet when it comes to linking to ODBC-defined data sources.

By installing the ODBC interface on your workstation and how to use the ODBC Administrator program to install ODBC driver sets and define data sources for ODBC connections.

The building of a program library that uses a minimum set of ODBC API calls along with several Visual Basic wrapper routines. This library set provides the basic functions necessary to read and write data to and from a defined ODBC data source. You can use these routines to create fully functional data entry forms for ODBC data sources.

Database Replication

In database replication terminology, the main or central database is referred to as the Design Master. A copy of the Design Master is referred to as the replica. The combination of the Design Master and all replicas is referred to as the replica set. Database replication is the process of synchronizing data so that it is the same across all members of the replica set.

Database replication is a good tool to use in the development of systems deployed across a WAN or to remote users. Replication can also be used to make copies of databases that cannot be shut down. Replication is also good for creating reporting databases and data marts.

Do not use database replication when a centralized data storage facility can be used, such as a Web-enabled application. Also, don't use replication in heavily transaction-oriented applications, or in applications where up-to-the minute accuracy is of paramount importance. Tables, fields, and properties are added to a database when it is made a Design Master. The addition of these items is necessary to track changes to data and to facilitate the synchronization between members of the replica set. These additions, however, consume additional physical hard drive space.

Creating and changing the Replicable property of a database to `T` creates a Design Master. Once the Design Master is created, you can use the `Make Replica` method to make copies of it. Finally, you use the `Synchronize` method to replicate data changes to members of the replica set. Data synchronization is the act of copying data changes from one member of a replica set to another.

The `Synchronize` method can be used to import data changes, export data changes, perform "two-way" data changes, and even perform data exchanges over the Internet.

Synchronization errors occur when two members of a replica set try to synchronize records that both have changed. Errors may also occur during the synchronization process when design changes are made to a database that are violated by replicas prior to synchronization of the changes. Violation of referential integrity can be encountered by a replica that added records to its database that uses validation records deleted in another replica. Record locking in a multiuser environment can also cause synchronization errors.

There are four topologies for the synchronization of replicas. These are the star, linear, ring, and fully connected topologies. The star is the most common, but like all the other topologies it has certain strengths and weaknesses.

There may be times when you do not want to replicate objects contained in one database to other members of the replica set. If such is the case, use the `KeepLocal` method before you create the Design Master. This method keeps the object from being copied to other replica set members.

Securing Your Database Applications

There are several methods that can improve user and application-level security for your Visual Basic database applications. the limitations of using the Microsoft Access `SYSTEM` security file and database encryption.

Adding application-level security to your Visual Basic programs by adding user login/logout routines and creating a user access rights scheme for your applications. In this lesson, you designed and implemented an OLE Server DLL library that you can use for all your Visual Basic applications, and you created several screens for maintaining user lists and managing access rights for each user.

Adding an audit trail option to your programs. You add routines to a new OLE Server DLL library that logs all critical user activity to an audit trail file, including user logins, database modifications, and all critical program operations, such as running reports or processing mass database updates.

STOCK CONTROL SYSTEM

This project the Stock Control System is a real time application that has a role in administering the case of stock and the related transactions that take place during an order for product and making the appropriate balance.

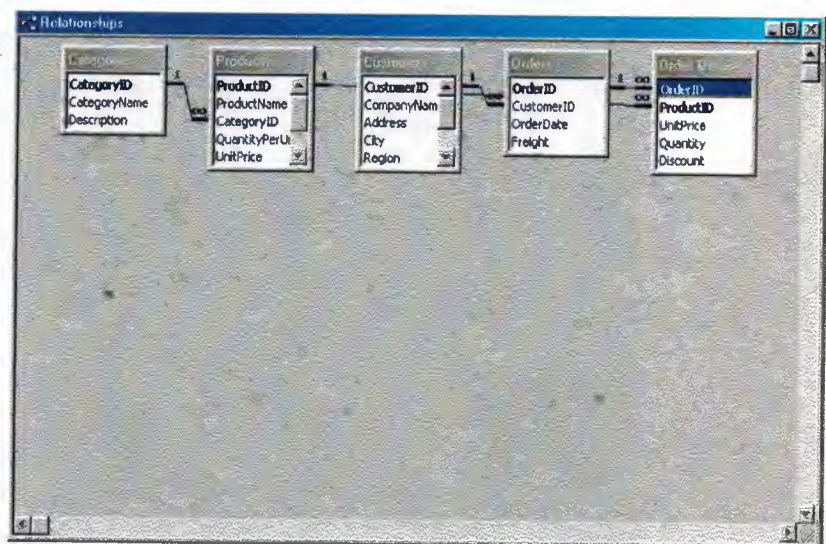
The program uses Microsoft Access as a database relating pot only for the reason of providing easy tables and queries.

After the construction of the tables in Access we use the record sources of the database mdb File as the provider of data, which will be used, on various parts of the Visual Basic project.

The project screens and layouts are provided below along with Microsoft Access Tables, Queries and relations.

Tables and relations "Access":

- Categories (Stocks).
- Products.
- Customers.
- Orders.
- Order Details.



Screens and Layouts “VB”:

- Log in:

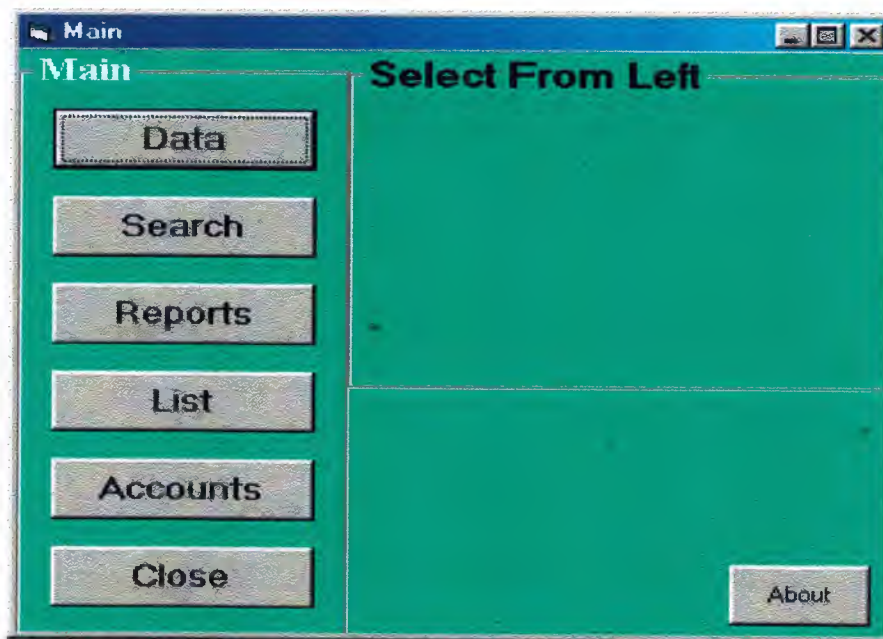
This screen to login and open the program.



