National Housing and Construction Company (NHCC)

National Housing & Construction Company Ltd
Titles Management System (TMS)

Tms

Title Management System

Technical Report

Version No	1.0
Date	March 2020
Author	InfoTronics Business Systems

InfoTronics Business Systems

25 North-South Close

P.O. BOX 16190

Kampala Uganda

Table of Contents

Table of Figures

Acronyms and Abbreviations

BI	Business Intelligence
HTML	Hypertext Markup Language
NHCC	National Housing and Construction Company Limited
ICT	Information and Communication Technology
IIS	Internet Information Services
IT	Information Technology
MB	Mega Bytes
TMS	Title Management System
URL	Uniform Resource Locator
SQL	Structured-Query-Language
UNC	Universal Naming Convention

Introduction

1 Purpose

The hand over technical report is meant to inform the NHCC staff on the basic management, use and maintainance of the TMS system; that is intended to be useful in the day-to-day operations in the organisation. It also forms the basis of setting up the system, should it be required by the systems administrator.

2 System Overview and Configuration

2.1 System Requirements

2.1.1 Overall System Requirements

The minimum server requirements for optimal operation of TMS are:

A Server with at least 2 x Intel® Xeon® E5-2620(6 Core/2.0GHz/15MB/95W) processor; 8-32GB RAM; P410i/512MB FBWC cache, 6x300GB SAS HDD; 2x750W Power supplies; Four port 1 GB NIC; Internal RAID Controller with interface cables; Operating System: Licensed Windows Server Standard 2012; the .NET Framework and Internet Information Services (IIS); Microsoft SQL Server 201&/R2 or later versions.
Some of the features such as 6x300GB SAS HDD; 2x750W Power supplies; Four Port 1 GB NIC; Internal RAID Controller with interface cables are for ensuring redundancy.

2.1.2 The .NET Framework

The .NET Framework (pronounced dot net) is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library known as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (as contrasted to the hardware environment), known as Common Language Runtime (CLR), and an application virtual machine that provides services such as security, memory management, and exception handling. FCL and CLR together constitute .NET Framework.
FCL provides user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications.

Programmers produce software by combining their own source code with .NET Framework and other libraries. .NET Framework is intended to be used by most new applications created for Windows, web and mobile platforms. Microsoft also produces an integrated development environment largely for .NET software called Visual Studio.

The .NET Framework provides the following:

- A consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- A code-execution environment that minimizes software deployment and versioning conflicts.
- A code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- A code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- Make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- Build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

2.1.3 Microsoft SQL Server

Microsoft SQL Server is the requisite database environment used by TMS. SQL Server makes it easier and more cost effective to build high-performance, mission critical applications, enterprise ready Big Data assets, and Business Intelligence (BI) solutions that help employees make better decisions faster. These solutions have the flexibility of being deployed on premises, in the cloud or in a hybrid environment, and can be managed through a common and familiar tool set.

2.1.4 C# Programming Language

TMS was developed primarily using the C# (C Sharp) programming language. C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

For one to make changes to the TMS source code, he/she should be having sound hands-on experience in writing C# code.

2.2 Registering the Web Application

For TMS to be visible as an application, it has to be registered on the Internet Information Services (IIS) to run. To register TMS with Internet Information Services (IIS), proceed as follows:

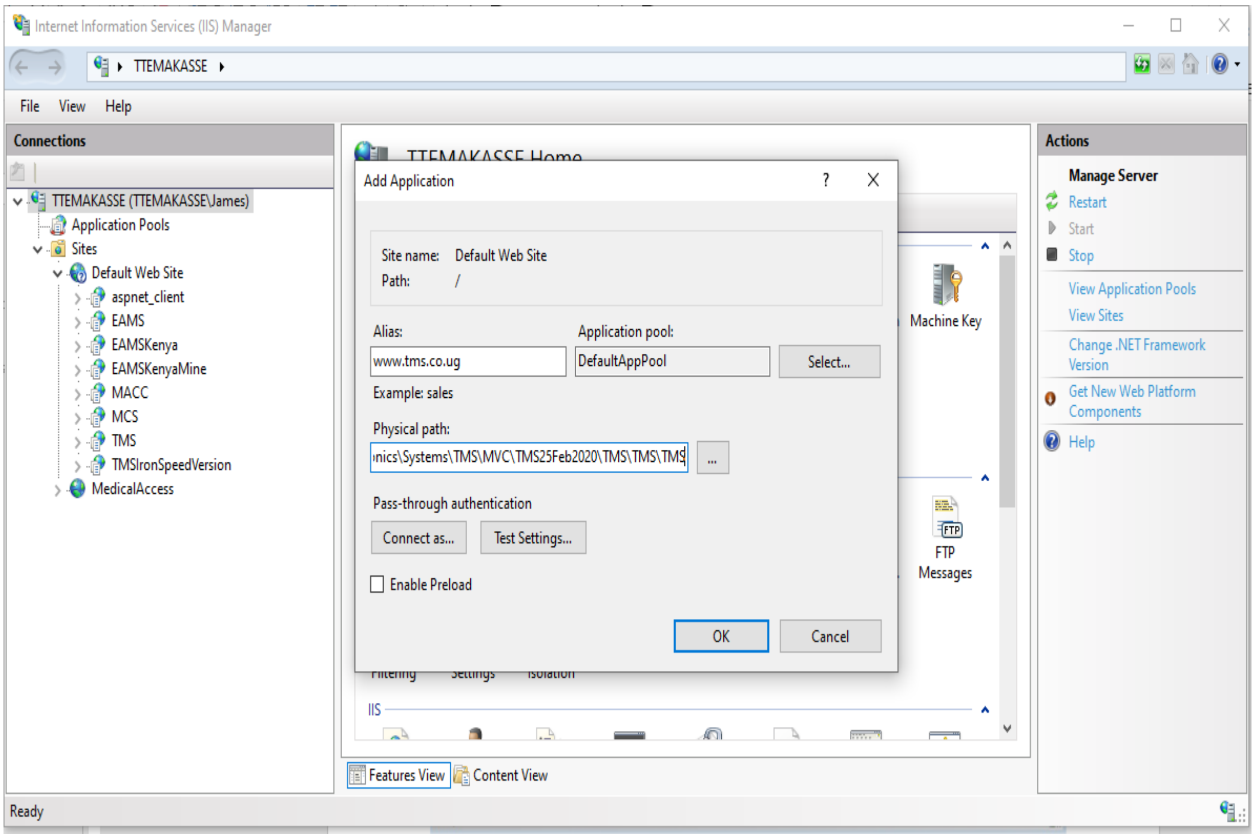
i. Copy the source code files to a folder on the server machine (or on the network).

ii. **Open IIS on the server:**

- Select Windows Start - Control Panel
- Select System and Security - Administrative Tools
- Select Internet Information Services (IIS) Manager
- Navigate the tree view to Server Name - Web Sites - Default Web Site

iii. **Create a Virtual Site:**

- Right-click default web site and select - New Application. The application creation wizard window will open (***Figure 1***).
- Enter the Alias – for example www.tms.co.ug and click next to continue.



Add Application

Site name: Default Web Site
Path: /

Alias: www.tms.co.ug
Application pool: DefaultAppPool
Example: sales

Physical path: nics\Systems\TMS\MVC\TMS25Feb2020\TMS\TMS\TMS

Figure 1:Application creation wizard window

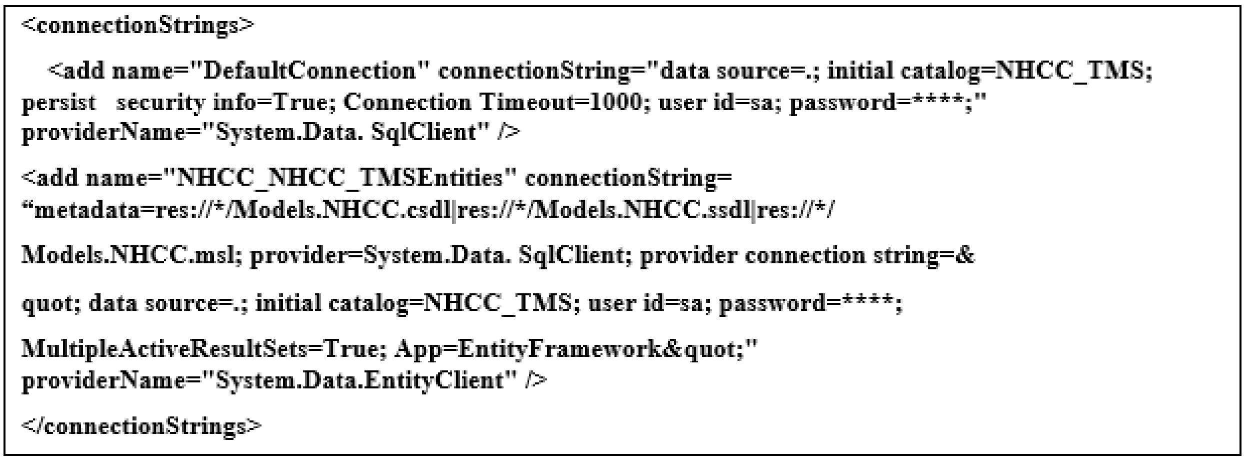
- Under the Physical path, enter the file(s) location for example E:\InfoTronics\Systems\TMS25Feb2020\TMS.
- Leave the default security permissions (i.e. Read on and Run Scripts on) and click next to continue.
- Click (***OK***) to complete the registration of the web site.

