ISSN: 2473-4901
v12 n6761

# Enabling Foundational Learning Of Database Applications Design And Development

Anand Jeyaraj
anand.jeyaraj@wright.edu
Wright State University
Dayton, OH, 45435, USA

## Abstract

Database-driven information systems have become fundamental to business operations in modern organizations. Therefore, the design and development of database applications is a necessary skill for information systems professionals. This paper describes an instructional framework consisting of five stages—eliciting, scoping, designing, developing, and testing—to help students acquire necessary skills for designing and developing database applications for business operations. Further, it illustrates effective ways to facilitate student engagement and learning as they navigate the various stages of database applications design and development. Students reported effective learning experience through the instructional framework. The instructional approach may be implemented in the latter half of introductory programming courses or intermediate/ advanced programming courses for IS students.

**Keywords:** Teaching, Database applications, Applications design, Applications development

# Enabling Foundational Learning Of Database Applications Design And Development

*Anand Jeyaraj*

## 1. INTRODUCTION

Modern organizations continue to rely on various information systems (IS) to enable their everyday operations. This is particularly true of database-driven transactional IS to handle data underlying critical business processes such as sales, production, and procurement. Since IS applications are largely reliant on persistent databases to store and manipulate data, it is ideal for students to gain proficiency in the analysis, design, and development of database-driven IS applications as they get trained for their professional careers.

This paper describes an instructional framework to enable students to engage in foundational learning of database-driven applications. Students are shown the process of creating IS from business requirements by moving through multiple stages. Each stage includes specific considerations that facilitate students to progressively learn the fundamentals of systems development. While the approach can be used with any software development environment, it was implemented with Microsoft Visual Basic.NET (VB.NET) for software development and Microsoft Access for the database support.

The instructional approach may be implemented in the latter half of introductory courses, especially if there is only one programming course available in the degree program. Alternatively, it may be incorporated in intermediate and advanced courses if multiple programming courses are possible in the degree program. The instructional approach, illustrated with a simple banking scenario seen in real-world settings, offers a unique opportunity for students to better understand the database-driven applications required in modern computing and business environments.

The remainder of the paper is organized as follows. The next section introduces prior literature on the pedagogical approaches used in teaching IS design and development. The subsequent section describes the instructional approach proposed in this paper. The discussion section offers insights into the usefulness of the instructional approach and potential extensions that could be applied. The conclusion section wraps up the paper.

## 2. LITERATURE

IS education has grappled with various aspects related to IS development. One broad area is concerned with teaching the fundamentals of programming using a high-level language. While the high-level language for the introductory programming course has changed over the years, different languages such as C#, Java, Python, Visual Basic, and C++ are typically found in IS programs (e.g., Smith & Jones 2021). These languages support one or more paradigms such as structured programming, object-oriented programming, and event-driven programming (Ritzhaupt & Zucker 2006) and require different skillsets to be mastered. However, courses on introductory programming typically deal with the programming fundamentals such as data types, constants, variables, expressions, arrays, files, and functions and various statements for assignment, condition checking, and iteration regardless of specific paradigms.

Another perspective concerns learning beyond the programming fundamentals and design IS applications for specific business purposes. Simkin (2004) described an application designed in Visual Basic.Net to prepare income tax returns for citizens while Simkin (2007) documented an application involving a shopping cart for ordering products from retailers. Ritzhaupt & Zucker (2006) presented a series of progressively more challenging programming exercises to develop an application to calculate the future value of an investment using different options such as simple interest, compound interest, and compound interest with annuity payments. Newby & Nguyen (2007) portrayed a trip cost calculator application for a travel company that offers tour packages for different geographic regions to customers. Sharma et al. (2020) mentioned applications such as making payments in a grocery store and toll booth

ISSN: 2473-4901
v12 n6761

payments as examples of real-world applications that can be used for in-class instruction and learning. Menon (2023) described a shopping cart application that students can use to learn the data and computations for the total price of all items on the cart.

