**BUS MANAGEMENT**

**PROJECT Report**

***Submitted by***

**ISHITA JOSHI RA1911028010096**

**PRASOON MISHRA RA1911028010107**

**SANKALP MISHRA RA1911028010110**

***in partial fulfillment for the award of the degree***

***of***

**B.TECH**

***in***

COMPUTER SCIENCE and ENGINEERING

IN

CLOUD COMPUTING

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

KATTANKULATHUR

**April, 2022**

**ABSTRACT**

The Main aim of Bus Management Mini DBMS project is to ease the process of ticket booking and keeping records of busses and the drivers or the conductors. We aim to demonstrate the use of create, read, update and delete MySQL operations through this project. This project starts by adding stations then adding details of employee either a driver or a conductor. Once, registration is done user can add bus details with the route. Now, the user can use the ticket counter to book tickets for the passengers.

**Modules:**

Bus Management Mini DBMS Project contains 4 modules:

1. Bus details: adding or updating new buses and routes.
2. Stations: adding new stations.
3. Employee: Employee details either driver or conductor.
4. Ticket Counter: ticket booking for the passengers.

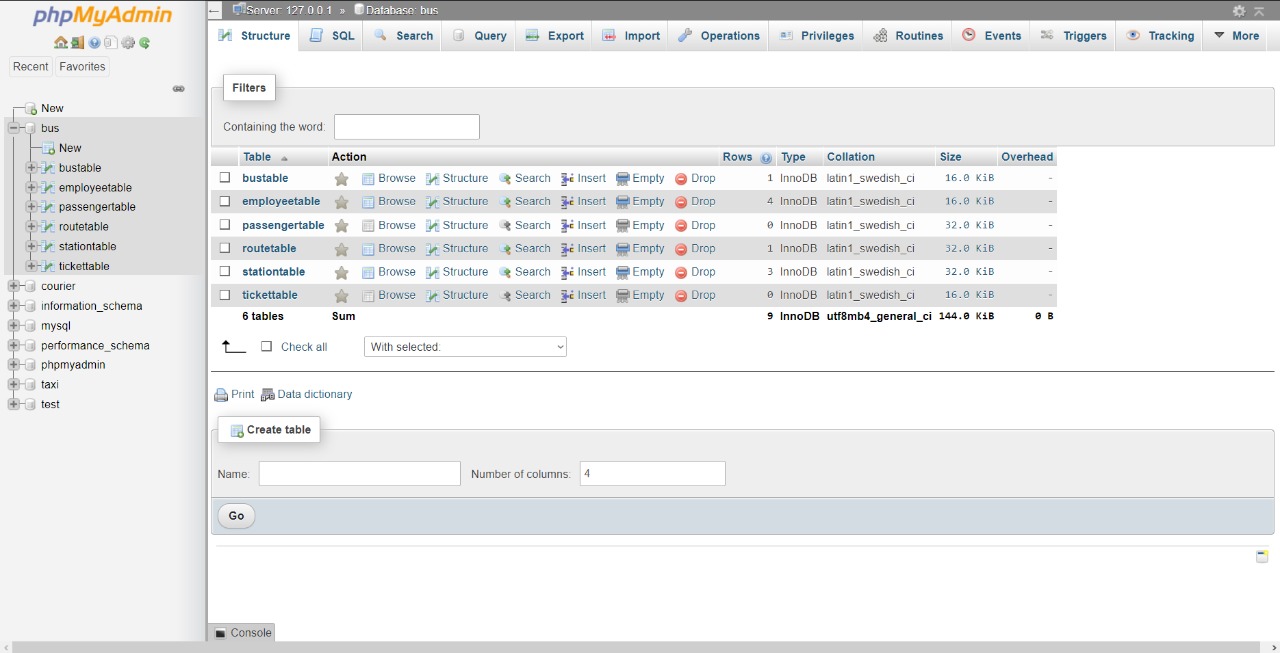
**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | **2** |
|  | **LIST OF TABLES** | **6** |
|  | **LIST OF FIGURES** | **1** |
|  | **LIST OF ABBREVIATIONS** | **10** |
| **1** | **INTRODUCTION** | **4** |
| **2** | **Entity Relationship Diagram** | **1** |
| **3** | **Normalized Database Table** | **7** |
| **4** | **SQL Queries with results** | **6** |
| **5** | **Conclusion and Future Work** | **1** |
|  | **References** |  |

