ITCS 5102 - SURVEY OF PROGRAMMING LANGUAGES

PATIENT ADMISSION ANALYSIS IN HOSPITALS

FINAL REPORT

ABSTRACT

Hospitals apart from taking care of patients and providing the necessary healing to them, will also maintain a database with the details of all of their patients. This has become quite common these days that one can understand clearly how necessary it is, as it gives a detailed and significant report of the patients from the data that is already available in the system.

Hospitals usually maintain a record of all their patients. Right from the patient Id, their name, the problem for which they have been admitted and further such details. The system that we have designed keeps track of all the patients who are admitted in the hospital. This application tells us if the patients are newly admitted or are returning patients. If in case they are new patients, a clear set of all of their details is stored immediately in the hospital database. If they are returning patients, they would have been provided with a patient Id on their first visit, once they show or tell this Id, all of the patient details could be immediately recovered from the database.

Though this might look trivial and very simple, this basic activity of retrieving the details of patients of the hospital will prove very useful at critical times. On retrieval of such basic information, the hospital staff can clearly get a detailed note of what exact problem that the patient have had previously, when was he admitted, how long did it take to for him to completely recover, what was the medication that he was prescribed, which doctor was assigned to him. And many such information of that sort. All of these details will be available by just a button click, which is the ultimate goal or aim of the details being stored in a database.

Similarly once a new patient arrives, all the details of the patient is being collected and stored in the hospital database. So that, if at all he returns after a year back or likewise, we need not have to get all of their basic information all again and do a detailed diagnosis, but could just start with the knowledge of the ailment that the patient had been suffering from previously and thereby giving the doctors an idea of the previous health history of the patient.

Our System would also suggest regular checkup alerts to the patients whose medical records require a further consultation. The system designed by us would analyze the medical data and patient records and forecast those patients that are likely to seek readmission within a few months of discharge.

INTRODUCTION

Most of the hospitals which are small in size will have a manual data noting system. For example it could be in the form of a register or a notebook wherein the patients details are noted, which could include their name, phone number, address, the date on which they had visited the clinic, the reason for them to visit, the doctor who had diagnosed the patients.

It is not always possible to note these details manually, and in the case of huge organizations, which has a chain of hospitals under the same name, we need to maintain a database which could store the bulk details of thousands and thousands of patients who visit these hospitals every day.

Our project mainly deals with the second kind of data storage mechanism, where we create a database to store the details of the new patients who visit the clinic and also the data of the old patients can be retrieved if they are to visit again.

EXPERIMENTAL PROCESS

We have used F# to develop the application for the Patient Admission Analysis in Hospitals. F# is a strongly-typed, functional-first programming language for writing simple code to solve complex problems. And from a business perspective, the primary role of F# will be to reduce the time-to-deploy for the analytical software components in the modern enterprise.

The F# code is consistently shorter, easier to read, easier to refactor and contains far fewer bugs. F# is just focused on a different "way" of programming, whereas C# and VB-Net is focused on the same way. F# is vastly superior. You can use F# as an object oriented language, just like C# or VB. On top of that it has features like currying, pattern matching, algebraic data types, type inference, tuples, computation expressions, tail call optimization. F# is a functional language like Erlang or Lisp.

One of the major advantage of our project is that it will save a lot of man power, and also the data will not the tampered, and the loss of data will be reduced to null. A lot of time will be saved during the whole process by saving the data on the database.

TECHNICAL DOCUMENTATION

We chose F# as our programming language because it considerably increases speed of software development and execution which is very important in any organization when the development resources are limited. The most enjoyable fact about using this language is that the developer can reason about the code instead of relying only on unit tests. This functional-First programming language helps in creating simple code to solve complex scenarios. This language is very reliable and it encourages Reason Driven Development methodology which leads to virtually bug-free code. F# runs on Windows, MacOS and Linux.

