

CBR Project Report

COMSCI 767 - INTELLIGENT AGENTS Assignment 1

A. Student

1234567 - astu001

Table of Contents

CBR Project Report	1
Table of Contents	2
Introduction	3
I. CBR logic behind the application.....	4
I.1 <i>Holiday Type</i>	5
I.2 <i>Price</i>	6
I.3 <i>Number of Person</i>	6
I.4 <i>Transportation</i>	7
I.5 <i>Duration</i>	7
I.6 <i>Season</i>	7
I.7 <i>Accommodation</i>	8
I.8 <i>Adaptation process</i>	8
I.9 <i>Feature Weights</i>	8
II. Design and implementation of the application.....	10
II.1 <i>SQL Database</i>	10
II.1.1 <i>ER Diagram</i>	10
II.1.2 <i>Table Definition</i>	10
II.1.3 <i>Procedures and Functions</i>	12
II.2 <i>Design and Implementation of application</i>	13
II.2.1 <i>CBRSetup</i>	13
II.2.2 <i>DAL</i>	14
II.2.3 <i>FrontEnd</i>	14
II.2.4 <i>Shared</i>	15
II.3 <i>User Manual</i>	15
Acknowledgements.....	19
References.....	19
Appendix.....	20

Introduction

This report is written to describe and further explain the logic behind the CBR project application that developed by the student. The application reflects on the Case Base Reasoning also known as CBR that is frequently discussed in COMSCI 767 class. Main reason of developing this application for me as a student is to gain understanding of why is CBR method used, what is CBR in software development, and how to apply it.

What is CBR? The most popular definition mentioned by Riesbeck and Schank is: "A case-based reasoner solves problems by using or adapting solutions to old problems." In other word, as described by the lecturer Dr Ian Watson, CBR is considered as one of problem solving methodology that does not use any smart specific technology but simply use old solution to similar problems that is stored in the system and apply that to solve the new one. There are four major activities that define a CBR: Retrieve old case that is similar to the new problem, Reuse the solution to that old case, Revise the solution to match the new problem, if required, and lastly Retain the new solution after it has been validated.

The application is built based upon the travel case base data, which has been previously provided and what it does is recommend user by returning a set of travel/ journey (from the given data) that similarly match the variables that user selected as restriction. The CBR logic behind the application is using k-NN or k-Nearest Neighbour algorithm. This algorithm is the most popular technology used in CBR because in theory it is relatively easy to understand. The core principle behind k-NN is comparing the similarity between the target problem and the source case. In order to that, we need to come up and justify the local similarity point for every variable that define the problem and compare them with what is in the case base data. Moreover, calculating the sum of all local similarity from each variables may not seems sufficient to distinguish global similarity because in real life one aspect of problem is more important that the others. For example, in terms of buying any product, price is obviously more important than other aspects such as colour or country where the product is made from. That is the reason why we also need to come up with weighting factor for each variable and take that into account when calculating the overall similarity point.

This report explains more in detail about the CBR logic (local and global similarity point calculation) in relation to the travel recommender application in chapter 1 and the overall design and implementation of the application in chapter 2. Lastly a brief user manual is provided in chapter 3.

I. CBR logic behind the application

By looking at the sample data provided from the travel case-base excel file below, we can see that there are 11 variables those can be used to calculate similarity point starting from case name, journey code, and all the way down to hotel name.

985

objects

case	Journey985
JourneyCode:	985
HolidayType:	Wandering,
Price:	1672
NumberOfPersons:	3
Region:	Thuringia,
Transportation:	Car,
Duration:	7
Season:	December,
Accommodation:	ThreeStars,
Hotel:	Berghotel Friedrichroda, Thuringia.

The data provided are not all useful and the first goal here is to extract variables those are comparable or at least meaningful and to exclude the ones that are unnecessary or vague. Variables such as case and JourneyCode are obviously redundant and only used perhaps as an index so we can confidently exclude those out.

Other data such as Region and Hotel may be arguably useful to be included as variables but the problem here is we do not have more information or at least a justification that we can use as a comparison base. For example, we do not know whether BlackForest region match with Harz or we also do not know if Sporthotel in Erz Gebirge supposedly better than Aparthotel in Harz since the two of them are rated as three stars accommodation.

