



**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

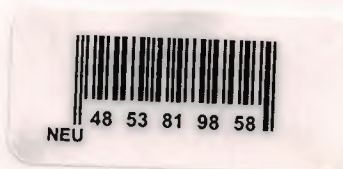
**SOFTWARE DEVELOPMENT FOR CUSTOM  
OPERATIONS USING DELPHI PORGRAMMING**

**Graduation Project  
COM- 400**

**Student: Ahmad Deeb Khaled (992065)**

**Supervisor: Assoc. Prof. Dr Rahib ABIYEV**

**Nicosia - 2003**



## ACKNOWLEDGMENT

*"First, I would like and foremost to thank Allah whom its accomplishment would not have been possible.*

*Second, I would like also to thank my supervisor Assoc. Prof. Dr RAHIB ABIYEV for his invaluable advice and belief in my work and myself over the course of this Graduation Project.*

*Third, I am deeply indebted to my parents for their love and financial support. They have always encouraged me to pursue my interests and ambitions throughout life.*

*Especially my father: Al Haj Deeb Khaled, my uncle: Al Haj Nemer Khaled,  
And my Family*

*Forth, I thank my family for their constant encouragement and support during the preparation of this project.*

*Last but not least I would also like to thank all of my friends especially Ahmad Allan, Ali Almassri, and Ms. Gonul Gokdemir, they were always available for my assistance throughout this project."*

## ABSTRACT



The purpose of this project is the development of custom department system. The main problems which we have come across in custom department information system have been analyzed. The algorithms for holder's registration, cars registration, and calculations of custom cost are described. The main structures and elements of database system for these problems are clarified. The operation principles of each blocks of the information system are modeled in Delphi programming language. The developed system allows to make registration of holders and cars, calculation of tax's and gross price easily and decreasing time response of the system. Over the past decades people have change form special to the public in maintaining records through paper and pen, and now we are evolving into the technology aria.

This project has taken a lot of time and effort to send out a very clear and simple program in Delphi concerning any university. This system has been designed in a way that it would work more speedy than the normal record keeping system.

## TABEL OF CONTENTS

<b>ACKNOWLEDGMENT</b>	<b>i</b>
<b>ABSTRACET</b>	<b>ii</b>
<b>TABEL OF CONTENTS</b>	<b>iii</b>
<b>LIST OF TABLES</b>	<b>v</b>
<b>LIST OF FIGURES</b>	<b>vi</b>
<b>INTRODUCTION</b>	<b>1</b>
<b>CHAPTER ONE: COMPUTER RELIZATION OF DATABASE</b>	<b>2</b>
1.1. Introduction	2
1.2. Database Structure	3
1.3. Choosing a RelationshipDatabase	4
1.4. Nonprofit Developed Database Products	4
1.5. Hosted Database Application Services	5
1.6. Primary and Foreign Keys	7
1.6.1. Define Primary Key Attribute	7
1.6.2. Composite Key	8
1.6.3. Artificial Keys	9
1.6.4. Primary Key Migration	9
1.6.5. Define Key Attribute	9
1.6.6. Validate Keys and Relationships	10
1.7. Foreign Key	10
1.7.1. Identifying Foreign Keys	10
1.7.2. Foreign Key Ownership	11
1.7.3. Diagramming Foreign Keys	11
1.8. Introduction to Relationship Database Design	11
1.9. Goals of Relational Database Design	11
1.10. Rules of Relational Database Design	12
1.10.1. The Rules of Tables	12
1.10.2. The Rules of Uniqueness and Keys	12
1.11. Foreign Keys and Domains	13
1.12. Normalization and Normal Forms	13
1.12.1. First Normal Form	14
1.12.2. Second Normal Form	14
1.12.3. Third Normal Form	14
1.13. Demoralization Purposely Violating the Rules	15
1.14. Integrity Rules	15
1.15. Overall Rules	15
1.16. Database Specific Rules	16
1.17. Examining the Type of Relationships	16
1.17.1. One to Many	16
1.17.2. One to One	17
1.17.3. Many to Many	18
<b>CHAPTER TWO: STRUCTURE OF THE PROGRAM</b>	<b>19</b>
2.1. Program Structure	19
2.1.1. Explanation of the Block Diagram	19
2.2. Program Flow Chart	20
2.2.1. Explanation of Add New Holder Flow-Chart	20
2.2.2. Explanation of Records Flow-Chart	20



2.2.3. Explanation of Calculations Total Cost Flow-Chart	21
2.2.4. Explanation of the Search Flow-Chart	21
2.2.5. Explanation of the Cost Report Flow-Chart	21
<b>CHAPTER THREE: DEVELOPMENT OF CUSTOM INFORMATION SYSTEM</b>	<b>52</b>
3.1. Determine the Purpose of Database	52
3.2. Determine the Field You Need in the Database	52
3.3. Determine the Table You Need in the Database	53
3.4. Determine Which Table Each Field Belongs to	53
3.5. Identify the Field or Fields with Unique Value in Each Record	54
3.6. Determine the Relationships Between Tables	54
3.7. Refining Design	54
3.8. Designing Project Database Structure	55
3.9. Define Relationships Between Tables	57
3.10. Layout of the Application	58
3.10.1. Main Menu Screen	58
3.10.2. Cars Information Screen	62
3.10.3. Order Payments and Options Screen	62
3.11. Buttons Function	64
<b>CONCLUSION</b>	<b>65</b>
<b>APPINDEX</b>	<b>66</b>
<b>REFERENCES</b>	<b>95</b>

## LIST OF FIGURES

### Figure

1.1	Entities with Key Attributes	9
2.1	Structure of the Program	18
2.2	Main Menu Flow Chart	21
2.3	Enter Main Menu Flow Chart	22
2.4	Add New Holders in Public or Special Cars Flow Chart	23
2.5	Add New Car Information Public or Special Flow Chart	24
2.6	Calculate Car Options Public or Special Cars Flow Chart	25
2.7	Report Public and Special Cars Information Flow Chart	26
2.8	Add New Holders in Tourism, Transportation and Others Flow Chart	27
2.9	Add New Car Information in Tourism, Transportation and Others Flow Chart	28
2.10	Calculate Car Options Tourism, Transportation and Others Flow Chart	29
2.11	Report for Tourism, Transportation and Others Information Flow Chart	30
2.12	Change Entry From Special to Public Flow Chart	31
2.13	Tax to Change Entry From Special to Public Flow Chart	32
2.14	Option Menu Flow Chart	33
2.15	Change Password Flow Chart	34
2.16	Set Option Menu Flow Chart	35
2.17	Set Options for Public Car Flow Chart	36
2.18	Set Options for Special Cars Flow Chart	37
2.19	Set Options for Tourism, Transportation and Others Cars Flow Chart	38
2.20	Report for Public and Special Cars Information Flow Chart	39
2.21	Report for Tourism, Transportation and Others Cars Flow Chart	40
2.22	Search Options Rate Menu Flow Chart	41
2.23	Search Rate Option for Public Cars Flow Chart	42
2.24	Search Rate Option For Special Cars Flow Chart	43
2.25	Search Rate Option For Others Cars Flow Chart	44
2.26	Report Main Menu Flow Chart	45
2.27	Display All Public Cars with Holders Flow Chart	46
2.28	Display All Special Cars with Holders Flow Chart	47
2.29	Display All Others Cars with Holders Flow Chart	48
2.30	Display All was Changed with Holders	49
2.31	Display All Public, Special and Other Cars Options with Holders	50
3.1	Relationships	57
3.2	Main Menu	57
3.3	Holder Information Screen	58
3.4	Change Password	58
3.5	Set Rate Options and Cars Model	59
3.6	Rate of Options	59
3.7	Show All Holders was Registered with Car Details for Each Entry	60
3.8	Show All Cars Options in All Entries	60
3.9	Car Information	61
3.10	Calculate Custom Cost	62
3.11	Options	62

## LIST OF TABLES

### Table

1.1	Example of Composite Key Work	8
3.1	Option for All Cars	54
3.2	Holder Information	55
3.3	Rate Option for All Car	55
3.4	Car Option	56

## INTRODUCTION

The aim of the project is the development of custom department' information system using Delphi programming. 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 Development.

In this project, I use the one of the programming language that we are learned in our university - Delphi programming language. In this language there are many things that we can use to create any kind of project. But in this project I use some standard components and database components to create this project. In this language there is special procedure called Database Desktop to create some tables that used in the project. We will see this later, regarding this program, which basically divided, into tow main sections: Registration and Calculation of custom cost. The section for Registration, which consist of holder's information, cars information. Another section is the Calculation, which includes calculation of custom cost and some tax cost of change. Each member has been assigned a special form in this program. These forms are updated consistently depending on his working hours. Another important think about this project is to searching by holder name or file number in this project we are called other car by tourism, transportation, and the car using in construction...Etc



## CHAPTER ONE

### COMPUTER REALIZATION OF DATABASE

#### 1.1 Introduction

Delphi includes a number of specialty applications designed to help you work more efficiently. The following links provide easy access to the Help systems for these tools. The Delphi Enterprise edition includes all of the tools described below. Delphi Professional does not offer SQL Builder, SQL Monitor, or Team Source. Delphi Standard includes only the Image Editor. The Image Editor lets you create, open, and save icons, cursors, and bitmaps for use in your applications. Insight provides debugging information about window classes, windows, and messages. You can use this tool to examine how any application creates classes and endows, and monitor how windows send and receive messages. (Available in the enterprise and Professional editions only.) Borland Database Engine (BDE) is the 32-bit Windows-based core database engine and connectivity software behind Borland products, as well as Paradox for Windows and usual dBase for Windows. This Help file offers a reference to the BDE's features and language elements. (Available in the Enterprise and Professional editions only.) The BDE Administrator lets you configure the Borland Database Engine (BDE), configure numerous database drivers, create and delete ODBC drivers, and create and maintain database aliases. (Available in the Enterprise and Professional editions only.) Borland SQL Links for Windows (32-bit version) is a set of BDE-hosted driver connections to database servers. By creating queries, SQL Links emulates full navigation capabilities, enabling users to access and manipulate data in SQL databases by using convenient features in Borland applications. (Available in the Enterprise and Professional editions only.) Local SQL (online reference). Local SQL is the subset of the SQL-92 specification used to access dBase, Paradox, and FoxPro tables. On receiving local SQL statements from front-end applications, the Borland Database Engine (BDE) translates the statements into BDE API functions. (Available in the Enterprise and Professional editions only.) Data Pump lets you move data (both database schema and content) between databases. (Available in the Enterprise and Professional editions only.) Database Explorer is a hierarchical database browser with editing capabilities, letting you browse and edit database server-specific schema

objects, including tables, fields, stored procedure definitions, triggers, and indexes. (Available in the Enterprise and Professional editions only.)

SQL Builder lets you visually and interactively create and execute SQL queries, and serves as a tool for SQL. (Available in the Enterprise edition only.) SQL Monitor lets you view statement calls made through SQL Links to a remote server or through the ODBC socket to an ODBC data source. (Available in the Enterprise edition only.)

The Team Source workflow management tool uses a parallel model of source control to help with the management and coordination of work in a shared development environment. Note: The Team Source tool, available only in the Enterprise edition, is a separate product and requires a separate installation

## 1.2 Database Structure

First thing as we know Delphi's support for database applications is one of the key features of the programming environment. Many programmers spend most of their time writing data-access code, which needs to be the most robust portion of a database application. You can create very complex database applications, starting from a blank form or one generated by Delphi's Database form wizard. On a computer, permanent data-including database data is always stored in files. There are several techniques you can use to accomplish this storage. Delphi can use both approaches; or more precisely, it uses a custom approach that works well with both underlying structures. You always refer to a database with its name or an alias, which is a sort of a nickname of a database, but this reference can be to a database file or to a directory containing files with tables. The approach used by Delphi depends on the database format you are using:

- \* Paradox and dBase tables define databases as directories and each table as a separate file or actually multiple files if you include indexes.

- \* Access, interBase, and most SQL server use a single huge file containing the entire database, with all tables and indexes.



### **1.3 Choosing a Relationship Database**

One of the most challenging technology problems that any nonprofit organization faces is how to manage its relationship database. It's a particular challenge for grassroots groups who are lucky to have a development director, let alone a database administrator! The Time, having a database system that works well with a minimum of fuss can spell the difference between a sustainable base of membership/major donor income and chronic financial crisis. This article provides an overview of four options for developing an effective membership database system. We're not going to strongly advocate for any one of them; each one has pros and cons, and each one might be an appropriate choice for a particular organization. The four options are: Commercial off-the-shelf solutions such as Paradigm, Raiser's Edge, and others; Nonprofit products such as ebase and ODB; Hosted database application services such as DonorLinkIT and e-Tapestry; "Homebrewed" databases, typically in FileMaker Pro or Microsoft Access.

### **1.4 Nonprofit-Developed Database Products**

In 1998, Desktop Assistance (now TechRocks), a nonprofit technology assistance provider from Helena, MT, released ebase, a membership database solution aimed at grassroots nonprofits. In March 2001, TechRocks released ebase 2.0, which is a from-the-ground-up rewrite of ebase, and includes many welcome improvements. Ebase costs nothing to download and get running, but to use it for anything more than simple testing requires that you purchase FileMaker Pro, about \$170. To support more than a couple of simultaneous users, you may require FileMaker Pro Server (~\$900). FileMaker does have a software donation program, now run through Gifts-in-Kind that will allow you receive additional licenses of FileMaker Pro after you purchase one licensed copy at retail. Expect to pay a \$125/year Gifts-In-Kind membership fee, plus about \$15/copy for FileMaker products. File Maker goosefoot donate File Maker Server. Because ebase is based in FileMaker Pro, it runs on both Macs and PCs. Ebase was designed by and for grassroots nonprofits, and thus incorporates a great deal of good thinking about the typical business processes and tasks of small nonprofit advocacy groups. In addition, ebase can be customized quite extensively--the entire program can be modified by skilled FileMaker users. Its low cost, high power and nonprofit-friendliness have made ebase very popular in the North West conservation

movement. Ebase does have some downsides, though. Because ebase is developed by a nonprofit, and not by a professional software development firm, its user interface is a bit cluttered, and it can be a bit confusing to novice users. Although ebase includes a very powerful "import" routine, migrating existing data into ebase often requires assistance from a skilled consultant. And unfortunately, the community of skilled ebase consultants is still extremely small. This is primarily because FileMaker has far fewer users than competing products such as Microsoft Access. However, ebase does have an enthusiastic and increasingly knowledgeable peer-user community that is supported by the knowledgeable folk skate Tech Rocks. The bottom line is that ebase, despite its low up-front cost, isn't free. Like all database products, there can be a significant need for skilled database consultants to help with startup and customization, and to troubleshoot if things go awry. Ebase is a good choice for organizations with relatively high internal technology skill levels that don't want to spend much cash upfront on a database. Ebase is also a good choice if your organization has the need--and the expertise--to extensively customize your membership database. For a more in-depth comparison of the differences between ebase 1.0 and ebase 2.0, along with advice for groups currently using ebase 1.0, see "Comparing ebase v1 and ebase v2".

A simpler nonprofit-developed relationship management database is ODB developed by the folks at Organizers' Collaborative. It runs on Windows only, and offers a simple, configurable database tool that is less customizable than ebase, but considerably easier to get up and running.

## 1.5 Hosted Database Application Services

This is the newest--and perhaps the most intriguing--type of database product. "Hosted" databases are database programs that reside entirely on the servers of an "Application Service Provider" (ASP) company. There are several nonprofit-oriented donor/member database services that have started up in the past year. The two with which we're most familiar are Donor Link IT and e-Tapestry. You purchase ASP-based databases as a service rather than as a product. There's no software to purchase or install on your machines--all you need is a Web browser and an Internet connection (56k works fine, although obviously a high-speed connection is better). While ASP databases can be customized quite a bit, the fundamental workflow can't be modified as extensively as ebase can permit. However, because these are hosted applications, they



are upgraded often, and upgrades are automatically and seamlessly rolled out to all users. Hosted databases have several advantages: Hosted databases can be easily accessed by multiple users in multiple locations. This is extremely difficult with most other database solutions.

Hosted databases can include features that allow your members to interact with you through your database. For example, DonorLinkIT allows you to create and distribute surveys to your members, and for responses to be incorporated directly into your database for immediate use. e-Tapestry has a module that can allow donors to log in and view their donation history with your organization. Hosted databases can integrate directly with ecommerce software, allowing easy integration of online giving with your membership database.

Hosted databases make it easy to send out bulk email such as an email newsletter to your members. Power users can even customize the content of the emails so that users receive content that is customized based on information in your database.

Hosted databases take care of all the software installation, maintenance for you. Even better, they also take care of backups. And because there's no data stored on your machines, you never have to worry about how to recover your data if your machines crash. The cost of these hosted database services generally depends on the number of records in your database. E-Tapestry has a FREE level of service for groups with databases with 500 records or less, and only \$30/month for groups with 501-1000 records. DonorLink has base pricing of \$99/month for databases with up to 5000 records. Both offer additional services such as e-commerce integration. You should also anticipate spending some time and money to import your existing data into the new hosted database product, and to customize your new database to your organization's business processes.

There are several downsides to hosted database solutions:

One downside is the relatively high ongoing cost. Most grassroots environmental groups should anticipate spending \$100-200/month for a hosted database solution. This is quite a bit more than the upfront cost of a base, but it is quite a bit cheaper than most ordinary commercial software, and quite a bit cheaper than paying a consultant to write a custom database.

Hosted database solutions are not as highly customizable as ebase or a homebrewed database. But they can be customized enough to meet many organizations' needs. Most hosted database providers roll out software upgrades every three to six months, and upgrades are automatically delivered to all users at no additional cost.

Hosted databases are not as fast as databases that reside on your machine or your network. This is generally not a huge problem, but it can slow down large data entry projects. And if you don't have reliable "always-on" Internet access, you may find it frustrating to work with an online database tool.

Both e-Tapestry and DonorLink are relatively new products, but we've been very impressed by what we've seen. e-Tapestry is particularly attractive to smaller groups with <1000 members, as it's free for small groups. DonorLinkIT is notable for its powerful email distribution and Web-based surveying and response mechanisms.

The ongoing cost of hosted database solutions is significant, but that cost has to be weighed against the time and expense of developing your own system or even that of customizing a low-upfront cost system such as ebase. If you don't need the total customizability of ebase or a custom solution, and would rather spend some cash than your precious time, then a hosted database solution might be worth investigating.

## **1.6 Primary and Foreign Keys**

Primary and foreign keys are the most basic components on which relational theory is based. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an association between two entities. The next step in building the basic data model to

1. identify and define the primary key attributes for each entity
2. validate primary keys and relationships
3. migrate the primary keys to establish foreign keys

### **1.6.1 Define Primary Key Attributes**

The primary key is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.