The benefits of the F# language in a nutshell can be explained as follows:

- **Quick Coding** - F#'s powerful type inference means less coding.
- **Agile Coding** - Type-inferred code is easily refactored,
- **Scripting** - Hands-on exploration, (Readability of code)
- **Performance** - Immediate scaling to massive sets of data,

- **Memory-Faithful** - Mega-data structures on 16GB machines,
- **Succinctness** - Live in the domain, not the language,
- **Symbolic** - Schema compilation and “Schedules”

F# programs tend to be much shorter than their equivalents in other languages. With fewer lines of code we get higher productivity.

The productivity and the efficiency of this language can be illustrated by a simple comparison between C# and F# for a “SUM” Functionality.

Example → Comparing F# with C#: A simple sum

Problem Definition: Attempt to sum the squares from 1 to N without using a loop

To see what some real F# code looks like, let's start with a simple problem: "sum the squares from 1 to N".

We'll compare an F# implementation with a C# implementation. First, the F# code:

```
let square x = x * x      // define the square function
let sumOfSquares n = [1..n] |> List.map square |> List.sum    // define the sumOfSquares function
sumOfSquares 100
```

Explanation:

|>- The pipe operator. It just pipes the output of one expression into the input of the next. So the code for sumOfSquares reads as:

- ✓ **Step1:** Create a list of 1 to n (square brackets construct a list).
- ✓ **Step2:** Pipe the list into the library function called List.map, transforming the input list into an output list using the "square" function we just defined.
- ✓ **Step3:** Pipe the resulting list of squares into the library function called List.sum.
- ✓ **Step4:** There is no explicit "return" statement. The output of List.sum is the overall result of the function.

Next, here's a C# implementation using the classic (non-functional) style of a C-based language.

```
public static class SumOfSquaresHelper
{
  public static int Square(int i)
  {
    return i * i;
  }

  public static int SumOfSquares(int n)
  {
```

```
    int sum = 0;
    for (int i = 1; i <= n; i++)
    {
      sum += Square(i);
    }
    return sum;
  }
}
```

By just comparing the code of F# and C# virtually, we can infer that:

- The F# code is more compact
- The F# code didn't have any type declarations
- F# saves the time for implementation.

F# has an interactive window where you can test the code immediately and play around with it. This Reason Driven Development methodology with the interactive development environment makes the code bug-free.

RESULTS AND DISCUSSION ON F#

The following are the results and the favorable features that we inferred from the project that we developed on F#.

Advantages:

- F# doesn't allow NULL's, instead the options have to be used. This feature guarantees safety and serves a similar purpose of representing missing objects like NULL's in C#.
- F# supports for Algebraic Data Types and describes business entities more accurately in terms of data structures reducing the chance of misinterpretation of the data and resulting in higher quality and efficiency of code.
- F# has an advanced type inference system that deduces types of values from how these values are used, hence it is not necessary to mention the types. This results in typing less and results in higher productivity and better maintainability of the code. Because of this F# syntax appears more elegant and easier to read.
- F# eliminates the problem of parallel computation completely by disallowing objects from being modified after creation, whenever a change is needed a new object is constructed by applying a transformation on the original object. This means that the code written in F# can naturally be run in parallel with very little additional efforts.
- The basic unit of logic in F# is a function, Functions in F# can be passed as parameters to other functions and be returned as a result of a function. Writing functions takes less

effort than writing classes. This enables a finer level of modularity and more sophisticated composition scenarios at lower cost. The code written this way is more maintainable and easy to refactor.

- F# has a unique mechanism for working with heterogeneous sources of data in a convenient unified way using type providers. Type providers abstract the implementation details of various sources of data, ensure the type safety and expose standard well-defined interfaces. They also feature on-demand type discovery when new types are loaded only when needed. There is a repository of type providers to various sources of data, which is constantly updated and contributed to by the community. Type providers enable information-rich programming by making it easier to consume the data from different sources.

