VMEP Dealer Management

Spare Parts Management System

Software Design Document

Issue 1.0.6

Table of Contents

[VMEP Dealer Management 1](#_Toc243972468)

[Spare Parts Management System 1](#_Toc243972469)

[Software Design Document 1](#_Toc243972470)

[Issue 1.0.6 1](#_Toc243972471)

[Chapter 1 Introduction 9](#_Toc243972472)

[1.1 Purpose 9](#_Toc243972473)

[1.2 Scope 9](#_Toc243972474)

[1.3 Definitions, Acronyms, and Abbreviations 9](#_Toc243972475)

[1.4 References 9](#_Toc243972476)

[Chapter 2 Introduction to LINQ and LINQ in VDMS 9](#_Toc243972477)

[2.1 What is the LINQ 9](#_Toc243972478)

[2.2 Architecture of LINQ in .Net Framework 3.5 10](#_Toc243972479)

[2.2.1 Standard Query Operators 10](#_Toc243972480)

[2.2.2 Language Extensions 12](#_Toc243972481)

[2.2.3 LINQ to SQL 14](#_Toc243972482)

[2.3 Apply LINQ to VDMS 15](#_Toc243972483)

[Chapter 3 Detail Design 15](#_Toc243972484)

[3.1 System Overview 15](#_Toc243972485)

[3.1.1 Basic data setting 15](#_Toc243972486)

[3.1.2 Security 16](#_Toc243972487)

[3.1.3 Main Business 18](#_Toc243972488)

[3.2 Use-Case Realizations 21](#_Toc243972489)

[3.2.1 Query database using LINQ and Devart component 21](#_Toc243972490)

[3.2.2 Interact with Tip-Top Sequence Diagram 22](#_Toc243972491)

[3.3 Application Design 24](#_Toc243972492)

[3.3.1 Main menu 24](#_Toc243972493)

[3.3.2 Homepage 25](#_Toc243972494)

[3.3.3 Common Message 27](#_Toc243972495)

[3.3.4 System setting 28](#_Toc243972496)

[3.4 Basic Data Setting 29](#_Toc243972497)

[3.4.1 Part/Accessory Setting 29](#_Toc243972498)

[3.4.2 Category Setting 30](#_Toc243972499)

[3.4.3 Customer Setting 32](#_Toc243972500)

[3.4.4 Vendor List 34](#_Toc243972501)

[3.5 Security 35](#_Toc243972502)

[3.5.1 Account Management 35](#_Toc243972503)

[3.5.2 Role Management 37](#_Toc243972504)

[3.5.3 Right Setup 38](#_Toc243972505)

[3.6 Main Business 40](#_Toc243972506)

[3.6.1 Cycle Counting Adjust 40](#_Toc243972507)

[3.6.2 Dealer List 42](#_Toc243972508)

[3.6.3 Part sale 44](#_Toc243972509)

[3.6.4 Order parts 46](#_Toc243972510)

[3.6.5 Order List and Receive 51](#_Toc243972511)

[3.6.6 Undo Auto Receive 55](#_Toc243972512)

[3.6.7 Special Import/Export 56](#_Toc243972513)

[3.6.8 Special Import/Export history 57](#_Toc243972514)

[3.6.9 Not Good Form 58](#_Toc243972515)

[3.6.10 Not Good Confirm 60](#_Toc243972516)

[3.6.11 Not Good Form tracking 61](#_Toc243972517)

[3.6.12 Stock Transfer 62](#_Toc243972518)

[3.6.13 Stock Transfer history 63](#_Toc243972519)

[3.6.14 Bin Card Query 64](#_Toc243972520)

[3.6.15 In Short Supply 65](#_Toc243972521)

[3.6.16 Monthly Close 66](#_Toc243972522)

[3.7 Report 68](#_Toc243972523)

[3.7.1 In Short supplier report 68](#_Toc243972524)

[3.7.2 Abnormal receive report 69](#_Toc243972525)

[3.7.3 Warranty Return Report 70](#_Toc243972526)

[3.7.4 Over-shipping Span report 71](#_Toc243972527)

[3.7.5 Supply and Service rate report 71](#_Toc243972528)

[3.7.6 Sales detail report 72](#_Toc243972529)

[3.7.7 Part Inventory report 73](#_Toc243972530)

[3.7.8 In/Out/Stock report 74](#_Toc243972531)

[Chapter 4 Software Class Description 75](#_Toc243972532)

[4.1 Architecturally Significant Design Packages 75](#_Toc243972533)

[4.2 The Data Package 75](#_Toc243972534)

[4.2.1 Data Context 75](#_Toc243972535)

[4.2.2 Tip-Top 76](#_Toc243972536)

[Chapter 5 Tip-Top Interfaces 77](#_Toc243972537)

[5.1 Summary 77](#_Toc243972538)

[5.2 Interface transaction 79](#_Toc243972539)

[5.2.1 The Interface defined 79](#_Toc243972540)

[5.3 New order 80](#_Toc243972541)

[5.4 Quantity Confirm 82](#_Toc243972542)

[5.5 Payment Confirm 82](#_Toc243972543)

[5.6 Change Order (after quotaton) 83](#_Toc243972544)

[5.7 Delete & Recove Order 84](#_Toc243972545)

[5.8 Delivery Date 86](#_Toc243972546)

[5.9 Update new SYM Part 86](#_Toc243972547)

[Chapter 6 Database Design 86](#_Toc243972548)

[6.1 Common Rules 86](#_Toc243972549)

[6.1.1 Naming Rules 86](#_Toc243972550)

[6.1.2 Default Value Rules 86](#_Toc243972551)

[6.2 UMS – User Management System 87](#_Toc243972552)

[6.3 Security Component 87](#_Toc243972553)

[6.3.1 Table SiteMap 88](#_Toc243972554)

[6.3.2 Table Task 88](#_Toc243972555)

[6.3.3 Table RolesInTasks 88](#_Toc243972556)

[6.3.4 Table RolesInPaths 89](#_Toc243972557)

[6.4 Sub-store Management 89](#_Toc243972558)

[6.4.1 Table Dealer 90](#_Toc243972559)

[6.4.2 Table Warehouse 90](#_Toc243972560)

[6.4.3 Table Contact 90](#_Toc243972561)

[6.5 Message Store 91](#_Toc243972562)

[6.5.1 Table Message 91](#_Toc243972563)

[6.5.2 Table MessageBox 91](#_Toc243972564)

[6.5.3 Table File 92](#_Toc243972565)

[6.6 Basic Data Setting 92](#_Toc243972566)

[6.6.1 Table Customer 93](#_Toc243972567)

[6.6.2 Table Vendor 93](#_Toc243972568)

[6.6.3 Table AccessoryType 93](#_Toc243972569)

[6.6.4 Table Accessory 93](#_Toc243972570)

[6.6.5 Table Category 94](#_Toc243972571)

[6.6.6 Table PartInfo 94](#_Toc243972572)

[6.6.7 Table Favorite 94](#_Toc243972573)

[6.7 Order Part 95](#_Toc243972574)

[6.7.1 Table OrderHeader 96](#_Toc243972575)

[6.7.2 Table OrderDetail 97](#_Toc243972576)

[6.7.3 Table ReceiveHeader 97](#_Toc243972577)

[6.7.4 Table ReceiveDetail 97](#_Toc243972578)

[6.8 Inventory 98](#_Toc243972579)

[6.8.1 Table Inventory 99](#_Toc243972580)

[6.8.2 Table SalesHeader 99](#_Toc243972581)

[6.8.3 Table SalesDetail 99](#_Toc243972582)

[6.8.4 Table TransferHeader 100](#_Toc243972583)

[6.8.5 Table TransferDetail 100](#_Toc243972584)

[6.8.6 Table CycleCountHeader 100](#_Toc243972585)

[6.8.7 Table CycleCountDetail 101](#_Toc243972586)

[6.8.8 Table TransactionHistory 101](#_Toc243972587)

[6.9 NG Form 102](#_Toc243972588)

[6.9.1 Table NGFormHeader 102](#_Toc243972589)

[6.9.2 Table NGFormDetail 103](#_Toc243972590)

[Chapter 7 Non-Functional Requirements 103](#_Toc243972591)

[7.1 Performance and Scalability 103](#_Toc243972592)

[7.1.1 Compress with IIS 6.0 103](#_Toc243972593)

[7.1.2 Cache web result 104](#_Toc243972594)

[7.2 Security 104](#_Toc243972595)

[7.3 Language 105](#_Toc243972596)

[7.4 Ajax 105](#_Toc243972597)

[7.4.1 Ajax Rules 106](#_Toc243972598)

[7.4.2 Ajax Result 106](#_Toc243972599)

**Figure list**

[Figure 1: Basic data setting 16](#_Toc243972600)

[Figure 2: Security component 17](#_Toc243972601)

[Figure 3: System Flow overview 19](#_Toc243972602)

[Figure 4: Query database using LINQ and Devart component 22](#_Toc243972603)

[Figure 5: Interact with Tip-Top Sequence Diagram 23](#_Toc243972604)

[Figure 6: VDMS's Main Menu 24](#_Toc243972605)

[Figure 7: VDMS's Homepage for dealer 25](#_Toc243972606)

[Figure 8: VDMS's Homepage for VMEP 25](#_Toc243972607)

[Figure 9: Homepage Data flow 26](#_Toc243972608)

[Figure 10: Message list 27](#_Toc243972609)

[Figure 11: Create new message 27](#_Toc243972610)

[Figure 12: System parameter setting 28](#_Toc243972611)

[Figure 13: Excel structure setting 28](#_Toc243972612)

[Figure 14: Part/Accessory Setting 29](#_Toc243972613)

[Figure 15: Basic Data Setting Data flow 30](#_Toc243972614)

[Figure 16: Main screen 30](#_Toc243972615)

[Figure 17: Category Setting Data flow 31](#_Toc243972616)

[Figure 18: Browse/Create Customer data 32](#_Toc243972617)

[Figure 19: Customer data flow 33](#_Toc243972618)

[Figure 20: Browse/Create new vendor 34](#_Toc243972619)

[Figure 21: Vendor Data flow 35](#_Toc243972620)

[Figure 22: Search users 35](#_Toc243972621)

[Figure 23: Edit existing user 36](#_Toc243972622)

[Figure 24: Create new user 36](#_Toc243972623)

[Figure 25: Manage roles 37](#_Toc243972624)

[Figure 26: Edit permissions setting 38](#_Toc243972625)

[Figure 27: Right Management 39](#_Toc243972626)

[Figure 28: Cycle Count adjust 40](#_Toc243972627)

[Figure 29: Cycle Count Data flow 41](#_Toc243972628)

[Figure 30: Main screen 42](#_Toc243972629)

[Figure 31: Create/Edit Basic dealer data 42](#_Toc243972630)

[Figure 32: Create/Edit Additional dealer data 42](#_Toc243972631)

[Figure 33: Manual key in data 44](#_Toc243972632)

[Figure 34: Upload data from excel file 44](#_Toc243972633)

[Figure 35: Sales Part data flow 45](#_Toc243972634)

[Figure 36: Search orders 46](#_Toc243972635)

[Figure 37: Manual create order 46](#_Toc243972636)

[Figure 38: Create order by excel file 46](#_Toc243972637)

[Figure 39: Create New Order Flow 47](#_Toc243972638)

[Figure 40: Upload Excel Order 48](#_Toc243972639)

[Figure 41: Order Part Data flow 50](#_Toc243972640)

[Figure 42: Search screen 51](#_Toc243972641)

[Figure 43: Update in stock/lack quantity 52](#_Toc243972642)

[Figure 44: Auto NG form 52](#_Toc243972643)

[Figure 45: Receive Part Data flow 53](#_Toc243972644)

[Figure 46: Undo auto confirm 55](#_Toc243972645)

[Figure 47: Special Import/Export 56](#_Toc243972646)

[Figure 48: Special Import/Export Data flow 57](#_Toc243972647)

[Figure 49: Special Import/Export history 57](#_Toc243972648)

[Figure 50: Search NG form 58](#_Toc243972649)

[Figure 51: Create NG form manual 58](#_Toc243972650)

[Figure 52: NG Form Data flow 59](#_Toc243972651)

[Figure 53: Confirm NG Form 60](#_Toc243972652)

[Figure 54: Confirm NG Form Data flow 60](#_Toc243972653)

[Figure 55: Not Good tracking 61](#_Toc243972654)

[Figure 56: Stock Transfer 62](#_Toc243972655)

[Figure 57: Stock Transfer Data flow 63](#_Toc243972656)

[Figure 58: Stock Transfer history 63](#_Toc243972657)

[Figure 59: Bin card info 64](#_Toc243972658)

[Figure 60: Bin Card Query Data flow 65](#_Toc243972659)

[Figure 61: In Short Supply 65](#_Toc243972660)

[Figure 62: Monthly summarization 66](#_Toc243972661)

[Figure 63: Monthly Close Data flow 67](#_Toc243972662)

[Figure 64: In Short Reason 68](#_Toc243972663)

[Figure 65: In short query data flow 69](#_Toc243972664)

[Figure 66: Abnormal receive report 69](#_Toc243972665)

[Figure 67: Warranty return report 70](#_Toc243972666)

[Figure 68: Over-shipping span report 71](#_Toc243972667)

[Figure 69: Supply and Service rate report 71](#_Toc243972668)

[Figure 70: Sales detail report 72](#_Toc243972669)

[Figure 71: Part Inventory report 73](#_Toc243972670)

[Figure 72: In/Out/Stock report 74](#_Toc243972671)

[Figure 73: Packages Diagram 75](#_Toc243972672)

[Figure 74: Domain Component 76](#_Toc243972673)

[Figure 75: Tip-Top Database Access Component 76](#_Toc243972674)

[Figure 76: Tip-Top Interface table 79](#_Toc243972675)

[Figure 77: User Management System Schema 87](#_Toc243972676)

[Figure 78: Security Schema 88](#_Toc243972677)

[Figure 79: Sub-store Management Schema 89](#_Toc243972678)

[Figure 80: Message Store 91](#_Toc243972679)

[Figure 81: Basic Data Setting Schema 92](#_Toc243972680)

[Figure 82: Part Order Schema 95](#_Toc243972681)

[Figure 83: Inventory Schema 98](#_Toc243972682)

[Figure 84: NG Form schema 102](#_Toc243972683)

[Figure 85: Retrieve Item Process 104](#_Toc243972684)

[Figure 86: Security Checking 105](#_Toc243972685)

**Modification tracking**

|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Page** |
| **1** | Add Basic Data Setting schema and processing | 15 |
| **2** | Add Security schema and processing | 16 |
| **3** | Modify the Main Business | 18,19 |
| **4** | Add the System Setting user-case | 28 |
| **5** | Update Dealer List user-case | 42 |
| **6** | Update Order List and Receive user-case | 51 |
| **7** | Add Special Import/Export history user-case | 57 |
| **8** | Modify the NG form user-case | 58-60 |
| **9** | Add Not Good Form tracking user-case | 61 |
| **10** | Add Stock Transfer history user-case | 63 |
| **11** | Modify In-short Supply user-case | 65 |
| **12** | Add all Report user-case | 67-72 |
| **13** | Modify the table Dealer | 89 |
| **14** | Modify the table OrderHeader | 95 |
| **15** | Add table CycleCountHeader and table CycleCountDetail | 99,100 |
| **16** | Modify table NGFormHeader and table NGFormDetail | 101,102 |

# Introduction

## Purpose

This document is descriptions of a software product that a software designer writes in order to give a software development team an overall guidance of the architecture of the software project. It accompanies an architecture diagram and has pointers to the detailed feature specifications of smaller pieces of the design.

## Scope

The Software Design Document for the VDMS is composed of descriptions of the detail design, the interface design and the non-functional requirements.

The detail design section includes use-case realizations and software class descriptions.

## Definitions, Acronyms, and Abbreviations

All terms, acronyms, and abbreviations used in this document have been defined in the project’s Glossary document.

## References

This document is related to these documents below:

* The Software Architecture Document
* The User Requirement Document

# Introduction to LINQ and LINQ in VDMS

VDMS use LINQ to interact with database (creates, read, delete, update, query). This chapter shows the brief about LINQ and the way to apply LINQ to VDMS.

## What is the LINQ

Language Integrated Query (LINQ, pronounced "link") is a Microsoft .NET Framework component that adds native data querying capabilities to .NET languages.

Microsoft LINQ defines a set of query operators that can be used to query, project and filter data in arrays, enumerable classes, XML, relational database, and third party data sources. While it allows any data source to be queried, it requires that the data be encapsulated as objects. So, if the data source does not natively store data as objects, the data must be mapped to the object domain. Queries written using the query operators are executed either by the LINQ query processing engine or, via an extension mechanism, handed over to LINQ providers which either implement a separate query processing engine or translate to a different format to be executed on a separate data store (such as on a database server as SQL queries). The results of a query are returned as a collection of in-memory objects that can be enumerated using a standard iterator function such as C#'s foreach.

Many of the concepts that LINQ has introduced were originally tested in Microsoft's Cω research project. LINQ was released as a part of .NET Framework 3.5 on November 19, 2007.

## Architecture of LINQ in .Net Framework 3.5

### Standard Query Operators

The set of query operators defined by LINQ are exposed to the user as the Standard Query Operator API. The query operators supported by the API are:

#### Select / SelectMany

The Select statement is used to perform a projection on the collection to select either all the data members that make up the object or a subset of it. The SelectMany operator is used to perform a one-to-many projection, i.e., if the objects in the collection contain another collection as a data member, SelectMany can be used to select the entire sub-collection. The user supplies a function, as a delegate, which projects the data members. Selection creates an object of a different type, which has either some or as many data members as the original class. The class must be already defined for the code to be compliable.

#### Where

The Where operator allows the definition of a set of predicate rules which are evaluated for each object in the collection, and objects which do not match the rule are filtered away. The predicate is supplied to the operator as a delegate.

#### Sum / Min / Max / Average / Aggregate

These operators take a predicate that retrieves a certain numeric value from each element in the collection and uses it to find the sum, minimum, maximum, average or aggregate values of all the elements in the collection, respectively.

#### Join / GroupJoin

The Join operator performs an inner join on two collections, based on matching keys for objects in each collection. It takes two functions as delegates, one for each collection, that it executes on each object in the collection to extract the key from the object. It also takes another delegate via which the user specifies which data elements, from the two matched elements, should be used to create the resultant object. The GroupJoin operator is used to perform a group join. Like the Select operator, the results of a join are instantiations of a different class, with all the data members of both the types of the source objects, or a subset of them.

#### Take / TakeWhile

The Take operator is used to select the first n objects from a collection, while the TakeWhile operator, which takes a predicate, selects those objects which match the predicate.

#### Skip / SkipWhile

The Skip and SkipWhile operators are complements of Take and TakeWhile - they skip the first n objects from a collection, or those objects which match a predicate (for the case of SkipWhile).

#### OfType

The OfType operator is used to select the elements of a certain type.

#### Concat

The Concat operator concatenates two collections.

#### OrderBy / ThenBy

The OrderBy operator is used to specify the primary sort ordering of the elements in a collection according to some key. The default ordering is in ascending order, to reverse the order the OrderByDescending operator is to be used. ThenBy and ThenByDescending specifies subsequent ordering of the elements. The function to extract the key value from the object is specified by the user as a delegate.

#### Reverse

The Reverse operator reverses a collection.

#### GroupBy

The GroupBy operator takes a delegate that extracts a key value and returns a collection of IGrouping<Key, Values> objects, for each distinct key value. The IGrouping objects can then be used to enumerate all the objects for a particular key value.

#### Distinct

The Distinct operator removes duplicate instances of a key value from a collection. The function to retrieve the key value is to be supplied as a delegate.

#### Union/Intersect/Except

These operators are used to perform a union, intersection and difference operation on two sequences, respectively.

#### EqualAll

The EqualAll operator checks if all elements in two collections are equal.

#### First/FirstOrDefault/Last/LastOrDefault

These operators take a predicate. The First operator returns the first element for which the predicate yields true or throws an exception if nothing matches. The FirstOrDefault operator is like the First operator except that it returns the default value for the element types (usually a null reference) in case nothing matches the predicate. The last operator retrieves the last element to match the predicate, or throws an exception in case nothing matches. The LastOrDefault returns the default element value if nothing matches.

#### Single

The Single operator takes a predicate and returns the element which matches the predicate. An exception is thrown if none or more than one element matches the predicate.

#### ElementAt

The ElementAt operator retrieves the element at a given index in the collection.

#### Any / All / Contains

The Any operator checks if there are any element in the collection matching the predicate. It does not select the element, but returns true for a match. The All operator checks if all elements match the predicate. The Contains operator checks if the collection contains a given value.

#### Count

The Count operator counts the number of elements in the given collection.

The Standard Query Operator also specifies certain operators which converts a collection into other types:

* AsEnumerable: converts the collection to IEnumerable<T> type.
* ToQueryable: converts the collection to IQueryable<T> type.
* ToArray: converts the collection to an array.
* ToList: converts the collection to IList<T> type.
* ToDictionary: converts the collection to IDictionary<K, T> type, indexed by the key K.
* ToLookup: converts the collection to ILookup<K, T> type, indexed by the key K.
* Cast: converts a non-generic IEnumerable collection to one of IEnumerable<T> by casting each element to type T. Throws an exception for incompatible types.
* OfType: converts a non-generic IEnumerable collection to one of IEnumerable<T>. Only elements of type T are included.

The query operators are defined in the IEnumerable<T> interface as generic extension methods, and a concrete implementation is provided in the Sequence class. As a result, any class which implements the IEnumerable<T> interface has access to these methods and are queryable. LINQ also defines a set of generic Func delegates, which define the type of delegates handled by the LINQ query methods. Any function wrapped in a Func delegate can be used by LINQ. Each of these methods return an IEnumerable<T>, so the output of one can be used as input to another, resulting in query composability. The functions, however, are lazily evaluated, i.e., the collections are enumerated only when the result is retrieved. The enumeration is halted as soon as a match is found, and the delegates evaluated on it. When a subsequent object in the resultant collection is retrieved, the enumeration of the source collection is continued beyond the element already evaluated. However, grouping operations, like GroupBy and OrderBy, as well as Sum, Min, Max, Average and Aggregate, require data from all elements in collection, and force an eager evaluation. LINQ does not feature a query optimizer and the query operators are evaluated in the order they are invoked. The LINQ methods are compliable in .NET Framework 2.0, as well.

### Language Extensions

While LINQ is primarily implemented as a library for .NET Framework 3.5, it also defines a set of language extensions that can be optionally implemented by languages to make queries a first class language construct and provide syntactic sugar for writing queries. These language extensions have initially been implemented in C# 3.0, VB 9.0 and Oxygene, with other languages like F# and Nemerle having announced preliminary support. The language extensions include:

* Query syntax: Languages are free to choose a query syntax, which it will recognize natively. These language keywords must be translated by the compiler to appropriate LINQ method calls. The languages can implement operator reordering and other optimizations at the keyword level.
* Implicitly typed variables: This enhancement allows variables to be declared without specifying their types. The languages C# 3.0 and Oxygene declare them with the var keyword. In VB9.0, the use of the Dim keyword without type declaration accomplishes the same declaration. Such objects are still strongly typed; for these objects the compiler uses type inference to infer the type of the variables. This allows the result of the queries to be specified and their result defined without declaring the type of the intermediate variables.
* Anonymous types: Anonymous types allow classes, which contain only data member declarations, to be inferred by the compiler. This is useful for the Select and Join operators, whose result types may differ from the types of the original objects. The compiler uses type inference to determine the fields contained in the classes and generates accessors and mutators for these fields.
* Object Initializer: Object initializers allow an object to be created and initialized in a single scope, this allows creation of delegates that extract fields from an object, create a new object and assign the extracted data to the fields of the new object in a single statement, as is required for Select and Join operators.
* Lambda expressions: Lambda expressions are used to create delegates inline with other code. This allows the predicates and extraction functions to be written inline with the queries.

For example, in the query to select all the objects in a collection with SomeProperty less than 10

|  |
| --- |
| int someValue = 5;    var results = from c in someCollection  let x = someValue \* 2  where c.SomeProperty < x  select new {c.SomeProperty, c.OtherProperty};    foreach (var result in results)  {  Console.WriteLine(result);  } |

the types of variables result, c and results all are inferred by the compiler - assuming SomeCollection is IEnumerable<SomeClass>, c will be SomeClass, results will be IEnumerable<SomeOtherClass> and result will be SomeOtherClass, where SomeOtherClass will be a compiler generated class with only the SomeProperty and OtherProperty properties and their values set from the corresponding clauses of the source objects. The operators are then translated into method calls as:

|  |
| --- |
| IEnumerable<SomeOtherClass> results =  SomeCollection.Where  (  c => c.SomeProperty < (SomeValue \* 2)  )  .Select  (  c => new {c.SomeProperty, c.OtherProperty}  )  foreach (SomeOtherClass result in results)  {  Console.WriteLine(result.ToString());  } |

### LINQ to SQL

The LINQ to SQL provider allows LINQ to be used to query SQL Server databases as well as SQL Server Compact databases. Since SQL Server data resides on a remote server, and because it already includes a querying engine, LINQ to SQL does not use the query engine of LINQ. Instead, it converts a LINQ query to SQL query which is then sent to SQL Server for processing.[6] However, since SQL Server stores the data as relational data and LINQ works with data encapsulated in objects, the two representations must be mapped to one another. For this reason, LINQ to SQL also defines the mapping framework. The mapping is done by defining classes which corresponds to the tables in database, and containing all or a subset of the columns in the table as data members. The correspondence, along with other relational model attributes such as primary keys are specified using LINQ to SQL-defined attributes. For example,

|  |
| --- |
| [Table(Name="Customers")]  public class Customer  {  [Column(IsPrimAryKey = true)]  public int CustID;    [Column]  public string CustName;  } |

this class definition maps to a table named Customers and the two data members correspond to two columns. The classes must be defined before LINQ to SQL can be used. Visual Studio 2008 includes a mapping designer which can be used to create the mapping between the data schemas in the object as well as relational domain. It can automatically create the corresponding classes from a database schema, as well as allow manual editing to create a different view by using only a subset of the tables or columns in a table.