To qualify as a primary key for an entity, an attribute must have the following properties:

it must have a non-null value for each instance of the entity

the value must be unique for each instance of an entity

the values must not change or become null during the life of each entity instance

In some instances, an entity will have more than one attribute that can serve as a primary key. Any key or minimum set of keys that could be a primary key is called a *candidate key*. Once candidate keys are identified, choose one, and only one, primary key for each entity. Choose the identifier most commonly used by the user as long as it conforms to the properties listed above. Candidate keys which are not chosen as the primary key are known as alternate keys.

An example of an entity that could have several possible primary keys is Employee. Let's assume that for each employee in an organization there are three candidate keys: Employee ID, Social Security Number, and Name.

Name is the least desirable candidate. While it might work for a small department where it would be unlikely that two people would have exactly the same name, it would not work for a large organization that had hundreds or thousands of employees. Moreover, there is the possibility that an employee's name could change because of marriage. Employee ID would be a good candidate as long as each employee was assigned a unique identifier at the time of hire. Social Security would work best since every employee is required to have one before being hired.

### 1.6.2 Composite Keys

Sometimes it requires more than one attribute to uniquely identify an entity. A primary key that made up of more than one attribute is known as a composite key. Table 1.1 shows an example of a composite key. Each instance of the entity Work can be uniquely identified only by a composite key composed of Employee ID and Project ID.

Employee ID	Project ID	Hours_Worked
01	01	200
01	02	120
02	01	50
02	03	120
03	03	100
03	04	200

Table 1.1 Example of Composite Key Work

### 1.6.3 Artificial Keys

An artificial key's one that has no meaning to the business or organization. Artificial keys are permitted when no attribute has all the primary key properties, or the primary key is large and complex.

### 1.6.4 Primary Key Migration

Dependent entities, entities that depend on the existence of another entity for their identification, inherit the entire primary key from the parent entity. Every entity within a generalization hierarchy inherits the primary key of the root generic entity.

### 1.6.5. Define Key Attributes

Once the keys have been identified for the model, it is time to name and define the attributes that have been used as keys. There is no standard method for representing primary keys in ER diagrams. For this document, the name of the primary key followed by the notation (PK) is written inside the entity box. An example.



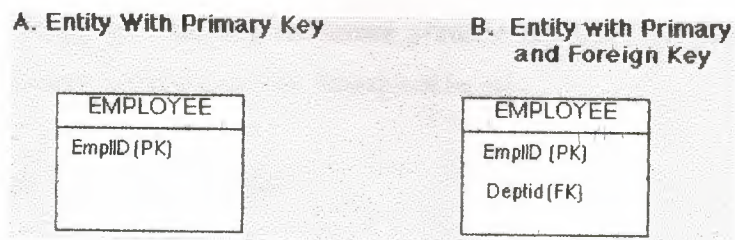


Figure 1.1: Entities with Key Attributes

### 1.6.6 Validate Keys and Relationships

Basic rules governing the identification and migration of primary keys are:

- Every entity in the data model shall have a primary key whose values uniquely identify entity instances.
- The primary key attribute cannot be optional (i.e., have null values).
- The primary key cannot have repeating values. That is, the attribute may not have more than one value at a time for a given entity instance is prohibited. This

is known as the No Repeat Rule.

- Entities with compound primary keys cannot be split into multiple entities with simpler primary keys. This is called the Smallest Key Rule.
- Two entities may not have identical primary keys with the exception of entities within generalization hierarchies.
- The entire primary key must migrate from parent entities to child entities and from super type, generic entities, to subtypes, category entities.

### 1.7 Foreign Keys

A foreign key is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data (called referential integrity) and for navigating between different instances of an entity. Every relationship in the model must be supported by a foreign key.

#### 1.7.1 Identifying Foreign Keys

Every dependent and category (subtype) entity in the model must have a foreign key for each relationship in which it participates. Foreign keys are formed in dependent

and subtype entities by migrating the entire primary key from the parent or generic entity. If the primary key is composite, it may not be split.

### **1.7.2 Foreign Key Ownership**

Foreign key attributes are not considered to be owned by the entities to which they migrate, because they are reflections of attributes in the parent entities. Thus, each attribute in an entity is either owned by that entity or belongs to a foreign key in that entity.

If the primary key of a child entity contains all the attributes in a foreign key, the child entity is said to be "identifier dependent" on the parent entity, and the relationship is called an "identifying relationship." If any attributes in a foreign key do not belong to the child's primary key, the child is not identifier dependent on the parent, and the relationship is called "non identifying."

### **1.7.3 Diagramming Foreign Keys**

Foreign keys attributes are indicated by the notation (FK) beside them. An example is shown in Figure 2 (b) above.

## **1.8 Introduction to Relational Database Design**

Many people believe that Access is such a simple product to use, that database design is something they don't need to worry about. I couldn't disagree more! Just as a house without a foundation will fall over, a database with poorly designed tables and relationships will fail to meet the needs of the users.

### **1.9 Goals of Relational Database Design**

The number one goal of relational database design is to, as closely as possible, develop a database that models some real-world system. This involves breaking the real-world system into tables and fields, and determining how the tables relate to each other. Although, on the surface, this might appear to be a trivial task, it can be an extremely cumbersome process to translate a real-world system into tables and fields. A properly designed database has many benefits. The process of adding, editing, deleting,



and retrieving table data is greatly facilitated by a properly designed database. Reports are easy to build. Most importantly, the database becomes easy to modify and maintain.

## **1.10 Rules of Relational Database Design**

To adhere to the relational model, certain rules must be followed. These rules determine what is stored in a table, and how the tables are related.

### **1.10.1 The Rules of Tables**

Each table in a system must store data about a single entity. An entity usually represents a real-life object or event. Examples of objects are customers, employees, and inventory items. Examples of events include orders, appointments, and doctor visits.

### **1.10.2 The Rules of Uniqueness and Keys**

Tables are composed of rows and columns. To adhere to the relational model, each table must contain a unique identifier. Without a unique identifier, it becomes programmatically impossible to uniquely address a row. You guarantee uniqueness in a table by designating a primary key, which is a single column or a set of columns that uniquely identifies a row in a table. Each column or set of columns in a table that contains unique values is considered a candidate key. One candidate key becomes the *primary key*. The remaining candidate keys become *alternate keys*. A primary key made up of one column is considered a simple key. A primary key comprised of multiple columns is considered a composite key. It is generally a good idea to pick a primary key that is Minimal (has as few columns as possible) Stable (rarely changes) Simple (familiar to the user) Following these rules greatly improves the performance and maintainability of your database application, particularly if you are dealing with large volumes of data.

Consider the example of an employee table. An employee table is generally composed of employee-related fields such as social security number, first name, last name, hire date, salary, and so on. The combination of the first name and the last name fields could be considered a primary key. This choice might work, until the company hires two employees with the same name. Although the first and last names could be combined with additional fields to constitute uniqueness (for example, hire date), this

would violate the rule of keeping the primary key minimal. Furthermore, an employee might get married and her last name might change.

Using a name as the primary key violates the principle of stability. The social security number might be a valid choice, but a foreign employee might not have a social security number. This is a case where a derived, rather than a natural, primary key is appropriate. A derived key is an artificial key that you create. A natural key is one that is already part of the database.

I would suggest adding EmployeeID as an AutoNumber field. Although the field would violate the rule of simplicity (because an employee number is meaningless to the user), it is both small and stable. Because it is numeric, it is also efficient to process. In fact, I use AutoNumber fields (an identity field in SQL Server) as primary keys for most of the tables that I build.

### **1.11 Foreign Keys and Domains**

A foreign key in a table is the field that relates to the primary key in a second table. For example, the CustomerID is the primary key in the Customers table. It is the foreign key in the Orders table. A *domain* is a pool of values from which columns are drawn. A simple example of a domain is the specific data range of employee hire dates. In the case of the Order table, the domain of the CustomerID column is the range of values for the CustomerID in the Customers table.

### **1.12 Normalization and Normal Forms**

One of the most difficult decisions that you face as a developer is what tables to create, and what fields to place in each table, as well as how to relate the tables that you create. *Normalization* is the process of applying a series of rules to ensure that your database achieves optimal structure. Normal forms are a progression of these rules. Each successive normal form achieves a better database design than the previous form did. Although there are several levels of normal forms, it is generally sufficient to apply only the first three levels of normal forms. They are described in the following sections.

#### **1.12.1 First Normal Form**

To achieve first normal form, all columns in a table must be atomic. This means, for example, that you cannot store first name and last name in the same field. The



reason for this rule is that data becomes very difficult to manipulate and retrieve if multiple values are stored in a single field. Using the full name as an example, it would become impossible to sort by first name or last name independently if both values are stored in the same field. Furthermore, extra work must be done to extract just the first name or the last name from the field. Another requirement for first normal form is that the table must not contain repeating values. An example of repeating values is a scenario in which Item1, Quantity1, Item2, Quantity2, Item3, and Quantity3 fields are all found within the Orders table. This design introduces several problems. What if the user wants to add a fourth item to the order? Furthermore, finding the total ordered for a product requires searching several columns. In fact, all numeric and statistical calculations on the table become extremely cumbersome. The alternative, achieves first normal form. Notice that each item ordered is located in a separate row.

### **1.12.2 Second Normal Form**

To achieve second normal form, all non-key columns must be fully dependent on the primary key. In other words, each table must store data about only one subject. Notice the table includes information about the order (OrderID, CustomerID, and OrderDate) and information about the items being ordered (Item and Quantity). To achieve second normal form, this data must be broken into two tables, an order table and an order detail table. The process of breaking the data into two tables is called decomposition. It is considered to be *non-loss* decomposition because no data is lost during the decomposition process. Once the data is broken into two tables, you can easily bring the data back together by joining the two tables in a query. These two tables achieve second normal form.

### **1.12.3 Third Normal Form**

To attain third normal form, a table must meet all the requirements for first and second normal form, and all non-key columns must be mutually independent. This means that you must eliminate any calculations, and you must break out data into lookup tables.

An example of a calculation stored in a table is the product of price multiplied by quantity. Rather than storing the result of this calculation in the table, instead you

would generate the calculation in a query, or in the control source of a control on a form or a report.

### **1.13 Demoralization Purposely Violating the Rules**

Although the developer's goal is normalization, there are many times when it makes sense to deviate from normal forms. This process is referred to as demoralization. The primary reason for applying demoralization is to enhance performance. An example of when demoralization might be the preferred tact could involve an open invoices table and a summarized accounting table. It might be impractical to calculate summarized accounting information for a customer when it is needed. The summary calculations are maintained in a summarized accounting table so that they are easily retrieved as needed. Although the upside of this scenario is improved performance, the downside is that the summary table must be updated whenever changes are made to the open invoices. This imposes a definite trade-off between performance and maintainability. You must decide whether the trade-off is worth it.

If you decide to demoralize, document your decision. Make sure that you make the necessary application adjustments to ensure that the demoralized fields are properly maintained. Finally, test to ensure that performance is actually improved by the demoralization process.

### **1.14 Integrity Rules**

Although integrity rules are not part of normal forms, they are definitely part of the database design process. Integrity rules are broken into two categories. They include overall integrity rules and database-specific integrity rules.

### **1.15 Overall Rules**

The two types of overall integrity rules are referential integrity rules and entity integrity rules. Referential integrity rules dictate that a database does not contain any orphan foreign key values. This means that Child rows cannot be added for parent rows that do not exist. In other words, an order cannot be added for a nonexistent customer. A primary key value cannot be modified if the value is used as a foreign key in a child table. This means that a CustomerID cannot be changed if the orders table contains rows with that CustomerID.



A parent row cannot be deleted if child rows are found with that foreign key value. For example, a customer cannot be deleted if the customer has orders in the order table. Entity integrity dictates that the primary key value cannot be Null. This rule applies not only to single-column primary keys, but also to multi-column primary keys. In fact, in a multi-column primary key, no field in the primary key can be Null. This makes sense because, if any part of the primary key can be Null, the primary key can no longer act as a unique identifier for the row. Fortunately, Access does not allow a field in a primary key to be Null.

### **1.16 Database Specific Rules**

The other set of rules applied to a database are not applicable to all databases, but are, instead, dictated by business rules that apply to a specific application. Database-specific rules are as important as overall integrity rules. They ensure that only valid data is entered into a database. An example of a database-specific integrity rule is that the delivery date for an order must fall after the order date.

### **1.17 Examining the Types of Relationships**

Three types of relationships can exist between tables in a database: one-to-many, one-to-one, and many-to-many. Setting up the proper type of relationship between two tables in your database is imperative. The right type of relationship between two tables ensures Data integrity optimal performance Ease of use in designing system objects the reasons behind these benefits are covered throughout this chapter. Before you can understand the benefits of relationships, though, you must understand the types of relationships available.

#### **1.17.1 One-to-Many**

A one-to-many relationship is by far the most common type of relationship. In a one-to-many relationship, a record in one table can have many related records in another table. A common example is a relationship set up between a Customers table and an Orders table. For each customer in the Customers table, you want to have more than one order in the Orders table. On the other hand, each order in the Orders table can belong to only one customer. The Customers table is on the one side of the relationship, and the Orders table is on the many side. In order for this relationship to be

implemented, the field joining the two tables on the one side of the relationship must be unique.

In the Customers and Orders tables' example, the CustomerID field that joins the two tables must be unique within the Customers table. If more than one customer in the Customers table has the same customer ID, it is not clear which customer belongs to an order in the Orders table. For this reason, the field that joins the two tables on the one side of the one-to-many relationship must be a primary key or have a unique index. In almost all cases, the field relating the two tables is the primary key of the table on the one side of the relationship. The field relating the two tables on the many side of the relationship is called a foreign key.

### **1.17.2 One-to-One**

In a one-to-one *relationship*, each record in the table on the one side of the relationship can have only one matching record in the table on the many side of the relationship. This relationship is not common and is used only in special circumstances. Usually, if you have set up a one-to-one relationship, you should have combined the fields from both tables into one table. The following are the most common reasons why you should create a one-to-one relationship: The amount of fields required for a table exceeds the number of fields allowed in an Access table. Certain fields that are included in a table need to be much more secure than other fields included in the same table. Several fields in a table are required for only a subset of records in the table.

The maximum number of fields allowed in an Access table is 255. There are very few reasons why a table should ever have more than 255 fields. In fact, before you even get close to 255 fields, you should take a close look at the design of your system. On the rare occasion when having more than 255 fields is appropriate, you can simulate a single table by moving some of the fields to a second table and creating a one-to-one relationship between the two tables.

The second reason to separate into two tables data that logically would belong in the same table involves security. An example is a table containing employee information. Certain information, such as employee name, address, city, state, ZIP Code, home phone, and office extension, might need to be accessed by many users of the system. Other fields, including the hire date, salary, birth date, and salary level, might be highly confidential. Field-level security is not available in Access. You can



simulate field-level security by using a special attribute of queries called Run with Owner's permissions. "Advanced Query Techniques."

The alternative to this method is to place all the fields that can be accessed by all users in one table and the highly confidential fields in another. Only a special Admin user (a user with special security privileges, not one actually named Admin) is given access to the table containing the confidential fields. ActiveX Data Objects (ADO) code is used to display the fields in the highly confidential table when needed. This is done using a query with Run with Owner's permissions, based on the special Admin user's permission to the highly secured table. "Advanced Security Techniques."

If your application utilizes data stored in a SQL Server database, you can use views to easily accomplish the task of implementing field-level security. In such an environment, the process of splitting the data into two tables is unnecessary. The last situation in which you would want to define one-to-one relationships is when certain fields in a table are going to be used for only a relatively small subset of records. An example is an Employee table and a Vesting table. Certain fields are required only for employees who are vested. If only a small percentage of a company's employees are vested, it is not efficient, in terms of performance or disk space, to place all the fields containing information about vesting in the Employee table. This is especially true if the vesting information requires a large volume of fields. By breaking the information into two tables and creating a one-to-one relationship between them, you can reduce disk-space requirements and improve performance. This improvement is particularly pronounced if the Employee table is large.

### **1.17.3 Many-to-Many**

In a many-to-many relationship, records in both tables have matching records in the other table. A many-to-many relationship cannot be directly defined in Access; you must develop this type of relationship by adding a table called a junction table. The junction table is related to each of the two tables in one-to-many relationships. An example is an Orders table and a Products table. Each order probably will contain multiple products, and each product is found on many different orders. The solution is to create a third table called Order Details. The Order Details table is related to the Orders table in a one-to-many relationship based on the OrderID field. It is related to the Products table in a one-to-many relationship based on the ProductID field.

## CHAPTER TWO

### STRUCTURE OF THE PROGRAM

#### 2.1 Program Structure

The program includes the following blocks figure 2.1

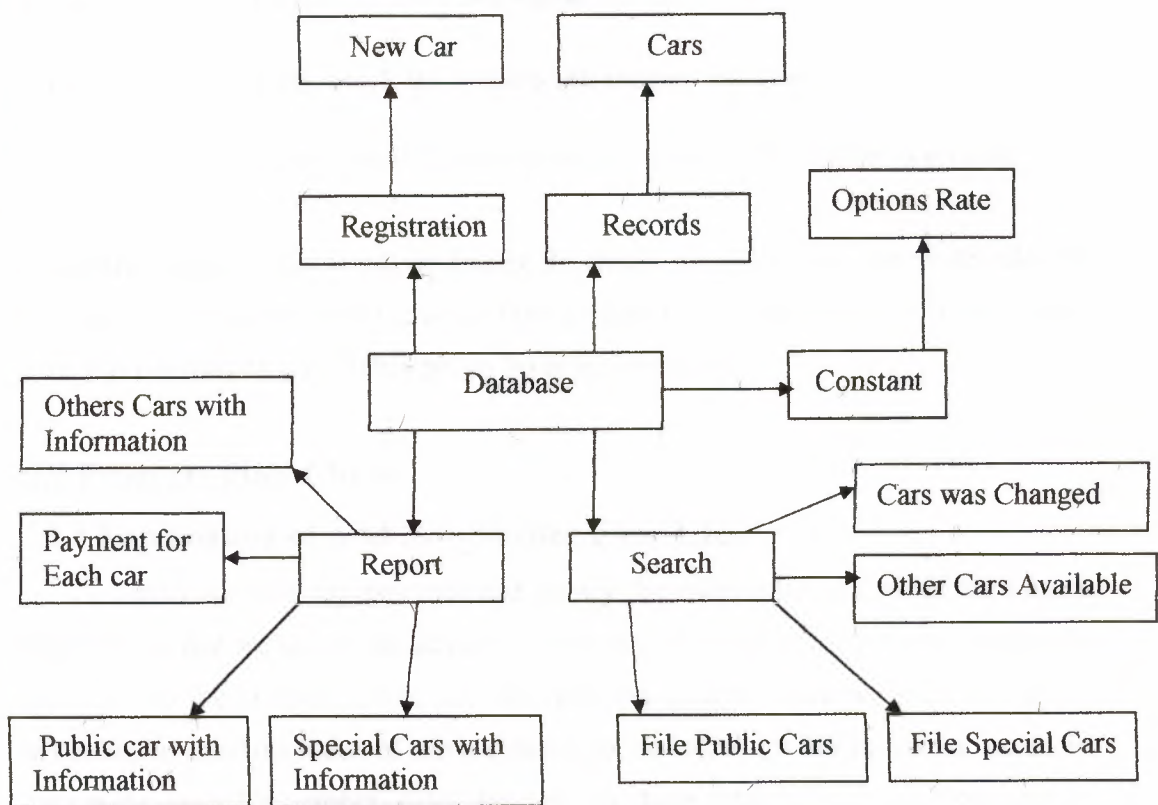


Figure 2.1 Structure of the Program

##### 2.1.1 Explanation of the Block Diagram

- **Registration Block:** In this block we enter the holder's information. It is realized by choosing the new car block.