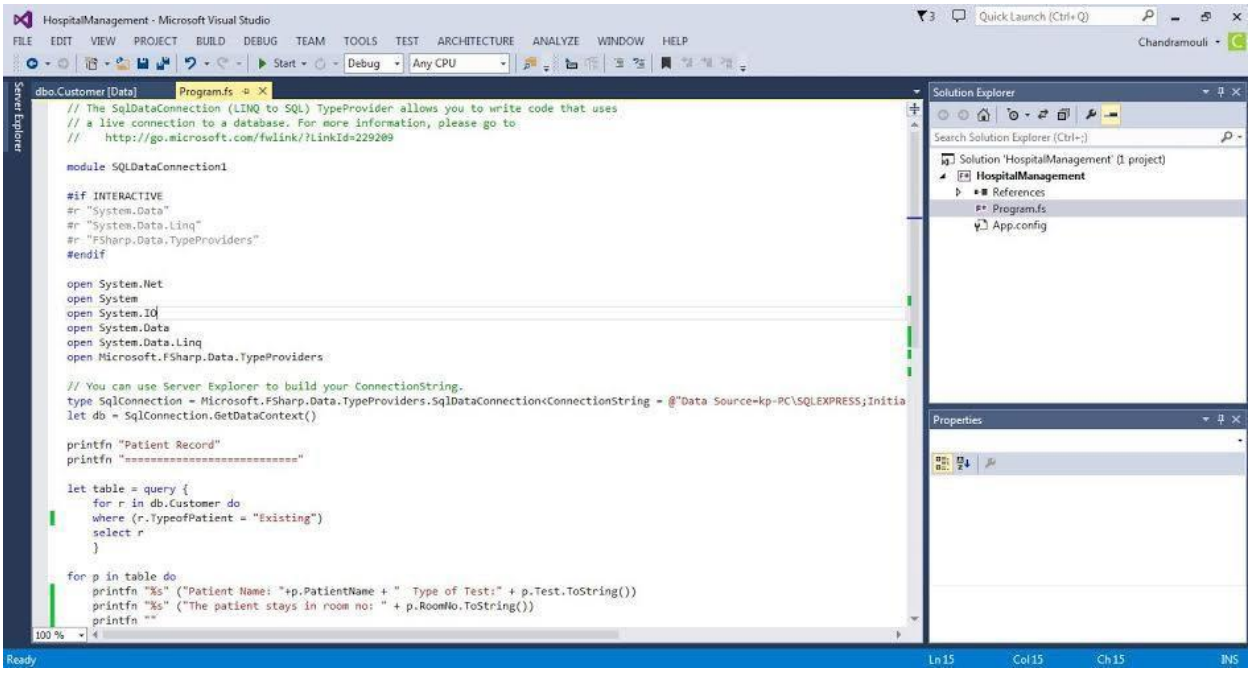
Disadvantages:

- For a person who hasn't had any experience in functional programming it can be challenging to adjust to a new way of thinking while coding in F#. The less popularity and training services is considered to be a big negative point as far as F# is concerned.
- The transformation-over-mutation approach used in this language, uses a more advanced data-structures to do efficient manipulations on the code. For example one has to use a binary-tree in F# instead of a hash-table as they would do in C#. This makes the implementation of data structures difficult.
- Following the principle of using the transformation-over-mutation results in creating more objects that need to be disposed then and there. This puts a bigger load on the garbage collector. Developers must come up with a balanced approach and use this principle effectively.
- F# doesn't have a function overloading feature. So two functions in F# that are in the same module cannot have the same name. This creates a difficulty in naming them uniquely. Coming up with a consistent naming convention for different functions is a challenging task in F#.