Prior literature has also proposed different approaches for teaching system development to students. Paxton (2002) suggested "live programming" in which the necessary application was coded live by the instructor during regular class times. West & Pardieck (2004) argued for a team-based instructional design possibly involving a contractual arrangement in which the team members choose the application to be designed and strive to make it a reality. Owolabi et al. (2013) recommended a "pair programming" instructional strategy in which two students work together on a single computer system to build the required application, which could offer advantages relative to the "solo programming" strategy. Sharma et al. (2020) suggested several strategies such as semi-structured projects, authentic tasks, and additional Internet resources to enable student learning. Wang & Wang (2021) described a no-code/ low-code approach by which to develop business applications. Jiang (2022) reported a "small teaching tactics" approach in which students were expected to make incremental changes to knowledge they previously gained in understanding the capabilities and developing new solutions. Menon (2023) suggested "code demonstrations" in which the instructors develop the solutions that students can emulate in their independent work. Jeyaraj (2024) presented a scaffolding approach in which students can apply piecewise or progressive integration of prior knowledge to build the solutions.

## 3. INSTRUCTIONAL APPROACH

### 3.1 Context

The instructional approach was carried out in an undergraduate course on IS applications development taken by IS students in a large public university in midwestern USA. Students have previously taken an introductory programming course that deals with the fundamentals of VB.NET including the GUI elements (i.e., Form, Label, Textbox, RadioButton, CheckBox, ListBox, ComboBox, MaskedTextBox, DateTimePicker, ListView, NumericUpDown), variables, expressions, operations, arrays, functions, subprocedures, and data files along with the necessary assignment, conditional (IF, SELECT), iterative (FOR, WHILE, DO), and other (CALL, RETURN) statements. The introductory course requires students to build VB.NET applications such as income tax calculator, shopping cart, and data tables (e.g., unique customers) using arrays and data files but not databases. The business scenario used for in-class instruction is as follows: "MyFirstBank offers two types of accounts to customers: Checking and Savings. A customer may open and hold multiple accounts of either type. The bank allows two types of operations on accounts held by customers: Deposit and Withdrawal. A customer may close any account as desired. Develop a VB.NET application for the bank supported by a Microsoft Access database."

### 3.2 Framework

Figure 1 depicts the framework that can be used for teaching students the process of IS applications development. The framework takes students through the activities surrounding business requirements, project scope, system design, system development, and system testing. These stages may resemble typical systems development lifecycle (SDLC); however, there are subtle differences aimed at instruction and learning. For instance, the SDLC does not explicitly include the "scoping" stage, which is particularly relevant for instruction as students grapple with the fundamentals.

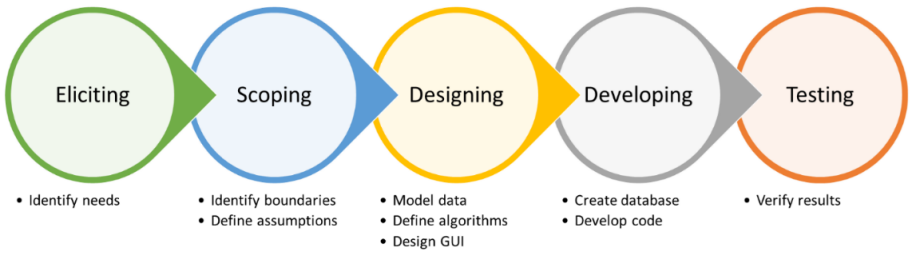


**Figure 1**. Framework for instruction

### 3.3 Eliciting

The eliciting stage deals with the identification of needs and requirements for the IS to enable business operations. Based on the bank scenario, it is possible to elicit the following capabilities for the IS: (a) open new account for a new or returning customer, (b) deposit money into existing account, (c) withdraw money from existing account, and (d) close existing account.

### 3.4 Scoping

The scoping stage represents activities related to identifying the boundaries of the IS and the assumptions underlying the IS.

**Identify boundaries**. Boundaries are crucial to prevent or minimize scope creep in system development since it is possible to keep expanding the set of features. For the bank scenario, it is possible to identify several boundary conditions.

a) Only one bank branch exists; thus, all data is centralized and there is no need to model multiple bank locations.
b) Only on-premise transactions are supported; thus, all transactions will need to completed at the bank location and there is no need to support online transactions.
c) Only the bank employee (i.e., associate or teller) will be the sole user of the IS; other user roles such as manager or customer need not be supported.
d) Only two types of accounts, i.e., checking and savings, are needed; others such as investment or mortgage accounts common in banks need not be included.
e) Only four types of transactions, i.e., open, close, deposit, withdraw, are needed; others such as fees and interest that are commonly seen in banks need not be included.