- **Records Block:** In this block we enter the information for the car by choosing the modes.

- **Search Block:** We use this block to search cars information and we use it to search for file number of the car. We use it also to search for whole car of holders with all information. For this task the program gets data from the user compares it with the data which in database, if the program finds it, then the rest of the data information will appear on the monitor. And also we can appear the whole car with details and.

- **Report Block:** In this block the holder's information and cars information that same file number with the total cost of custom in the any selected file number is printed,

**Constant Block:** In this block we display the model car of the year (the model allowed to enter to the country) and the cost of the options rate for each entrance type in each year that depends on the country policy on orders available.

## **2.2 Program Flow Chart:**

### **2.2.1 Explanation of Add New Holder Flow-Chart:**

Firstly we open the data base and display the information to register the holders details, after that we save these details, the file number of the holder (A) that entered is not equal to the number in the data base then the program generate the file number according to some operation of number data type, and we know this procedure is no tow cars at the same file number. After that we save these details. We want to register car details (A), at the same file number this doing on deferent form, the procedure repeat it self for each different entry type for the new car, as shown the flow chart for each process (figure 2.4) below.

### **2.2.2 Explanation of Records Flow-Chart:**

At the beginning the user should enter the information about the holder, and the form number is given by the each car has form number that is manufacturing company



pot it according their system, the file number impossible to find tow number the same it impossible to save these tow number.

### **2.2.3 Explanation of Calculates Total Cost Flow-Chart**

At the beginning as we know the formula to get the total cost of the car, each manufacturing make some option in their product and available technical, some country take money for these options if found it in the entry car, each option has different custom value to another (possible more than one option at same custom value), for each car has product cost they was put it up to type of development technical. For the option rate from the car cost, the result is what the car have option multiply the cost of car, suppose total car cost is 200\$ and has one option rate of it 20% that's mean.

Total Custom Cost =  $(20 * 200 / 100)$  and so on,

We can see it in (figure 2.6), and the same procedure will used when we calculate the for all option available,

### **2.2.4 Explanation of the Search Flow Chart**

The required data, which is entered to the program to search, will be checked to whether the data is matched or not according to the database. If the data is not matched then alert message will occurs otherwise the program continue the process that shows the rest information. We can see it in (figure 2.21) below.

### **2.2.5 Explanation of the Cost Report Flow Chart**

some times the country put some tax's for each entry car like open file cost and whatever these also entered on the total added to what we have after calculate the car options, At the beginning the user should enter the tax's for the procedure, and net cost for the car we see it in (figure 2.7).

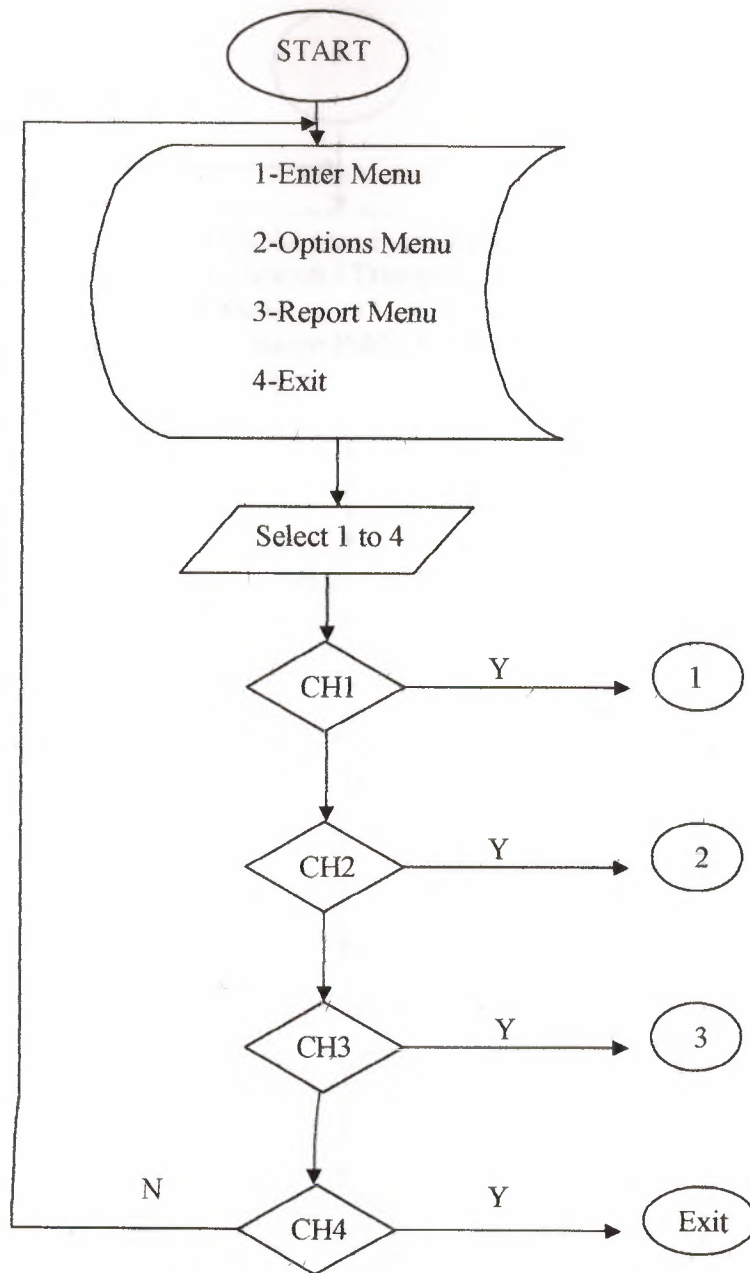


Figure 2.2 Main menus Flow Chart

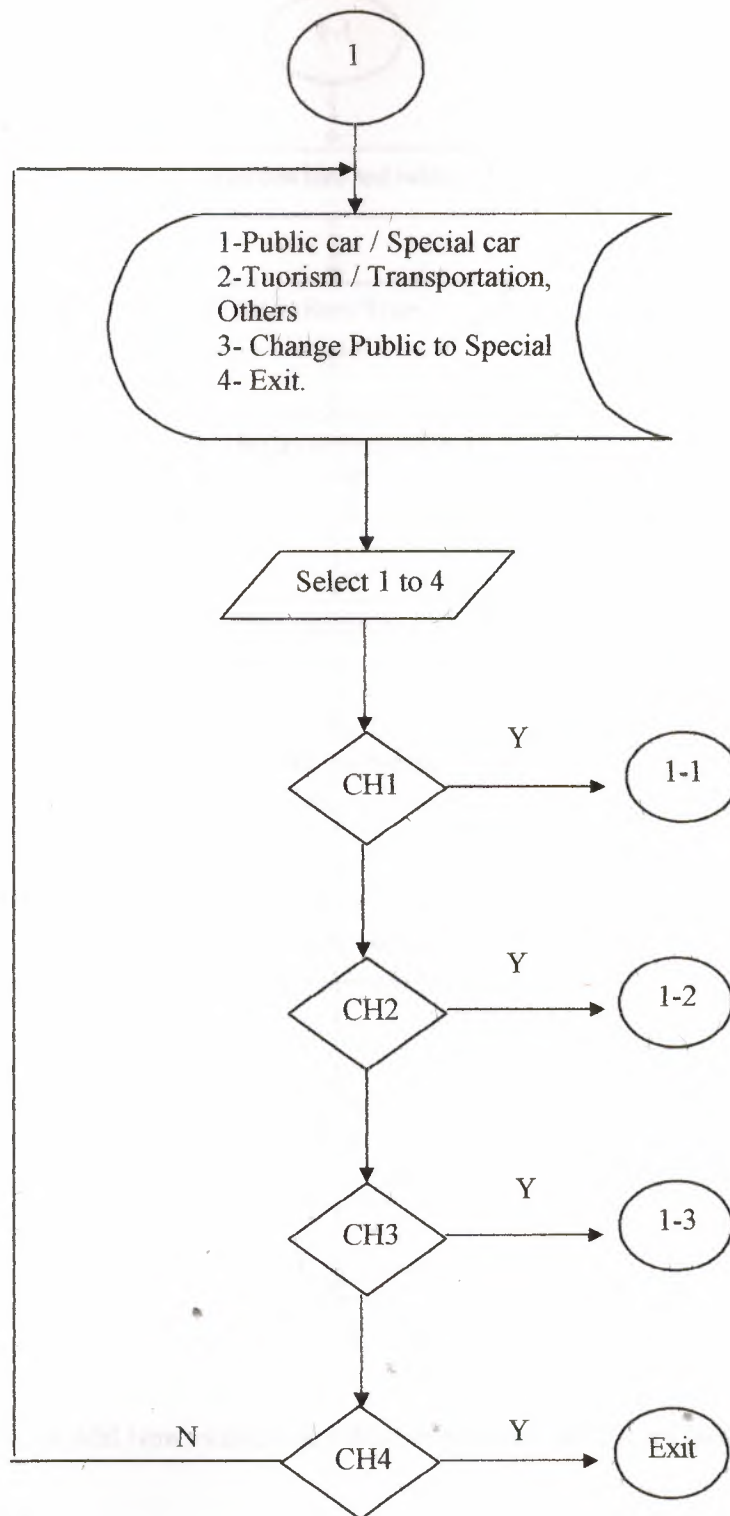


Figure 2.3 Enter Menus Flow Chart



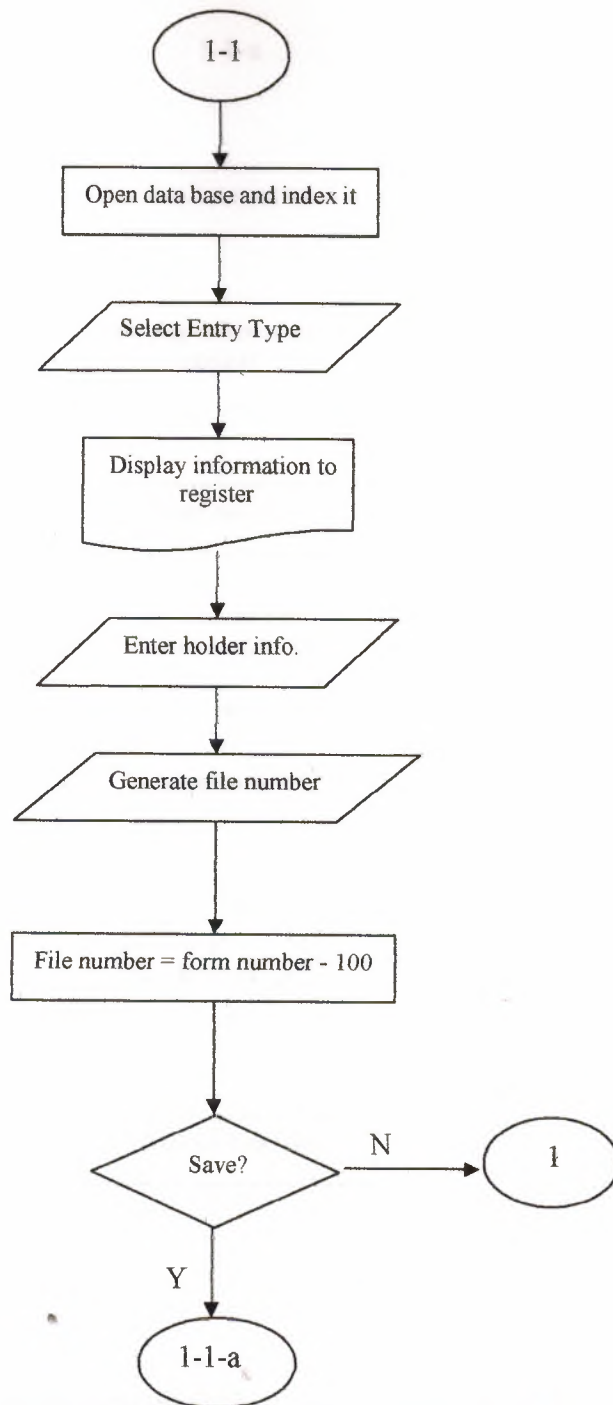


Figure 2.4 Add New Holders in Public or Special Cars Flow Chart

- Create file number.
- Terminator 1 refers to main menu flow chart.
- Terminator 1-1-a refer to continue operation.

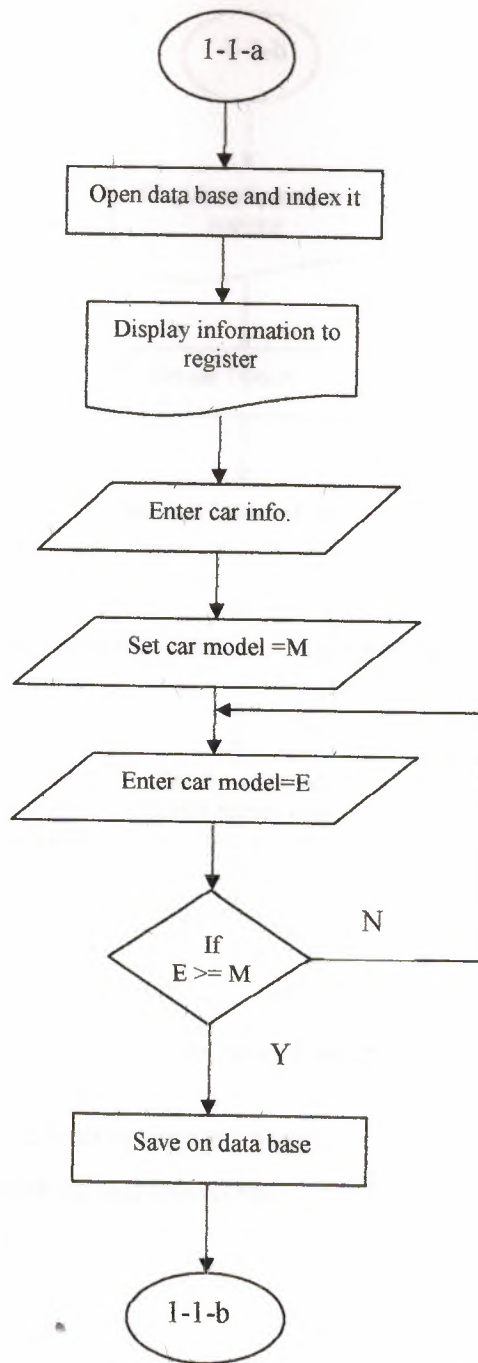


Figure 2.5 Add new car information Public or Special Cars Flow Chart

- Comparing car model operation
- Save information in database
- Terminator 1-1-b refer to continue operation

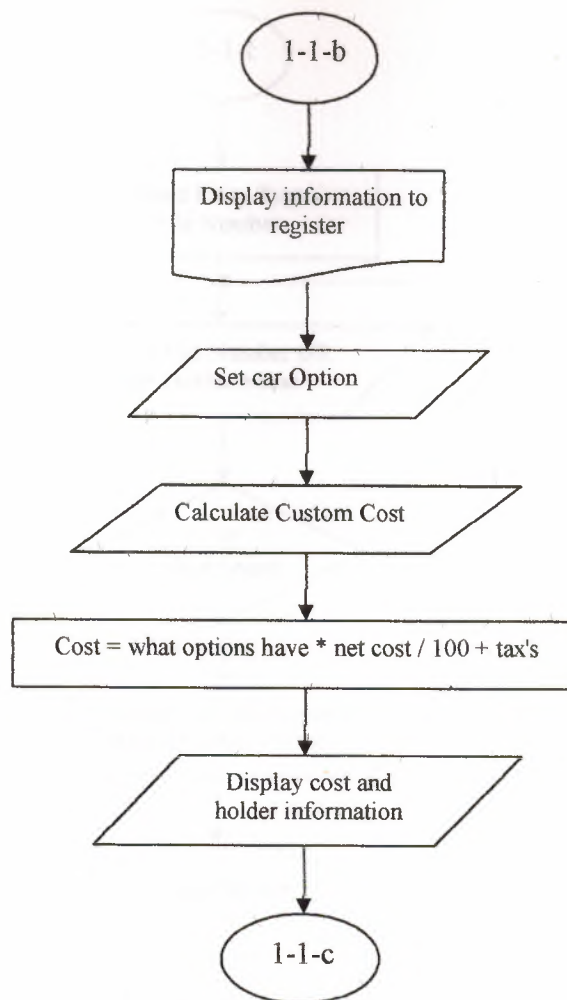


Figure 2.6 Calculate Car Options Public or Special Cars Flow Chart

- Calculate car option refer the what options car have
- Print report for holder information and total costs
- Terminator 1-1-c continue operations



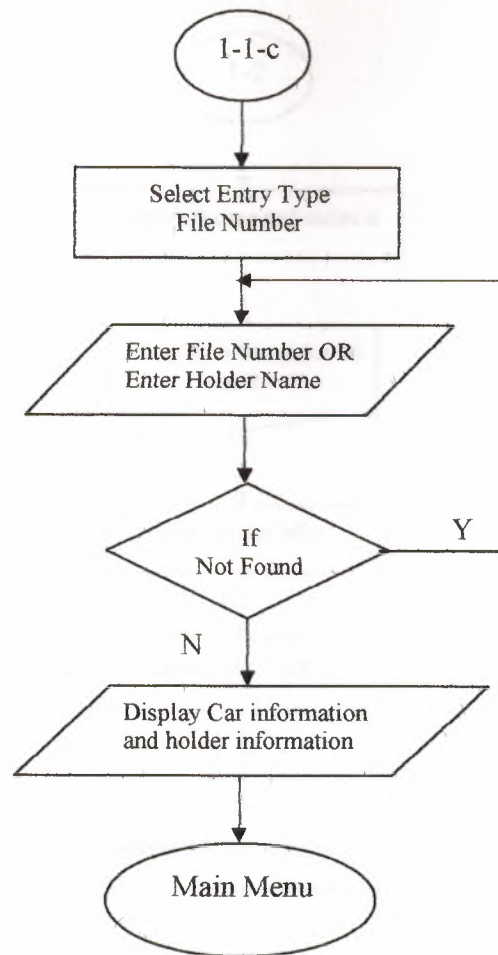


Figure 2.7 Report for Public and Special Cars Information Flow Chart

- Select entry type.
- Search by file number or holder name.
- If not found either file number or holder name show not found message.
- Print information's report.