2.3 System Access and Database Security

By default, when a user accesses the web site, they log onto your server using the default anonymous user account. This special user account is automatically created when IIS is installed. By default, this user will **NOT** have any access to data residing in SQL-Server. You must ensure that the anonymous user is setup on SQL-Server as an authorized user.

Alternatively, you can use a named user and password to access the database through setting up the username and password to the database in the connection string as shown below:

you shall need to change the database connection string as follows (***Figure 2***):



```
<connectionStrings>
   <add name="DefaultConnection" connectionString="data source=.; initial catalog=NHCC_TMS;
persist   security info=True; Connection Timeout=1000; user id=sa; password=****;"
providerName="System.Data. SqlClient" />
<add name="NHCC_NHCC_TMSEntities" connectionString=
“metadata=res://*/Models.NHCC.csdl|res://*/Models.NHCC.ssdl|res://*/
Models.NHCC.msl; provider=System.Data. SqlClient; provider connection string=&
quot; data source=.; initial catalog=NHCC_TMS; user id=sa; password=****;
MultipleActiveResultSets=True; App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
</connectionStrings>
```

Figure 2:Connection String

2.4 Performance

Please note that the first time the system is accessed it will take longer time loading than normal. This is because ASP.NET has to compile all the user controls. Once this is done, the application will perform considerably faster on the subsequent access.

3 System Architecture

3.1 High Level Architecture

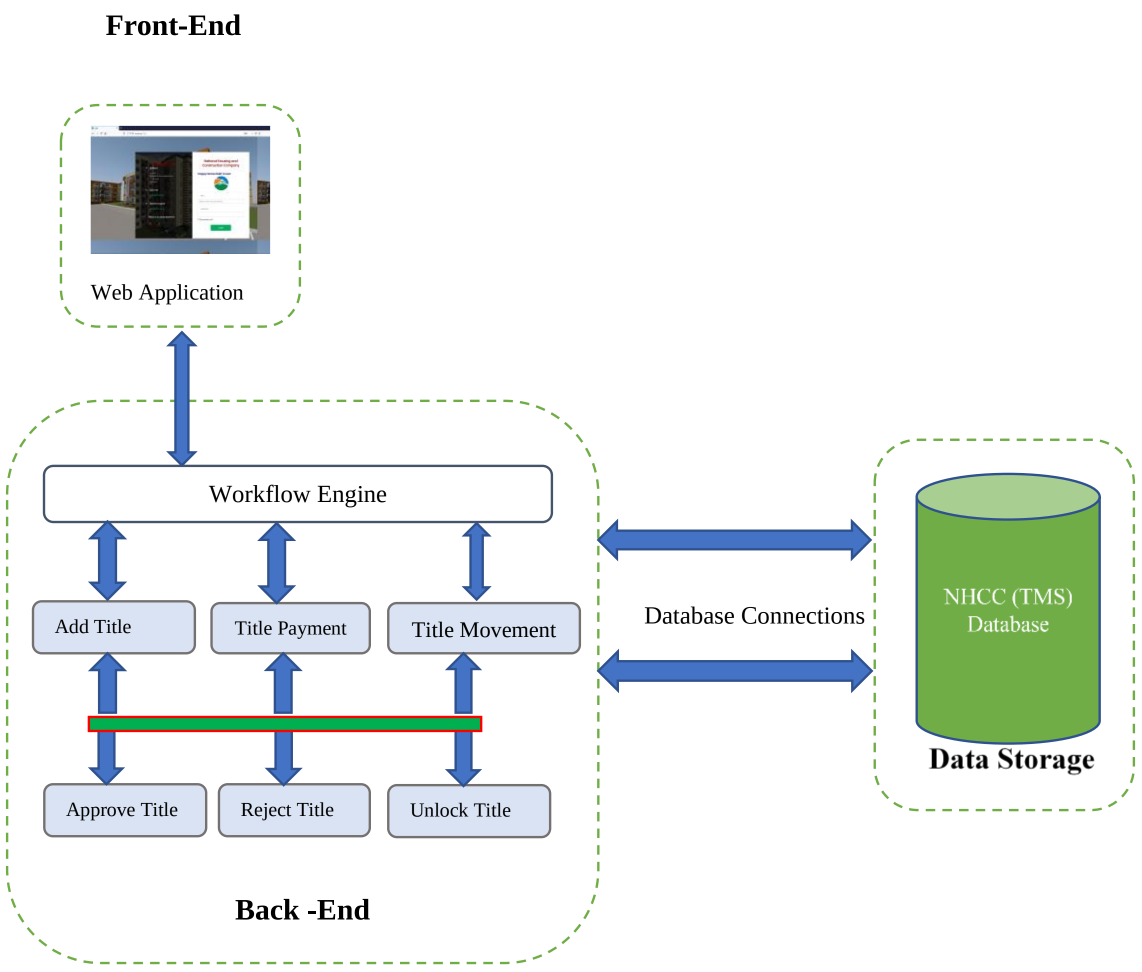


Figure 3:System High Level Architecture

The system high level architecture above represents how the different components in the TMS application coordinate and interact with each other.

3.1.1 Database Connection

The TMS web application connects to the TMS database through the ADO.NET Entity framework. The database connection was developed on the database - up approach while using Entity framework in ASP.NET. With the database - up approach, the database tables were created first together with their relationships and associations before coming up with the application controllers and views.

The ADO.NETframework provides consistent access to data sources like SQL server that the applications can use to retrieve, handle, and update data in the TMS database.

3.1.2 Workflow Engine

The TMS web application consists of three main modules namely: **Adding titles**, **Moving titles** and **Recording title payments**.

For each of the titles that have been added in the system, the following actions may be performed on them accordingly:

- Approving Titles
- Unlocking Approved Titles
- Rejecting Titles

The Title Management System is customly built to perform all the relevant actions mentioned above on the data (titles) recorded in the system.

However, for these actions to be carried out successfully, the Workflow Engine bus terminal acts as a bridge between the modules and the TMS web application as shown in ***Figure 3***.

