|  |
| --- |
| **Vehicles Sales Report**  **Capabilities tested:**   * CDCP003: Work with C# programming constructs to create C# programs * CDCP006: Define and understand object oriented design principles * CDCP007: Design Dotnet classes which adhere to the object oriented principles * CDCP012: Use exception handling features to catch runtime exceptions * CDCP013: Use custom exception handling to catch runtime exceptions * CDCP015: Use basic collection class to handle business objects * CDCP016: Use appropriate collection classes to handle group of business objects * CDET004: Write unit tests using Nunit framework to test business classes of an application * CDCP020:Define relational database to build database for an application * CDCP021:Write DDL & DML SQL statements to query the database * CDAD001: Use ADO.NET features to connect to database and perform a simple query operation * CDCP022: Create stored procedure and function in a database to group complex queries * CDCP023: Write group by, joins and index based queries to query the database * CDAD002: Build persistence layer using ADO.NET for an enterprise application to establish connection with DB and work with DML commands/stored procs * CDCP018 : Perform file handling using appropriate .net classes * CDCP009 : Work with windows forms to create desktop applications |

**Problem Statement**

Design and implement an application program to automate the process of adding sales details of cars. The application stores a collection of sales done for different car models and provides mechanism of displaying reports on total number of each car models sold.

Your application program should follow the control flow as, start from class **VehiclesSalesReportClient-> SalesReportManager->VehiclesSalesReportDAO class -> Database**

Following projects are created in the solution project **“VehiclesSalesReport”**:

* VehiclesSalesReportDAL
* VehiclesSalesReportManager
* VehiclesSalesReportEntities
* VehiclesSalesReportClient
* VehiclesSalesReportExceptions
* VehiclesSalesReportNUnit

**Explanation:**

1. The main form “**AddSalesDetails**” should be displayed as below when **“VehiclesSalesReportClient”** application is run:

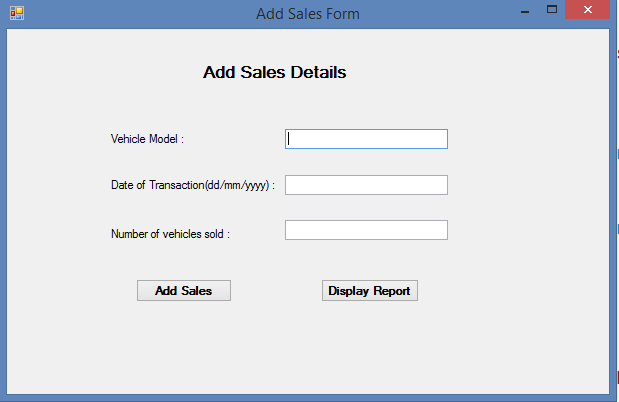


Figure: when application is started, “**AddSalesDetail**” form is displayed

It has fields to be entered for sales details and 2 buttons **“Add Sales**” and “**Display Report**”.

2. After entering Sales details user clicks “**Add Sales**” button and sales record gets saved in the ‘**Transaction**” table in database with “**transactionid**”, ”**date\_of\_trans**”,”**vehicleid**”, and “**no\_of\_sellings**” and message is displayed as below:

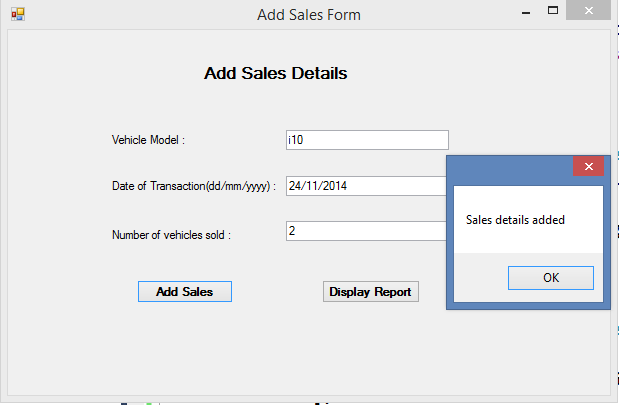


Figure: when “Add Sales” button is clicked after entering all the valid data

3. Following validations should be performed when “**Add Sales**” button is checked:

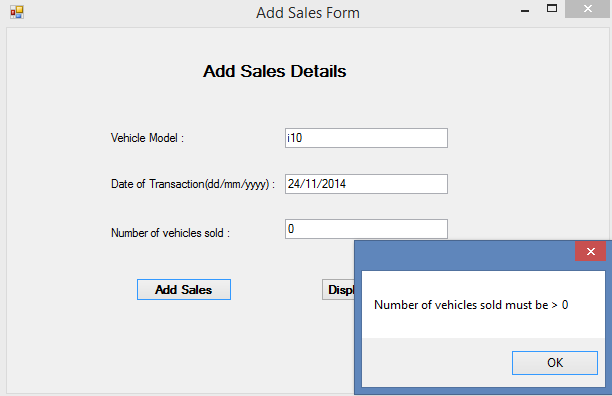


Figure: when amount is <=0 and “Add Sales” button is clicked

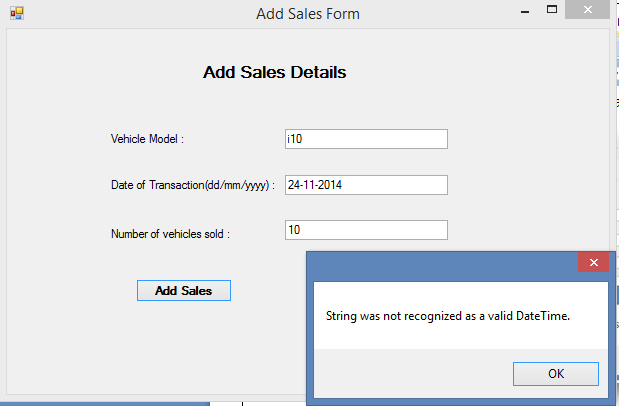


Figure: when transaction date format is other than “dd/MM/yyyy” and “Add Sales” button is clicked

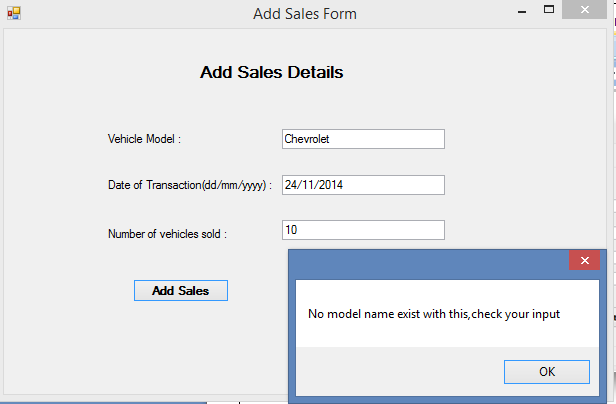
****

Figure: when model name entered is not present in database and “Add Sales” button is clicked

4. Once sales details are added and user wants to display the sales report of a different car models, user clicks “**Display Report**” and the second form “**DisplaySalesReport**” is opened as below in which total number of cars sold for each model is displayed in the **decreasing order of total number of sales** done as shown below in the DataGrid control.

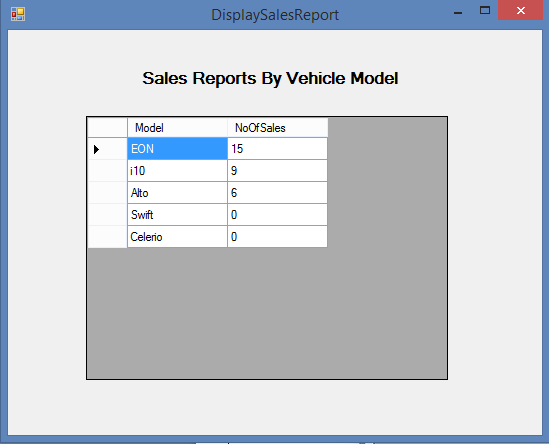


Figure: when user click the “Display Report” button in first form

**Note: If there are no cars sold for any particular car model, it should display 0 for that model**

**Note:**

1. Create following two tables **‘Vehicle’** and **‘Transaction’** which will be used in the application. For Table **‘Vehicle’** manually insert the values as per values given in this document. The values for **‘Transaction’** must be inserted through your application.
2. Read the complete question before coding
3. **Upload your table script with inserted values and stored procedure script along with project solution.**
4. All user defined exceptions must be created under VehiclesSalesReportExceptions project
5. All exceptions must be handled in Manager Class. DAO class should not handle any exception; it must throw it to Manager Class.
6. Loose coupling: Create a separate Dao interfaces and implementation classes. Database interaction code should be only in DAO classes. Dao classes should not have UI code.
7. **Write Nunit test cases for ‘SalesReportManager’ class**

