LogicGem 4.0
QuickStart

QuickStart Contents

Introduction

Whether your need is to:

- Build a representation of Business Rules
- Program a large, complex logic construct
- Create a software "Testing Suite" with the confidence that it is 100% complete
- Build a Knowledge Base for an Expert System
- Validate the completeness of the logic in an Embedded System before it ships
- Verify, compile and document procedural logic or processes

This "QuickStart" guide will bring alive some of the many ways LogicGem can help you accomplish any of the above tasks plus many others.

You can also view our tutorials on using LogicGem on our web site at:

https://logicgem.com/videos/

Thank you for your interest in LogicGem!

LogicGem Overview

The LogicGem logic processor software tool is designed to provide a familiar, easy-to-use way to create, edit, verify and compile decision table logic. LogicGem was created for both *Programmers* and *Non-programmers*. Domain experts such as business analysts can create decision tables using LogicGem to develop a logically concise representation of a business process. LogicGem also provides the means to prototype and expand a high-level procedural concept into a decision table, work it through analysis and design phases, and then generate English, French, German or Spanish language documentation and programming source code in various programming languages.

LogicGem provides tools to ensure that a table's logic is complete, unambiguous and contains every applicable rule in an expanded or reduced format. In addition, an optional feature can be used to optimize the generated program code.

Decision Tables

Decision tables, also known as *edit* or *logic* tables, have been used for centuries to represent logic in a tabular form. They continue to be a processing method of choice because they provide a quick and easy way to read, understand and execute procedures.

Figures 1 and 2 illustrate how to understand the various components of a decision table.

A decision table is a map representing the relationships of combinations of *conditions* to combinations of *actions.* The first combination of conditions to actions in Figure 1 is read from top to bottom from the *rule* column as: *if* Condition 1 is true *and* Condition 2 is true *and* Condition 3 is true *then do* Action 1 *and then do* Action 4 *and then do* Action 2.

Condition 1 Condition 2 Condition 3								
Action 1 Action 2 Action 3 Action 4								

Figure 1 – Decision Table Structure

The four components of a decision table (*condition stubs, condition entries, action stubs* and *action entries*) are shown in Figure 2.

Conditions to evaluate in a procedure are entered in the *condition stub* area of a decision table. Each condition stub entry is followed with a yes/no matrix in the *condition entry* area of the table. The yes/no matrix is used to represent all the possible true/false combinations of conditions in the table. You can also enter "-" (the hyphen character) in the cell to signify that you "don't care" if the condition's state is true or false.

Actions to perform under a set of true/false conditions are entered in the *action stub* area. A number in the *action entry* area identifies the action(s) to perform and in the required sequence.

The data in the *condition entries* together with the data in the *action entries* is read from top to bottom by column. The data grouped in each column is called a *rule*.

Condition Stubs	Condition Entries
Action Stubs	Action Entries

Figure 2 – Decision Table Components

LogicGem can be used to automatically reduce a decision table matrix but it is up to the developer to determine if the rules are logical. Furthermore, LogicGem does not understand the syntax contained in the conditions and actions entered in a decision table.

Each column in a table should be different. A decision table is referred to as *mechanically perfect* when all contradictions and redundancies are removed.

LogicGem provides the tools to automatically uncover common decision table errors such as:

Incompleteness: Not all conditions are covered.

Contradiction: Two rules with the same conditions lead to different actions.

Redundancy: Two rules with the same conditions lead to the same action.

Ambiguity: A reduced table with contradictory and/or redundancy errors.

Figure 3 shows examples of redundant, contradictory and ambiguous rules.

Rule 5 and Rule 6 are redundant.

Rule 5 and Rule 6 are contradictory to Rule 7.

Rule 1, Rule 3 and Rule 8 are ambiguous.

What to do today?								
Is it raining? Is it snowing? Are you married?								
Go golfing								
Go skiing								
Go for a picnic								
Watch movies								
Watch sports on TV								
Cuddle by the fire								

Figure 3 - Redundant, Contradictory and Ambiguous rules

This guide serves as the point-of-entry for someone new to decision table technology. The best way to gain familiarity with decision tables in general and LogicGem in particular is through a series of specific example applications. In the next two sections, we'll present examples for using LogicGem to develop non-programming business rules that might be designed by a business analyst. The end product will be English language procedural rules that represent the requirements. Then, we'll present programming specific examples which ultimately yield programming code that can be integrated directly into a software application.

Business Rule Examples

This section provides two examples of how to use LogicGem for creating bullet proof business logic. In each case, we'll start with a description of the business requirements in general terms (the kind you typically get in a strategy meeting) and then demonstrate how to incrementally build a decision table to represent the rules that represent the requirements. Once the table is logically complete, we'll finish up by generating the English language rule-set that can be the basis of software documentation.

Example 1: RFID Wine Distribution Center

The first business rules example will demonstrate how LogicGem can be used to solve a simple logistics problem for a wine distribution center equipped with radio frequency identifier (RFID) technology.

The scenario goes as follows. A wine distribution business receives cases of wine bottles in a warehouse. There are three lines of wine received at a receiving dock. Each line has a unique stock keeping unit (SKU). Within the warehouse, there is one assembly line for each SKU where workers provide special packaging and premiums for retail stores.

The business rules are simple; for each different SKU, activate a special forklift that transports the cases to the appropriate assembly line. In addition, there is a temperature sensor at the receiving dock that determines whether the temperature is outside the optimal range for wine storage. If so, the case needs to be checked for spoilage before being transported to the assembly line. In the case of SKU #303, we need to check for spoilage regardless of the sensor's reading (this particular wine has special needs).

Here are the conditions we need to consider in this simple example:

- RFID reader scans wine SKU #101
- RFID reader scans wine SKU #202
- RFID reader scans wine SKU #303
- Temperature sensor shows 50-57 degrees

Here are the actions we need to consider in this simple example:

- Activate forklift to assembly line 1
- Activate forklift to assembly line 2
- Activate forklift to assembly line 3
- Check spoilage
- Recalibrate RFID reader due to false read

The resulting decision table is shown in Figure 4. You can load a completed version of this table by selecting **File** | **Open**, highlighting the rfid.lgt table file in the LogicGem Examples folder, and clicking the Open button.

Let's consider the rules to see how the actions are tied to conditions. Rule 1 is the simplest as it considers the case where the temperature sensor is out of range. So regardless of the values of the other conditions ("don't care" values), if the value for C4 is N, then we carry out the action to check spoilage. Rule 4 is interesting because there are two actions to carry out. First, if conditions C3 and C4 are both Y, then we need to activate the forklift for assembly line 3, and check spoilage. Rule 5 is sort of a catch-all rule accommodating the case where the RFID reader reads a case, but does not recognize it as a valid SKU.