3.2 System Process Architecture

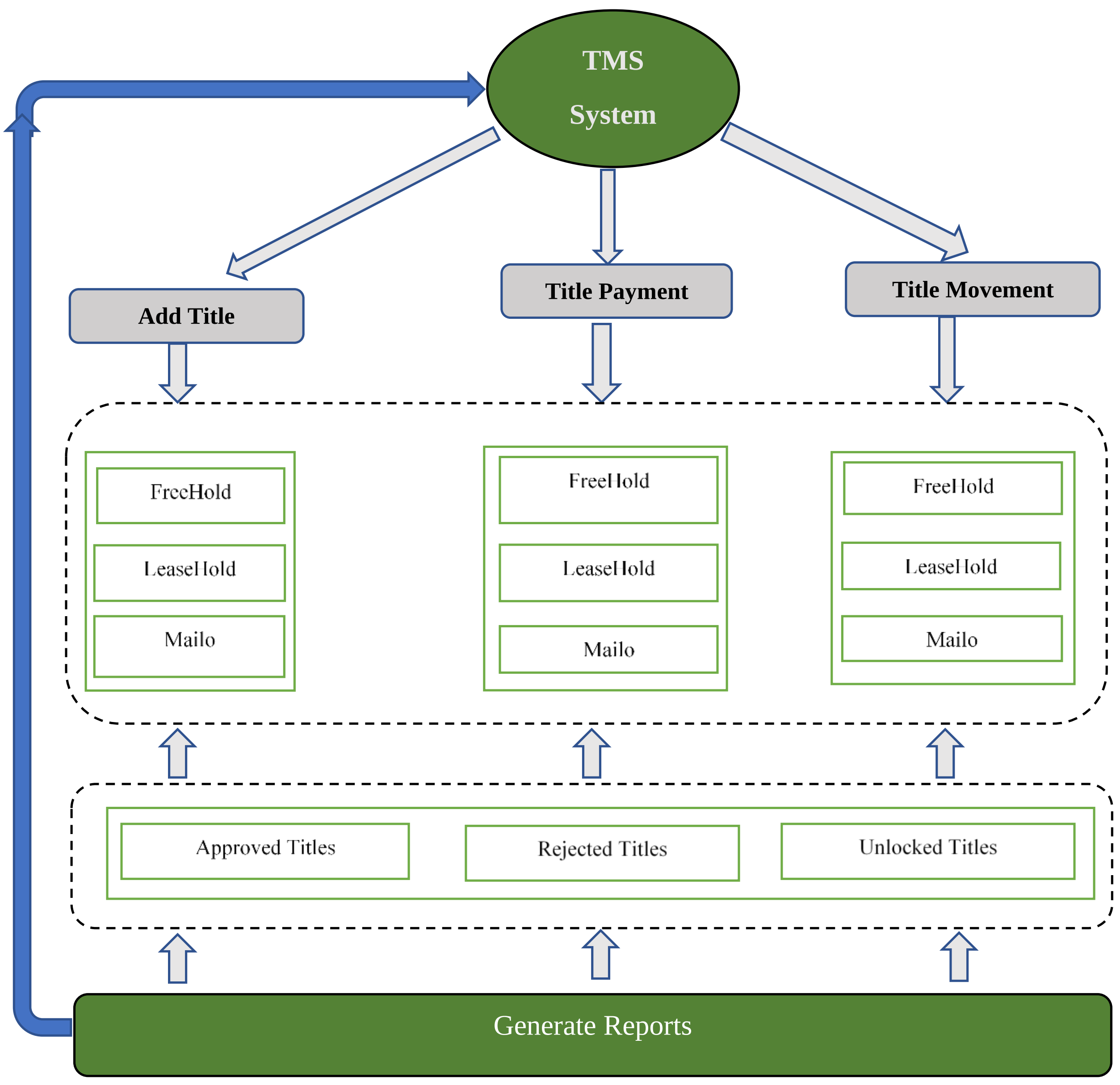


Figure 4:System Process Architecture

Explanation of the modules in the System Process Architecture

- **Add Title:** New titles are entered through this module. These titles may include: Freehold, Lease Hold and Mailo Land Titles. The system provides separate forms for the different land types. Depending on which land type the title is, the user selects the corresponding form to fill in the title data that is stored in the database.
- **Title Payment:** This module is used to record details of payments made on the different titles. It provides the user with a form to record the payment details of the title.

- **Title Movement:** Records details of titles that have been moved. It provides a form with fields that are filled in regarding details of the moved titles.Therefore , all titles that have been moved are recorded through this title movement module.

3.3 Application Technologies Architecture

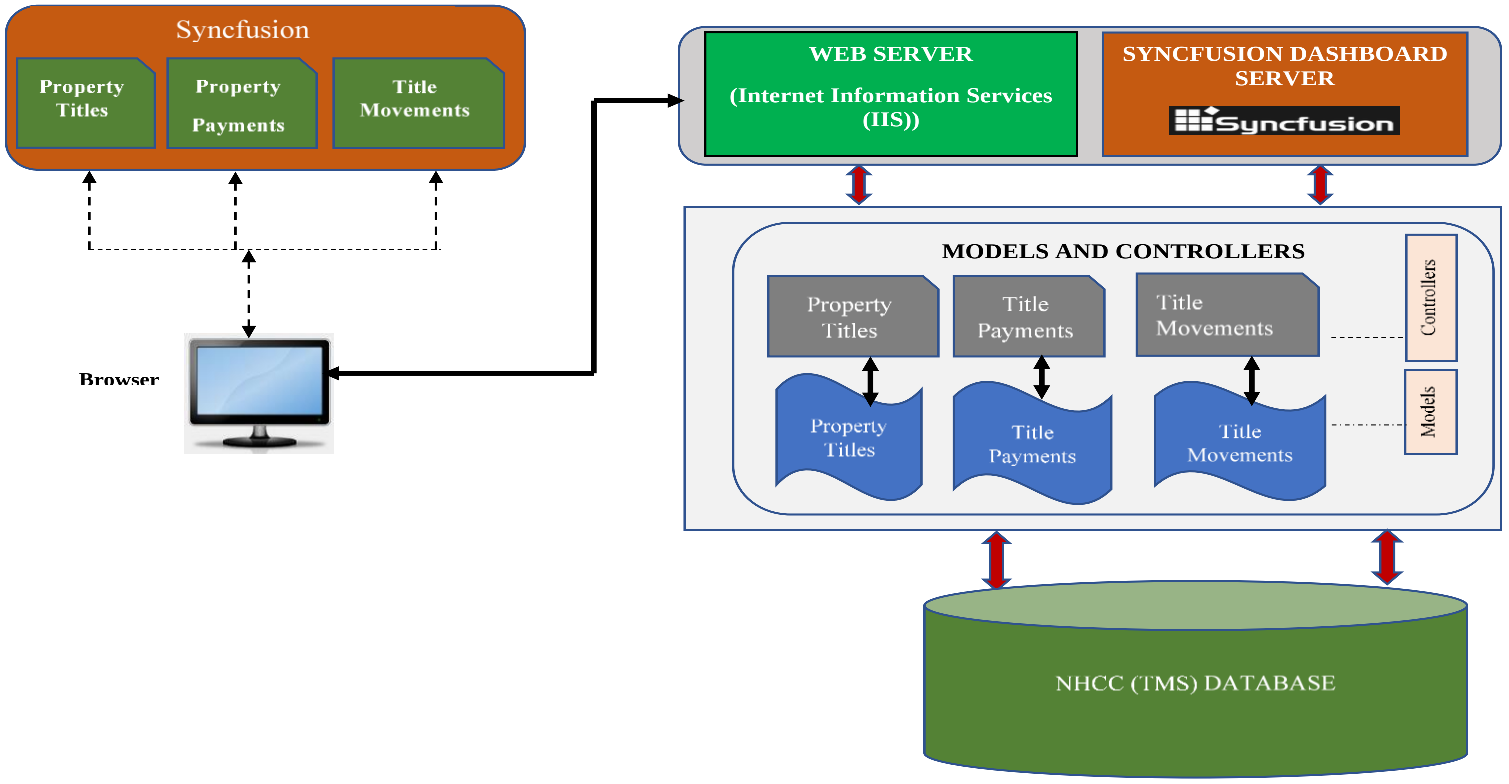


Figure 5:TMS Application Technology Architecture

3.3.1 Front End Applications

The TMS web application dashboards are developed using Syncfusion as a third-party component. The dashboards enable TMS users to keep track of real time data each time a change is made in the database.

3.3.2 Back End Technologies

3.3.2.1 TMS Web Application

The user of the TMS web application interacts with the TMS system through IIS (Internet Information Services) on the personal computer. The web application consists of three main modules namely: Adding Titles, Adding Title Payments and Moving existing titles.

When a user issues a request on the system in any of the modules, IIS carries out specific tasks as they are set in the controllers and views before returning information to the user.

The Models as shown in ***Figure 5*** above directly interface with the TMS database to send and retrieve data from the database; which is in turn sent to the controllers that perform the required tasks on the data that is accessed on the computers.

3.3.3 Middle Ware Technologies

3.3.3.1 Internet Information Services (IIS)

IIS is a web server that runs on the Microsoft .NET platform on the Windows Operating System. The TMS web application is registered to run on the TMS server through the Internet Information services as explained in **Section *2.2 above*.**

3.3.3.2 Syncfusion Dashboard Server

Syncfusion is a third-party component that is used to design dashboards on the server side of the TMS application and to ensure that the dashboards are up and running at all times. The Syncfusion Dashboard Server efficiently organizes and shares dashboards through a web interface.

The key features of the Syncfusion Dashboard Server include:

- **Dashboard management:** Efficiently organises dashboards into categories inorder to give view permissions to specific users or groups.
- **Designer Integration:** To seamlessly design and publish the dashboards from the Syncfusion Dashboard Designer application.

- **User Management:** Users are organized into groups to accurately map the structure of small and large organizations.
- **Scheduling:** This is a flexible functionality by which dashboards can be exported into an image file and emailed according to a schedule.
- **Flexible Permissions:** The scheme controls the access to read, write, delete and download Dashboards.
- **View Dashboards:** The built-in HTML 5 Dashboard Viewer control lets the user to view Dashboards from within the browser.
- **Export:** Dashboards can be exported to image file formats.
- **Custom Branding:** The Dashboard server has built-in customization capabilities such as allowing you to add your organization's name, logo, welcome note, and many more options.

3.3.4 Model – View – Controller Architecture

The TMS web application system is developed on the basis of the ***Model – View – Controller*** architecture.

3.3.4.1 Model

The Model is the central component of the Model-View-Controller architectural pattern as it interconnects both the Views and the Controller. The Model connects and communicates with the database to perform all the **CRUD (Create, Read, Update and Delete)** operations in the database. Therefore, it handles all the business logic.

3.3.4.2 Views

Views are a representation of information like charts, diagrams, tables, graphs, log-In screens and any other forms that the system is built with. Users of the web application issue commands through the View (Client side) depending on what actions they would like to carry out.

3.3.4.3 Controller

When users issue requests on the Views; (for example; by clicking a Load button to view a report), the request commands are sent to the controller that contains all classes that connect the Model and the View.

At the controller, the requests from the users are accepted and then converted to commands that can fetch data from the Model that is connected to the specific database. When the Model returns data to the Controller, the Controller passes back the data to the Views displaying the information the user requested for.