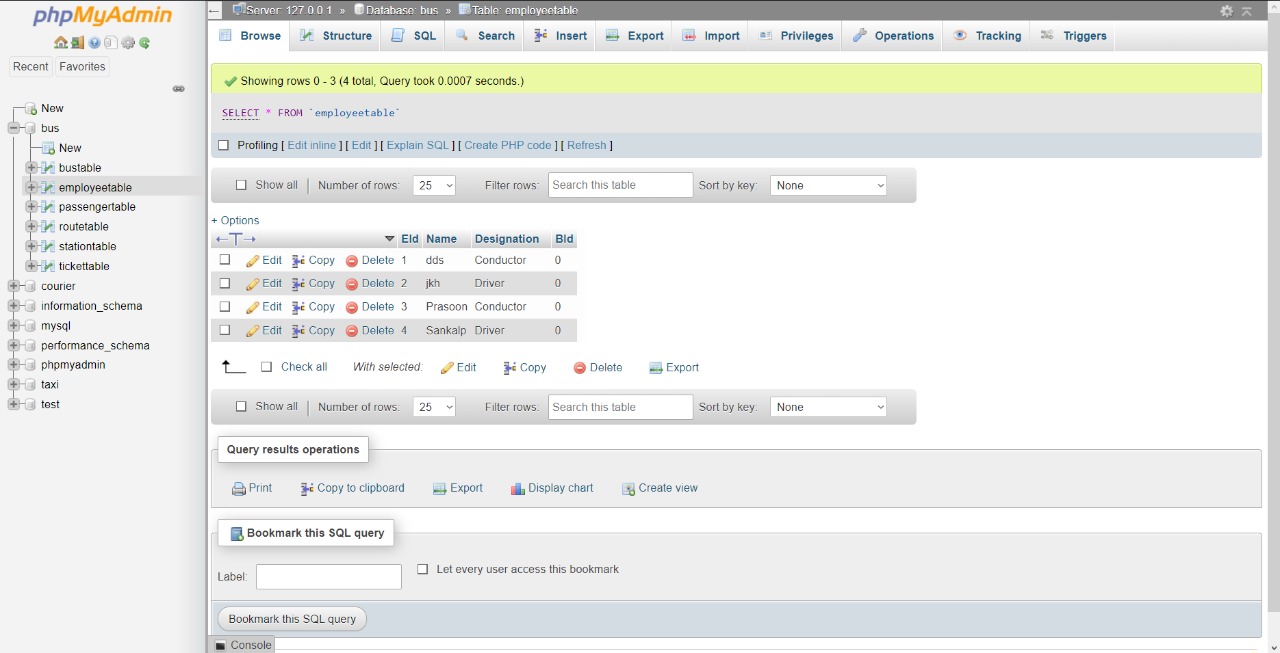
**LIST OF TABLES**

|  |  |
| --- | --- |
| **TABLE NO.** | **TABLE NAME** |
| **1** | **Bus table** |
| **2** | **Employee table** |
| **3** | **Passenger table** |
| **4** | **Route table** |
| **5** | **Station Table** |
| **6** | **Ticket Table** |