The mapping is implemented by the DataContext which takes a connection string to the server, and can be used to generate a Table<T> where T is the type that the database table will be mapped to. The Table<T> encapsulates the data in the table, and implements the IQueryable<T> interface, so that the expression tree is created, which the LINQ to SQL provider handles. It converts the query into T-SQL and retrieves the result set from the database server. Since the processing happens at the database server, local methods, which are not defined as a part of the lambda expressions representing the predicates, cannot be used. However, it can use the stored procedures on the server. Any changes to the result set are tracked and can be submitted back to the database server.

## Apply LINQ to VDMS

In VDMS, LINQ is used to interact with database. Because the LINQ component of Microsoft does not work with Oracle, so the system use the [LINQ to Oracle of Devart Company](http://devart.com/dotconnect/) instead.

For construct the data access component for VDMS, follow the steps below:

* Create VDMS Oracle database (see chapter 6 for detail)
* Using Entity Developer to generate the DataContext
* Copy the DataContext to App\_Code folder in VDMS Web project
* Work as usual.

In this document, from this time, have some word with the same meaning:

* Table in database mean Class in C#
* Row in database mean Object/Entity in C#
* Column in database mean Property in C#

# Detail Design

## System Overview

This section descript the overview of system, so the stakeholder will have the basic concept about system, include the basic database, the business flow and the integration between VDMS-I, VDMS-II and Tip-Top.

### Basic data setting

The basic data is the data which be used in entire system. Is also used by VDMS-I. There is dealer list (which contains parent-child relationship), warehouse list, part information…

|  |  |  |
| --- | --- | --- |
| **No** | **Title** | **Description** |
| **1** | Part | The information about part, mapping to ima\_file in Tip-Top. This is the original data of part, that contains the part code, part name, model using for \*import\* (order, receive, special import) part to the inventory |
| **2** | PartInfo | After the first time \*import\* to the inventory (stock), the information of part will be saved to VDMS side, separated by each dealer. The data used for \*export\* (sales, special export, transfer…) |
| **3** | Accessory/Category | Store the user defined part. Dealer can import and export item on accessory base on the current quantity |
| **4** | Dealer | The information of dealer (dealer code, dealer name…) is copied from Tip-Top. In VDMS, it handle the parent-child structure for multi-organization management |
| **5** | Warehouse | Each warehouse in VDMS correspond with one address of dealer in Tip-Top and it has its own data items |
| **6** | Customer/Vendor | Separated by dealer, customer store the information about buyer (sales part) and vendor store the information about seller (special import) |

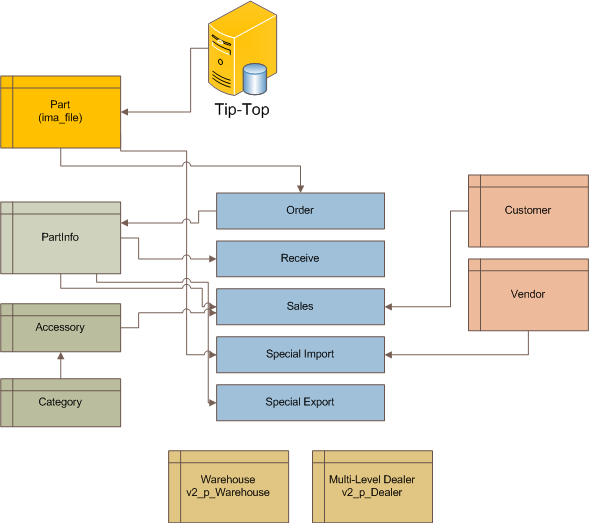


Figure 1: Basic data setting

At initial, some data can be import from Tip-Top by several ways: migrate from the table, import from excel file or manual insert into database.

### Security

The security component control the security of system, from the login, navigator, do action, execute separated data…

|  |  |  |
| --- | --- | --- |
| **No** | **Title** | **Description** |
| **1** | Check authentication | When server has new request, first it check the authentication (by session). If not, it redirects to login page and verify the username/password. Done by asp.net security component |
| **2** | Check authorization | After check authentication (\*), the system will check the authorization by Membership component. At this step, it checks the roles of user to ensure the navigator page or post action. In addition, VDDMS also verify the dealer code, warehouse identify to separate the data of user.  *\* Check authentication mean the system ask* ***Who are you****, and check authorization mean the system says* ***I know you, but you are restrict to do some action*** |
| **3** | Membership/Role provider | The ASP.NET security component that handle information of user, and the roles that user in. |
| **4** | Right Management: Role in Path, Role in Task | In web application, there are two actions: POST and GET. GET mean user want to get data (download) in a page by navigator to url. Role In Path will handle the permission of role in path, and we can calculate user can or cannot view a page. All the page init by sitemap (display the menu structure) and copy to database (table app\_Path)  POST mean user post some data to server, for example post the edited order to server (he wants to save order). Role in Task help system detect user can or cannot post data to server (in each page) |
| **5** | Separate data by user: Dealer and Warehouse | All data in VDMS is separated by dealer/warehouse. When user logged in to system, he has information about dealer (or VMEP) and warehouse. For each request, data is filtered by dealer code and warehouse identify. |
| **6** | Separate data by group: database, area, level… | Another parameter is database (HTP/DNP), area (HCM, HN…) is copied from Tip-Top. |

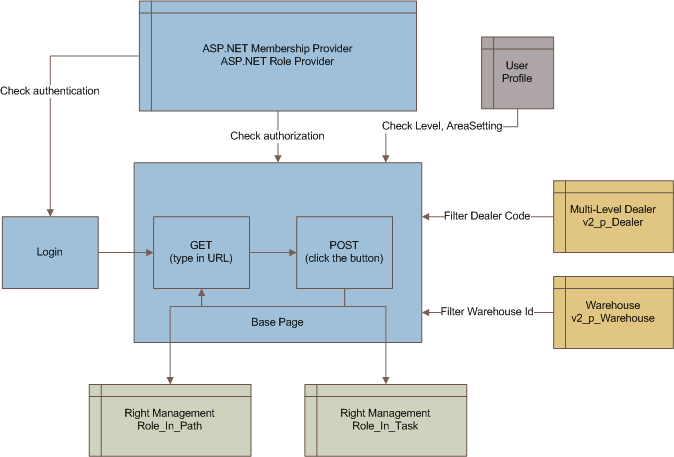


Figure 2: Security component

### Main Business

The main business process these main functions in the system, they are Order Part, Part Receive, Part Sales, NG Process and Inventory

#### Order Part

This component allows dealer make an order and send to VMEP. First, it gets the dealer and warehouse information to make the header of order, and then, it collects the part data from part table, make the order body and save to database.

The order also is saved to interface file, so the Tip-Top can read this order. After do quotation, Tip-Top will write the changed data back to interface, and activate the trigger so the changing is synchronize with VDMS’s database.

#### Part Receive

After payment, VMEP ship the order to dealer, and dealer will import to system. In this case, system write the import data, tracking the import status for make the Not Good form (combine with NG Form process), ~~create Sub-Order is necessary~~, and update the inventory of dealer.

It also provides the auto receive and undo receive feature: instead of import by manual, the system do the receiving (calculate data, change the status) automatically. All the changing be logged, for revert if necessary.

#### Part Sales

After order part (mean the inventory not null), dealer can sell the part to end customer. The Part Sales will update the inventory, transaction and customer information.

#### Not Good form

The Not Good form be created in tow cases: first is by the receive data: it collects the abnormal part and create the Not Good form automatically (combine with Receive parts process). Second, dealer can make the not good form that contains the broken part not belong to receive transaction. When dealer send to VMEP, the Not Good form is processed by many level users.

#### Inventory

Process the inventory of dealer: import, export, on-hand. Provide the cycle count adjust to adjust the current on-hand, and special import/export to do some special inventory.

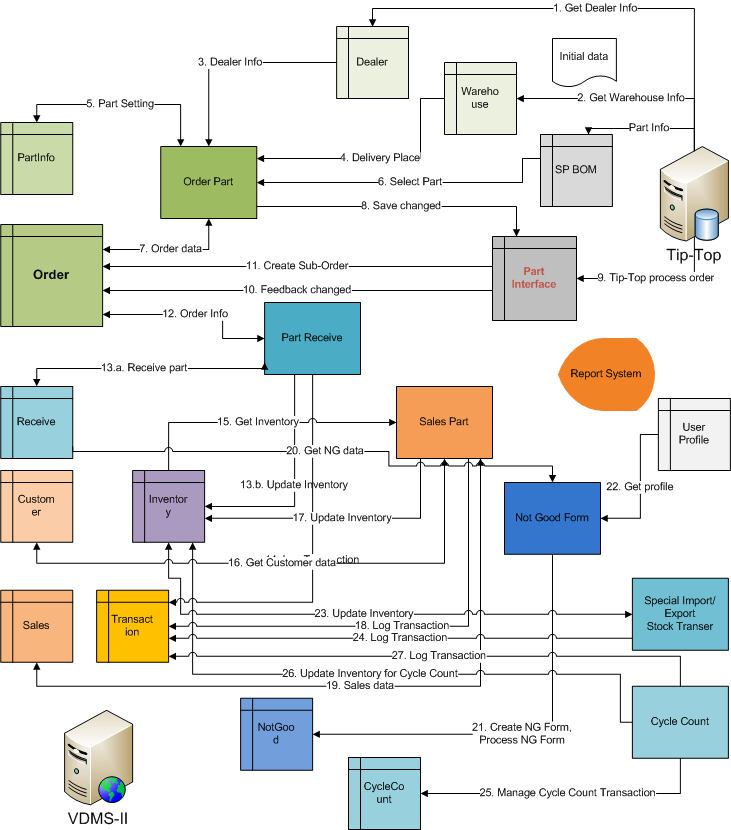


Figure 3: System Flow overview

|  |  |  |
| --- | --- | --- |
| **No** | **Title** | **Description** |
| **Init data from Tip-Top** | | |
| **1** | Get Dealer Info | Get the dealer list from Tip-Top, and make the dealer tree structure. This function maintains the dealers which use system, and only they can login and process the request in the system. |
| **2** | Get Warehouse Info | When get the dealer, also get the warehouse list of each dealer. |
| **Making Part Order** | | |
| **3** | Dealer Info | For making order part, get the dealer info (dealer code from the dealer list). This save to header of order |
| **4** | Delivery Place | Like Dealer Info, get the delivery place from the warehouse list of dealer. |
| **5** | Part Setting | The Order Part get/set part setting from/to PartInfo. The setting includes favorite, rank, safety stock. When select, copy part code to the order body. |
| **6** | Select Part | Select (search) the part info (part code, part name, model) from the where-user table from the Tip-Top |
| **7** | Order data | After make the order, save to the order data. It also can load the old order for editing |
| **8** | Save changed | When save order to order table, it also saves to the interface file. For detail, see the interface structure. |
| **9** | Tip-Top process order | When order exists in interface file, Tip-Top can query and process the order, and update the data into interface file. For detail, see the interface process. |
| **10** | Feedback changed | When data in the interface file changed (cause of Tip-Top), it active the trigger and update the changed into the order table. For detail, see the interface process. |
| **11** | Create sub-order | Special case: when make payment, the trigger create sub-order if necessary. |
| **Receive Part** | | |
| **12** | Order Info | The receiver gets the order info from order table to display/receive part. If order already received, depend on the NG Form, it allows dealer receives the lack part, or only shows the information. If order has not received, it take the quotation from order and allow user do in stock |
| **13.a** | Receive part | Collect the data from user: save good data to receive and transfer broken, lack and wrong automatically to NG Form. Do in-stock and send NG form to VMEP at the same time. |
| **13.b** | Update Inventory | When save received, it also updates the inventory. Only good part can be saved to inventory, the other wait for NG Form |
| **14** | Log Transaction | The inventory log to transaction for reporting |
| **Sale Part** | | |
| **15** | Get Inventory | Get the inventory information: the main data is current stock, user can not sale out more than current stock |
| **16** | Get Customer data | Get the customer data from customer table. Another case is leaving this field, so only customer name can be collected. |
| **17** | Update Inventory | After sale out, update inventory with negative quantity |
| **18** | Log Transaction | The sale transaction save to Transaction table for reporting |
| **19** | Sales data | The sales data can be save/load to/from the sale table |
| **Not Good Form** | | |
| **20** | Get NG data | Get the NG data from the receive (broken, wrong, lack), and copy to screen |
| **21** | Create NG Form, Process NG Form | Create the NG form with the data collected above. It also allow user approve NG Form by change the status of NG Form. There are 3 level approved, and the level take from User Profile |
| **22** | Get Profile | The User Profile stored the NG level, in which NG Form can be processed as in the flow. Maximum has 3 levels, and at level 3, the status of NG change to approved. |
| **Inventory Management: Special Import/Export – Stock Transfer** | | |
| **23** | Update Inventory | The inventory management update the stock inventory (plus if import, minus if export) |
| **24** | Log Transaction | All transaction save to Transaction table for reporting |
| **Cycle count** | | |
| **25** | Manage Cycle Count Transaction | Manage all the cycle count transaction, the data excel file upload for cycle count. |
| **26** | Do Cycle Count | Do the cycle count and update the inventory of warehouse |
| **27** | Log Transaction | Log the cycle count transaction |

## Use-Case Realizations

This section shows the architecturally significant use-case realizations. Each use case realization is a collaboration of design elements from the business components and common elements and services. For each use case we show use case dynamic view. That is the collaboration between the use case participants.

It is important to notice, that all use case realizations have a very similar structure. So that we are only introduce these most important diagrams as below.

### Query database using LINQ and Devart component

The diagram below describes the flow of logic how to query and execute SQL command in the system. It starts from ASP.NET client, such as Web Browser. The HttpRequest will send to ASP.NET Server (IIS), and process by the MasterPage. The MasterPage check the security and if user has enough permission, it calls the web page for process event and business logic. The business logic is creates the data context by using Data Context Factory, after that the Data Context execute LINQ command and pass value result to Web Page.



Figure 4: Query database using LINQ and Devart component

Flow explanations:

1. User is giving call to system by executing query through server for retrieve a list of entity via web browser (such as Internet Explorer or FireFox).
2. ASP.NET Server calls the Page object, and this object call the function to check the permission of user.
3. The check permission function return the value indicates user can or cannot do the action.
4. The Page creates a data context by calling the Data Context Factory.
5. The Data Context Factory returns the correct context factory.
6. The Page creates the LINQ and pass to the data context.
7. The data context passes to Devart component to execute the query.
8. The Devart component returns the result to data context.
9. The data context returns the object result to the page.
10. The page draws the result and passes the http response to client.

### Interact with Tip-Top Sequence Diagram

The diagram below describes the flow of logic in that may interact with Tip-Top via interface. In this flow below, there are two phases: the first is retrieve data from VDMS database and the second is query a command from Tip-Top database via interface.



Figure 5: Interact with Tip-Top Sequence Diagram

Flow explanations:

1. User query data from web client.
2. The Web Page creates the Data Context.
3. The Data Context executes LINQ command.
4. The Devart return the object result to Data Context.
5. The Data Context returns the object result to Web Page.
6. The Web Server returns the Http stream to user.
7. User query data from Tip-Top.
8. The Web Page call Query method from TipTop Component
9. The TipTop Component execute query from Enterprise Library.
10. The Enterprise Library return result to TipTop Component
11. The TipTop Component return result to Web Page
12. The Web Server returns the Http stream to user.

## Application Design

This section contains the detail description of each screen; include the function screen, the function description and the instructions of fields.

### Main menu

#### Function screen

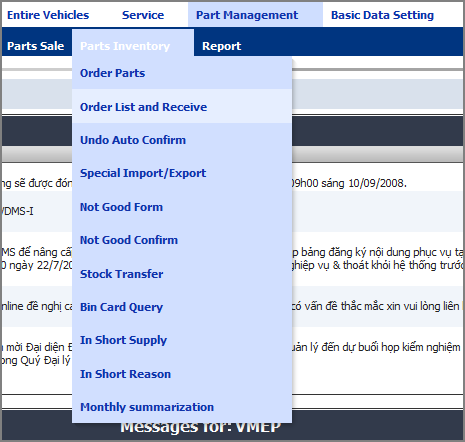


Figure 6: VDMS's Main Menu

#### Function description

Main menu allow user to choose the working page. It render by using Menu control, and map to two SiteMapPath files (one for English version and one for Vietnamese version)

### Homepage

#### Function screen

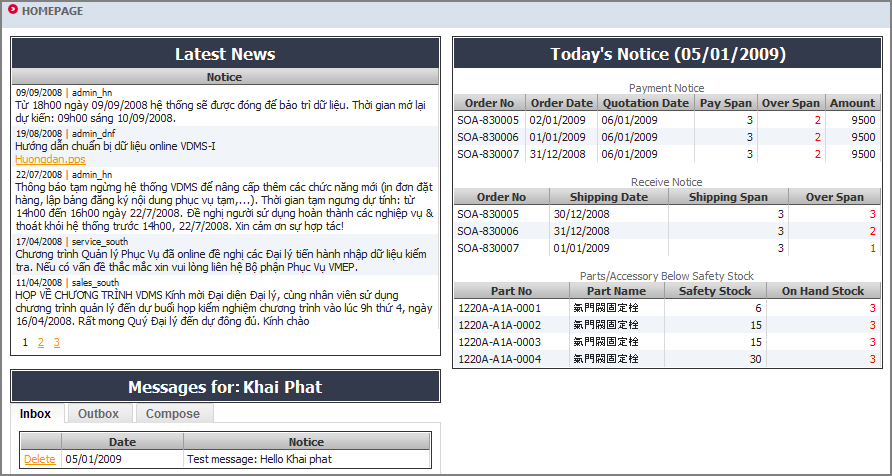


Figure 7: VDMS's Homepage for dealer



Figure 8: VDMS's Homepage for VMEP

#### Function description

This screen shows the list of news in the system, messages between dealer and VMEP. There is not any search condition, but the result will be paged. If user is dealer there is additional box call Today’s notices that show all notices about payment, receive and safety stock, these values are calculated by system by looking [order data](#_Table_p_OrderHeader) and [receive data](#_Table_p_Receive_Header) and compare to [dealer spans time setting](#_Table_p_Dealer).

#### Function constrains

* Message have attachment file with size less than 5M.

#### Data flow



Figure 9: Homepage Data flow

* Message separated by group: Vehicle, Sales and Spare Parts. The group of user stored in table UserProfile and loaded when user logging to system. When dealer send message to VMEP, this message will send to user which belong to correspond group.
* Message can send to multi-user, all user can see/delete the message with attachment, system only save the relationship between user and inbox/outbox, and we have only one instance of message/attachment in the system.
* For automatically show the warning message for dealer, system create a background thread that active each 5 second and check and send message.

#### Instructions of fields

System news is stored in Message table, but the relationship between user and message stored in MessageBox table.

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Date** | Date that message has been sent, stored in CreatedDate field. Default value is current time. |
| **From** | Dealer that send message, stored in FromUser field in table MessageBox. Deault value is username of current user |
| **To** | Dealer that receive message, stored in ToUser field in table MessageBox. |
| **Notice** | Message content, stored in Body field. |
| **Position** | Inbox or Outbox, select from table MessageBox with Position is ‘I’ (Inbox) or ‘O’ (Outbox) |
| **Answer or not?** | ParentId in Message is not NULL if answer, NULL mean not answer. |

### Common Message

#### Function screen

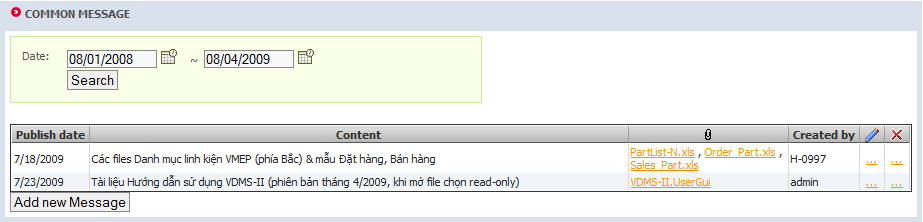


Figure 10: Message list

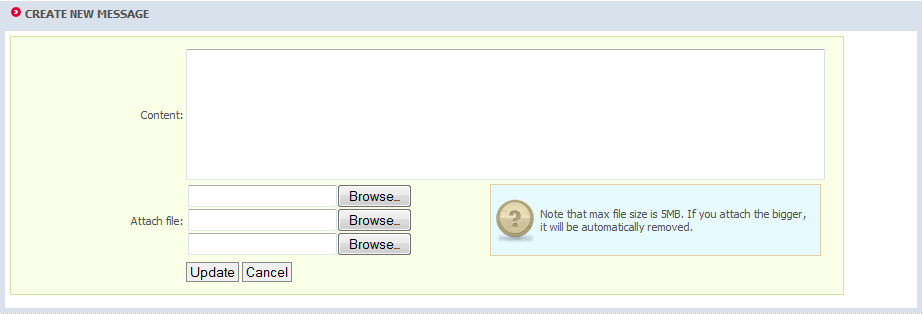


Figure 11: Create new message

#### Function description

This function used to maintain system messages on home page.

#### Instructions of fields

The message will be saved to table Message

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Created date** | Date that message created, saved to CreatedDate field. Default value is current time |
| **Create by** | Current user who created message. |
| **Body** | Message content, saved to Body field. |
| **Flag** | In table Message, select only rows with flag = ‘C’ (Common message) |

Message’s attachments will be saved to table File

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Filename** | File name of attachment file, saved to FileName field. |
| **Body** | Attach file content, saved to Body field. |
| **MessageId** | Reference to corresponding message by MessageId field. |

### System setting

#### Function screen

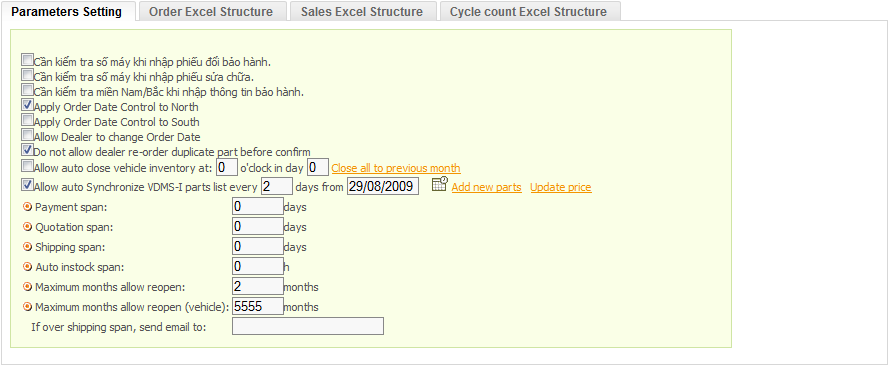


Figure 12: System parameter setting

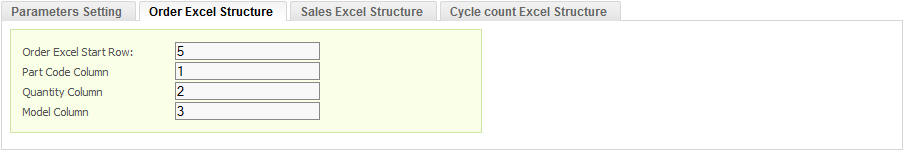


Figure 13: Excel structure setting

#### Function description

This function defines the setting parameter of system, and saves to setting file. The setting as below:

* Apply order date control: allow VMEP apply the function to control the order of dealer (allow or deny dealer make order in special day).
* Allow or deny dealer change order date.
* Allow or deny send order if this order contains the part the does not quotation in last order.
* Allow system do the automatically function in the special time
* Setting the default span for system
* Define the excel structure for dealer
* System setting values are stored in xml file (same with VDMS-I).