**Define assumptions**. Assumptions determine the conditions under which the IS may be used or appropriate for business operations. Several assumptions may be defined for the bank scenario to facilitate design of the IS.

a) Each account should be associated with one customer.
b) Deposit and withdrawal transactions possible only on an existing account.
c) Withdrawal transaction possible only when sufficient funds are available in account.
d) Only an existing account can be closed.
e) Customer records cannot be updated to accommodate changes to attributes.
f) Account records cannot be updated except for the account balance due to transactions (i.e., deposit increases account balance while withdrawal decreases account balance).
g) Activity records (i.e., deposit/withdrawal) cannot be updated. Another reverse transaction will have to be completed if the original transaction needs to be canceled.

### 3.5 Designing

The designing stage encompasses activities related to modeling the data and the algorithms (i.e., the sequence of steps) needed to enable business operations. Due to the choice of VB.NET as the development environment, this stage can also include designing the GUI.

**Model data**. Data modeling is aimed at identifying the data enabling operations. For the bank scenario, the necessary data can be organized into three logical groups. First, the bank offers two types of accounts, which gives rise to the ACCOUNT table and its attributes. At a minimum, this table should have the following attributes: Account number (to uniquely identify each account), Account type (to indicate if it is a Checking or Savings account), Account status (to indicate if it is open or closed), Account balance (to maintain the current balance), and Customer number (since each account should be tied to a customer). Second, the customer information needs to be known, the CUSTOMER table becomes a necessity and will need the following attributes: Customer number (to uniquely identify each customer), Customer name, and Phone number (to contact the customer). Several other attributes such as address (street, city, state, zip code), email address, and credit rating (which could be obtained by the bank from external sources) may be relevant, but are excluded for simplicity of system design. Finally, the bank allows transactions on the accounts held by customers. The ACTIVITY table will be needed to record the different transactions completed and will contain at least the following attributes: Transaction number (to uniquely identify the transaction), Transaction date, Transaction type (i.e.,

ISSN: 2473-4901
v12 n6761

deposit or withdrawal; and also open and close if they are to be defined as unique types; otherwise, they will reduce to deposit and withdrawal transactions respectively), Transaction amount (that was deposited to or withdrawn from account), and Account number (to indicate the account on which the transaction was completed).

Further, three additional tables may be designed to enable consistency within the database and the IS. The ACCOUNTTYPES tables is used to record Checking and Savings types of accounts, the ACCOUNTSTATUSES table is used to represent the Open and Closed status for accounts, and the TRANSTYPES table is used to store the Deposit and Withdrawal types of transactions on accounts. Figure 2 depicts the final design of the data model. The "key" icons shown against the attributes indicate the primary key attribute for the respective tables.

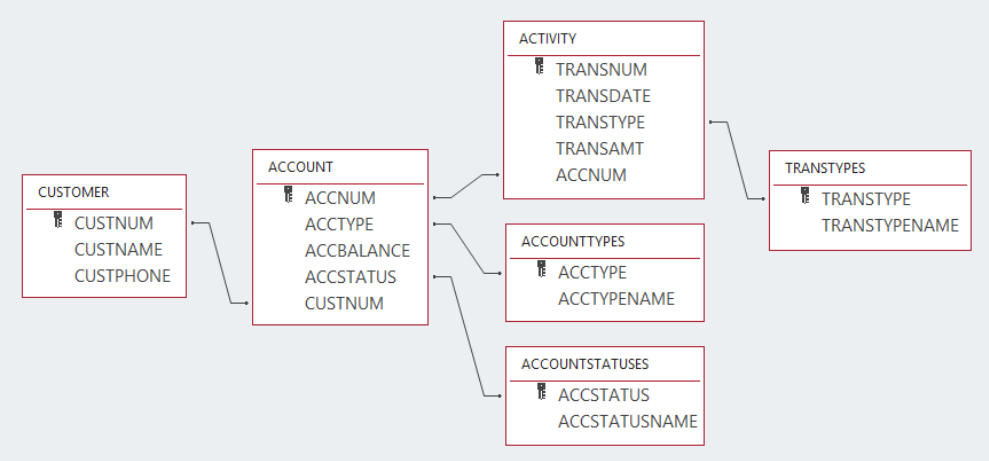


**Figure 2**. Data model with tables and attributes

It is also possible to discuss the relationships between the tables and their cardinalities. For instance, the CUSTOMER and ACCOUNT tables share a 1:M relationship such that each customer record can be associated to multiple account records while each account record should be associated with only one customer record. Similarly, there will be a 1:M relationship between ACCOUNT and ACTIVITY, 1:M relationship between ACCOUNT and ACCOUNTTYPES, 1:M relationship between ACCOUNT and ACCOUNTSTATUSES, and 1:M relationship between ACTIVITY and TRANSTYPES.