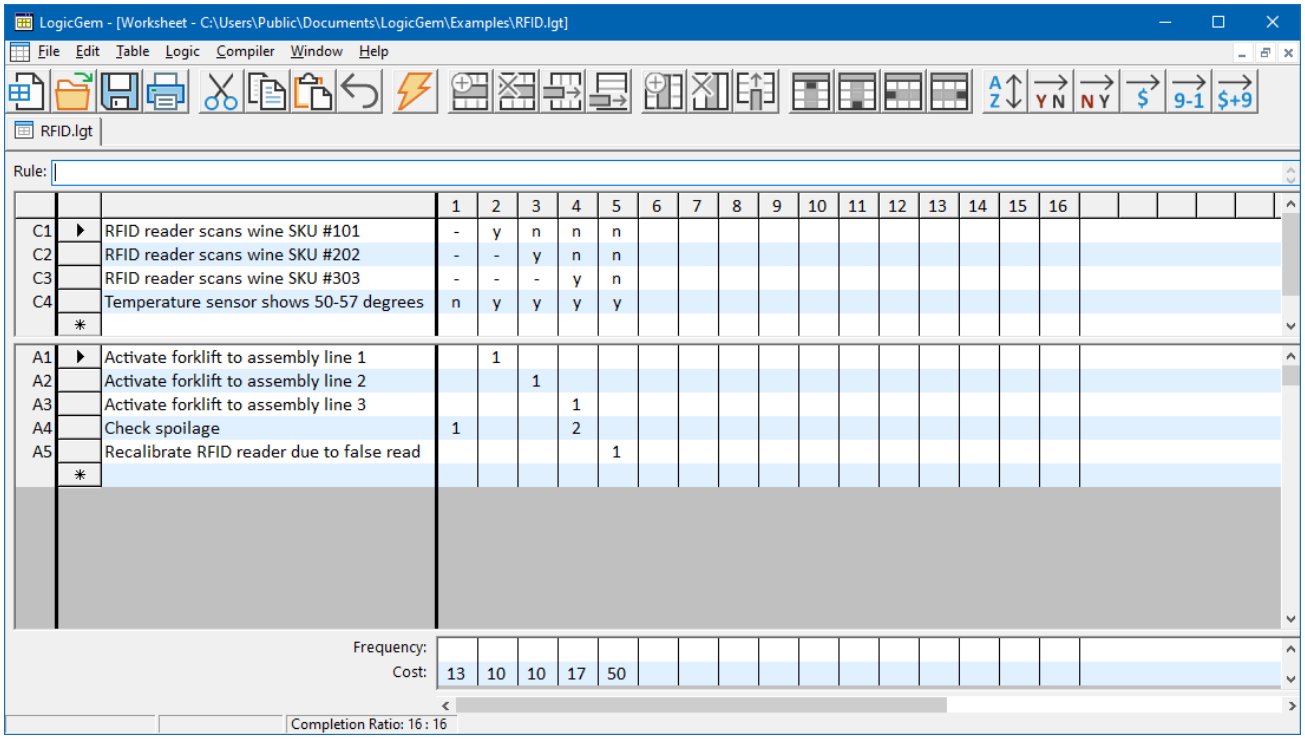

Figure 4 - RFID Wine Distribution Center

Cost: you're able to enter a two digit number in the cost row indicating a relative cost estimate assigned to a specific rule. Cost is a subjective measure of how much a rule will cost to execute.

Notice that we entered some integer values in the cost row at the bottom of the decision table. Assigning weighted costs to each rule adds another dimension to how LogicGem can optimize business logic. In this case, we assign a relatively high value of 50 to the rule where recalibration of the RFID reader is needed, and relatively low values of 10 to simple rules where only forklift activation is required. LogicGem's logic compiler is able to sort the resulting business logic by cost to obtain a potentially more optimal view of the logic.

Let's see what our business logic looks like by asking LogicGem to use the logic compiler to convert the decision table to English language rules. Start by selecting the **Compiler** | **Language** menu and select "English" and then selecting **Compiler** | **Structure** | **Rule List** (a good option for compiling decision tables representing business requirements as opposed to programming code). Lastly, select **Compiler**

| **Options** menu and turn on Rule Numbers and Rule Descriptions (turn on the right hand side check boxes).

Next select **Compiler** | **Compile** or press the **F7** function key to tell LogicGem to compile the decision table in order to yield business rules. See the English business rules below.

```
Procedure: rfid.lgt

1
If  Temperature sensor shows 50-57 degrees is not true , then
             rule 1
             *** When sensor show out-of-range, must check for spoilage
            Check spoilage,

2
If  RFID reader scans wine SKU #101 is true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 2
             *** Use assembly line 1 forklift to transport cases of SKU #101
            Activate forklift to assembly line 1,

3
If  RFID reader scans wine SKU #101 is not true and
    RFID reader scans wine SKU #202 is true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 3
             *** Use assembly line 2 forklift to transport cases of SKU #202
            Activate forklift to assembly line 2,

4
If  RFID reader scans wine SKU #101 is not true and
    RFID reader scans wine SKU #202 is not true and
    RFID reader scans wine SKU #303 is true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 4
             *** Use assembly line 3 forklift to transport cases of SKU #303,
                 and also check for spoilage
            Activate forklift to assembly line 3,
            Check spoilage,

 5
If  RFID reader scans wine SKU #101 is not true and
    RFID reader scans wine SKU #202 is not true and
    RFID reader scans wine SKU #303 is not true and
    Temperature sensor shows 50-57 degrees is true , then
             rule 5
             *** We got a false read from RFID reader, must recalibrate the
                 device
            Recalibrate RFID reader due to false read,

End
```

Example 2: Complex Shipping Charge Calculation

With the next business rules example, we'll describe a more complex set of business requirements to show how easily LogicGem can address intricate logic and provide the business analysts with a sense of confidence that the rule-set is complete.

The example we'll consider in this section is a shipping charge calculation. Determining a shipping charge to fulfill a product order is often comprised of a number of interrelated factors such as:

- Dollar value of the order
- Customer status (e.g. new vs. favored)
- Order weight
- Destination (e.g. domestic vs. international)
- Special handling requirements (e.g. hazardous materials)
- Carrier (e.g. Federal Express, Airborne Express, etc.)

For this example, we'll consider the following business rules, each playing a role in determining the shipping charge:

- Is it a favored customer?
- Is the total order amount more than $500?
- Is the total weight of the order more than 25kg?
- Domestic or international delivery?
- Does the order require special handling for hazardous materials?

You can load a completed version of this table by selecting **File** | **Open**, highlighting the shipping.lgt table file in the LogicGem Examples folder, and clicking the Open button.

Figure 5 shows the selected condition stubs and rules for this business problem. We can examine a few rules to make sure we understand the intent. First let's consider the first condition stub "Favored customer." If the current customer has favored status then shipping is easy, it's free. This means that the other conditions that determine the cost of shipping can be ignored using "don't cares" for C2 and C3 as shown in Rule 1. Rule 9 is interesting because we need to itemize all its Y or N values: not a favored customer, small order amount, light weight order, domestic destination, and no special handling. This rule is probably the most common rule due to all factors. Finally, Rule 7 is the most complex: not a favored customer, small order amount, international destination, and special handling required. In this case, the shipping business requirements say that weight isn't a factor in determining a charge.

You might also consider using the Rule field located just above the decision table area of the screen. Here you can write descriptive text for each individual rule. When you go to compile the decision table, you'll have the option of including the Rule text for the purpose of documenting the compiled table. Notice in Figure 5 that with Rule 7 selected, its rule description appears in this field.

Rule: You're able to enter descriptive text in the Rule field for each rule in the decision table.