## Basic Data Setting

### Part/Accessory Setting

#### Function screen

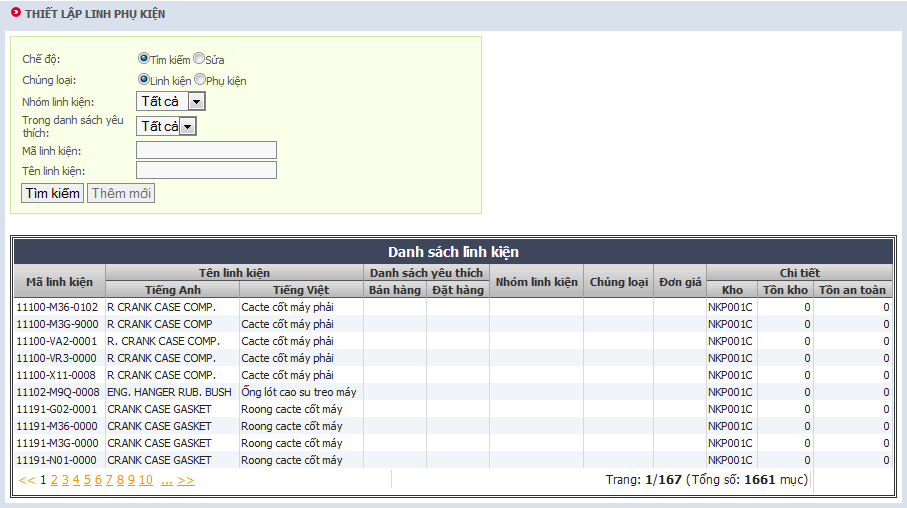


Figure 14: Part/Accessory Setting

#### Function description

1. This function used to query part/accessory information and maintain safety stock value.
2. SYM part list get from Tip-Top by VIEW, at VDMS side only setting: safety stock (table PartSafety), favorite flag and favorite rank (table Favorite)
3. Favorite rank: system defined value from 1 🡪 10, using for both SYM & Non-SYM part
4. Update part list of inventory when have new part: When dealer key in “receive order”, system will check (compare with dealer’s inventory part list at VDMS) and automatically update parts into inventory part list of dealer
5. System provides dealers with Non-SYM parts information upload function from excel file: Using Common/ExcelDataReader function.
6. The part of VMEP and Non-SYM is distingue by the Flag:
   1. SYM: S
   2. Non-SYM: N

#### Data flow

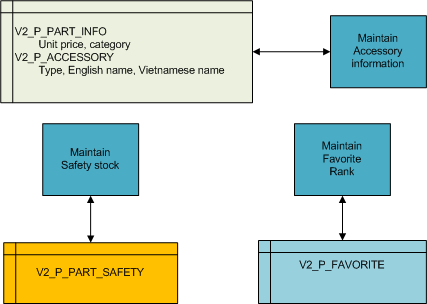


Figure 15: Basic Data Setting Data flow

#### Instructions of fields

Part information (PartCode, PartName, Specification, Model, Category) get from view in Tip-Top.

|  |  |
| --- | --- |
| Field name | Field description |
| Price | The price of part, in table PartInfo, read-only for part |
| Part Name | Name of part, English and Vietnamese, read-only |
| Safety Stock | Safety Stock value, stored in SAFETY\_QUANTITY field (table PartSafety). |
| Favorite Rank | This part belong to which rank of favorite (order or sale’s favorite), in table Favorite, value 1 to 10, two values for order and sales |
| On hand | Current quantity in stock |

### Category Setting

#### Function screen

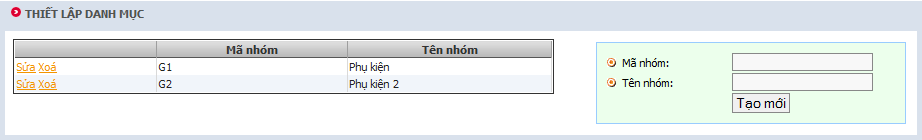


Figure 16: Category setting

#### Function description

Setup category for accessories includes non SYM part.

#### Function constrains

* + This function used only by dealer users, dealer administrators or system administrators with dealer role.
  + These categories belong to individual dealer so category code cannot be duplicated in the same dealer: CategoryCode + dealercode are unique.
  + Category code and category name cannot be empty.
  + Category data for SYM parts is gotten from Tip-Top system by VIEW (same method with SYM part setting)

#### Data flow



Figure 17: Category Setting Data flow

#### Instructions of fields

The part category will be saved to table V2\_P\_CATEGORY

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Category Name** | Name of category, saved in Name field. |
| **Category Code** | The code of category, save in Code field |

### Customer Setting

#### Function screen

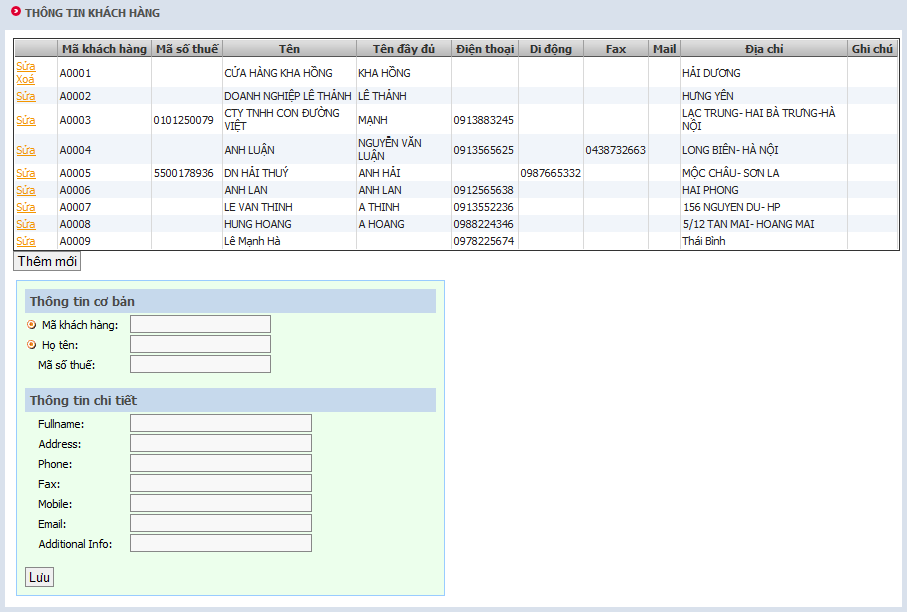


Figure 18: Browse/Create Customer data

#### Function description

This function used to maintain importance customers for dealer, and then can be reference by Part sale function.

#### Data flow

#### 

Figure 19: Customer data flow

#### Instructions of fields

The customer will be saved to table V2\_P\_CUSTOMER

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Code** | Customer identifier, stored in CODE field. |
| **Name** | Name of customer, stored in NAME field. |

And table V2\_P\_CONTACT

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Full name** | Full name of customer, stored in FULL\_NAME field. |
| **Address** | Customer address, stored in ADDRESS field. |
| **Phone** | Phone number, stored in PHONE field. |
| **Email** | Email address, stored in EMAIL field. |
| **Additional info** | Other information, stored in ADDITIONAL\_INFO field. |

### Vendor List

#### Function screen

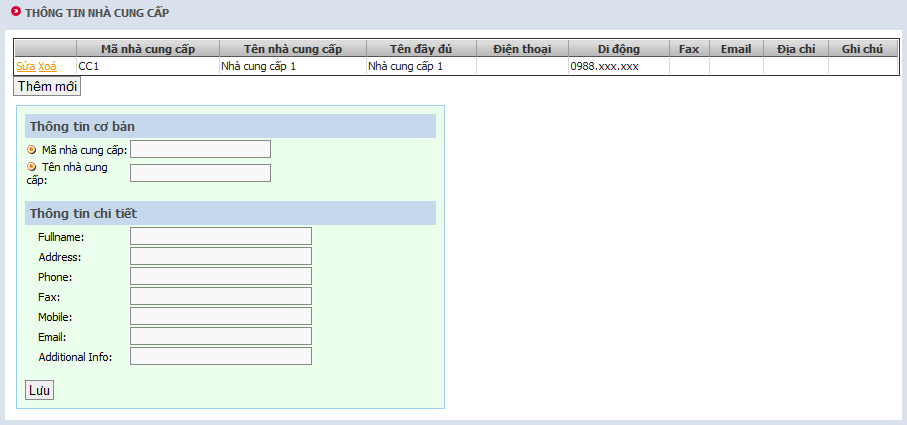


Figure 20: Browse/Create new vendor

#### Function description

Maintain list of vendor for dealer, these information will be used in “[Special import/export](#_Special_Import/Export)” function.

#### Function constrains

* + This function used only by dealer users, dealer administrators or system administrators with dealer role.
  + These vendors belong to individual dealer so vendor code cannot be duplicated in the same dealer: vendor Code + dealer code are unique.
  + One vendor can be deleted only if there are no references by using special import function.
  + Vendor list are shown on grid and paged with page site is 10, delete function placed with each row.
  + Creating new vendor:
    1. Vendor code and vendor name cannot be empty.
    2. If any contact information entered, create new contact and link it to new vendor otherwise leave it as null.

#### Data flow



Figure 21: Vendor Data flow

#### Instructions of fields

The vendor will be saved to table V2\_P\_VENDOR and V2\_P\_CONTACT (see [Customer setting](#_Customer_Setting) for more information)

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Code** | Vendor identifier, stored in CODE field. |
| **Name** | Vendor name, stored in NAME field. |

### Part Replacing

#### Function screen

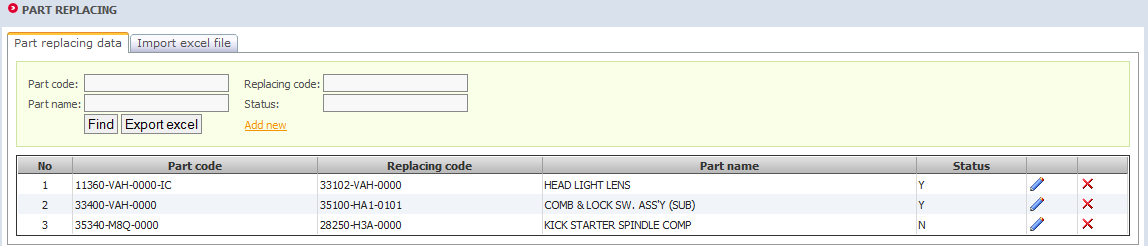


Figure 22: Part replacing screen

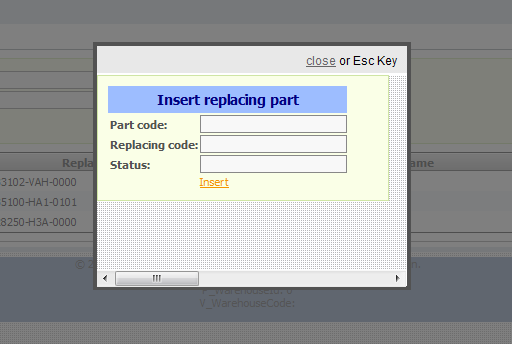


Figure 23: Add new replacing part

#### Function description

Manage the part replacing rules in case user order the part which was stoped, VDMS allows dealer to select the replacing part.

#### Function constraints

Part code must be appeared in ima\_file in order to add a new replacing part rule.

#### Data flow

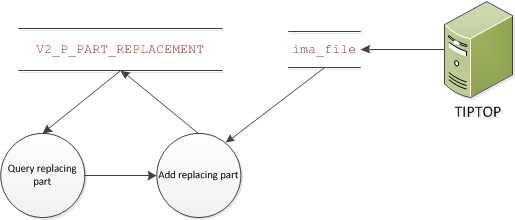


Figure 24: Part replacing data flow

#### Instruction of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part code** | Replaced code, stored in PART\_CODE |
| **Replacing part code** | Replacing code, stored in REPLACE\_PART\_CODE |
| **Part name** | Name of replacing code |
| **Status** | Flag that this rule is affected or not, stored in STATUS |

### Part Specification

#### Function screen

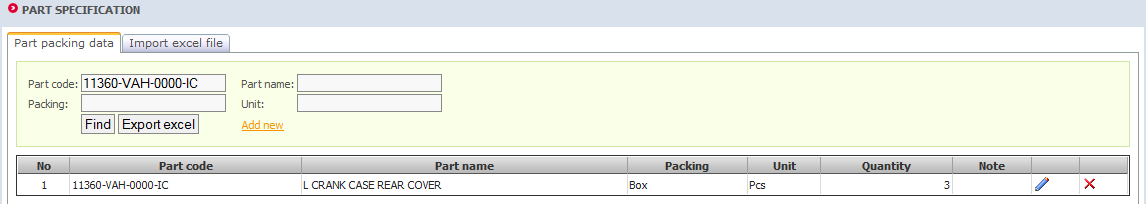


Figure 25: Part specification screen

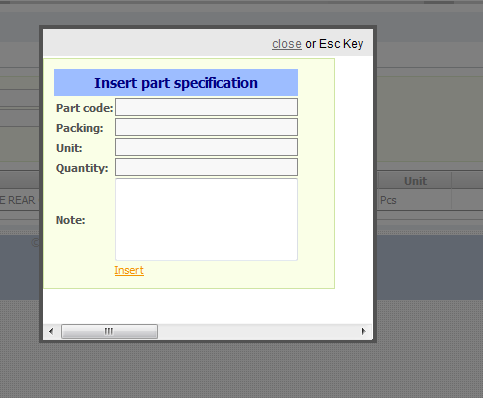


Figure 26: Add new part specification

#### Function description

For the part parking with Box, Set… (for examble: Oil, gasket, Bugi…), If quantity of dealer request do not enough a Box, Set, VDMS will round up to next unit of Box, Set.

#### Function constraints

Part code must be appeared in ima\_file in order to add a new part specification rule.

#### Data flow

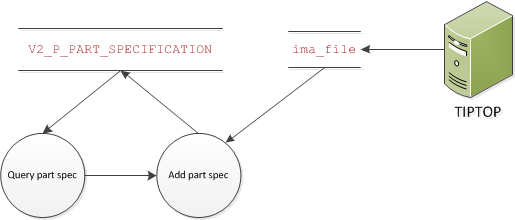


Figure 27: Part specification data flow

#### Instruction of fields

|  |  |
| --- | --- |
| Field name | Field description |
| Part code | Part spec code, stored in PART\_CODE |
| Part name | Part name |
| Packing | How will part be packed (Box, Bag…), stored in PACK\_BY |
| Unit | Unit of each part, stored in PACK\_UNIT |
| Quantity | Quantity of parts in pack, stored in PACK\_QUANTITY |
| Note | Specification note, stored in SPEC\_NOTE |

### Define Excel Structure

#### Function screen

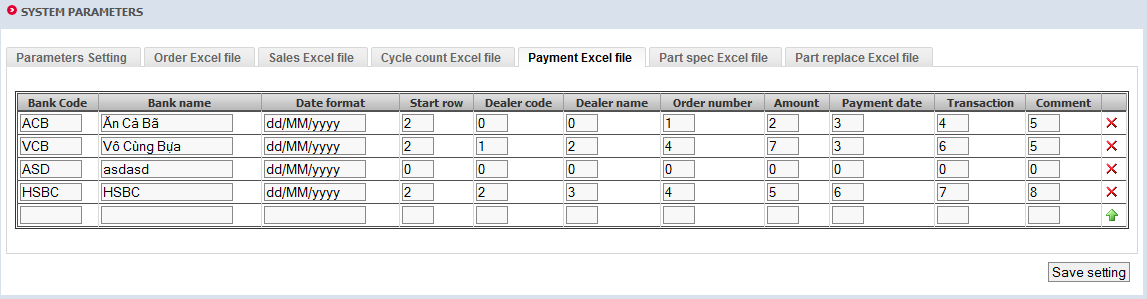


Figure 28: Excel structure screen

#### Function description

Define the structure of each type of importing excel file, including Order Excel file, Sales Excel file, Cycle count Excel file, Payment Excel file, Part spec Excel file, Part replace Excel file.

#### Instruction of fields

|  |  |
| --- | --- |
| Field name | Field description |
| Start row | The row which the data begins in excel file, usually comes after the header |
| Data column | Specify what data contains in each column |

### Account Management

#### Function screen

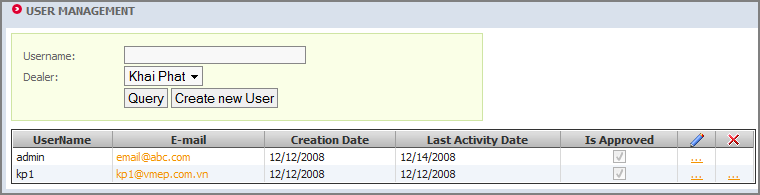


Figure 29: Search users

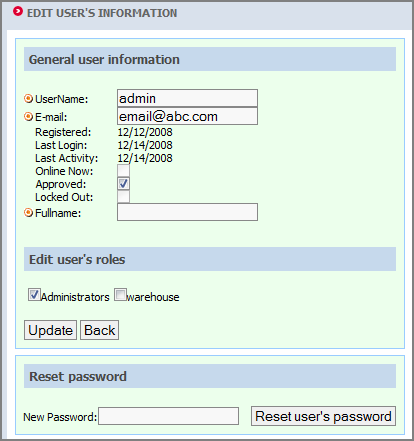


Figure 30: Edit existing user

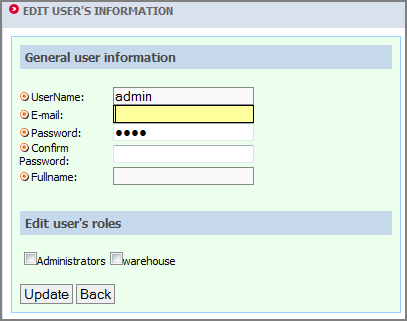


Figure 31: Create new user

#### Function description

Maintain all users in system. Information about user includes username, password, roles.

#### Instructions of fields

All data are managed by .net like VDMS-I.

### Role Management

#### Function screen

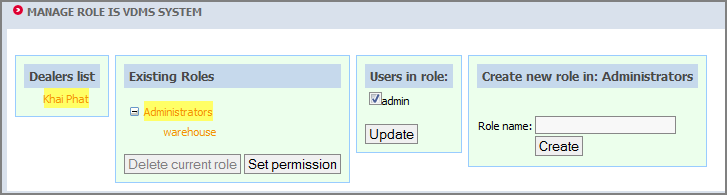


Figure 32: Manage roles

#### Function description

This function used to manage roles in system. Roles are in hierarchy and one organization always has a role as top level called “Administrators”. When selected one role, users in role will be displayed and then can be removed them from role. From here user also can set permission for selected role.

Role can be deleted only when it has no sub role and no user in role.

#### Instructions of fields

All data are managed by .net like VDMS-I.

### Right Setup

#### Function screen

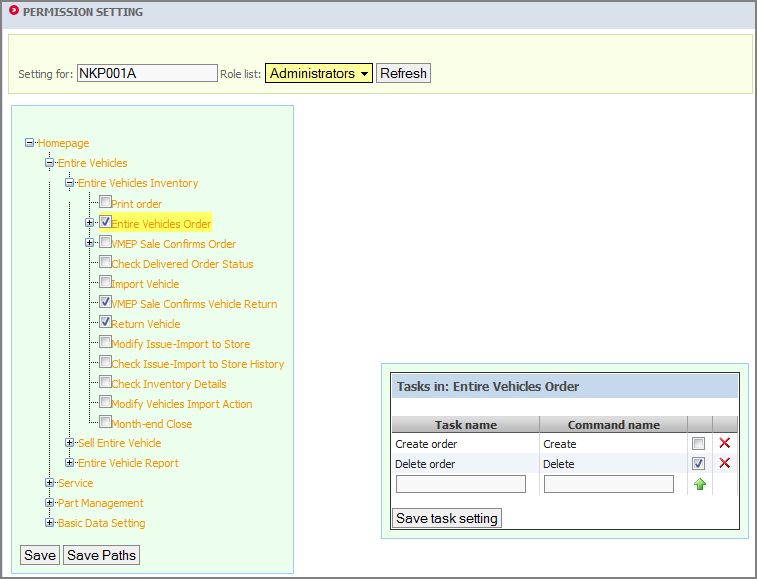


Figure 33: Edit permissions setting

#### Function description

From here user can set permission for roles in his “organization”. If user is administrator of dealer, he can only set permission for all other role in his dealer and all “Administrators” of all sub dealers.

User can turn on access permission to one function if he can access this function by himself.

Only system Administrator can create new task for a webpage.

#### Data flow

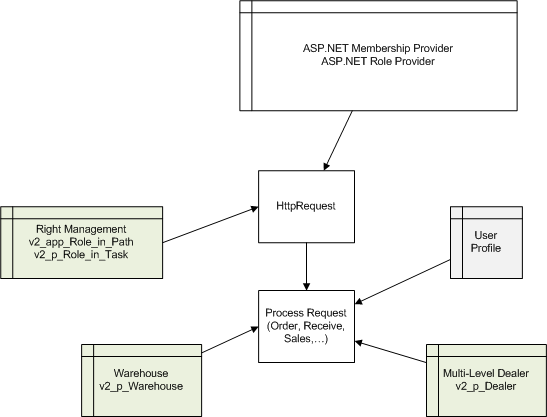


Figure 34: Right Management

* Membership check the authentication: username/password, roles.
* Depend of roles, system get the **url and function code (task – responbility)** that user can access in the table RoleInPath and RoleInTask. If the current url or task not exist, then redirect user to access deny page.

#### Instructions of fields

List of pages are saved to table V2\_APP\_SITEMAP.

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Url** | Url of webpage, saved to URL field. |

List of functions in webpage are saved to table V2\_APP\_TASK.

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Task name** | Task description, reference to TASKNAME field. |
| **Command name** | Command name used on webpage of task, reference to COMMANDNAME field. |

## Main Business

### Cycle Counting Adjust

#### Function screen

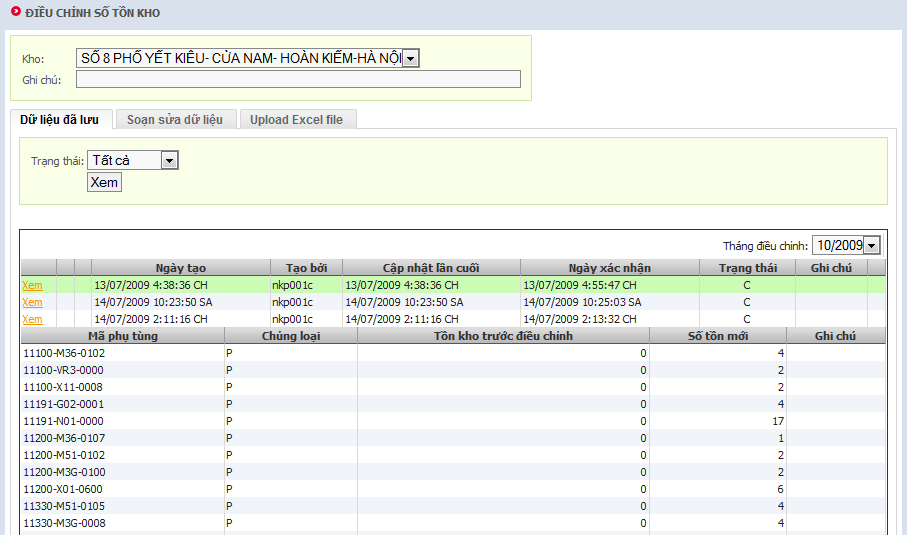


Figure 35: Cycle Count adjust

#### Function description

After period time (ex 6/12 month) user use this function to adjust in stock part quantity.

The data of cycle count can be key-in by user directly or by upload an excel file

System provides current inventory quantity before do cycle count (for user confirm).

Cycle count process: Save new qty 🡪 Confirm 🡪 Update new inventory of current month, and add the log to transaction table

#### Function constrains

* Can only adjust in the current month
* Excel file can be exported from query result
* Search parts:
  + Join PartInfos with PartSafeties on PartCode
  + Return new list objects with these information:
    - PartCode
    - CategoryId
    - CurrentStock (on hand quantity)
    - SafetyQuantity
    - Part (part object)
    - PartName (select English or Vietnamese based on current language)
* Stock adjust function: below function in VDMS.II.PartManagement.PartDAO to adjust quantity:

public static void StockAdjust(string partCode, string dealerCode,   
long wareHouseId, DateTime transDate, string transCode, decimal cost, int qty, string comment, string invNo, long? vendorId)

{

// Check inventory locked or open?

// change last months quantity

// change current stock quantity

// making log: transaction and inventory day action

}

#### Instructions of fields

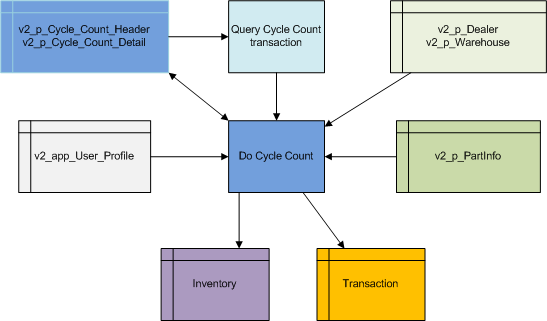


Figure 36: Cycle Count Data flow

Part information placed in table V2\_P\_PART\_INVENTORY, V2\_P\_PART\_INFO and V2\_P\_PART

|  |  |
| --- | --- |
| Field name | Field description |
| Part code | Part code, reference to PART\_CODE field. |
| Type | Part type: P means part, A means accessory. |
| Quantity before Cycle Count | The current quantity in stock |
| New Quantity | The new value of quantity, must greater than 0 |
| Comment | The comment about of this transaction |

### Dealer List

#### Function screen

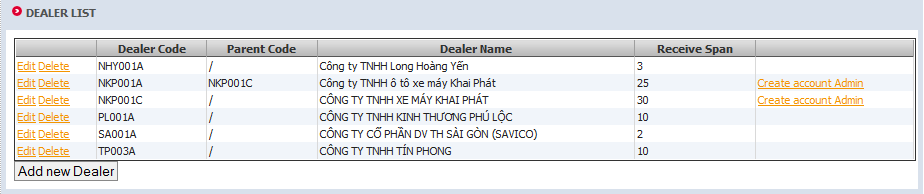


Figure 37: Main screen

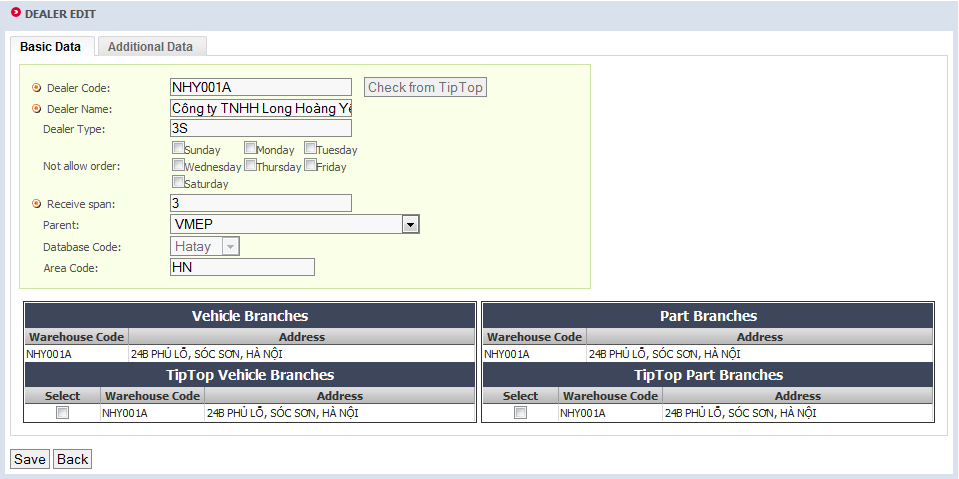


Figure 38: Create/Edit Basic dealer data

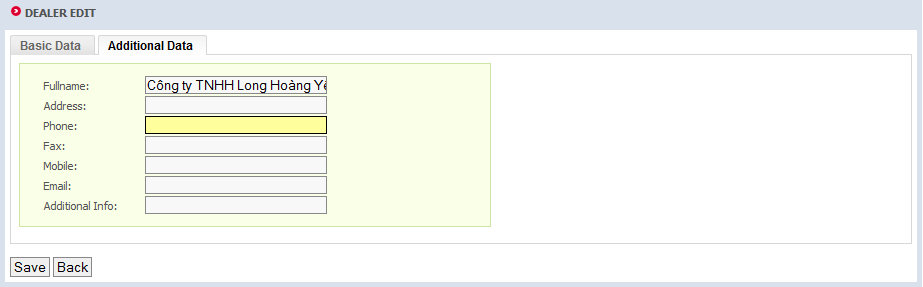


Figure 39: Create/Edit Additional dealer data

#### Function description

VDMS-II copies the basic information of dealer from Tip-Top to its own database. This screen allow user maintain list of dealer in system.