Given the above situation, for the purpose of this application, we are going to use only these variables for calculation: holiday type, price, number of persons, transportation, duration, season and accommodation

I.1 Holiday Type

There are 9 holiday types found in travel case-base file:

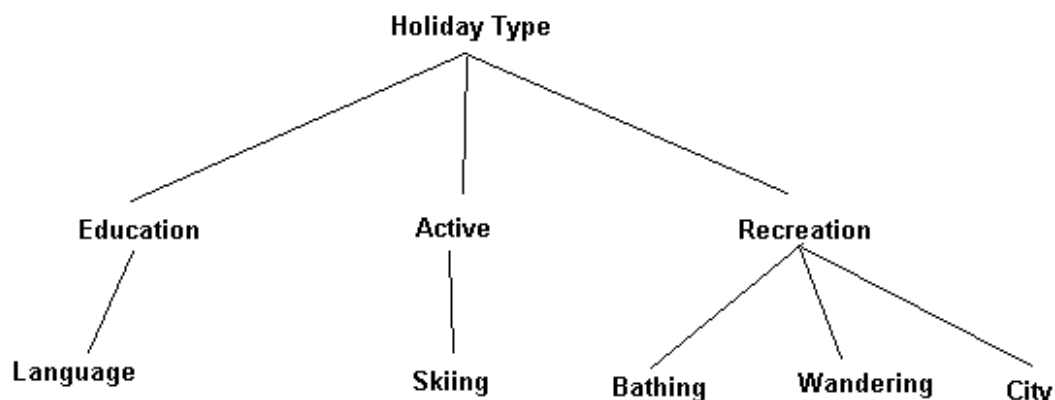
- Active
- Bathing
- City
- Education
- Language
- Recreation
- Skiing
- Wandering

In order to calculate the local similarity, I classified holiday types into 3 categories: active, education, and recreation. Active contains skiing; education contains language; and recreation contains bathing, city, and wandering.

Using the above categories, I made up the local similarity metrics using taxonomy or also called hierarchical tree. If input holiday type equals to journey's type then it will result in perfect score of 1. Moreover, if input holiday type and journey type falls under one category, it will return default score of 0.5 (value depends on parameter in database called 'TYPE_IN_ONE_CATEGORY')

In addition, I also appoint holiday type as the most important variable and it is used as a base to restrict the travel case base when it comes to the calculation. For example, if user inputs education as holiday type, the logic will search through and perform computation to entries which type is either education or language. The rest of travel case will be ignored.

The above assumption comes from experience and observation: before people going for travel, they always have something in mind what they are going to do, what they want to achieve and purpose of their travel. Due to that observation, holiday type then must always contain an input value from the user unlike the rest of the variables.



I.2 Price

Price is rather a complex variable because it has relation to some other variables such as number of person and duration. A 3 days travel for 2 people at \$500 is considered more expensive than a week holiday for a family at \$1000. However, this issue will further be discussed at the adaptation process on chapter I.8

Talk about price as an individual variable, there is always an assumption that less is better! Given that assumption, it does not mean that less price is ALWAYS better, there is also another assumption that must be taken into account: price never lies, for example most people will question the authenticity of a holiday ad that says a week holiday for two for the price of \$50 and that is because it is too cheap.

Due to that reason, the logic to calculate local similarity point of price is constructed using the following rules:

- If journey price is within the range of 75% (value depends on parameter in database called 'MINIMUM_PERFECT_PRICE_RANGE') to 100% of input price, then the return score will be 1
- If journey price is within the range of 0 to 75% of input price, the result will be generated using formula: $\text{difference}(\text{input price} - \text{journey price}) / (\text{price range from 0 to 75\% of input price})$
- If journey price is within the range of 100% to 150% (value depends on parameter in database called 'MAXIMUM_PRICE_RANGE') of input price, the result will be generated using formula: $\text{difference}(\text{journey price} - \text{input price}) / (\text{price range from 100\% to 150\% of input price})$
- Else, if journey price is outside the 150% range of unit price, it will return 0

I.3 Number of Person

Working out the local similarity point by comparing two numeric values is the easiest. For the simplicity of the application, we can safely use the formula below to get the result:

Similarity point = $\frac{\text{*range} - \text{difference between journey and input}}{\text{*range}}$

*range = default at 20 (value depends on parameter in database called 'PERSONS_MAX')

However, the equation above is not bullet proof. There might be a situation where the distance between journey and input is greater than the range. At this point it will return negative value which is a bias and outside the Euclidean distance.