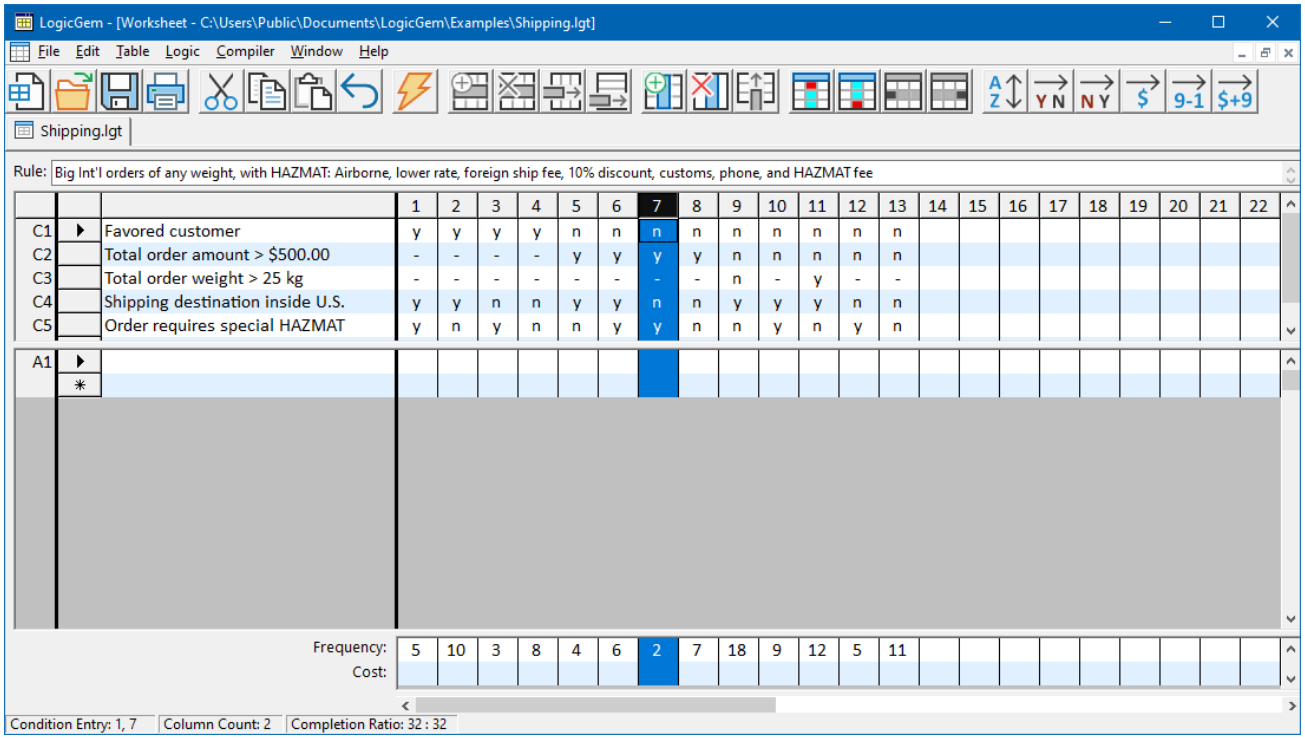

Figure 5

Now let's review the completed decision table with the actions filled in as shown in Figure 6. The business requirements call for the following actions to be considered:

- Apply a 10% discount on the shipping charge.
- Free shipping.
- Use Federal Express as the shipping vendor.
- Use Airborne Express as the shipping vendor.
- Have customer service phone the customer to discuss special handling requirements.
- Special customs documentation for international destinations.
- Shipping fee calculation: total order weight times the factor $1.50/kg.
- Shipping fee calculation: total order weight times the factor $1.25/kg.
- Add $45 special handling fee for hazardous materials.
- Add $17.50 foreign shipping fee for international orders.

Going back to Rule 7, we see that seven actions are required to satisfy this rule.

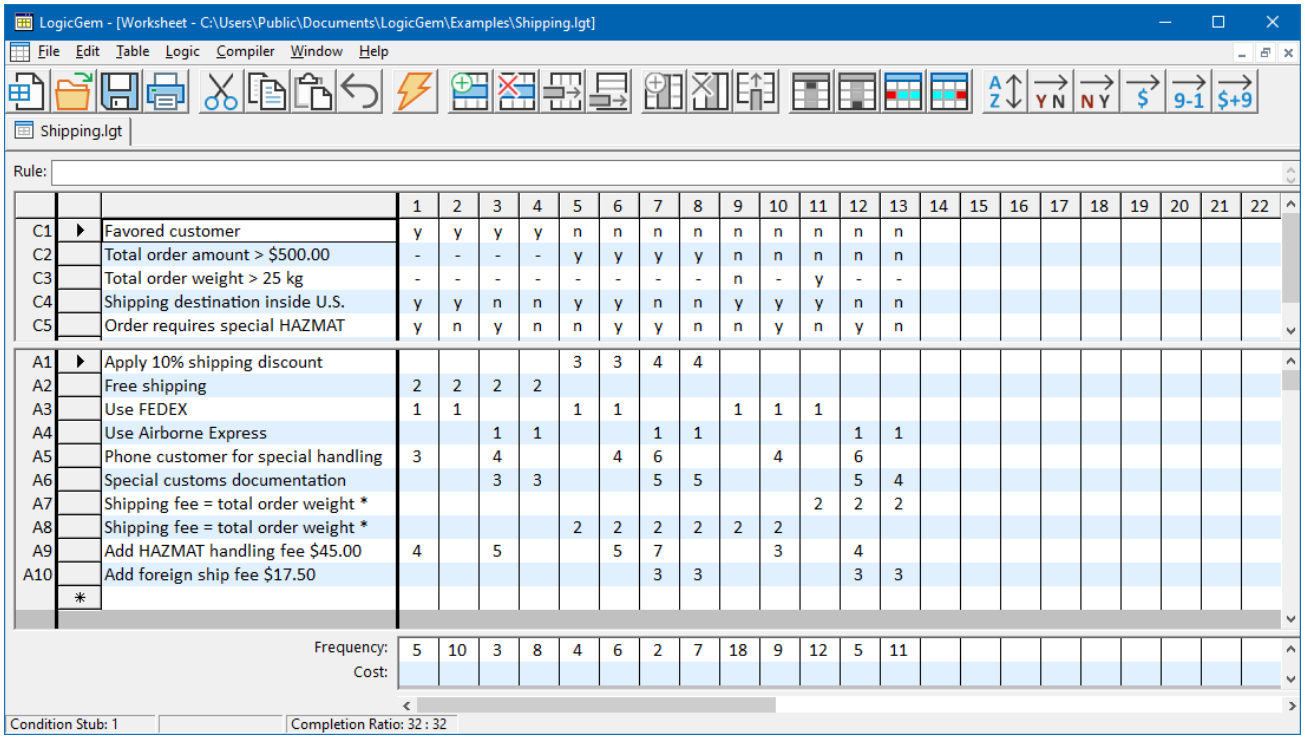

Figure 6

Note also that we filled in the frequency row values at the bottom of the decision table with values showing the expected relative frequency of the rules. LogicGem provides an option for sorting rules in the resulting compilation based on frequency. Using this feature, the most frequently fired rules will be at the top of your code segment, resulting in the performance of your code being automatically optimized.

Frequency: You're able to enter a two digit number in the frequency row indicating an estimate of how often a particular rule might be expected to be invoked.