**Define algorithms**. An algorithm is a set of logical steps needed to complete the business operations typically described in natural or pseudo language without reference to the high-level computer language to be used for system development. The design should include an algorithm for each operation included in the scope for the IS. For the bank scenario, four separate algorithms, one each for opening account, depositing money, withdrawing money, and closing account. Figure 3 shows the algorithm for creating a new account. Since a new account could be created for: a) a new customer (which requires customer details to be gathered for the CUSTOMER table), or b) a returning customer (which does not require customer details since they are already available on the CUSTOMER table), the algorithm should accommodate both scenarios. Thus, a new record will be added to the CUSTOMER table only if a new customer asks to open an account whereas new records will be added to the ACCOUNT and ACTIVITY tables for both new and returning customers.

Alternatively, it is possible to design two algorithms to accomplish the same task—the first one will be used to create a new customer profile (i.e., gather customer details) and the other one will be used to create a new account (i.e., gather account details)—and require users to activate the algorithms in sequence to complete the operation. The algorithm also relies on the tables and attributes in the data model. For instance, the data for all relevant attributes should be gathered when a new record is to be added to any table whereas a lookup (or search) for a specific record using a key attribute is necessary when a record on any table is to be updated. The algorithm steps can be optimized as shown in the same exhibit.

ISSN: 2473-4901
v12 n6761

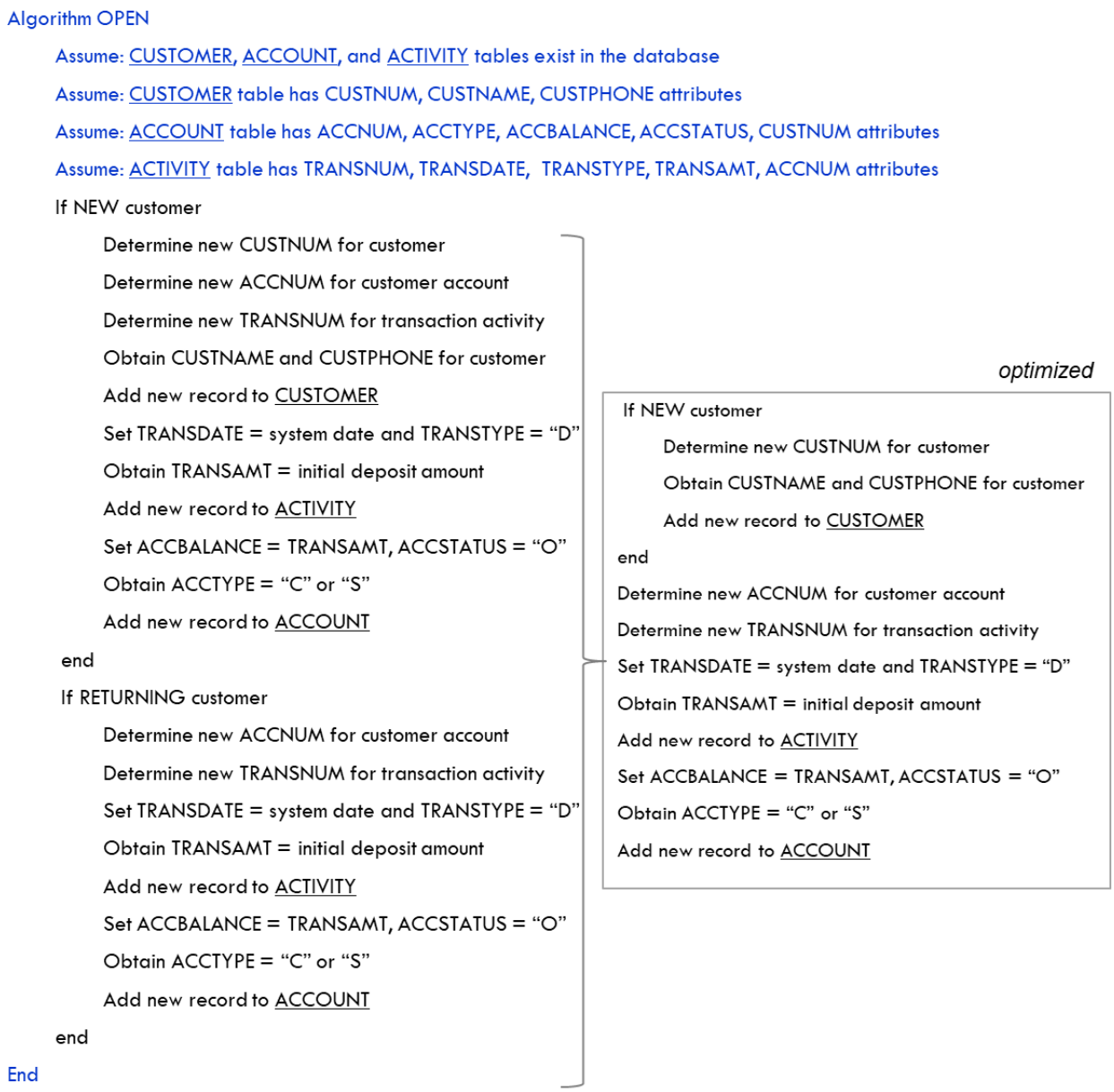


**Figure 3**. Algorithm for opening new bank account

**Design GUI**. The GUI is designed to support the business operations consistent with the data model and the algorithms. To support the four operations in the bank scenario, it is best to design separate GUI screens that allow the user to focus on the specific requirements. Figure 4 depicts the GUI designed for the bank IS. The GUI includes a menu system that allows the user to choose account activities (i.e., open, close) or transaction activities (i.e., deposit, withdrawal). It also includes three ListView elements to display the contents of the CUSTOMER, ACCOUNT, and ACTIVITY tables such that the user can quickly verify if the necessary changes to the tables are correctly handled.

These ListView elements are not mandatory since the user can always directly open the database and view the contents of the tables if necessary, but they are included to enhance the student learning experience.

The TabControl element (with separate TabPage pages) is used to organize the specific requirements for the operations.