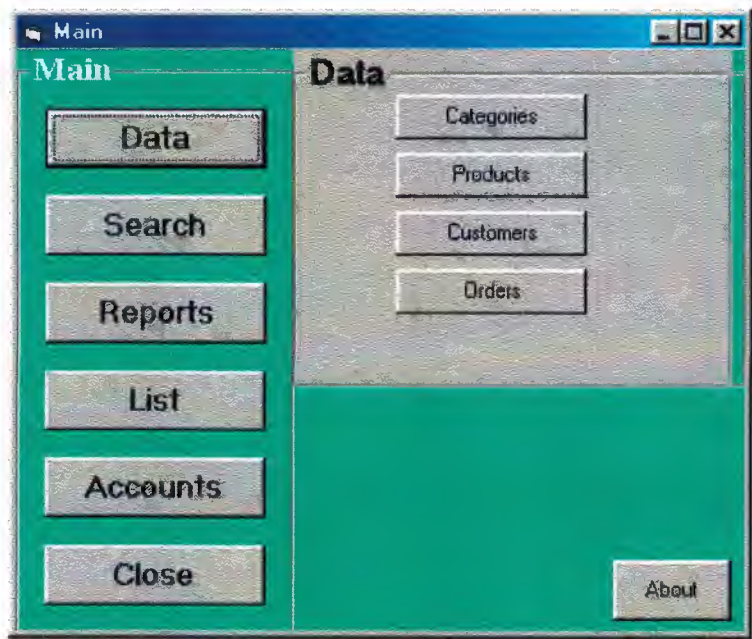
- Main Menu:

The main screen describes the 1st user interface, which guide the user through in the different screens in the program.

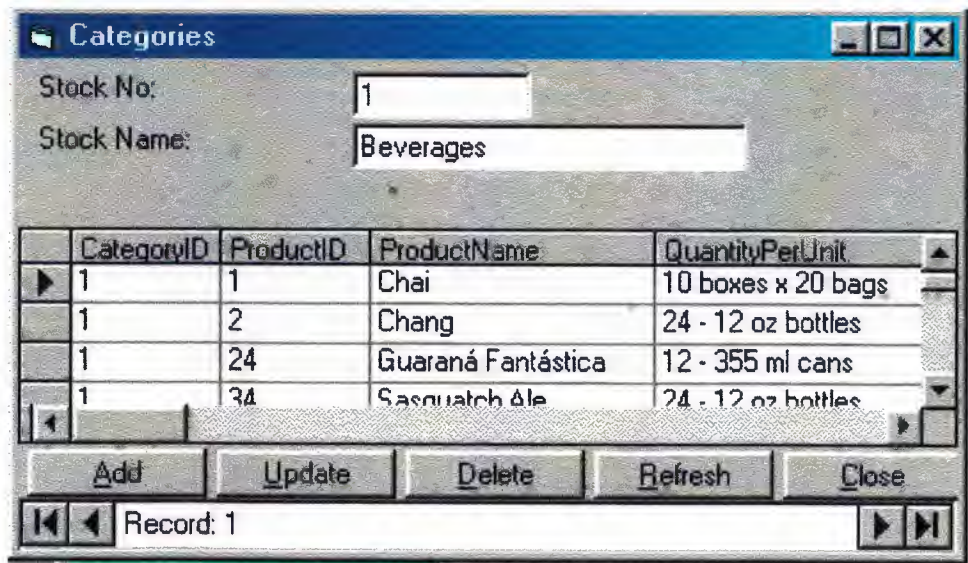
It includes 6 bottoms: Data, Search, Reports, List, Accounts, and Close quiet the program.



1. Data: Here we automate a real life procedure, when we divide the Data into four sub screens: Categories (Stocks), Products, Customers, and Orders.

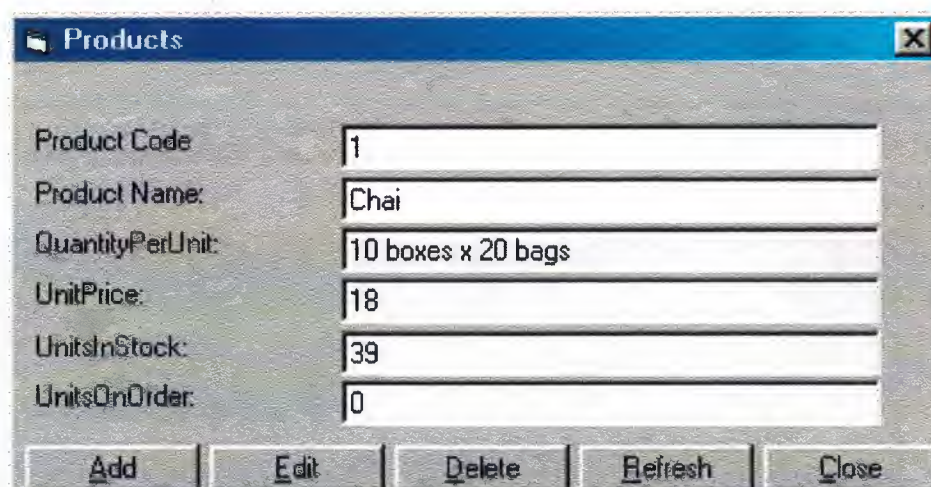


1.1: Categories (Stocks): This screen to enter the information & Data of the new stock.



1.2: Products:

Here to enter the information's Product of the new stock.

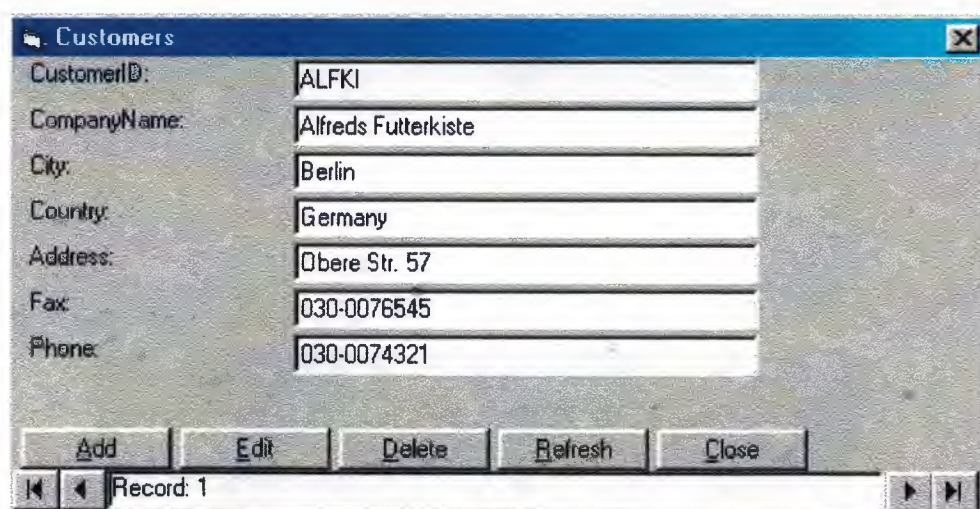


Product Code	1
Product Name:	Chai
QuantityPerUnit:	10 boxes x 20 bags
UnitPrice:	18
UnitsInStock:	39
UnitsOnOrder:	0

Buttons: Add, Edit, Delete, Refresh, Close

1.3: Customers:

Customer's names, and privet information are located in the designated CUSTOMER screen.