Now let's have LogicGem generate our business logic by using the logic compiler to convert the decision table to English language rules. Start by selecting the **Compiler** | **Language** menu and select "English" and then selecting **Compiler** | **Structure** | **Nested If/Else**. Lastly, select **Compiler** | **Options** menu and turn on **Rule Numbers** and **Rule Descriptions** (turn on the right hand side check boxes).

Now select **Compiler** | **Compile** to tell LogicGem to compile the decision table in order to yield business rules. See the English business rules below. It is a useful exercise to trace through the English business rules and compare them with the decision table rules. By performing this manual check on a couple of rules, you'll better understand how the decision table is translated by LogicGem into a logically complete structure.

```
Procedure: shipping.lgt

    If Favored customer is true, then
        If Shipping destination inside U.S. is true, then
            If Order requires special HAZMAT (hazardous materials)
   packing is true, then
                rule 1
                Favored USA customer with HAZMAT: FEDEX, free
                    shipping, phone, and HAZMAT fee
                Use FEDEX,
                Free shipping,
                Phone customer for special handling,
                Add HAZMAT handling fee $45.00,
            Otherwise,
                rule 2
                Favored USA customer, no HAZMAT: FEDEX and free
                    shipping
                Use FEDEX,
                Free shipping,
        Otherwise,
            If Order requires special HAZMAT (hazardous materials)
                packing is true, then
                rule 3
                Favored Int'l customer with HAZMAT:  Airborne,
                    free shipping, customs, phone, HAZMAT fee
                Use Airborne Express,
                Free shipping,
                Special customs documentation,
                Phone customer for special handling,
                Add HAZMAT handling fee $45.00,
            Otherwise,
                rule 4
                Favored Int'l customer, no HAZMAT: Airborne,
                    free shipping, and customs
                Use Airborne Express,
                Free shipping,
                Special customs documentation,
```

```
Otherwise,
    If Total order amount > $500.00 is true, then
        If Shipping destination inside U.S. is true, then

            If Order requires special HAZMAT (hazardous materials)
                packing is not true, then
                rule 5
                Big USA orders, of any weight, no HAZMAT:
                    FEDEX, lower rate and 10% discount
                Use FEDEX,
                Shipping fee = total order weight * $1.25/kg,
                Apply 10% shipping discount,
            Otherwise,
                rule 6
                Big USA orders of any weight, with HAZMAT:
                    FEDEX, lower rate,10% discount, phone, and HAZMAT fee
                Use FEDEX,
                Shipping fee = total order weight * $1.25/kg,
                Apply 10% shipping discount,
                Phone customer for special handling,
                Add HAZMAT handling fee $45.00,
        Otherwise,
            If Order requires special HAZMAT (hazardous materials)
                packing is true, then
                rule 7
                Big Int'l orders of any weight, with HAZMAT:
                    Airborne, lower rate, foreign ship fee, 10% discount,
                    customs, phone, and HAZMAT fee
                Use Airborne Express,
                Shipping fee = total order weight * $1.25/kg,
                Add foreign ship fee $17.50,
                Apply 10% shipping discount,
                Special customs documentation,
                Phone customer for special handling,
                Add HAZMAT handling fee $45.00,
            Otherwise,
                rule 8
                Big Int'l orders of any weight, no HAZMAT:
                    Airborne, lower rate, foreign ship fee, 10% discount,
                    and customs
                Use Airborne Express,
                Shipping fee = total order weight * $1.25/kg,
                Add foreign ship fee $17.50,
                Apply 10% shipping discount,
                Special customs documentation,
    Otherwise,
        If Total order weight > 25 kg is not true, then
            If Shipping destination inside U.S. is true, then
                If Order requires special HAZMAT (hazardous
                    materials) packing is not true, then
                    rule 9
                    Small USA orders with low weight, no
                        HAZMAT: FEDEX, and lower rate
                    Use FEDEX,
                    Shipping fee = total order weight * $1.25/kg,
                Otherwise,
                    rule 10
                    Small USA orders, of any weight with
                        HAZMAT: FEDEX, lower rate, HAZMAT fee, and phone
                    Use FEDEX,
                    Shipping fee = total order weight * $1.25/kg,
                    Add HAZMAT handling fee $45.00,
                    Phone customer for special handling,
```

```
            Otherwise,
                If Order requires special HAZMAT (hazardous
                    materials) packing is true, then
                    rule 12
                    Small Int'l orders with HAZMAT of any
                        weight: Airborne, higher rate, foreign fee,
                        HAZMAT fee, customs, and phone
                    Use Airborne Express,
                    Shipping fee = total order weight * $1.50/kg,
                    Add foreign ship fee $17.50,
                    Add HAZMAT handling fee $45.00,
                    Special customs documentation,
                    Phone customer for special handling,
                Otherwise,
                    rule 13
                    Small Int'l orders of any weight, no
                        HAZMAT: Airborne, higher rate, foreign fee
                        and customs
                    Use Airborne Express,
                    Shipping fee = total order weight * $1.50/kg,
                    Add foreign ship fee $17.50,
                    Special customs documentation,
        Otherwise,
            If Shipping destination inside U.S. is true, then
                If Order requires special HAZMAT (hazardous
                    materials) packing is true, then
                    rule 10
                    Small USA orders, of any weight with
                        HAZMAT: FEDEX, lower rate, HAZMAT fee, and phone
                    Use FEDEX,
                    Shipping fee = total order weight * $1.25/kg,
                    Add HAZMAT handling fee $45.00,
                    Phone customer for special handling,
                Otherwise,
                    rule 11
                    Small USA order, high weight and no HAZMAT:
                        FEDEX at higher rate
                    Use FEDEX,
                    Shipping fee = total order weight * $1.50/kg,
            Otherwise,
                If Order requires special HAZMAT (hazardous
                    materials) packing is true, then
                    rule 12
                    Small Int'l orders with HAZMAT of any
                        weight: Airborne, higher rate, foreign fee,
                        HAZMAT fee, customs, and phone
                    Use Airborne Express,
                    Shipping fee = total order weight * $1.50/kg,
                    Add foreign ship fee $17.50,
                    Add HAZMAT handling fee $45.00,
                    Special customs documentation,
                    Phone customer for special handling,
                Otherwise,
                    rule 13
                    Small Int'l orders of any weight, no
                        HAZMAT: Airborne, higher rate, foreign fee
                        and customs
                    Use Airborne Express,
                    Shipping fee = total order weight * $1.50/kg,
                    Add foreign ship fee $17.50,
                    Special customs documentation,
End
```

Programming Examples

This section provides two examples of how to use LogicGem for the solution of programming problems. In each case, we'll start with a description of the business requirements and then demonstrate how to incrementally build a decision table to represent the logic. Once the table is logically complete, we'll finish up by generating the programming code that can be integrated with a software application.

Example 3: Sales Discount Calculation

The first programming example illustrates the use of a decision table to represent the programming logic for calculating a sales discount. The discount is to be based on two business requirements. First, a special discount is assigned if it is a new customer making the purchase. Second, a discount schedule is applied for existing customers based on the sales amount of the order, where a bigger discount is given for larger orders. Table 1 shows the discount rate schedule provided by the sales department of the company.