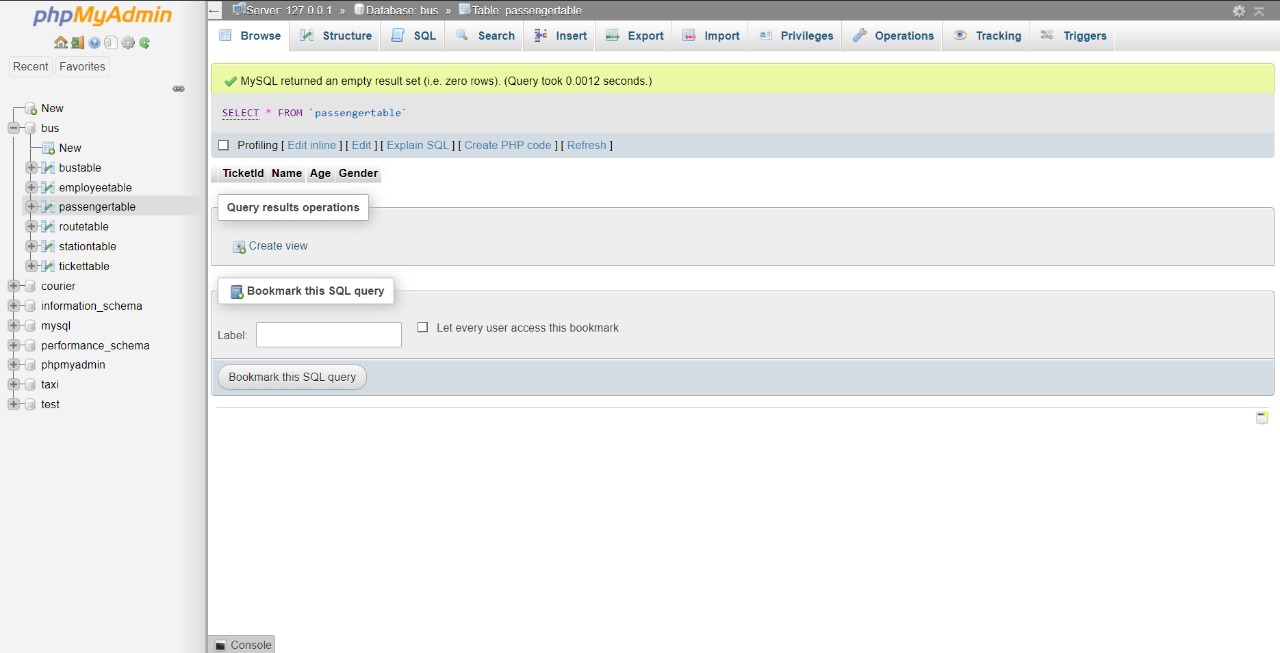
**1)Bus Table**

****

**2)Employee Table**

****

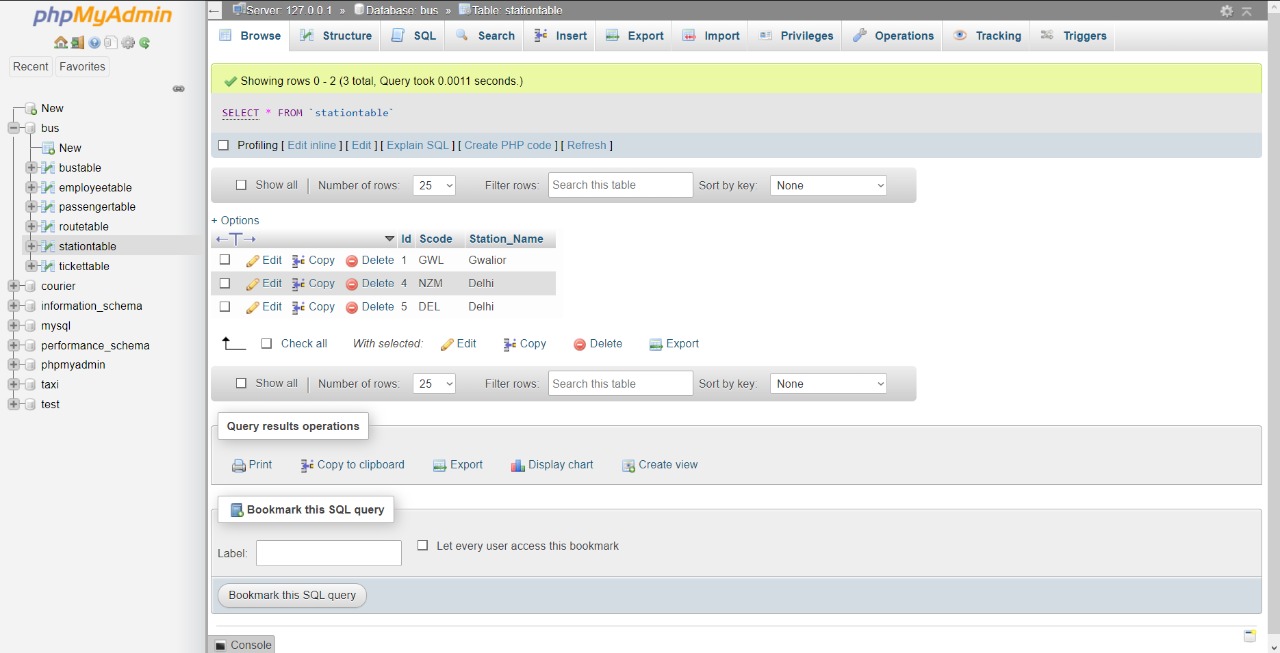