I.4 Transportation

There are 4 means of transportation recorded in the data file:

- Car
- Coach
- Plane
- Train

Local similarity point for this variable must be pre-defined and stored in a decision table. The values are constructed based on assumption and they are considered as asymmetric which means the role of input and output is important, e.g. Sim(car as input, train as output) may not be equal to Sim(train as input, car as output)

Below is the decision table constructed for transportation: (all the values depend on parameters in database, e.g. 'TRANSPORT_CAR_TRAIN' defines the similarity point value of car as input and train as output)

IN / OUT	Car	Coach	Plane	Train
Car	1	0.66	0.33	0.66
Coach	0.33	1	0.33	0.66
Plane	0	0	1	0
Train	0.33	0.66	0.33	1

I.5 Duration

Similar to number of persons, we can use the formula below to get the local similarity point of the duration:

Similarity point = $\frac{\text{*range} - \text{difference between journey and input}}{\text{*range}}$

*range = default at 30 (value depends on parameter in database called 'DURATION_MAX')

I.6 Season

The values are range from January to December. In order to easily calculate the local similarity values, we need to do the following:

1. Map the month of the year to numeric value, e.g. January to 1, May to 5
2. Properly calculate the difference between input season and journey case season, e.g. the gap between 3 (March) and 11 (November) should be 4 instead of 8.

Once numeric difference has been established, we can safely use the common formula to calculate similarity point from two numbers:

Similarity point = $\frac{\text{*range} - \text{difference between journey and input}}{\text{*range}}$

*range = default at 12 (value depends on parameter in database called 'SEASON_RANGE')

1.7 Accommodation

Accommodation values are ranged from holiday flat to five star accommodations. Given that values, we may assume that higher rated accommodation is always better. So in order to calculate local similarity point of accommodation, following rules are applied:

1. Map the rating to numeric value: holiday flat = 1; one star = 2, two star = 3 and so on.
2. If journey rating is greater than or equal to input then returns perfect score of 1
3. Else get the return value by using formula: $\text{range} - \text{difference} / \text{range}$

1.8 Adaptation process

The adaptation process calculates similarity point from the combination of multiple variables. In this application, the adaptation logic is trigger when user inputs values for either the combination of price and duration, price and number of persons, or both price, duration, and number of persons.

The logic is described as follows:

1. If input duration contains value, set local similarity point of duration to 1
2. Update the journey price by formula: $\text{input duration} / \text{journey duration} * \text{journey price}$
3. If input number of persons contains value, set local similarity point of number of persons to 1
4. Update the journey price by formula: $\text{input num of persons} / \text{journey num of persons} * \text{journey price}$
5. Calculate the final journey price by adding surcharge price of \$100 (value depends on parameter in database called 'EXTRA_PRICE_COST')
6. Get the local similarity point of updated price by applying rules described in chapter 1.2
7. Sum local similarity points from price, duration, and number of persons.

1.9 Feature Weights

Weighting factor for each variables are constructed based on common sense assumption and result of observation through system test running the application. It is also based on other factors that are mentioned on the previous chapters such as the importance of holiday type and adaptation process where price is clearly more important than duration or number of persons.

Below is the weighting factors for each variable:

Variable	Parameter name in DB	Value
Holiday type	WEIGHT_TYPE	5
Price	WEIGHT_PRICE	3
Number of persons	WEIGHT_PERSONS	1

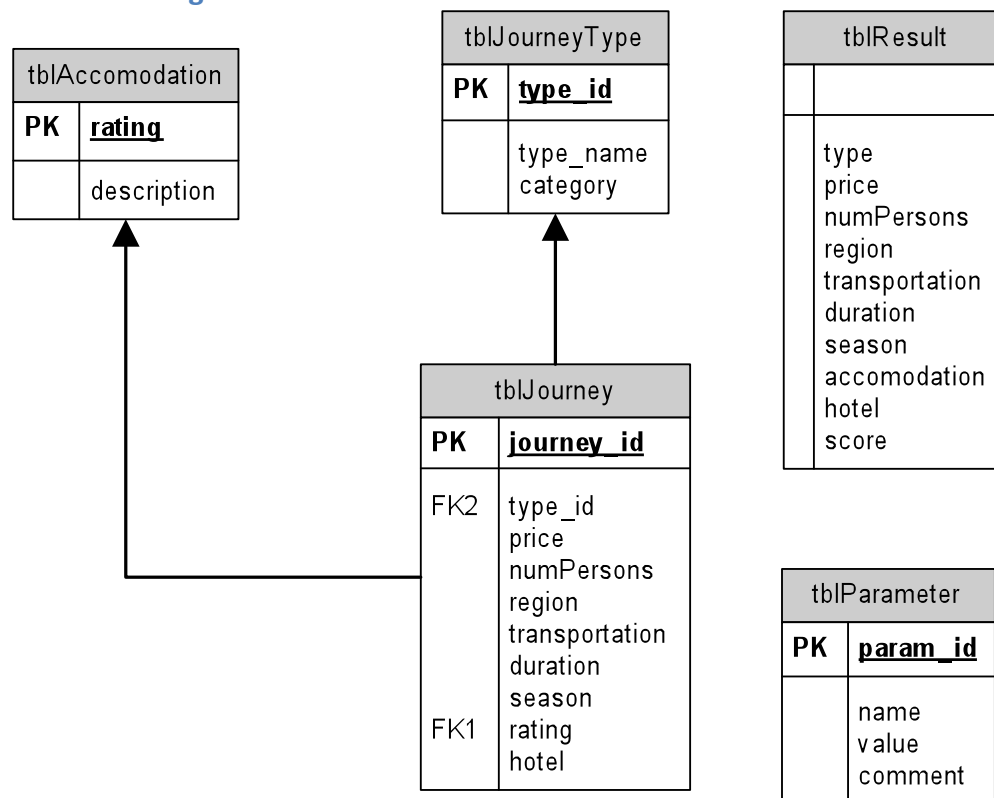
Transportation	WEIGHT_TRANSPORT	2
Duration	WEIGHT_DURATION	1
Season	WEIGHT_SEASON	2
Accommodation	WEIGHT_ACCOMODATION	1