4 Database Design

4.1 Setting up the Database

To setup the database of the TMS System, Microsoft SQL Server 2016 should have been installed. Any of the following approaches can be used to successfully setup the database:

4.1.1 Attaching Database Files

The database has been created and kept in two files: NHCC_TMS.mdb and NHCC_TMS.mdf. To use this database, the two files have to be attached to an instance of the SQL Server as follows:

i. Copy the files to a selected location.
ii. Open SQL Server Management Studio as an Administrator.
iii. Open the Object explorer.
iv. Right-click the Database node and Select '**Attach…**'
v. The Attach database wizard pops-up as shown in ***Figure 6 below***:

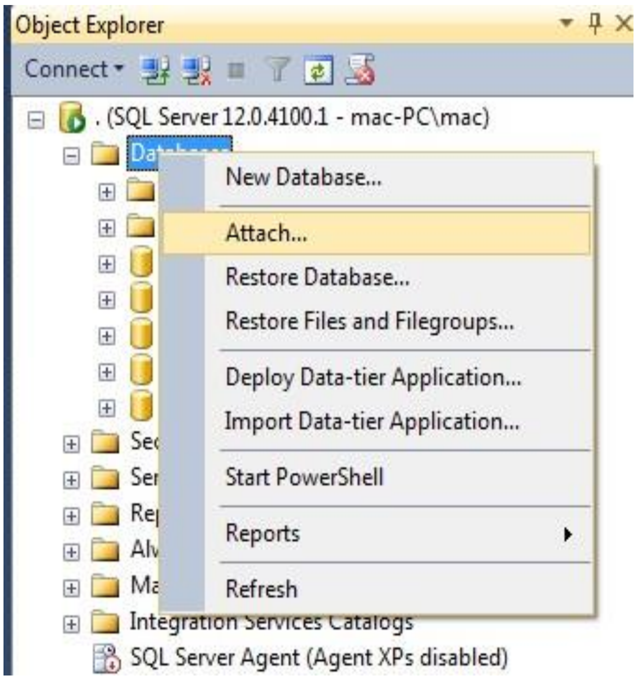
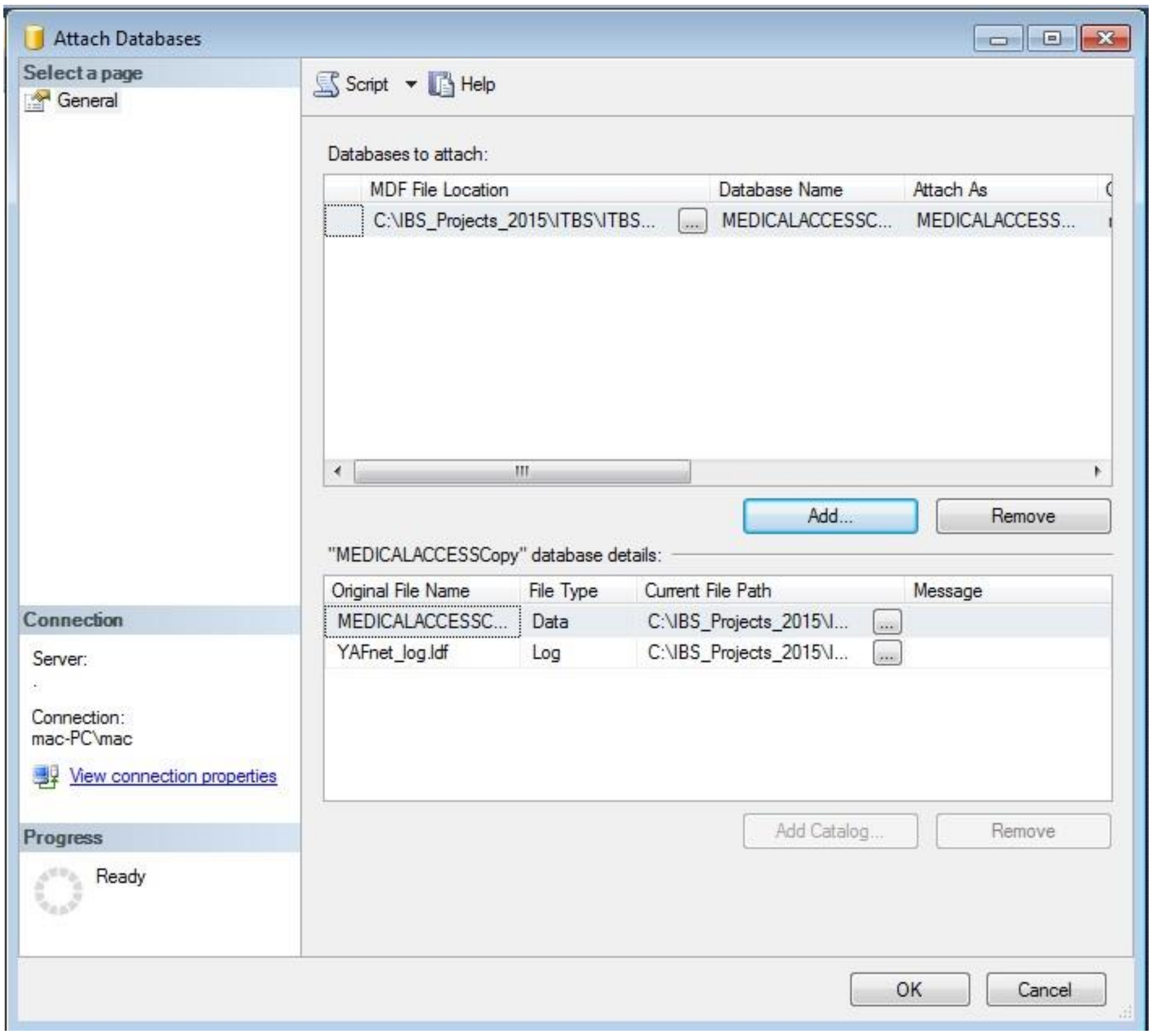

Object Explorer
Connect
. (SQL Server 12.0.4100.1 - mac-PC\mac)
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh
Integration Services Catalogs
SQL Server Agent (Agent XPs disabled)

Figure 6:Attach database file window pop up

vi. When you click on attach, a new window shown in ***Figure 7 below*** pops up.
vii. From the wizard, click on the '**Add**' button which opens the File selection dialog.



Connection

Server:
.

Connection:
mac-PC\mac

MDF File Location	Database Name	Attach As
C:\IBS_Projects_2015\ITBS\ITBS...	MEDICALACCESSC...	MEDICALACCESS...

"MEDICALACCESSCopy" database details:

Original File Name	File Type	Current File Path	Message
MEDICALACCESSC...	Data	C:\IBS_Projects_2015\I...	
YAFnet_log.ldf	Log	C:\IBS_Projects_2015\I...	

Figure 7:Add database file attachment window

viii. Navigate to the file storage location selected in (i) above

ix. Select the file and click OK.

NOTE. If the Systems Administrator is good at SQL, he/she can use command line to attach the database.

4.1.2 Running an SQL Script

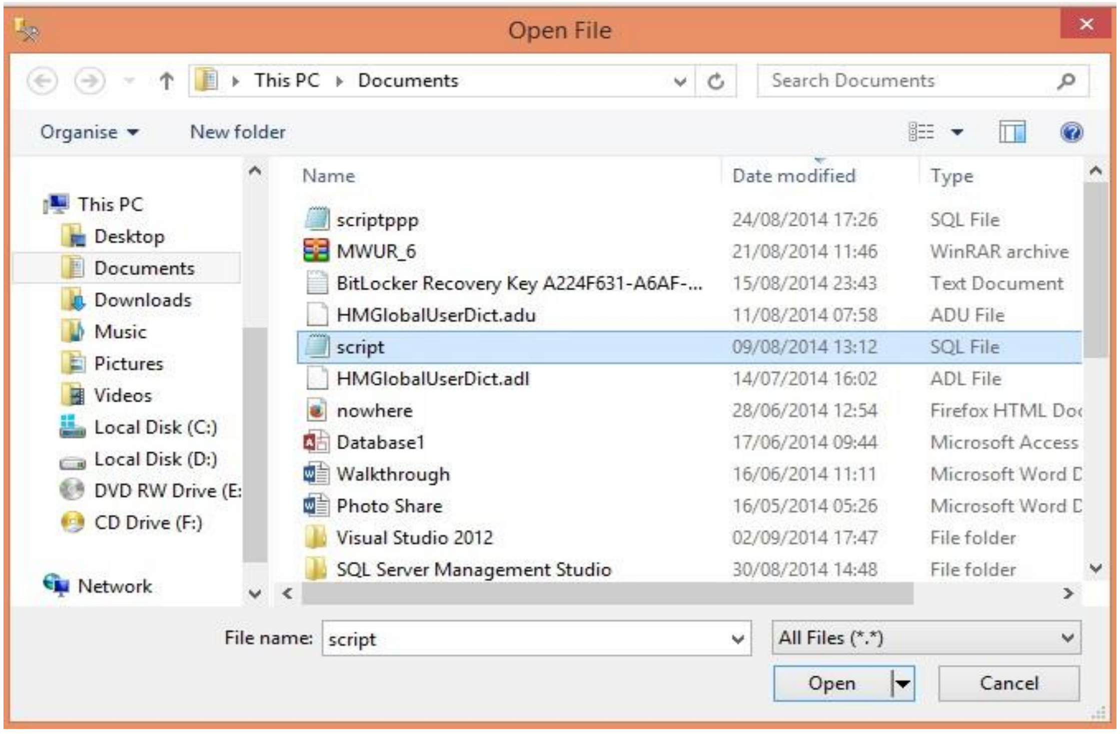
An alternative is to script the entire database and run the script from a file as follows:

i. Establish the script file location (***Figure 8 below***).

