

Petrol Bunk Management System

A dissertation submitted in partial fulfilment of the requirements for the award of degree

Bachelor of Computer Applications

BY

ARSATH ASAN ALIYAR S

REG NO: 2213181033010

UNDER THE GUIDANCE OF

Dr. G. BHARANIDHARAN, M.C.A., M.Phil., Ph.D.,

Assistant Professor



DEPARTMENT OF COMPUTER APPLICATIONS
THE NEW COLLEGE (AUTONOMOUS) CHENNAI – 600014

April-2025

DEPARTMENT OF COMPUTER APPLICATIONS

THE NEW COLLEGE (AUTONOMOUS) CHENNAI - 600014

APRIL-2025



BONAFIDE CERTIFICATE

This is to certify that the project work Entitled “ **PETROL BUNK MANAGEMENT SYSTEM** ” is a Bonafide record work done by **Mr ARSATH ASAN ALIYAR S** , bearing **Register Number: 2213181033010** has completed his work in a partial fulfilment of the requirement for the award of the degree of Bachelors of Computer Applications.

Project Guide

Head of the Department

Submitted for the Viva Voce Examination held on

Internal Examiner

External Examiner

Place :

Date :

ACKNOWLEDGEMENT

“In the name of Allah, the most Beneficent, the most Merciful”

I would like to thank our **Principal Dr. M. ASRAR SHERIFF, M.Sc., M.Phil., Ph.D.**, for his support in my entire project work.

I also thank **Mr A. S. MOHAMED HAMSA M.C.A., M.Phil.** Head of the Department (Computer Applications), for his guidance and support to complete this project.

I am extremely grateful to my project guide. **Dr. G. BHARANIDHARAN, M.C.A., M.Phil., Ph.D., Assistant Professor of Computer Applications**, for suggesting me and guiding me at the right time and share the discussions on very useful topic needed in the course of the work.

I'm also grateful to **all the staff of Department of Computer Applications**. My heartfelt thanks goes to **my parents and friends** who encouraged me to do this mini project, and only because of their Best wishes and invaluable help, this project is seeing the light of the day.

ARSATH ASAN ALIYAR

ABSTRACT

Petrol Pump Management System project is developed using C# Language. The Project is based on a concept to maintain and generate the petrol pump's transaction. Talking about the system, Before entering the main menu, the user should pass through the login system to get access. From, where the user can add/edit supplier account, staff details, supplier withdraw pump account details, customer deposit pump account detail, and customer bill.

While entering a supplier account, he/she has to supplier id, supplier name, and supplier address. Similarly in order to enter staff details, name, address, city, phone number, D.O.B, and joined date respectively. Likewise machines that will helps to order the fuel machines, order to quantity of machines, In option to choose company, description details and fuel management, he/she has to fuel id, fuel name, fuel price, fuel supplier.

The stock details can also be mentioned here and the main essential feature is about billing which is available in this project. In order to billing section, the user has to provide machine, fuel, quantity, price amount. This details order to provide bill to customer.

INTRODUCTION

Introduction

The Main Aim of this petrol Bunk Management system is to track the sales of the daily activities in a bunk which makes the administrator to maintain the stocks and inventory details systematically. This Application created through Html, Css, Javascript as front end and PHP as back end for database mysql will be using, which makes a users to work easily to maintain the stock details and also calculate the daily transactions held in the bunk.

User Authentication and Authorization:

- Secure login system to ensure that only authorized personnel can access the system.
- Different user roles such as admin, cashier, and manager with specific privileges.

Inventory Management:

- Keep track of fuel stock levels, including petrol, diesel, and other related products.

Sales and Billing:

- Process fuel sales transactions efficiently.
- Generate detailed invoices for customers.
- Support for various payment methods such as cash, credit cards, or digital payments.

Customer Management:

- Maintain a customer database with details like name, contact information, and purchase history.

Employee Management:

- Record and manage employee details, attendance, and roles.
- Track employee performance and activities.

Reports and Analytics:

- Generate reports on daily, weekly, and monthly sales.
- Analyse trends and make informed decisions based on data.

Pump Monitoring:

- Monitor the status of fuel dispensers and pumps.
- Automated alerts for maintenance and malfunctions.

Security:

- Implement security measures to safeguard sensitive information.
- Audit trails to track system activities.

User-Friendly Interface:

- Intuitive and easy-to-use interface for quick navigation.
- Quick access to important features for faster transactions.

PROJECT MODULE

Login Page:

This module allows authorized users to access the system by entering their credentials, ensuring secure and controlled access to the system's functionalities and data.

Staffs:

The Staffs module manages information related to employees working at the petrol bunk. It includes features such as tracking employee attendance, salary details, contact information, and work timings.

Clients:

The Clients module focuses on managing customer information and interactions. It enables tracking of sales to individual clients, maintaining contact details, and possibly loyalty programs.

Stocks:

The Stocks module is responsible for inventory management, tracking the quantity of petrol, diesel, and oil available at the petrol bunk. It helps in monitoring stock levels, reordering, and managing stock transactions.

Sales: The Sales module handles all aspects related to sales transactions at the petrol bunk. It includes tracking sales of petrol, diesel.

Machines: The Machines module may refer to the management of equipment and machinery at the petrol bunk.

Report: The Report module plays a crucial role in generating various reports for decision-making and analysis. It includes functionalities to create reports on sales, stocks, employee performance, daily earnings, and other relevant data for management purposes.

HARDWARE AND SOFTWARE REQUIREMENTS

SYSTEM REQUIREMENT

Hardware Specification

Processor : AMD RYZAN 5 7640HS

Hard Disk : 512GB

Ram : 16GB

Software Specification

Text Editor : Visual Studio Dot net

Front End : Html, CSS, Bootstrap, JavaScript

Back End : Asp.Net C sharp

Database : SQL Server

Server : Microsoft IIS

Operating System : Windows 10 or Higher

SOFTWARE DESCRIPTION

INTRODUCTION TO .NET FRAMEWORK

The **Microsoft .NET Framework** is a vehicle technology that is available with several Microsoft Windows operating systems. It includes a large library of pre-coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.

The pre-coded solutions that form the framework's Base Class Library cover a large range of programming needs in a number of areas, including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications.

Programs written for the .NET Framework execute in a vehicle environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling. The class library and the CLR together compose the .NET Framework.

PRINCIPAL DESIGN FEATURES INTEROPERABILITY

Because interaction between new and older applications is commonly required, the .NET Framework provides means to access functionality that is implemented in programs that execute outside the .NET environment. Access to COM

components is provided in

the System.Runtime.InteropServices and System.EnterpriseServices namespaces of the framework; access to other functionality is provided using the P/Invoke feature.

COMMON RUNTIME ENGINE

The Common Language Runtime (CLR) is the virtual machine component of the .NET framework. All .NET programs execute under the supervision of the CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.

BASE CLASS LIBRARY

The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions, including file reading and writing, graphic rendering, database interaction and XML document manipulation.

SIMPLIFIED DEPLOYMENT

Installation of computer vehicle must be carefully managed to ensure that it does not interfere with previously installed vehicle, and that it conforms to security requirements. .NET framework includes design features and tools that help address these requirements.

SECURITY

The design is meant to address some of the vulnerabilities, such as buffer overflows, that have been exploited by malicious vehicle. Additionally, .NET provides a common security model for all applications.

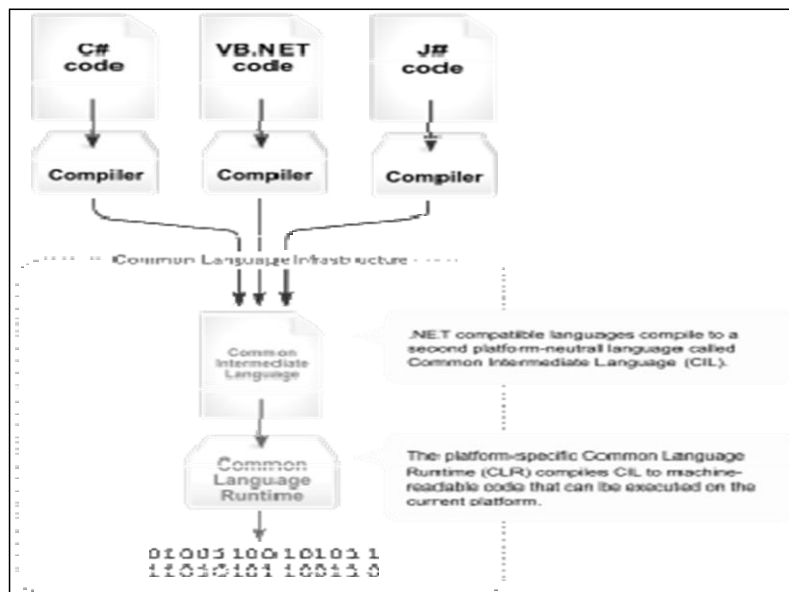
PORTABILITY

The design of the .NET Framework allows it to theoretically be platform agnostic, and thus cross-platform compatible. That is, a program written to use the framework should run without change on any type of system for which the framework is implemented. Microsoft's commercial implementations of the framework cover Windows, Windows CE, and the Xbox

360.

In addition, Microsoft submits the specifications for the Common Language Infrastructure (which includes the core class libraries, Common Type System, and the Common Intermediate Language), the C# language, and the C++/CLI language to both ECMA and the ISO, making them available as open standards. This makes it possible for third parties to create compatible implementations of the framework and its languages on other platforms.

ARCHITECTURE



Visual overview of the Common Language Infrastructure (CLI)

COMMON LANGUAGE INFRASTRUCTURE

The core aspects of the **.NET framework** lie within the Common Language Infrastructure, or **CLI**. The purpose of the CLI is to provide a language-neutral platform for application development and execution, including functions for exception handling, garbage collection, security, and interoperability. Microsoft's implementation of the CLI is called the **Common Language Runtime** or **CLR**.

ASSEMBLIES

The intermediate CIL code is housed in .NET assemblies. As mandated by specification, assemblies are stored in the Portable Executable (PE) format, common on the Windows platform for all DLL and EXE files. The assembly consists of one or more files, one of which must contain the manifest, which has the metadata for the assembly. The complete name of an assembly (not to be confused with the filename on disk) contains its simple text name, version number, culture, and public key token.

The public key token is a unique hash generated when the assembly is compiled, thus two assemblies with the same public key token are guaranteed to be identical from the point of view of the framework. A private key can also be specified known only to the creator of the assembly and can be used for strong naming and to guarantee that the assembly is from the same author when a new version of the assembly is compiled (required to add an assembly to the Global Assembly Cache).