II. Design and implementation of the application

The overall application is developed under ASP .Net 2.0 using C# and MS SQL Server 2005 as the database. Most of the logic especially the k-NN algorithm is written in SQL using procedures and functions. More details about the database structure and methods will be described in chapter II.1. The application side is only used to handle user inputs and display the result as a set of travel recommendations. Application design and implementation will be described in chapter II.2. Finally a brief user manual on how to install and operate the application is described in chapter II.3

II.1 SQL Database

II.1.1 ER Diagram



II.1.2 Table Definition

tblJourney

Stores all journey data that has been processed from external file

List of attributes:

- *journey_id: int (PK)*
- *type_id: int (FK references tblJourneyType.type_id)*
- *price: decimal(8,2)*
- *numPersons: int*
- *region: nvarchar(50)*
- *transportation: nvarchar(50)*
- *duration: int*
- *season: int*
- *rating: int (FK references tblAccomodation.rating)*
- *hotel: nvarchar(255)*

Sample entry: 2, 8, 3066.00, 3, 'Egypt', 'Plane', 14, 4, 3, 'Hotel White House'

tblJourneyType

Stores journey types with their categories

List of attributes:

- *type_id: int (PK)*
- *type_name: nvarchar(50)*
- *category: int*

See appendix for entries stored in tblJourneyType

tblAccomodation

Stores rating and its description

List of attributes:

- *rating: int (PK)*
- *description: nvarchar(50)*

See appendix for entries stored in tblAccomodation

tblParameter

Stores all parameters used to perform the calculation, purpose of having this table is giving an ability to user to modify the value more easily.

List of attributes:

- *param_id: int (PK)*
- *name: nvarchar(50)*
- *value: nvarchar(255)*
- *comment: nvarchar(255)*

See appendix for list of available parameters stored in tblParameter.

tblResult

Stores travel result and global similarity point after computation has finished. May also be considered as a View and very similar to tblJourney. Data from this table is sent back to main application for display purpose.

List of attributes:

- *type: nvarchar(50)*
- *price: decimal(8,2)*
- *numPersons: int*
- *region: nvarchar(50)*
- *transportation: nvarchar(50)*
- *duration: int*
- *season: int*
- *accommodation: nvarchar(50)*
- *hotel: nvarchar(50)*
- *score: decimal(5,2)*

Sample entry: 'Bathing', 2498.00, 2, 'Egypt', 'Plane', 14, 4 'TwoStars', 'Hotel White House', 7.05

II.1.3 Procedures and Functions

sp_tblResult_GenerateResult

This is the main procedure that gets called and takes input variables such as holiday type, price, and season from the application.

It populates data to tblResult, calls fn_CalculateOverallScore function, and return the result set back to main application.

fn_CalculateOverallScore

This function is called by sp_tblResult_GenerateResult. General purpose of this function is to sum up local similarity point that are calculated by other sub-functions, weight them, and return the overall similarity point for each case.

