Contents

Change log (class-library)

A. Models
   a. Item class renamed to ***Model*** and ItemJoin class renamed to ***ModelJoin***
   b. Each model have attribute specifying the table name and primary key(s)
   c. Models attributes now have the equivalent type of the ones in the database. Not all are strings, there are DateTime, int, bool, decimal, etc… And the value types that allow null in the database are marked as Nullable in C#.

B. ***DataList*** class
   a. Abstract class now, only could be inherited not instantiated.
   b. Accept generic type of ***Model*** (T) instead of the Item, all other subclasses follow the same.
   c. Does not take the table name, primary key from the constructor. But will retrieve them from the passed generic type attribute. (See A.b)
   d. Populate(Item item) method have been renamed to Fill
   e. **Add** method have been rewritten and implemented ***InsertClause*** class which parse any given model into the insert clause with the SqlParamters. It does support checks for auto-increment fields, does check for null and handle types appropriately.
   f. **Update** method have been rewritten and implemented ***SetClause*** class which parse any given model into the set clause with the SqlParamters. It does check for non-updatable fields like the primary keys. In addition, it does handle both Model and ***ModelJoin*** without the need to override this method in the ***DataListJoin***.
   g. **Delete** method have been rewritten and handle both ***Model*** and ***ModelJoin*** without the need to override this method in the ***DataListJoin***.
   h. All methods that use SqlDataReader, SqlCommand are using that object for one time only and disposed probably by the using (..) {} statement. Connection is opened/closed as required with some check to avoid exceptions [**OpenConnection()** + **CloseConnection()**]
   i. All methods that calls the database have try { } catch {} finally {} block to handle any possible exception.
   j. Methods that should populate the current list return Boolean to indicate if there is any matching results. Same goes for methods that mutate single rows. While other methods that mutate multiple rows return the number of affected rows.

C. ***DataListJoin*** class

    a. Have less repeated code than the original one (Populate, Update & Delete) and reuse the base methods.

D. ***WhereClause*** class is implemented which use builder design pattern to help passing dynamic conditions to the Populate and Filter methods in the DataList. This could build any WHERE statement and passing only the new object instance will be sufficient instead of writing end-less number of parameters in the methods. (See Appendix B)

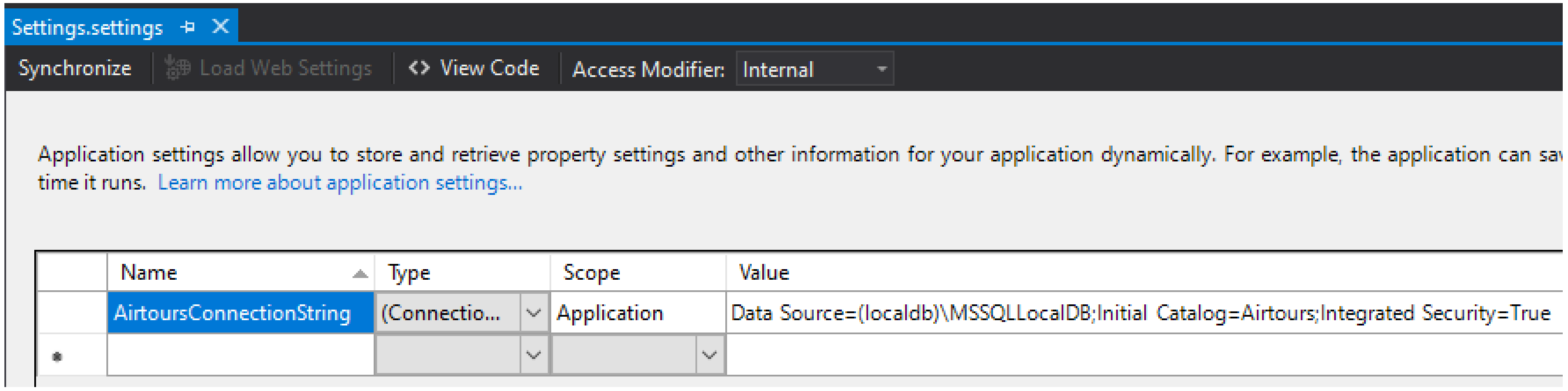
E. Database connection string is saved under Properties -> Settings.settings with the key name AirtoursConnectionString. (See Appendix A)

Web application functionality

- Register
- Login: Done using FormsAuthentication and Session
- Browse for Outward and Return flights: Have been merged into one page, the customer could enter the origin and destination along with the departure date and return date (optional) which will show matching results without the need for the customer to re-enter the detais in an another form. The customer could skip picking return flight too.
- Make a Reservation and Add Passenger details: Have been merged into one page, the customer can add passenger details before confirming the reservation.
- View future reservation: Have been implemented using an extra method, which will lookup any possible future reservation based on the Flight Date of either the outward or the return flights.
- Edit future reservation: Have been implemented to change the passengers and the class level of the seats (Business or economy). This will be enabled only if the outward flight and the return flight have not occurred yet (After the outward flight it is not possible to modify the reservation)
- Delete future reservation: Have been implemented where if the outward flight and the return flight have not occurred yet, the customer could delete the reservation along with all details from the database, with a confirmation message.