METADATA

All CLI is self-describing through .NET metadata. The CLR checks the metadata to ensure that the correct method is called. Metadata is usually generated by language compilers but developers can create their own metadata through custom attributes. Metadata contains information about the assembly, and is also used to implement the reflective programming capabilities of .NET Framework.

SECURITY

.NET has its own security mechanism with two general features: Code Access Security (CAS), and validation and verification. Code Access Security is based on evidence that is associated with a specific assembly. Typically the evidence is the source of the assembly (whether it is installed on the local machine or has been downloaded from the intranet or Internet). Code Access Security uses evidence to determine the permissions granted to the code. Other code can demand that calling code is granted a specified permission. The demand causes the CLR to perform a call stack walk: every assembly of each method in the call stack is checked for the required permission; if any assembly is not granted the permission a security exception is thrown.

When an assembly is loaded the CLR performs various tests. Two such tests are validation and verification. During validation the CLR checks that the assembly contains valid metadata and CIL, and whether the internal tables are correct. Verification is not so exact. The verification mechanism checks to see if the code does anything that is 'unsafe'. The algorithm used is quite conservative; hence

occasionally code that is 'safe' does not pass. Unsafe code will only be executed if the assembly has the 'skip verification' permission, which generally means code that is installed on the local machine.

NET Framework uses appdomains as a mechanism for isolating code running in a process. Appdomains can be created and code loaded into or unloaded from them independent of other appdomains. This helps increase the fault tolerance of the application, as faults or crashes in one appdomain do not affect rest of the application. Appdomains can also be configured independently with different security privileges. This can help increase the security of the application by isolating potentially unsafe code. The developer, however, has to split the application into sub domains; it is not done by the CLR.

CLASS LIBRARY

Namespaces in the BCL

System

System. Code Dom System. Collections System. Diagnostics System. Globalization System. IO

System. Resources

System. Text

System. Text. Regular Expressions

Microsoft **.NET Framework** includes a set of standard **class libraries**. The class library is organized in a hierarchy of namespaces. Most of the built in APIs are part of either System.* or Microsoft.* namespaces. It encapsulates a large number of

common functions, such as file reading and writing, graphic rendering, database interaction, and XML document manipulation, among others. The .NET class libraries are available to all .NET languages. The .NET Framework class library is divided into two parts: the **Base Class Library** and the **Framework Class Library**.

The **Base Class Library** (BCL) includes a small subset of the entire class library and is the core set of classes that serve as the basic API of the Common Language Runtime. The classes in mscorlib.dll and some of the classes in System.dll and System.core.dll are considered to be a part of the BCL. The BCL classes are available in both .NET Framework as well as its alternative implementations including .NET Compact Framework, Microsoft Silver light and Mono. The **Framework Class Library** (FCL) is a superset of the BCL classes and refers to the entire class library that ships with .NET Framework. It includes an expanded set of libraries, including Win Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation among others. The FCL is much larger in scope than standard libraries for languages like C++, and comparable in scope to the standard libraries of Java.

MEMORY MANAGEMENT

The .NET Framework CLR frees the developer from the burden of managing memory (allocating and freeing up when done); instead it does the memory management itself. To this end, the memory allocated to instantiations of .NET types (objects) is done contiguously from the managed heap, a pool of memory

managed by the CLR. As long as there exists a reference to an object, which might be either a direct reference to an object or via a graph of objects, the object is considered to be in use by the CLR. When there is no reference to an object, and it cannot be reached or used, it becomes garbage. However, it still holds on to the memory allocated to it. .NET Framework includes a garbage collector which runs periodically, on a separate thread from the application's thread, that enumerates all the unusable objects and reclaims the memory allocated to them.

The .NET Garbage Collector (GC) is a non-deterministic, compacting, mark-and-sweep garbage collector. The GC runs only when a certain amount of memory has been used or there is enough pressure for memory on the system. Since it is not guaranteed when the conditions to reclaim memory are reached, the GC runs are non-deterministic. Each .NET application has a set of roots, which are pointers to objects on the managed heap (*managed objects*). These include references to static objects and objects defined as local variables or method parameters currently in scope, as well as objects referred to by CPU registers.

When the GC runs, it pauses the application, and for each object referred to in the root, it recursively enumerates all the objects reachable from the root objects and marks them as reachable. It uses .NET metadata and reflection to discover the objects encapsulated by an object, and then recursively walk them. It then enumerates all the objects on the heap (which were initially allocated contiguously) using reflection. All objects not marked as reachable are garbage. This is the *mark* phase. Since the memory held by garbage is not of any consequence, it is

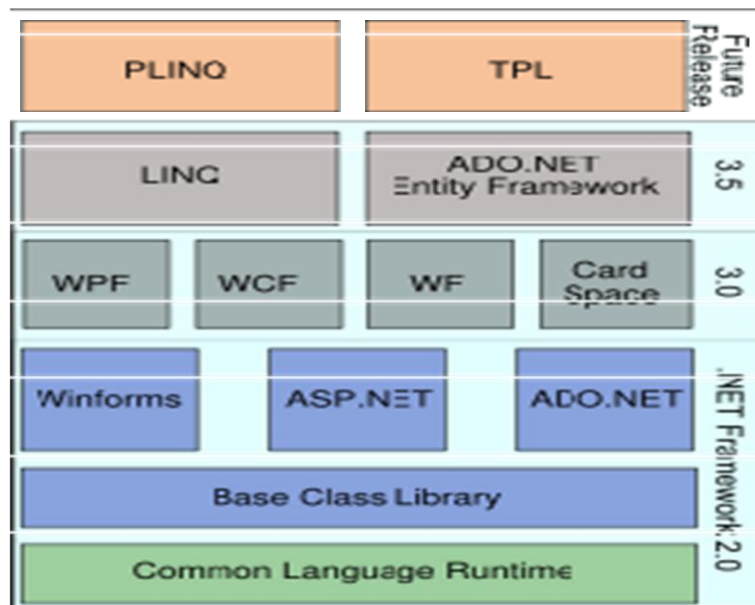
considered free space. However, this leaves chunks of free space between objects which were initially contiguous. The objects are then *compacted* together, by using memory to copy them over to the free space to make them contiguous again. Any reference to an object invalidated by moving the object is updated to reflect the new location by the GC.

The application is resumed after the garbage collection is over. The GC used by .NET Framework is actually *generational*. Objects are assigned a *generation*; newly created objects belong to *Generation 0*. The objects that survive a garbage collection are tagged as *Generation 1*, and the *Generation 1* objects that survive another collection are *Generation 2* objects. The .NET Framework uses up to *Generation 2* objects. Higher generation objects are garbage collected less frequently than lower generation objects. This helps increase the efficiency of garbage collection, as older objects tend to have a larger lifetime than newer objects. Thus, by removing older (and thus more likely to survive a collection) objects from the scope of a collection run, fewer objects need to be checked and compacted.

FRAMEWORK

Microsoft started development on the .NET Framework in the late 1990s originally under the name of Next Generation Windows Services (NGWS). By late 2000 the first beta versions of

.NET 1.0 was release



The .NET Framework stack.

Version	Version Number	Release Date
1.0	1.0.3705.0	2002-01-05
1.1	1.1.4322.573	2003-04-01
2.0	2.0.50727.42	2005-11-07
3.0	3.0.4506.30	2006-11-06
3.5	3.5.21022.8	2007-11-09

C#.NET

ACTIVE X DATA OBJECTS.NET

ADO.NET Overview

ADO.NET is an evolution of the ADO data access model that directly addresses user requirements for developing scalable applications. It was designed specifically for the web with scalability, statelessness, and XML in mind.

ADO.NET uses some ADO objects, such as the **Connection** and **Command** objects, and also introduces new objects. Key new ADO.NET objects include the **Dataset**, **Data Reader**, and **Data Adapter**.

The important distinction between this evolved stage of ADO.NET and previous data architectures is that there exists an object -- the **Dataset** -- that is separate and distinct from any data stores. Because of that, the **Dataset** functions as a standalone entity. You can think of the Dataset as an always disconnected record set that knows nothing about the source or destination of the data it contains. Inside a **Dataset**, much like in a database, there are tables, columns, relationships, constraints, views, and so forth.

A **Data Adapter** is the object that connects to the database to fill the **Dataset**. Then, it connects back to the database to update the data there, based on operations performed while the **Dataset** held the data. In the past, data processing has been primarily connection-based. Now, in an effort to make multi-tiered apps more efficient, data processing is turning to a message-based approach that revolves around chunks of information. At the center of this approach is the **Data Adapter**, which provides a bridge to retrieve and save data between a **Dataset** and its source data

store. It accomplishes this by means of requests to the appropriate SQL commands made against the data store.

The XML-based **Dataset** object provides a consistent programming model that works with all models of data storage: flat, relational, and hierarchical. It does this by having no 'knowledge' of the source of its data, and by representing the data that it holds as collections and data types. No matter what the source of the data within the **Dataset** is, it is manipulated through the same set of standard APIs exposed through the **Dataset** and its subordinate objects.

While the **Dataset** has no knowledge of the source of its data, the managed provider has detailed and specific information. The role of the managed provider is to connect, fill, and persist the **Dataset** to and from data stores. The OLE DB and SQL Server .NET Data Providers (System.Data.OleDb and System.Data.SqlClient) that are part of the .Net Framework provide four basic objects: the **Command**, **Connection**, **Data Reader** and **Data Adapter**. In the remaining sections of this document, we'll walk through each part of the **Dataset** and the OLE DB/SQL Server .NET Data Providers explaining what they are, and how to program against them.

The following sections will introduce you to some objects that have evolved, and some that are new. These objects are:

- **Connections.** For connection to and managing transactions against a database.
- **Commands.** For issuing SQL commands against a database.
- **Data Readers.** For reading a forward-only stream of data records from a SQL Server data source.

- **Datasets.** For storing, remoting and programming against flat data, XML data and relational data.
- **Data Adapters.** For pushing data into a **Dataset**, and reconciling data against a database.

When dealing with connections to a database, there are two different options: SQL Server .NET Data Provider (System.Data.SqlClient) and OLE DB .NET Data Provider (System.Data.OleDb). In these samples we will use the SQL Server .NET Data Provider. These are written to talk directly to Microsoft SQL Server. The OLE DB .NET Data Provider is used to talk to any OLE DB provider (as it uses OLE DB underneath).

Connections

Connections are used to 'talk to' databases, and are represented by provider-specific classes such as **SQL Connection**. Commands travel over connections and result sets are returned in the form of streams which can be read by a **Data Reader** object, or pushed into a **Dataset** object.