ii. Open SQL Server Management Studio as an Administrator.

iii. Navigate the tree view to File -> Open -> File…

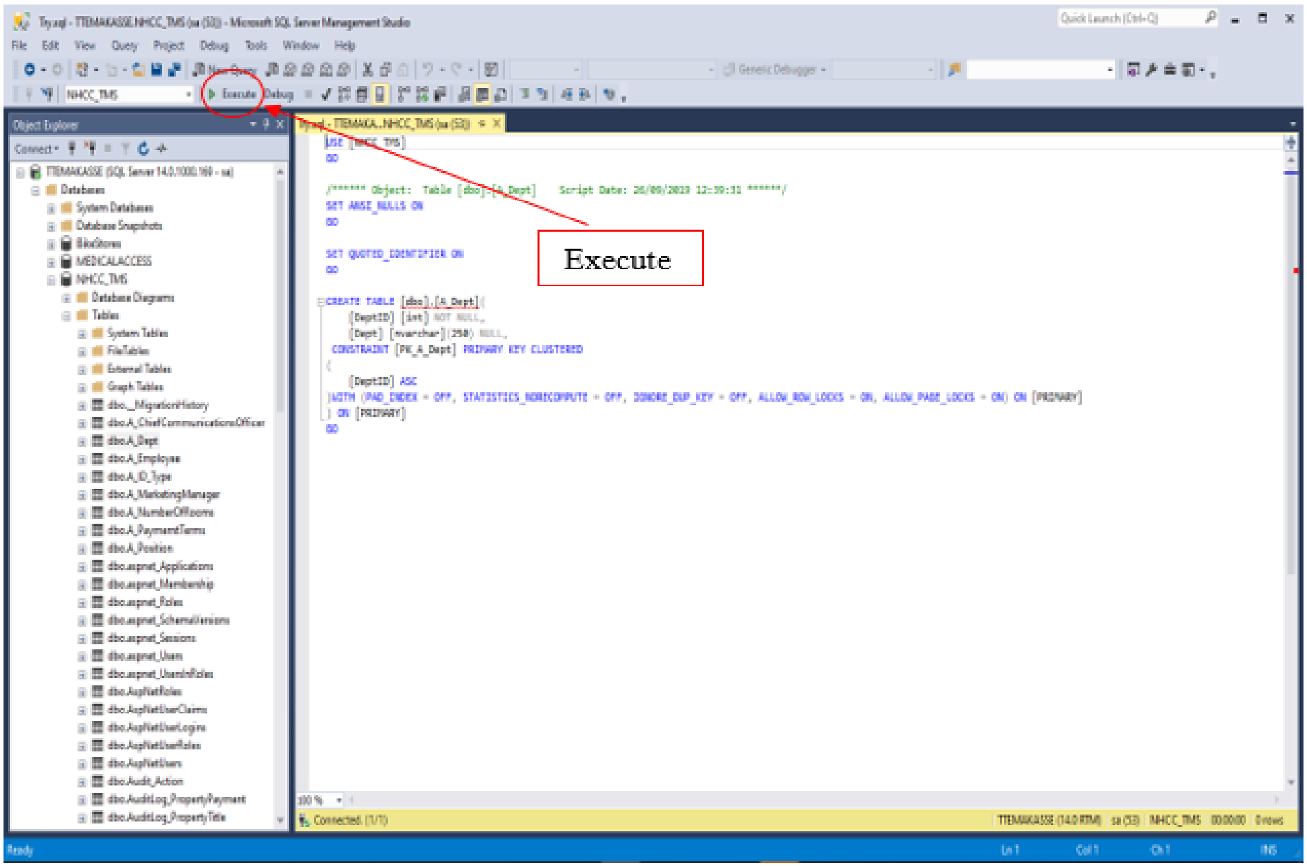
iv. Click on the Execute button (***Figure 9 below***).



Name	Date modified	Type
scriptppp	24/08/2014 17:26	SQL File
MWUR_6	21/08/2014 11:46	WinRAR archive
BitLocker Recovery Key A224F631-A6AF-...	15/08/2014 23:43	Text Document
HMGlobalUserDict.adu	11/08/2014 07:58	ADU File
script	09/08/2014 13:12	SQL File
HMGlobalUserDict.adl	14/07/2014 16:02	ADL File
nowhere	28/06/2014 12:54	Firefox HTML Doc
Database1	17/06/2014 09:44	Microsoft Access
Walkthrough	16/06/2014 11:11	Microsoft Word D
Photo Share	16/05/2014 05:26	Microsoft Word D
Visual Studio 2012	02/09/2014 17:47	File folder
SQL Server Management Studio	30/08/2014 14:48	File folder

File name: script
All Files (*.*)

Figure 8: Navigating to the script location



```
USE [NHCC_TMS]
GO

/****** Object:  Table [dbo].[A_Dept]    Script Date: 26/09/2019 12:59:31 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[A_Dept](
    [DeptID] [int] NOT NULL,
    [Dept] [nvarchar](250) NULL,
 CONSTRAINT [PK_A_Dept] PRIMARY KEY CLUSTERED
(
    [DeptID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figure 9:Executing the script in SQL Server Management Studio

4.1.3 Restoring a Backed-Up Database

Please refer to ***Section 4.3*** that gives details of how to restore a database from a backup.

4.2 Database Backup

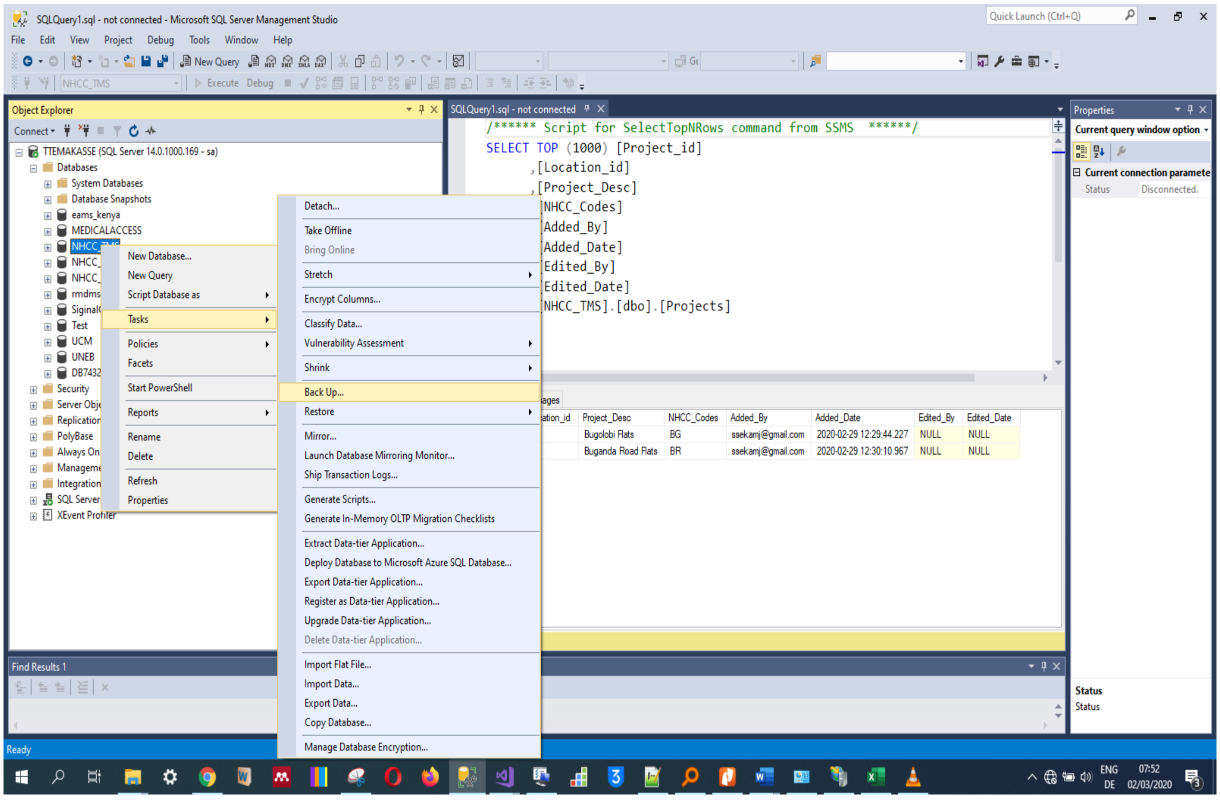
This section describes how to create a full database backup in SQL Server by using SQL Server Management Studio, Transact-SQL, or PowerShell.

4.2.1 Using SQL Server Management Studio to back up a database

i. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.

ii. Expand Databases, and select NHCC_TMS.

iii. Right-click the database (NHCC_TMS), point to **Tasks**, and then click **Back Up**.

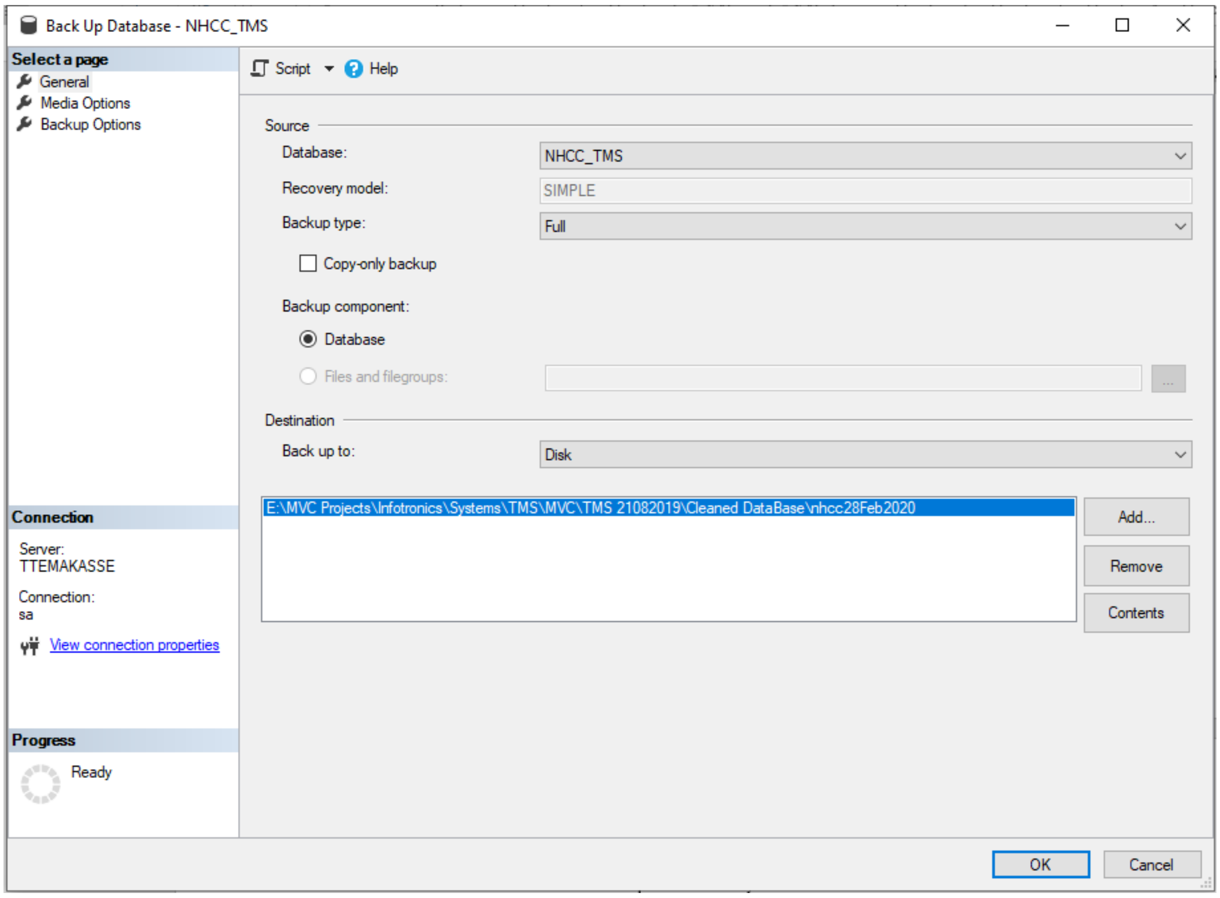