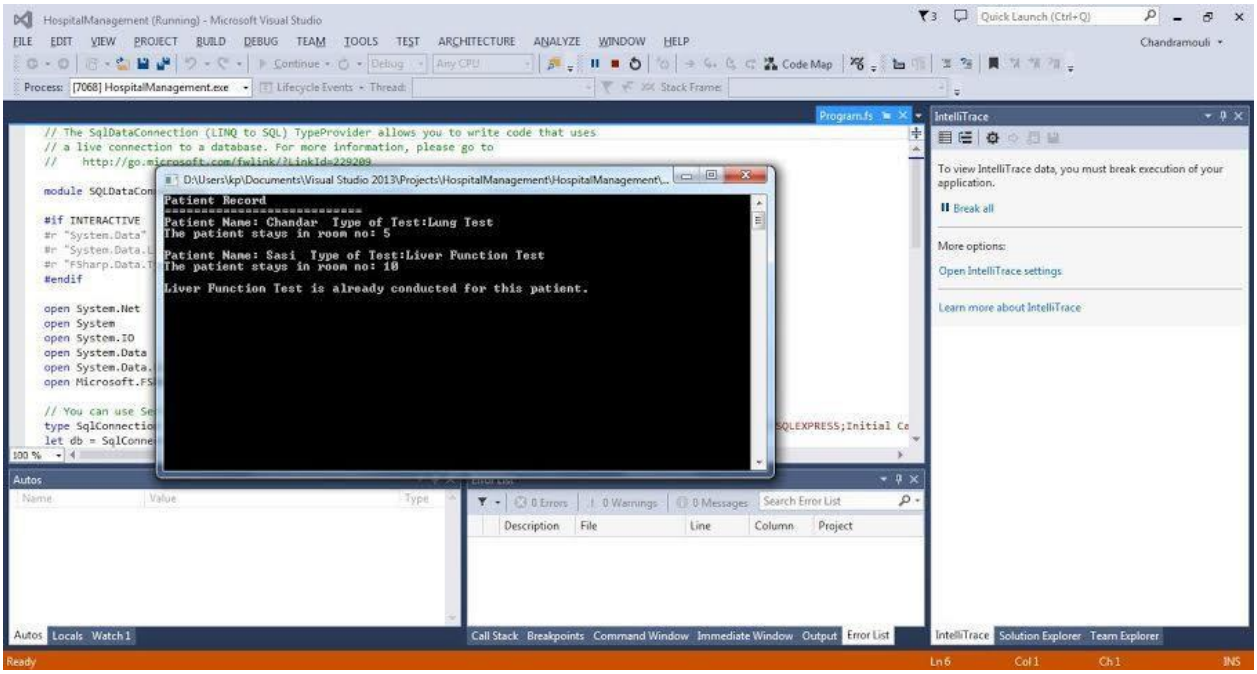
Summarizing our experience with language

It was very convenient to use F#, as functions are first class objects, it is very easy to create powerful and reusable code by creating functions that have other functions as parameters, or that combine existing functions to create new functionality. We were always able to write correct code, because it is caught at compile time as a type error. Moreover F# also has a message queuing system, and excellent support for event handling and reactive programming. And because data structures are immutable by default, sharing state and avoiding locks is much easier.

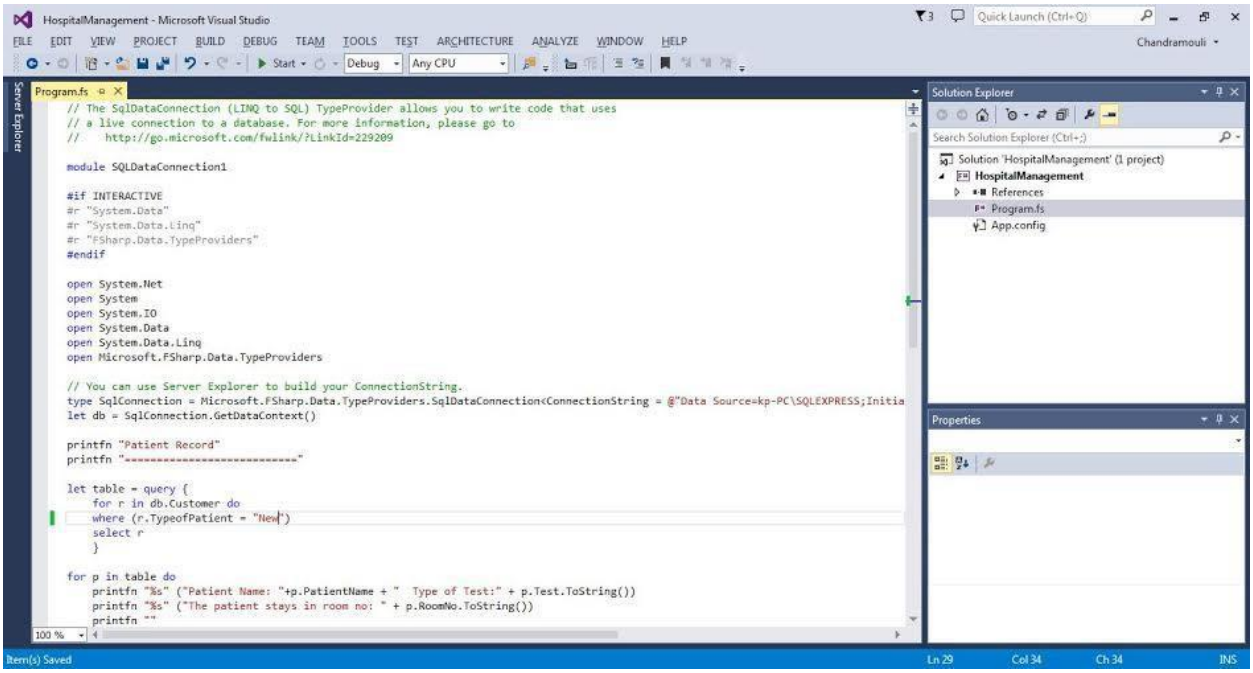
OUTPUT:



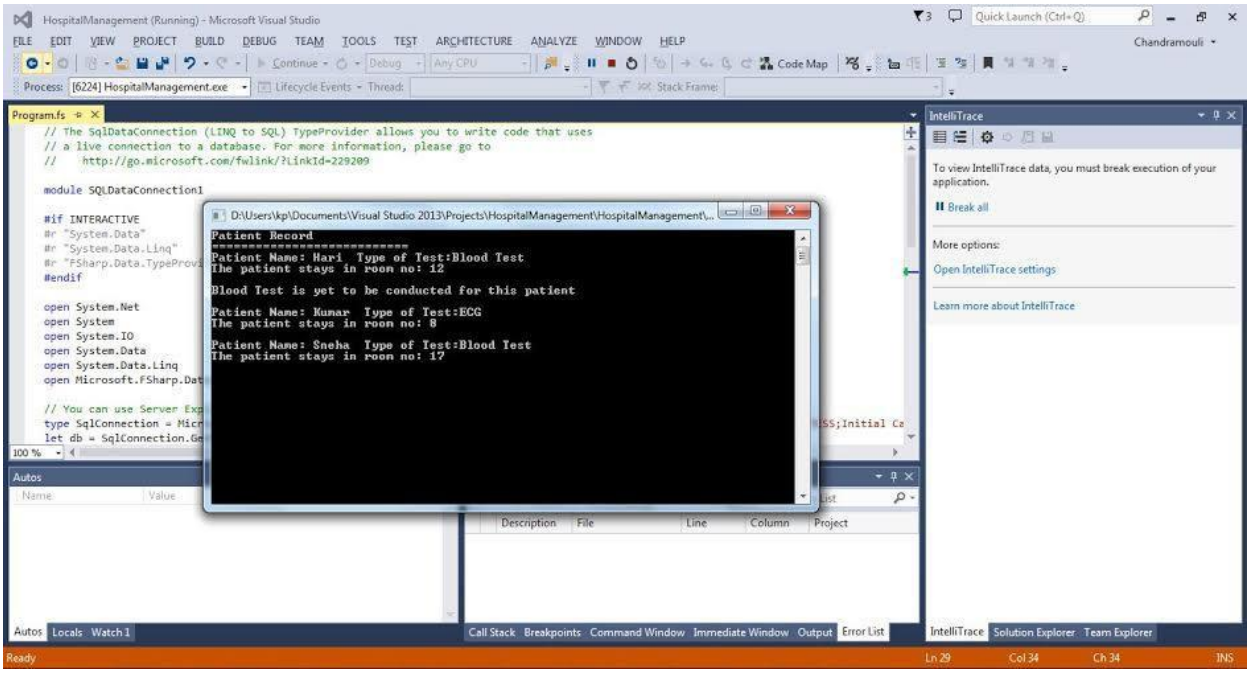
In this screenshot, we give the type of patient as “existing”. This line of code helps us to extract the existing patients from the database.