Commands

Commands contain the information that is submitted to a database, and are represented by provider-specific classes such as **SQL Command**. A command can be a stored procedure call, an UPDATE statement, or a statement that returns results. You can also use input and output parameters, and return values as part of your command syntax. The example below shows how to issue an INSERT statement against the **Northwind** database.

Data Readers

The **Data Reader** object is somewhat synonymous with a read-only/forward-only cursor over data. The **Data Reader** API supports flat as well as hierarchical data. A **Data Reader** object is returned after executing a command against a database. The format of the returned **Data Reader** object is different from a record set. For example, you might use the **Data Reader** to show the results of a search list in a web page.

Data Sets and Data Adapters

Datasets

The **Dataset** object is similar to the ADO **Record set** object, but more powerful, and with one other important distinction: the **Dataset** is always disconnected. The **Dataset** object represents a cache of data, with database-like structures such as tables, columns, relationships, and constraints. However, though a **Dataset** can and does behave much like a database, it is important to remember that **Dataset** objects do not interact directly with databases, or other source data. This allows the developer to work with a programming model that is always consistent, regardless of where the source data resides. Data coming from a database, an XML file, from code, or user input can all be placed into **Dataset** objects. Then, as changes are made to the **Dataset** they can be tracked and verified before updating the source data. The **Get Changes** method of the **Dataset** object actually creates a second **Dataset** that contains only the changes to the data. This **Dataset** is then used by a **Data Adapter** (or other objects) to update the original data source.

The **Dataset** has many XML characteristics, including the ability to produce and consume XML data and XML schemas. XML schemas can be used to describe

schemas interchanged via Web Services. In fact, a **Dataset** with a schema can actually be compiled for type safety and statement completion.

Data Adapters (OLEDB/SQL)

The **Data Adapter** object works as a bridge between the **Dataset** and the source data. Using the provider-specific **SQL Data Adapter** (along with its associated **SQL Command** and **SQL Connection**) can increase overall performance when working with a Microsoft SQL Server databases. For other OLE DB-supported databases, you would use the **OleDbData Adapter** object and its associated **OleDb Command** and **OleDb Connection** objects.

The **Data Adapter** object uses commands to update the data source after changes have been made to the **Dataset**. Using the **Fill** method of the **Data Adapter** calls the **SELECT** command; using the **Update** method calls the **INSERT**, **UPDATE** or **DELETE** command for each changed row. You can explicitly set these commands in order to control the statements used at runtime to resolve changes, including the use of stored procedures. For ad-hoc scenarios, a **Command Builder** object can generate these at run-time based upon a select statement. However, this run-time generation requires an extra round-trip to the server in order to gather required metadata, so explicitly providing the **INSERT**, **UPDATE**, and **DELETE** commands at design time will result in better run-time performance.

1. ADO.NET is the next evolution of ADO for the .Net Framework.
2. ADO.NET was created with n-Tier, statelessness and XML in the forefront.

Two new objects, the **Dataset** and **Data Adapter**, are provided for these scenarios.

3. ADO.NET can be used to get data from a stream, or to store data in a cache for updates.
4. There is a lot more information about ADO.NET in the documentation.
5. Remember, you can execute a command directly against the database in order to do inserts, updates, and deletes. You don't need to first put data into a **Dataset** in order to insert, update, or delete it.
6. Also, you can use a **Dataset** to bind to the data, move through the data, and navigate data relationships

4.4 SQL SERVER -2008

A database management, or DBMS, gives the user access to their data and helps them transform the data into information. Such database management systems include dBase, paradox, IMS, SQL Server and SQL Server. These systems allow users to create, update and extract information from their database.

A database is a structured collection of data. Data refers to the characteristics of people, things and events. SQL Server stores each data item in its own fields. In SQL Server, the fields relating to a particular person, thing or event are bundled together to form a single complete unit of data, called a record (it can also be referred to as raw or an occurrence). Each record is made up of a number of fields. No two fields in a record can have the same field name.

During an SQL Server Database design project, the analysis of your business needs identifies all the fields or attributes of interest. If your business needs change over time, you define any additional fields or change the definition of existing fields.

SQL SERVER TABLES

SQL Server stores records relating to each other in a table. Different tables are created for the various groups of information. Related tables are grouped together to form a database.

PRIMARY KEY

Every table in SQL Server has a field or a combination of fields that uniquely identifies each record in the table. The Unique identifier is called the Primary Key, or simply the Key. The primary key provides the means to distinguish one record from all other in a table. It allows the user and the database system to identify, locate and refer to one particular record in the database.

RELATIONAL DATABASE

Sometimes all the information of interest to a business operation can be stored in one table. SQL Server makes it very easy to link the data in multiple tables. Matching an employee to the department in which they work is one example. This is what makes SQL Server a relational database management system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the table and enables you to define relationships between the tables.

FOREIGN KEY

When a field in one table matches the primary key of another field is referred to as a foreign key. A foreign key is a field or a group of fields in one table whose

values match those of the primary key of another table.

REFERENTIAL INTEGRITY

Not only does SQL Server allow you to link multiple tables, it also maintains consistency between them. Ensuring that the data among related tables is correctly matched is referred to as maintaining referential integrity.

DATA ABSTRACTION

A major purpose of a database system is to provide users with an abstract view of the data. This system hides certain details of how the data is stored and maintained. Data abstraction is divided into vehicles levels.

Physical level: This is the lowest level of abstraction at which one describes how the data are actually stored.

Conceptual Level: At this level of database abstraction all the attributed and what data are actually stored is described and entries and relationship among them.

View level: This is the highest level of abstraction at which one describes only part of the database.

ADVANTAGES OF RDBMS

- Redundancy can be avoided
- Inconsistency can be eliminated
- Data can be Shared
- Standards can be enforced
- Security restrictions can be applied
- Integrity can be maintained
- Conflicting requirements can be balanced
- Data independence can be achieved.

DISADVANTAGES OF DBMS

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing or developing the software, the hardware has to be upgraded to allow for the extensive programs and the workspace required for their execution and storage. While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in case of failure the data can be recovered.

FEATURES OF SQL SERVER (RDBMS)

SQL SERVER is one of the leading database management systems (DBMS) because it is the only Database that meets the uncompromising requirements of today's most demanding information systems. From complex decision support systems (DSS) to the most rigorous online transaction processing (OLTP) application, even application that require simultaneous DSS and OLTP access to the same critical

data, SQL Server leads the industry in both performance and capability.

SQL SERVER is a truly portable, distributed, and open DBMS that delivers unmatched performance, continuous operation and support for every database.

SQL SERVER RDBMS is high performance fault tolerant DBMS which is specially designed for online transactions processing and for handling large database application.

SQL SERVER with transactions processing option offers two features which contribute to very high level of transaction processing throughput, which are the row level lock manager

ENTERPRISE WIDE DATA SHARING

The unrivaled portability and connectivity of the SQL SERVER DBMS enables all the systems in the organization to be linked into a singular, integrated computing resource.

PORTABILITY

SQL SERVER is fully portable to more than 80 distinct hardware and operating systems platforms, including UNIX, MSDOS, OS/2, Macintosh and dozens of proprietary platforms. This portability gives complete freedom to choose the database server platform that meets the system requirements.

OPEN SYSTEMS

SQL SERVER offers a leading implementation of industry –standard SQL.

SQL Server's open architecture integrates SQL SERVER and non –SQL SERVER DBMS with industry's most comprehensive collection of tools, application, and third party vehicle products SQL Server's Open architecture provides transparent access to data from other relational database and even non-relational database.

DISTRIBUTED DATA SHARING

SQL Server's networking and distributed database capabilities to access data stored on remote server with the same ease as if the information was stored on a single local computer. A single SQL statement can access data at multiple sites. You can store data where system requirements such as performance, security or availability dictate.

UNMATCHED PERFORMANCE

The most advanced architecture in the industry allows the SQL SERVER DBMS to deliver unmatched performance.

SOPHISTICATED CONCURRENCY CONTROL

Real World applications demand access to critical data. With most database Systems application becomes “contention bound” – which performance is limited not by the CPU power or by disk I/O, but user waiting on one another for data access. SQL Server employs full, unrestricted row-level locking and contention free queries to minimize and in many cases entirely eliminates contention wait

times.

NO I/O BOTTLENECKS

SQL Server's fast commit groups commit and deferred write technologies dramatically reduce disk I/O bottlenecks. While some database write whole data block to disk at commit time, SQL Server commits transactions with at most sequential log file on disk at commit time. On high throughput systems, one sequential writes typically group commit multiple transactions. Data read by the transaction remains as shared memory so that other transactions may access that data without reading it again from disk. Since fast commits write all data necessary to the recovery to the log file, modified blocks are written back to the database independently of the transaction commit, when written from memory to disk.

GUI STANDARDS

Labels

A label consists of read only text or graphics, it identifies components and communication the status of a process. You can use labels with a component or can use it to describe a group of components.

Text Field

A Text Field is a rectangular box that displays a single line or text. If the line is too long to fit in the text field, the text automatically Scrolls horizontally.

Password Field

A Password Field is a variation of text field. When you type in a password field, instead of characters asterisks (*) are displayed. The asterisks are referred to as a Masking agent.

List Box

A List Box is used to display a set of items. One can use a list to present user with a set of choices.

Command Buttons

The Command Buttons is a component with a rectangular box that displays a single line or text. The text typically consists of a single word that represents the action associated with that button.

REPORTS

The system should generate the following reports:

- Costumer's details list.

SYSTEM DESIGN

SYSTEM DESIGN

DEFINITION

The most creative and challenging face of the system development is System Design. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Design goes through the logical and physical stages of development.

In designing a new system, the system analyst must have a clear understanding of the objectives, which the design is aiming to fulfill. The first step is to determine how the output is to be produced and in what format. Second, input data and master files have to be designed to meet the requirements of the proposed output. The operational phases are handled through program construction and testing.

Design of a system can be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Thus system design is a solution to “how to” approach to the creation of a new system. This important phase provides the understanding and the procedural details necessary for implementing the system recommended in the feasibility study. The design step provides a data design, architectural design, and a procedural design.

Input Design

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties –

- It should serve specific purpose effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding –
 - What are the inputs needed for the system?
 - How end users respond to different elements of forms and screens.

Objectives for Input Design

The objectives of input design are –

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

Data Input Methods

It is important to design appropriate data input methods to prevent errors while entering data. These methods depend on whether the data is entered by customers in forms manually and later entered by data entry operators, or data is directly entered by users on the PCs.

A system should prevent user from making mistakes by –

- Clear form design by leaving enough space for writing legibly.
- Clear instructions to fill form.
- Clear form design.
- Reducing key strokes.
- Immediate error feedback.