#### Instructions of fields

The dealer will be saved to table V2\_P\_DEALER and V2\_P\_CONTACT (see [Customer setting](#_Customer_Setting) for more information)

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer code** | Dealer code, stored in DEALER\_CODE field. |
| **Dealer name** | Dealer name, stored in DEALER\_NAME field. |
| **Receive span** | Receive time span, stored in RECEIVE\_SPAN field. |
| **Auto in stock span** | Auto in stock time span, stored in AUTO\_IN\_STOCK\_SPAN field. |
| **Parent** | Parent dealer that dealer belong to, reference by PARENT\_CODE field. |
| **Area code** | Area code that dealer belong to, stored in AREA\_CODE field. |
| **Database code** | Database code that dealer belong to, stored in DATABASE\_CODE field. |
| **Order Date Control** | The order date is the integer number that contains a list of 7 bit, in which bit 0 is set means allow dealer order in Sunday, bit 0 not set means do not allow order in Sunday, bit 1 for Monday, and bit 2 for Tuesday… |

### Part sale

#### Function screen

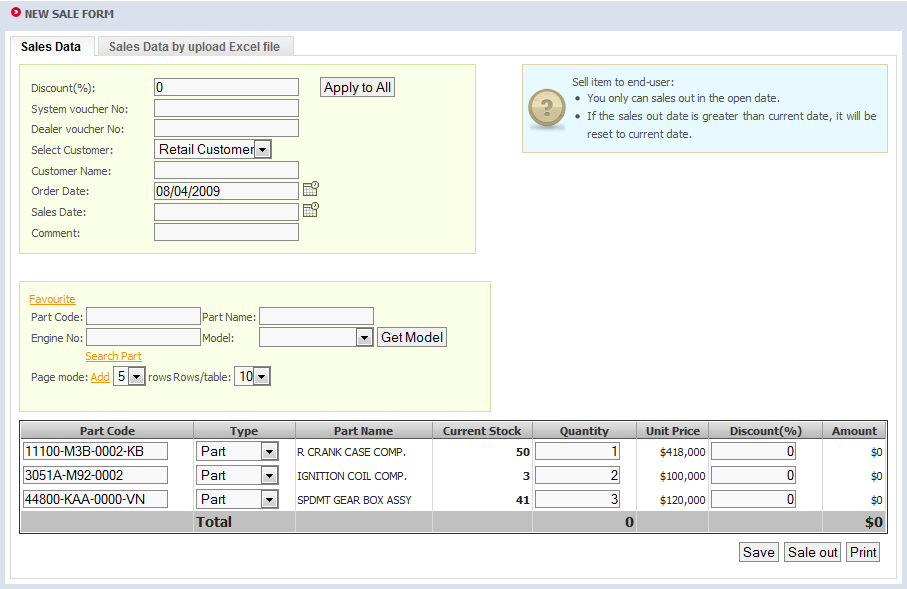


Figure 40: Manual key in data

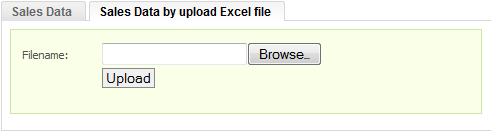


Figure 41: Upload data from excel file

#### Function description

This function used to sell part to customer, part list can be key in by manual or load from uploaded excel file.

#### Function constrains

* User query part and sales part in his owner warehouse.
* Cannot sale part with quantity more than inventory
* Sale data is stored in Excel file using Common/ExcelDataReader function for upload

#### Data flow

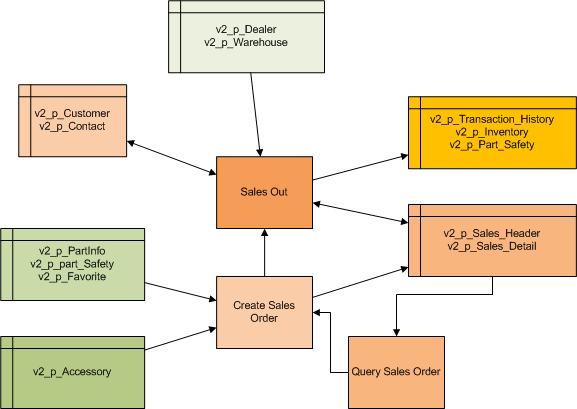


Figure 42: Sales Part data flow

#### Instructions of fields

The sale header will be saved to table V2\_P\_SALES\_HEADER

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Customer data** | Customer information, reference to CUSTOMER\_ID / CUSTOMER\_NAME field. |

The sale detail will be saved to table V2\_P\_SALES\_DETAIL

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part no** | Part code, reference to PartCode field. |
| **Description** | Additional information for part. |
| **Quantity** | Selling quantity, must greater than 0, be saved to OrderQuantity field. |
| **Unit price** | Part price, read-only from Tip-Top, be saved to UnitPrice field. |
| **Discount** | Discount b percent, (from 0 to 100%) saved to PercentDiscount field. |
| **Amount** | Selling amount, calculated automatically by system. |
| **Status** | In SalesHeader, the Status = “OP” mean new order (can edit, sale). “CF” mean sale out |

### Order parts

#### Function screen

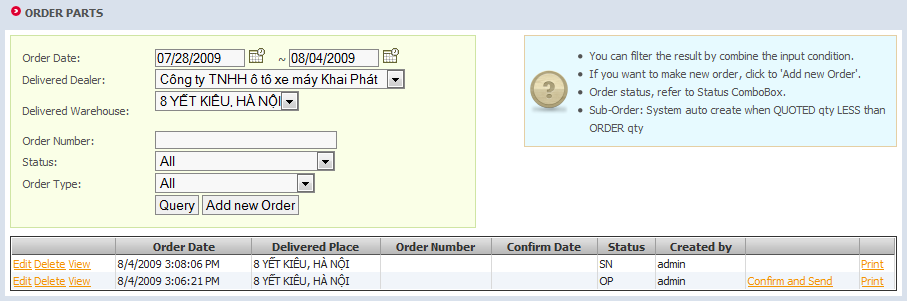


Figure 43: Search orders

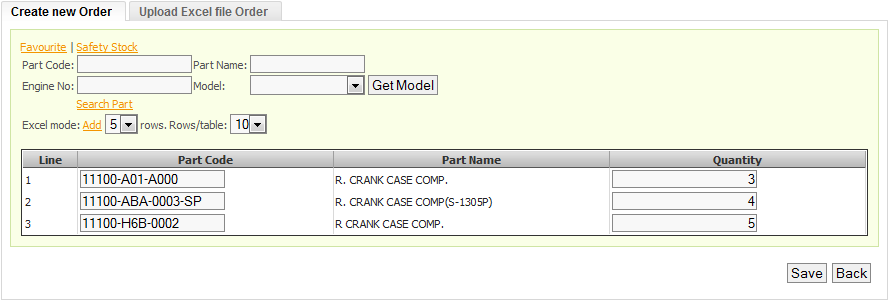


Figure 44: Manual create order

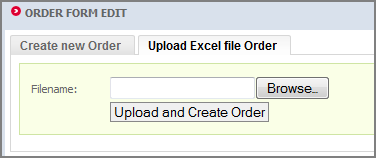


Figure 45: Create order by excel file

#### Function flow



Figure 46: Create New Order Flow

Flow explanations:

1. User send Create New Order request to Order Page.
2. The Order Page check the security to ensure that user has permission to create new order.
3. User set the dealer and warehouse where order delivery to
4. User get the favorite part by call the Favorite Page
5. In Favorite Page, user can toggle the favorite status of part
6. The Favorite Page send back the part list which user selected to Order Page
7. User get the safety stock part (depend on dealer and warehouse in step 2) by call the Safety Stock Page
8. The Safety Stock Page send back the part list which user selected to Order Page
9. User search other part by call the Search Part Page
10. In the Search Part Page, user can change the search condition, and the page will refresh the result
11. The Search Part Page send back the part list which user selected to Order Page
12. To save order, the Order Page create a LINQ and send to Data Context object
13. The Data Context execute query by call the Devart component
14. The Devart component send back the result if any
15. The Data Context send back the order (which saved to database) to Order Page
16. The Order Page refresh the result in the screen



Figure 47: Upload Excel Order

Flow explanations:

1. User send Excel file Order Page.
2. The Order Page check the security to ensure that user has permission to create new order.
3. The Order Page call the Create Excel Order in PartOrderDAO object
4. The PageOrderDAO send back the order (now it is a list of part in the excel file) to Order Page
5. The Order Page valid part by checking each part in order
6. If there is not any error, the Order Page create a LINQ and send to Data Context object
7. The Data Context execute query by call the Devart component
8. The Devart component send back the result if any
9. The Data Context send back the order (which saved to database) to Order Page
10. The Order Page refresh the result in the screen

#### Function description

Use this function to maintain part order. At the beginning, list all order from database (which belong to current user) and display to screen. After create new order user can edit, delete or send to VMEP. When create order, user can key in data by manual or auto load by upload excel file. Order type always is “Normal” include sub order. Order processing has needed to “communicate” with TIPTOP see [Tip-Top Interfaces](#_Tip-Top_Interfaces) for more detail.

When create new order, if the part was stoped, VDMS show the button allow dealer select the replacing part and then fill ne replacing part in the part code textbox

For the part parking with Box, Set… (for examble: Oil, gasket, Bugi…), If quantity of dealer request do not enough a Box, Set, VDMS will round up to next unit of Box, Set.

Examble: A box of Oil contain 24 bottles of Oil, if dealer order 25 bottles, VDMS will rounding the Qty to 48 (2box)

#### Function constrains

* For upload excel, use the class ExcelDataReader to read excel file and get the result as a dataset. After that, take the start row, part code column, quantity column and model column form the Setting class and read the part list as order body.
* For the Page mode:
  + The grid allow user edit in all rows
  + When user switch page, the data in current page will be saved to server’s memory cache, so the system can query faster and more stable.
  + When user save the order, all the page (that means all the part) are saved to database at the same time.
* For distingue the order created by VDMS and order created Tip-Top, use the OrderSource field in OrderHeader, in there “T” mean order created by Tip-Top and “V” by VDMS.
* The order status:
  + Open (OP): is default when dealer create new order
  + Save (SA): After users create orders saved Order at the dealer, not send to VMEP.
  + Sent (SE): Order is sent to VMEP
  + Confirmed (CF): Order is confirmed by VMEP
  + Closed (NC):

\* Quote Qty = Issues Qty (normal receive)

\* After sent <NG> (abnormal receive)

* VDMS Order number: using sequence for create VDMS Order number. This number is passed to interface file to synchronize between VDMS and Tip-Top.
* Type of order: in OrderType field, “N” means normal, “S” means Sub-Order and “O” means other.
* Anytime, when order data is change, it recorded into History column (who change, What change, When change)
* In the order detail, part code cannot duplicate: if user key-in duplicate part code, then system must re-calculate by sum the quantity group by part code

#### Data flow

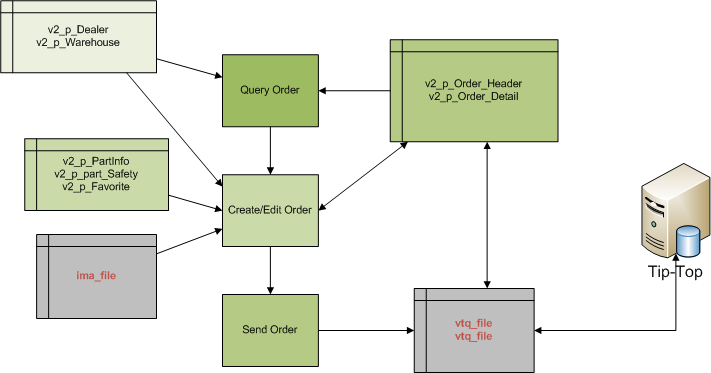


Figure 48: Order Part Data flow

#### Instructions of fields

The order header will be saved to table V2\_P\_ORDER\_HEADER

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Delivered place** | Place to ship parts, saved to WAREHOUSEID field |
| **Order date** | Date of order, saved to ORDER\_DATE field. |

The order detail will be saved to table V2\_P\_ORDER\_DETAIL

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Line** | Line sequence, saved to LINE\_NUMBER field. |
| **Part no** | Part code, reference to PART\_CODE field. |
| **Description** | Part description |
| **Quantity** | Order quantity, must greater than 0, be saved to ORDER\_QUANTITY field. |

### Order List and Receive

#### Function screen

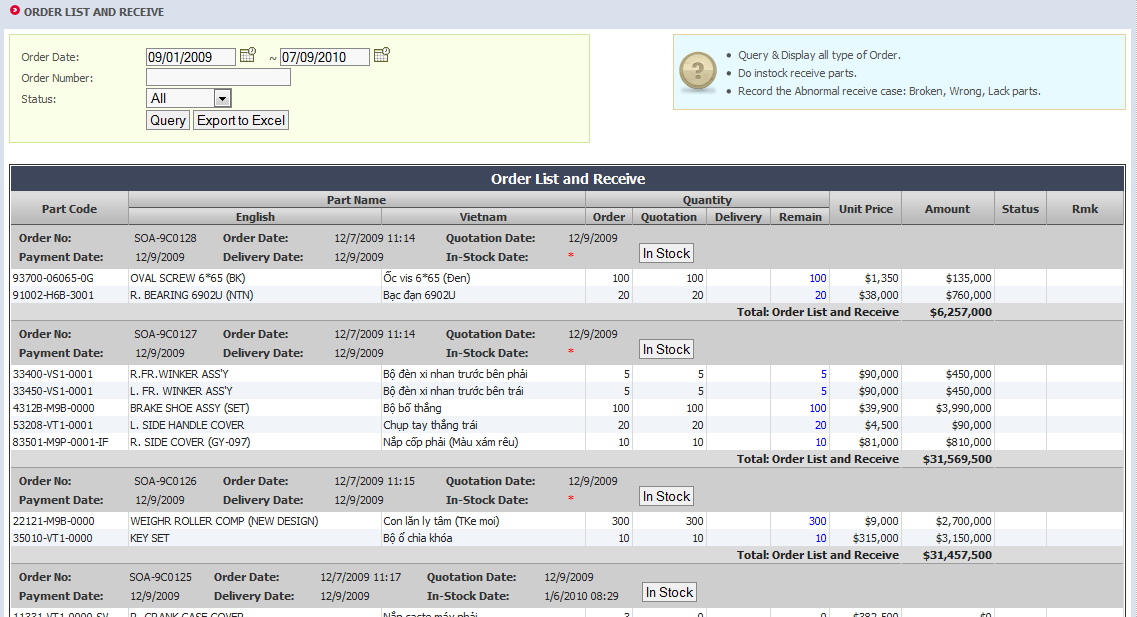


Figure 49: Search screen

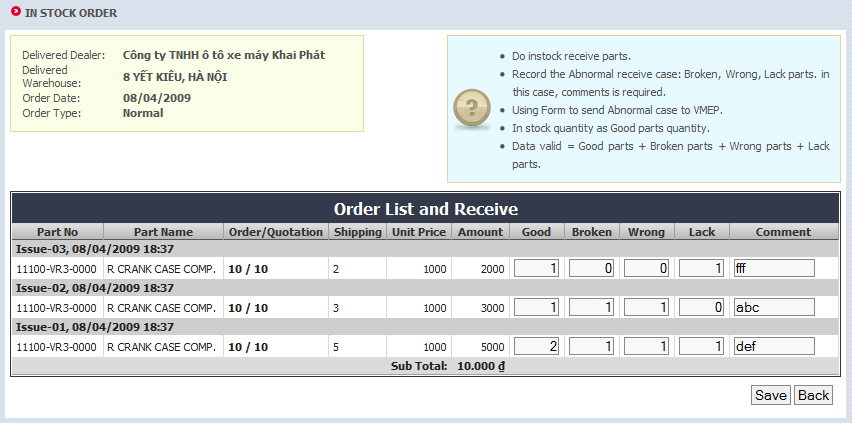


Figure 50: Update in stock/lack quantity

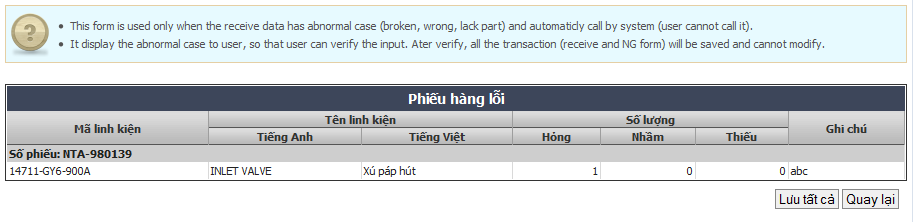


Figure 51: Auto NG form

#### Function description

Listing order and processing receiving action.

* In the query screen, looking all the orders and their descriptions: Order, Quotation, Delivery and Remain Quantity, which are belong to the current dealer and warehouse. For VMEP, only display data with read-only property. For dealer, if the order has shipping data, display the “In Stock” button
* Receiving:
  + Collect the data from the grid. Check the quantity follows the equation: Quotation = Good Quantity + Broken Quantity + Wrong Quantity + Lack Quantity.
  + For first time processing receive issue, before save data, check for any part code will be receive but did not exist in parts list of receiving dealer and create new part information about this part code for receiving dealer.
  + In stock only the Good Quantity. If a new part do in stock, then save new information into table PartInfor: it can be done in StockAdjust() function.
* Auto NG form
  + After receiving, if any NG item (Good Quantity <> Quotation Quantity), then system automatically collect data and transfer to auto NG form.
  + This form display only data of Broken, Wrong, Lack quantity (user cannot edit), but user can change the receive data by click to “Back” button. Otherwise, user save all data (receive data and NG data) to database.

#### Data flow

#### 

Figure 52: Receive Part Data flow

#### Instructions of fields

The order header has been referenced to table V2\_P\_ORDER\_HEADER

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Order no** | Order number, reference to TIP\_TOP\_NUMBER field. |
| **Order date** | Date of order, reference to ORDER\_DATE field. |
| **Quotation date** | Date of quotation, reference to QUOTATION\_DATE field. |
| **Payment date** | Date of payment, reference to PAYMENT\_DATE field. |
| **Shipping date** | Date of shipping, reference to SHIPPING\_DATE field. |
| **Change remark** | Change remark, reference to CHANGE\_REMARK field. |
| **Status** | Status query in OrderHeader table. Close mean Status = NC (Normal Close) or AC (Abnormal Close). Open mean Status = CF (Order Confirm) or RO (ReOpen). All mean Close and Open |

The receive header will be saved to table V2\_P\_RECEIVE\_HEADER and receive detail will be saved to table V2\_P\_RECEIVE\_DETAIL. Note that **Good + Broken + Wrong + Lack = Quotation Qty and all quantity value must be greater than 0**

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part no** | Part code, reference to PART\_CODE field. |
| **Part name** | Part code, reference to PART\_CODE field in V2\_P\_PART table. |
| **Order** | Order quantity, reference to ORDER\_QUANTITY field in V2\_P\_ORDER\_DETAIL table. |
| **Quotation** | Quotation quantity, reference to QUOTATION\_QUANTITY field. |
| **Unit price** | Part price, read-only, reference to UNIT\_PRICE field in V2\_P\_ORDER\_DETAIL table. |
| **Amount** | Part amount, calculated by system. |
| **Good** | Good parts, saved to GOOD field. |
| **Broken** | Broken parts, saved to BROKEN field. |
| **Wrong** | Wrong parts, saved to WRONG field. |
| **Lack** | Lacking parts, saved to LACK field. |
| **Comment** | Dealer comment, reference to DEALER\_COMMENT field. |

In stock value of imported parts will be updated to table p\_PartInventory

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Order no** | Part code, reference to PART\_CODE field. |
| **Quantity** | Part quantity, updated to QUANTITY field. |
|  | Update to record that MONTH and YEAR field are matching with import date. |

After receive, if any NG case is found, go to NG screen and save the NG data.

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part no** | Part code, save to Part\_Code field. |
| **Description** | Part description. |
| **Return quantity** | Part quantity, saved to Quantity field. |
| **Comment** | Comment about abnormal part; include dealer and VMEP comment, saved to DealerComment and VMEPComment field. |
|  | Part statuses are distinguished by PART\_STATUS field.   * B: broken * W: wrong * L: lack |

### Undo Auto Receive

#### Function screen

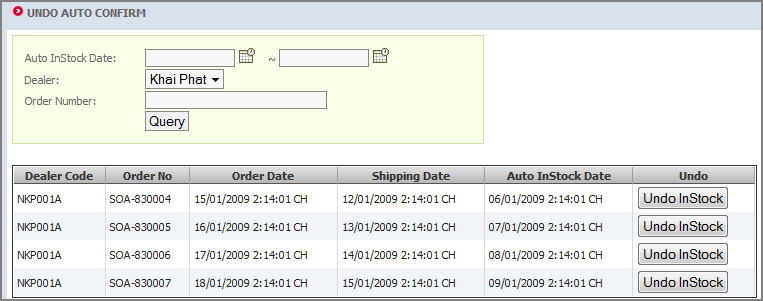


Figure 53: Undo auto confirm

#### Function description

When receive span expired and order was auto imported, sale use this function to undo import action.

If after VMEP has UNDO operation automatically import, the system will be tested in the next 24h, if over this time that dealer still continue not to import part into the stock, the system automatically re-import the stock again and this time Sales cannot restore with no limit times.

#### Instructions of fields

The order information has been referenced to table V2\_P\_ORDER\_HEADER ([see more detail](#_Instructions_of_fields))

When undo auto receive system create transaction (see [p\_TransactionHistory](#_Table_p_TransactionHistory)), change in stock quantity value by update to table [p\_PartInventory](#_Table_p_PartInventory) and mark receive that has been undo ([see more detail](#_Table_p_Receive_Header)).

### Special Import/Export

#### Function screen

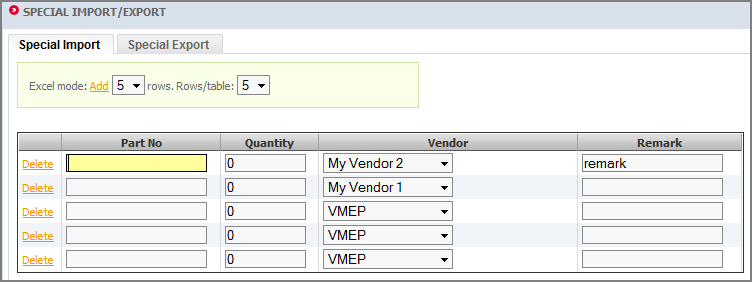


Figure 54: Special Import/Export

#### Function description

Using for special cases: buy part from other vendor (not VMEP), exchange part with issues (special import), and return part to VMEP (special export)

#### Function constrains

* Parts searching:
  + Using three popup: Favorite, Safety stock and Search part (with for condition: part code, part name, engine no, model) for searching, selected parts on popup will be saved to session variable on server side and then add to grid.
* Data saving:
  + Part code not null, quantity must greater than 0 for valid part when import/export. If part no is empty corresponding row will be removed.
  + Use VDMS.II.PartManagement.PartDAO.StockAdjust() function to make change to inventory and create transaction with logging (see Cycle Counting Adjust for more information about StockAdjust() function).

#### Data flow

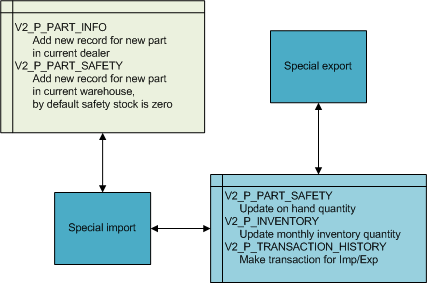


Figure 55: Special Import/Export Data flow

#### Instructions of fields

Special import/export transaction will be saved to table p\_TransactionHistory.