Business Rule	Discount
New customer	5%
Amount of sale less than $500.00	0%
Amount of sale $500.00 - $1,499.99	1%
Amount of sale $1,500.00 - $4,999.99	2%
Amount of sale greater than or equal to $5,000.00	5%

Table 1 - Sample Discount Rate Schedule

Our job as a programmer is to code this discount schedule in the programming language of choice, and make sure that all conditions have been considered. Typically, developers will attempt to convert business rules such as this without any software engineering tools. The resulting code for simple requirements (such as this example) has a high probability of success, however for complex sets of requirements it becomes increasingly likely that some rules will be missed, resulting in incomplete program logic, and fallible software applications.

Let's use Logic Gem to develop the required program logic in a way that we're 100% confident that the code will be complete.

You can load a completed version of this table by selecting **File** | **Open**, highlighting the sale_discount.lgt table file in the LogicGem Examples folder, and clicking the Open button. Optionally, you can begin experiencing how easy LogicGem is to use by actually building this next example from scratch.

We'll start by opening up a new decision table by selecting **File** | **New**. The first thing we'll do is fill in all the condition stubs we can think of. We can obtain the list of conditions from the discount rate schedule table and enter them as condition stubs. Since this is a programming example, the condition stubs must be valid logical expressions of the target programming language (in our case Visual Basic). Figure 7 shows four condition stubs: C1, C2, C3, and C4.

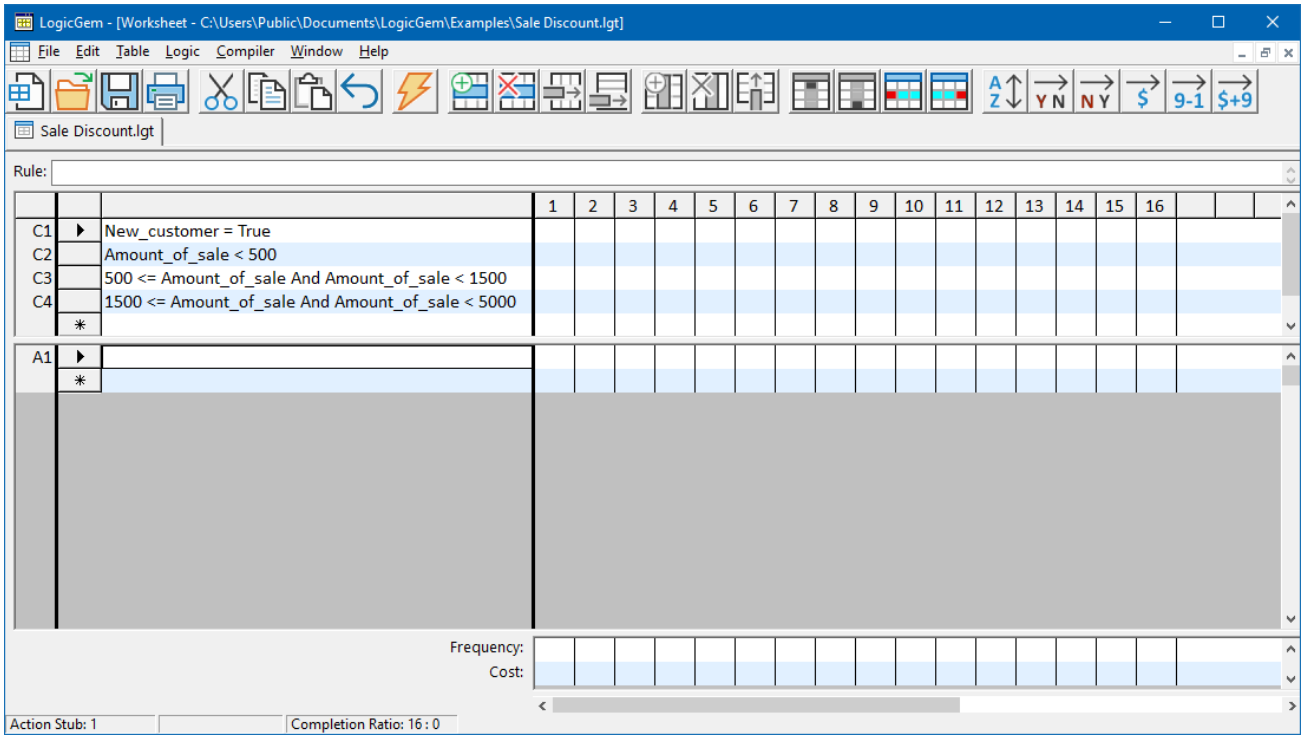

Figure 7

You might wonder what about the last rule in the discount rate schedule table? We can add the last rule which accounts for large orders over $5,000, but if you consider how a decision table works, you'll see that we can get away with eliminating this rule. We'll return to this issue later when we fill in the rules.

Next, let's fill in all the action stubs we can come up with. We can base the action stubs on the above discount table as well. Specifically, we can use the discount percentages in the table to formulate the actions. Since we wish for Logic Gem to generate actual programming code for this example, we must code the action stubs in the appropriate programming language. In this example we'll use Visual Basic to code the stubs, and the stubs shall contain simple assignment statements.

We see that the action stubs are simple programming assignment statements that calculate the discount amount based on the variable "Amount_of_sale" and then assign it to another variable "Discount_amount". So based on the business requirements, we come up with action stubs: A1, A2, A3, and A4 as shown in Figure 8.

Notice that the Completion Ratio appearing in the status bar of the screen shows "16:0" meaning that based on the conditions you've entered, there are 16 rules that must be accounted for. Currently no rules are defined so a 0 is shown as the number accounted for.