Some of the popular data input methods are –

- Batch input method (Offline data input method)
- Online data input method
- Computer readable forms
- Interactive data input

Input Integrity Controls

Input integrity controls include a number of methods to eliminate common input errors by end-users. They also include checks on the value of individual fields; both for format and the completeness of all inputs.

Audit trails for data entry and other system operations are created using transaction logs which gives a record of all changes introduced in the database to provide security and means of recovery in case of any failure.

Output Design

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

Objectives of Output Design

The objectives of input design are –

- To develop output design that serves the intended purpose and eliminates the production of unwanted output.
- To develop the output design that meets the end users requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decisions.

Let us now go through various types of outputs –

External Outputs

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipients.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

Internal outputs

Internal outputs are present inside the system, and used by end-users and managers. They support the management in decision making and reporting.

There are three types of reports produced by management information –

- **Detailed Reports** – They contain present information which has almost no filtering or restriction generated to assist management planning and control.
- **Summary Reports** – They contain trends and potential problems which are categorized and summarized that are generated for managers who do not want details.
- **Exception Reports** – They contain exceptions, filtered data to some condition or standard before presenting it to the manager, as information.

Output Integrity Controls

Output integrity controls include routing codes to identify the receiving system, and verification messages to confirm successful receipt of messages that are handled by network protocol.

Printed or screen-format reports should include a date/time for report printing and the data. Multipage reports contain report title or description, and pagination. Pre-printed forms usually include a version number and effective date.

Forms Design

Both forms and reports are the product of input and output design and are business document consisting of specified data. The main difference is that forms provide fields for data input but reports are purely used for reading. For example, order forms, employment and credit application, etc.

- During form designing, the designers should know –
 - who will use them
 - where would they be delivered
 - the purpose of the form or report
- During form design, automated design tools enhance the developer's ability to prototype forms and reports and present them to end users for evaluation.

Objectives of Good Form Design

A good form design is necessary to ensure the following –

- To keep the screen simple by giving proper sequence, information, and clear captions.
- To meet the intended purpose by using appropriate forms.
- To ensure the completion of form with accuracy.
- To keep the forms attractive by using icons, inverse video, or blinking cursors etc.
- To facilitate navigation.

Types of Forms

Flat Forms

- It is a single copy form prepared manually or by a machine and printed on a paper. For additional copies of the original, carbon papers are inserted between copies.
- It is a simplest and inexpensive form to design, print, and reproduce, which uses less volume.

Unit Set/Snap out Forms

- These are papers with one-time carbons interleaved into unit sets for either handwritten or machine use.
- Carbons may be either blue or black, standard grade medium intensity. Generally, blue carbons are best for handwritten forms while black carbons are best for machine use.

Continuous strip/Fanfold Forms

- These are multiple unit forms joined in a continuous strip with perforations between each pair of forms.
- It is a less expensive method for large volume use.

No Carbon Required (NCR) Paper

- They use carbonless papers which have two chemical coatings (capsules), one on the face and the other on the back of a sheet of paper.
- When pressure is applied, the two capsules interact and create an image.

Software Detailed Design

A software module is the lowest level of design granularity in the system. Depending on the software development approach, there may be one or more modules per system. This section should provide enough detailed information about logic and data necessary to completely write source code for all modules in the system (and/or integrate COTS software programs).

If there are many modules or if the module documentation is extensive, place it in an appendix or reference a separate document. Add additional diagrams and information, if necessary, to describe each module, its functionality, and its hierarchy. Industry-standard module specification practices should be followed. Include the following information in the detailed module designs:

- A narrative description of each module, its function(s), the conditions under which it is used (called or scheduled for execution), its overall processing, logic, interfaces to other modules, interfaces to external systems, security requirements, etc.; explain any algorithms used by the module in detail
- For COTS packages, specify any call routines or bridging programs to integrate the package with the system and/or other COTS packages (for example, Dynamic Link Libraries)
- Data elements, record structures, and file structures associated with module input and output
- Graphical representation of the module processing, logic, flow of control, and algorithms, using an accepted diagramming approach (for example, structure charts, action diagrams, flowcharts, etc.)
- Data entry and data output graphics; define or reference associated data elements; if the project is large and complex or if the detailed module designs will be incorporated into a separate document, then it may be appropriate to repeat the screen information in this section
- Report layout

FILE Design AND DATABASE DESIGN

Interact with the Database Administrator (DBA) when preparing this section. The section should reveal the final design of all database management system (DBMS) files and the non-DBMS files associated with the system under development. Additional information may add as required for the particular project. Provide a comprehensive data dictionary showing data element name, type, length, source, validation rules, maintenance (create, read, update,

delete (CRUD) capability), data stores, outputs, aliases, and description. Can be included as an appendix.

Database Management System Files

This section reveals the final design of the DBMS files and includes the following information, as appropriate (refer to the data dictionary):

- Refined logical model; provide normalized table layouts, entity relationship diagrams, and other logical design information
- A physical description of the DBMS schemas, sub-schemas, records, sets, tables, storage page sizes, etc.
- Access methods (such as indexed, via set, sequential, random access, sorted pointer array, etc.)
- Estimate of the DBMS file size or volume of data within the file, and data pages, including overhead resulting from access methods and free space
- Definition of the update frequency of the database tables, views, files, areas, records, sets, and data pages; estimate the number of transactions if the database is an online transaction-based system

Non-Database Management System Files

In this section, provide the detailed description of all non-DBMS files and include a narrative description of the usage of each file—including if the file is used for input, output, or both; if this file is a temporary file; an indication of which modules read and write the file, etc.; and file structures (refer to the data dictionary). As appropriate, the file structure information should:

- Identify record structures, record keys or indexes, and reference data elements within the records
- Define record length (fixed or maximum variable length) and blocking factors
- Define file access method—for example, index sequential, virtual sequential, random access, etc.
- Estimate the file size or volume of data within the file, including overhead resulting from file access methods
- Define the update frequency of the file; if the file is part of an online transaction-based system, provide the estimated number of transactions per unit time, and the statistical mean, mode, and distribution of those transactions.

SYSTEM TESTING AND VALIDATION

SYSTEM TESTING AND VALIDATION

SYSTEM TESTING

System testing is defined as testing of a complete and fully integrated software product. This testing falls in black-box testing wherein knowledge of the inner design of the code is not a pre-requisite and is done by the testing team. It is the final test to verify that the product to be delivered meets the specifications mentioned in the requirement document. It should investigate both functional and non-functional requirements.

UNIT TETSING

The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behavior. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results.

INTEGRATION TESTING

In integration testing modules are combined and tested as a group. Modules are typically code modules, individual applications, source and destination applications on a network, etc. Integration Testing follows unit testing and precedes system

testing. Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large in hopes that they will buy the final product when it is released.

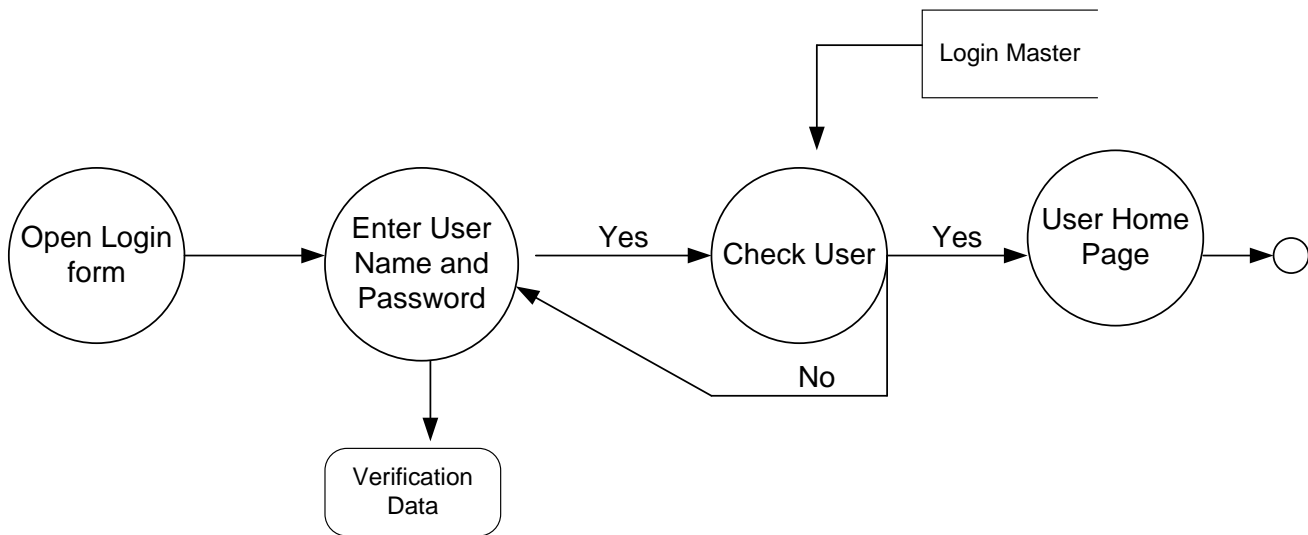
VALIDATION

Validation is the process of evaluating the final product to check whether the software meets the customer expectations and requirements. It is a dynamic mechanism of validating and testing the actual product. Validation is determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs. Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.