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part no** | Part code, saved to PartCode field. |
| **Quantity** | Part quantity, saved to Quantity field. |
| **Vendor** | Vendor that supply part, reference to VendorId field. |
| **Remark** | Comment on each part, saved to Comment field. |

And then in stock value will be updated to table V2\_P\_PART\_INVENTORY ([more detail](#update_p_PartInventory))

### Special Import/Export history

#### Function screen

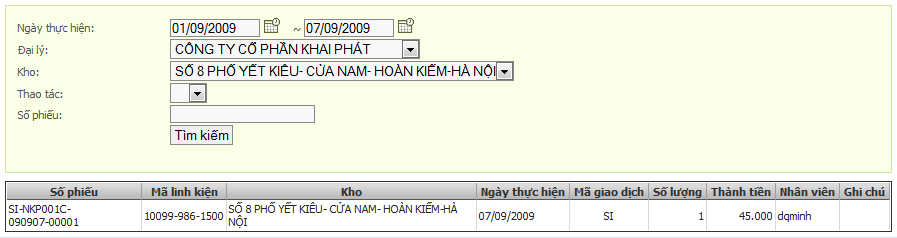


Figure 56: Special Import/Export history

#### Function description

This function allows dealer query the history about the special import/export transaction.

#### Function constrains

No special constrains, only follow the condition key-in by user.

#### Data flow

See the Special Import/Export data flow.

#### Instructions of fields

See the Special Import/Export instruction of field.

### Not Good Form

#### Function screen

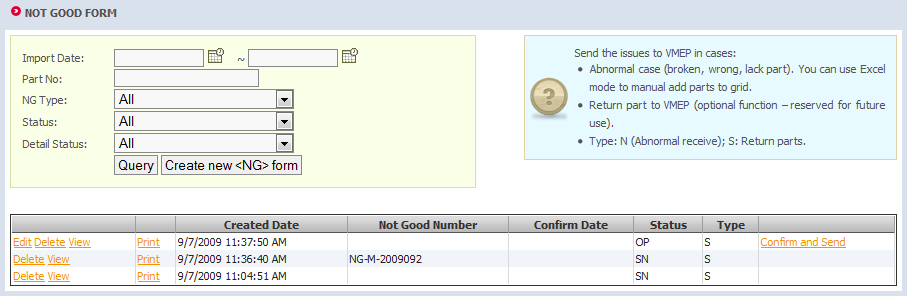


Figure 57: Search NG form

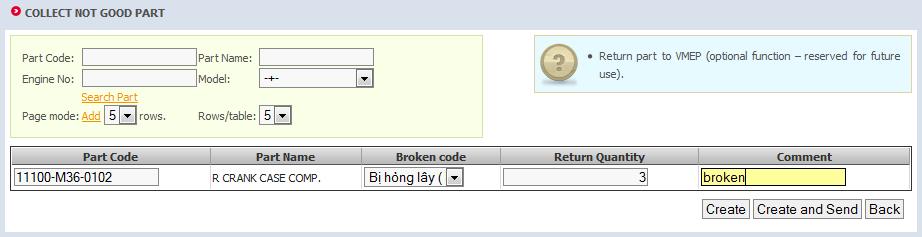


Figure 58: Create NG form manual

#### Function description

* Purpose:
  + Query abnormal receive issues (broken, wrong, lack part) to VMEP
  + Return part (change part, warranty part) to VMEP
* Change:
  + The NG Form automatically created by Receive function, this form only creates the return part list.
* The <NG> form number is create by system follow format: NG-M+yyyyMM+seq

#### Data flow

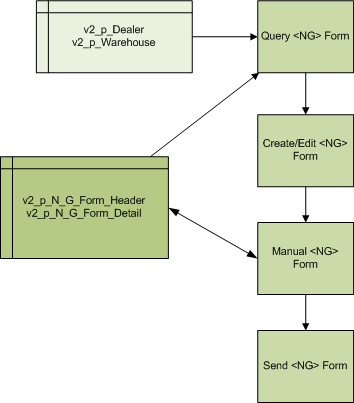


Figure 59: NG Form Data flow

#### Instructions of fields

In this screen, only the NG in abnormal case is created. The NG items will be saved to table NGFormDetail. When creating NG form by issues, Issues were searched from receive data (see [Order List and Receive](#_Order_List_and) for more detail).

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part no** | Part code, save to Part\_Code field. |
| **Description** | Part description. |
| **Return quantity** | Part quantity, saved to Quantity field. |
| **Comment** | Comment about abnormal part; include dealer and VMEP comment, saved to DealerComment and VMEPComment field. |
|  | Part statuses are distinguished by PART\_STATUS field.   * B: broken * W: wrong * L: lack |

### Not Good Confirm

#### Function screen

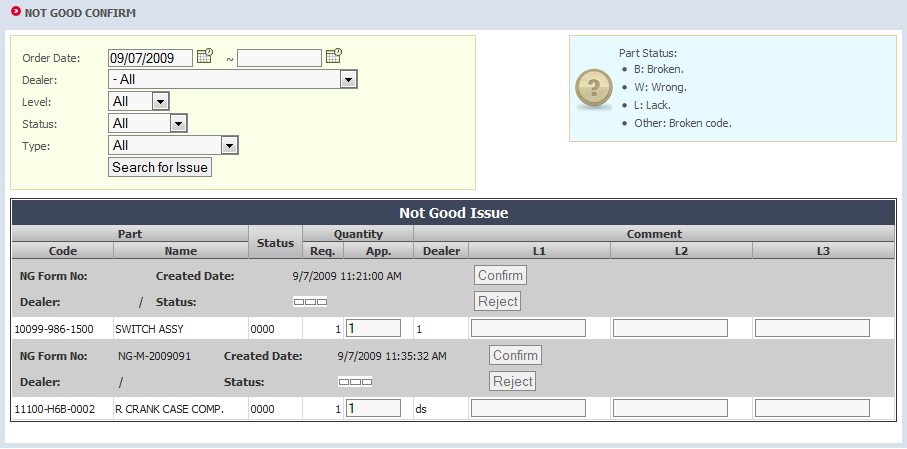


Figure 60: Confirm NG Form

#### Function description

Display all NG form has been send by dealers to confirm, and confirm or reject the request.

#### Function constrains

In this function, we have the confirm flow, in that, one user must has higher level if he want to reject/confirm a form. This level stored in UserProfile, when display data, depend on value of Status field, and the Dept of user (in UserProfile) that we reject or allow user to do the action.

#### Data flow

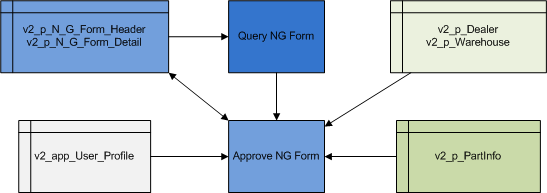


Figure 61: Confirm NG Form Data flow

#### Instructions of fields

The NG items will be saved to table NGFormDetail (see [Not Good Form](#_Not_Good_Form) for more detail). Each value after edited and approved by VMEP will be saved to VMEPApproved field. After confirmed, status of NGFormHeader (field Status) will be changed.

* Status field:
  + OP: open, not send by dealer
  + SN: sent by dealer
  + CF: already confirm by VMEP (only when Approve level = 3)
  + RJ: reject by VMEP (only when Approve level = 0)
* Approve level: 0 to 3, 0 mean by dealer, 1-3: mean by VMEP

### Not Good Form tracking

#### Function screen

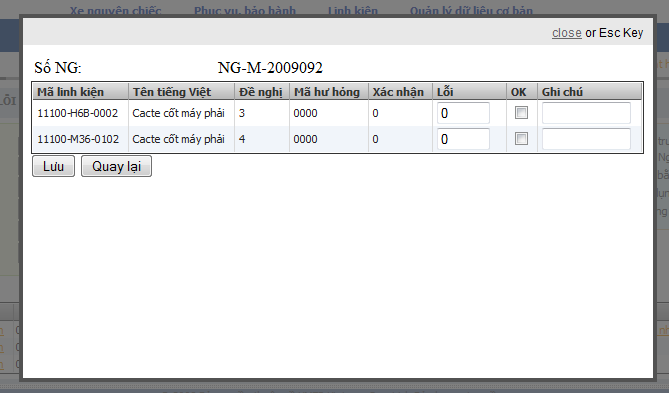


Figure 62: Not Good tracking

#### Function description

This screen shows the detail of a NG form, and allows dealer tracking by check the item is passed or not. Dealer can query and check the NG at any time.

#### Function constrains

In the “Problem again”, means after process the NG form, the items still have problem, and dealer cannot key-in this value greater than origin value.

#### Data flow

See the Not Good from data flow

#### Instructions of fields

The “Problem again” value is saved to ProblemAgainQuantity in table NGFormDetail.

### Stock Transfer

#### Function screen

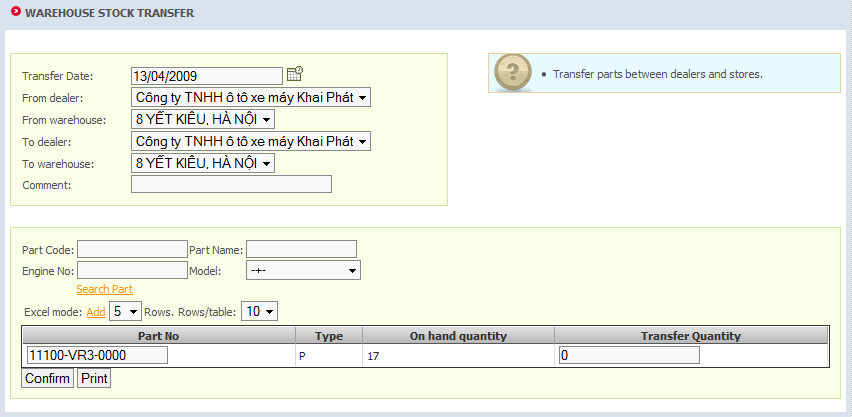


Figure 63: Stock Transfer

#### Function description

This function used to transfer parts from a warehouse to another.

#### Function constrains

* Filter dealer list by current logged in dealer: all other dealer that has relationship in “family tree” with current dealer.
* Parts searching:
  + Using three popup: Favorite, Safety stock and Search part (with for condition: part code, part name, engine no, model) for searching, selected parts on popup will be saved to session variable on server side and then add to grid.
* Data saving:
  + Part no, transfer quantity cannot be empty for valid part when saving. If part no is empty corresponding row will be removed.
  + Use StockAdjust() function to make change to inventory and create transaction with logging (see [Cycle Counting Adjust](#_Cycle_Counting_Adjust) for more information). With each part codes call this function two times, one for source warehouse and one for destination warehouse.

#### Data flow



Figure 64: Stock Transfer Data flow

#### Instructions of fields

Transfer transaction will be save to TransactionHistory table ([more detail](#p_TransactionHistory_detail)), and then update in stock value to PartInventory table ([more detail](#update_p_PartInventory))

### Stock Transfer history

#### Function screen

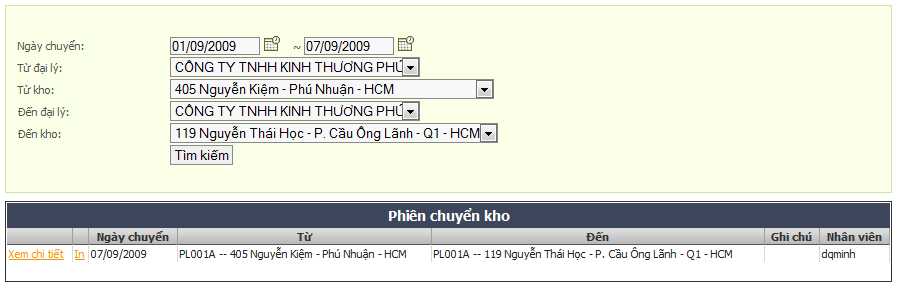


Figure 65: Stock Transfer history

#### Function description

This function allows dealer query the history about the stock transfer import/export transaction.

#### Function constrains

No special constrains, only follow the condition key-in by user.

#### Data flow

See the Stock transfer data flow.

#### Instructions of fields

See the Stock transfer instruction of field.

### Bin Card Query

#### Function screen

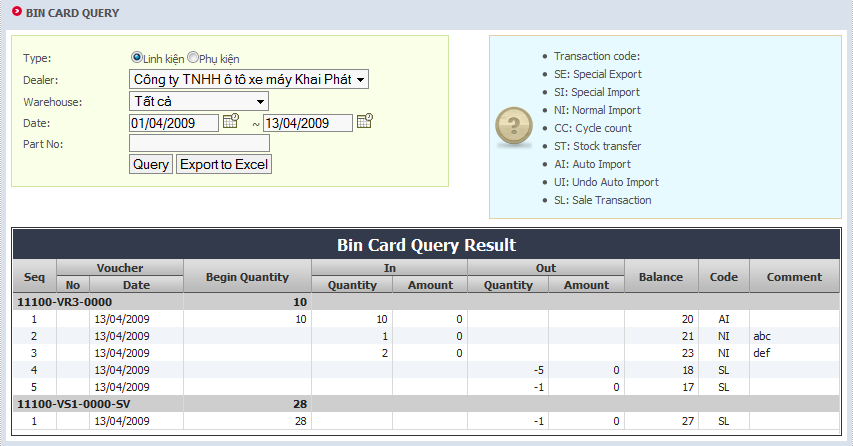


Figure 66: Bin card info

#### Function description

Show all actions (import/export – with quantity, amount) and stock quantity begin and after actions about parts in period order time.

* Balance qty formula:
  + Balance qty = begin qty + in-qty – out-qty.
  + Balance qty (Today) = begin qty (next day).
  + The query date can be no limited (the bigger range of date, the slowly query run).
* Query result is made by T-SQL at Oracle side to improve the speed

#### Data flow

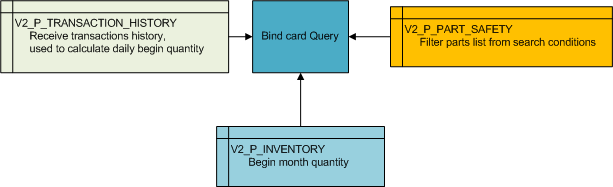


Figure 67: Bin Card Query Data flow

#### Instructions of fields

This is only a report; data are generated from corresponding tables.

### In Short Supply

#### Function screen



Figure 68: In Short Supply

#### Function description

List all in short order (use for dealer checking the part that the quotation less than order).

#### Instructions of fields

The order detail will be saved to table OrderDetail (for other information sees [Order parts](#_Order_parts(change_layout)))

|  |  |
| --- | --- |
| Field name | Field description |
| Part Code | The code of part |
| Part Name(en/vn) | The name of part |
| Order Quantity | The order quantity |
| Quotation quantity | Quotation quantity by VMEP, reference to QuotationQuantity field. |
| Short quantity | Short = (Total Order in origin order) – quotation. |

### Monthly Close

#### Function screen

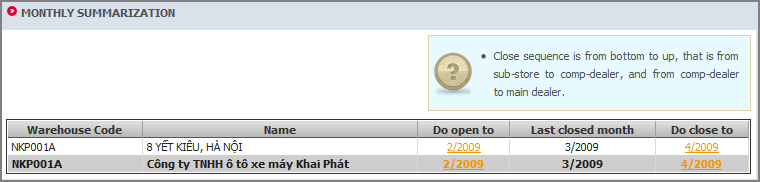


Figure 69: Monthly summarization

#### Function description

Do summarization after month closed, and then inventory will be locked, all actions that make to change inventory are denied.

#### Data flow

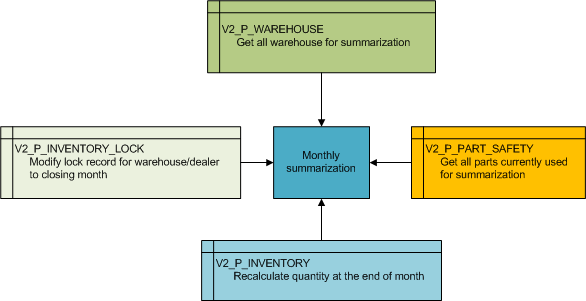


Figure 70: Monthly Close Data flow

#### Instructions of fields

* Warehouses get from V2\_P\_WAREHOUSE table, belong to current logged dealer.
* Parts to be summarization get from V2\_P\_PART\_SAFETY table and belong to closing warehouse/dealer.
* Summary result place to V2\_P\_INVENTORY table (Dealer, Warehouse, Part Infor, Quantity, Month, Year)
* Modify lock record of warehouse/dealer to closing month by insert new record to the table Inventory Lock (Dealer, Warehouse, Year, Month).

#### Constrains

* For the first time do summarization user must choose first month to close and lock inventory, this first month cannot be later than the month that first transaction(that change inventory quantity) happen.
* Open and close action must be step by step and can reopen 2 months from current month (this value can be set in system setting function).
* Dealer can close when all warehouses and sub dealers were closed.

## Report

### In Short supplier report

#### Function screen

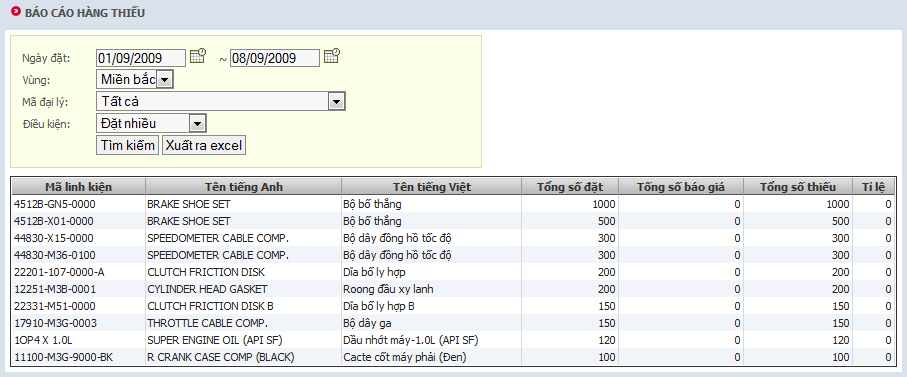


Figure 71: In Short Reason

#### Function description

Query almost in short parts; depend on dealer, area (for VMEP used).

#### Function constrains

* Displays orders that belong to selected dealer or all dealers (based on query condition) and had in short parts only.
* Dealers list filtered by region (HTF/DNF).
* Columns:
  + Total order quantity: The summary of order quantity, group by part code
  + Total quotation quantity: The summary of quotation quantity, group by part code
  + Total short: Total order quantity – Total quotation quantity
  + Rating: Total quotation quantity / Total order quantity

#### Data flow

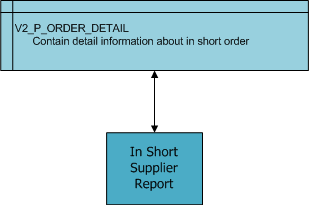


Figure 72: In short query data flow

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part name** | See [Part/Accessory Setting](#_Part/Accessory_Setting) for more detail |
| **Part code** | See [Part/Accessory Setting](#_Part/Accessory_Setting) for more detail |
| **Total order** | Calculated from in short order |
| **Total quotation** | Calculated from in short order |
| **Total short** | Short=(Total Order in origin order)-quotation |
| **Rate** | Total quotation / Total order |

### Abnormal receive report

#### Function screen

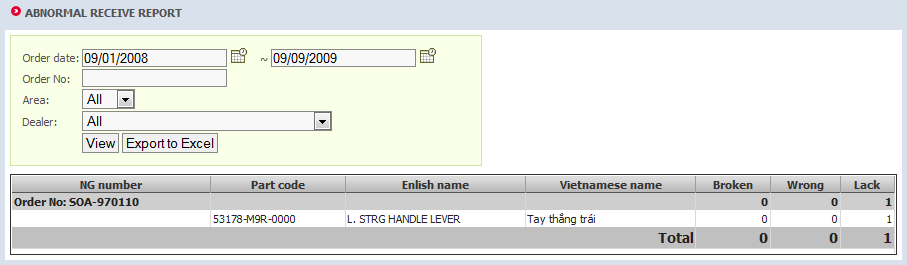


Figure 73: Abnormal receive report

#### Function description

Show the abnormal receive of dealer for VMEP. This abnormal receive (NG form) must be approved by VMEP.

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part name** | The name of part |
| **Part code** | The code of part |
| **Broken** | The broken quantity (in receive data) |
| **Wrong** | The wrong quantity (in receive data) |
| **Lack** | The lack quantity (in receive data) |

### Warranty Return Report

#### Function screen

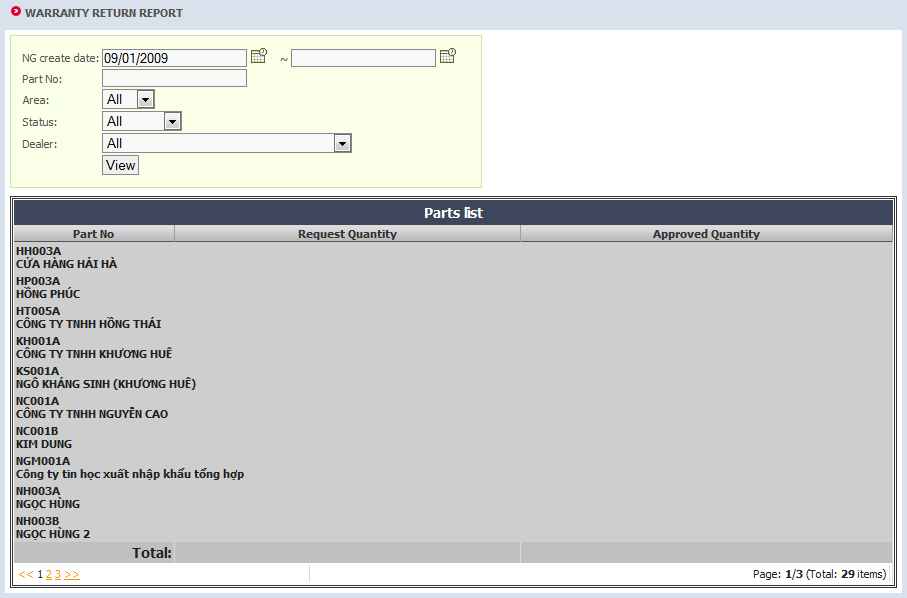


Figure 74: Warranty return report

#### Function description

Show the part return of dealer for VMEP. This abnormal receive (NG form) must be approved by VMEP.

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part name** | The name of part |
| **Part code** | The code of part |
| **Broken** | The broken quantity (in receive data) |
| **Wrong** | The wrong quantity (in receive data) |
| **Lack** | The lack quantity (in receive data) |

### Over-shipping Span report

#### Function screen

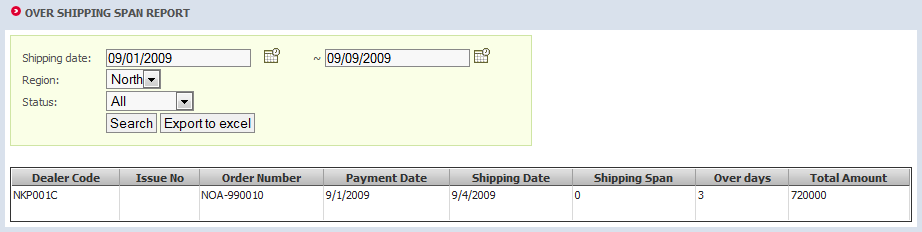


Figure 75: Over-shipping span report

#### Function description

Show the order list that over shipping span (over shipping mean VDMS do not ship goods after shipping span date since has payment time) for dealer.

There are two cases: over shipping but not ship at this time and over shipping but already ship at this time.

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer Code** | The code of dealer |
| **Issue No** | The issue number |
| **Order Number** | The order number |
| **Payment Date** | The date dealer do payment |
| **Shipping Date** | The date VMEP do the shipping |
| **Shipping Span** | Get form system setting |
| **Over days** | The days between Payment date and Shipping Date |
| **Total Amount** | Total amount of order |

### Supply and Service rate report

#### Function screen

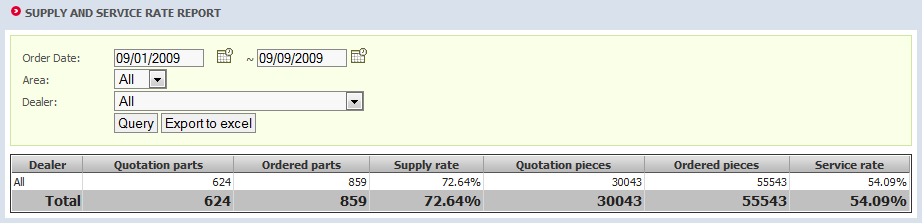


Figure 76: Supply and Service rate report

#### Function description

Show the rating of service (quotation per order) of VMEP.

Service rate = count (part code when shipping > 0 (original order+sub orders)) / count(order part code(original order only))

Service rate = sum (shipping qty (original order+sub orders)) / sum(order qty(original order only)

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer** | Dealer name or all |
| **Quotation part** | The total quotation parts |
| **Order part** | The total order parts (only origin order create by dealer, exclude sub-order) |
| **Supply rate** | Quotation part / Order part |
| **Quotation pieces** | The total quotation pieces |
| **Order pieces** | The total order pieces (only origin order create by dealer, exclude sub-order) |
| **Service rate** | Quotation pieces / Order pieces |

### Sales detail report

#### Function screen

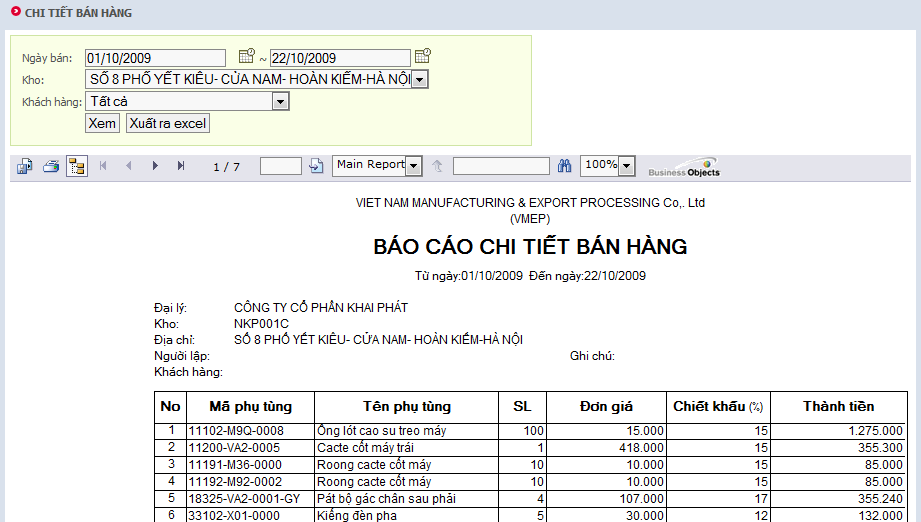


Figure 77: Sales detail report

#### Function description

Show the detail of all sales transaction in a duration time.