```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [Project_id]
      ,[Location_id]
      ,[Project_Desc]
      [NHCC_Codes]
      [Added_By]
      [Added_Date]
      [Edited_By]
      [Edited_Date]
      [NHCC_TMS].[dbo].[Projects]
```

ation_id	Project_Desc	NHCC_Codes	Added_By	Added_Date	Edited_By	Edited_Date
	Bugolobi Flats	BG	ssekamj@gmail.com	2020-02-29 12:29:44.227	NULL	NULL
	Buganda Road Flats	BR	ssekamj@gmail.com	2020-02-29 12:30:10.967	NULL	NULL

Figure 10:Creating a database back up

iv. On choosing Back Up, a new dialog window shown in ***Figure 11 below*** shows up.



Source

Database: NHCC_TMS

Recovery model: SIMPLE

Backup type: Full

☐ Copy-only backup

Backup component:

● Database

○ Files and filegroups:

Destination

Back up to: Disk

E:\MVC Projects\Infotronics\Systems\TMS\MVC\TMS 21082019\Cleaned DataBase\nhcc28Feb2020

Connection

Server:
TTEMAKASSE

Connection:
sa

Progress

Ready

Figure 11:Select database back up storage location

Points to note during database back up

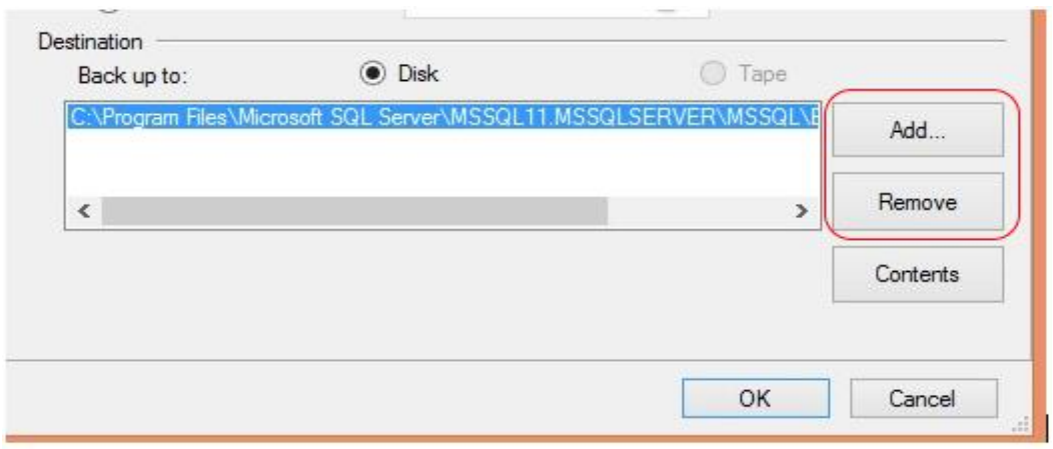
1. In the Database list box, verify the database name.
2. You can perform a database backup for any recovery model (FULL, BULK_LOGGED, or SIMPLE).
3. In the Backup type list box, select Full.
4. Alternatively, you can select **Copy Only Backup** to create a copy-only backup.
   A copy only backup is an SQL Server backup that is independent of the sequence of conventional SQL Server backups.

Note

When the **Differential** option is selected, you cannot create a copy-only backup

5. For Backup component, click Database.
6. Either accept the default backup set name suggested in the Name text box, or enter a different name for the backup set.

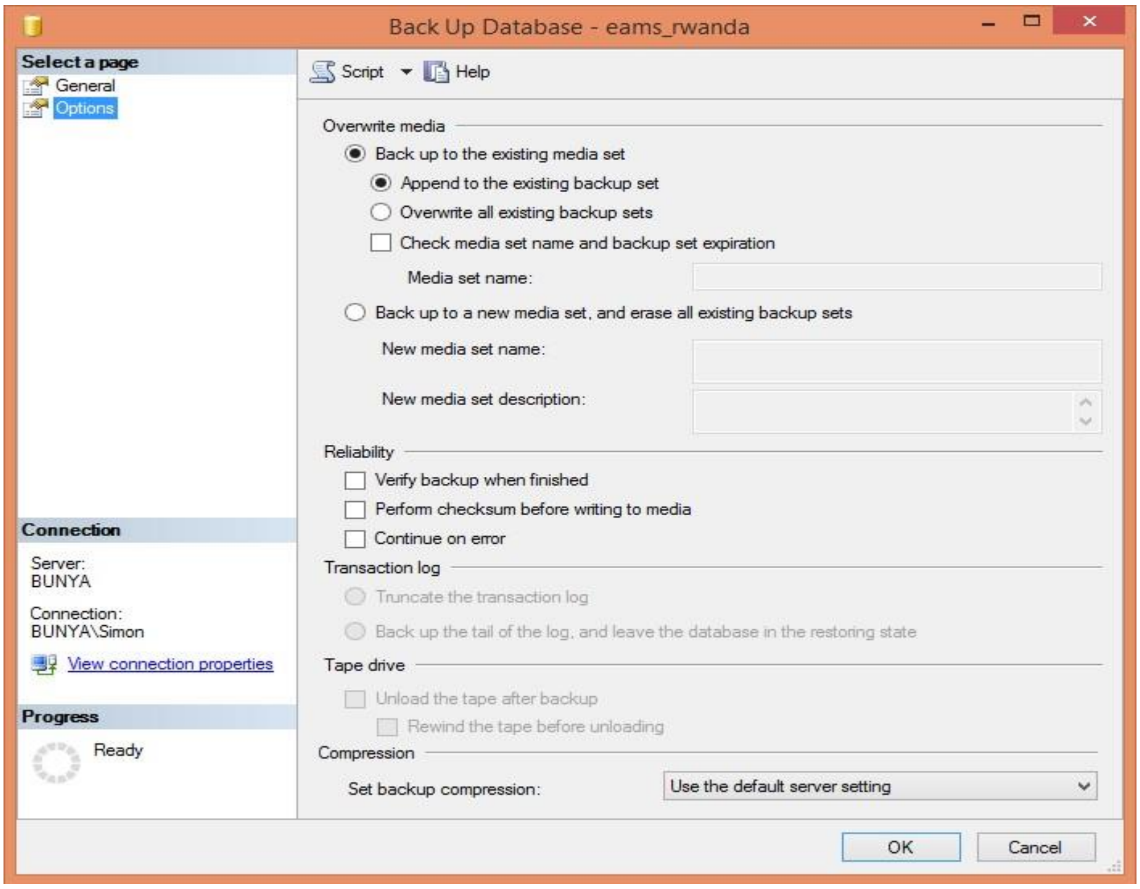
7. Alternatively, in the description text box, enter a description of the backup set.
8. Choose the type of backup destination by clicking Disk, Tape or URL. To select the paths of up to 64 disk or tape drives containing a single media set, click Add. The selected paths are displayed in the Backup to list box.
9. To remove a backup destination, select it and click Remove (***Figure 12 below***). To view the contents of a backup destination, select it and click Contents.



C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\E

Figure 12:Change database back up location

10. To view or select the media options, click Media Options in the Select a page pane.



Overwrite media
- (●) Back up to the existing media set
  - (●) Append to the existing backup set
  - ( ) Overwrite all existing backup sets
  - [ ] Check media set name and backup set expiration
  - Media set name:
- ( ) Back up to a new media set, and erase all existing backup sets
  - New media set name:
  - New media set description:

Reliability
- [ ] Verify backup when finished
- [ ] Perform checksum before writing to media
- [ ] Continue on error

Transaction log
- ( ) Truncate the transaction log
- ( ) Back up the tail of the log, and leave the database in the restoring state

Tape drive
- [ ] Unload the tape after backup
- [ ] Rewind the tape before unloading

Compression
- Set backup compression: Use the default server setting

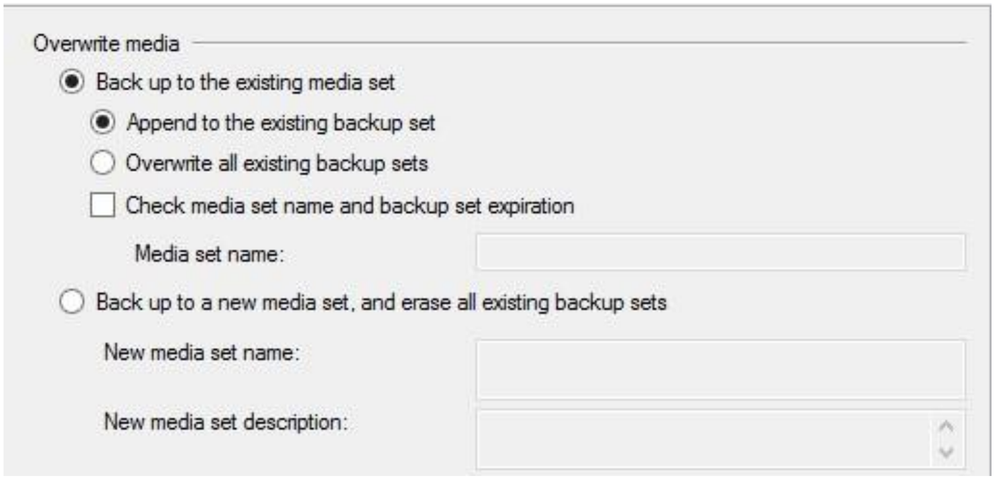
Connection
Server: BUNYA
Connection: BUNYA\Simon
View connection properties

Progress
Ready

11. Select an Overwrite Media option, by clicking one of the following:

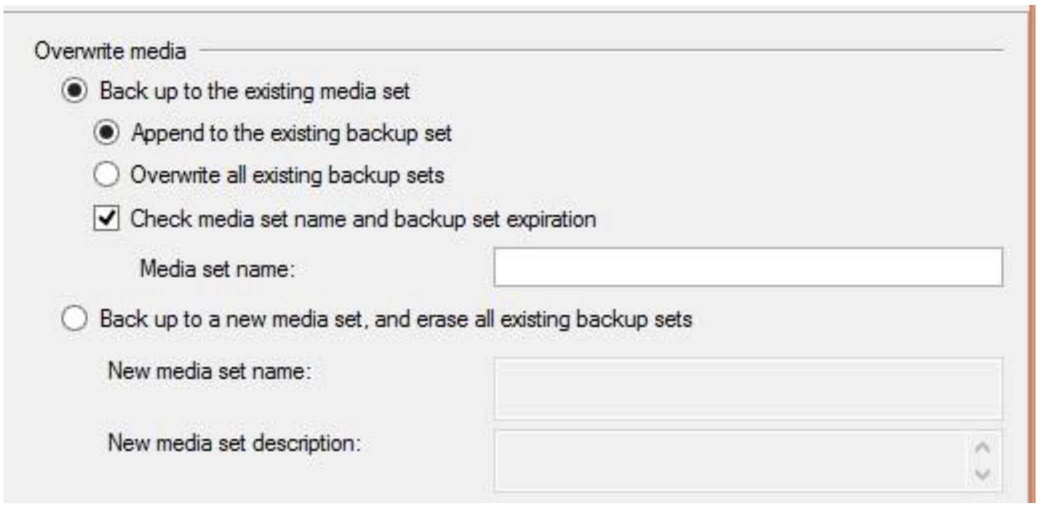
➢ **Back up to the existing media set**

For this option, click either Append to the existing backup set or overwrite all existing backup sets.



Overwrite media
- (●) Back up to the existing media set
  - (●) Append to the existing backup set
  - ( ) Overwrite all existing backup sets
  - [ ] Check media set name and backup set expiration
  - Media set name:
- ( ) Back up to a new media set, and erase all existing backup sets
  - New media set name:
  - New media set description:

Alternatively, select **Check media set name and backup set expiration** to cause the backup operation to verify the date and time at which the media is set and the backup is set to expire.