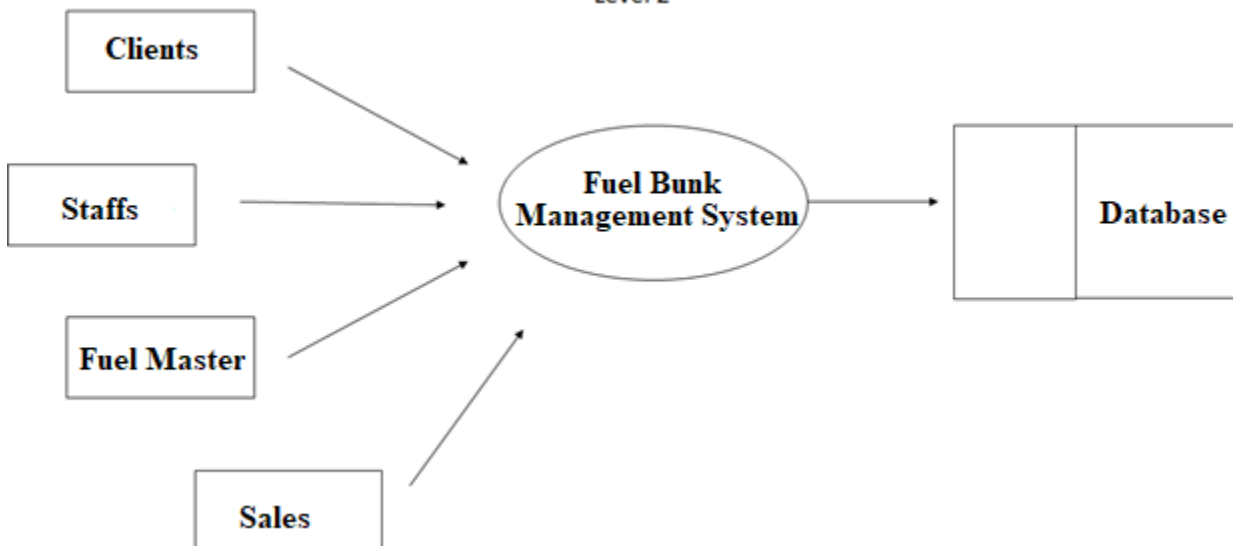
DFD DIAGRAM

DFD Diagram

Level 1

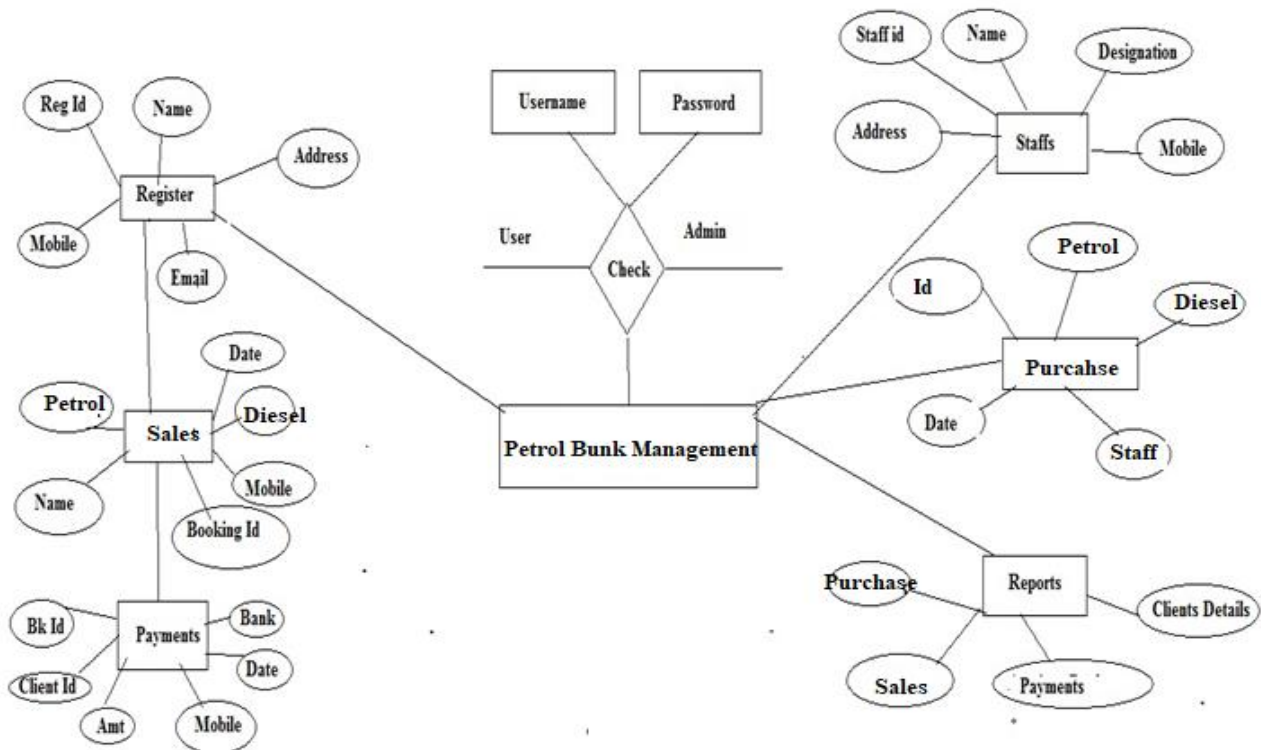


Level 2



ER DIAGRAM

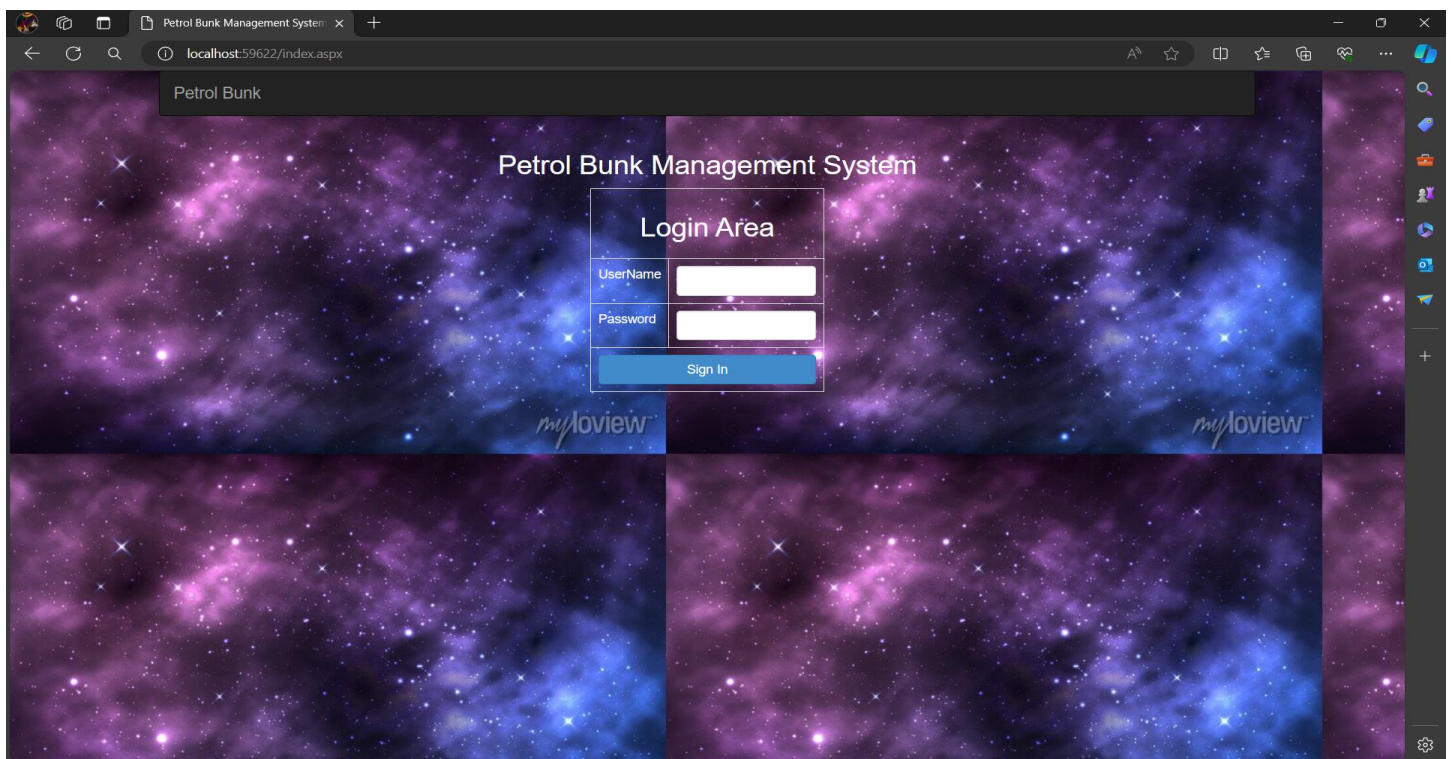
ER Diagram



Form Layout

Form Layout:

Login Page:



The screenshot shows a web browser window with the title "Petrol Bunk Management System". The address bar displays "localhost:59622/index.aspx". The page features a dark, starry background with a central "Login Area" box. The login area contains fields for "UserName" and "Password", and a "Sign In" button. The page is divided into four quadrants by a central vertical and horizontal line, each containing a "myloview" logo. A sidebar on the right side of the page contains various icons for navigation and settings.

Petrol Bunk

Petrol Bunk Management System

Login Area

UserName

Password

Sign In




myloview myloview

Staffs:

Petrol Bunk

StaffsClient MasterStock MasterSalesMachinesReportsLogout

Staffs Details

Id	staffname	qualification	department	mobile	address	Staffs Photo	Id	staffname	qualification	department	mobile	address	photo
1	Salf	B.C.A Computer Application	Supervisor	9874563214	Chennai		1	Salf	B.C.A Computer Application	Supervisor	9874563214	Chennai	~/profile/10.jpg
2	Ibrahim	B.C.A Computer Application	Supervisor	9876543212	Chennai		2	Ibrahim	B.C.A Computer Application	Supervisor	9876543212	Chennai	~/Uploads/1.jpeg
3	mufeeth	BBA Business	Billing	08526129118	chennai		3	mufeeth	BBA Business	Billing	08526129118	chennai	~/Uploads/default_profile.jpg

myloview

Staff Id

4

Staff Name

Qualification

Department

Billing

Mobile

Address

Profile Image

Choose File

No file chosen

Save

Update

Delete

Show

Clients Master:

Petrol Bunk

StaffsClient MasterStock MasterSalesMachinesReportsLogout

clientid	clientname	client_type	email	mobile	address	regdate
1	Parveen Travels	Travels	parveentravels@gmail.com	9874563214	Chennai Egmore	2024-03-19

Client Id2Client Name

Client TypeTravelsEmail

MobileAddressDatedd/mm/yyyy

SaveUpdateDeleteShow

myloview

myloview

Stock Master:

Petrol Bunk

StaffsClient MasterStock MasterSalesMachinesReportsLogout

	petrol	diesel
Id		
1	15900	15385

Stock Master

Invoice Id16

Purchase Datedd/mm/yyyy

Diesel

Staff InchargeIbrahim

Supplier Name

Petrol

Vehicle No

Save

myloview

Sales:

Petrol Bunk

StaffsClient MasterStock MasterSalesMachinesReportsLogout

Sales

Invoice Id7

Sales Datedd/mm/yyyy

Diesel

Staff NameIbrahim

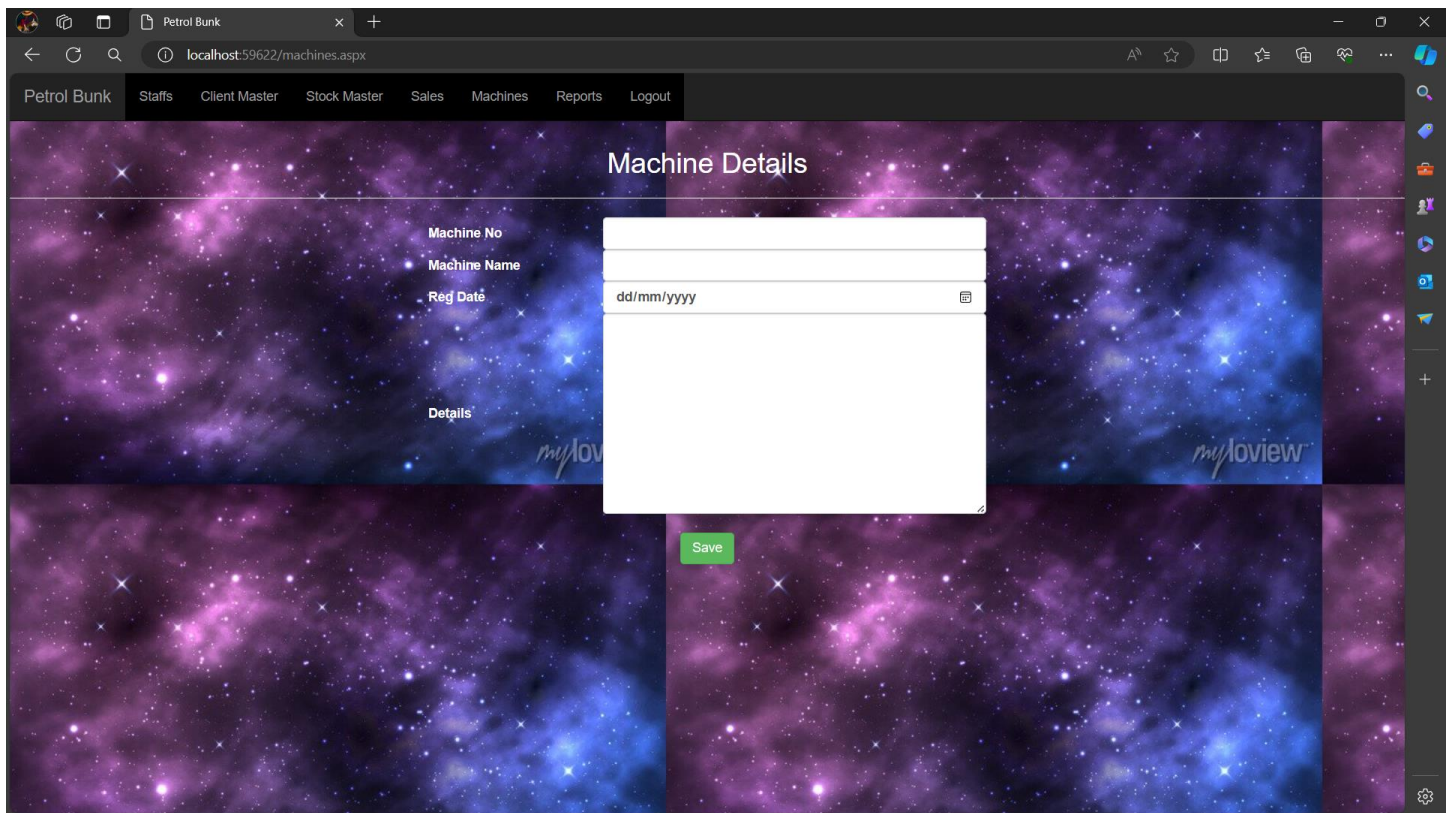
Petrol

Save

myloview

myloview

Machines:



The screenshot shows a web browser window with the address bar displaying 'localhost:59622/machines.aspx'. The browser's address bar and tabs are visible at the top. The application's navigation menu includes 'Petrol Bunk', 'Staffs', 'Client Master', 'Stock Master', 'Sales', 'Machines', 'Reports', and 'Logout'. The 'Machines' menu item is currently selected.

The main content area features a 'Machine Details' form. The form is set against a background image of a galaxy. The form fields are as follows:

- Machine No**: A text input field.
- Machine Name**: A text input field.
- Reg Date**: A date input field with a placeholder 'dd/mm/yyyy' and a calendar icon.
- Details**: A large text area for additional information.

A green 'Save' button is located at the bottom of the form. The background image of the galaxy is visible behind the form and the navigation menu.

Reports:

Petrol Bunk

localhost:59622/reports.aspx

Petrol Bunk

Staffs

Client Master

Stock Master

Sales

Machines

Reports

Logout

Stock Reports

Id	Petrol	Diesel
1	15900	15395

myloview

myloview

Table Structure

Login Table:

dbo.login [Design] | dbo.login [Data] | MasterAdmin.master | clients.aspx | sales.aspx | machines.aspx

Update | Script File: dbo.login.sql

	Name	Data Type	Allow Nulls	Default	
PK	Id	int	<input type="checkbox"/>		
	username	varchar(50)	<input checked="" type="checkbox"/>		
	password	varchar(50)	<input checked="" type="checkbox"/>		
	role	varchar(50)	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Keys (1)

<unnamed> (Primary Key, Clustered: Id)

Check Constraints (0)

Indexes (0)

Foreign Keys (0)

Triggers (0)

Clients Reg:

dbo.client [Design] | dbo.client [Data] | MasterAdmin.master | clients.aspx

Update | Script File: dbo.client.sql

	Name	Data Type	Allow Nulls	Default	
PK	clientid	int	<input type="checkbox"/>		
	clientname	varchar(100)	<input checked="" type="checkbox"/>		
	client_type	varchar(100)	<input checked="" type="checkbox"/>		
	email	varchar(100)	<input checked="" type="checkbox"/>		
	mobile	varchar(50)	<input checked="" type="checkbox"/>		
	address	varchar(250)	<input checked="" type="checkbox"/>		
	regdate	varchar(50)	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Keys (1)

<unnamed> (Primary Key, Clustered: clientid)

Check Constraints (0)

Indexes (0)

Foreign Keys (0)

Triggers (0)

Staffs:

dbo.staffs [Design] dbo.client [Design] dbo.login [Design] dbo.login [Data] MasterAdmin.master					
Update Script File: dbo.staffs.sql					
	Name	Data Type	Allow Nulls	Default	
	Id	int	<input type="checkbox"/>		Keys (1) <unnamed> (Primary Key, Clustered: Id) Check Constraints (0) Indexes (0) Foreign Keys (0) Triggers (0)
	staffname	varchar(50)	<input checked="" type="checkbox"/>		
	qualification	varchar(50)	<input checked="" type="checkbox"/>		
	department	varchar(50)	<input checked="" type="checkbox"/>		
	mobile	varchar(50)	<input checked="" type="checkbox"/>		
	address	varchar(250)	<input checked="" type="checkbox"/>		
	photo	varchar(250)	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		
Design T-SQL					

Stocks:

dbo.stockss [Design] X

dbo.staffs [Design]

dbo.client [Design]

dbo.login [Design]

dbo.login [Data]

Update

Script File:

dbo.stockss.sql

	Name	Data Type	Allow Nulls	Default	
	Id	int	<input type="checkbox"/>		<div>Keys (1)</div> <div><unnamed> (Primary Key, Clustered: Id)</div> <div>Check Constraints (0)</div> <div>Indexes (0)</div> <div>Foreign Keys (0)</div> <div>Triggers (0)</div>
	petrol	int	<input checked="" type="checkbox"/>		
	diesel	int	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Purchase:

dbo.machine [Design] × dbo.stockss [Design] dbo.staffs [Design] dbo.client [Design] dbo.login [Design]

Update Script File: dbo.machine.sql

	Name	Data Type	Allow Nulls	Default	
PK	Id	int	<input type="checkbox"/>		Keys (1) <unnamed> (Primary Key, Clustered: Id) Check Constraints (0) Indexes (0) Foreign Keys (0) Triggers (0)
	machineno	varchar(50)	<input checked="" type="checkbox"/>		
	name	varchar(100)	<input checked="" type="checkbox"/>		
	regdate	varchar(100)	<input checked="" type="checkbox"/>		
	details	varchar(400)	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Sales:

dbo.sales [Design] × dbo.machine [Design] dbo.stockss [Design] dbo.staffs [Design] dbo.client [Design]

Update Script File: dbo.sales.sql

	Name	Data Type	Allow Nulls	Default	
PK	salesid	int	<input type="checkbox"/>		Keys (1) <unnamed> (Primary Key, Clustered: salesid) Check Constraints (0) Indexes (0) Foreign Keys (0) Triggers (0)
	salesdate	date	<input checked="" type="checkbox"/>		
	staffname	varchar(50)	<input checked="" type="checkbox"/>		
	petrol	int	<input checked="" type="checkbox"/>		
	diesel	int	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Source Code

Index Page :

Intex.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="index.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Petrol Bunk Management System</title>
    <link rel="stylesheet" href="Concern/bootstrap.min.css"/>

    <script type="text/javascript" src="js/jquery-
2.2.2.min.js"></script>
    <script type="text/javascript"
src="js/bootstrap.min.js"></script>
    <script type="text/javascript" src="js/main.js"></script>
</head>
<body style="background-image:url(images/b2.jpg);color:brown;">
<div class="container">
<div class="row">

    <nav class="navbar navbar-inverse" role="navigation">
        <div class="container">
            <!-- Brand and toggle get grouped for better mobile
display -->
            <div class="navbar-header">

                <a class="navbar-brand" href="#">Petrol Bunk</a>
                <img src="" class="navbar-image"/>
            </div>
            <!-- Collect the nav links, forms, and other content for
toggling -->
            <div class="collapse navbar-collapse" id="bs-example-
navbar-collapse-1">
                <ul class="nav navbar-nav">
                    <ul>
                </div>
            <!-- /.navbar-collapse -->
        </div>
```



```

        <!-- /.container -->
    </nav>

</div>
</div>

<div class="container">
<div class="row">

        <h2 align="center" style="color:white;">Petrol Bunk
Management System</h2>
        <div class="table-responsive" align="center">
            <form id="form1" runat="server">
                <table class="table table-bordered"
style="width:250px;color:#fff;" align="center">

                    <form class="col-md-12" action="#"
method="post">

                        <tr><td colspan="2"><h2
align="center">Login Area</h2></td></tr>
                        <tr><td>UserName</td>
                            <td>
                                <asp:TextBox ID="txtusername"
class="form-control input-md" runat="server"></asp:TextBox>
                            </td></tr>

                        <tr><td>Password</td>
                            <td> <asp:TextBox
ID="txtpassword" TextMode="Password" class="form-control input-md"
runat="server"></asp:TextBox>
                            </td></tr>
                        <tr><td colspan="2">