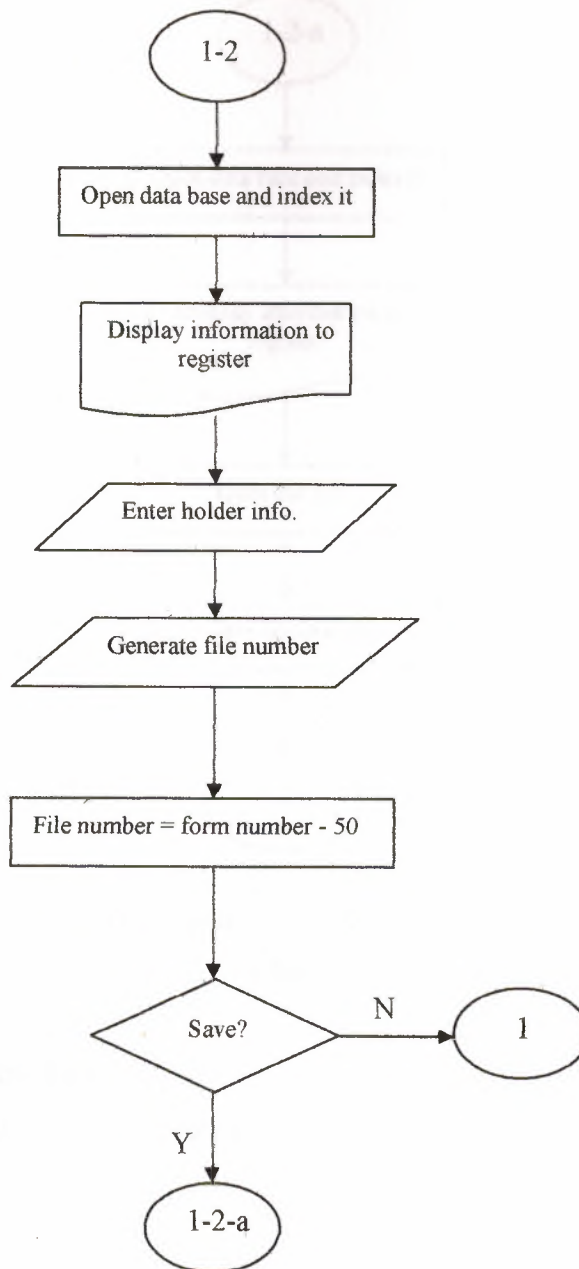


Figure 2.8 Add New Holders in Tourism and Transportation and Others Cars Flow Chart

- Create file number
- Terminator 1 refer to main menu flow chart
- Terminator 1-2-a refer to continue operation

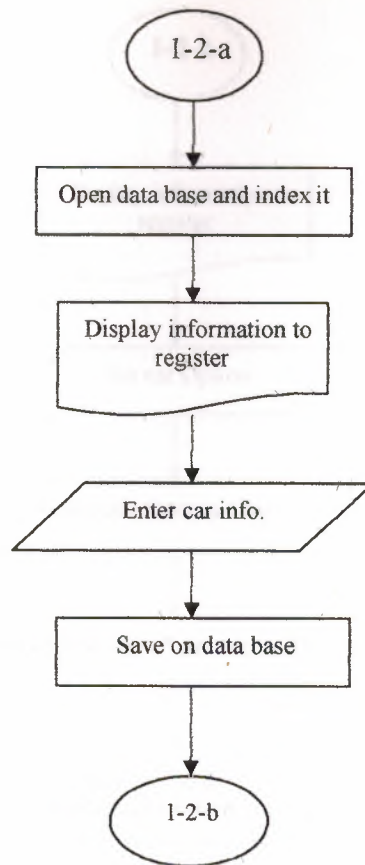


Figure 2.9 Add New Car Information Tourism and Transportation and Others Cars Flow Chart

-Save information in database

-Terminator 1-2-b refer to continue operation



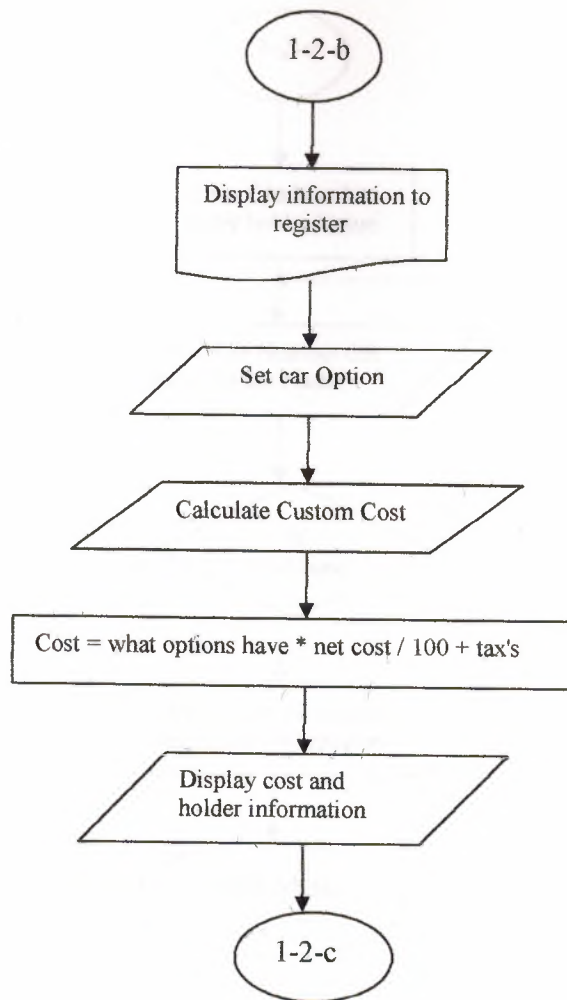


Figure 2.10 Calculate Car Options Tourism and Transportation and Others Cars Flow Chart

- Calculate car option refer the what options car have
- Print report for holder information and total costs
- Terminator 1-2-c continue operations

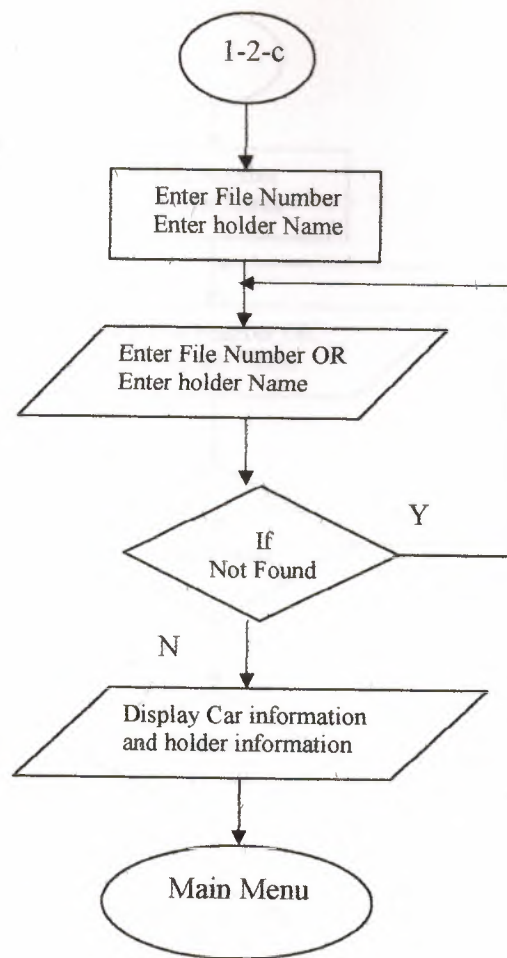


Figure 2.11 Report for Tourism and Transportation and Others Cars Information Flow Chart

- Search by file number OR holder name.
- If not found either file number or holder name show not found message.
- Print information's report.

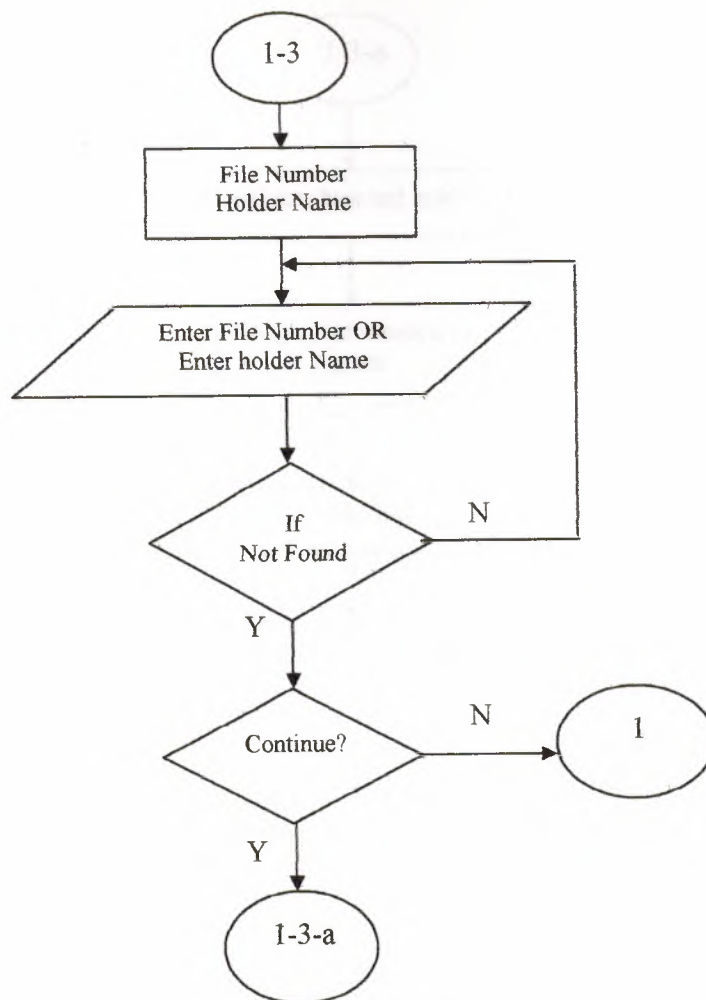


Figure 2.12 Change Entries from Special to Public Flow Chart

- Search by the file number OR holder name.
- If searching not found show message not found
- Terminator 1 refers to main menu.
- Terminator 1-3-a continue operation.