fn_CalculateType

Called by fn_CalculateOverallScore to compute local similarity point between input and case holiday type. Main logic of this function is described previously on chapter I.1

fn_CalculatePrice

Called by fn_CalculateOverallScore to compute local similarity point between input and case price. Main logic of this function is described previously on chapter I.2

fn_CalculateNumOfPersons

Called by fn_CalculateOverallScore to compute local similarity point between input and case number of persons. Main logic of this function is described previously on chapter I.3

fn_CalculateTransportation

Called by fn_CalculateOverallScore to compute local similarity point between input and case transportation. Main logic of this function is described previously on chapter I.4

fn_CalculateDuration

Called by fn_CalculateOverallScore to compute local similarity point between input and case duration. Main logic of this function is described previously on chapter I.5

fn_CalculateSeason

Called by fn_CalculateOverallScore to compute local similarity point between input and case accommodation. Main logic of this function is described previously on chapter I.6

fn_CalculateAccomodation

Called by fn_CalculateOverallScore to compute local similarity point between input and case accommodation. Main logic of this function is described previously on chapter I.7

fn_CalculatePriceDurationPersons

Called by fn_CalculateOverallScore when user input the combination of price and duration, price and num of persons, or both price, duration, and num of persons as part of the adaptation process. The logic of this function is described in chapter I.8

II.2 Design and Implementation of application

The main application contains four separate projects as described below:

II.2.1 CBRSetup

Setup project provided as a template by Visual Studio 2005. It generates an installation file for the main application.

II.2.2 DAL

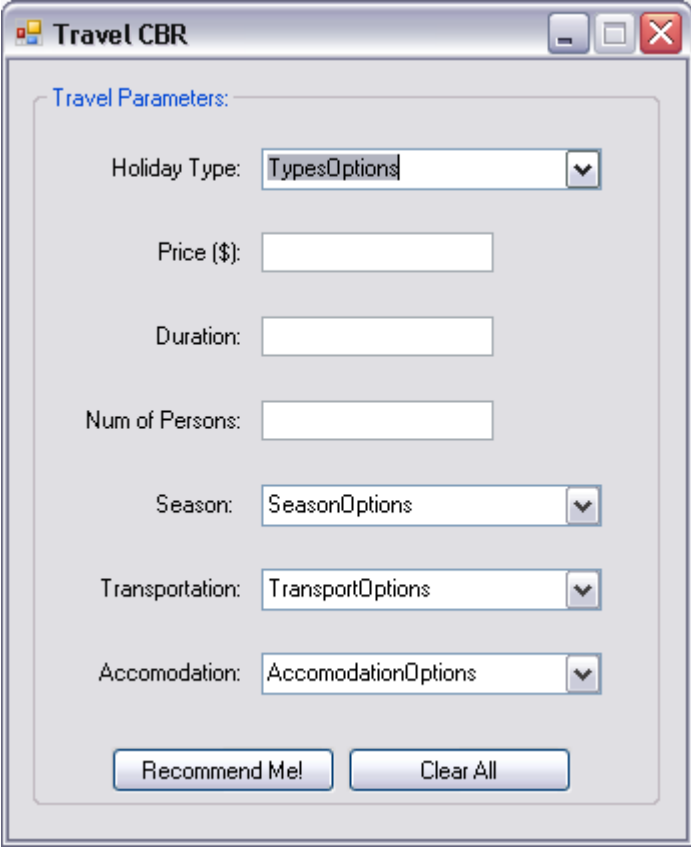
Also called the Data Access Layer, this project creates connection to the database. Static method *GenerateResult_DataSet()* in *tblResult* class is used by the application to call *sp_tblResult_GenerateResult* procedure in SQL.

Moreover, as a note, DAL project is automatically generated using a 3rd party application called Crest Development Toolkit that was developed by my development team where I used to work for Crest Technologies Ltd.

II.2.3 FrontEnd

This project is the GUI layer. There are 2 WinForms used in this application

- **Main Form:** handle user inputs and perform validation to those inputs

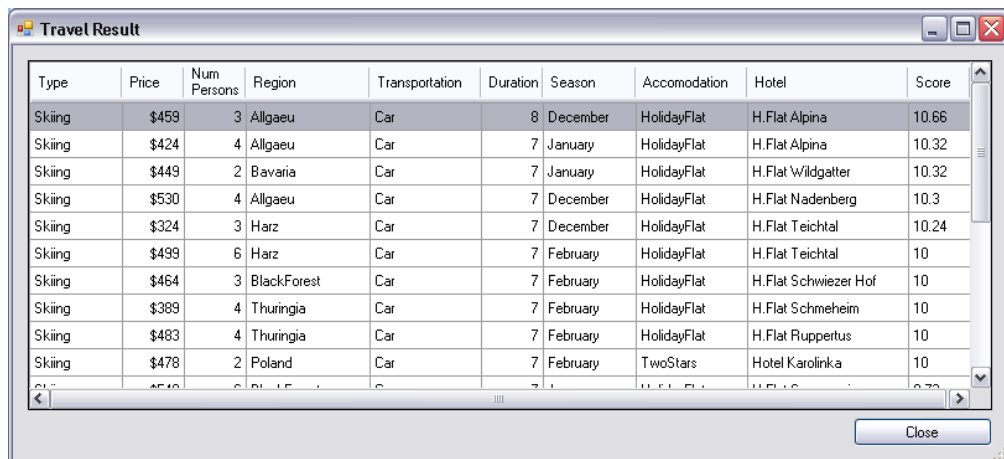


The screenshot shows a Windows Form titled "Travel CBR". The form contains a section labeled "Travel Parameters:" with the following controls:

- Holiday Type: A dropdown menu with "TypesOptions" selected.
- Price (\$): A text input field.
- Duration: A text input field.
- Num of Persons: A text input field.
- Season: A dropdown menu with "SeasonOptions" selected.
- Transportation: A dropdown menu with "TransportOptions" selected.
- Accommodation: A dropdown menu with "AccommodationOptions" selected.

At the bottom of the form, there are two buttons: "Recommend Me!" and "Clear All".

- **Result Form:** display recommended journeys returned by the database



Type	Price	Num Persons	Region	Transportation	Duration	Season	Accommodation	Hotel	Score
Skiing	\$459	3	Allgaeu	Car	8	December	HolidayFlat	H.Flat Alpina	10.66
Skiing	\$424	4	Allgaeu	Car	7	January	HolidayFlat	H.Flat Alpina	10.32
Skiing	\$449	2	Bavaria	Car	7	January	HolidayFlat	H.Flat Wildgatter	10.32
Skiing	\$530	4	Allgaeu	Car	7	December	HolidayFlat	H.Flat Nadenberg	10.3
Skiing	\$324	3	Harz	Car	7	December	HolidayFlat	H.Flat Teichtal	10.24
Skiing	\$499	6	Harz	Car	7	February	HolidayFlat	H.Flat Teichtal	10
Skiing	\$464	3	BlackForest	Car	7	February	HolidayFlat	H.Flat Schweizer Hof	10
Skiing	\$389	4	Thuringia	Car	7	February	HolidayFlat	H.Flat Schmeheim	10
Skiing	\$483	4	Thuringia	Car	7	February	HolidayFlat	H.Flat Ruppertus	10
Skiing	\$478	2	Poland	Car	7	February	TwoStars	Hotel Karolinka	10

II.2.4 Shared

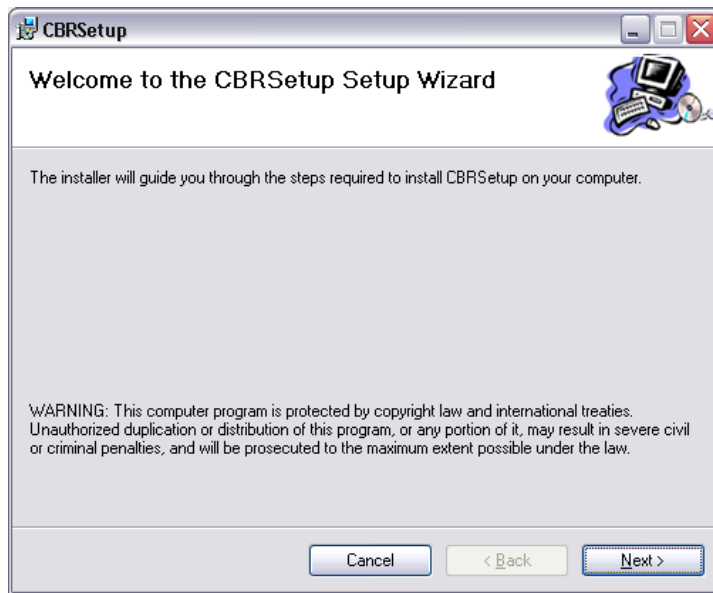
This project acts as the logic layer that connects the DAL and FrontEnd. It contains three classes as described below:

- **Enums**
Static class that represents enumeration, e.g. January = 1, OneStar accommodation = 2
- **Journey**
This class represents journey object and its properties.
- **Model**
Static class with one public method GenerateResult() that takes input variables passed in by the FrontEnd and calls DAL.GenerateResult_DataSet(). A dataset returned is processed as a list of journey objects and passed back to FrontEnd to be displayed on result form