                            <asp:Button ID="Button1" class="btn
btn-primary btn-md btn-block"
runat="server" Text="Sign In"
onclick="Button1_Click" />

                        </td>

```

```

        </tr>

        </form>
    </table>
</form>
</div>
</div>
<div class="col-md-2"><div class="row">

</div></div>

```

```

</div>
</div>
</body>
</html>

```

Master Admin Form:

```

<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterAdmin.master.cs" Inherits="MasterAdmin" %>

```

```

<!DOCTYPE html>

```

```

<html xmlns="http://www.w3.org/1999/xhtml">

```

```

    <head>
        <title>Petrol Bunk</title>
        <link rel="stylesheet" href="css/bootstrap.min.css"/>
        <script type="text/javascript" src="js/jquery-
2.2.2.min.js"></script>
        <script type="text/javascript"
src="js/bootstrap.min.js"></script>
        <script type="text/javascript" src="js/main.js"></script>

```

```

    <asp:ContentPlaceholder id="head" runat="server">

```

```

</asp:ContentPlaceholder>
</head>

```

```

<body style="background-image:url(images/b2.jpg);">
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container-fluid">
        <div class="navbar-header">
            <a href="#" class="navbar-brand">Petrol Bunk</a>

```

```

        </div>
        <ul class="nav navbar-nav">

            <li style='background-color:#000'><a
href='staffs.aspx'>Staffs</a></li>

            <li style='background-color:#000'><a
href='clients.aspx'>Client Master</a></li>
            <li style='background-color:#000'><a
href='stockmaster.aspx'>Stock Master</a></li>
            <li style='background-color:#000'><a
href='sales.aspx'>Sales</a></li>
            <li style='background-color:#000'><a
href='machines.aspx'>Machines</a></li>

            <li style='background-color:#000'><a
href='reports.aspx'>Reports</a></li>
            <li style='background-color:#000'><a
href='index.aspx'>Logout</a></li>
        </ul>
    </div>
</div>
<p><br /></p><p><br /></p>

<div class="container-fluid">
    <div class="row">
        <div class="col-md-2"></div>

        <form id="form1" runat="server">
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">

</asp:ContentPlaceHolder>
        </form>

    </div>
</div>

</body>
</html>

```

Clients Form Design:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterAdmin.master"
AutoEventWireup="true" CodeFile="clients.aspx.cs" Inherits="clients"
%>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">

</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2"
Runat="Server">
    <div align="center">
        <center>

            <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False" CssClass="table table-hover"
CellPadding="4" DataKeyNames="clientid" DataSourceID="SqlDataSource1"
ForeColor="#333333" GridLines="None">
                <AlternatingRowStyle BackColor="White"
ForeColor="#284775" />
                <Columns>
                    <asp:BoundField DataField="clientid"
HeaderText="clientid" ReadOnly="True" SortExpression="clientid" />
                    <asp:BoundField DataField="clientname"
HeaderText="clientname" SortExpression="clientname" />
                    <asp:BoundField DataField="client_type"
HeaderText="client_type" SortExpression="client_type" />
                    <asp:BoundField DataField="email" HeaderText="email"
SortExpression="email" />
                    <asp:BoundField DataField="mobile"
HeaderText="mobile" SortExpression="mobile" />
                    <asp:BoundField DataField="address"
HeaderText="address" SortExpression="address" />
                    <asp:BoundField DataField="regdate"
HeaderText="regdate" SortExpression="regdate" />
                </Columns>
                <EditRowStyle BackColor="#999999" />
                <FooterStyle BackColor="#5D7B9D" Font-Bold="True"
ForeColor="White" />
                <HeaderStyle BackColor="#5D7B9D" Font-Bold="True"
ForeColor="White" />
```

```

        <PagerStyle BackColor="#284775" ForeColor="White"
HorizontalAlign="Center" />
        <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
        <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True"
ForeColor="#333333" />
        <SortedAscendingCellStyle BackColor="#E9E7E2" />
        <SortedAscendingHeaderStyle BackColor="#506C8C" />
        <SortedDescendingCellStyle BackColor="#FFFDF8" />
        <SortedDescendingHeaderStyle BackColor="#6F8DAE" />
    </asp:GridView>

```

```

        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:objcon %>"
SelectCommand="SELECT * FROM [client]"></asp:SqlDataSource>

```

```

    <hr />

```

```

        <asp:Table ID="Table1" runat="server" Width="499px"
style="font-family:'Times New Roman';font-
weight:400;color:yellow;background-color:black;">
            <asp:TableRow>
                <asp:TableCell>
                    <label for="Name">Client Id</label>
                </asp:TableCell>
                <asp:TableCell>
                    <asp:TextBox class="form-control" ID="txtid"
runat="server"></asp:TextBox>
                </asp:TableCell>

                <asp:TableCell>
                    <label for="Name">Client Name</label>
                </asp:TableCell>
                <asp:TableCell>
                    <asp:TextBox class="form-control"
ID="txtclientname" runat="server"></asp:TextBox>
                </asp:TableCell>
            </asp:TableRow>

            <asp:TableRow>
                <asp:TableCell>
                    <label for="Name">Client Type</label>
                </asp:TableCell>
                <asp:TableCell>

```

```

        <asp:DropDownList ID="txttype" runat="server"
ForeColor="#FF3300">
<asp:ListItem>Travels</asp:ListItem>
<asp:ListItem>Transport</asp:ListItem>
<asp:ListItem>Office</asp:ListItem>

        </asp:DropDownList>

    </asp:TableCell>

    <asp:TableCell>
        <label for="Name">Email</label>
    </asp:TableCell>
    <asp:TableCell>
        <asp:TextBox class="form-control"
ID="txtemail" runat="server"></asp:TextBox>
    </asp:TableCell>
</asp:TableRow>

<asp:TableRow>
    <asp:TableCell>
        <label for="Name">Mobile</label>
    </asp:TableCell>
    <asp:TableCell>
        <asp:TextBox class="form-control"
ID="txtmobile" runat="server"></asp:TextBox>
    </asp:TableCell>

    <asp:TableCell>
        <label for="Name">Address</label>
    </asp:TableCell>
    <asp:TableCell>
        <asp:TextBox class="form-control"
ID="txtaddress" runat="server" Rows="5"></asp:TextBox>
    </asp:TableCell>

    <asp:TableCell>
        <label for="Name">Date</label>
    </asp:TableCell>

```

```

        <asp:TableCell>
            <asp:TextBox class="form-control" ID="txtdate"
type="date" runat="server"></asp:TextBox>
        </asp:TableCell>

    </asp:TableRow>
    <asp:TableRow><asp:TableCell><br /></asp:TableCell>
</asp:TableRow>
    <asp:TableRow>
        <asp:TableCell VerticalAlign="Middle">
            <asp:Button ID="Button1" class="btn btn-
success btn-md center-block"
                                runat="server" Text="Save"
onclick="Button1_Click" />

        </asp:TableCell>
        <asp:TableCell>
            <asp:Button ID="Button2" class="btn btn-
success btn-md center-block"
                                runat="server" Text="Update"
onclick="Button2_Click"/>
        </asp:TableCell>

        <asp:TableCell>
            <asp:Button ID="Button3" class="btn btn-
success btn-md center-block"
                                runat="server" Text="Delete"
onclick="Button3_Click"/>
        </asp:TableCell>
        <asp:TableCell>
            <asp:Button ID="Button4" class="btn btn-
success btn-md center-block"
                                runat="server" Text="Show"
onclick="Button4_Click"/>
        </asp:TableCell>
    </asp:TableRow>
</asp:Table>

</center>

```

</div>

</asp:Content>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;

public partial class clients : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        { maxno(); }
    }
    void maxno()
    {
        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();
        string str = "select max(clientid)+1 from client";
        SqlCommand sqlcmd = new SqlCommand(str, con);
        txtid.Text = sqlcmd.ExecuteScalar().ToString();
        con.Close();
    }
    protected void Button1_Click(object sender, EventArgs e)
    {

        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();

        SqlCommand cmd;
```



```
cmd = new SqlCommand("Insert Into client Values('" +  
txtid.Text + "',''" + txtclientname.Text + "',''" + txttype.Text + "',''" +  
+ txtemail.Text + "',''" + txtmobile.Text + "',''" + txtaddress.Text +  
"',''" + txtdate.Text + "')", con);  
cmd.ExecuteNonQuery();
```

```
con.Close();  
Response.Redirect("clients.aspx");
```

```
}  
protected void Button2_Click(object sender, EventArgs e)  
{  
    string sqlstr =  
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;  
    SqlConnection con = new SqlConnection(sqlstr);  
    con.Open();  
    SqlCommand cmd;  
    cmd = new SqlCommand("update client set clientname='" +  
txtclientname.Text + "','client_type='" + txttype.Text + "','email='" +  
txtemail.Text + "','mobile='" + txtmobile.Text + "','address='" +  
txtaddress.Text + "' where clientid='" + txtid.Text + "'", con);  
    cmd.ExecuteNonQuery();  
    con.Close();  
    Page.RegisterStartupScript("Script Description", "<script  
type=\"text/javascript\">alert('Updated Success');</script>");  
    Response.Redirect("clients.aspx");  
}
```

```
protected void Button3_Click(object sender, EventArgs e)  
{  
    string sqlstr =  
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;  
    SqlConnection con = new SqlConnection(sqlstr);  
    con.Open();  
    SqlCommand cmd;  
    cmd = new SqlCommand("delete from client where clientid='" +  
txtid.Text + "'", con);  
    cmd.ExecuteNonQuery();  
    con.Close();  
    Page.RegisterStartupScript("Script Description", "<script  
type=\"text/javascript\">alert('Staff Record Deleted');</script>");  
    Response.Redirect("staffs.aspx");  
}
```

```

    }

    protected void Button4_Click(object sender, EventArgs e)
    {
        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();
        SqlCommand cmd = new SqlCommand("select * from client where
clientid='" + txtid.Text + "'", con);

        SqlDataReader sqldr = cmd.ExecuteReader();
        sqldr.Read();
        if (sqldr.HasRows)
        {
            txtclientname.Text = sqldr[1].ToString();
            txttype.SelectedValue = sqldr[2].ToString();
            txtemail.Text = sqldr[3].ToString();
            txtmobile.Text = sqldr[4].ToString();
            txtaddress.Text = sqldr[5].ToString();

            sqldr.Close();
        }
        con.Close();
    }
}

```

```

<%@ Page Title="" Language="C#" MasterPageFile="~/MasterAdmin.master"
AutoEventWireup="true" CodeFile="machines.aspx.cs" Inherits="machines"
%>

```