ISSN: 2473-4901
v12 n6761

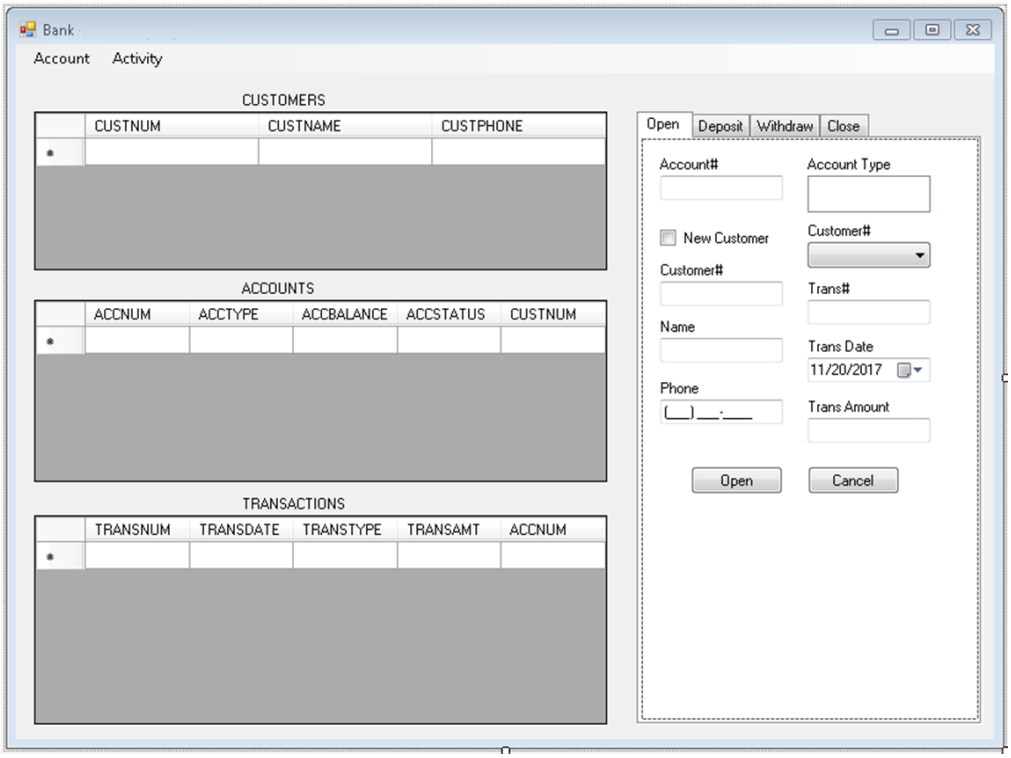


**Figure 4**. VB.NET GUI for opening new bank account

The TabPage for opening a new account contains several elements. The account number is auto-generated and shown in a TextBox. The account type (Checking or Savings) can be selected from the ListBox by the user. The CheckBox can be checked if the account is for a new customer, in which case, customer number (auto-generated, TextBox), customer name (TextBox), and phone number (MaskedTextBox) are obtained from the user. If the CheckBox is unchecked, an existing customer can be selected from the ComboBox by the user. The transaction number (TextBox) is auto-generated and the transaction date (DateTimePicker) is the system date. The TextBox can be used to obtain the initial deposit amount (also the transaction amount). The user can finalize by clicking the Open button or abandon the operation using the Cancel button.

### 3.6 Developing

The developing stage includes activities surrounding the creation of the database and the development of the code for the IS.

**Create database**. The database is created in Microsoft Access using the native capabilities to specify tables and the attributes with the necessary data types and the primary key attributes for the tables. **Figure 5** shows the schema with the table names, attribute names, data types, and field sizes for the bank database.

| Name | Type | Size |
|---|---|---|
| **CUSTOMER** | | |
| CUSTNUM | Long Integer | 4 |
| CUSTNAME | Short Text | 255 |
| CUSTPHONE | Short Text | 255 |
| **ACCOUNT** | | |
| ACCNUM | Long Integer | 4 |
| ACCTYPE | Short Text | 1 |
| ACCBALANCE | Single | 4 |
| ACCSTATUS | Short Text | 1 |
| CUSTNUM | Long Integer | 4 |
| **ACTIVITY** | | |
| TRANSNUM | Long Integer | 4 |
| TRANSDATE | Date With Time | 8 |
| TRANSTYPE | Short Text | 1 |
| TRANSAMT | Single | 4 |
| ACCNUM | Long Integer | 4 |
| **ACCOUNTSTATUSES** | | |
| ACCSTATUS | Short Text | 1 |
| ACCSTATUSNAME | Short Text | 10 |
| **ACCOUNTTYPES** | | |
| ACCTYPE | Short Text | 1 |
| ACCTYPENAME | Short Text | 10 |
| **TRANSTYPES** | | |
| TRANSTYPE | Short Text | 1 |
| TRANSTYPENAME | Short Text | 10 |