**3)Passenger Table**

****

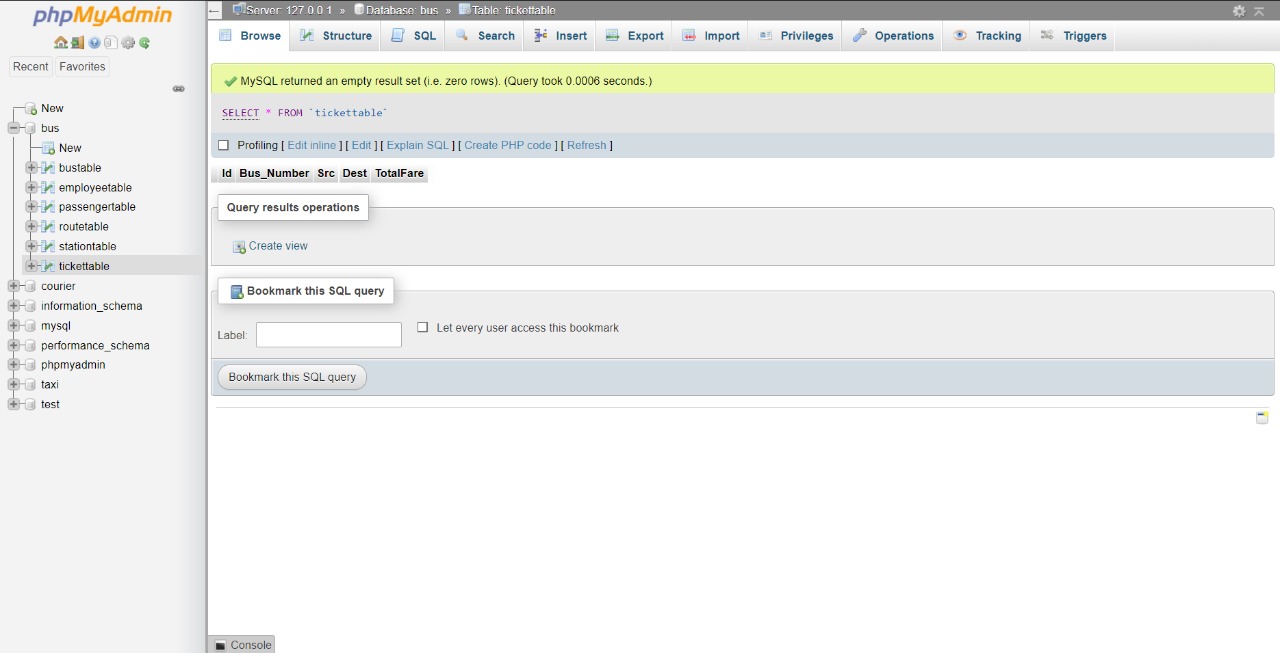
**4)Route TableGraphical user interface, text, application, email