```

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2"
Runat="Server">
    <div align="center">
        <center>
            <h2 style="color:white;">Machine Details</h2>
            <hr />
            <table CssClass="table table-hover" width="40%"
style="font-weight:bold;color:white;">
                <tr>
                    <td>Machine No</td>

```

```

        <td><asp:TextBox class="form-control" ID="txtno"
type="text" runat="server"></asp:TextBox></td>
    </tr>
    <tr>
        <td>Machine Name</td>
        <td><asp:TextBox class="form-control" ID="txtname"
type="text" runat="server"></asp:TextBox></td>
    </tr>
    <tr>
        <td>Reg Date</td>
        <td><asp:TextBox class="form-control" ID="txtdate"
type="date" runat="server"></asp:TextBox></td>
    </tr>
    <tr>
        <td>Details</td>
        <td><asp:TextBox class="form-control"
ID="txtdetails" type="text" runat="server" TextMode="MultiLine"
Rows="10"></asp:TextBox></td>
    </tr>
    <tr><td> <br /></td></tr>
    <tr>
        <td colspan="2"> <asp:Button ID="Button1"
class="btn btn-success btn-md center-block"
runat="server" Text="Save"
onclick="Button1_Click" /></td>
    </tr>
</table>
</center>
</div>
</asp:Content>

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;

```

```

public partial class machines : System.Web.UI.Page

```

```

{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {

        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();

        SqlCommand cmd;

        cmd = new SqlCommand("Insert Into machine Values('" +
txtno.Text + "',''" + txtname.Text + "',''" + txtdate.Text + "',''" +
txtdetails.Text + "')", con);
        cmd.ExecuteNonQuery();

        Page.RegisterStartupScript("Script Description", "<script
type=\"text/javascript\">alert('Machine Details Added
Success');</script>");

        con.Close();
        Response.Redirect("machines.aspx");

    }
}

```

Stock Master

```

<%@ Page Title="" Language="C#" MasterPageFile="~/MasterAdmin.master"
AutoEventWireup="true" CodeFile="stockmaster.aspx.cs"
Inherits="stockmaster" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>

```

```

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2"
Runat="Server">
    <div align="center">

        <center>
            <hr />

            <asp:GridView ID="GridView1" runat="server"
CssClass="table" style="color:white;" AutoGenerateColumns="False"
DataKeyNames="Id" DataSourceID="SqlDataSource1">
                <Columns>
                    <asp:BoundField DataField="Id" HeaderText="Id"
ReadOnly="True" SortExpression="Id" />
                    <asp:BoundField DataField="petrol"
HeaderText="petrol" SortExpression="petrol" />
                    <asp:BoundField DataField="diesel"
HeaderText="diesel" SortExpression="diesel" />
                </Columns>
            </asp:GridView>

            <hr />
            <h2 style="color:white;">Stock Master</h2>
            <hr />

            <hr />
            <table style="color:white;">
                <tr><td>Invoice Id</td>
                    <td colspan="3"> <asp:TextBox class="form-
control" ID="txtid" runat="server"></asp:TextBox></td>
                </tr>
                <tr>
                    <td>Purchase Date</td>
                    <td> <asp:TextBox class="form-control"
ID="txtdate" type="date" runat="server"></asp:TextBox></td>
                    <td>Supplier Name</td>
                    <td> <asp:TextBox class="form-control"
ID="txtsupplier" type="text" runat="server"></asp:TextBox></td>
                </tr>
                <tr>
                    <td>Diesel</td>
                    <td> <asp:TextBox class="form-control"
ID="txtdiesel" type="text" runat="server"></asp:TextBox></td>

```

```

        <td>Petrol</td>
        <td> <asp:TextBox class="form-control"
ID="txtpetrol" type="text" runat="server"></asp:TextBox></td>

    </tr>
    <tr>
        <td>Staff Incharge</td>
        <td> <asp:DropDownList ID="txtstaff"
runat="server" DataTextField="staffname" DataValueField="staffname"
DataSourceID="SqlDataSource2"></asp:DropDownList>
        <asp:SqlDataSource ID="SqlDataSource2"
runat="server" ConnectionString="<%$ ConnectionStrings:objcon %>"
SelectCommand="SELECT DISTINCT [staffname] FROM
[staffs]"></asp:SqlDataSource>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:objcon %>"
SelectCommand="SELECT * FROM [stockss]"></asp:SqlDataSource></td>
        <td>Vehicle No</td>
        <td> <asp:TextBox class="form-control"
ID="txtvehicle" type="text" runat="server"></asp:TextBox></td>

    </tr>
    <tr>
        <td colspan="4"> <asp:Button ID="Button1"
class="btn btn-success btn-md center-block"
runat="server" Text="Save"
onclick="Button1_Click" /></td>

    </tr>
</table>

```

```

    </center>
</div>
</asp:Content>

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;

```

```

public partial class stockmaster : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        maxno();
    }
    void maxno()
    {
        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();
        string str = "select max(pid)+1 from purchase";
        SqlCommand sqlcmd = new SqlCommand(str, con);
        txtid.Text = sqlcmd.ExecuteScalar().ToString();
        con.Close();
    }

    protected void Button1_Click(object sender, EventArgs e)
    {

        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();

        SqlCommand cmd;

        cmd = new SqlCommand("Insert Into purchase Values('" +
txtid.Text + "',' ' + txtstaff.Text + "',' ' + txtdiesel.Text + "',' ' +
txtpetrol.Text + "',' ' + txtdate.Text + "',' ' + txtstaff.Text + "',' ' +
txtvehicle.Text + ''))", con);
        cmd.ExecuteNonQuery();

        SqlCommand cmd1;

        int p = int.Parse(txtpetrol.Text);
        int d = int.Parse(txtdiesel.Text);

        int id = 1;

```

```

        cmd1 = new SqlCommand("update stockss set petrol= petrol + '"
+ p + "',diesel=diesel + '" + d + "' where id='" +id + "'", con);
        cmd1.ExecuteNonQuery();

        Page.RegisterStartupScript("Script Description", "<script
type=\"text/javascript\">alert('Purchase info Updated
Success');</script>");

        con.Close();
        Response.Redirect("stockmaster.aspx");

    }
}

```

Sales Form Design

```

<%@ Page Title="" Language="C#" MasterPageFile="~/MasterAdmin.master"
AutoEventWireup="true" CodeFile="sales.aspx.cs" Inherits="sales" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2"
Runat="Server">
    <div align="center">
        <center>
            <h1 style="color:white;"> Sales</h1>
            <hr />
            <table style="color:white;font-weight:500;">
                <tr><td>Invoice Id</td>
                    <td colspan="3"> <asp:TextBox class="form-
control" ID="txtid" runat="server"></asp:TextBox></td>
                </tr>
                <tr>
                    <td>Sales Date</td>
                    <td> <asp:TextBox class="form-control"
ID="txtdate" type="date" runat="server"></asp:TextBox></td>
                    <td>Staff Name</td>
                    <td> <asp:DropDownList ID="txtstaff"
runat="server" DataSourceID="SqlDataSource1" DataTextField="staffname"
DataValueField="staffname"></asp:DropDownList>
                    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:objcon %>"

```



```

SelectCommand="SELECT DISTINCT [staffname] FROM
[staffs]"></asp:SqlDataSource></td>

        </tr>
        <tr>
            <td>Diesel</td>
            <td> <asp:TextBox class="form-control"
ID="txtdiesel" type="text" runat="server"></asp:TextBox></td>
            <td>Petrol</td>
            <td> <asp:TextBox class="form-control"
ID="txtpetrol" type="text" runat="server"></asp:TextBox></td>

        </tr>

        <tr>
            <td colspan="4"> <asp:Button ID="Button1"
class="btn btn-success btn-md center-block"
runat="server" Text="Save"
onclick="Button1_Click" /></td>

        </tr>
    </table>
</center>
</div>
</asp:Content>

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;

```

```

public partial class sales : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        maxno();
    }
    void maxno()
    {

```

```

        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();
        string str = "select max(salesid)+1 from sales";
        SqlCommand sqlcmd = new SqlCommand(str, con);
        txtid.Text = sqlcmd.ExecuteScalar().ToString();
        con.Close();
    }
    protected void Button1_Click(object sender, EventArgs e)
    {

        string sqlstr =
ConfigurationManager.ConnectionStrings["objcon"].ConnectionString;
        SqlConnection con = new SqlConnection(sqlstr);
        con.Open();

        SqlCommand cmd;

        cmd = new SqlCommand("Insert Into sales Values('" +
txtdate.Text + "',' ' + txtstaff.Text + "',' ' + txtdiesel.Text + "',' ' +
txtpetrol.Text + ')", con);
        cmd.ExecuteNonQuery();

        SqlCommand cmd1;

        int p = 0;
        int d = 0;

        p = int.Parse(txtpetrol.Text);

        d = int.Parse(txtdiesel.Text);

        int id = 1;
        cmd1 = new SqlCommand("update stockss set petrol= petrol - '"
+ p + "',' diesel=diesel - '" + d + "' where id='" + id + "'", con);
        cmd1.ExecuteNonQuery();

        Page.RegisterStartupScript("Script Description", "<script
type=\"text/javascript\">alert('sales info Updated
Success');</script>");
    }

```

```
        con.Close();
        Response.Redirect("sales.aspx");
    }
}
```

Reports

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterAdmin.master"
AutoEventWireup="true" CodeFile="reports.aspx.cs" Inherits="reports"
%>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2"
Runat="Server">
    <div align="center">
        <center>

            <hr />
            <h1 style="color:white;">Stock Reports</h1>
            <hr />
            <table width="500px" align="center" CssClass="table
table-hover" cellpadding="2" cellspacing="2" border="1"
style="background-color:#000;color:yellow;">
                <tr align="center" >

                    <td align="center" > Id </td>
                    <td align="center" >Petrol</td>
                    <td align="center" >Diesel</td>
                </tr>

                <%=getWhileLoopData()%>

            </table>

        </center>
    </div>
</asp:Content>
```

Conclusion

Conclusion

All the objectives that had been charted out in the initial phases were achieved successfully.

System Features:

System satisfies all the requirements for which the company developed the system. System has strong security. System is fully GUI based. It is easy operate and user friendly. Platform includes the inbuilt backup and recovery facility.

Through working this project I understand the importance of planning and designing of Online Petrol Bunk Management System development.

The petrol bunk management system project emphasizes the automation and efficiency it brings to petrol bunks. It showcases the benefits of automation in petrol bunks, offering a modern and efficient solution to manage fuel distribution effectively.