**Figure 5**. Microsoft Access schema for the bank database

**Develop code**. The coding process is the translation of the algorithms using a high-level language consistent with the data model. For the bank IS, the high-level language is VB.NET (with the Microsoft Access database) and all algorithms are translated into the equivalent VB.NET code. Appendix A shows the VB.NET code for opening a new account, which assumes that the Open button is clicked on the TabPage for opening an account.

### 3.7 Testing

The testing stage represents activities related to verifying the results of the IS. This includes whether the computations are accurate, whether the operations on the data tables are correctly handled (e.g., insert and update records), whether the GUI interactions are consistent with the business operations (e.g., customer details obtained only for new customers, whether the IS correctly handles exceptions (e.g., withdrawal under the condition of insufficient funds), and whether the IS adequately satisfies the business requirements.

## 4. DISCUSSION

### 4.1 Student Engagement

The design and development solutions were constructed by engaging students in brainstorming activities. This was possible for all activities described in the instructional framework. Based on their own experiences with banking, students were able to immediately relate the features that should be included in the IS application. Students were asked to help identify the specific requirements, boundaries, and assumptions for the IS application. They engaged in a process of critically thinking about the business activities relating to customers and accounts. Students also participated in designing the data and algorithms. This was done by asking them to think about their own activities at the bank including when they opened a new account and the transactions (deposit and withdrawal) they may have completed at the bank. Students were encouraged to reflect on the data items that would be necessary and the steps that would be needed to complete these activities. This critical thinking activity resulted in the design of the database structures along with the algorithms necessary to complete the activities. Students were given opportunities to implement the database in Microsoft Access and design the GUI in VB.NET consistent with the data models finalized for the IS application. Students were

motivated to identify various test cases using which the IS application should be tested.