Description automatically generated**

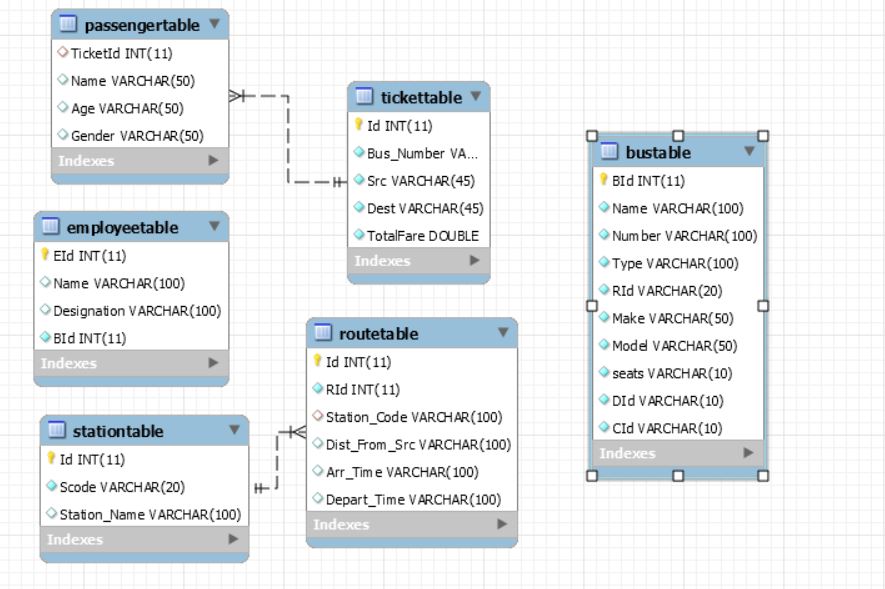
**5)Station Table**

****

**6)Ticket Table**

****

**LIST OF FIGURES**

****

Graphical user interface

Description automatically generated

**Introduction**

The systems objectives outlined during the feasibility study serve as the basis from which the work of system design is initiated. Much of the activities involved at this stage is of technical nature requiring a certain degree of experience in designing systems, sound knowledge of computer related technology and thorough understanding of computers available in the market and the various facilities provided by the vendors. Nevertheless, a system cannot be designed in isolation without the active involvement of the user. The user has a vital role to play at this stage too. As we know that data collected during feasibility study will be utilized systematically during the system design. It should, however, be kept in mind that detailed study of the existing system is not necessarily over with the completion of the feasibility study. Depending on the plan of feasibility study, the level of detailed study will vary and the system design stage will also vary in the amount of investigation that still needs to be done. This investigation is generally an urgent activity during the system design as the designer needs to study minute’s details in all aspects of the system. Sometimes, but rarely, this investigation may form a separate stage between Feasibility Study and Computer System Design. Designing a new system is a creative process, which calls for logical as well as lateral thinking. The logical approach involves systematic moves towards the end product keeping in mind the capabilities of the personnel and the equipment at each decision making step. Lateral thought implies encompassing of ideas beyond the usual functions and equipment. This is to ensure that no efforts are being made to fit previous solutions into new situations.

**System Design Considerations:**

The system design process is not a step-by-step adherence of clear procedures and guidelines. Though, certain clear procedures and guidelines have emerged in recent days, but still much of design work depends on knowledge and experience of the designer.

When designer starts working on system design, he will face different type of problems. Many of these will be due to constraints imposed by the user or limitations of the hardware and software available in the market. Sometimes, it is difficult to enumerate the complexity of the problems and solutions thereof since the variety of likely problems is so great and no solutions are exactly similar. However, following considerations should be kept in mind during the system-designing phase:

**The primary objective of the design:** Of course, is to deliver the requirements as specified in the feasibility report. In general, the following design objectives should be kept in mind:

* **Practicality**: The system must be stable and can be operated by people with average
* **Efficiency**: This involves accuracy, timeliness and comprehensiveness of the system
* output.
* **Cost**: it is desirable to aim for a system with a minimum cost subject to the condition that it must satisfy all the requirements.
* **Flexibility**: The system should be modifiable depending on the changing needs of the user. Such modifications should not entail extensive reconstructing or recreation of software. It should also be portable to different computer systems.
* **Security**: This is very important aspect of the design and should cover areas of hardware reliability, fall back procedures, physical security of data and provision for detection of fraud and abuse.

System design involves first logical design and then physical construction of the system. The logical design describes the structure and characteristics of features, like the outputs, inputs, files, databases and procedures. The physical construction, which follows the logical design, produces actual program software, files and a working system.

The designer normally will work under following constraints:

* **Hardware**: The existing hardware will obviously affect the system design.
* **Software**: The available software (operating system, utilities, language etc.) in the market will constrain the design.
* **Budget**: The budget allocated for the project will affect the scope and depth of design.
* **Time-scale**: The new system may be required by a particular time (e.g. the start of a financial year). This may put a constraint on the designer to find the best design.
* **Interface with other systems**: The new system may require some data from another computerized system or may provide data to another system in which case the files must be compatible in format and the system must operate with a certain processing cycle.