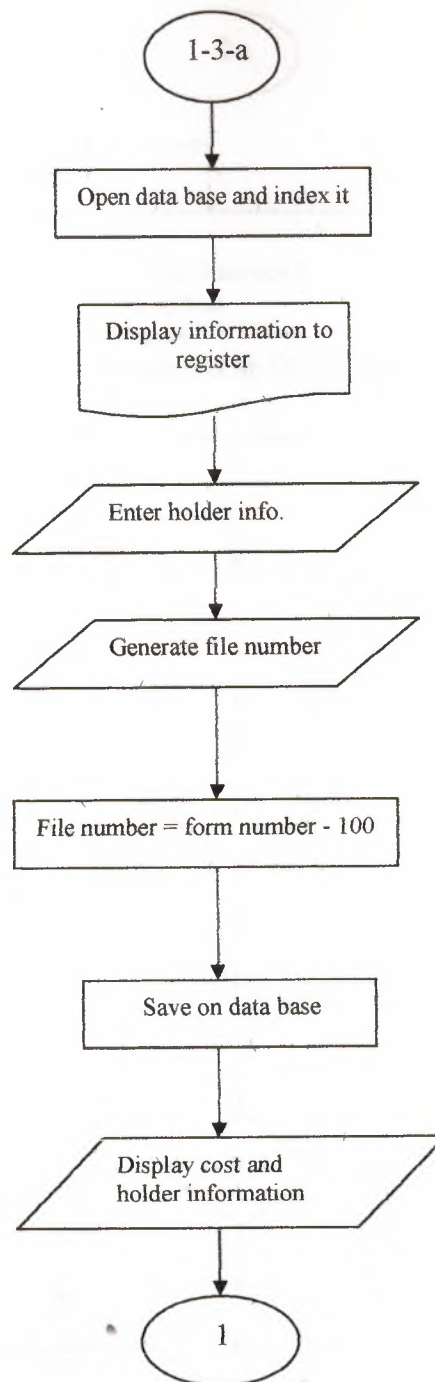


Figure 2.13 Taxes to Change Entry from Special to Public Flow Chart

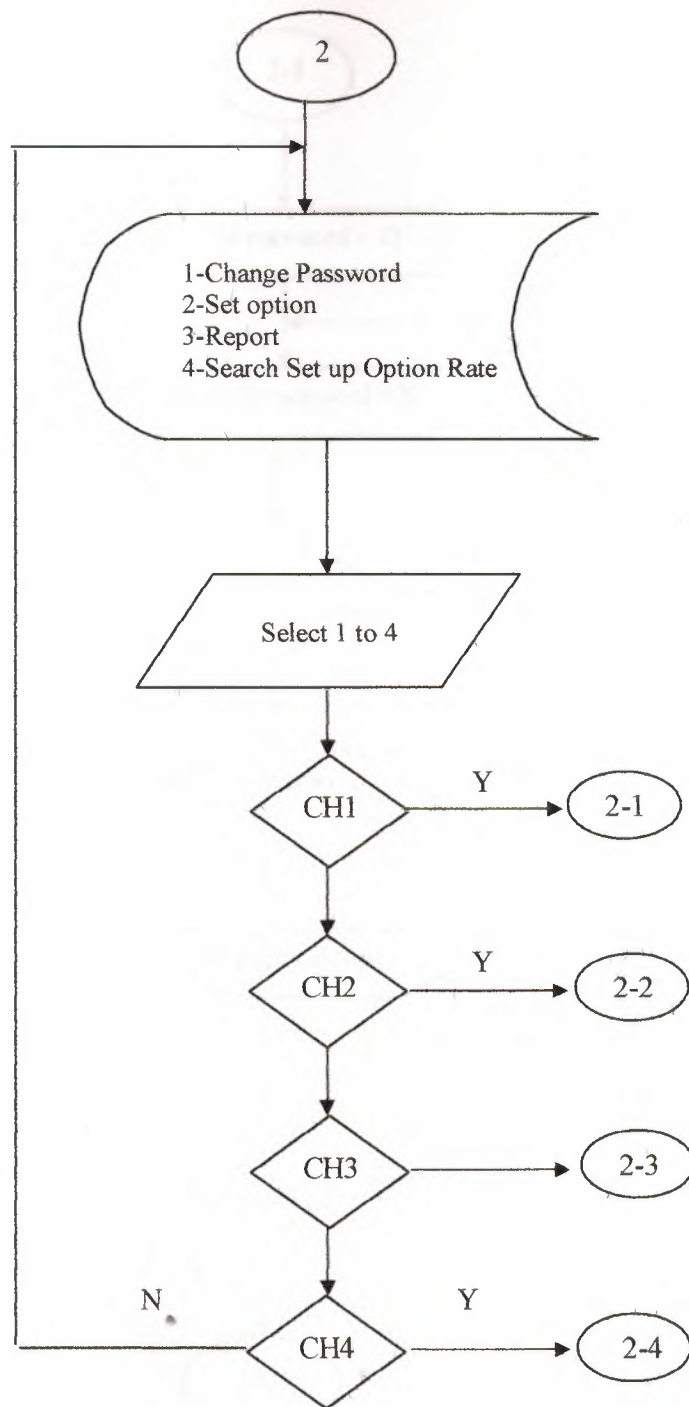


Figure 2.14 Option Menu Flow Chart

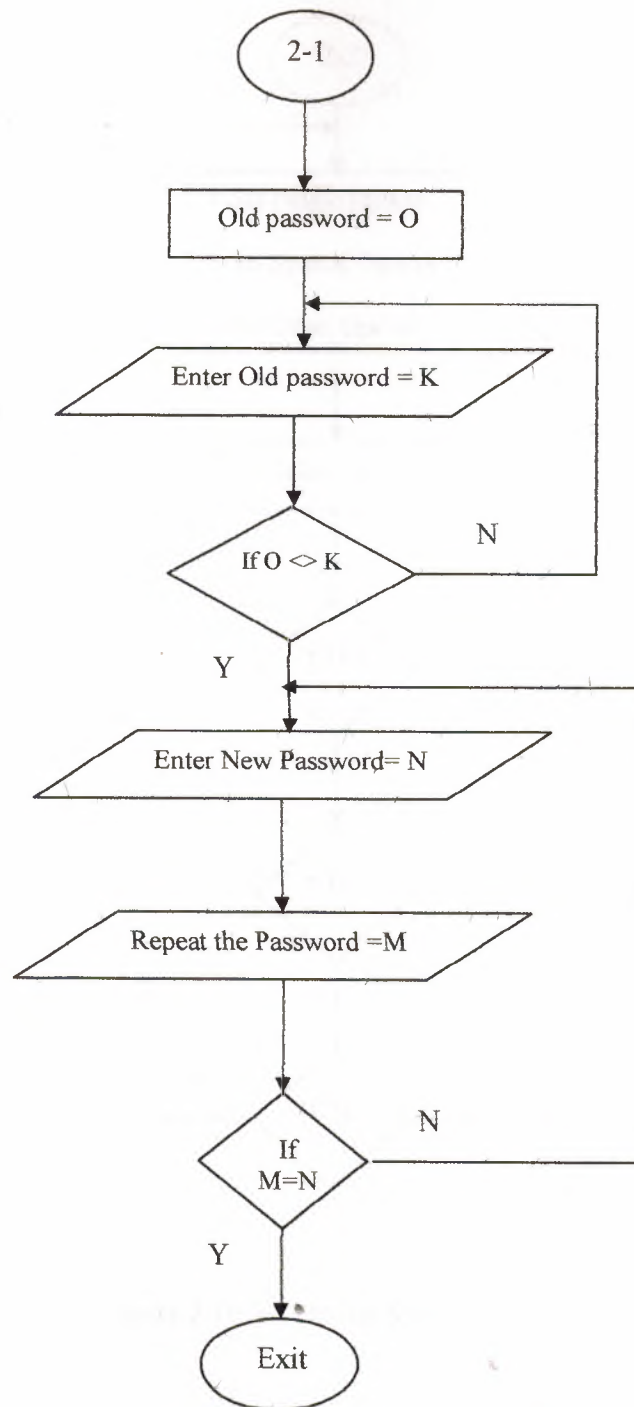


Figure 2.15 Change Password Flow Chart

-Show message after operations wrong filling



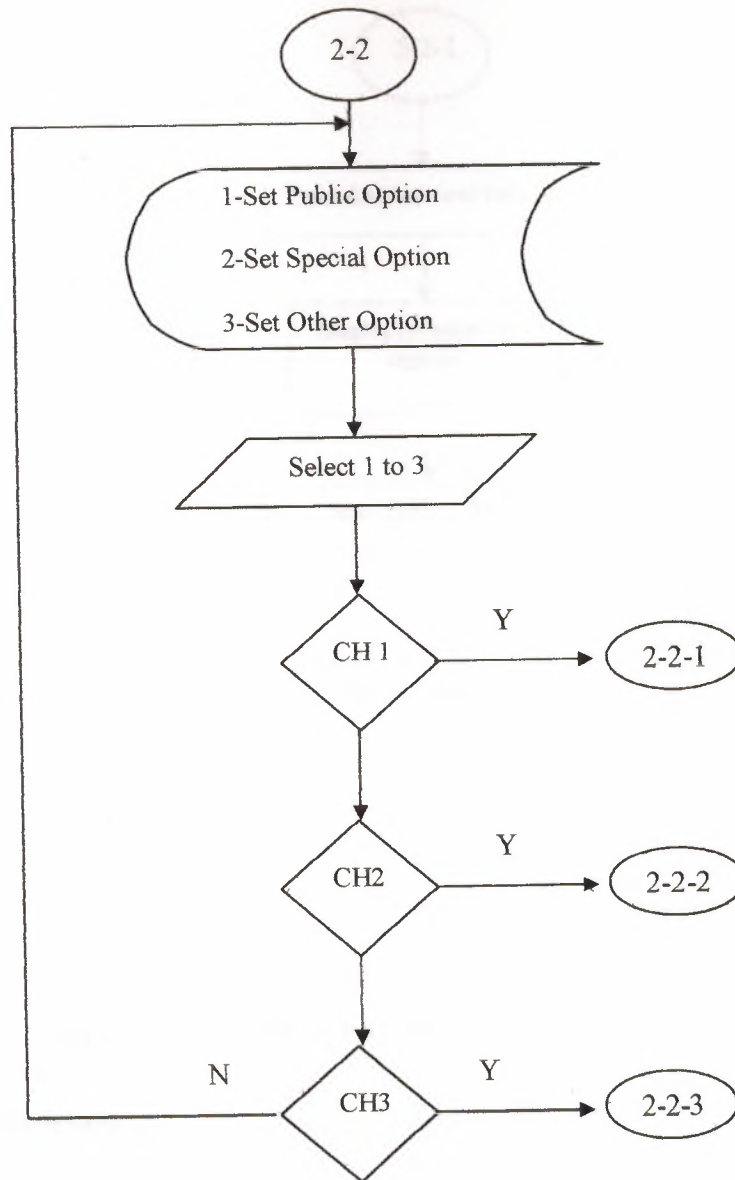


Figure 2.16 Set Option Menu Flow Chart

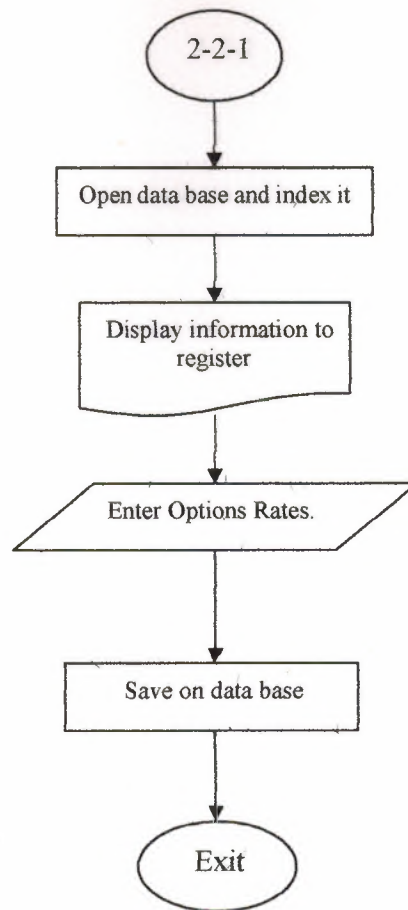


Figure 2.17 Set Options for Public Car Flow Chart

- Filling the blank rates
- Saving the rates

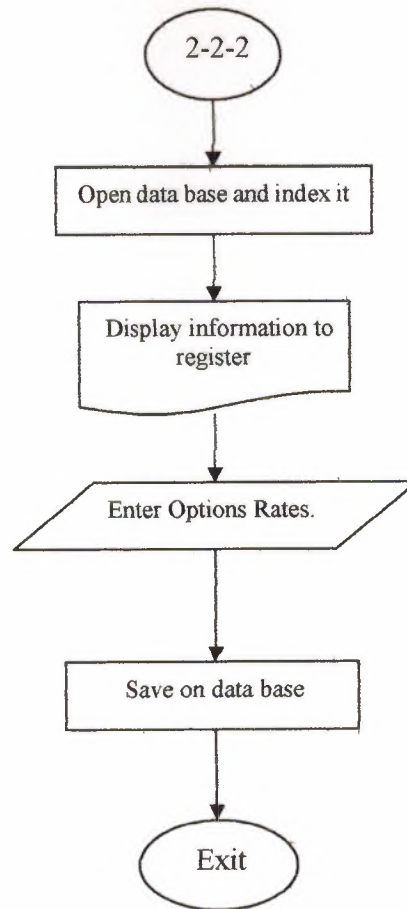


Figure 2.18 Set Options for Special Car Flow Chart

- Filling the blank rates
- Saving the rates



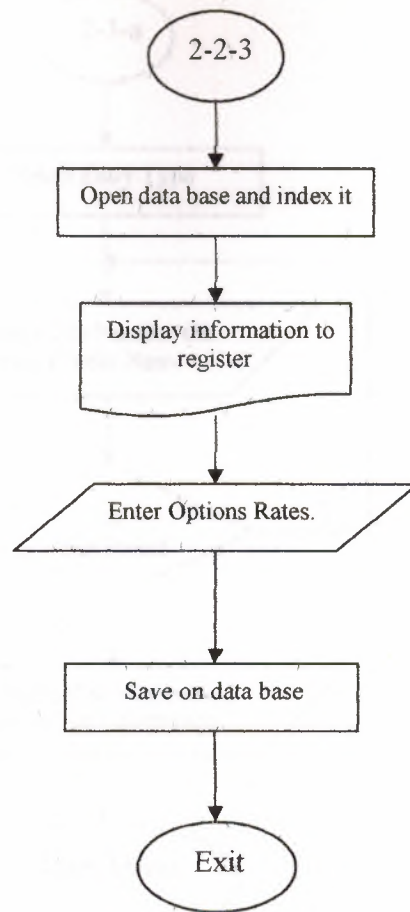


Figure 2.19 Set Options for Tourism, Transportation and Others Cars Flow Chart

- Filling the blank rates
- Saving the rates

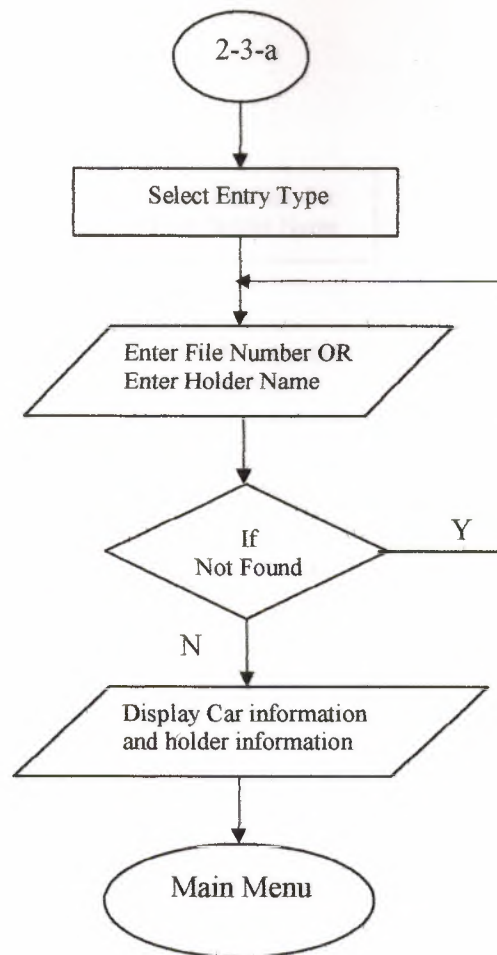


Figure 2.20 Report for Public and Special Cars information Flow Chart

- Select entry type.
- Search by file number OR holder name.
- If not found either file number or holder name show not found message.
- Print information's report.

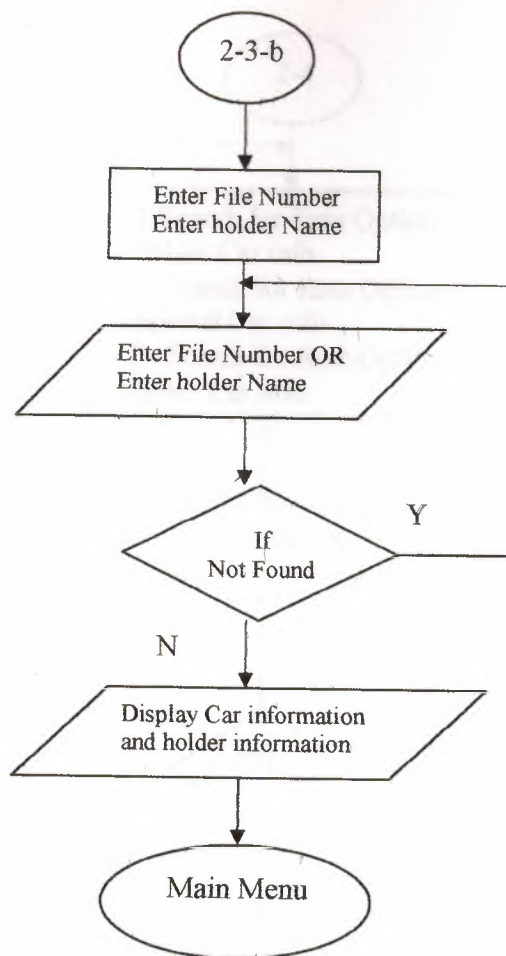


Figure 2.21 Report for Tourism, Transportation and Others Cars Information Flow Chart

- Search by file number OR holder name.
- If not found either file number or holder name show not found message.
- Print information's report.



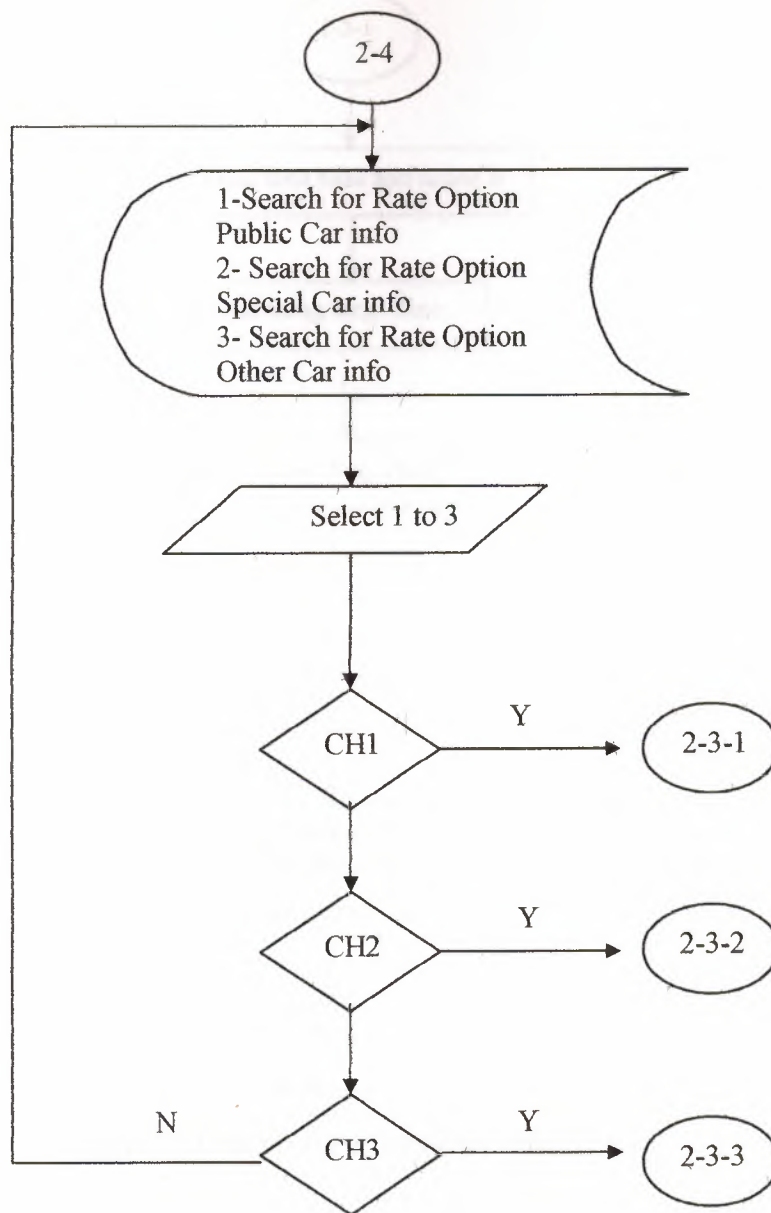


Figure 2.22 Search Options Rate Menu Flow Chart

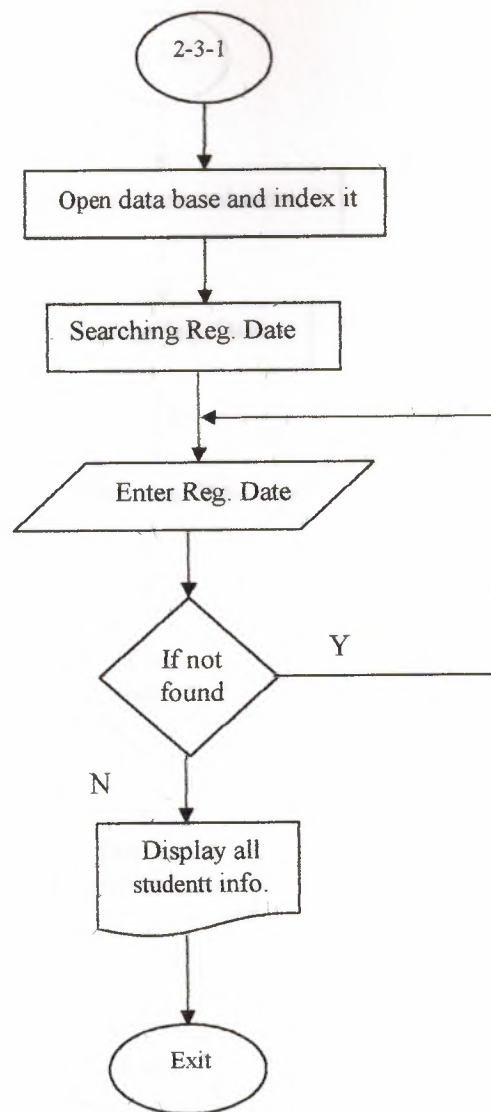


Figure 2.23 Search Rate Option for Public Cars

- Searching by registration date.
- Show wrong message if not found date

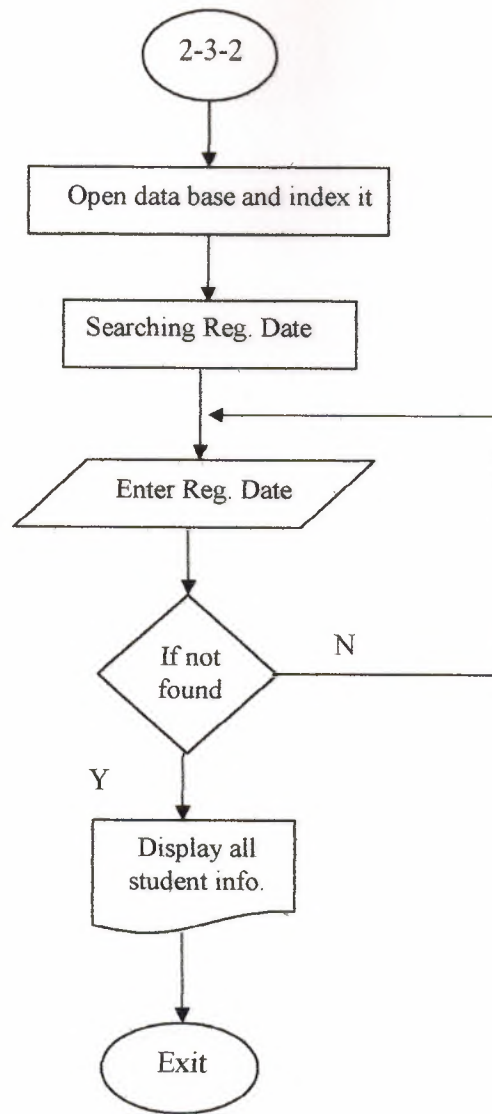


Figure 2.24 Search Rate Option for Special Cars.

- Searching by registration date.
- Show wrong message if not found date

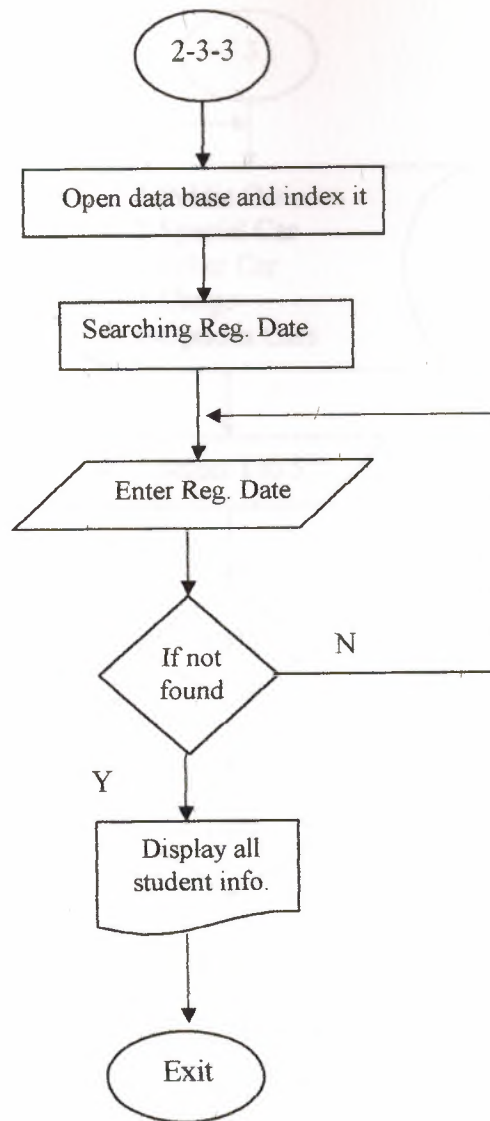


Figure 2.25 Search Rate Option for Other Cars.