CustomerID:	ALFKI
CompanyName:	Alfreds Futterkiste
City:	Berlin
Country:	Germany
Address:	Obere Str. 57
Fax:	030-0076545
Phone:	030-0074321

Buttons: Add, Edit, Delete, Refresh, Close

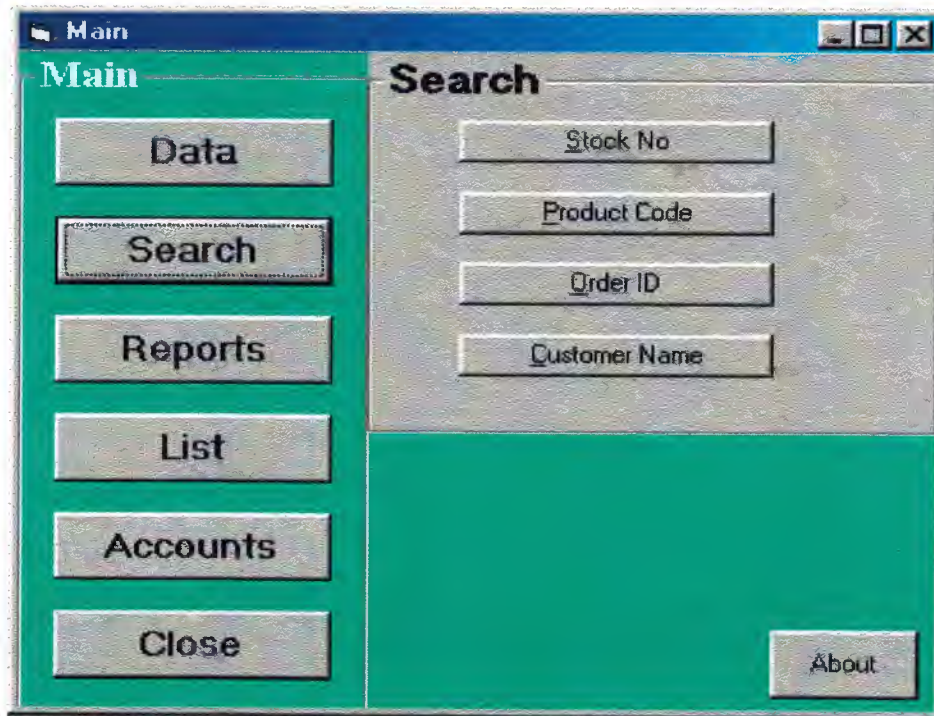
Record: 1

1.4: Orders:

This screen for the ordered products information.

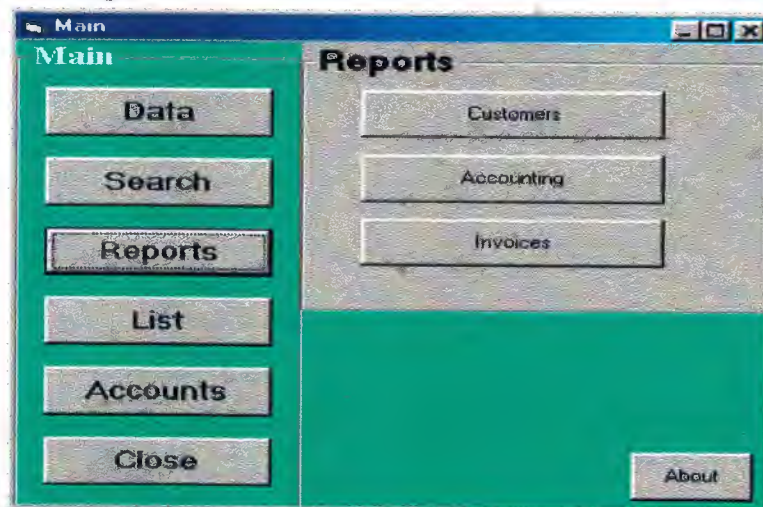
2. Search:

In order to offer a high quality services inside the system, the 2nd sub screen of the main menu is the SEARCH engine, our searching engine can give the users a very soon result for any question about the main tow categories inside our system: Stocks, Products, Orders, and Customers. And locate the users with their available information.



3. Reports:

The 3rd sub screen of the main menu is Reports. And it has also three sub screens: Customers, Accounting, and Invoices.



3.1: Customer Information:

DataReport1

Zoom 100%

CUSTOMER INFORMATION

Customer Id	Company Name	City	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Berlin	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y	México D.F.	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	México D.F.	Mexico	(5) 555-3932	
AROUT	Around the Horn	London	UK	(171)	(171)
BERGS	Berglunds snabbköp	Luleå	Sweden	0921-12 34 65	0921-12 34
BLAUS	Blauer See Delikatessen	Mannheim	Germany	0621-08460	0621-08924
BLONP	Blondel père et fils	Strasbourg	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Madrid	Spain	(91) 555 22	(91) 555 91
BONAP	Bon app'	Marseille	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Tsawassen	Canada	(604)	(604)
BSBEV	B's Beverages	London	UK	(171)	
CACTU	Cactus Comidas para llevar	Buenos Aires	Argentina	(1) 135-5555	(1) 135-4892
CENTC	Centro comercial Moctezuma	México D.F.	Mexico	(5) 555-3392	(5) 555-7293
CHOPS	Chop-suey Chinese	Bern	Switzerland	0452-076545	
COMMI	Comércio Mineiro	São Paulo	Brazil	(11) 555-7647	
CONSH	Consolidated Holdings	London	UK	(171)	(171)
DRACD	Drachenblut Delikatessen	Aachen	Germany	0241-039123	0241-059428

3.2: Accounting:

DataReport2

Zoom 100%

ACCOUNTING

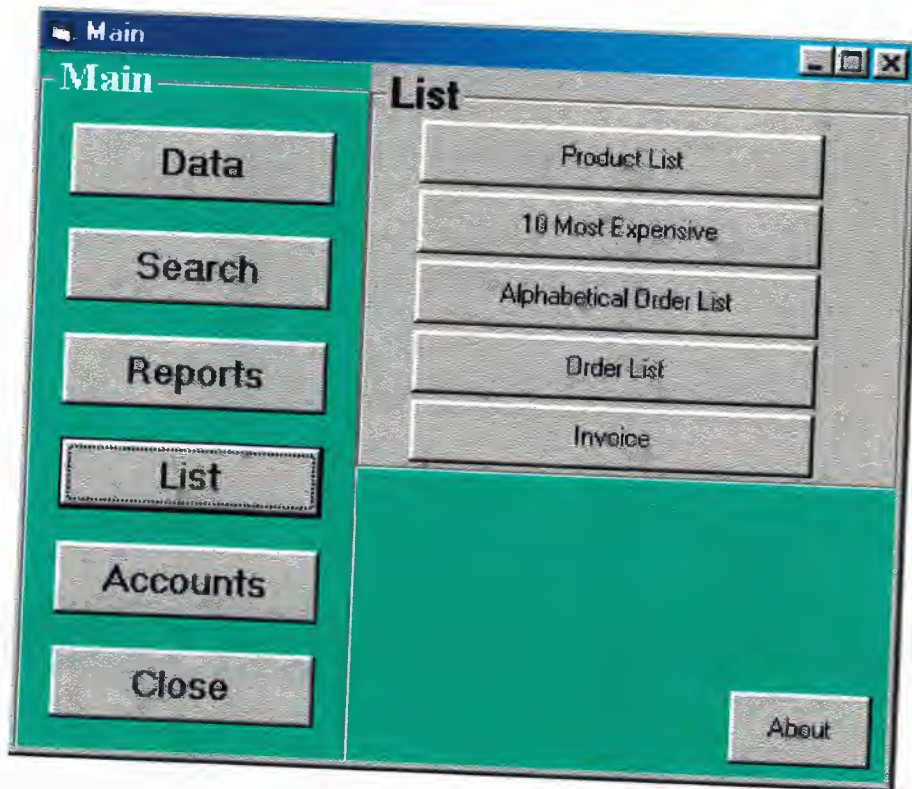
Code	Name	Stock No	Quantity	Unit Price
1	Chef	1	10 boxes x 20 bags	18
2	Chang	1	24 - 12 oz bottles	19
3	Aniseed Syrup	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	36 boxes	21,35
6	Grandma's Boysenberry Spread	2	12 - 8 oz jars	25
7	Uncle Bob's Organic Dried	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry Sauce	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	6	18 - 500 g pkgs.	97
10	Ikura	8	12 - 200 ml jars	31
11	Queso Cabrales	4	1 kg pkg.	21
12	Queso Manchego La Pastora	4	10 - 500 g pkgs.	38
13	Konbu	8	2 kg box	6
14	Tofu	7	40 - 100 g pkgs.	23,25
15	Genen Shouyu	2	24 - 250 ml bottles	15,5
16	Pavlova	3	32 - 500 g boxes	17,45
17	Alice Mutton	6	20 - 1 kg tins	39
18	Carnarvon Tigers	8	16 kg pkg.	62,5
19	Teatime Chocolate Biscuits	3	10 boxes x 12 pieces	9,2
20	Sir Rodney's Marmalade	3	30 gift boxes	81
21	Sir Rodney's Scones	3	24 pkgs. x 4 pieces	10