**Processing Techniques:**

The processing options available to the designer are:

* Batch processing
* Real-time processing
* On-line processing
* A combination of all the above

You are already aware of these techniques. It is quite interesting to note, however, that a combination of these is often found to be ideal in traditional data processing applications. This increases throughput of the system as also brings down the response time of on-line activities. In most of die business applications, 24-hour data is acceptable enough and hence it is possible to update voluminous data after office-hours in batch mode.

**DESIGN METHODOLOGIES**

The scope of the systems design is guided by the framework for the new system developed during analysis. More clearly defined logical method for developing system that meets user requirements has led to new techniques and methodologies that fundamentally attempt to do the following:

* Improve productivity of analysts and programmers
* Improve documentation and subsequent maintenance and enhancements.
* Cut down drastically on cost overruns and delays
* Improve communication among the user, analyst, designer, and programmer.
* Standardize the approach to analysis and design
* Simplify design by segmentation.

**STRUCTURED DESIGN**

Structured design is a data flow based methodology. The approach begins with a system specification that identifies inputs and outputs and describes the functional aspects of the system. The specifications then are used as a basis for the graphic representation. The next step is the definition of the modules and their relationships to one another in a form called a structure chart, using a data dictionary and other structured tools.

Logical design proceeds from the top down. General features, such as reports and inputs are identified first. Then each is studied individually and in more detail. Hence, the structured design partitions a program into small, independent modules. They are arranged in a hierarchy that approximates a model of the business area and is organized in a top-down manner. Thus, structured design is an attempt to minimize the complexity and make a problem manageable by subdividing it into smaller segments, which is called Modularization or decomposition. In this way, structuring minimizes intuitive reasoning and promotes maintainable provable systems.

A design is said to be top-down if it consists of a hierarchy of modules, with each module having a single entry and a single exit subroutine. The primary advantages of this design are as follows:

* Critical interfaces are tested first.
* Early versions of the design, though incomplete, are useful enough to resemble the real system.
* Structuring the design, perse, provides control and improves morale.
* The procedural characteristics define the order that determines processing.

**Major System Design Activities:**

Several development activities are carried out during structured design. They are data base design, implementation planning, system test preparation, system interface specification, and user documentation.

* **Data base design:** This activity deals with the design of the physical database. A key is to determine how the access paths art to be implemented.
* **Program design:** In conjunction with database design is a decision on the programming language to be used and the flowcharting, coding, and debugging procedure prior to conversion. The operating system limits the programming languages that will run of the system.
* **System and program test preparation:** Each aspect of the system has a separate test requirement. System testing is done after all programming and testing completed the test cases cover every aspect of the proposed system, actual operations, user interface and so on. System and program test requirements become a part of design specifications - a pre requisite to implementation.

**MODULE COUPLING**

Coupling is the measure of the degree of interdependence between modules. Two modules with high coupling are strongly interconnected and thus, dependent on each other. Two modules with low coupling are not dependent on one another. “Loosely coupled” systems are made up of modules which are relatively independent. “Highly coupled” systems share a great deal of dependence between modules.

**DATA COUPLING**

The dependency between module A and B is said to be coupled it their dependency is based on the fact they communicate by only passing of data.

**STAMP COUPLING**

Stamp coupling occurs between module A and B when complete data structure is passed from one module to another.

**CONTROL COUPLING**

Module A and B are said to be control coupled if they communicate by passing of control information.

**EXTERNAL COUPLING**

A form of coupling in which has a dependency to other module, external to the software being developed or to a particular type of hardware.

**COMMON COUPLING**

With common coupling, module A and B have shared data. Global data areas are commonly found in programming languages. Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of change.

**CONTENT COUPLING**

Content coupling occurs when module A changes data of module B or when control is passed from one to the middle of another.

**MODULE COHESION**

Cohesion is a measure of the degree to which the elements of a module are functionally related. A strongly cohesive module implements functionality that is related to one feature of the solution and requires little or no interaction with other modules.

**FUNCTIONAL COHESION**

X and Y are part of a single functional task. This is very good reason for them to be contained in the same procedure.

**SEQUENTIAL COHESION**

X output some data which forms the input to Y. This is the reason for them to be contained in the same procedure.

**COMMUNICATIONAL COHESION**

X and Y both operate on the same input data or contribute towards the same output data. This is okay, but we might consider making them separate procedures.