### 4.2 Student Learning

Students were able to apply their learning based on the instructional approach to other scenarios during the semester. This was particularly true for their course project that required them to design and develop and IS application for inventory management that supported product sales to consumers, product replenishment orders to suppliers, and product restocking when the orders were received from suppliers. Multiple deliverables were required for the project over time such that students can independently apply the framework and also receive feedback from the instructor between stages. The first deliverable dealt with the eliciting and scoping stages. Students were required to identify the needs, identify the boundaries, and define the assumptions for the project using the scenario description. The second deliverable dealt with the design stage. Students were required to design the data model, design the algorithms for the activities, and design the GUIs or mock-ups. The third deliverable dealt with the development and testing stages. Students were required to create the database in Microsoft Access, develop the code in VB.NET, and test the application for accuracy, consistency, and adequacy. The statistics of the overall student performance in the course project was good with mean = 87% and standard deviation = 5%.

At the end of the semester, students shared several positive comments on their learning experience in their course and how the instructional approach enabled their learning. One student stated: "I think the analysis and design stages… will give the best chance of understanding the requirements that the IS application should address… and 80% of your time should be spent in these stages."

Another student stated: "One of the biggest things I have learned is that to successfully build a system that meets all business' requirements, you need to ask questions and expand on what exactly the business is looking for in their system… I didn't know how much thought goes into building a system before you even touch a database or build an application. I learned that when developing a system, it is important to analyze the requirements as a whole."

Yet another student stated: "I have learned that a system design is determined by a client needs… I have learned that algorithms should be efficient, effective, and portable across systems… I have learned that the system should meet business processes, policies, and rules."

### 4.3 Project Extensions

The scenario can be extended with other features for additional instruction and learning opportunities depending on the time available in the semester. One set of extensions could be changes to the business operations and policies. For instance, the bank could introduce different types of checking accounts and charge service fees. The service fees can be fixed (e.g., flat rate of $10 per month for any account) or varying (e.g., $0 for free checking accounts, $5 per month for student checking accounts, $10 per month for non-student checking accounts). The bank may also allow interest on all premium checking accounts. The interest rates may be fixed or varying depending on the type of account (e.g., 0% for free accounts, 2.5% for premium accounts). Also, the date of opening for each account could be tracked. These extensions would require changes to the database structure in terms of attributes and changes to the functionalities of the IS. Algorithms will then be needed to: a) apply service fees on all active checking accounts at fixed intervals (e.g., first day of every month), b) apply service fees on all active checking accounts at rolling intervals (e.g., 31 days after opening date or 31 days after the last date fees was applied), c) apply interest to all accounts at fixed intervals (e.g., last day of every month), and d) apply interest to all accounts at rolling intervals (e.g., 31 days after opening date or 31 days after the last date interest was applied).

Another set of extensions could be reporting functionalities that rely on querying databases through the LINQ system or even the Structured Query Language (SQL) capabilities embedded within the IS. Reports could include: a) monthly report of all transactions for each account organized by money-in and money-out (on the first day of the new month), b) monthly report of all transactions by account for each customer (on the first day of the new month), c) report of all interest amounts applied to accounts by each account for each customer (on the first day of the new year), and d) report of all transactions organized by money-in and money-out for the day (at the end of business each day). These set of

extensions could also include analytical querying and reporting such as: a) total balance across all accounts held by each customer, b) total deposits by time periods such as day, week, month, or year, and c) total interest paid to customers by time periods such as month or year.

## 5. CONCLUSION

This paper proposes an instructional framework for teaching the design and development of database-based IS applications to undergraduate students. Since business operations are largely reliant on databases for persistent storage of data, the importance of database applications development cannot be understated. The instructional framework allows for a structured mechanism by which students can effectively learn the design and development of database applications for business operations.

## 6. REFERENCES

Jeyaraj, A. (2024). Scaffolding in Business Analytics Education: Using Python for Web Scraping. *Journal of Information Systems Education*, *35*(4), 438-450.

Jiang, Y. (2022). Apply Small Teaching Tactics in an Introductory Programming Course: Impact on Learning Performance *Journal of Information Systems Education*, *33*(2), 149-158.

Menon, P. (2023). An Example-Based Instructional Method to Develop Students' Problem-Solving Efficacy in an Introductory Programming Course. *Journal of Information Systems Education*, *34*(1), 1-15.