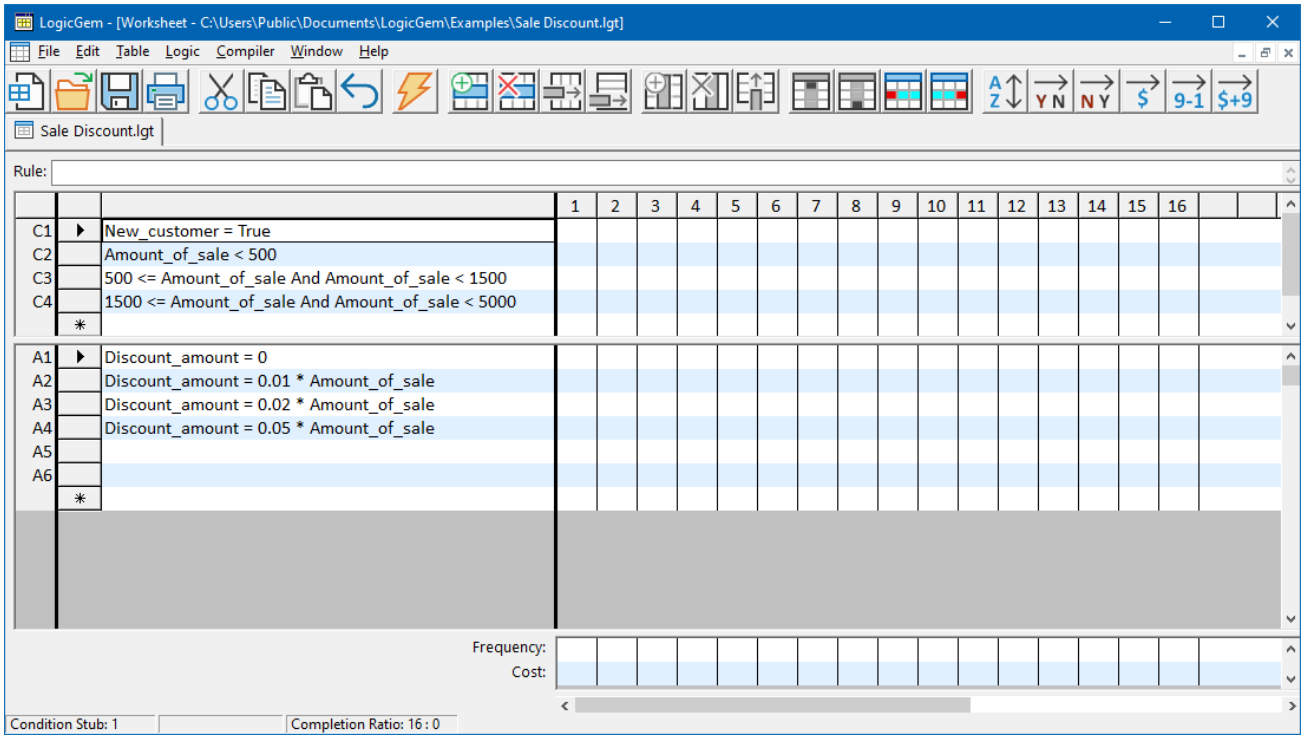

Figure 8

The next step in building the decision table is to start creating rules based on the condition and action stubs. Let's start with an easy one. Our business requirements state that all new customers should receive a 5% discount on their first purchase. We'll assume there is a Boolean variable available in the code base called "New_customer" which has a True value if the customer is new. With this understanding, let's place a "Y" in the Rule 1 column for condition C1, and place a don't-care "-" symbol in the Rule 1 column for conditions C2, C3, and C4. This says that if the customer is new, it doesn't matter what the amount of sale is to get the discount.

Now, let's place a "1" in the Rule 1 column for action A4. Coupled with the condition part of Rule 1, this rule says to calculate a 5% discount amount if the customer is new. You can document the rule in the decision table by entering a rule description in the Rule textbox: "New customers get highest discount as incentive." When LogicGem generates code, you'll have the option of including rule descriptions.

We can add another rule to represent a small order of less than $500 by entering "Y" in the Rule 2 column for condition C2, and "N" for both C3 and C4. We also need to place "N" in C1 since this order is not for a new customer.

Rule 3 is defined in a similar way, "N" for C1, C2, and C4, and "Y" for C3. For Rule 4, enter "N" for C1, C2, and C3, and "Y" for C4.

The final rule is Rule 5 and is handled in an interesting way. This rule satisfies the requirement for large orders over $5,000. We do this by entering "N" for C1, C2, C3, and C4. The business requirement for "Amount of sale greater than or equal to $5,000.00" is handled by a process of elimination since if conditions C1, and C2, and C3 are all false, then we reference action A4 in the action stub.

The completed table with the rules representing all the business requirements is shown in Figure 9. But notice that the Completion Ratio is 16:12 which means the table is logically incomplete. We need to reconsider the rules we've defined thus far to see if there is a way to reconstruct any of them to make the table complete.

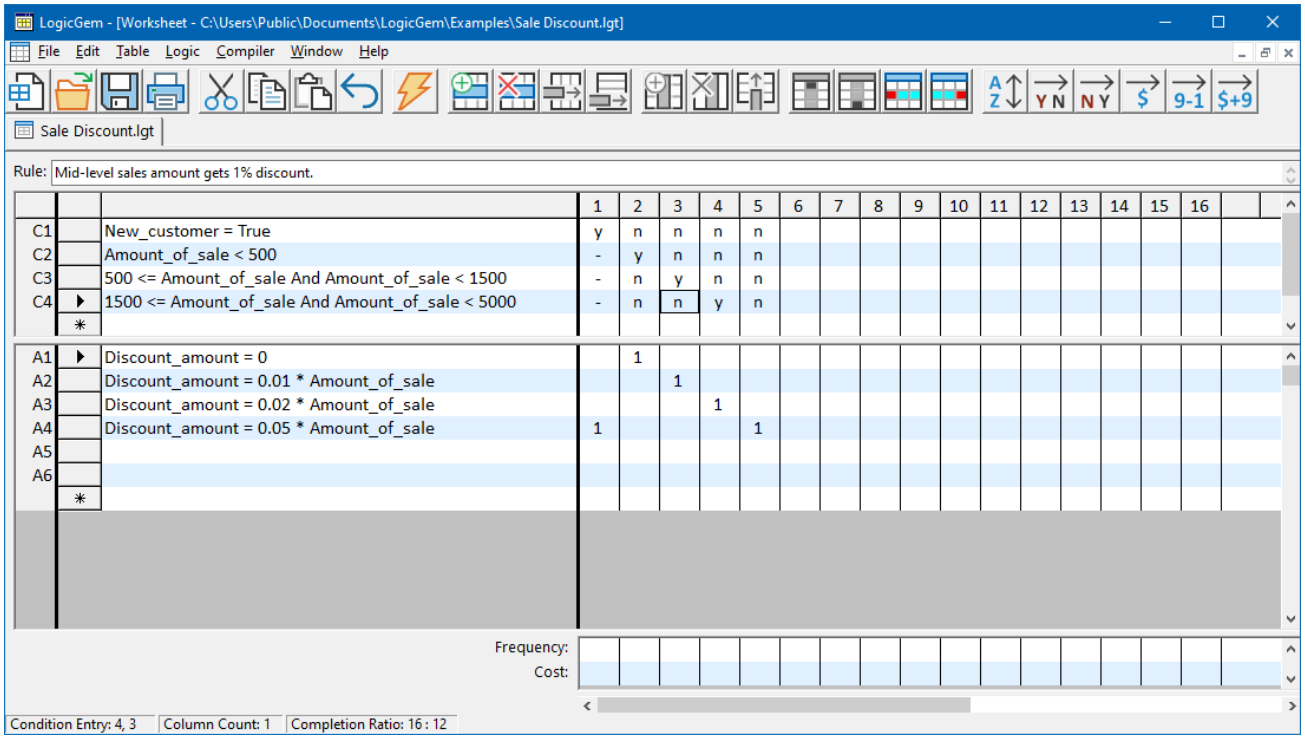

Figure 9

We can modify Rule 2 by entering don't care values for C3 and C4 since if C2 is true, we don't really care what C3 and C4 are. The action for Rule 2 should be A1 set to 1. We can continue with Rule 3 by entering a don't care value for C4. The action for Rule 3 should be A2 set to 1. Rule 4 may be left as-is, with the action A3 set to 1. Now the more optimized table in Figure 10 shows a Completion Ratio of 16:16 indicating a logically complete decision table. For good measure you can select the Logic | Ambiguity Check in order to check the robustness of the decision table. If there are no ambiguous rules in the table, LogicGem will report "Table is unambiguous." If the table is ambiguous, you can select the **Logic** | **Disambiguate** menu option, or the **Logic** | **Expand** option to continue building the logic.