**PROCEDURAL COHESION**

X and Y are both structured in the same way. This is a poor reason for putting them in the same procedure. Thus, procedural cohesion occurs in modules whose instructions although accomplish different tasks yet have been combined because there is a specific order in which the tasks are to be completed.

**TEMPORAL COHESION**

X and Y both must perform around the same time. So, module exhibits temporal cohesion when it contains tasks that are related by the fact that all tasks must be executed in the same time-span.

**LOGICAL COHESION**

X and Y perform logically similar operations. Therefore, logical cohesion occurs in modules that contain instructions that appear to be related because they fall into the same logical class.

**COINCIDENTAL COHESION**

X and Y here no conceptual relationship other than shared code Hence, coincidental cohesion exists in modules that contain instructions that have little or no relationship to one another.

**STRATEGY OF DESIGN**

A good system design strategy is to organize the program modules in such a way that are easy to develop and later to, change. Structured design techniques help developers to deal with the size and complexity of programs. Analysts create instructions for the developers about how code should be written and how pieces of code should fit together to form a program.

**Bottom Up Design**

These approach lead to a design where we decide how to combine these modules to provide larger ones; to combine those to provide a larger ones, and so on, till we arrive at one big module which is the whole of the desired program. The set of these modules form a hierarchy. This is a cross-linked tree structure in which each module is subordinate to those in which it is used.

Since the design progressed from bottom layer upwards, the method is called bottom-up design. This method has one terrible weakness; we need to use a lot of intuition to design exactly what functionality a module should provide. If we get it wrong, then at higher level, we will find that it is not as per requirements; then we have to redesign at a lower level.

**Top- Down Design**

A top design approach starts by identifying the major modules of the system, decomposing them into lower level and iterating until the desired level of detail is achieved. This is a stepwise refinement; starting from an abstract design, in each step the design is refined to a more concrete level, until we reach a level where no refinement is needed and the design can be implemented directly. Most design methodologies are based on this approach is suitable, if the specifications are clear and development is from the scratch.

**Hybrid Design**

Hybrid approach has really become popular after the acceptance of reusability of modules. Standard libraries, Microsoft foundation classes, object oriented concepts are steps in this direction. We may soon have internationally acceptable standards for reusability.

# SYSTEM TESTING AND IMPLEMENTATION

Software testing is the process of executing a program with the intention of finding errors in the code. It is the process of exercising or evaluating a system or system component by manual or by automatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

The objective of testing is to show incorrectness and testing is considered to succeed when an error is detected. An error is a conceptual mistake made by either the programmer or the designer or a discrepancy between a computed value and a theoretically correct value. A fault is a specific manifestation of an error. An error may be cause of several faults. A failure is the inability of a system or component to perform its required function within the specified limits. A failure may be produced when a fault is executed or exercised.

Other activities that are often associated with software are static analysis and dynamic analysis. Static analysis investigates the source code of software, looking for problems and gathering metrics without actually executing the code. Dynamic analysis looks at the behavior of software while it is executing, to provide information such as execution traces, timing profiles and test coverage information.

**7.1 Levels of testing**

**7.1.1 Unit Testing or Module Testing**

The starting point of testing is Unit testing. In this, a module is tested separately at each step. This helps to detect syntax and logical errors in the program and is performed by the coder himself /herself during coding.

**7.1.2 Integration Testing**

The modules, which are tested in the Unit Testing, are integrated to build the overall system. It is observed that many errors crop up when the modules are joined together. Integration testing uncovers these errors while integrating the modules. It helps in establishing confidence (correctness) in the complete, assembled system. It tests the System Design. It focus on control, communication, interfaces, performance (other system qualities). It make use of stubs, test-beds, data generators. It is the phase of software testing in which individual software modules are combined and tested as a group. It follows unit testing and precedes system testing.

Integration testing takes as its input [modules](http://en.wikipedia.org/wiki/Module_%28programming%29) that have been [unit tested](http://en.wikipedia.org/wiki/Unit_testing), groups them in larger aggregates, applies tests defined in an integration [test plan](http://en.wikipedia.org/wiki/Test_plan) to those aggregates, and delivers as its output the integrated system ready for [system testing](http://en.wikipedia.org/wiki/System_testing).