Pages: 1

3.3: Invoices.

4. Lists:

The 4th sub screen of the main menu is List. And it has also five sub screens: Product List, 10 Most Expensive List, Alphabetical Order List, Orders List, and Invoices List.



4.1: Product List:

The screenshot shows a Windows-style application window titled 'Product List'. It contains a table with two columns: 'ProductID' and 'ProductName'. The table lists 25 products. The first product, 'Alice Mutton', is selected, indicated by a mouse cursor icon in the first column.

ProductID	ProductName
17	Alice Mutton
3	Aniseed Syrup
40	Boston Crab Meat
60	Camembert Pierrot
18	Carnarvon Tigers
1	Chai
2	Chang
39	Chartreuse verte
4	Chef Anton's Cajun Sea
5	Chef Anton's Gumbo Mix
48	Chocolade
38	Côte de Blaye
78	dîdî
58	Escargots de Bourgogne
79	fifi
52	Filo Mix
71	Flotemysost
33	Geitost
15	Genen Shouyu
56	Gnocchi di nonna Alice
31	Gorgonzola Telino
6	Grandma's Boysenberry

4.2: 10 Most Expensive List:

10 Most Expensive		
	TenMostExpensiveProduct	UnitPrice
▶	Côte de Blaye	263,5
	Thüringer Rostbratwurst	123,79
	Mishi Kobe Niku	97
	Sir Rodney's Marmalade	81
	Carnarvon Tigers	62,5
	Raclette Courdavault	55
	Manjimup Dried Apples	53
	Tarte au sucre	49,3
	Ipoh Coffee	46
	Rössle Sauerkraut	45,6

4.3: Alphabetical Order List:

alpha-prod-list							
	ProductID	ProductName	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder
▶	1	Chai	1	10 boxes x 20 bags	18	39	0
	2	Chang	1	24 - 12 oz bottles	19	17	40
	3	Aniseed Syrup	2	12 - 550 ml bottles	10	13	70
	4	Chef Anton's Cajun Sea	2	48 - 6 oz jars	22	53	0
	5	Chef Anton's Gumbo Mix	2	36 boxes	21,35	0	0
	6	Grandma's Boysenberry	2	12 - 8 oz jars	25	120	0
	7	Uncle Bob's Organic Dried	7	12 - 1 lb pkgs.	30	15	0
	8	Northwoods Cranberry Sauce	2	12 - 12 oz jars	40	6	0
	9	Mishi Kobe Niku	6	18 - 500 g pkgs.	97	29	0
	10	Ikura	8	12 - 200 ml jars	31	31	0
	11	Queso Cabrales	4	1 kg pkg.	21	22	30
	12	Queso Manchego La Pasa	4	10 - 500 g pkgs.	38	86	0
	13	Konbu	8	2 kg box	6	24	0
	14	Tofu	7	40 - 100 g pkgs.	23,25	35	0
	15	Genen Shouyu	2	24 - 250 ml bottles	15,5	39	0
	16	Pavlova	3	32 - 500 g boxes	17,45	29	0
	17	Alice Mutton	6	20 - 1 kg tins	39	0	0
	18	Carnarvon Tigers	8	16 kg pkg.	62,5	42	0
	19	Teatime Chocolate Biscuits	3	10 boxes x 12 pieces	9,2	25	0

4.4: Order List:

Order List						
OrderID	CustomerID	OrderDate	Freight	CompanyName	Address	City
10643	ALFKI	25.08.1997	29.46	Alfreds Futterkiste	Obere Str. 57	Berlin
10952	ALFKI	16.03.1998	40.42	Alfreds Futterkiste	Obere Str. 57	Berlin
10692	ALFKI	03.10.1997	61.02	Alfreds Futterkiste	Obere Str. 57	Berlin
10835	ALFKI	15.01.1998	69.53	Alfreds Futterkiste	Obere Str. 57	Berlin
11011	ALFKI	09.04.1998	1.21	Alfreds Futterkiste	Obere Str. 57	Berlin
10702	ALFKI	13.10.1997	23.94	Alfreds Futterkiste	Obere Str. 57	Berlin
10759	ANATR	28.11.1997	11.99	Ana Trujillo Emparedado	Avda. de la Constitución	México
10926	ANATR	04.03.1998	39.92	Ana Trujillo Emparedado	Avda. de la Constitución	México
10308	ANATR	18.09.1996	1.61	Ana Trujillo Emparedado	Avda. de la Constitución	México
10625	ANATR	08.08.1997	43.9	Ana Trujillo Emparedado	Avda. de la Constitución	México
10682	ANTON	25.09.1997	36.13	Antonio Moreno Taquer	Mataderos 2312	México
10535	ANTON	13.05.1997	15.64	Antonio Moreno Taquer	Mataderos 2312	México
10365	ANTON	27.11.1996	22	Antonio Moreno Taquer	Mataderos 2312	México
10573	ANTON	19.06.1997	84.84	Antonio Moreno Taquer	Mataderos 2312	México
10856	ANTON	28.01.1998	58.43	Antonio Moreno Taquer	Mataderos 2312	México
10507	ANTON	15.04.1997	47.45	Antonio Moreno Taquer	Mataderos 2312	México
10677	ANTON	22.09.1997	4.03	Antonio Moreno Taquer	Mataderos 2312	México
10741	AROUT	14.11.1997	10.96	Around the Horn	120 Hanover Sq.	London
10383	AROUT	16.12.1996	34.24	Around the Horn	120 Hanover Sq.	London
10355	AROUT	15.11.1996	41.95	Around the Horn	120 Hanover Sq.	London
10453	AROUT	21.02.1997	25.36	Around the Horn	120 Hanover Sq.	London
10864	AROUT	02.02.1998	3.04	Around the Horn	120 Hanover Sq.	London

4.5: Invoices List:

Invoice					
CustomerID	CompanyName	Address	City	Region	Country
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ALFKI	Alfreds Futterkiste	Obere Str. 57	Berlin		Germany
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANATR	Ana Trujillo Emparedado	Avda. de la Constitución	México D.F.		Mexico
ANTON	Antonio Moreno Taquer	Mataderos 2312	México D.F.		Mexico

CODE

Log in - Code:

Option Explicit

Public LoginSucceeded As Boolean

Private Sub cmdCancel_Click()