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part name** | The name of part |
| **Part code** | The code of part |
| **Quantity** | Get from Quantity column |
| **Price** | Get from Price column |
| **Discount (%)** | Get from Discount column |
| **Total** | System calculate: Quantity \* Price \* (100 – Discount) / 100 |

### Part Inventory report

#### Function screen

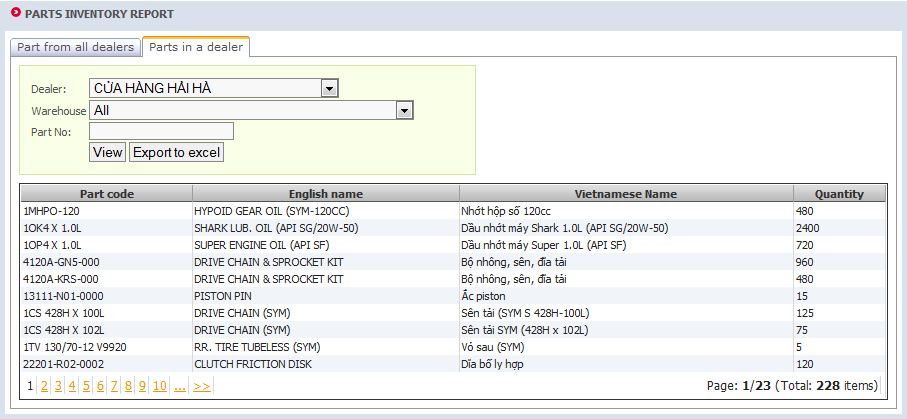


Figure 78: Part Inventory report

#### Function description

This report shows the current inventory of by dealer or by warehouse.

#### Instructions of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Part code** | The code of part |
| **English name** | The name of part in English |
| **Vietnamese name** | The name of part in Vietnamese |
| **Quantity** | The current inventory quantity |

### In/Out/Stock report

Show the all action about In/Out/Stock report, base on transaction history. This report exported to Crystal report as below.

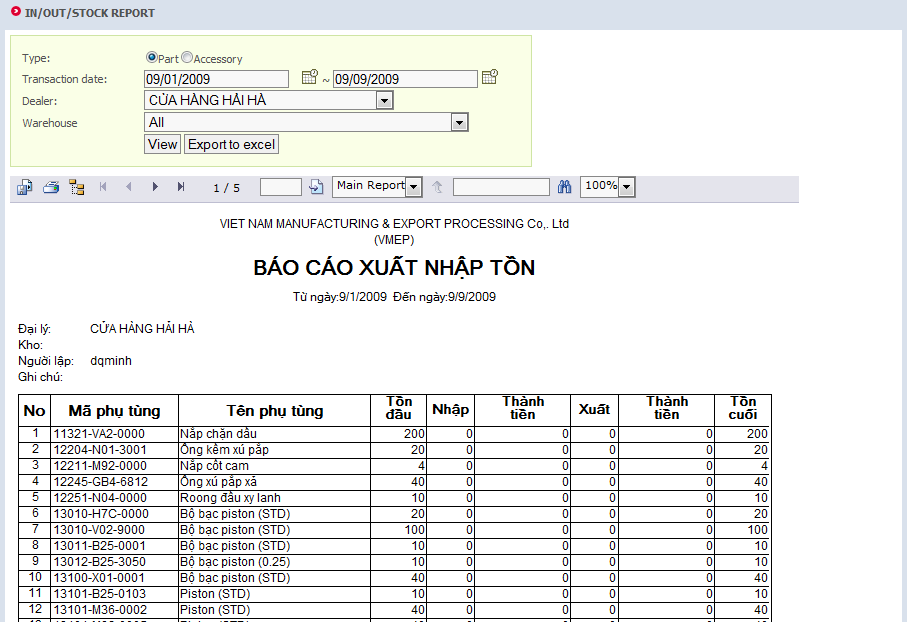


Figure 79: In/Out/Stock report

#### Function description

The report show the detail input/output for a dealer in a duration time.

* Columns:
  + Part code, part name: The information of part
  + BEGIN balance: Calculate by begin balance of the nearest close month with the transaction to the begin report time
  + IN quantity: the summary of import transaction quantity
  + OUT quantity: the summary of export transaction quantity
  + END balance: BEGIN balance + IN quantity – OUT quantity

### Order Detail Report

#### Function Screen

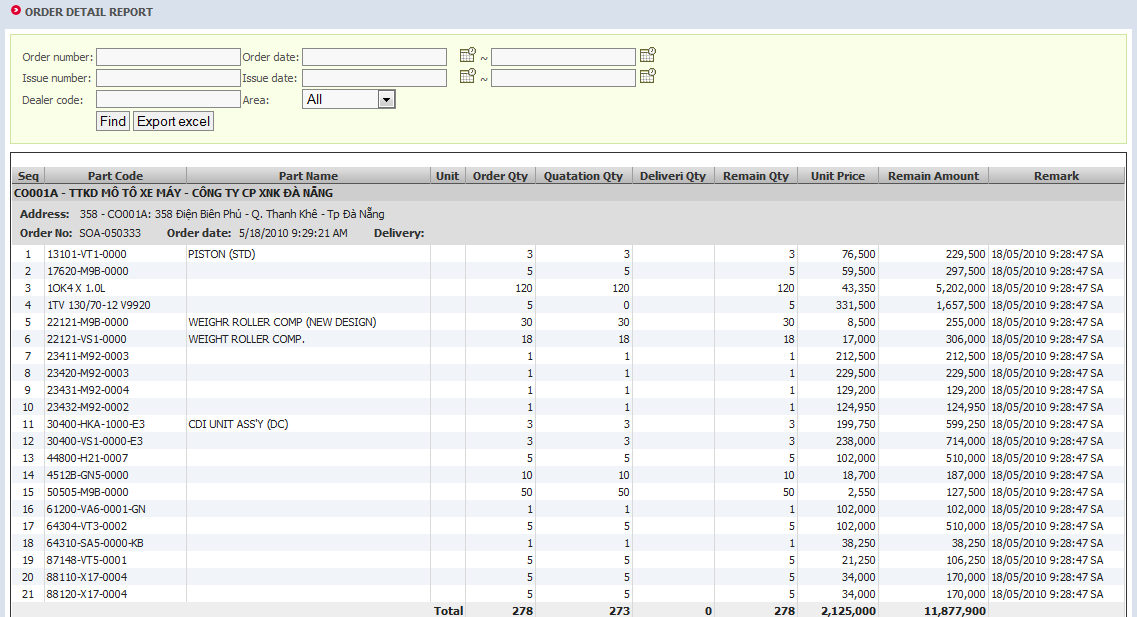


Figure 80: Order Detail Report

#### Function Description

The screen shows details or remaining orders

#### Instruction of fields

|  |  |
| --- | --- |
| Field name | Field description |
| Part code | The code of part |
| Part name | The name of part in English |
| Unit | Counting unit |
| Order quantity | Number of ordered parts |
| Quotation quantity | Quotation quantity |
| Delivery quantity | Delivery quantity |
| Remain quantity | = Order quantity - Delivery quantity |
| Unit price | Price of each unit |
| Remain amount | = Remain quantity \* unit price |
| Remark |  |

# Software Class Description

## Architecturally Significant Design Packages

As talking in Software Architecture Document, the diagram below depicts the logical design packages involved in the system. It splits into 4 main components: the Data Access Layer, the Business Layer, the User Interface Layer and the Shared Layer.

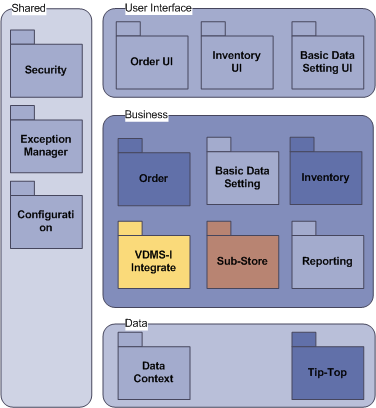


Figure 81: Packages Diagram

## The Data Package

### Data Context

This component contain the classes are mapped to the entities in VDMS’s database using dotConnect for Oracle of Devart.



Figure 82: Domain Component

It contains two areas, the first is VDMS.II.LINQ. This package contains the classes which inherited from DataContext of Devart, allow manipulate the interaction with database (retrieve data, update data, insert data, delete data).

Each table in database will be map to an entity. These entity classes expand their own properties, and override some necessary functions.

### Tip-Top

This component contains the classes which allow querying data from Tip-Top’s database.

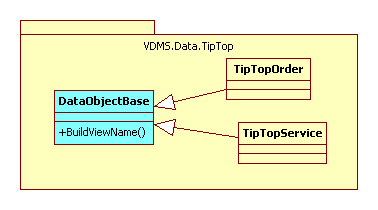


Figure 83: Tip-Top Database Access Component

# Tip-Top Interfaces

## Summary

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Order** | **Trans** | **Add New** | **Qty Confirm** | **Payment Confirm** | **Change** | **Del** | **Recover** | **Delivery** | **Remarks** |
| **Flag** | **(NE)** | **(QU)** | **(CF)** | **(CH)** | **(DE)** | **(RE)** | **(DL)** |  |
| **Manual Order - VDMS** | **M** | x | x | x | x | x | x | x |  |
| **Sub-Order - VDMS** | **S** | x | x | x | x | x | x | x | Can only change and Delete order at Tip-Top side |
| **Others Order - Tip-Top** | **O** | x | x | x | x | x | x | x |  |
| **Abnormal Receive** (sales return) **- Tip-Top** | **R** | x | n/a | n/a | x | x | x | x | Defined as Return group: “R” |
| **Tip-Top get?**  **TT\_Process** | **TT\_Process** | “N” or ”Y” | “Y” | “Y” | “Y” | “Y” | “Y” | “Y” | “Y”: Tip-Top already processed |



## Interface transaction

### The Interface defined

In this case, the interface file is three tables which all VDMS and Tip-Top has full access right.



Figure 84: Tip-Top Interface table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Field name | Type | Property | Description |
| **Table TC\_VDP\_FILE: contains the header of order** | | | | |
| 1 | VDMSOrderNumber | varchar(30) | PK | The primary key |
| 2 | TipTopOrderNumber | Varchar(30) | NULL | At the beginning, it is NULL. After sales process in Tip-Top side, the new value will save here |
| 3 | DealerName | varchar(30) |  | The dealer code |
| 4 | ToDealer | varchar(30) |  | The dealer code that receive order |
| 5 | ToLocation | varchar(250) |  | The warehouse address |
| 6 | OrderDate | datetime |  | The order date |
| 7 | Status | char(2) |  |  |
| **Table TC\_VDQ\_FILE: contains the order detail** | | | | |
| 1 | VDMSOrderNumber | varchar(30) | PK, FK | Refer to table OIH |
| 2 | LineNumber | integer | PK | The line number |
| 3 | PartCode | varchar(30) |  | The part code |
| 4 | OrderQuantity | integer |  | The order quantity |
| 5 | ConfirmQuantity | integer | NULL | The confirm quantity. Before confirm quantity state at Tip-Top, it is NULL |
| 6 | Price | long | NULL | The price of part. Before confirm quantity state at Tip-Top, it is NULL |
| 7 | Status | char(2) |  | The status of item |
| **Table TC\_VDR\_FILE: contain the multi receive data** | | | | |
| 1 | VDMSOrderNumber | varchar(30) | PK, FK | Refer to table OIH |
| 2 | LineNumber | integer | PK | The line number |
| 3 | Issues\_Number | varchar(30) |  | Issue Number |
| 4 | PartCode | varchar(30) |  | The part code |
| 5 | ShippingQuantity | integer |  | The shipping quantity |
| 6 | ShippingDate | Date |  | The date to shipping |
| 7 | Flag | Char (1) |  | The status of item |
| **Table TC\_VDS\_FILE: using for NG form (Send NGNo to Tip-Top)** | | | | |
| **1** | IssueNo | varchar(30) |  | Issue Number |
| **2** | NGNo | varchar(30) |  | Not Good Number |

## New order

1. VDMS 🡪 Tip-Top:
   1. Case 1: Manual Order



* VDMS make new order:
  + VDMS only make VDMS order number
  + Tip-Top Order number is blank
  + OrderType=”M”; Flag = “NE”; TT\_Process = “N”
* When Tip-Top get Order data to process (make order): update TT\_Process = “Y” & and update Tip-Top Order number (dealer cannot change order data)
* Data is transferred to Tip-Top: “**CommonData**” with:
  + OrderType=”M”; Flag = “NE”; TT\_Process = “N”
  1. Case 2: Sub-Order



* + - VDMS automatically make new order
    - Process is same with Manual Order type
    - When Tip-Top get Order data to process: TT\_Process = “Y” & and update Tip-Top Order number (dealer cannot change order data).
    - Data is transferred to Tip-Top: “**CommonData**” with
      * OrderType=”S”; Flag = “NE”; TT\_Process = “N”

1. Tip-Top 🡪 VDMS:
   1. Other Order: Tip-Top provide description for each type
   2. Sales Return Order: Combine into “R-Return” group



* + - Tip-Top make new order: “**CommonData**” with
      * OrderType = “R”
      * OrderType = “O”
      * Flag = “NE”
      * TT\_Process = “Y”
    - When Tip-Top insert data into Tip-Top interface, the INS\_TRIGGER at VDMS side is active: update & synchronized order data by VDMS’s Order No.

## Quantity Confirm

1. Tip-Top 🡪 VDMS:
   1. *Case 1:* Manual Order



* + - Tip-Top update data into Tip-Top interface, the UDP\_TRIGGER at VDMS side is active: update & synchronized order data by VDMS’s Order No.
    - Tip-Top do “quoted” order:
      * OrderType = “M”
      * Flag = “QU”
      * TT\_Process = “Y”
  1. *Case 2:* Sub-Order & Others Order
* Process is the same with Manual Order: only difference flag content
  + OrderType = “S”
  + OrderType = “O”
  + TT\_Process = “Y”
  1. *Case 3:* Sale Return Order: Not available

*Notice:*

1. Tip-Top will add more action: “Send to VDMS - Interface file” – Add more <Send> button on Order screen. Data is transferred to Interface file.
2. Tip-Top does not:
   1. Add more Order line
   2. Delete Order line
   3. Change the Order line

## Payment Confirm

1. Tip-Top 🡪 VDMS:
   1. *Case 1:* Manual Order



* + - Tip-Top do “payment confirm” order: Update payment date
    - When Tip-Top update data into Tip-Top interface, the UPD\_TRIGGER at VDMS side is active: update & synchronized order data by VDMS’s Order No.
    - Data is transferred to VDMS: “PaymentDate” with
      * OrderType = “M”
      * Flag = “CF”
      * TT\_Process = “Y”
  1. *Case 2:* Sub-Order & Others Order
* Process is the same with Manual Order: only difference flag content
  + OrderType = “S”
  + OrderType = “O”
  + Flag = “CF”
  + TT\_Process = “Y”
  1. *Case 3:* Sale Return Order: Not available

*Notice:*

\*Tip-Top add more one field <Payment Date> at Order screen

## Change Order (after quotaton)

* 1. Before Quotation: Define as Quotation phase at Tip-Top side.
  2. After Quotation: Define in Flag Definition:
     1. Tip-Top 🡪 VDMS:



*Case 1:* Manual Order

* + - Tip-Top change order after “do quotatio”
    - Tip-Top change update entire order content and update data into Tip-Top interface, the UDP\_TRIGGER at VDMS side is actived: update & synchronized order data by VDMS’s Order No.
    - Data is transferred to VDMS: Any field can be changed (except: VDMSOrderNo, TipTopOrderNo, OrderLineNo)
      1. OrderType = “M”
      2. Flag = “CH”
      3. TT\_Process = “Y”

*Case 2:* Sub Order & Other type of Order & Sale return:

* + - Process is the same with Manual Order: only difference flag content
      * OrderType = “S”
      * OrderType = “O”
      * OrderType = “R”
      * Flag = “CH”
      * TT\_Process = “Y”
    - *Notice:*

Before sales do quotation at Tip-Top, dealer can change Order in VDMS, after confirm Sales change Order at Tip-Top (using change order function) and send back to VDMS then do re-quotation

## Delete & Recove Order

1. Tip-Top 🡪 VDMS:
   1. *Case 1:* Delete Manual Order



* + - Tip-Top do “delete” order: Do action “VOID” order (do not delete order)
    - VDMS: Reject: Marked as Rejected
    - When Tip-Top update data into Tip-Top interface, the UPD\_TRIGGER at VDMS side is active: Order data in VDMS is marked as “Rejected”
    - Data is transferred to VDMS:
      * OrderType = “M”
      * Flag = “DE”
      * TT\_Process = “Y”
  1. *Case 2:* Tip-Top permanent delete (before confirm). Update flag
     + Flag = “NE”
     + TT\_Process = “N”
  2. *Case 3:* Sub-Order & Others Order & Sale Return Order: Process is the same with Manual Order: only difference flag content
     + 1. OrderType = “S”
       2. OrderType = “O”
       3. OrderType = “R”
       4. Flag = “DE”
       5. TT\_Process = “Y”

1. Recover Order
   1. At Tip-Top: Do “UN VOID” at and transfer Flag = ”RE” to interface file
   2. At VDMS: UDP\_TRIGGER is actived: the order is recover – “Rejected” status is removed (order is read only)

## Delivery Date



1. At tip-Top side: In Delivery screen when confirm the delivery, Tip-Top send the delivery date to interface file
   * 1. Delivery date
     2. Shipping Date = Delivery Date + 3

\*Tip-Top delivery screen need be changed to send delivery date

1. When Tip-Top delivery (multi receive) then insert in to tc\_vdr\_file follow the db design. In one issue the part code cannot is duplicated

## Update new SYM Part

* Tip-Top make a view that map exact the part table
* VDMS only read in this view

# Database Design

## Common Rules

### Naming Rules

In this chapter, we descript the detail of database design. Because the database is used in the same physical schema in Oracle with VDMS-I database, so for separate between them, we have some common rules as below.

* The table name is started v2
* Each component has distinct prefix (after prefix v2)
* The names of field use underscore to separate between each word. In the description below we remove underscore and use Pascal calm standard.

### Default Value Rules

Some field in database has default value, it means that this value is setting default in the input screen, and user needn’t to change this value (but they can change if necessary)

* Created Date: default is current date
* Created By: default is name of current user
* DealerCode: default is code of current dealer (user cannot change)
* WarehouseId: default is Id of current warehouse (user cannot change)

## UMS – User Management System

In system, the UMS provide feature management user and role. It holds information of all users, roles and relationship between user and role. But it does not hold the access control list (ACL): this data control by security component.

We use the feature Membership Provider of Devart component. The database of component is created as below. Note that, it only holds the basic information about user in system.



Figure 85: User Management System Schema

## Security Component

The security component is reputedly the extend of UMS, it control the right and permission of each role in system, so system can determine a user can or cannot navigate to a page as well as do a special task.



Figure 86: Security Schema

### Table SiteMap

This table stores the site map of system. Because we use the Site Map provider, so this table will be synchronize with the XML site map of website.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | PathId | interger | PK | Primary key |
| **2** | URL | varchar(4000) |  | The URL, map to XML file |

### Table Task

This table stores all the definition of task in VDMS system, for example, the Approve Order task.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | TaskId | long | PK | Primary key |
| **2** | PathId | long | FK | Refer to table aspnet\_Paths |
| **3** | TaskName | nvarchar(50) |  | The name of task |
| **4** | CommandName | varchar(30) |  | The Name of command in web |

### Table RolesInTasks

This table stores the relationship between table aspnet\_Roles and table app\_Tasks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | RoleTaskId | interger | PK | The primary key |
| **2** | TaskId | interger | FK | Refer to table app\_Tasks |
| **3** | RoleName | varchar(255) |  | The name of role. Refer to Membership and Role Component |
| **4** | ApplicationName | varchar(255) |  | The name of Application. Refer to Membership and Role Component |

### Table RolesInPaths

This table stores the relationship between table aspnet\_Roles and table app\_Paths.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | RolePathId | interger | PK | The primary key |
| **2** | RoleName | varchar(255) |  | The name of role. Refer to Membership and Role Component |
| **3** | ApplicationName | varchar(255) |  | The name of Application. Refer to Membership and Role Component |
| **4** | PathId | long | FK | Refer to table app\_SiteMap |

## Sub-store Management

This component controls the data of relationship between the dealers and their warehouse.



Figure 87: Sub-store Management Schema

|  |  |  |
| --- | --- | --- |
| **No** | **Table name** | **Description** |
| **1** | p\_Warehouse | Store the information about the warehouse of dealer |
| **2** | p\_Contact | Store the contact detail |
| **3** | p\_Dealer | Store all dealers in the system |

### Table Dealer

This table stores all dealers in system. Note that does not like VDMS-I, VDMS-II keep its own the information about dealer, so we can put the dealer to the multi-level tree. On other hand, it means that system will support the sub-dealer management.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | DealerCode | varchar(30) | PK | Primary key. Synchronize with Tip-Top |
| **2** | ParentCode | varchar(30) | FK | Foreign, self reference |
| **3** | ContactId | long | FK | Refer to contact detail |
| **4** | DealerName | nvarchar(256) |  | The name of dealer |
| **5** | Address | nvarchar(256) |  | The address of dealer |
| **6** | DatabaseCode | varchar(10) |  | Database of Tip-Top dealer used |
| **7** | AreaCode | varchar(25) |  | The business area code |
| **8** | ReceiveSpan | tinyint |  | The receive span of dealer |
| **9** | DefaultWarehouseId | long |  | The default warehouse for user in this dealer |
| **10** | OrderDateControl | Int |  | The value contains the list of order date control |

### Table Warehouse

This table stores the warehouse of dealer, in which dealer can indicate the shipping address and make the inventory for each individual stock.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | WarehouseId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | Code | varchar(30) |  | The code of warehouse |
| **4** | Address | nvarchar(255) |  | The address of warehouse |
| **5** | Source | char(1) |  | Reserve |

### Table Contact

This table stores the contact detail of each object (dealer, customer, provider…) in system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | ContactId | long | PK | Primary key |
| **2** | FullName | nvarchar(50) |  | The name of contact |
| **3** | Address | nvarchar(255) |  | The address |
| **4** | Phone | varchar(20) |  | The phone number |
| **5** | Email | varchar(255) |  | The email address |
| **6** | AdditionInfo | nvarchar(255) |  | The addition information |

## Message Store



Figure 88: Message Store

This component stores the message in system. The message can be the common message, private message (from dealer to VMEP and VMEP to dealer)

|  |  |  |
| --- | --- | --- |
| **No** | **Table name** | **Description** |
| **1** | data\_Message | The message body |
| **2** | data\_MessageBox | The relationship between user and message in system |
| **3** | data\_File | The customer of each dealer |

### Table Message

This table stores the message for communicate between dealer and sales. It also contains the message that automatic create by system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | MessageId | long | PK | Primary key |
| **2** | ParenId | long | FK | The parent message |
| **3** | Body | nvarchar(255) |  | The body of message |
| **4** | CreatedDate | datetime |  | Time created message |
| **5** | CreatedBy | nvarchar(256) |  | Personal who created message |
| **6** | Flag | char(1) |  | Which type of message: P private, S: system, C: common |

### Table MessageBox

This table stores the relationship between message and user, so system can determine which message is belongs to which inbox/outboxes of user.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | MessageBoxId | long | PK | Primary key |
| **2** | MessageId | long | FK | Refer to Message table |
| **3** | FromUser | nvarchar(255) |  | The user that create message (NULL if system) |
| **4** | ToUser | nvarchar(255) |  | The user that receive the message |
| **5** | Flag | char(1) |  | U if not answered, A if answered |
| **6** | Position | char(1) |  | I if inbox, O if outbox |

### Table File

This table stores the attachment of message.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | FileId | long | PK | Primary key |
| **2** | MessageId | long | FK | Refer to Message table |
| **3** | FileName | nvarchar(128) |  | The name of file |
| **4** | Body | image |  | The body of file |

## Basic Data Setting



Figure 89: Basic Data Setting Schema

This component stores the setting in system. This setting includes the message system and the setting of dealer about the part business.

|  |  |  |
| --- | --- | --- |
| **No** | **Table name** | **Description** |
| **1** | p\_Dealer | The information about dealer |
| **2** | p\_Customer | The customer of each dealer |
| **3** | p\_Vendor | The vendor of each dealer |
| **4** | p\_Favourite | The favorite setting |
| **5** | p\_Category | The category setting |
| **6** | p\_PartInfo | The information of part setting for each dealer |
| **7** | p\_Accessory | The accessory of each dealer |
| **8** | p\_AccessoryType | The type of accessory for all dealers in system |

### Table Customer

This table stores the customer of dealer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Field name | Type | Property | Description |
| 1 | CustomerId | long | PK | Primary key |
| 2 | DealerCode | varchar(30) | FK | Refer to table Dealer |
| 3 | ContactId | long | FK, NULL | Refer to table Contact |
| 4 | Code | varchar(30) |  | The code of customer. Defined by dealer. |
| 5 | Name | nvarchar(255) |  | The name of customer |
| 6 | VATCode | varchar(30) | NULL | The VAT code of customer |