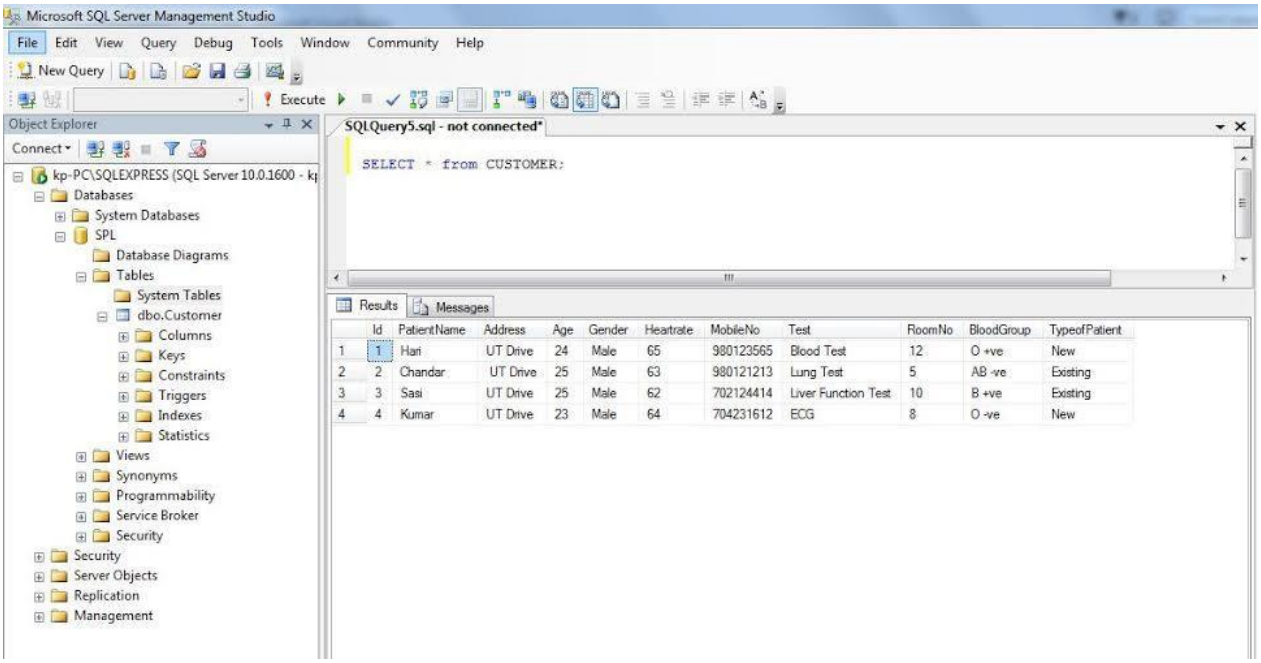
This screenshot shows us the output screen which displays the existing patients from the database.



In this screenshot, we give the type of patient as “new”. This line of code helps us to extract the existing patients from the database.



This screenshot shows us the output screen which displays the new patients from the database.



	Id	PatientName	Address	Age	Gender	Heartrate	MobileNo	Test	RoomNo	BloodGroup	TypeofPatient
1	1	Hari	UT Drive	24	Male	65	980123565	Blood Test	12	O +ve	New
2	2	Chandar	UT Drive	25	Male	63	980121213	Lung Test	5	AB -ve	Existing
3	3	Sasi	UT Drive	25	Male	62	702124414	Liver Function Test	10	B +ve	Existing
4	4	Kumar	UT Drive	23	Male	64	704231612	ECG	8	O -ve	New

The above screenshot shows the customer table which displays details of the new and existing patients.

References

- **http://msdn.microsoft.com/en-us/library/dd233249.aspx**
- **http://tomasp.net/articles/fsharp-i-introduction/article.pdf**