 'set the global var to false

 'to denote a failed login

 LoginSucceeded = False

 Me.Hide

End Sub

Private Sub cmdOK_Click()

 'check for correct password

 If txtPassword = "yasir" Then

 'place code to here to pass the

 'success to the calling sub

 'setting a global var is the easiest

 LoginSucceeded = True

 frmmain.Show

 Me.Hide

 Else

 MsgBox "Invalid Password, try again!", , "Login"

 txtPassword.SetFocus

 SendKeys "{Home}+{End}"

 End If

End Sub

Main – Code:

Private Sub cmdlist_Click(Index As Integer)

 Select Case Index

 Case 0

 Form1.Show 1

 Case 1

 Form2.Show 1

 Case 2

 Form3.Show 1

 Case 3

 Form4.Show 1

 Case 4

 Form5.Show 1

End Select

End Sub

```
Private Sub cmdrepot_Click(Index As Integer)
```

```
Select Case Index
```

```
Case 0
```

```
    DataReport1.Show 1
```

```
Case 1
```

```
    DataReport2.Show 1
```

```
Case 2
```

```
    Datareport3.Show 1
```

```
End Select
```

```
End Sub
```

```
Private Sub Command1_Click(Index As Integer)
```

```
cmdpanel(Index).ZOrder
```

```
End Sub
```

```
Private Sub Command2_Click()
```

```
frmAbout.Show 1
```

```
End Sub
```

```
Private Sub Command4_Click()
```

```
Unload Me
```

```
End Sub
```

```
Private Sub Datacmd_Click(Index As Integer)
```

```
Select Case Index
```

```
Case 0
```

```
    category.Show 1
```

```
Case 1
```

```
    product.Show 1
```

```
Case 2
```

```
    customer.Show 1
```

```
Case 3
```

```
    order.Show 1
```

```
End Select
```

```
End Sub
```

Categories - Code:

```
Private Sub Form_Load()
```

```
Set grdDataGrid.DataSource = datPrimaryRS.Recordset("ChildCMD").UnderlyingValue
```

```
End Sub
```



```

Private Sub Form_Resize()
    On Error Resume Next
    'This will resize the grid when the form is resized
    grdDataGrid.Width = Me.ScaleWidth
    grdDataGrid.Height = Me.ScaleHeight - grdDataGrid.Top - datPrimaryRS.Height - 30 -
picButtons.Height
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)
    Screen.MousePointer = vbDefault
End Sub

```

```

Private Sub datPrimaryRS_Error(ByVal ErrorNumber As Long, Description As String,
ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal
HelpContext As Long, fCancelDisplay As Boolean)
    'This is where you would put error handling code
    'If you want to ignore errors, comment out the next line
    'If you want to trap them, add code here to handle them
    MsgBox "Data error event hit err:" & Description
End Sub

```

```

Private Sub datPrimaryRS_MoveComplete(ByVal adReason As ADODB.EventReasonEnum,
ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pRecordset
As ADODB.Recordset)
    'This will display the current record position for this recordset
    datPrimaryRS.Caption = "Record: " & CStr(datPrimaryRS.Recordset.AbsolutePosition)
End Sub

```

```

Private Sub datPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This is where you put validation code
    'This event gets called when the following actions occur
    Dim bCancel As Boolean

```

```

Select Case adReason
    Case adRsnAddNew
    Case adRsnClose
    Case adRsnDelete
    Case adRsnFirstChange
    Case adRsnMove
    Case adRsnRequery
    Case adRsnResynch
    Case adRsnUndoAddNew
    Case adRsnUndoDelete
    Case adRsnUndoUpdate
    Case adRsnUpdate
End Select

```

```
If bCancel Then adStatus = adStatusCancel
End Sub
```

```
Private Sub cmdAdd_Click()
    On Error GoTo AddErr
    datPrimaryRS.Recordset.AddNew
```

```
Exit Sub
AddErr:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdDelete_Click()
    On Error GoTo DeleteErr
    With datPrimaryRS.Recordset
        .Delete
        .MoveNext
        If .EOF Then .MoveLast
    End With
    Exit Sub
DeleteErr:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdRefresh_Click()
    'This is only needed for multi user apps
    On Error GoTo RefreshErr
    datPrimaryRS.Refresh
    Set grdDataGrid.DataSource = datPrimaryRS.Recordset("ChildCMD").UnderlyingValue
    Exit Sub
RefreshErr:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdUpdate_Click()
    On Error GoTo UpdateErr

    datPrimaryRS.Recordset.UpdateBatch adAffectAll
    Exit Sub
UpdateErr:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdClose_Click()
    Unload Me
End Sub
```

Customer - Code:

```
Dim WithEvents adoPrimaryRS As Recordset
```

```

Dim mbChangedByCode As Boolean
Dim mvBookMark As Variant
Dim mbEditFlag As Boolean
Dim mbAddNewFlag As Boolean
Dim mbDataChanged As Boolean

Private Sub Form_Load()
    Dim db As Connection
    Set db = New Connection
    db.CursorLocation = adUseClient
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data
Source=C:\stock\STOCKYA.mdb;"

    Set adoPrimaryRS = New Recordset
    adoPrimaryRS.Open "select
Address,City,CompanyName,Country,CustomerID,Fax,Phone,Region from Customers Order
by CustomerID", db, adOpenStatic, adLockOptimistic

    Dim oText As TextBox
    'Bind the text boxes to the data provider
    For Each oText In Me.txtFields
        Set oText.DataSource = adoPrimaryRS
    Next

    mbDataChanged = False
End Sub

Private Sub Form_Resize()
    On Error Resume Next
    lblStatus.Width = Me.Width - 1500
    cmdNext.Left = lblStatus.Width + 700
    cmdLast.Left = cmdNext.Left + 340
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    If mbEditFlag Or mbAddNewFlag Then Exit Sub

    Select Case KeyCode
        Case vbKeyEscape
            cmdClose_Click
        Case vbKeyEnd
            cmdLast_Click
        Case vbKeyHome
            cmdFirst_Click
        Case vbKeyUp, vbKeyPageUp
            If Shift = vbCtrlMask Then
                cmdFirst_Click
            Else
                cmdPrevious_Click
            End If
    End Select
End Sub

```




```
Case vbKeyDown, vbKeyPageDown
  If Shift = vbCtrlMask Then
    cmdLast_Click
  Else
    cmdNext_Click
  End If
End Select
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
  Screen.MousePointer = vbDefault
End Sub
```

```
Private Sub adoPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
  'This will display the current record position for this recordset
  lblStatus.Caption = "Record: " & CStr(adoPrimaryRS.AbsolutePosition)
End Sub
```

```
Private Sub adoPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
  'This is where you put validation code
  'This event gets called when the following actions occur
  Dim bCancel As Boolean
```

```
  Select Case adReason
    Case adRsnAddNew
    Case adRsnClose
    Case adRsnDelete
    Case adRsnFirstChange
    Case adRsnMove
    Case adRsnRequery
    Case adRsnResynch
    Case adRsnUndoAddNew
    Case adRsnUndoDelete
    Case adRsnUndoUpdate
    Case adRsnUpdate
  End Select
```

```
  If bCancel Then adStatus = adStatusCancel
End Sub
```

```
Private Sub cmdAdd_Click()
  On Error GoTo AddErr
  With adoPrimaryRS
    If Not (.BOF And .EOF) Then
      mvBookMark = .Bookmark
    End If
```

```

.AddNew
lblStatus.Caption = "Add record"
mbAddNewFlag = True
SetButtons False
End With

Exit Sub
AddErr:
MsgBox Err.Description
End Sub

Private Sub cmdDelete_Click()
On Error GoTo DeleteErr
With adoPrimaryRS
.Delete
.MoveNext
If .EOF Then .MoveLast
End With
Exit Sub
DeleteErr:
MsgBox Err.Description
End Sub

Private Sub cmdRefresh_Click()
'This is only needed for multi user apps
On Error GoTo RefreshErr
adoPrimaryRS.Requery
Exit Sub
RefreshErr:
MsgBox Err.Description
End Sub

Private Sub cmdEdit_Click()
On Error GoTo EditErr

lblStatus.Caption = "Edit record"
mbEditFlag = True
SetButtons False
Exit Sub

EditErr:
MsgBox Err.Description
End Sub

Private Sub cmdCancel_Click()
On Error Resume Next

SetButtons True
mbEditFlag = False
mbAddNewFlag = False
adoPrimaryRS.CancelUpdate