- Searching by registration date.
- Show wrong message if not found date



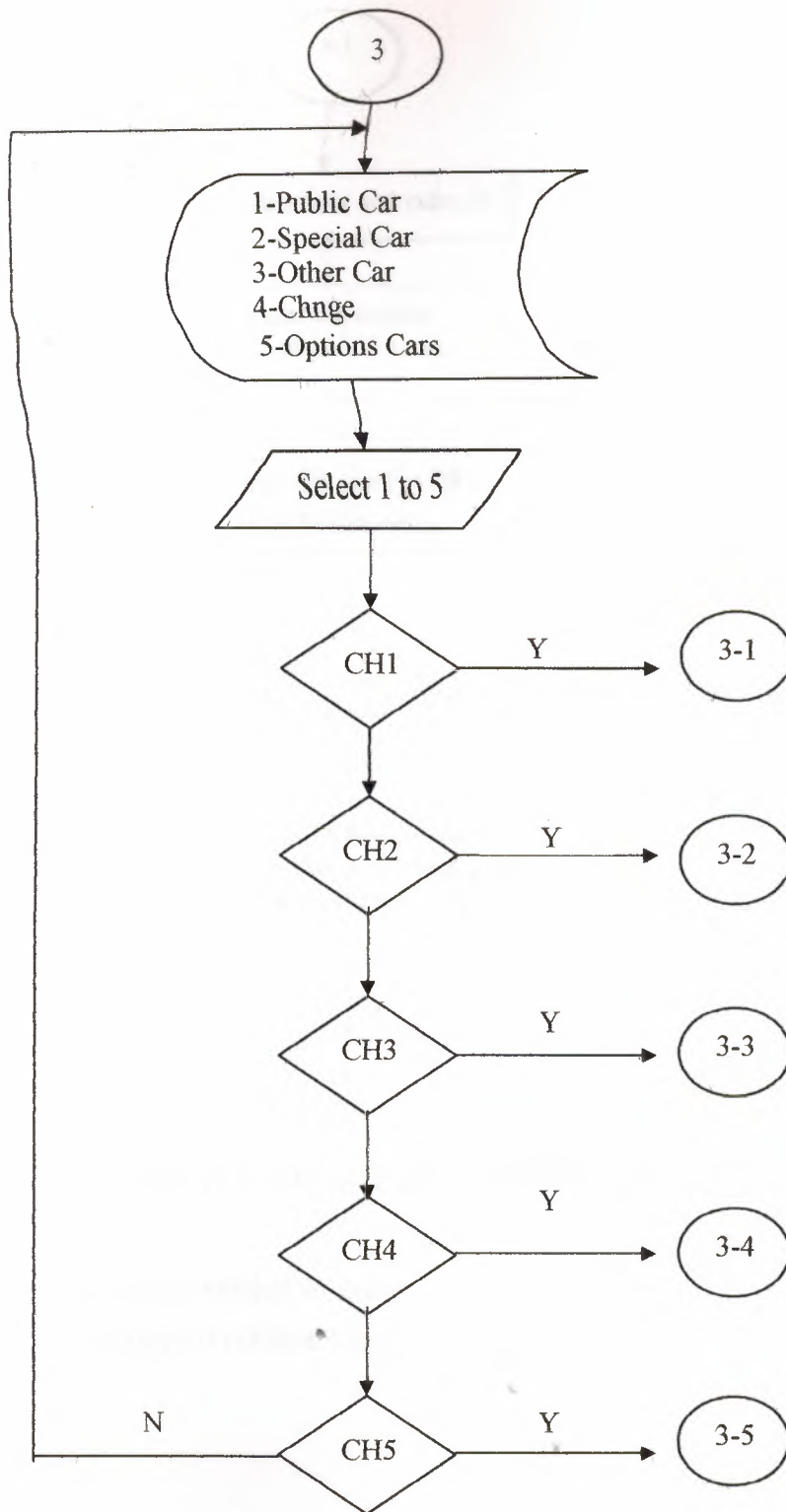


Figure 2.26 Report Main Menu

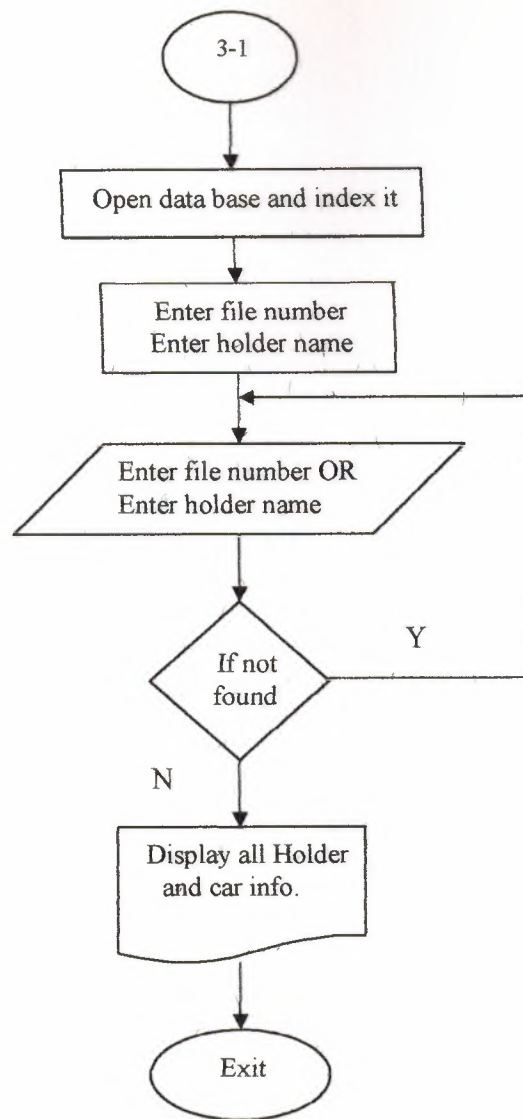


Figure 2.27 Display all Public Cars with Holders

- Search by file number OR holder name
- Show wrong message if not found date

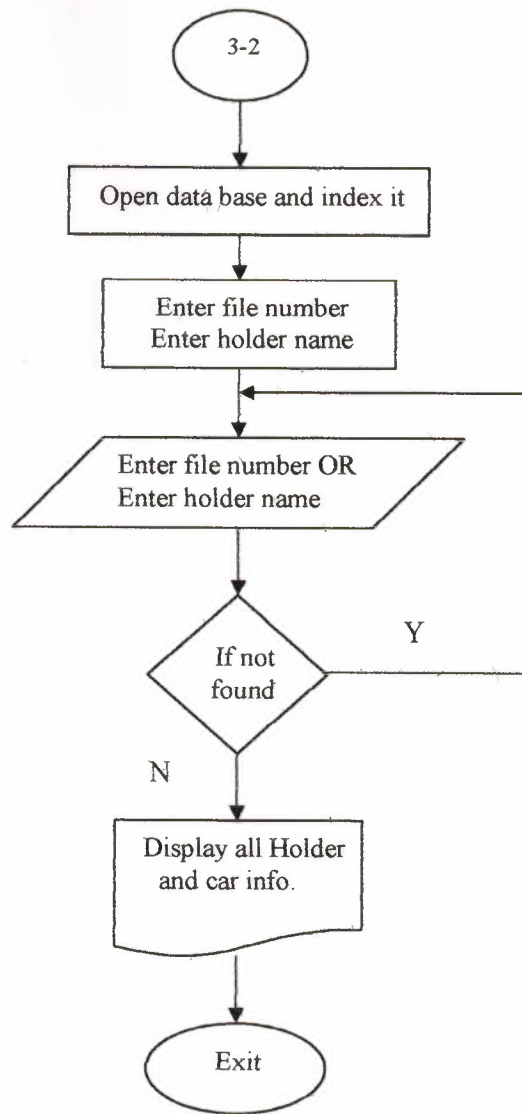


Figure 2.28 Display all Special Cars with Holders

Search by file number OR holder name

Show wrong message if not found date

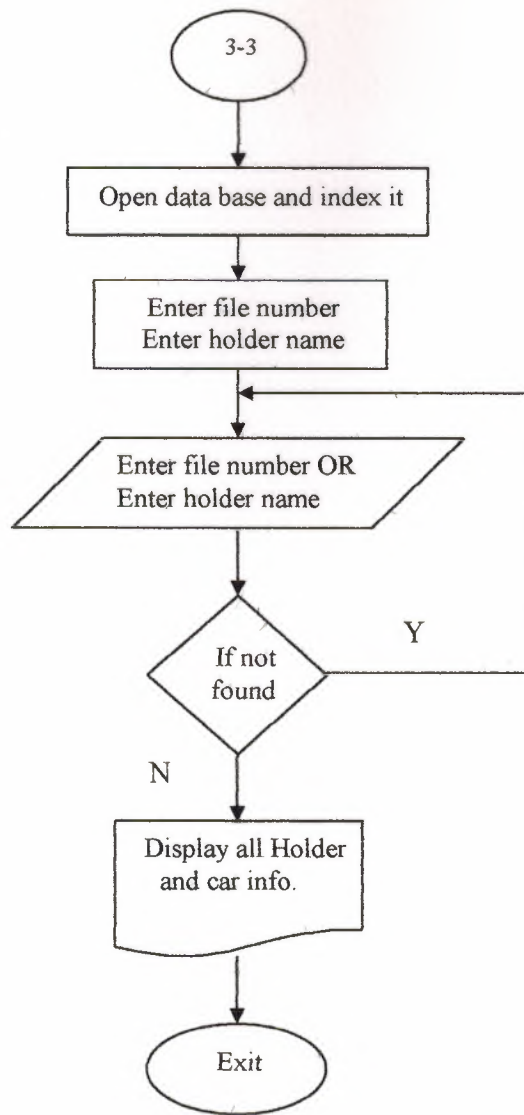


Figure 2.29 Display all Other Cars with Holders

- Search by file number OR holder name
- Show wrong message if not found date



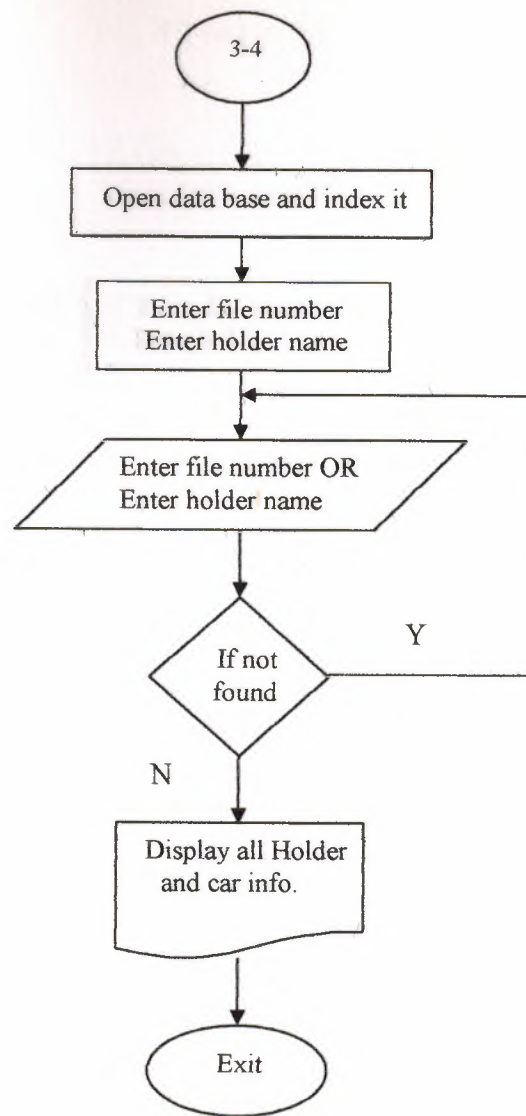


Figure 2.30 Display all was Changed Cars with Holders

- Search by file number OR holder name
- Show wrong message if not found date

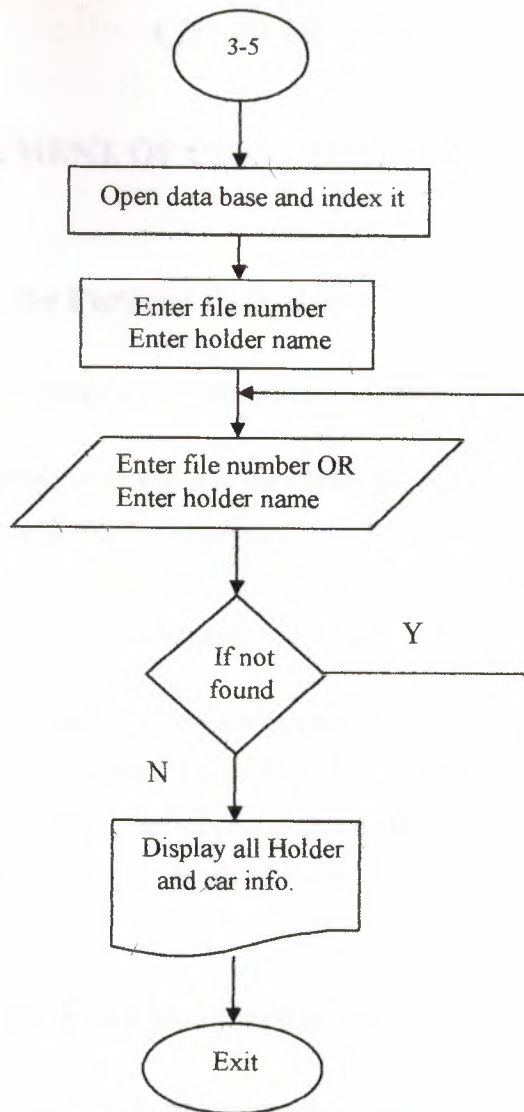


Figure 2.31 Display all Public, Special and Other Cars with Holders

- Search by file number OR holder name
- Show wrong message if not found date

## **CHAPTER THREE**

### **DEVELOPMENT OF CUSTOM INFORMATION SYSTEM**

#### **3.1 Determine the Purpose Database**

The first step in designing a database is to determine its purpose and how it's to be used:

- Talk to people who will use the database. Brainstorm about the questions you and they would like the database to answer.
- Sketch out the reports you'd like the database to produce.
- Gather the forms you currently use to record your data.

As you determine the purpose of your database, a list of information you want from the database will begin to emerge. From that, you can determine what facts you need to store in the database and what subject each fact belongs to. These facts correspond to the fields (columns) in your database, and the subjects that those facts belong to correspond to the tables.

#### **3.2 Determine the Field You Need in the Database**

Each field is a fact about a particular subject. For example, you might need to store the following facts about your customers: company name, address, city, state, and phone number. You need to create a separate field for each of these facts. When determining which fields you need, keep these design principles in mind:

- Include all of the information you will need.
- Store information in the smallest logical parts. For example, employee names are often split into two fields, FirstName and LastName, so that it's easy to sort data by LastName.
- Don't create fields for data that consists of lists of multiple items. For example, in a Suppliers table, if you create a Products field that contains a comma-separated list of each product you receive from the supplier, it will be more difficult to find only the suppliers that provide a particular product.



- Don't include derived or calculated data (data that is the result of an expression (expression: Any combination of mathematical or logical operators, constants, functions, and names of fields, controls, and properties that evaluates to a single value. Expressions can perform calculations, manipulate characters, or test data.)). For example, if you have a UnitPrice field and a Quantity field, don't create an additional field that multiplies the values in these two fields.
- Don't create fields that are similar to each other. For example, in a Suppliers table, if you create the fields Product1, Product2, and Product3, it will be more difficult to find all suppliers who provide a particular product. Also, you will have to change the design of your database if a supplier provides more than three products. You need only one field for products if you put that field in the Products table instead of in the Suppliers table.

### **3.3 Determine the Tables You Need in the Database**

Each table should contain information about one subject. Your list of fields will provide clues to the tables you need. For example, if you have a HireDate field, its subject is an employee, so it belongs in the Employees table. You might have a table for Customers, a table for Products, and a table for Orders.

### **3.4 Determine Which Table each Field Belongs to**

When you decide which table each field belongs to, keep these design principles in mind:

- Add the field to only one table.
- Don't add the field to a table if it will result in the same information appearing in multiple records in that table. If you determine that a field in a table will contain a lot of duplicate information, that field is probably in the wrong table. For example, if you put the field containing the address of a customer in the Orders table, that information will probably be repeated in more than one record, because the customer will probably place more than one order. However, if you put the address field in the Customers table, it will appear only once. In this respect, a table in differs from a table in a flat file database such as a spreadsheet. When each piece of information is stored only once, you update it



in one place. This is more efficient, and it also eliminates the possibility of duplicate entries that contain different information.

### **3.5 Identify the Field or Fields with Unique Value in Each Record**

In order for Microsoft Access to connect information stored in separate tables for example, to connect a customer with all the customer's orders each table in your database must include a field or set of fields that uniquely identifies each individual record in the table. Such a field or set of fields is called a primary key (primary key: One or more fields (columns) whose value or values uniquely identify each record in a table. A primary key cannot allow Null values and must always have a unique index. A primary key is used to relate a table to foreign keys in other tables.).

### **3.6 Determine the Relationships Between Tables**

Now that you've divided your information into tables and identified primary key (primary key: One or more fields (columns) whose value or values uniquely identify each record in a table. A primary key cannot allow Null values and must always have a unique index. A primary key is used to relate a table to foreign keys in other tables.) Fields, you need a way to tell Microsoft Access how to bring related information back together again in meaningful ways. To do this, you define relationships (relationship: An association established between common fields (columns) in two tables. A relationship can be one-to-one, one-to-many, or many-to-many.) Between tables. You may find it useful to view the relationships in an existing well-designed database such as the North wind sample database.

### **3.7 Refining Design**

After you have designed the tables, fields, and relationships (relationship: An association established between common fields (columns) in two tables. A relationship can be one-to-one, one-to-many, or many-to-many.) You need, it's time to study the design and detect any flaws that might remain. It is easier to change your database design now than it will be after you have filled the tables with data. When you are satisfied that the table structures meet the design principles described here, then it's time

to go ahead and add all your existing data to the tables. You can then create other database objects

### 3.8 Designing Project Database Structure

Good database design ensures that your database is easy to maintain. You store data in tables and each table contains data about only one subject, such as customers. Therefore, you update a particular piece of data, such as an address, in just one place and that change automatically appears throughout the database.

A well-designed database usually contains different types of queries that show the information you need. A query might show a subset of data, such as all customers in London, or combinations of data from different tables, such as order information combined with customer information. The results you want from your database the forms and data access pages (data access page: A Web page, published from Access that has a connection to a database. In a data access page, you can view, add to, edit, and manipulate the data stored in the database. A page can also include data from other sources, such as Microsoft Excel. And Access ) You want to use, and the reports you want to print don't necessarily provide clues about how you should structure the tables in your database, because you often base forms, reports, and data access pages on queries instead of tables.