Overwrite media

(●) Back up to the existing media set

(●) Append to the existing backup set

( ) Overwrite all existing backup sets

[✓] Check media set name and backup set expiration

Media set name:

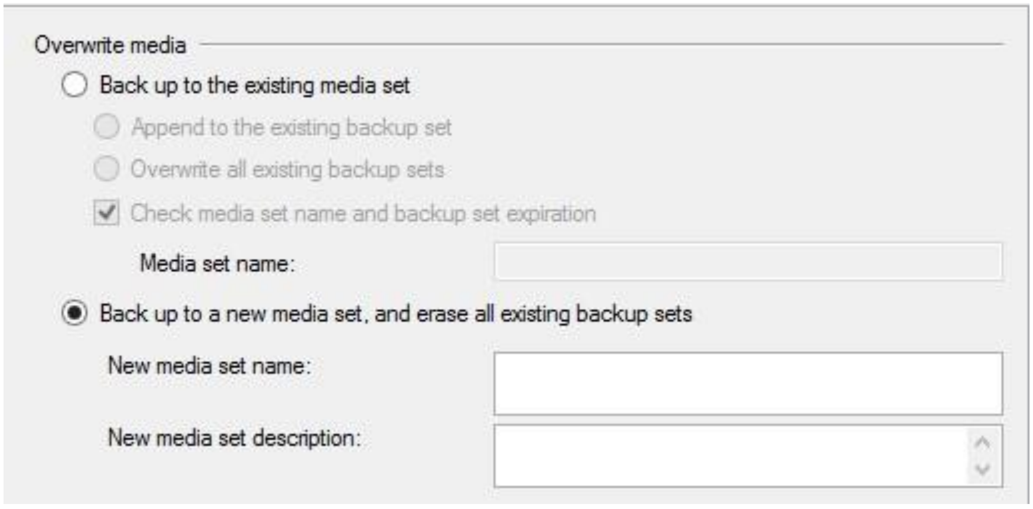
( ) Back up to a new media set, and erase all existing backup sets

New media set name:

New media set description:

Alternatively, enter a name in the **Media set name** text box. If no name is specified, a media set with a blank name is created. If you specify a media set name, the media (tape or disk) is checked to see whether the actual name matches the name you enter here.

- **Back up to a new media set, and erase all existing backup sets**


Overwrite media

( ) Back up to the existing media set

( ) Append to the existing backup set

( ) Overwrite all existing backup sets

[✓] Check media set name and backup set expiration

Media set name:

(●) Back up to a new media set, and erase all existing backup sets

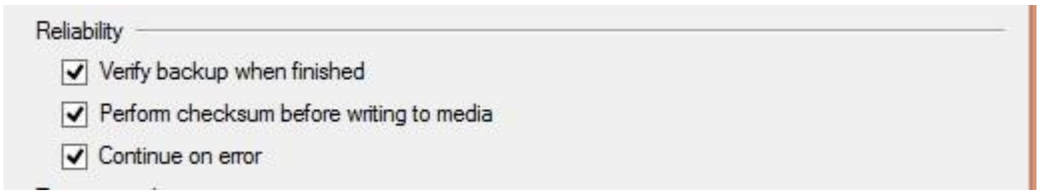
New media set name:

New media set description:

For this option, enter a name in the **New media set name** text box; alternatively, describe the media set in the **New media set description** text box.

12. In the Reliability section, alternatively check:

- Verify backup when finished.
- Perform checksum before writing to media, and,
- Continue on error. For information on checksums.

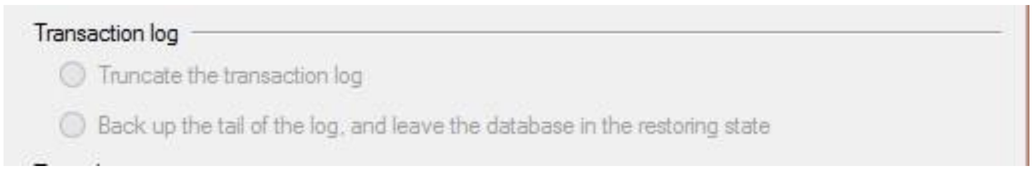

Reliability

- [✓] Verify backup when finished
- [✓] Perform checksum before writing to media
- [✓] Continue on error

13. If you are backing up to a tape drive (as specified in the destination section of the General page), the Unload the tape after backup option is active. Clicking this option activates Rewind the tape before unloading option.

Note

The options in the Transaction log section are inactive unless you are backing up a transaction log (as specified in the Backup type section of the General page).

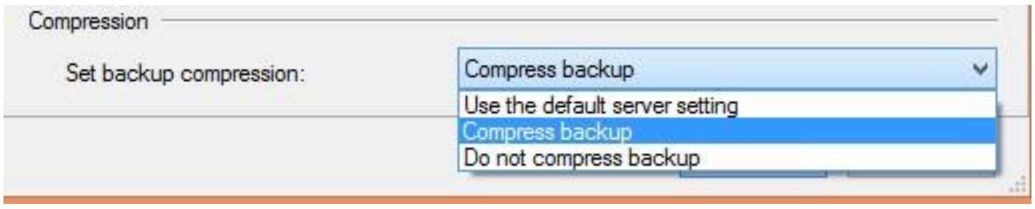

Transaction log

- ○ Truncate the transaction log
- ○ Back up the tail of the log, and leave the database in the restoring state

14. To view or select the backup options, click Backup Options in the Select a page pane.
16. Specify when the backup set will expire and can be overwritten without explicitly skipping verification of the expiration data:

- To have the backup set expire after a specific number of days, click after (the default option), and enter the number of days after set creation that the set will expire. This value can be from 0 to 99999 days; a value of 0 days means that the backup set will never expire.
- The default value is set in the default backup media retention (in days) option of the server properties dialog box (Database Settings Page). To access this, right-click the server name in Object Explorer and select properties; then select the Database Settings page.
- To have the backup set expire on a specific date, click On, and enter the date on which the set will expire.

17. SQL Server 2008 Enterprise and later supports backup compression. By default, whether a backup is compressed depends on the value of the **backup-compression default** server configuration option. However, regardless of the current server-level default, you can compress a backup by choosing **Compress backup**, or you can prevent compression by choosing **Do not compress backup**.



Compression

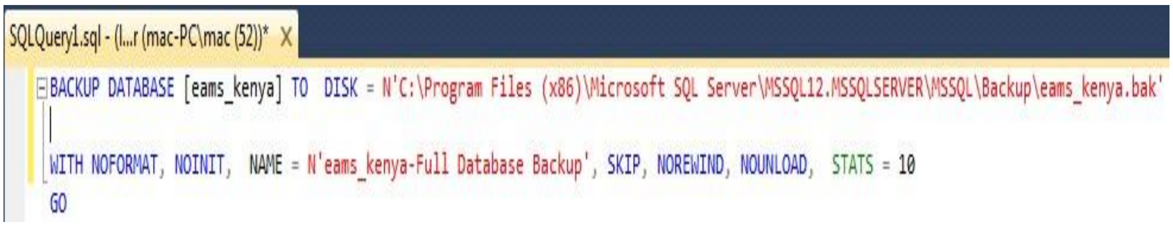
Set backup compression: Compress backup

- Use the default server setting
- Compress backup
- Do not compress backup

18. Specify whether to use encryption for the backup. Select an encryption algorithm to use for the encryption step, and provide a Certificate or Asymmetric key from a list of existing certificates or asymmetric keys. Encryption is supported in SQL Server 2014 or later.

Note

When you specify a backup task by using SQL Server Management Studio, you can generate the corresponding Transact-SQL BACKUP script by clicking the Script button and selecting a script destination.