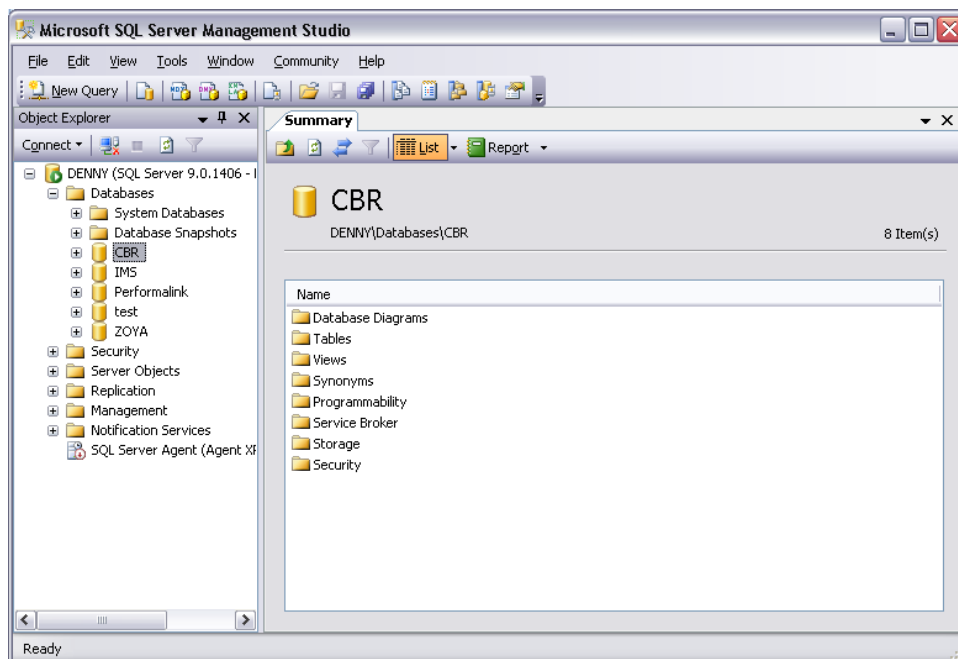
II.3 User Manual

Installation

1. Double click on CBRSetup.msi file



2. Follow the installation steps
3. The application will usually be installed under: **'C:\Program Files\Denny Riadi\CBRSetup\FrontEnd.exe'**
4. Open MS SQL Server 2005/ 2008 and create new database called CBR or any other name as shown below



5. Restore the CBR database with the provided backup file **'CBR.bak'**

6. Once application and database have successfully setup, open the application's configuration file located on 'C:\Program Files\Denny Riadi\CBRSetup\FrontEnd.exe.config' using any text editor or Visual Studio
7. Look at the following lines in the config file and change the Data Source parameter to match your server name and Initial Catalog to match your database name created in step 4


```
<add key="DataAccessLayerDSN" value="Integrated
Security=SSPI;Persist Security Info=True;Data
Source=DENNY;Initial Catalog=CBR;" />

<add key="ExceptionTrackerDSN" value="Integrated
Security=SSPI;Persist Security Info=True;Data
Source=DENNY;Initial Catalog=CBR;" />
```
8. Save the updated config file.
9. Run the executable file and main form will appear on your screen.

Running the application

1. The application has straightforward process. First of all, user needs to fill the input parameters. Example is shown below:

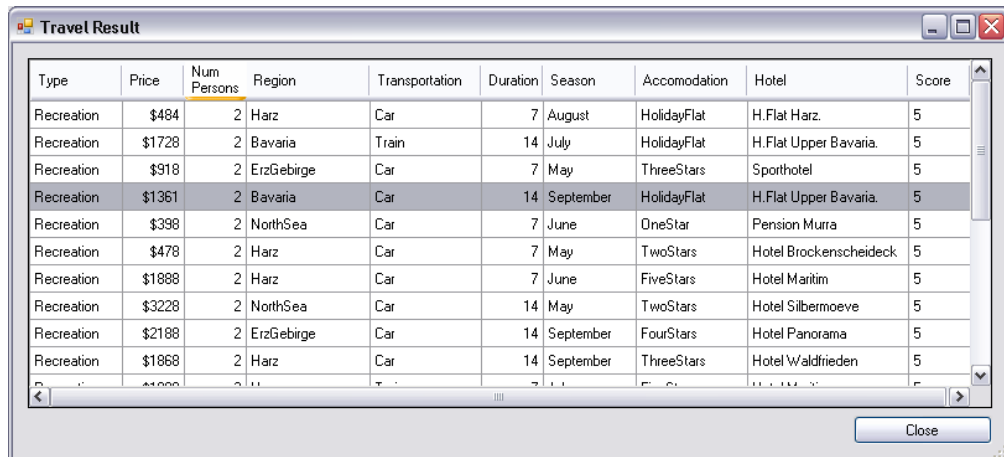
The screenshot shows a window titled "Travel CBR" with a "Travel Parameters:" section. The parameters are as follows:

Parameter	Value
Holiday Type	Wandering
Price (\$)	
Duration	5
Num of Persons	
Season	February
Transportation	TransportOptions
Accommodation	OneStar

At the bottom of the form, there are two buttons: "Recommend Me!" and "Clear All".