Newby, M., & Nguyen, T. (2007). Using the Same Problem with Different Techniques in Programming Assignments. *Journal of Information Systems Education*, *18*(3), 279-282.

Owolabi, J., Adedayo, O. A., Amao-Kehinde, A. O., & Olayanju, T. A. (2013). Effects of Solo and Pair Programming Instructional Strategies on Students' Academic Achievement in Visual-Basic.Net Computer Programming Language. *GSTF Journal on Computing*, *3*(3), 1-4.

Paxton, J. (2002). Live Programming as a Lecture Technique. *Journal of Computing Sciences in Colleges*, *18*(2), 51-56.

Ritzhaupt, A. D., & Zucker, R. J. (2006). Teaching Object-Oriented Programming Concepts using Visual Basic .NET. *Journal of Information Systems Education*, *17*(2), 163-169.

Sharma, M., Biros, D., Ayyalasomayajula, S., & Dalal, N. (2020). Teaching Programming to the Post-Millennial Generation: Pedagogic Considerations for an IS Course *Journal of Information Systems Education*, *31*(2), 96-105.

Simkin, M. G. (2004). A First Case Project in Visual Basic.Net: Preparing an Income Tax Return. *Journal of Information Systems Education*, *15*(2), 119-125.

Simkin, M. G. (2007). A Term Project in Visual Basic: The Downhill Snowboard Shop. *Journal of Information Systems Education*, *18*(1), 21-30.

Smith, T. C., & Jones, L. (2021). First Course Programming Languages within US Business College MIS Curricula *Journal of Information Systems Education*, *32*(4), 283-293.

Wang, S., & Wang, H. (2021). A Teaching Module of No-Code Business App Development *Journal of Information Systems Education*, *32*(1), 1-8.

West, C., & Pardieck, S. C. (2004). Contractual and Team Programming in a Visual Basic .NET Course. *Issues in Information Systems*, *V*(2), 706-712.

ISSN: 2473-4901
v12 n6761

**APPENDIX A**
**VB.NET Code for opening new account**

```
    Private Sub btnOpen_Click(sender As Object, e As EventArgs) Handles btnOpen.Click
        If ValidateControls(tpgOpen) = True Then
            ' Gather data for ACCOUNT table
            Dim AccNum As Integer = Val(txtAccNum.Text)
            Dim AccType As Char = lstAccType.SelectedValue
            Dim AccBal As Single = Val(txtOTransAmt.Text)
            Dim AccStatus As Char = "O"
            Dim CustNum As Integer
            If chkNewCustomer.Checked = True Then
                CustNum = Val(txtCustNum.Text)
            Else
                CustNum = Val(cboCustNum.Text)
            End If
            ' Gather data for ACTIVITY table
            Dim TransNum As Integer = Val(txtOTransNum.Text)
            Dim TransDate As Date = dtpTransDate.Text
            Dim TransType As Char = "D"
            Dim TransAmt As Single = Val(txtOTransAmt.Text)
            ' Gather data for CUSTOMER table
            Dim CustName = txtCustName.Text
            Dim CustPhone = mskCustPhone.Text
            ' STEP A: Populate the database tables
            ' 1. CUSTOMER table
            If chkNewCustomer.Checked = True Then
                Try
                    BankDataSet.CUSTOMER.AddCUSTOMERRow(CustNum, CustName, CustPhone)
                Catch ex As Exception
                    MessageBox.Show(ex.Message)
                End Try
            End If
            ' 2. ACCOUNT table
            Try
                BankDataSet.ACCOUNT.AddACCOUNTRow(AccNum, AccType, AccBal, AccStatus,
              CustNum)
            Catch ex As Exception
                MessageBox.Show(ex.Message)
            End Try
            ' 3. ACTIVITY table
            Try
                BankDataSet.ACTIVITY.AddACTIVITYRow(TransNum, TransDate, TransType,
                    TransAmt, AccNum)
            Catch ex As Exception
                MessageBox.Show(ex.Message)
            End Try
            ' STEP B: Commit the changes to the database
            If chkNewCustomer.Checked = True Then
                CUSTOMERTableAdapter.Update(BankDataSet.CUSTOMER)
            End If
            ACCOUNTTableAdapter.Update(BankDataSet.ACCOUNT)
            ACTIVITYTableAdapter.Update(BankDataSet.ACTIVITY)
        Else
            MessageBox.Show("Data missing!")
        End If
    End Sub
```