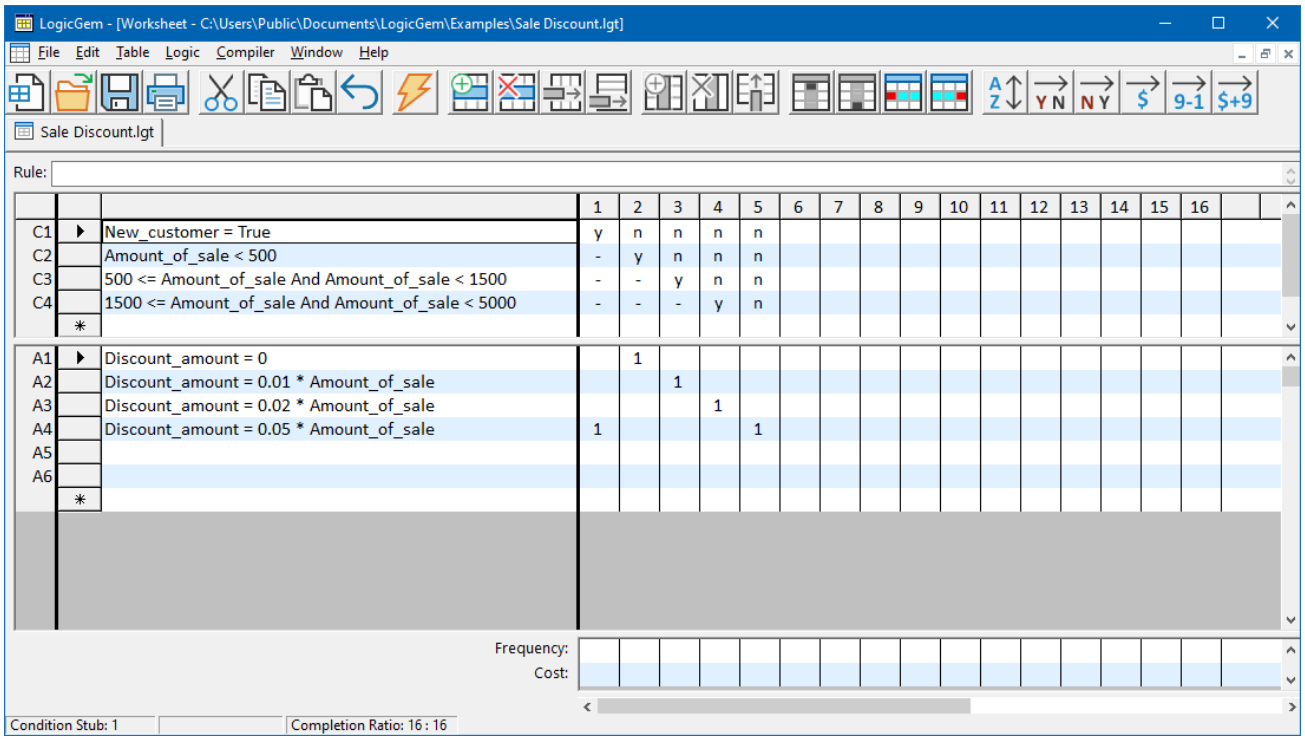

Figure 10

Our final task is to configure Logic Gem's logic compiler to convert the decision table to programming code. Start by selecting the **Compiler** | **Language** menu and select "Visual Basic" and then selecting **Compiler** | **Structure** | **Nested If/Else**. Lastly, select **Compiler** | **Options** menu and turn on **Rule Numbers** and **Rule Descriptions** (turn on the right hand side check boxes).

Now select **Compiler** | **Compile** to tell LogicGem to compile the decision table in order to yield programming code. See the Visual Basic code below.

```
' sale_discount.lgt

If (New_customer = True) Then
    ' rule 1
    ' New customers get highest discount as incentive.
    Discount_amount = 0.05 * Amount_of_sale
Else
    If (Amount_of_sale < 500) Then
        ' rule 2
        ' Small sales get no discount.
        Discount_amount = 0
```

```
    Else
        If (500 <= Amount_of_sale And Amount_of_sale < 1500) Then
            ' rule 3
            ' Mid-level sales amount gets 1% discount.
            Discount_amount = 0.01 * Amount_of_sale
        Else

            If (1500 <= Amount_of_sale And Amount_of_sale < 5000) Then
                ' rule 4
                ' Higher sales amount gets 2% discount.
                Discount_amount = 0.02 * Amount_of_sale
            Else
                ' rule 5
                ' Large sales, over $5,000 get 5% discount.
                Discount_amount = 0.05 * Amount_of_sale
            End If
        End If
    End If
End If
```

The resulting VB .NET code easily can be merged with a main program in order to calculate the sales discount. Of course program variables used in the logic code must be declared in the main program.

Example 4: Dynamically Build SQL Statements

The final example use of LogicGem illustrates how decision tables can be useful in generating programming code to dynamically build structured query language (SQL) statements. Quite often, programmers are faced with accessing databases in a manner that is dictated by a complex set of business rules. The resulting SQL SELECT statement, for example, might access a specific table in a database (or tables via a relational join), select a specific field list, and retrieve records based on a series of selection criteria. In the case we'll consider in this section, the decision table will drive the process of defining logic to build the SQL WHERE clause which is often the most complex part of the SELECT statement.

The business requirement is to pull data from a customer data table in a relational database. The table has two fields that will participate in the selection criteria: CompleteDate, and Username. The values used for comparison with the field values are held in three program variables: dtStartDate, dtEndDate, and sUser. The program also defines two additional variables: sSQL and sWhere to store the constructed SQL statement.

One of the conditions the decision table needs to consider is whether the WHERE clause includes a range of date values for the field CompleteDate, or just the upper date range. It must also consider the case where no Username field value is provided.

Review the decision table in Figure 11 that includes all the conditions, actions, and rules for this example. You can load a completed version of this table by selecting **File** | **Open**, highlighting the build_sql.lgt table file in the LogicGem Examples folder, and clicking the Open button.

The conditions and actions are shown using Visual Basic syntax. As an example, let's take a closer look at Rule 1 which accounts for the case where there is both an upper and lower value for CompleteDate, and there is a value for Username. In this case, the following code will be generated by the decision table based on five different actions included in Rule 1.

```
sSQL = "SELECT * FROM CustData "
                sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
                sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"
                sWhere += " AND RTRIM(Username) = '" & sUser & "'"
                sSQL = += sWhere
```

But there is a deficiency in the table because the Completion Ratio is 8:7 which means we're missing a rule. As is often the case (and the primary reason for using a logic tool like LogicGem) we need help to find the missing rule. Fortunately, we can use a tool within LogicGem to come up with the rule. Select the Missing tool from the Logic menu, and the missing Rule 8 is determined. You can then proceed to enter in the appropriate actions. See Figure 12 for the completed decision table.