2. User clicks on 'Recommend Me!' button and it will bring up the result form

- Alternatively, user can click on 'Clear All' button to reset the parameters.
- In Result form, user can click on one of the column header to sort the list based on the column.



Type	Price	Num Persons	Region	Transportation	Duration	Season	Accomodation	Hotel	Score
Recreation	\$484	2	Harz	Car	7	August	HolidayFlat	H.Flat Harz.	5
Recreation	\$1728	2	Bavaria	Train	14	July	HolidayFlat	H.Flat Upper Bavaria.	5
Recreation	\$918	2	ErzGebirge	Car	7	May	ThreeStars	Spothotel	5
Recreation	\$1361	2	Bavaria	Car	14	September	HolidayFlat	H.Flat Upper Bavaria.	5
Recreation	\$398	2	NorthSea	Car	7	June	OneStar	Pension Murra	5
Recreation	\$478	2	Harz	Car	7	May	TwoStars	Hotel Brockenscheideck	5
Recreation	\$1888	2	Harz	Car	7	June	FiveStars	Hotel Maritim	5
Recreation	\$3228	2	NorthSea	Car	14	May	TwoStars	Hotel Silbermoeve	5
Recreation	\$2188	2	ErzGebirge	Car	14	September	FourStars	Hotel Panorama	5
Recreation	\$1868	2	Harz	Car	14	September	ThreeStars	Hotel Waldfrieden	5

Acknowledgements

The application is partly built using 3rd party software called Crest Development Toolkit. Credit and thank you goes to Crest Technologies Ltd for developing the tool.

References

Aamodt, A., Plaza, E. Case-based reasoning: foundational issues methodological variations, and system approaches. *AI Communications* 7 (1994) 39-59

Riesbeck, C.K., Schank, R. *Inside Case-based Reasoning*. Erlbaum. Northvale, NJ. 1989.

Watson, I. *Applying Case-based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, CA, USA. 1997

Watson, I. Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems* 12 (1999) 303-308

Appendix

tblAccomodation

Rating	Description
1	HolidayFlat
2	OneStar
3	TwoStars
4	ThreeStars
5	FourStars
6	FiveStars

tblJourneyType

ID	Type Name	Category
1	Education	1
2	City	2
3	Skiing	3
4	Recreation	2
5	Wandering	2
6	Language	1
7	Active	3
8	Bathing	2

tblParameter

ID	Name	Value	Comments
1	WEIGHT_TYPE	5	Weight of holiday type
2	WEIGHT_PRICE	3	Weight of price
3	WEIGHT_DURATION	1	Weight of duration
4	WEIGHT_PERSONS	1	Weight of number of persons
5	WEIGHT_SEASON	2	Weight of season
6	WEIGHT_TRANSPORT	2	Weight of transportation
7	WEIGHT_ACCOMODATION	1	Weight of accomodation
8	SEASON_RANGE	12	Season range from Jan to Dec
9	DURATION_MAX	30	Duration range
10	PERSONS_MAX	20	Number of persons range
11	TRANSPORT_TRAIN_PLANE	0.33	train as input, plane as output
12	TRANSPORT_TRAIN_CAR	0.33	train as input, car as output
13	TRANSPORT_TRAIN_COACH	0.66	train as input, coach as output
14	TRANSPORT_PLANE_TRAIN	0	plane as input, train as output
15	TRANSPORT_PLANE_CAR	0	plane as input, car as output
16	TRANSPORT_PLANE_COACH	0	plane as input, coach as

			output
17	TRANSPORT_CAR_TRAIN	0.66	car as input, train as output
18	TRANSPORT_CAR_PLANE	0.33	car as input, plane as output
19	TRANSPORT_CAR_COACH	0.66	car as input, coach as output
20	TRANSPORT_COACH_TRAIN	0.66	coach as input, train as output
21	TRANSPORT_COACH_PLANE	0.33	coach as input, plane as output
22	TRANSPORT_COACH_CAR	0.33	coach as input, car as output
23	MINIMUM_PERFECT_PRICE_RANGE	0.75	minimum percentage price range to get perfect score
24	EXTRA_PRICE_COST	100.00	surcharge cost, part of adaptation process
25	MAXIMUM_PRICE_RANGE	1.5	max price range, value above this range will return 0
26	TYPE_IN_ONE_CATEGORY	0.5	local similarity point when input and journey holiday type is in same category
27	RESULT_SET	20	number of recommendation returned