Field Name	Data Type	Field Size	Description
Audio_system	Alpha	2	Audio system
Elec_mirror	Alpha	2	Electrical Mirror
Air_condition	Alpha	2	
Elec_windows	Alpha	2	Electrical Windows
Center_lock	Alpha	2	Center Lock
Auto_gear	Alpha	2	Automatic Gear
Alert_tone	Alpha	2	Alert Tone
Power_streeing	Alpha	2	Power Steering
Injection	Alpha	2	
Air_bag	Alpha	2	Air Bag
Roof	Alpha	2	Roof has Slash
Open_roof	Alpha	2	With out Roof

Table 3.1 Option for All Cars




Field Name	Data Type	Field Size	Description
File_number	Number		File for car (*)
Name	Alpha	20	Holder Name
Suranme	Alpha	15	Holder Surname
ID_No	Number		ID Holder Number
Address	Alpha	20	Address of Holder
DOB	Date		Date of Birth
Nationality	Alpha	10	Nationality of Holder
Tele	Number		Holder Available Telephone
Orig	Alpha	10	Original of Car
Form_No	Number		Car Form Number
Entry_date	Date		Entry Car Date
Comp_Name	Alpha	20	Company whose Bring The Car
Resid_no	Number		Residence Number for Foreign Holders
Issue	Date		Issue Date for Residence
Expiry	Date		Expiry Date for Residence

Table 3.2 Holder Information

Field Name	Data Type	Field Size	Description
Audio_system	Number		Audio system
Elec_mirror	Number		Electrical Mirror
Air_condition	Number		Air Condition
Elec_windows	Number		Electrical Windows
Center_lock	Number		Center Lock
Auto_gear	Number		Automatic Gear
Alert_tone	Number		Alert Tone
Power_streeing	Number		Power Steering
Injection	Number		
Air_bag	Number		Air Bag
Roof	Number		has Slash
Open_roof	Number		With out Roof
Emp_name/number	Alpha	20	Employee Name and Number
Date	Date		Date of Set option Rate
Order_No	Number		Set Order Option Rate

Table 3.3 Rate Option for All Cars



Field Name	Data Type	Field Size	Description
File_number	Number		File for car (*)
Kind	Alpha	10	Kind of Car
Type	Alpha	10	Type of Car
Car_Model	Number		Model of Car
Type_of_foul	Alpha	10	Car Foul (Benzene, Solar)
Date	Date		Registration Date
Nationality	Alpha	10	Nationality of Holder
Persons	Number		How Many Persons?
Car_weight	Number		Weight of Car
Remark	Alpha	20	Any Remark Available
Car_color	Alpha	10	Car Color
Loading	Number		Loading (kg)
Auditor_Name	Alpha	20	Auditor Name
Auditor_number	Number		Auditor Number
Motor_number	Number		Motor Car Number
Body_number	Number		Body Number for Car
Doors_number	Number		How Many Door in the car?

Table 3.4 Car Information

### 3.9 Define Relationships Between Tables

When we create a relationship, the related fields don't have the same names. However, related fields must have the same data type unless the primary key field is an AutoNumber field. We can match an AutoNumber field with a number field only if the fieldsize property of both of the matching fields is the same. For examples, we can match an AutoNumber field and a field number field if the fieldSize property of both fields is Long Integer. Even when both matching fields are Number fields, they must have the same fieldSize property setting. We can see below the relationships between the tables of this project:



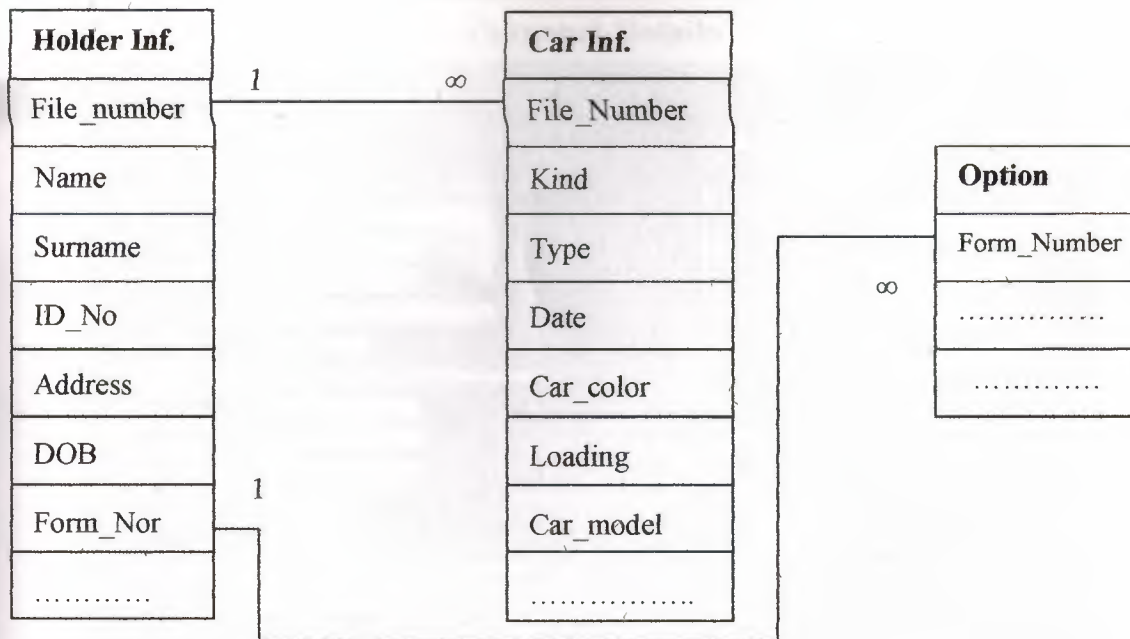


Figure 3.1 Relationships

### 3.10 Layout of the Application:

#### 3.10.1 Main Menu Screen:

It consists of seven Buttons, each button has a specific mission, and these missions will be explaining as follow:



Figure 3.2 Main Menu

- 1- Enter Button: we can use this button to decided entry car like Public, Special and Other entry, and change from special to public and close the program as shown in figure 3.3. at first select entry type toy want, generate button generate file number for the holder but next button go to next screen car details, back button return to main menu, same type for the others cars (tourism, transportation and other).

### Personal Details

Select Cars:

☒ Public Cars ☐ Special Cars

<p>Name: <input type="text"/></p> <p>Surname: <input type="text"/></p> <p>ID Number: <input type="text"/></p> <p>Address: <input type="text"/></p> <p>Date of Birth: <input type="text"/></p> <hr/> <p>Nationality: <input type="text"/></p> <p>Passport No: <input type="text"/></p> <p>Residence No: <input type="text"/></p> <p>Date of Issue: <input type="text"/></p> <p>Date of Expiry: <input type="text"/></p> <hr/> <p>Telephone: <input type="text"/></p>	<p>Original: <input type="text"/></p> <p>Form Number: <input type="text"/></p> <p>Entry Date: <input type="text"/></p> <p>Company Name: <input type="text"/></p> <hr/> <p>File Number: <input type="text"/> <input type="button" value="Generate"/></p>
---	---

Figure 3.3 Holder Information Screen

NOTE: if the holder was foreign the entry date of issue and date of expiry data for passport

- 2- Option Button: this button contain change password figure 3.4, option and setup option for the cars (Options Rate), Report for the holder and car details, search option rate for cars as shown in figure 3.5

### Change Password

Enter Old Password:

---

Enter New Password:

Confirm:

Figure 3.4 Change Password



set option private cars

Automatic Gear ☐ Power Steering ☐

Center Lock ☐ Injection ☐

Electrical Windows ☐ Air Bag ☐

Air Condition ☐ Roof ☐

Electrical Mirror ☐ Opening Roof ☐

Alert Tone ☐ Audio System ☐

Set Model: 1995

Employee Name: \_\_\_\_\_ Employee Number: \_\_\_\_\_

Date (d/m/y): \_\_\_\_/\_\_\_\_/\_\_\_\_ Order Number: \_\_\_\_\_

Figure 3.5 Set Rate Options and Cars Model

In this option button had another feature for the user to easy to used search rate of the option was in that period time this option show of the user rate of the option after time how much it was, shown in figure 3.6

Form25

Search Delete First Last

**Public Cars**

Order number	Emp. name	Emp. number	Auto gear	Center lock	Elec. windows	Air condition	Elec. mirror
512	AHMAD DEEB	10	13	15	15	15	15
512	AHMAD DEEB	10	13	15	15	15	15
512	AHMAD DEEB	10	13	15	15	15	15

Next -> <- Back

Search Delete First Last

**Special cars**

Order number	Emp. name	Emp. number	Auto gear	Center lock	Elec. windows	Air condition	Elec. mirror
512	AHMAD DEEB	10	10	10	10	20	20
512	AHMAD DEEB	10	10	10	10	20	20
512	AHMAD DEEB	10	10	10	10	20	20

Next -> <- Back

Search Delete First Last

**Tou, Tran, Other**

Order number	Emp. name	Emp. number	Auto gear	Center lock	Elec. windows	Air condition	Elec. mirror
512	AHMAD DEEB	10	10	10	10	10	10
512	AHMAD DEEB	10	10	10	10	10	10
512	AHMAD DEEB	10	10	10	10	10	10

Next -> <- Back

OK

Figure 3.6 Rates of Options



- 3- Report Button: the usage of this button is to find a certain data that we entered for the holder information and the car that same file number for each entry car shown below in figure 3.7, and show every car what have options in the same form number shown in figure 3.8 in each entry type.

**Public Cars**

File Number:  Name:

Search Search First Next < Back Last

File number	Name	Surname	Id no	Address	DOB	Nationality	Tel
90	ali issied	almassri	0	ortekoy	11/11/80	lobanonian	236
99	halif ahmet	orham	256398	gonyli	11/11/75	cyp	
100	fayez nemar	kasem	215487	lefkosa	22/02/80	cyp	24
114	orhan ahmet	daimanci	562578	ortekoy	03/03/75	cyp	52
116	mohmad ali	ganinam	125465	lefkosa	30/01/80	cyp	25
117	zeya solem	ahmet	0	ortekoy	20/04/80	jordanian	
120	khaled waleed	e'amar	236594	lefkosa	11/11/82	cyp	26
122	omar ahmet	ozgur	985236	lefka	25/05/70	cyp	18

File number	Kind	Type	Car model	Type of fuel	Date	Persons	Car weight	Remark
90	bmw	320	2000 benzene	09/08/01	5	1200		

Ok

Figure 3.7 Show all Holders was Registered with Car Details for each Entry

**Option in Public Car**

Search First Next Back Last

File number	Audio system	Elec mirror	Air condition	Elec windows	Center lock	Auto gear	Alert tone	Power steering	Injection	Air bag
295	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
11111	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

**Option in Special Car**

Search First Next Back Last

File number	Audio system	Elec mirror	Air condition	Elec windows	Center lock	Auto gear	Alert tone	Power steering	Injection	Air bag
755	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
1001	N	N	N	N	N	N	Y	Y	Y	Y

**Option in Rest Cars**

Search First Next Back Last

File number	Audio system	Elec mirror	Air condition	Elec windows	Center lock	Auto gear	Alert tone	Power steering	Injection	Air bag
258	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
11111	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Ok

Figure 3.8 Show All Cars Options in All Entries



### 3.10.2 Cars Information Screen

In this screen the user must fill the data blank, and should be care for the model car must greater or equal was setting in figure 3.5, if less than that the program show message "Invalid Car Model" the figure 3.9 shown bellow refer to car information. in the type blank data type for car 'Number of the Kind' and in the remark filled type of car.

Car Details	
Name of Holder	ahmad deeb
Surname	khaled
ID Number	992065
File Number	1654
Date	___/___/___
Kind	[dropdown]
Type	[text]
Car Model	[text]
Motor Number	[text]
Doors Numbers	[text]
Type of Fuel	[dropdown]
Body Number	[text]
# Persons	[text]
Car Weight	[text]
Remark	[text]
Motor Capacity	[text]
Car Color	[text]
Loading	[text]
Auditor Name	[text]
Number	[text]
[Next >>]	

Figure 3.9 Car Information

NOTE: holder name, surname, ID number are taken automatically from holder information screen in figure 3.3, this feature also available in each car information entries

### 3.10.3 Order Payments and Options Screen:

This screen allow us to enter costs such as: government tax's and other tax and the user must fill the net price for the car (Estimating Cost) for calculate the Gross Price figure 3.10. With setting car options in figure 3.11 after that getting report for holder information with total cost. Using print report button.



NOTE: holder name, surname, ID number and file number are taken automatically from holder information

Payment Order  
Options

**Taxes**

Holder Name: ahmad deeb  
Surname: khaled  
ID Number: 214563  
File Number: 82036

Open File:   
Roads:   
Services:   
Others:   
Government:

Net Price:   
Gross Price:

**CUSTOM PRICE**  
Set Option:

Bind Form:   
Calculate:   
Next >>:

Figure 3.10 Calculate Custom Cost

In the top show red label that is mean type of entry, and custom price not allowed in special entry car that is mean not include in gross calculation price.

**Options for Cars**

☐ Automatic Gear  
☐ Center Lock  
☒ Electrical Windows  
☐ Air Condition  
☒ Electrical Mirror  
☒ Audio System  
☒ Alert Tone

☒ Power Steering  
☒ Inejection  
☐ Air Bag  
☒ Set Roof

Figure 3.11 Options

This screen attached to database each one was checked input in database "Y", else "N" as shown in figure 3.8 dependent to form number.

NOTE: After finish filling the operation for each different entry the user should have print a report for some information that required to traffic department, in the end of report appear type of entry and date year, month and day are separated by '>>' and '<<'.  
<img alt="Screenshot of a report showing entry type and date." data-bbox="130 130 550 170"/>

### 3.11 Buttons Function

There are many buttons are using in the program that refers to specific function. Dependent from each other button. Each button has caption what it job easy to knew function of the button there are some buttons are bellow.

- Delete: delete record when the pointer is stopped in the database
- First: address of pointer on the first record in the database.
- Last: address of pointer on the last record in the database.
- Next: move pointer to the next record.
- Back: move pointer to previous record (in database), show previous careen.
- Search: allocate the pointer on record by enter file number or hold name
- Next: show next screen
- Ok: operation acceptation
- Cancel: ignore operation.
- Exit: exit the program
- Calculate: calculate the gross price.
- Set option: show the option screen
- Generate: Generate file number
- Main menu: show the main menu screen
- Print: print report.
- Print Preview: preview report before print action

If there are any button not on the above list that mean common use by caption.



## CONCLUSION

In this project I learned a lot of thing that in the first time and even though not all of things I wanted to do in this project but this is mainly because of the lack of time and knowledge in programming with Delphi Programming. But we can say the Delphi database support is very extensive and complete. I have very high hopes on expanding the capability of this program in near future and from there I will take off in mastering Delphi to design any project. I will try to take a lot of experience which is very important tool that I will need to take any obstacles being faced in the future.

In the graduation project the description of course, holder's registrations, gross price calculation are given. The structure of holder information system is presented. Its main database modules are developed. The algorithms for car, holder registration, and gross price calculation are presented. The implementation of course, holder registration, gross price calculation problem in Delphi programming language is carried out. Developed program allow automating.



## APPENDIX

In this section available to show some codes for some operations refer to procedures custom carrying out of operation using user effort to appear the correct output, this codes some operation about public and special process as they generating in Delphi compiler accepted.

### 1. Password Screen Code

```
unit com400;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```

```
Dialogs, StdCtrls, ExtCtrls, DB, DBTables, Menus, ActnList;
```

```
type
```

```
TForm1 = class(TForm)
```

```
    Button1: TButton;
```

```
    Edit1: TEdit;
```

```
    Button2: TButton;
```

```
    Label1: TLabel;
```

```
    Panel1: TPanel;
```

```
    Panel2: TPanel;
```

```
    DataSource1: TDataSource;
```

```
    Table1: TTable;
```

```
    Image1: TImage;
```

```
    MainMenu1: TMainMenu;
```

```
    About1: TMenuItem;
```

```
    Auther1: TMenuItem;
```

```
    procedure FormCreate(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);
```

```

    procedure Button1Click(Sender: TObject);

private
    { Private declarations }

public
    { Public declarations }

end;

var
    Form1: TForm1;

implementation

uses Unit2, Unit4, Unit7;

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    edit1.clear ;
    BUTTON1.TabOrder:=1;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    close;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    X,code:integer;

```

```

ch:string;
begin
val(edit1.Text,x, code);
STR (X,CH);
if (CH=table1.FieldValues['pass']) then
begin
form4.show;
form1.Visible:=false;
edit1.Clear ;
end
else
begin
showmessage (' Enter the Password Carefully ');
edit1.clear;
end;
end;
end.

```

## 2. Holder Information Screen Code

```

unit Unit2;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls, DB, DBTables, ADODB, Mask;

type

TForm2 = class(TForm)

```



Button1: TButton;

Button2: TButton;

Panel2: TPanel;

Edit1: TEdit;

Edit2: TEdit;

Edit3: TEdit;

Edit4: TEdit;

Edit6: TEdit;

Edit8: TEdit;

Edit7: TEdit;

Edit9: TEdit;

Edit10: TEdit;

Edit11: TEdit;

Panel1: TPanel;

Panel3: TPanel;

Panel4: TPanel;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

Label7: TLabel;

Label9: TLabel;

Panel5: TPanel;

Label8: TLabel;

```
Label10: TLabel;  
Label11: TLabel;  
Label12: TLabel;  
Label14: TLabel;  
Edit13: TEdit;  
Edit14: TEdit;  
Button3: TButton;  
Panel6: TPanel;  
Label15: TLabel;  
DataSource1: TDataSource;  
Table1: TTable;  
MaskEdit1: TMaskEdit;  
MaskEdit2: TMaskEdit;  
MaskEdit3: TMaskEdit;  
MaskEdit4: TMaskEdit;  
Label16: TLabel;  
Label17: TLabel;  
RadioGroup1: TRadioGroup;  
RadioButton1: TRadioButton;  
RadioButton2: TRadioButton;  
DataSource2: TDataSource;  
Table2: TTable;  
procedure FormCreate(Sender: TObject);  
procedure Button1Click(Sender: TObject);  
procedure Button2Click(Sender: TObject);  
procedure Button3Click(Sender: TObject);
```

```

private
    { Private declarations }

public
    { Public declarations }

end;

var
    Form2: TForm2;

implementation

uses com400, Unit4, Unit5, Unit12, Unit10;

{$R *.dfm}

procedure TForm2.FormCreate(Sender: TObject);
begin
    // .EDIT14.Text:=EDIT14.Text;

    edit1.Clear;

    edit2.Clear;

    edit3.Clear;

    edit4.Clear;

    maskedit1.Clear;

    edit6.CLEAR;

    edit7.CLEAR;

    edit8.CLEAR;

    edit9.CLEAR;

    edit10.CLEAR;

    maskedit2.CLEAR;

    edit11.CLEAR;

    edit13.CLEAR;

```



```

end;

procedure TForm2.Button1Click(Sender: TObject);
begin
    form4.show;
    form2.Visible:=false;
end;

```