```

```

If mvBookMark > 0 Then
    adoPrimaryRS.Bookmark = mvBookMark
Else
    adoPrimaryRS.MoveFirst
End If
mbDataChanged = False

End Sub

Private Sub cmdUpdate_Click()
    On Error GoTo UpdateErr

    adoPrimaryRS.UpdateBatch adAffectAll

    If mbAddNewFlag Then
        adoPrimaryRS.MoveLast           'move to the new record
    End If

    mbEditFlag = False
    mbAddNewFlag = False
    SetButtons True
    mbDataChanged = False

    Exit Sub
UpdateErr:
    MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdFirst_Click()
    On Error GoTo GoFirstError

    adoPrimaryRS.MoveFirst
    mbDataChanged = False

    Exit Sub

GoFirstError:
    MsgBox Err.Description
End Sub

Private Sub cmdLast_Click()
    On Error GoTo GoLastError

    adoPrimaryRS.MoveLast
    mbDataChanged = False

```


Exit Sub

GoLastError:

MsgBox Err.Description

End Sub

Private Sub cmdNext_Click()

On Error GoTo GoNextError

If Not adoPrimaryRS.EOF Then adoPrimaryRS.MoveNext

If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then

Beep

'moved off the end so go back

adoPrimaryRS.MoveLast

End If

'show the current record

mbDataChanged = False

Exit Sub

GoNextError:

MsgBox Err.Description

End Sub

Private Sub cmdPrevious_Click()

On Error GoTo GoPrevError

If Not adoPrimaryRS.BOF Then adoPrimaryRS.MovePrevious

If adoPrimaryRS.BOF And adoPrimaryRS.RecordCount > 0 Then

Beep

'moved off the end so go back

adoPrimaryRS.MoveFirst

End If

'show the current record

mbDataChanged = False

Exit Sub

GoPrevError:

MsgBox Err.Description

End Sub

Private Sub SetButtons(bVal As Boolean)

cmdAdd.Visible = bVal

cmdEdit.Visible = bVal

cmdUpdate.Visible = Not bVal

cmdCancel.Visible = Not bVal

cmdDelete.Visible = bVal

cmdClose.Visible = bVal

cmdRefresh.Visible = bVal

cmdNext.Enabled = bVal

```

cmdFirst.Enabled = bVal
cmdLast.Enabled = bVal
cmdPrevious.Enabled = bVal
End Sub

```

Order Details – Code:

```

Dim WithEvents adoPrimaryRS As Recordset
Dim mbChangedByCode As Boolean
Dim mvBookMark As Variant
Dim mbEditFlag As Boolean
Dim mbAddNewFlag As Boolean
Dim mbDataChanged As Boolean

Private Sub Form_Load()
    Dim db As Connection
    Set db = New Connection
    db.CursorLocation = adUseClient
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data
Source=C:\stock\STOCKYA.mdb;"

    Set adoPrimaryRS = New Recordset
    adoPrimaryRS.Open "select Discount,OrderID,ProductID,Quantity,UnitPrice from [Order
Details] Order by OrderID", db, adOpenStatic, adLockOptimistic

    Dim oText As TextBox
    'Bind the text boxes to the data provider
    For Each oText In Me.txtFields
        Set oText.DataSource = adoPrimaryRS
    Next

    mbDataChanged = False
End Sub

Private Sub Form_Resize()
    On Error Resume Next
    lblStatus.Width = Me.Width - 1500
    cmdNext.Left = lblStatus.Width + 700
    cmdLast.Left = cmdNext.Left + 340
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    If mbEditFlag Or mbAddNewFlag Then Exit Sub

    Select Case KeyCode
        Case vbKeyEscape
            cmdClose_Click
        Case vbKeyEnd
            cmdLast_Click
    End Select

```

End Sub

```
Private Sub cmdAdd_Click()  
    On Error GoTo AddErr  
    With adoPrimaryRS  
        If Not (.BOF And .EOF) Then  
            mvBookMark = .Bookmark  
        End If  
        .AddNew  
        lblStatus.Caption = "Add record"  
        mbAddNewFlag = True  
        SetButtons False  
    End With
```

Exit Sub

AddErr:

MsgBox Err.Description

End Sub

```
Private Sub cmdDelete_Click()
```

```
    On Error GoTo DeleteErr
```

```
    With adoPrimaryRS
```

```
        .Delete
```

```
        .MoveNext
```

```
        If .EOF Then .MoveLast
```

```
    End With
```

Exit Sub

DeleteErr:

MsgBox Err.Description

End Sub

```
Private Sub cmdRefresh_Click()
```

```
    'This is only needed for multi user apps
```

```
    On Error GoTo RefreshErr
```

```
    adoPrimaryRS.Requery
```

```
    Exit Sub
```

RefreshErr:

MsgBox Err.Description

End Sub

```
Private Sub cmdEdit_Click()
```

```
    On Error GoTo EditErr
```

```
    lblStatus.Caption = "Edit record"
```

```
    mbEditFlag = True
```

```
    SetButtons False
```

```
    Exit Sub
```

EditErr:

MsgBox Err.Description


```

End Sub
Private Sub cmdCancel_Click()
    On Error Resume Next

    SetButtons True
    mbEditFlag = False
    mbAddNewFlag = False
    adoPrimaryRS.CancelUpdate
    If mvBookMark > 0 Then
        adoPrimaryRS.Bookmark = mvBookMark
    Else
        adoPrimaryRS.MoveFirst
    End If
    mbDataChanged = False
End Sub

Private Sub cmdUpdate_Click()
    On Error GoTo UpdateErr

    adoPrimaryRS.UpdateBatch adAffectAll

    If mbAddNewFlag Then
        adoPrimaryRS.MoveLast      'move to the new record
    End If

    mbEditFlag = False
    mbAddNewFlag = False
    SetButtons True
    mbDataChanged = False

    Exit Sub
UpdateErr:
    MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdFirst_Click()
    On Error GoTo GoFirstError

    adoPrimaryRS.MoveFirst
    mbDataChanged = False

    Exit Sub
GoFirstError:
    MsgBox Err.Description

```

End Sub

```
Private Sub cmdLast_Click()  
    On Error GoTo GoLastError
```

```
    adoPrimaryRS.MoveLast  
    mbDataChanged = False
```

Exit Sub

GoLastError:

```
    MsgBox Err.Description  
End Sub
```

```
Private Sub cmdNext_Click()  
    On Error GoTo GoNextError
```

```
    If Not adoPrimaryRS.EOF Then adoPrimaryRS.MoveNext  
    If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then
```

```
        Beep  
        'moved off the end so go back  
        adoPrimaryRS.MoveLast
```