### Table Vendor

This table stores the vendor who supplies part for dealer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | VendorId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | ContactId | long | FK, NULL | Refer to table Contact |
| **4** | Code | varchar(30) |  | The code of customer. Defined by dealer. |
| **5** | Name | nvarchar(255) |  | The name of customer |

### Table AccessoryType

This table stores the type of accessory (used for all dealers in system).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | AccessoryTypeCode | char(2) | PK | Primary key |
| **2** | AccessoryTypeName | nvarchar(50) | FK | The accessory type name |

### Table Accessory

This table stores the accessory used for each individual dealer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | AccessoryId | long | PK | Primary key |
| **2** | AccessoryCode | varchar(30) |  | The code of accessory |
| **3** | EnglishName | nvarchar(256) | NULL | Name of accessory, in English |
| **4** | VietnameName | nvarchar(256) | NULL | Name of accessory, in Vietname |
| **5** | DealerCode | varchar(30) |  | Refer to table Dealer |
| **6** | AccessoryTypeCode | char(2) |  | Refer to table AccessoryType |

### Table Category

This table stores the category of part that setting by dealer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | CategoryId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | ParentId | long | FK | Self refer to Category Id |
| **4** | Type | char(1) |  |  |

### Table PartInfo

This table stores the part setting for each dealer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | PartInfoId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | CategoryId | long | FK, NULL | Refer to table Category |
| **4** | PartCode | varchar(40) |  | The part code |
| **5** | AccessoryId | long | FK, NULL | Refer to table Accessory |
| **6** | PartType | char(1) |  | Type of Part: P: Part, A: Accessory |
| **7** | Price | long |  | The price of part, available only with accessory |
| **8** | Deleted | bit[[1]](#footnote-1) |  | This info is deleted or not |

### Table Favorite

This table stores the favorite part of dealer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | FavoriteId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | PartCode | varchar(40) | FK | The part code (in Tip-Top) |
| **4** | PartType | char(1) |  | The part type (P: Part, A: Accessory) |
| 5 | Rank | integer |  | Rank of favorite, from 1 to 10, default is 5 |
| 6 | Type | char(2) |  | The type of transaction |

## Order Part



Figure 90: Part Order Schema

|  |  |  |
| --- | --- | --- |
| **No** | **Table name** | **Description** |
| **1** | p\_Dealer | The dealer who order part |
| **2** | p\_OrderHeader | The order header |
| **3** | p\_OrderDetail | The order detail |
| **4** | p\_Warehouse | The warehouse where delivery order |
| **5** | p\_PartInfo | The information of part. Setting by dealer |
| **6** | p\_ReceiveHeader | The receive header |
| **7** | p\_ReceiveDetail | The receive detail |

### Table OrderHeader

This table stores the header of order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | OrderHeaderId | long | PK | Primary key |
| **2** | ReferenceId | long | FK, NULL | Self refer, used when system create sub-order |
| **3** | ReferenceRootId | long | FK,NULL | Reference the origin order (the root order) that created by dealer (the not root order always created by system) |
| 4 | CreatedDate | datetime |  | The date that order has been created |
| 5 | CreatedBy | nvarchar(256) |  | User has been created this order |
| 6 | Status | char(2) |  | The status of order[[2]](#footnote-2) |
| 7 | OrderType | char(1) |  | The type of order: Normal, Promotion[[3]](#footnote-3), Sub-Order |
| 8 | OrderSource | char(1) |  | The source of order[[4]](#footnote-4) |
| 9 | ToDealer | varchar(30) | FK | Shipping to dealer. In normal case, this field equal than DealerCode |
| 10 | DealerCode | varchar(3) | FK | Dealer the order part |
| **11** | ToLocation | long | FK | Refer to table Warehouse: where the order delivery to |
| **12** | OrderDate | datetime |  | The order date |
| **13** | ConfirmDate | datetime | NULL | The time that order has been confirm (when Tip-Top processed this order) |
| **14** | QuotationDate | datetime | NULL | The quotation date. NULL if not quotation |
| **15** | PaymentDate | datetime | NULL | The payment date. NULL if not payment |
| **16** | DelivaryDate | datetime | NULL | The delivery date. NULL if not delivery |
| **17** | ShippingDate | datetime | NULL | The shipping date. NULL if not shipping |
| **18** | AutoInStockDate | datetime | NULL | The date that order has been auto in-stock by system |
| **19** | AlreadyInStock | bit |  | This order has been in stock or not? |
| **20** | CanUndoAutoReceive | bit |  | Indicate auto receive can or cannot undo |
| **21** | ChangeMark | char(1) |  | Indicate the change order |
| **22** | TipTopNumber | varchar(10) |  | The number get from Tip-Top |
| **23** | TipTopProcessed | char(1) |  | Y/N value |
| **24** | Amount | long |  | The total amount of this order |
| **25** | SentWarningOverQuotation | bit |  | Indicate that order has been sent over quotation to user |
| **26** | SentWarningOverShipping | bit |  | Indicate that order has been sent over shipping to user |

### Table OrderDetail

This table stores the detail of order (order body).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | OrderDetailId | long | PK | Primary key |
| **2** | OrderHeaderId | long | FK | Refer to table OrderHeader |
| **3** | LineNumber | integer |  | The number of item in body. Compatible with Tip-Top |
| **4** | PartCode | varchar(40) | FK | Refer to table Part |
| **5** | OrderQuantity | integer |  | The order quantity |
| **6** | QuotationQuantity | integer |  | The quotation quantity |
| **7** | UnitePrice | long |  | The price of each unit |
| **9** | ModifyFlag | char(1) | NULL | Tracking the modify item |
| **10** | Note | varchar(250) | NULL | Reserve |
| **11** | PartCodeHistory | varchara(1000) |  | The changed code history. Update by trigger |

### Table ReceiveHeader

This table stores the header of order receiving.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | ReceiveHeaderId | long | PK | Primary key |
| **2** | OrderHeaderId | long | FK | Refer to table Order |
| **3** | DealerCode | varchar(40) | FK | Refer to table Dealer |
| **4** | WarehouseId | long | FK | Refer to Warehouse, the inventory is affected for this warehouse |
| **5** | IssueNumber | varchar(30) |  | The issue number, copy from Tip-Top |
| **6** | ReceiveDate | datetime |  | The date of receiving |
| **7** | IsLocked | bit |  | Allow or not allow user modify the body of receiving |
| **8** | IsAutomatic | bit |  | Indicate it is auto receiving by system or not |
| **9** | HasUndo | bit |  | Indicate it was undo auto receive by system or not |
| **10** | HasNGForm | bit |  | Indicate that dealer has created NG form or not |
| **11** | NotGoodNumber | varchar(30) | NULL | The Not Good Number, if any |

### Table ReceiveDetail

This table stores the body of order receiving.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | ReceiveDetailId | long | PK | Primary key |
| **2** | ReceiveHeaderId | long | FK | Refer to table ReceiveHeader |
| **3** | OrderHeaderId | long | FK | Refer to table OrderHeader |
| **4** | PartCode | varchar(40) | FK | Refer to table Part |
| **5** | Quotation | integer |  | Copy of quotation in order detail |
| **6** | GoodQuantity | integer |  | The number of good part |
| **7** | BrokenQuantity | integer |  | The number of broken part |
| **8** | WrongQuantity | integer |  | The number of wrong part |
| **9** | LackQuantity | integer |  | The number of lack part |
| **10** | Status | char(1) |  |  |
| **11** | DealerComment | nvarchar(250) | NULL | Dealer comment about this part |

## Inventory



Figure 91: Inventory Schema

|  |  |  |
| --- | --- | --- |
| **No** | **Table name** | **Description** |
| **1** | p\_PartInfo | Store the additional information of part (setting by dealer) |
| **2** | p\_Inventory | Store the inventory of all part (divide by warehouse) |
| **3** | p\_SalesHeader | The header of sales part |
| **4** | p\_SalesDetail | The detail of sales part |
| **5** | p\_TransferHeader | The header of transfer transaction |
| **6** | p\_TransferDetail | The detail of transfer transaction |
| **7** | p\_TransactionHistory | The other transaction in system |
| **8** | p\_Cycle\_Count\_Header | The header of cycle count transaction |
| **9** | p\_Cycle\_Count\_Detail | The detail of cycle count transaction |

### Table Inventory

This table stores the data of inventory of each part in a month, separate by warehouse.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | InventoryId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | WarehouseId | long | FK | Refer to table Warehouse |
| **4** | PartInfoId | long | FK | Refer to table PartInfo |
| **5** | Quantity | integer |  | The inventory quantity in a month |
| **6** | Month | integer |  | The month of inventory |
| **7** | Year | integer |  | The year of inventory |

### Table SalesHeader

This table stores the sales invoice of dealer (sell to end customer).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | SalesHeaderId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | CustomerId | long | FK, NULL | Refer to table Customer. NULL if retail customer |
| **4** | WarehouseId | long | FK | Refer to table Warehouse |
| **5** | OrderDate | datetime |  | The date that order has been created |
| **6** | SalesDate | datetime | NULL | The date that order has been soled |
| **7** | Status | char(1) |  | The status of header |
| **8** | SalesOrderNumber | varchar(50) |  | The bill number |
| **9** | ManualVoucherNumber | varchar(50) | NULL | The manual voucher, key in by dealer |
| **10** | CustomerName | nvarchar(50) | NULL | Customer name, NULL if not retail customer |
| **11** | SalesPerson | nvarchar(255) |  | User account who do sales |
| **12** | SubTotal | long |  | The total amount |
| **13** | TaxAmount | long |  | The tax amount |
| **14** | Discount | long |  | The discount amount (not include in total amount) |
| **15** | ModifiedDate | datetime |  | The modified date |
| **16** | WarehouseId | long |  | Refer to table Warehouse |

### Table SalesDetail

This table stores the body of sales invoice.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | SalesDatailId | long | PK | Primary key |
| **2** | SalesHeaderId | long | FK | Refer to table SalesHeader |
| **3** | PartInfoId | long | FK | Refer to table PartInfo |
| **4** | PartCode | varchar(40) | NULL | The Part Code, available only for sale part |
| **5** | PartName | nvarchar(256) |  | The name of part |
| **6** | PartType | char(1) |  | The type of part[[5]](#footnote-5) |
| **7** | OrderQuantity | intger |  | The order quantity of each item |
| **8** | UnitPrice | integer |  | The price of item |
| **9** | PercentDiscount | integer |  | The discount in percent |
| **10** | LineTotal | integer |  | UnitePrice \* OrderQuantity \* (100 – PercentDiscount) / 100 |
| **11** | ModifiedDate | datetime |  | The modified date |

### Table TransferHeader

This table stores the transfer transaction in the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | TransferHeaderId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | CreatedDate | datetime |  | The date that transaction created |
| **4** | CreatedBy | nvarchar(256) |  | The user that create transaction |
| **5** | TransferDate | datetime |  | The date that transaction is affected |
| **6** | FromWarehouseId | long |  | Transfer from source |
| **7** | ToWarehouseId | long |  | Transfer to destination |
| **8** | TransferComment | nvarchar(255) |  | The comment of this transaction |
| **9** | Status | char(2) |  | The status of transaction |
| **10** | VoucherNumber | varchar(30) | NULL | The voucher number key in by dealer |

### Table TransferDetail

This table stores the body of transfer transaction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | TransferDatailId | long | PK | Primary key |
| **2** | TransferHeaderId | long | FK | Refer to table TranferHeader |
| **3** | PartInfoId | long | FK | Refer to table PartInfo |
| **4** | PartCode | varchar(40) | NULL | The Part Code |
| **5** | Quantity | integer |  | The transfer quantity |
| **6** | PartComment | nvarchar(256) |  | Reserve |

### Table CycleCountHeader

This table stores the header of cycle count transaction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | CycleCountHeaderId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | WarehouseId | long | FK | Refer to table Warehouse |
| **4** | CreatedDate | datetime |  | The created date |
| **5** | CreatedBy | nvarchar(256) |  | The username who create transaction |
| **6** | LastEditedDate | datetime |  | The last edited time |
| **7** | CycleDate | datetime |  | The date do cycle count |
| **8** | Status | char(1) |  | The status of this transaction |
| **9** | TransactionComment | nvarchar(256) |  | The comment |

### Table CycleCountDetail

This table stores the body of cycle count transaction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | CycleCountDetailId | long | PK | Primary key |
| **2** | CycleCountHeaderId | long | FK | Refer to table TranferHeader |
| **3** | PartCode | varchar(40) |  | Refer to table PartInfo |
| **4** | PartType | char(1) |  | The Part Type |
| **5** | Quantity | integer |  | The transfer quantity |
| **6** | ItemComment | nvarchar(256) |  | Reserve |

### Table TransactionHistory

Every transaction in system, for example, receive, sales, special import and export, the adjust inventory… will affect to inventory and they will be catch and save to this table (quantity > 0 mean in-stock, quantity < 0 mean out-stock)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | TransactionHistoryId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | InvoiceNumber | varchar(50) | NULL | The number of invoice |
| **4** | TransactionDate | datetime |  | The transaction date |
| **5** | TransactionCode | char(1) | FK | Refer to table Transaction Type |
| **6** | VendorId | long | FK, NULL | Refer to table Vendor |
| **7** | Quantity | integer |  | The quantity of item in this transaction. Can be negative |
| **8** | ActualCost | long |  | The actual cost for this transaction |
| **9** | CreatedBy | nvarchar(256) |  | User account who created this data |
| **10** | CreatedDate | datetime |  | The date that transaction created |
| **11** | TransationComment | nvarchar(250) |  | Comment of dealer about this transaction |
| **12** | PartInfoId | long |  | Refer to Part Info table |
| **13** | WarehouseId | long |  | Refer to warehouse that raise transaction |
| **14** | SecondaaryWarehouseId | long | NULL | Used when transaction is from one warehouse to another warehouse |

## NG Form



Figure 92: NG Form schema

|  |  |  |
| --- | --- | --- |
| **No** | **Table name** | **Description** |
| **1** | v2\_app\_UserProfile | Store the profile of user in system |
| **2** | p\_ReceiveHeader | Store the header of receive transaction |
| **3** | p\_ReceiveDetail | The detail of receive transaction |
| 4 | p\_NGFormHeader | The header of NG form |
| 5 | p\_NGFormDetail | The detail of NG form |

### Table NGFormHeader

Store the header of Not Good Form.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | NGFormHeaderId | long | PK | Primary key |
| **2** | DealerCode | varchar(30) | FK | Refer to table Dealer |
| **3** | ReceiveHeaderId | integer | FK, NULL | Refer to table ReceiveHeader. NULL if create by manual (in case of abnormal return part) |
| **4** | NotGoodNumber | varchar(30) | NULL | The number of NG Form |
| **5** | RewardNumber | varchar(30) | NULL | The reward number |
| **6** | CreatedDate | datetime |  | The created date (default is current time) |
| **7** | ApproveDate | datetime |  | The approved date |
| **8** | ApproveLevel | integer |  | The approved level. There are three level: 1, 2, 3. The level refer to table UserProfile |
| **9** | Status | char(2) |  | The status of form: OP mean open (just created by dealer), SN mean sent to VMEP, CF mean Confirm by VMEP, RJ mean Reject by VMEP |
| **10** | NGType | char(1) |  | The type of NG Form: N mean Normal (created by receive data), S mean special (created by manual) |

### Table NGFormDetail

This table stores the body of Not Good form. Two cases: one is referring from receive, just copy the detail data; another is abnormal return part, dealer will create by manual.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Field name** | **Type** | **Property** | **Description** |
| **1** | NGFormDetailId | long | PK | Primary key |
| **2** | NGFormHeaderId | long | FK | Refer to table NGFormHeader |
| **3** | PartCode | varchar(40) | FK | Refer to table Dealer |
| **4** | PartStatus | char(1) |  | The status of part: Wrong Part, Broken Part, Lack Part |
| **5** | RequestQuantity | integer |  | The quantity of part |
| **6** | ApprovedQuantity | integer |  | The approved quantity of VMEP |
| **7** | ProblemAgainQuantity | integer |  | The quantity of part that is re-problem |
| **8** | Passed | bit |  | User check to ensure that NG case is passed |
| **9** | BrokenCode | varchar(30) |  | Refer to broken code in VDMS-I |
| **10** | DealerComment | nvarchar(250) |  | The comment of dealer for this item |
| **11** | L1Comment | nvarchar(250) |  | The comment of level 1 user |
| **12** | L2Comment | nvarchar(250) |  | The comment of level 2 user |
| **13** | L3Comment | nvarchar(250) |  | The comment of level 3 user |
| **14** | TransactionComment | nvarchar(256) |  | Comment about this transaction |

# Non-Functional Requirements

## Performance and Scalability

### Compress with IIS 6.0

To more efficiently use available bandwidth, we enable IIS HTTP compression. HTTP compression provides faster transmission time between compression-enabled browsers and IIS, regardless of whether our content is served from local storage or a UNC resource. In VDMS, we compress both static files and application response files. Compressing application response files is usually called *dynamic compression*.

After enable compression, the homepage size reduces from 45K to 9.6K, save 80%.

### Cache web result

For improving performance and scalability of system, we use these technique describe in [Turning .NET Application Performance](http://msdn2.microsoft.com/en-us/library/ms998583.aspx) article.

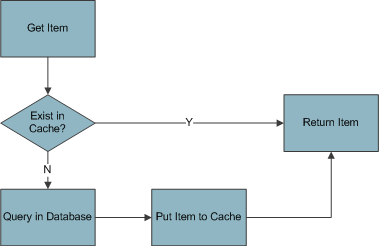


Figure 93: Retrieve Item Process

The cache in system will be managed by using the Cache component from [Microsoft’s Enterprise Library](http://msdn2.microsoft.com/en-us/practices/bb190347.aspx).

## Security

The user in VDMS system will be managed by Membership Provider of .NET 2.0. Because of this provider is only provide the way to manage user, role, and mapping user to role, so we expand to manage the mapping between role and path (is real, path mean URL), role and task (when user navigate to a path, the system display a window, and there are many task in a window).

The expanded area contains three classes: Task, RoleInPath and RoleInTask.

All user must be login to system before continue. It can be done by check the authentication setting in web’s configuration. After login to system, each user has some role. For each role, we determine the authorization to view page, execute some task or redirect him to deny page.

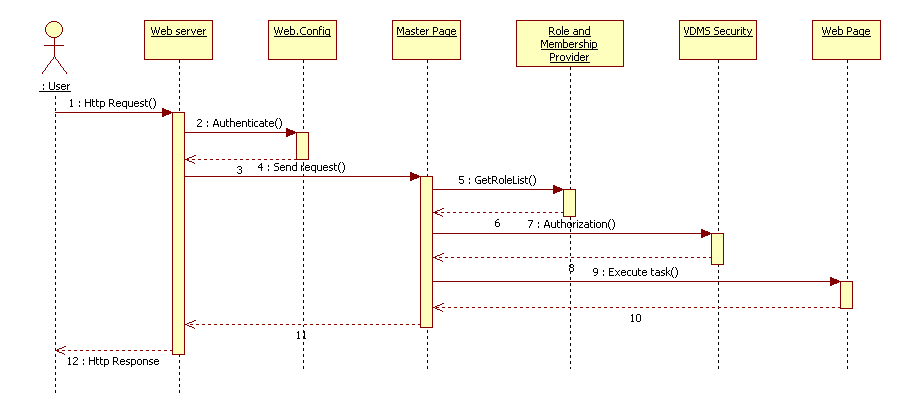


Figure 94: Security Checking

Flow explanations:

1. User sends an Http Request to web server.
2. Web server first check the authentication in web’s configuration.
3. The checking result return and web server redirect user to deny access page if user has not authentication.
4. The web server send request of user to the master page.
5. The master page gets the list of role of user from membership provider.
6. The list of role return from Role and Membership provider.
7. The master page checks the authorization of user.
8. The checking result return from VDMS security component.
9. The master page loads the web page and execute task.
10. The execute result return to master page.
11. The mater page returns the http stream to web server.
12. The Http response sends to user.

## Language

Be using [ASP.NET Globalization and Localization](http://msdn2.microsoft.com/en-us/library/c6zyy3s9.aspx), it very easy to support multi-language in VDMS system. For adding one new language, it simple translates only the XML resource, without rebuilt solution.

The VDMS has two languages built-in: they are English and Vietnamese.

## Ajax

[Ajax](http://www.adaptivepath.com/ideas/essays/archives/000385.php) is the technique used for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is intended to increase the web page's interactivity, speed, functionality, and usability.

In VDMS system, we use the [Microsoft ASP.NET AJAX](http://ajax.asp.net/), [Control Toolkit](http://www.asp.net/ajax/ajaxcontroltoolkit/), and the [jQuery](http://jQuery.com).

### Ajax Rules

For the maximum effective, VDMS-II following these rules below when applies ajax:

* Using UpdatePanel to exchange data and html code for the small piece in the screen;
* Using Control Toolkit to change behavior of web control for the common control (text input, datetime input…)
* Using jQuery with Popup screen to separate the input process of user, for example separate the header and body input

### Ajax Result

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Screen** | **Function** | **Technique** |
| **1** | Homepage | Inbox | Update Panel |
|  |  | Outbox | Update Panel |
|  |  | Compose new message | jQuery |
| **2** | Order List | Load Warehouse from Dealer | Update Panel |
| **3** | Create Order | Load Warehouse from Dealer | Update Panel |
|  |  | Input Order Date | Control Toolkit |
|  |  | Load Favorite, Safety Stock, Search part with condition | jQuery |
|  |  | Excel mode | Update Panel |
|  |  | Mark part as favorite | Call Web Service |
|  |  | Get Model from Part Code, Part Name, Engine Number | Update Panel |
| **4** | Order Receive | Input Order Date | Control Toolkit |
| **5** | NG Form List | Input Order Date | Control Toolkit |
| **6** | NG Form Edit | Input Order Date | Control Toolkit |
|  |  | Load Search part with condition | jQuery |
|  |  | Excel mode | Update Panel |
| **7** | In Short Query | Input Order Date | Control Toolkit |
| **8** | Load saved sales form | Input Order Date | Control Toolkit |
| **8** | Sales Part | Load Customer Information | Update Panel |
|  |  | Input Order Date | Control Toolkit |
|  |  | Load Favorite, Search part with condition | jQuery |
|  |  | Excel mode | Update Panel |
|  |  | Mark part as favorite | Call Web Service |
|  |  | Get Model from Part Code, Part Name, Engine Number | Update Panel |
| **8** | Special Import/Export | Load Search part with condition | jQuery |
|  |  | Excel mode | Update Panel |
| **9** | Stock Transfer | Load Search part with condition | jQuery |
|  |  | Excel mode | Update Panel |
| **10** | Bin Card query | Load Warehouse from Dealer | Update Panel |
|  |  | Input Date | Control Toolkit |

# VDMS-I

## Bonus

### Query Bonus

#### Function Screen

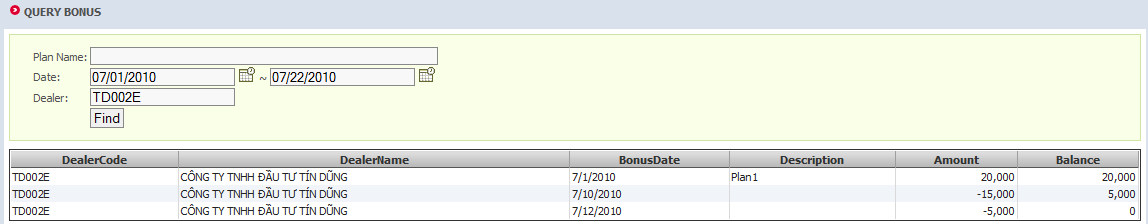


Figure 95: Query bonus screen

#### Function Description

The screen that shows the plans of each dealer, its confirmed bonus money and the balance of all plans.

#### Instruction of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer code** | The code of dealer |
| **Dealer name** | Dealer’s name |
| **Bonus date** | The date of each bonus amount |
| **Description** | Detailed desctiption of a bonus |
| **Amount** | The amount of each transaction |
| **Balance** | Remaining money |

### Bonus confirm

#### Function Screen

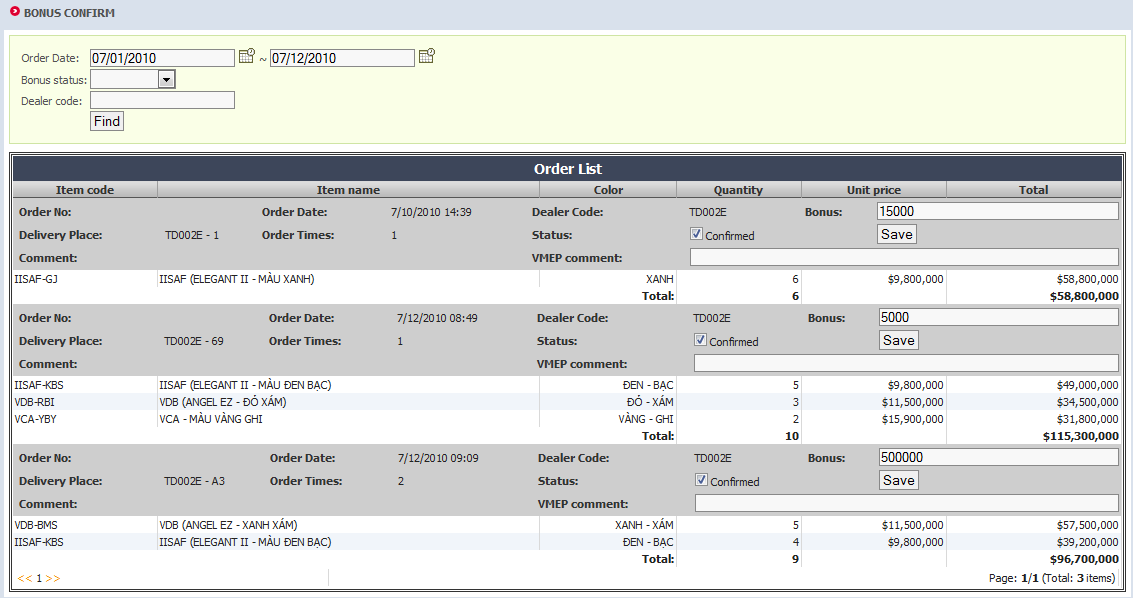


Figure 96: Bonus confirm screen

#### Function Description

Get Order data and confirm for bonus status only by sale man.