Integration testing concentrates entirely on module interactions, assuming that the details within each module are accurate. Module and Integration testing can be combined, verifying the details of each module's implementation in an integration context. Many projects compromise, combining module testing with the lowest level of subsystem integration testing, and then performing pure integration testing at higher levels. Each of these views of integration testing may be appropriate for any given project, so an integration testing method should be flexible enough to accommodate them all.

The System testing is bringing together of all programs that a system comprises for testing purposes. System testing is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic. Programs are typically integrated in a top-down, incremental fashion. It is a series of different tests whose primary purpose is to fully exercise the computer-based system. It includes the following tests: -

* **Recovery Testing: -** It is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.
* **Stress Testing:-** These are designed to confront program functions with abnormal situations. It executes a system in a manner that demands resources in abnormal quantity, frequency or volume.
* **Security Testing:-** This testing attempts to verify that protection mechanism built into a system will protect it from unauthorized penetration.

The system testing is an investigatory testing phase, where the focus is to have almost a destructive attitude and test not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

**7.2 Types of testing**

**7.2.1 Black Box Testing**

It is also known as Functional Testing. It tests the overall functional requirements of product. Inputs are supplied to product and outputs are verified. If the outputs obtained are the same as the expected ones then the product meets the functional requirements. In this, the internal procedures are not considered. In this the tester would only know the "legal" inputs and what the expected outputs should be, but not how the program actually arrives at those outputs. This Testing is more effective on larger units of code. In this test’s are done from user point of view.

**7.2.2 White Box Testing**

It is also known as Structure Testing. It focuses on the internal functioning of the product. It tests the loops of the Procedure, Decision points, Execution paths etc.

White box testing uses specific knowledge of programming [code](http://www.webopedia.com/TERM/W/code.html) to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. He or she can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission, and all visible code must also be readable. As the knowledge of internal coding structure is prerequisite, it becomes very easy to find out which type of input/data can help in testing the application effectively. The other advantage of white box testing is that it helps in optimizing the code. It helps in removing the extra lines of code, which can bring in hidden defects.

**7.2.3 Acceptance Testing**

This Testing is done when the software is developed for the specific customer. A series of tests are conducted to enable the customer to validate all requirements. The end user/ customer conducts these tests and may range from adhoc test to well-planned systematic series of tests. Acceptance testing may be conducted for few weeks or months. The discovered errors will be fixed and better quality software will be delivered to the customer.

Acceptance testing is performed by the [customer](http://en.wikipedia.org/wiki/Customer) on a [system](http://en.wikipedia.org/wiki/System) prior to the customer accepting delivery or accepting transfer of ownership of that system.

The customer specifies scenarios to test when a user story has been correctly implemented. A story can have one or many acceptance tests, what ever it takes to ensure the functionality works. Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. Acceptance tests are also used as regression tests prior to a production release. A user story is not considered complete until it has passed its acceptance tests. This means that new acceptance tests must be created each iteration or the development team will report zero progress.

#### 7.2.4 Alpha Testing

Testing after code is mostly complete or contains most of the functionality and prior to users being involved. Sometimes a select group of users are involved. More often this testing will be performed in-house or by an outside testing firm in close cooperation with the software engineering department.

In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

#### 7.2.5 Beta 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.

**IMPLEMENTATION**

The application can be uploaded in the AIRLINE MANAGEMENT SYSTEM. To access it, the user will just require running the executable file of the software. System must have Turbo c++ driver. Basically the application is for the recording of the Star Sport’s records. As implementation of AIRLINE MANAGEMENT System software fully automate the existing system. In the designed system implementation was done to replace a manual system with the computerized one. The objective was to put the tested system in to operation. Critical aspects of conversion are not disrupting the functioning of the organization. This phase gives us the clears pictures of our new system and all the points that have been carefully looked in when designing the computerized system.

Sincere efforts were taken for the implementation of the following goals.

* Maximizing the output reliability
* Maximizing the source test readability
* Minimizing the development time.

**Conclusion**

Finally, we have a system where a user who books the bus seats giving the details and doing the necessary payment option. He gets a token with all detail he boards on the bus at a given time. The conductor who is staff will check the ticket and punch the confirmation. The owner checks the bills in the end. The user drops at the place of destination. Hence completes the task of the system.