Write Nunit methods in NunitTest class under VehiclesSalesReportNUnit project

* Create test case methods with exactly the same names given in the table below with appropriate annotations.
* The test case method naming convention is as follows:
  + The test case testing valid data is **TestFunctionalityNameValidData or TestFunctionalityNameSuccess**
  + The test case testing a particular invalid attribute is **TestFunctionalityNameInvalidAttributeName**

1. **Database design:**

Create database “**Sales\_DB**” and create the following two tables used for the application: “**Vehicle** and a link table “**Transaction**”. Listed below is the table design with constraints**.**

Table Name: **Vehicle**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Data type** | **Constraints** | **Description** |
| vehicleid | int | Primary Key |  |
| company | varchar(50) | NOT NULL |  |
| model | varchar(50) | NOT NULL |  |

Below is the content of table **‘Vehicle’,** which must be manually inserted into table**:**

|  |  |  |
| --- | --- | --- |
| **VEHICLEID** | **COMPANY** | **MODEL** |
| 1 | Hyundai | EON |
| 2 | Hyundai | i10 |
| 3 | Maruthi | Alto |
| 4 | Maruthi | Swift |
| 5 | Maruthi | Celerio |

Table Name: **Transaction**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column name** | **Data type** | **Constraints** | **Description** |
| TRANS\_ID | INT | Primary Key | Auto incremented column |
| VEHICLEID | INT | FOREIGN KEY | it must reference vehicleid column of vehicle table |
| DATE\_OF\_TRANS | DATETIME | NOT NULL |  |
| NO\_OF\_SELLINGS | INT | NOT NULL |  |

Your **SalesReportManager class** code should check for the business rules given below and throw appropriate exception, when exception thrown corresponding message should be displayed to user, which is as given in the table.

All the exceptions occurring in this business layer has to be logged in **“Errors.txt”** text file :

|  |  |  |  |
| --- | --- | --- | --- |
| **Rule No** | **Business constraint** | **Errors to be displayed or User defined exceptions to be thrown** | **Message to user to be displayed** |
| 1 | **Model\_Name** entered should be present in the database. | If **Model\_Name** is invalid, throw **SalesReportCustomException** | “No model name exist with this, check your input” |

Test the manager class by using sample data in the table given below corresponding for each of the test case.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sno** | **Test case method** | **Model name** | **Date\_of\_transaction** | **No. of Vehicles sold** |
| 1 | TestAddSalesDetailsValidData | EON | 24/11/2014 | 3 |
| 2 | TestAddSalesDetailsInvalidModelName | **Chevrolet** | 24/11/2014 | 3 |

Test the manager class by using sample data in the table given below corresponding for each of the test case

|  |  |
| --- | --- |
| **Sno** | **Test case method** |
| 1 | TestDisplaySalesReportSuccess |

**Notes:**

1. Do not change the table structures.
2. Provide required properties in entity classes
3. “VehiclesSalesReportDAL” project should contain all the database interaction code. It should implement the interface IDataAccess
4. “Add Sales” should be done using **“SQL”** statement in ADO.NET, don’t use stored procedure or Linq To Sql for this.
5. Use collections to sort by descending order of total sales done in each car model.
6. Use outer join to retrieve vehicles and sales data from database and then group them by model name
7. **“Display Report” has to be done using stored procedure and provide the sql script used for the stored procedure for this.Do not sort the result using order by sql clause, result should be sorted using collection sort method.**
8. **VehiclesSalesReportNUnit** class should provide methods to test business layer methods
9. **Provide required methods in the SalesReportManager class**
10. **Provide custom exceptions classes in VehiclesSalesReportExceptions project**
11. **Use file handling features to log an error in custom exception class for exception errors details** occurring in this business layer**.**
12. **“VehiclesSalesReportClient**” contains “**AddSalesDetail**” and “**DisplaySalesReport**” which are empty, complete the design and code to interact with Manager Class**.**