#### Data Flow

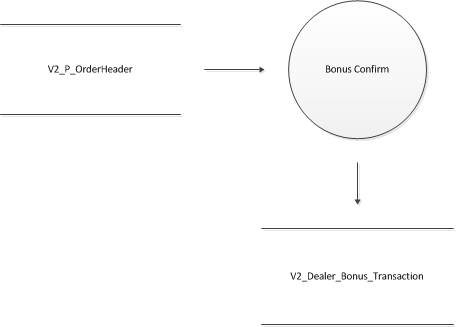


Figure 97: Bonus confirm data flow

#### Instruction of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer code** | The code of dealer |
| **Bonus** | Amount of money in the bonus |
| **Status** | Flag that the bonus is confirmed or not |
| **VMEP Comment** | Comment from VMEP |

### Bonus Plan

#### Function screen

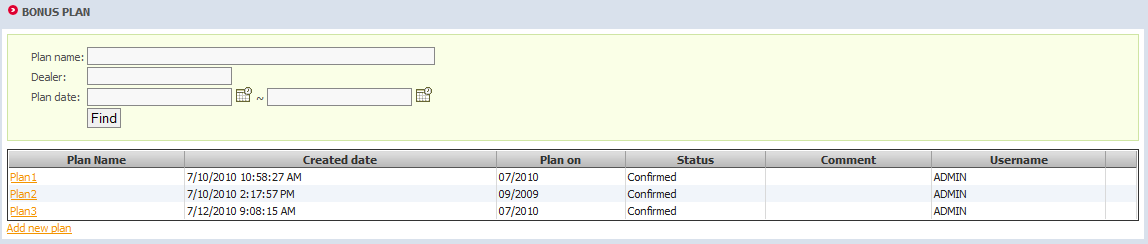


Figure 98: Query bonus plan

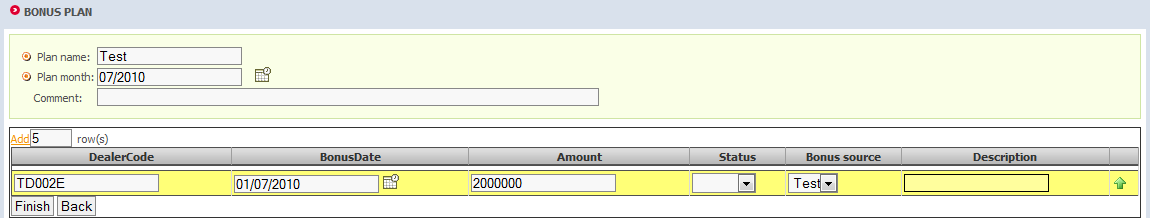


Figure 99: Add new plan

#### Function description

Sale ADM define the bonus plan for each dealer in each month (Insert, Update, Delete, Report balance)

#### Data flow

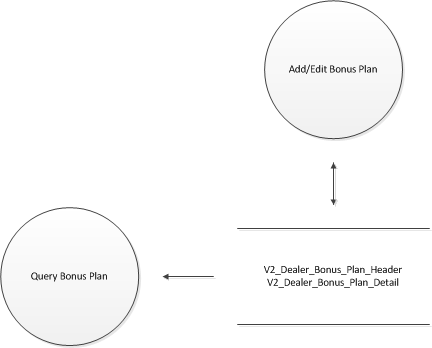


Figure 100: Bonus Plan data flow

#### Instruction of fields

Query bonus plan

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Plan name** | Name of a plan |
| **Created date** | Created date |
| **Plan on** | Which month the plan affects |
| **Status** | Indicates that the plan is comfirmed or not |
| **Comment** | Comment |
| **Username** | Created user |

Add bonus plan

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer code** | The code of dealer |
| **Bonus date** | Bonus date |
| **Amount** | Total amount of money for the plan |
| **Status** | Locked or not |
| **Bonus source** | Can be accessed in Basic Data Setting > VMEP Setting > Bonus Source |
| **Description** | Description |

### Monthly closing

#### Function screen

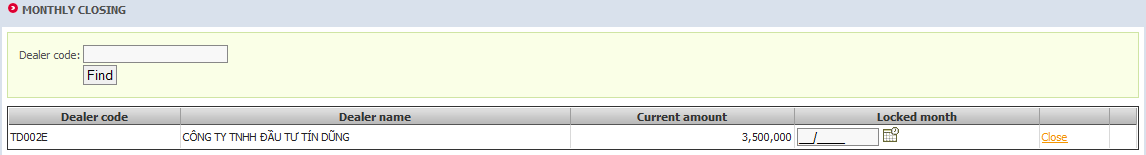


Figure 101: Monthly closing screen

#### Function description

Sale ADM close bonus in each month to calculate bonus balance.

#### Data flow

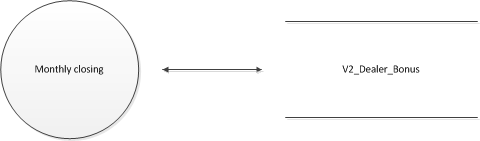


Figure 102: Monthly closing data flow

#### Instruction of fields

|  |  |
| --- | --- |
| **Field name** | **Field description** |
| **Dealer code** | The code of dealer |
| **Dealer name** | Dealer name |
| **Current amount** | Current amount of all plans of a dealer |
| **Locked month** | The month currently locked |

## Payment

### Import Payment Data

#### Function screen

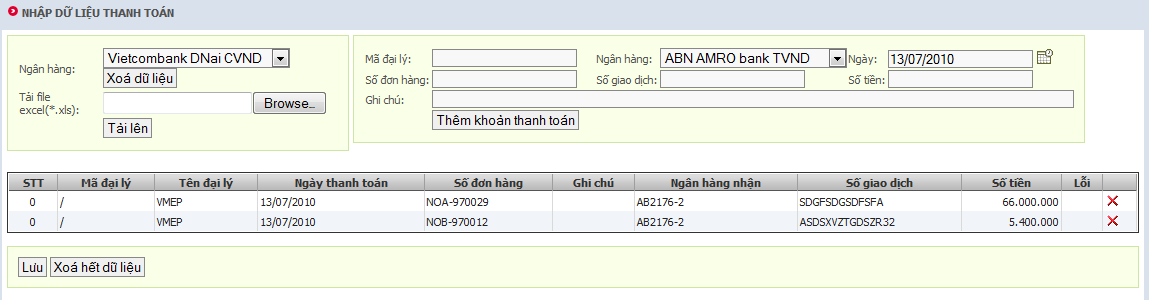


Figure 103: Import payment data screen

#### Function description

FIN can import payment data from each bank, which has different excel format. Those formats are all supported by the system, or can be modified via Basic Data Setting > VMEP Setting > System parameter > Payment excel file. Otherwise, FIN has an ability of manually input payment data one by one.

#### Data flow

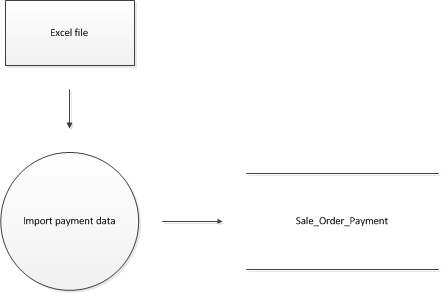


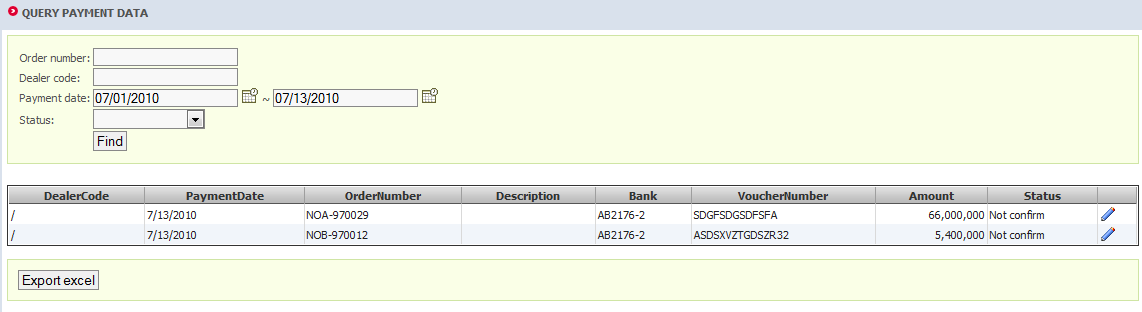
Figure 104: Import payment data – data flow

#### Instruction of fields

|  |  |
| --- | --- |
| Field name | Field description |
| Dealer code | The code of dealer |
| Dealer name | Dealer name |
| Payment date | The date of payment |
| Order number | Order number (from TIPTOP) |
| Description | Description |
| Bank code | Received bank |
| Transaction | Number of transaction |
| Amount | Money |

### Query Payment Data

#### Function screen



#### Function description

The previewing screen before process the Re-consign. This screen summary the payment from Dealers.

#### Instruction of fields

See Import Payment Data.

### Do Re-consign

#### Function screen

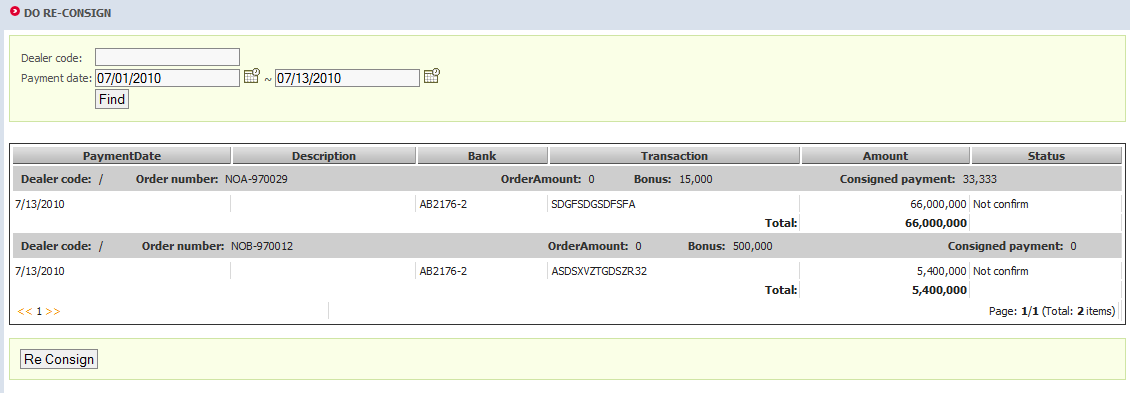


Figure 105: Do Re-consign screen

#### Function description

Combine the Payment amount (From Banks) and Bonus Amount (from Bonus Management).

* If payment amount greater than Order Amount: Separate the payment amount to 2 parts:
  + Equal to Order payment amount
  + Remaining amount (may be used for next orders, has remark)

The tiptop should used this data for module ANM. Allow tiptop get data has been confirm in VDMS side.

* In case payment amount less than Order Amount: high-light the Orders and the allow VMEP’s FIN change the Payment data with these fields: Description and the TIPTOP voucher (more details in FIN confirm payment)

#### Data flow

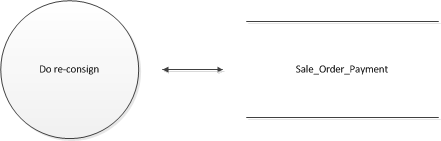


Figure 106: Do re-cosign data flow

#### Instruction of fields

See Import Payment Data.

### Confirm Payment

#### Function screen

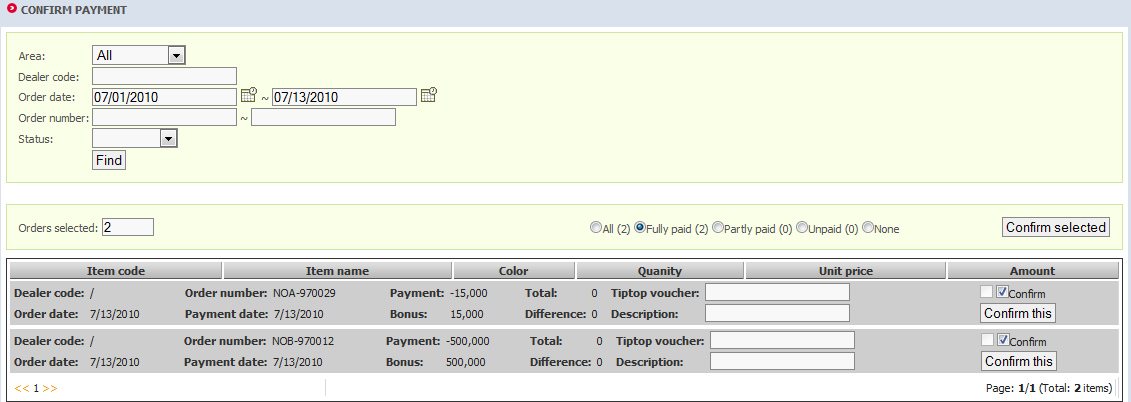


Figure 107: Confirm payment screen

#### Function description

FIN makes the payment(s) to be confirmed. Payments can be verified one by one or in group of selected rows.

#### Data flow

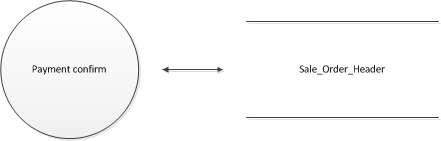


Figure 108: Confirm payment data flow

#### Instruction of fields

|  |  |
| --- | --- |
| Field name | Field description |
| Item code | Vehicle information |
| Item name |
| Color |
| Quantity |
| Unit price |
| Amount |
| TipTop voucher | Voucher number from Tiptop |
| Confirm | Mark it in order to confirm multiple orders |

## Entire Vehicles (update)

### VMEP Sales Confirm Order

#### Function screen

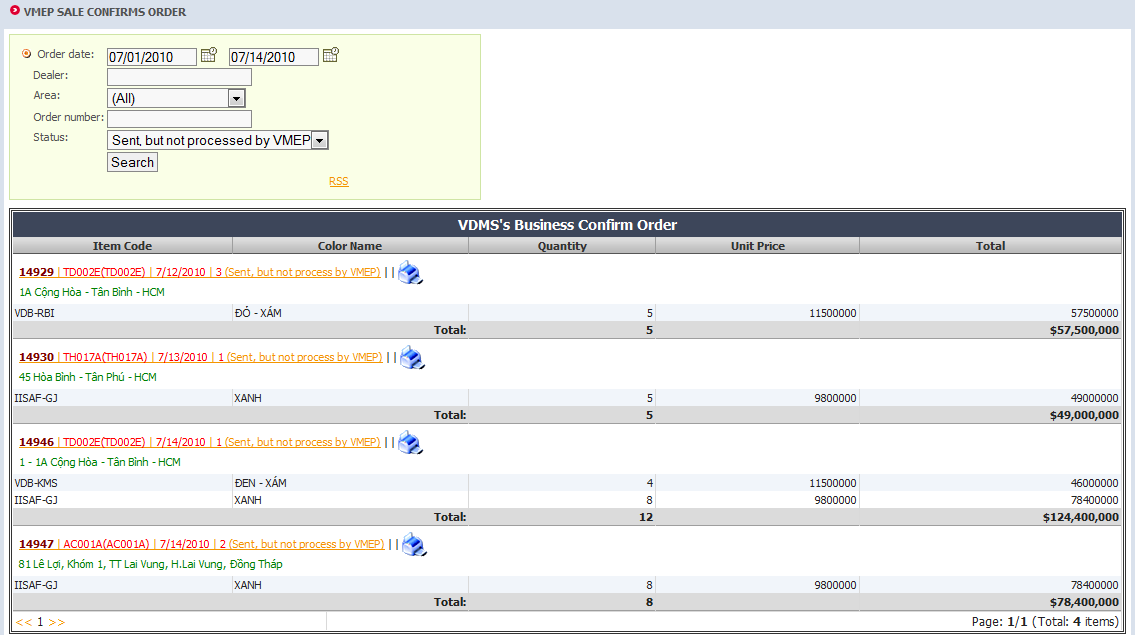


Figure 109: List Entire Order

#### Function description

This screen sales can search and display list of order, depend on the status. For each order, there is a hyperlink that navigates to process order screen.

#### Instructions of fields

Please see 2.2.6 for more detail.

|  |  |
| --- | --- |
| Field name | Field description |
| Status | When dealer create new order, the status is 0 if the order is draft, 1 if order is sent. When order is sent, it visible to sales, and he can confirm or lock the order. If order is confirmed, the status is 2, and if order is lock, the status is 4. |

### Process Order

#### Function screen

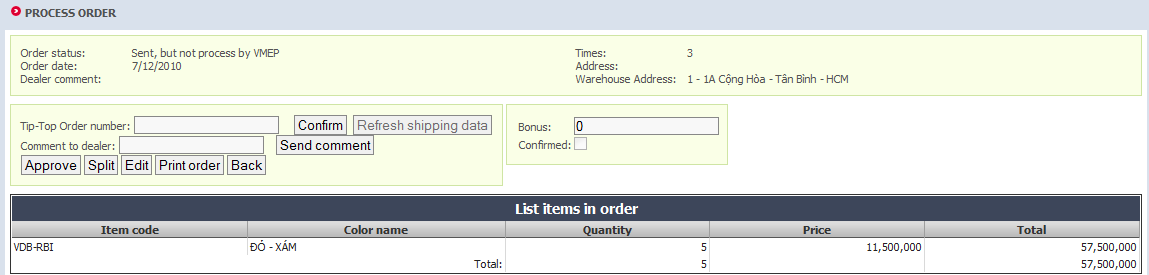


Figure 110: List Entire Order

#### Function description

After search order depends on query condition, sales can click to order to process it. The function list is:

1. Confirm: this function is confirm order, the status of order switch to confirm (4), and VMEP may export item to dealer. Note that this order must confirm in Tip-Top before confirm in VDMS.
2. Send comment: sales can send a comment about this order to dealer. The status of order does not change.
3. Approve: sales approve order, it mean that order is processing by sales and dealer cannot modify it. The order status is change to 4
4. Split: the order will be split into 2 new orders. There are two sub cases: split when order has not been confirmed, and split when order confirmed.
5. Edit: the order can be edited if it not confirmed. In this case, we use edit an entire order screen to do that

#### Instructions of fields

|  |  |
| --- | --- |
| Field name | Field description |
| Tip-top order number | The code of dealer |
| Comment to dealer | Dealer name |
| Bonus | The date of payment |
| Confirmed | Order number (from TIPTOP) |

### Edit an Entire Order

#### Function screen

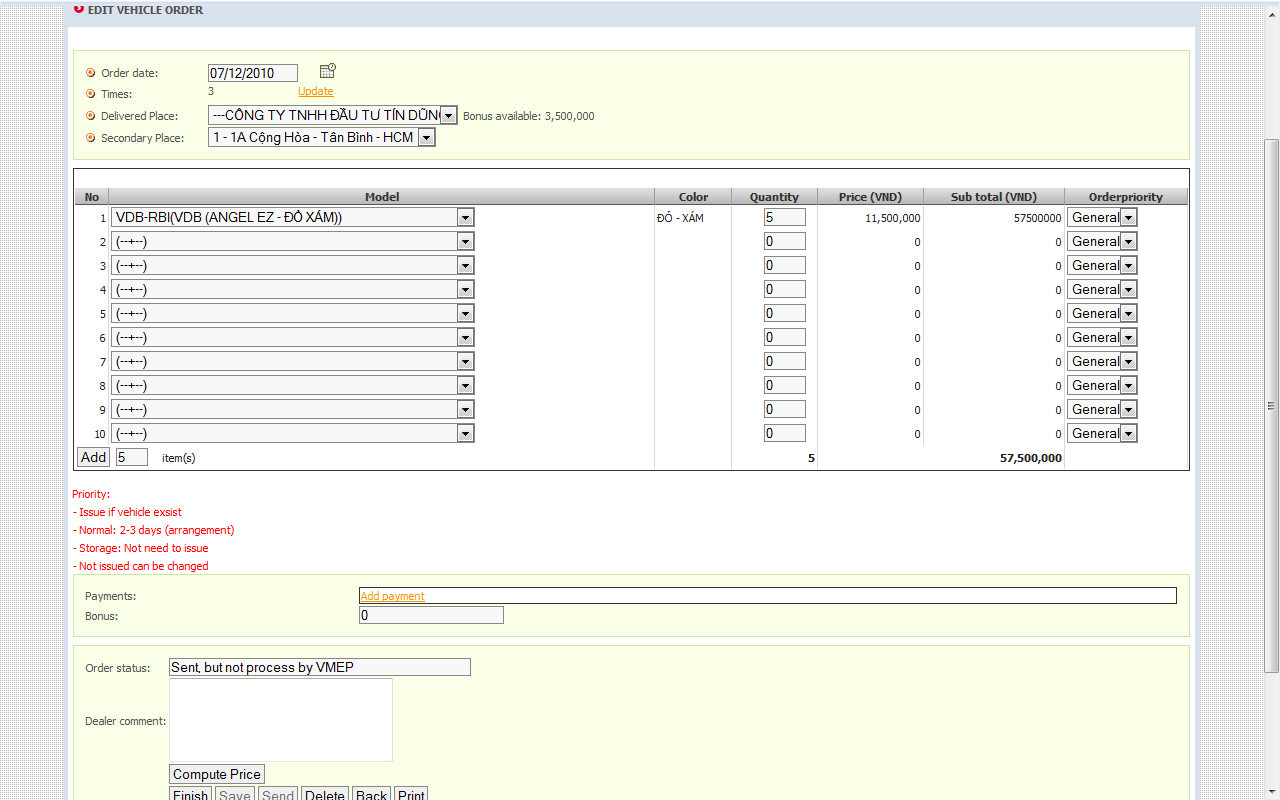


Figure 111: Edit an Entire Order

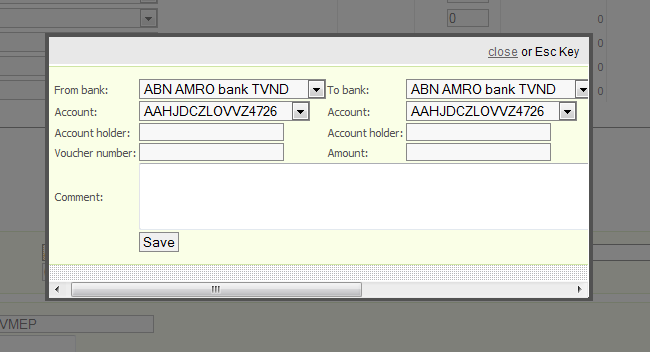


Figure 112: Add payment screen

#### Function description

This screen allow create a new entire order or modify an exist order. After edit, this can be saved to dealer’s data or send to sales. The detail includes:

* 1. The order header (order date, order time, address)
  2. List of item can be add/modify/delete from the grid
  3. Total quantity and total amount can be calculate automatically
  4. Payment data and Bonus amount which are optional
  5. Dealer can be sent a comment to sales.

#### Instructions of fields

The order header will be saved to table sale\_OrderHeader.

|  |  |
| --- | --- |
| Field name | Field description |
| Order date | The date of order, always equal today if the order is create new. In case of edit old order, it’s the old value (in previous saving). Refer to field OrderDate. |
| Times | The times that dealer create new order in day. Automatically increase by database, and reset to 1 in the beginning of day. Refer to field OrderTimes. |
| Delivered place | The branch of dealer, get from Tip-Top and save to ShippingTo field. |
| Dealer comment | The comment from dealer, save to DealerComment field. |
| Status | When order saved to database, if it save only for dealer (user click to Save button) the Status is 0, if it send to sales (user click to Send button) the Status is 1. Other case is if order is deleted, it is not removed from database, but it is set the status to 3. |
| Bonus | Amount of bonus money for an order |

The order detail will be saved to table sale\_OrderDetail.

|  |  |
| --- | --- |
| Field name | Field description |
| Item code | The code of motorbike, get from list of selling motor (table data\_Item). User can select only items that available (indicate by sales). It saved to ItemCode. |
| Color | The color of motor. Because it related to ItemCode, so it does not saved to database. |
| Motorbike | Like color, it only shows in screen and does not save to database. |
| Quantity | The quantity the dealer order for each item. Refer to OrderQty filed. |
| Price | The item price, get from Tip-Top (table data\_Item). Note that when sales confirm order, this price will be synchronizing with Tip-Top. Refer to UnitPrice field. |
| SubTotal | The sub total, equal to Quantity \* Price. Not saved to database |
| Priority | The priority of item. Saved to OrderPriority field |

The payment data will be saved to sale\_Payment.

1. For every bit field, 0 means false, 1 means true [↑](#footnote-ref-1)
2. OP: Order Open, SN: Order Sent, CF: Order Confirmed, NC: Order Closed Normal, RO: Order Re-Open, AC: Order Closed Abnormal, VD: Order Void [↑](#footnote-ref-2)
3. Promotion Order create by Tip-Top [↑](#footnote-ref-3)
4. V: Order created by VDMS, T: Order created by Tip-Top [↑](#footnote-ref-4)
5. P: Part, A: Accessory [↑](#footnote-ref-5)