Appendices

Appendix A

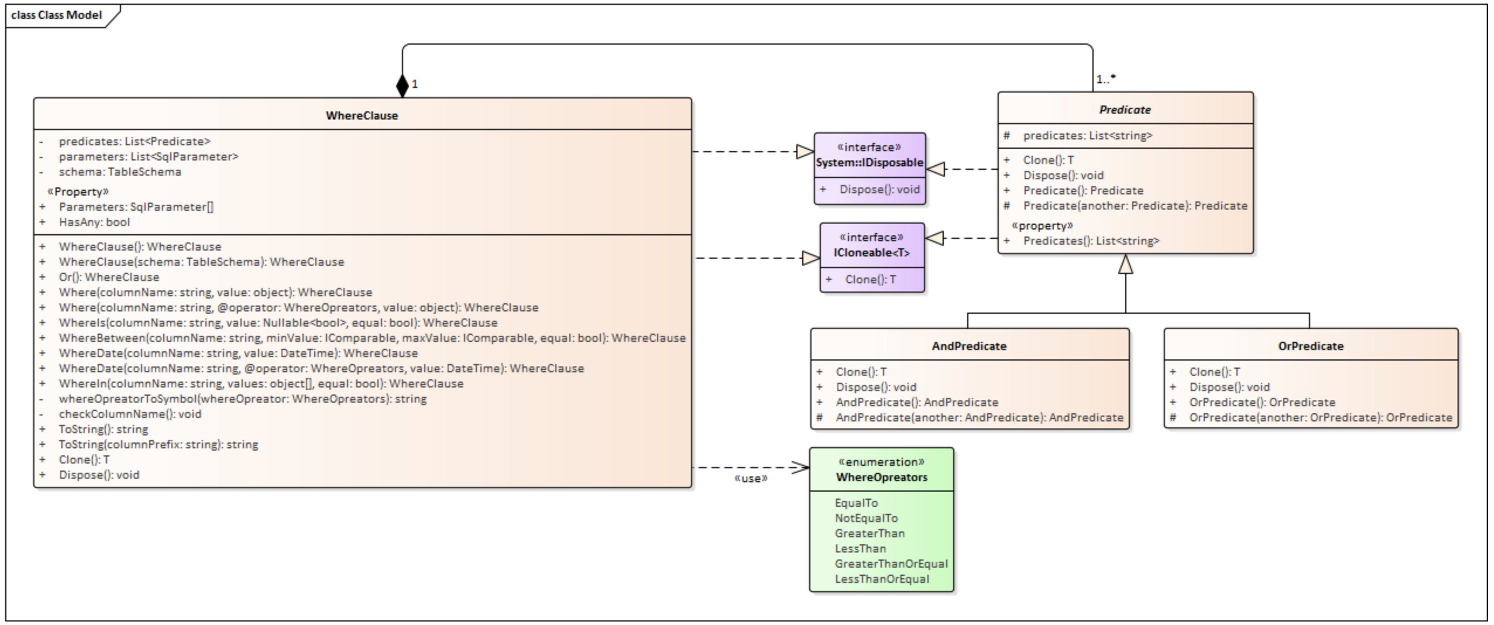


Access Modifier: Internal

Application settings allow you to store and retrieve property settings and other information for your application dynamically. For example, the application can sav
time it runs. Learn more about application settings...

	Name	Type	Scope	Value
	AirtoursConnectionString	(Connectio...	Application	Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Airtours;Integrated Security=True
*				

Appendix B


class Class Model
1
1..*
WhereClause
- predicates: List<Predicate>
- parameters: List<SqlParameter>
- schema: TableSchema
«Property»
+ Parameters: SqlParameter[]
+ HasAny: bool
+ WhereClause(): WhereClause
+ WhereClause(schema: TableSchema): WhereClause
+ Or(): WhereClause
+ Where(columnName: string, value: object): WhereClause
+ Where(columnName: string, @operator: WhereOpreators, value: object): WhereClause
+ WhereIs(columnName: string, value: Nullable<bool>, equal: bool): WhereClause
+ WhereBetween(columnName: string, minValue: IComparable, maxValue: IComparable, equal: bool): WhereClause
+ WhereDate(columnName: string, value: DateTime): WhereClause
+ WhereDate(columnName: string, @operator: WhereOpreators, value: DateTime): WhereClause
+ WhereIn(columnName: string, values: object[], equal: bool): WhereClause
- whereOpreatorToSymbol(whereOpreator: WhereOpreators): string
- checkColumnName(): void
+ ToString(): string
+ ToString(columnPrefix: string): string
+ Clone(): T
+ Dispose(): void
«interface»
System::IDisposable
+ Dispose(): void
«interface»
ICloneable<T>
+ Clone(): T
Predicate
# predicates: List<string>
+ Clone(): T
+ Dispose(): void
+ Predicate(): Predicate
# Predicate(another: Predicate): Predicate
«property»
+ Predicates(): List<string>
AndPredicate
+ Clone(): T
+ Dispose(): void
+ AndPredicate(): AndPredicate
# AndPredicate(another: AndPredicate): AndPredicate
OrPredicate
+ Clone(): T
+ Dispose(): void
+ OrPredicate(): OrPredicate
# OrPredicate(another: OrPredicate): OrPredicate
«use»
«enumeration»
WhereOpreators
EqualTo
NotEqualTo
GreaterThan
LessThan
GreaterThanOrEqual
LessThanOrEqual

Example of usage:

```
var passengerList = new PassengerList();

using (var where = new WhereClause()) {
    where.Where("FirstName", "Ahmed").Or().Where("LastName", "Naser");

    string whereClause = where.ToString(); // WHERE [FirstName] = @FirstName OR [LastName] = @LastName
    SqlParameter[] whereParm = where.Parameters; // Length of 2, with two SqlParamter that could be added to the SqlCommand (handled in the
DataList class)

    passengerList.Filter(where); // The DataList will be able to handle WhereClause and get the query string along with the parameters as above ^
}
```