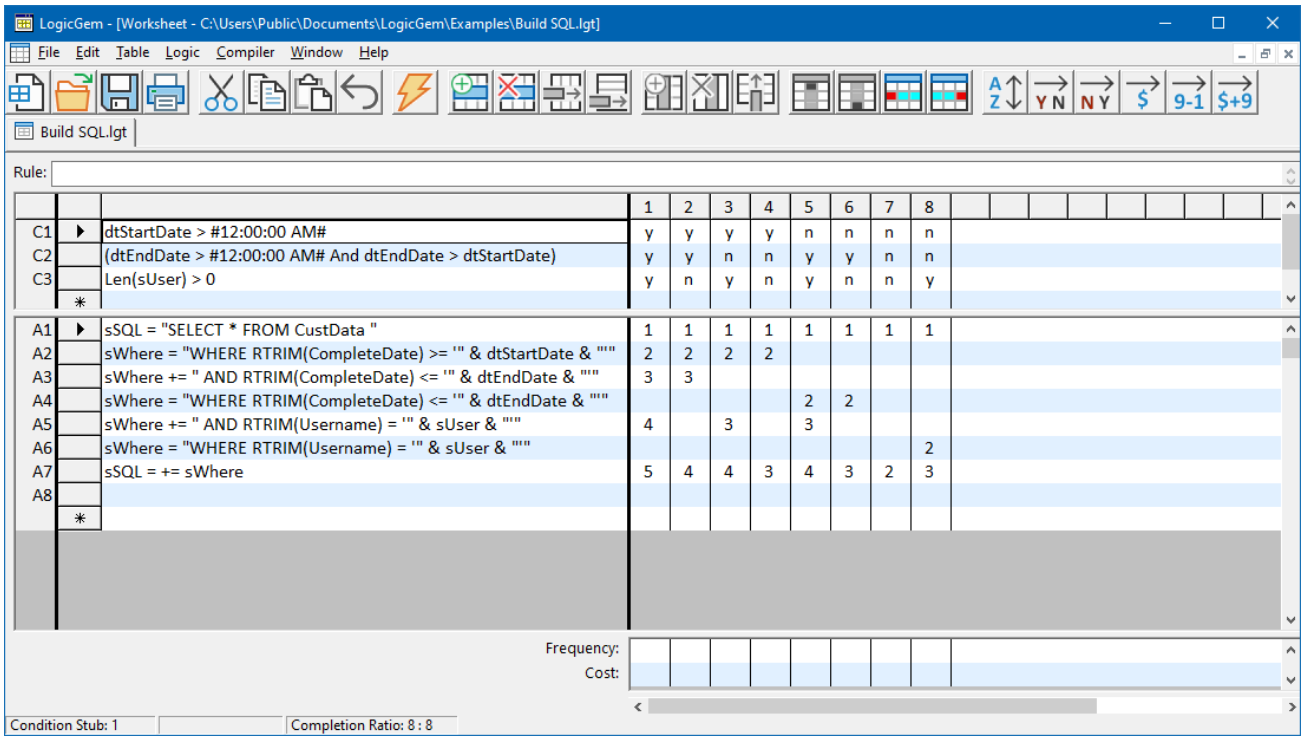

Figure 11

The last thing we need to do is to configure Logic Gem's logic compiler to convert the decision table to programming code. Select the Compiler | Language menu and choose "Visual Basic" and then selecting **Compiler** | **Structure** | **Nested If/Else**.

Now select **Compiler** | **Compile** to tell LogicGem to compile the decision table in order to yield programming code. See the Visual Basic code below. A programmer with reasonable talent could probably handle writing the code by hand, but if we add five more conditions and ten additional actions, the task becomes much more daunting. LogicGem can handle a higher degree of complexity than a human software engineer, all with a 100% level of completeness.

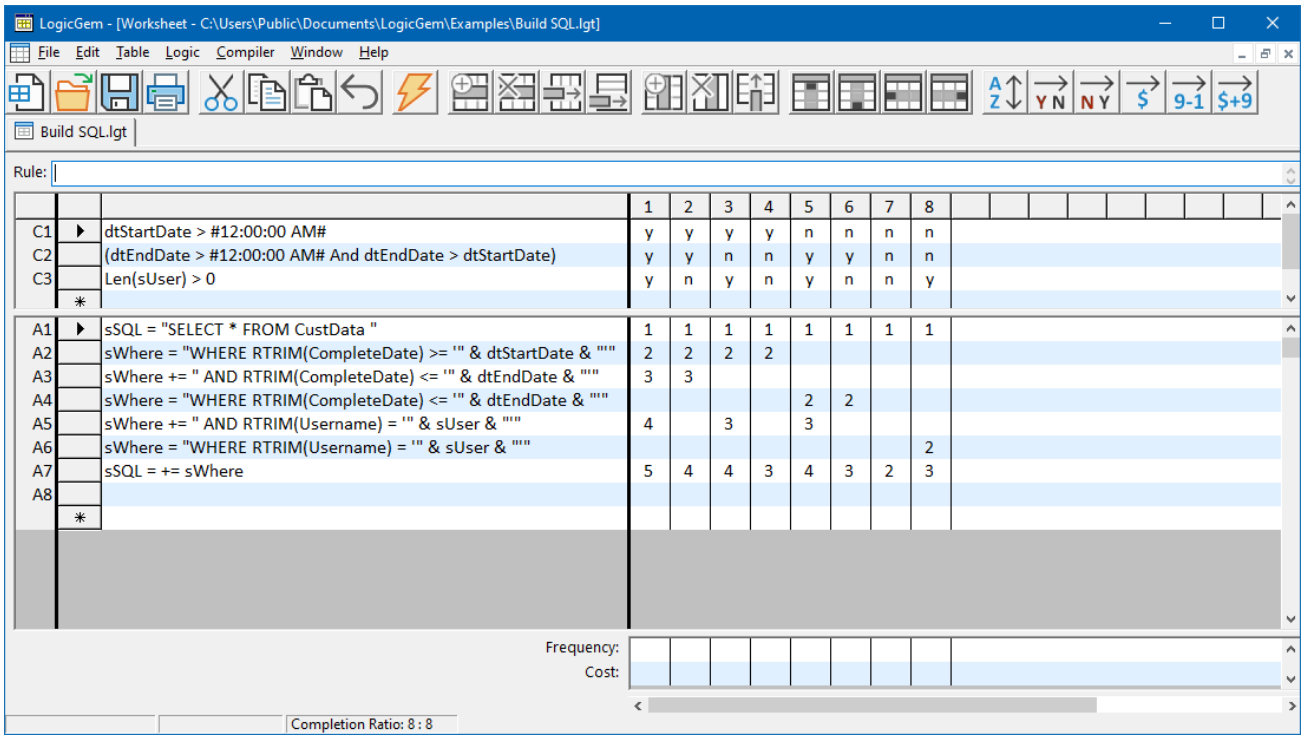

Figure 12

```
' build_sql_2.lgt

If (dtStartDate > #12:00:00 AM#) Then
    If ((dtEndDate > #12:00:00 AM# And dtEndDate > dtStartDate)) Then
        If (Len(sUser) > 0) Then
            ' rule 1
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sWhere += " AND RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        Else
            ' rule 2
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sWhere += " AND RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sSQL = += sWhere
        End If
```

```
    Else
        If (Len(sUser) > 0) Then
            ' rule 3
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sWhere += " AND RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        Else
            ' rule 4
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) >= '" & dtStartDate & "'"
            sSQL = += sWhere
        End If
    End If
Else
    If ((dtEndDate > #12:00:00 AM# And dtEndDate > dtStartDate)) Then
        If (Len(sUser) > 0) Then
            ' rule 5
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sWhere += " AND RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        Else
            ' rule 6
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(CompleteDate) <= '" & dtEndDate & "'"
            sSQL = += sWhere
        End If
    Else
        If Not (Len(sUser) > 0) Then
            ' rule 7
            sSQL = "SELECT * FROM CustData "
            sSQL = += sWhere
        Else
            ' rule 8
            sSQL = "SELECT * FROM CustData "
            sWhere = "WHERE RTRIM(Username) = '" & sUser & "'"
            sSQL = += sWhere
        End If
    End If
End If
```