End If

```
    'show the current record  
    mbDataChanged = False
```

Exit Sub

GoNextError:

```
    MsgBox Err.Description  
End Sub
```

```
Private Sub cmdPrevious_Click()  
    On Error GoTo GoPrevError
```

```
    If Not adoPrimaryRS.BOF Then adoPrimaryRS.MovePrevious  
    If adoPrimaryRS.BOF And adoPrimaryRS.RecordCount > 0 Then
```

```
        Beep  
        'moved off the end so go back  
        adoPrimaryRS.MoveFirst
```

End If

```
    'show the current record  
    mbDataChanged = False
```

Exit Sub

GoPrevError:

```
    MsgBox Err.Description  
End Sub
```

```
Private Sub SetButtons(bVal As Boolean)
```

```

cmdAdd.Visible = bVal
cmdEdit.Visible = bVal
cmdUpdate.Visible = Not bVal
cmdCancel.Visible = Not bVal
cmdDelete.Visible = bVal
cmdClose.Visible = bVal
cmdRefresh.Visible = bVal
cmdNext.Enabled = bVal
cmdFirst.Enabled = bVal
cmdLast.Enabled = bVal
cmdPrevious.Enabled = bVal
End Sub

```

Order -Code:

```

Dim WithEvents adoPrimaryRS As Recordset
Dim mbChangedByCode As Boolean
Dim mvBookMark As Variant
Dim mbEditFlag As Boolean
Dim mbAddNewFlag As Boolean
Dim mbDataChanged As Boolean

```

```

Private Sub Form_Load()
    Dim db As Connection
    Set db = New Connection
    db.CursorLocation = adUseClient
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data
Source=C:\stock\STOCKYA.mdb;"

```

```

    Set adoPrimaryRS = New Recordset
    adoPrimaryRS.Open "select CustomerID,Freight,OrderDate,OrderID from Orders Order by
OrderID", db, adOpenStatic, adLockOptimistic

```

```

    Dim oText As TextBox
    'Bind the text boxes to the data provider
    For Each oText In Me.txtFields
        Set oText.DataSource = adoPrimaryRS
    Next

```

```

    mbDataChanged = False
End Sub

```

```

Private Sub Form_Resize()
    On Error Resume Next
    lblStatus.Width = Me.Width - 1500
    cmdNext.Left = lblStatus.Width + 700
    cmdLast.Left = cmdNext.Left + 340
End Sub

```

```

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)

```


If mbEditFlag Or mbAddNewFlag Then Exit Sub

```
Select Case KeyCode
Case vbKeyEscape
    cmdClose_Click
Case vbKeyEnd
    cmdLast_Click
Case vbKeyHome
    cmdFirst_Click
Case vbKeyUp, vbKeyPageUp
    If Shift = vbCtrlMask Then
        cmdFirst_Click
    Else
        cmdPrevious_Click
    End If
Case vbKeyDown, vbKeyPageDown
    If Shift = vbCtrlMask Then
        cmdLast_Click
    Else
        cmdNext_Click
    End If
End Select
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    Screen.MousePointer = vbDefault
End Sub
```

```
Private Sub adoPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This will display the current record position for this recordset
    lblStatus.Caption = "Record: " & CStr(adoPrimaryRS.AbsolutePosition)
End Sub
```

```
Private Sub adoPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This is where you put validation code
    'This event gets called when the following actions occur
    Dim bCancel As Boolean
```

```
Select Case adReason
Case adRsnAddNew
Case adRsnClose
Case adRsnDelete
Case adRsnFirstChange
Case adRsnMove
Case adRsnRequery
Case adRsnResynch
```

```

Case adRsnUndoAddNew
Case adRsnUndoDelete
Case adRsnUndoUpdate
Case adRsnUpdate
End Select

```

```

If bCancel Then adStatus = adStatusCancel
End Sub

```

```

Private Sub cmdAdd_Click()
On Error GoTo AddErr
With adoPrimaryRS
If Not (.BOF And .EOF) Then
mvBookMark = .Bookmark
End If
.AddNew
lblStatus.Caption = "Add record"
mbAddNewFlag = True
SetButtons False
End With

```

```

Exit Sub
AddErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdDelete_Click()
On Error GoTo DeleteErr
With adoPrimaryRS
.Delete
.MoveNext
If .EOF Then .MoveLast
End With
Exit Sub
DeleteErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdRefresh_Click()
'This is only needed for multi user apps
On Error GoTo RefreshErr
adoPrimaryRS.Requery
Exit Sub
RefreshErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdEdit_Click()
On Error GoTo EditErr

```

```

lblStatus.Caption = "Edit record"
mbEditFlag = True
SetButtons False
Exit Sub

EditErr:
    MsgBox Err.Description
End Sub

Private Sub cmdCancel_Click()
    On Error Resume Next

    SetButtons True
    mbEditFlag = False
    mbAddNewFlag = False
    adoPrimaryRS.CancelUpdate
    If mvBookMark > 0 Then
        adoPrimaryRS.Bookmark = mvBookMark
    Else
        adoPrimaryRS.MoveFirst
    End If
    mbDataChanged = False

End Sub

Private Sub cmdUpdate_Click()
    On Error GoTo UpdateErr

    adoPrimaryRS.UpdateBatch adAffectAll

    If mbAddNewFlag Then
        adoPrimaryRS.MoveLast           'move to the new record
    End If

    mbEditFlag = False
    mbAddNewFlag = False
    SetButtons True
    mbDataChanged = False

    Exit Sub
UpdateErr:
    MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdFirst_Click()
    On Error GoTo GoFirstError

```



```
adoPrimaryRS.MoveFirst  
mbDataChanged = False
```

```
Exit Sub
```

```
GoFirstError:  
MsgBox Err.Description  
End Sub
```

```
Private Sub cmdLast_Click()  
On Error GoTo GoLastError
```

```
adoPrimaryRS.MoveLast  
mbDataChanged = False
```

```
Exit Sub
```

```
GoLastError:  
MsgBox Err.Description  
End Sub
```

```
Private Sub cmdNext_Click()  
On Error GoTo GoNextError
```

```
If Not adoPrimaryRS.EOF Then adoPrimaryRS.MoveNext  
If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then  
Beep
```

```
'moved off the end so go back  
adoPrimaryRS.MoveLast
```

```
End If
```

```
'show the current record  
mbDataChanged = False
```

```
Exit Sub
```

```
GoNextError:  
MsgBox Err.Description  
End Sub
```

```
Private Sub cmdPrevious_Click()  
On Error GoTo GoPrevError
```

```
If Not adoPrimaryRS.BOF Then adoPrimaryRS.MovePrevious  
If adoPrimaryRS.BOF And adoPrimaryRS.RecordCount > 0 Then  
Beep
```

```
'moved off the end so go back  
adoPrimaryRS.MoveFirst
```

```
End If
```

```
'show the current record  
mbDataChanged = False
```

Exit Sub

GoPrevError:

MsgBox Err.Description

End Sub

Private Sub SetButtons(bVal As Boolean)

cmdAdd.Visible = bVal

cmdEdit.Visible = bVal

cmdUpdate.Visible = Not bVal

cmdCancel.Visible = Not bVal

cmdDelete.Visible = bVal

cmdClose.Visible = bVal

cmdRefresh.Visible = bVal

cmdNext.Enabled = bVal

cmdFirst.Enabled = bVal

cmdLast.Enabled = bVal

cmdPrevious.Enabled = bVal

End Sub

Products – Code:

Dim WithEvents adoPrimaryRS As Recordset

Dim mbChangedByCode As Boolean

Dim mvBookMark As Variant

Dim mbEditFlag As Boolean

Dim mbAddNewFlag As Boolean

Dim mbDataChanged As Boolean

Private Sub Form_Load()

Dim db As Connection

Set db = New Connection

db.CursorLocation = adUseClient

db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data
Source=C:\stock\STOCKYA.mdb;"

Set adoPrimaryRS = New Recordset

adoPrimaryRS.Open "select

CategoryID,ProductID,ProductName,QuantityPerUnit,UnitPrice,UnitsInStock,UnitsOnOrder
from Products Order by ProductID", db, adOpenStatic, adLockOptimistic

Dim oText As TextBox

'Bind the text boxes to the data provider

For Each oText In Me.txtFields

Set oText.DataSource = adoPrimaryRS

Next

mbDataChanged = False

End Sub

```

Private Sub Form_Resize()
    On Error Resume Next
    lblStatus.Width = Me.Width - 1500
    cmdNext.Left = lblStatus.Width + 700
    cmdLast.Left = cmdNext.Left + 340
End Sub

```

```

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    If mbEditFlag Or mbAddNewFlag Then Exit Sub

```

```

    Select Case KeyCode
        Case vbKeyEscape
            cmdClose_Click
        Case vbKeyEnd
            cmdLast_Click
        Case vbKeyHome
            cmdFirst_Click
        Case vbKeyUp, vbKeyPageUp
            If Shift = vbCtrlMask Then
                cmdFirst_Click
            Else
                cmdPrevious_Click
            End If
        Case vbKeyDown, vbKeyPageDown
            If Shift = vbCtrlMask Then
                cmdLast_Click
            Else
                cmdNext_Click
            End If
        End Select
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)
    Screen.MousePointer = vbDefault
End Sub

```

```

Private Sub adoPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This will display the current record position for this recordset
    lblStatus.Caption = "Record: " & CStr(adoPrimaryRS.AbsolutePosition)
End Sub

```

```

Private Sub adoPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This is where you put validation code
    'This event gets called when the following actions occur
    Dim bCancel As Boolean

```



```

Select Case adReason
Case adRsnAddNew
Case adRsnClose
Case adRsnDelete
Case adRsnFirstChange
Case adRsnMove
Case adRsnRequery
Case adRsnResynch
Case adRsnUndoAddNew
Case adRsnUndoDelete
Case adRsnUndoUpdate
Case adRsnUpdate
End Select

```

```

If bCancel Then adStatus = adStatusCancel
End Sub

```

```

Private Sub cmdAdd_Click()
On Error GoTo AddErr
With adoPrimaryRS
If Not (.BOF And .EOF) Then
mvBookMark = .Bookmark
End If
.AddNew
lblStatus.Caption = "Add record"
mbAddNewFlag = True
SetButtons False
End With

```

```

Exit Sub
AddErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdDelete_Click()
On Error GoTo DeleteErr
With adoPrimaryRS
.Delete
.MoveNext
If .EOF Then .MoveLast
End With
Exit Sub
DeleteErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdRefresh_Click()
'This is only needed for multi user apps
On Error GoTo RefreshErr

```

```

adoPrimaryRS.Requery
Exit Sub
RefreshErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdEdit_Click()
On Error GoTo EditErr

```

```

lblStatus.Caption = "Edit record"
mbEditFlag = True
SetButtons False
Exit Sub

```

```

EditErr:
MsgBox Err.Description
End Sub

```

```

Private Sub cmdCancel_Click()
On Error Resume Next

```

```

SetButtons True
mbEditFlag = False
mbAddNewFlag = False
adoPrimaryRS.CancelUpdate
If mvBookMark > 0 Then
adoPrimaryRS.Bookmark = mvBookMark
Else
adoPrimaryRS.MoveFirst
End If
mbDataChanged = False

```

```

End Sub

```

```

Private Sub cmdUpdate_Click()
On Error GoTo UpdateErr

```

```

adoPrimaryRS.UpdateBatch adAffectAll

```

```

If mbAddNewFlag Then
adoPrimaryRS.MoveLast      'move to the new record
End If

```

```

mbEditFlag = False
mbAddNewFlag = False
SetButtons True
mbDataChanged = False

```

```

Exit Sub
UpdateErr:
MsgBox Err.Description

```

```
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub
```

```
Private Sub cmdFirst_Click()
    On Error GoTo GoFirstError

    adoPrimaryRS.MoveFirst
    mbDataChanged = False

Exit Sub
```

```
GoFirstError:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdLast_Click()
    On Error GoTo GoLastError

    adoPrimaryRS.MoveLast
    mbDataChanged = False

Exit Sub
```

```
GoLastError:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdNext_Click()
    On Error GoTo GoNextError

    If Not adoPrimaryRS.EOF Then adoPrimaryRS.MoveNext
    If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then
        Beep
        'moved off the end so go back
        adoPrimaryRS.MoveLast
    End If
    'show the current record
    mbDataChanged = False

Exit Sub
GoNextError:
    MsgBox Err.Description
End Sub
```

```
Private Sub cmdPrevious_Click()
    On Error GoTo GoPrevError
```



```

If Not adoPrimaryRS.EOF Then adoPrimaryRS.MovePrevious
If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then
    Beep
    'moved off the end so go back
    adoPrimaryRS.MoveFirst
End If
'show the current record
mbDataChanged = False

Exit Sub

GoPrevError:
    MsgBox Err.Description
End Sub

Private Sub SetButtons(bVal As Boolean)
    cmdAdd.Visible = bVal
    cmdEdit.Visible = bVal
    cmdUpdate.Visible = Not bVal
    cmdCancel.Visible = Not bVal
    cmdDelete.Visible = bVal
    cmdClose.Visible = bVal
    cmdRefresh.Visible = bVal
    cmdNext.Enabled = bVal
    cmdFirst.Enabled = bVal
    cmdLast.Enabled = bVal
    cmdPrevious.Enabled = bVal
End Sub

```

CONCLUSION

The database programming are one of the most growing up fields in the computer world, and the applications of it are spreading and entering everywhere in our real life.

Despite of it being one of the best Object Oriented Programming languages The Visual Basic is also one of the most used database programs builders referring to its high control on user interface of databases management and design, I believe that the entire Visual Basic had become more useful and gaining more ground on the application world, but the main reason for me to use Visual Basic is, because I have been very aggressive to learn it and I got some experience in solving database problems using database engines so its advisable to maintain new control through on objects through Visual Basic.

In this project, the Stock control database has been taken with in regards to a small Business Company, and it can apply the same services that is provided here to a more generalized Stock requirements of bigger Companies, which may recommend adding network capabilities.

This program is designed easy enough to use by any public user, security considerations was bared in mind and upgrading capabilities are wide open. All these features fall in favors of the program hoping to improve work in the future.

REFERENCES

- [1] Microsoft Corporation, *Visual Basic 6.0 Programmer's Guide*, Microsoft Press, Washington, 1998.
- [2] John Lewis, *Advanced Visual Basic Programming*, Jerry Blank/SIS, USA, 1998.
- [3] Gordon C., *Database Management*, McGraw-Hill, Singapore, 1986.
- [4] Database System Design.
“<http://www.gogle.com/edu/x893&z00cyp/dat.htm>”
- [5] Microsoft Access.
“http://www.microsoft.com/train_cert”