```

procedure TForm2.Button2Click(Sender: TObject);
begin
    if radiobutton1.Checked then
    begin
        with Table1 do begin
            Insert;

            fieldbyname('file_number').value:=edit14.TEXT;
            fieldbyname('NAME').value:=edit1.TEXT;
            fieldbyname('SURNAME').value:=edit2.Text;
            fieldbyname('id_no').value:=edit3.Text;
            fieldbyname('address').value:=edit4.Text;
            fieldbyname('dob').value:=maskedit2.Text;
            fieldbyname('Nationality').value:=edit6.Text;
            fieldbyname('pasp_no').value:=edit7.Text;
            fieldbyname('risda_no').value:=edit8.Text;
            fieldbyname('Issue').value:=maskedit3.Text;
            fieldbyname('Expiry').value:=maskedit4.Text;
            fieldbyname('tele').value:=edit9.Text;

```

```

        fieldbyname('orig').value:=edit10.Text;

        fieldbyname('form_number').value:=edit11.Text;

        fieldbyname('entry_date').value:=maskedit1.Text;

        fieldbyname('comp_name').value:=edit13.Text;

        Post;

        Refresh;

    end;

    showmessage('Save the Data in Document in Public Cars.... ');

    form5.show;

    form2.visible:=false;

    form5.Edit1.Text:=edit1.Text;

    form5.Edit2.Text:=edit2.Text;

    form5.Edit3.Text:=edit3.Text;

    form5.Edit10.Text:=edit14.Text;

    form12.Edit8.Text:=edit1.Text;

    form12.Edit9.Text:=edit2.Text;

    form12.Edit10.Text:=edit3.Text;

    form12.Edit11.Text:=edit14.Text;

    edit4.Clear;

    maskedit2.Clear;

    edit6.Clear;

    form5.show;

    form2.Close; end

else

    if radiobutton2.Checked then

        begin

```

```

with Table2 do begin
Insert;

    fieldbyname('file_number').value:=edit14.TEXT;
    fieldbyname('NAME').value:=edit1.TEXT;
    fieldbyname('SURNAME').value:=edit2.Text;
    fieldbyname('id_no').value:=edit3.Text;
    fieldbyname('address').value:=edit4.Text;
    fieldbyname('dob').value:=maskedit2.Text;
    fieldbyname('Nationality').value:=edit6.Text;
    fieldbyname('pasp_no').value:=edit7.Text;
    fieldbyname('risda_no').value:=edit8.Text;
    fieldbyname('Issue').value:=maskedit3.Text;
    fieldbyname('Expiry').value:=maskedit4.Text;
    fieldbyname('tele').value:=edit9.Text;
    fieldbyname('orig').value:=edit10.Text;
    fieldbyname('form_number').value:=edit11.Text;
    fieldbyname('entry_date').value:=maskedit1.Text;
    fieldbyname('comp_name').value:=edit13.Text;

Post;

Refresh;

end;

showmessage('Save the Data in Document in Special Cars.... ');
form5.show;

form2.visible:=false;

form5.Edit1.Text:=edit1.Text;
form5.Edit2.Text:=edit2.Text;

```

```
form5.Edit3.Text:=edit3.Text;  
form5.Edit10.Text:=edit14.Text;  
form12.Edit8.Text:=edit1.Text;  
form12.Edit9.Text:=edit2.Text;  
form12.Edit10.Text:=edit3.Text;  
form12.Edit11.Text:=edit14.Text;  
edit4.Clear;  
maskedit2.Clear;  
edit6.Clear;  
form5.show;  
form2.Close; end;  
end;
```

```
procedure TForm2.Button3Click(Sender: TObject);  
var  
  x,y,code:integer;  
  xh:string;  
begin  
  val (edit11.Text,x,code);  
  y:=x-100;  
  str (y,xh);  
  edit14.Text:=xh;  
end;  
end.
```



### 3. Car Information Code

```
unit Unit5;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Menus, StdCtrls, ExtCtrls, Mask, DBCtrls, DB, DBTables;

type
    TForm5 = class(TForm)
        Edit5: TEdit;
        Panel1: TPanel;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        Label7: TLabel;
        Label8: TLabel;
        Panel3: TPanel;
        Edit7: TEdit;
        Edit8: TEdit;
        Label9: TLabel;
        Label10: TLabel;
        Edit15: TEdit;
        Label13: TLabel;
        Label14: TLabel;
```

Label15: TLabel;  
Label16: TLabel;  
ComboBox1: TComboBox;  
ComboBox3: TComboBox;  
Label17: TLabel;  
Label11: TLabel;  
Panel2: TPanel;  
Edit9: TEdit;  
Label12: TLabel;  
Panel4: TPanel;  
Edit1: TEdit;  
Edit2: TEdit;  
Edit3: TEdit;  
Label18: TLabel;  
Edit11: TEdit;  
Label19: TLabel;  
Edit12: TEdit;  
Edit14: TEdit;  
Edit17: TEdit;  
Label20: TLabel;  
MaskEdit1: TMaskEdit;  
Table1: TTable;  
DataSource1: TDataSource;  
Edit10: TEdit;  
Label21: TLabel;  
Label22: TLabel;

```

Label23: TLabel;
Label24: TLabel;
DataSource2: TDataSource;
Table2: TTable;
Button2: TButton;
Label25: TLabel;
Memo1: TMemo;
Memo2: TMemo;
Memo3: TMemo;
Memo4: TMemo;

//procedure SetOption1Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
//procedure FormCreate(Sender: TObject);
//procedure Option2Click(Sender: TObject);
//procedure Button2Click(Sender: TObject);
//procedure FormCreate(Sender: TObject);

private
{ Private declarations }

public
{ Public declarations }

end;

var
Form5: TForm5;

```

```
implementation
uses Unit7, Unit9, Unit11, Unit12, Unit2, Unit14, Unit10;
{$R *.dfm}
```

```
procedure TForm5.Button1Click(Sender: TObject);
var
    code,x,y:integer;
begin
    val (form10.spinedit1.text,y,code);//form set optiton 10
    val (edit5.Text,x,code) ;//from 5
    if (x >= y) then {begin
        {if form2.radiobutton1.checked then begin
            with Table1 do begin
                Insert;

                fieldbyname('date').value:=maskedit1.Text;
                fieldbyname('kind').value:=combobox3.TEXT;
                fieldbyname('type').value:=edit4.TEXT;
                fieldbyname('car_model').value:=edit5.Text;
                fieldbyname('motor_number').value:=edit6.Text;
                fieldbyname('doors_number').value:=edit12.Text;
                fieldbyname('type_of_foul').value:=combobox1.Text;
                fieldbyname('body_number').value:=edit13.Text;
                fieldbyname('persons').value:=edit14.Text;
                fieldbyname('Car_wieght').value:=edit15.Text;
                fieldbyname('remark').value:=edit16.Text;
                fieldbyname('Motor_capacity').value:=edit17.Text;
```



```

fieldbyname('car_color').value:=edit8.Text;
fieldbyname('loading').value:=edit17.Text;
fieldbyname('Auditor_name').value:=edit9.Text;
fieldbyname('Auditor_number').value:=edit11.Text;
fieldbyname('File_number').value:=edit10.Text;
Post;
Refresh;
form12.show;
form5.close;
end;
end
else if form2.radiobutton2.checked then
begin
    with Table2 do begin
Insert;

        fieldbyname('date').value:=maskedit1.Text;
        fieldbyname('kind').value:=combobox3.TEXT;
        fieldbyname('type').value:=edit4.TEXT;
        fieldbyname('car_model').value:=edit5.Text;
        fieldbyname('motor_number').value:=edit6.Text;
        fieldbyname('doors_number').value:=edit12.Text;
        fieldbyname('type_of_foul').value:=combobox1.Text;
        fieldbyname('body_number').value:=edit13.Text;
        fieldbyname('persons').value:=edit14.Text;
        fieldbyname('Car_wieght').value:=edit15.Text;
        fieldbyname('remark').value:=edit16.Text;

```

```

fieldbyname('Motor_capacity').value:=edit17.Text;
fieldbyname('car_color').value:=edit8.Text;
fieldbyname('loading').value:=edit17.Text;
fieldbyname('Auditor_name').value:=edit9.Text;
fieldbyname('Auditor_number').value:=edit11.Text;
fieldbyname('File_number').value:=edit10.Text;

Post;

Refresh;

form12.show;

form5.close;

end;

end else }

panel4.Visible:=true;

end;

```

```

procedure TForm5.Button2Click(Sender: TObject);

var

code,x,y:integer;

begin

val (form10.spinedit1.text,y,code);//form set optiton 10

val (edit5.Text,x,code) ;//from 5

if (x >= y) then begin

if form2.radiobutton1.checked then begin

with Table1 do begin

Insert;

fieldbyname('date').value:=maskedit1.Text;

```

```

fieldbyname('kind').value:=combobox3.TEXT;
fieldbyname('type').value:=memo1.TEXT;
fieldbyname('car_model').value:=edit5.Text;
fieldbyname('motor_number').value:=memo2.Text;
fieldbyname('doors_number').value:=edit12.Text;
fieldbyname('type_of_foul').value:=combobox1.Text;
fieldbyname('body_number').value:=memo3.Text;
fieldbyname('persons').value:=edit14.Text;
fieldbyname('Car_wieght').value:=edit15.Text;
fieldbyname('remark').value:=memo4.Text;
fieldbyname('Motor_capacity').value:=edit17.Text;
fieldbyname('car_color').value:=edit8.Text;
fieldbyname('loading').value:=edit17.Text;
fieldbyname('Auditor_name').value:=edit9.Text;
fieldbyname('Auditor_number').value:=edit11.Text;
fieldbyname('File_number').value:=edit10.Text;

Post;

Refresh;

form12.show;

form5.close;

end;

end

else if form2.radiobutton2.checked then

begin

    with Table2 do begin

Insert;

```

```

fieldbyname('date').value:=maskedit1.Text;
fieldbyname('kind').value:=combobox3.TEXT;
fieldbyname('type').value:=memo1.TEXT;
fieldbyname('car_model').value:=edit5.Text;
fieldbyname('motor_number').value:=memo2.Text;
fieldbyname('doors_number').value:=edit12.Text;
fieldbyname('type_of_foul').value:=combobox1.Text;
fieldbyname('body_number').value:=memo3.Text;
fieldbyname('persons').value:=edit14.Text;
fieldbyname('Car_wieght').value:=edit15.Text;
fieldbyname('remark').value:=memo4.Text;
fieldbyname('Motor_capacity').value:=edit17.Text;
fieldbyname('car_color').value:=edit8.Text;
fieldbyname('loading').value:=edit17.Text;
fieldbyname('Auditor_name').value:=edit9.Text;
fieldbyname('Auditor_number').value:=edit11.Text;
fieldbyname('File_number').value:=edit10.Text;

Post;

Refresh;

end;

form12.show;

form5.close;

end ;

end;

panel4.Visible:=true;

end;

```



```

procedure TForm5.FormCreate(Sender: TObject);
begin
memo1.Clear;
memo2.Clear;
memo3.Clear;
memo4.Clear;
if form2.RadioButton1.Checked then
label23.Visible:=true
else
if form2.RadioButton2.Checked then
label24.Visible:=true;
end;
end.

```

#### 4. Calculate Payments Screen Code

```

unit Unit12;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Menus;

type
  TForm12 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit4: TEdit;

```

Edit5: TEdit;  
Edit6: TEdit;  
Label1: TLabel;  
Label2: TLabel;  
Label3: TLabel;  
Label4: TLabel;  
Label5: TLabel;  
Panel1: TPanel;  
Edit7: TEdit;  
Panel3: TPanel;  
Label7: TLabel;  
Label8: TLabel;  
Button2: TButton;  
Edit3: TEdit;  
Edit8: TEdit;  
Edit9: TEdit;  
Edit10: TEdit;  
Label6: TLabel;  
Label9: TLabel;  
Label10: TLabel;  
Panel2: TPanel;  
MainMenu1: TMainMenu;  
Option1: TMenuItem;  
SetOption1: TMenuItem;  
Button3: TButton;  
Edit11: TEdit;

```

form12.Enabled:=false;

end;

procedure TForm12.Button2Click(Sender: TObject);
var
code,CUSTOM,a,b,c,d,e,f,a10,b10,c10,d10,e10,f10,g10,h10,i10,j10,k10,l10,nettax:integer;
net:real;
aa,bb,cc,dd,ee,ff,aa10,bb10,cc10,dd10,ee10,ff10,gg10,hh10,ii10,jj10,kk10,ll10,grossprice:string;
begin
if form2.RadioButton1.Checked then begin
net:=0.0;
val (edit6.Text,a,code);
val (edit4.Text,f,code);
val (edit1.Text,b,code);
val (edit2.Text,c,code);
val (edit5.Text,d,code);
val (edit7.Text,e,code);
val (edit12.Text,CUSTOM,code);
val (form10.Edit1.Text,a10,code);
val (form10.Edit2.Text,b10,code);
val (form10.Edit3.Text,c10,code);
val (form10.Edit4.Text,d10,code);
val (form10.Edit5.Text,e10,code);
val (form10.Edit6.Text,f10,code);
val (form10.Edit7.Text,g10,code);

```

```

val (form10.Edit12.Text,h10,code);
val (form10.Edit8.Text,i10,code);
val (form10.Edit13.Text,j10,code);
val (form10.Edit16.Text,k10,code);
val (form10.Edit15.Text,l10,code);
str (a,aa);
str (b,bb);
str (c,cc);
str (d,dd);
str (e,ee);
str (f,ff);
str (a10,aa10);
str (b10,bb10);
str (c10,cc10);
str (d10,dd10);
str (e10,ee10);
str (f10,ff10);
str (g10,gg10);
str (h10,hh10);
str (i10,ii10);
str (j10,jj10);
str (k10,kk10);
str (l10,ll10);
nettax:=b+c+d+e+f;    //total tax
begin
if form9.checkbox1.checked then //auto gear

```



```

net:=net+a*a10/100; // net price for option
if form9.checkbox2.checked then //center luck
net:=net+a*b10 /100;
if form9.checkbox3.checked then // elec windows
net:=net+a*c10 /100;
if form9.checkbox4.checked then //air condution
net:=net+a*d10 /100;
if form9.checkbox5.checked then //power stiring
net:=net+a*e10 /100;
if form9.checkbox6.checked then //ingection
net:=net+a*f10 /100;
if form9.checkbox7.checked then //air bag
net:=net+a*g10 /100;
if form9.checkbox8.checked then //elec mirror
net:=net+a*h10/100;
if form9.radiobutton1.checked then //Roof
net:=net+a*i10 /100;
if form9.radiobutton2.checked then //Open roof
net:=net+a* j10 / 100;
if form9.checkbox10.checked then //audio system
net:=net+a*k10/100;
if form9.checkbox11.checked then //alert tone
net:=net+a*l10/100;
end;
net:=net+nettax+a+CUSTOM;
str (net:5:5,grossprice);

```

```

edit3.text:=grossprice; end
else
if form2.radiobutton2.checked then begin
net:=0.0;
val (edit6.Text,a,code);
val (edit4.Text,f,code);
val (edit1.Text,b,code);
val (edit2.Text,c,code);
val (edit5.Text,d,code);
val (edit7.Text,e,code);
val (edit12.Text,CUSTOM,code);
val (form19.Edit1.Text,a10,code);
val (form19.Edit2.Text,b10,code);
val (form19.Edit3.Text,c10,code);
val (form19.Edit4.Text,d10,code);
val (form19.Edit5.Text,e10,code);
val (form19.Edit6.Text,f10,code);
val (form19.Edit7.Text,g10,code);
val (form19.Edit12.Text,h10,code);
val (form19.Edit8.Text,i10,code);
val (form19.Edit13.Text,j10,code);
val (form19.Edit14.Text,k10,code);
val (form19.Edit15.Text,l10,code);
str (a,aa);
str (b,bb);
str (c,cc);

```

```

str (d,dd);
str (e,ee);
str (f,ff);
str (a10,aa10);
str (b10,bb10);
str (c10,cc10);
str (d10,dd10);
str (e10,ee10);
str (f10,ff10);
str (g10,gg10);
str (h10,hh10);
str (i10,ii10);
str (j10,jj10);
str (k10,kk10);
str (l10,ll10);
nettax:=b+c+d+e+f;    //total tax
begin
if form9.checkbox1.checked then //auto gear
net:=net+a*a10/100;      // net price for option
if form9.checkbox2.checked then //center luck
net:=net+a*b10 /100;
if form9.checkbox3.checked then // elec windows
net:=net+a*c10 /100;
if form9.checkbox4.checked then //air condution
net:=net+a*d10 /100;
if form9.checkbox5.checked then //power stiring

```

```

net:=net+a*e10 /100;
if form9.checkbox6.checked then //injection
net:=net+a*f10 /100;
if form9.checkbox7.checked then //air bag
net:=net+a*g10 /100;
if form9.checkbox8.checked then //elec mirror
net:=net+a*h10/100;
if form9.radiobutton1.checked then //Roof
net:=net+a*i10 /100;
if form9.radiobutton2.checked then //Open roof
net:=net+a*j10 / 100;
if form9.checkbox10.checked then //audio system
net:=net+a*k10/100;
if form9.checkbox11.checked then //alert tone
net:=net+a*l10/100;
end;
net:=net+nettax+a+CUSTOM;
str (net:5:5,grossprice);
edit3.text:=grossprice;
end;
end;

```

```

procedure TForm12.FormCreate(Sender: TObject);
begin
edit8.text:=form2.edit1.text ;
edit9.text:=form2.edit2.text ;

```



```

edit10.text:=form2.edit3.text ;
edit12.text:=form2.edit14.text ;
if form2.RadioButton1.Checked then begin
label11.Visible:=true;
edit12.Visible:=true;
label13.Visible:=true; end
else
if form2.RadioButton2.Checked then
label12.Visible:=true;
end;

```

```

procedure TForm12.SetOption1Click(Sender: TObject);
begin
if form2.radiobutton1.checked then begin
form10.show;
form12.Enabled:=false;
end
else
if form2.radiobutton2.checked then
form19.Show;
form12.Enabled:=false;
end;

```

```

procedure TForm12.Button3Click(Sender: TObject);
begin
form22.show;

```

end;

procedure TForm12.Button4Click(Sender: TObject);

begin

if form2.RadioButton1.Checked then

form26.show

else

if form2.RadioButton2.Checked then

form27.show;

end;

end.

NOTE: This codes for public and special operations.

## REFERENCES

- [1] Delphi Development Guide: Inprise press 1999.
- [2] Delphi Development Guide: Xavier Pacheco, Steve Teixeira. SAMS.1999.
- [3] Mastering Delphi 5: Marco Cantu. SYBEX. 1999.
- [4] Mastering Delphi 5: Marco Cantu. SYBEX. 1998.
- [5] Delphi HandBook: Marco Cantu. SYBEX. 1997.
- [6] WWW.Raize.com