```
BACKUP DATABASE [eams_kenya] TO  DISK = N'C:\Program Files (x86)\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\Backup\eams_kenya.bak'

WITH NOFORMAT, NOINIT,  NAME = N'eams_kenya-Full Database Backup', SKIP, NOREWIND, NOUNLOAD,  STATS = 10
GO
```

4.2.2 Using Transact-SQL

To create a full database back up using Transact-SQL

i. **Execute the BACKUP DATABASE statement to create the full database backup, specifying:**
    - The name of the database to back up.
    - The backup device where the full database backup is written.

The basic Transact-SQL syntax for a full database backup is:

BACKUP DATABASE NHCC_TMS

TO backup_device [**,**...n]

[WITH with_options [**,**...o]];

NHCC_TMS - Is the database that is to be backed up?

Specifies a list of 1 to 64 backup devices to use for the backup operation. You can specify a physical backup device, or you can specify a corresponding logical backup device, if already defined. To specify a physical backup device, use the DISK or TAPE backup_device

[option: **,**...n]

{ DISK | TAPE } = physical_backup_device_name

WITH

Alternatively, specifies one or more additional options, o. For with_options [information about some of the basic with options, **see step 2**.

,...o]

ii. Alternatively, specify one or more WITH options.

A few basic WITH options are described here.

{COMPRESSION | NO_COMPRESSION}

In SQL Server 2008 Enterprise and later versions, specifies whether backup compression is performed on this backup, overriding the server-level default.

ENCRYPTION (ALGORITHM, SERVER CERTIFICATE |ASYMMETRIC KEY)

In SQL Server 2014 and later versions, specify the encryption algorithm to use, and the Certificate or Asymmetric key to use to secure the encryption.

DESCRIPTION = { 'text' | @text variable }

Specifies the free-form text that describes the backup set. The string can have a maximum of 255 characters.

NAME = { backup_set_name | @backup_set_name_var }

Specifies the name of the backup set. Names can have a maximum of 128 characters.
If NAME is not specified, it is blank.

iii. Basic backup set WITH options.

By default, BACKUP appends the backup to an existing media set, preserving existing backup sets. To explicitly specify this, use the NOINIT option. For information about appending to existing backup sets.

Alternatively, to format the backup media, use the FORMAT option:

FORMAT [, MEDIANAME= { media_name | @media_name_variable }] [, MEDIADESCRIPTION = { text | @text_variable }]

Use the FORMAT clause when you are using media for the first time or you want to overwrite all existing data. Alternatively, assign the new media a media name and description.

Important

Take extreme caution when you are using the FORMAT clause of the BACKUP statement because this destroys any backups that were previously stored on the backup media.

4.2.3 Backup Examples (Transact-SQL)

4.2.3.1 Backing up to a disk device

The following example backs up the complete NHCC_TMS database to disk, by using **FORMAT** to create a new media set.

Transact-SQL

```
USE NHCC_TMS;
GO
BACKUP DATABASE NHCC_TMS
TO DISK = 'Z:\SQLServerBackups\ NHCC_TMS.Bak'
  WITH FORMAT,
    MEDIANAME = 'Z_SQLServerBackups',
    NAME = 'Full Backup of NHCC_TMS;
GO
```

4.2.3.2 Backing up to a tape device

The following example backs up the complete NHCC_TMS database to tape, appending the backup to the previous backups.

Transact-SQL

```
USE NHCC_TMS;
GO
BACKUP DATABASE NHCC_TMS
  TO TAPE = '\\.\Tape0'
  WITH NOINIT,
        NAME = 'Full Backup of NHCC_TMS;
    GO
```

4.2.3.3 Backing up to a logical tape device

The following example creates a logical backup device for a tape drive. The example then backs up the complete NHCC_TMS database to that device.

```
Transact-SQL
-- Create a logical backup device,
-- NHCC_TMS_Bak_Tape, for tape device \\.\tape0.
USE master;
GO
EXEC sp_addumpdevice 'tape', 'NHCC_TMS_Bak_Tape', '\\.\tape0';
USE NHCC_TMS;
GO
BACKUP DATABASE NHCC_TMS
  TO NHCC_TMS_Bak_Tape
  WITH FORMAT,
    MEDIANAME = 'NHCC_TMS_Bak_Tape',
    MEDIADESCRIPTION = '\\.\tape0',
    NAME = 'Full Backup of
NHCC_TMS'; GO
```

4.2.4 Backup Using PowerShell

- **Use the Backup-SqlDatabase cmdlet:** To explicitly indicate that this is a full database backup, specify the -BackupAction parameter with its default value, database. This parameter is optional for full database backups.

4.3 Restore a Database Backup

4.3.1 To restore a full database backup

i. After you connect to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree as shown in ***Figure 13 below***.

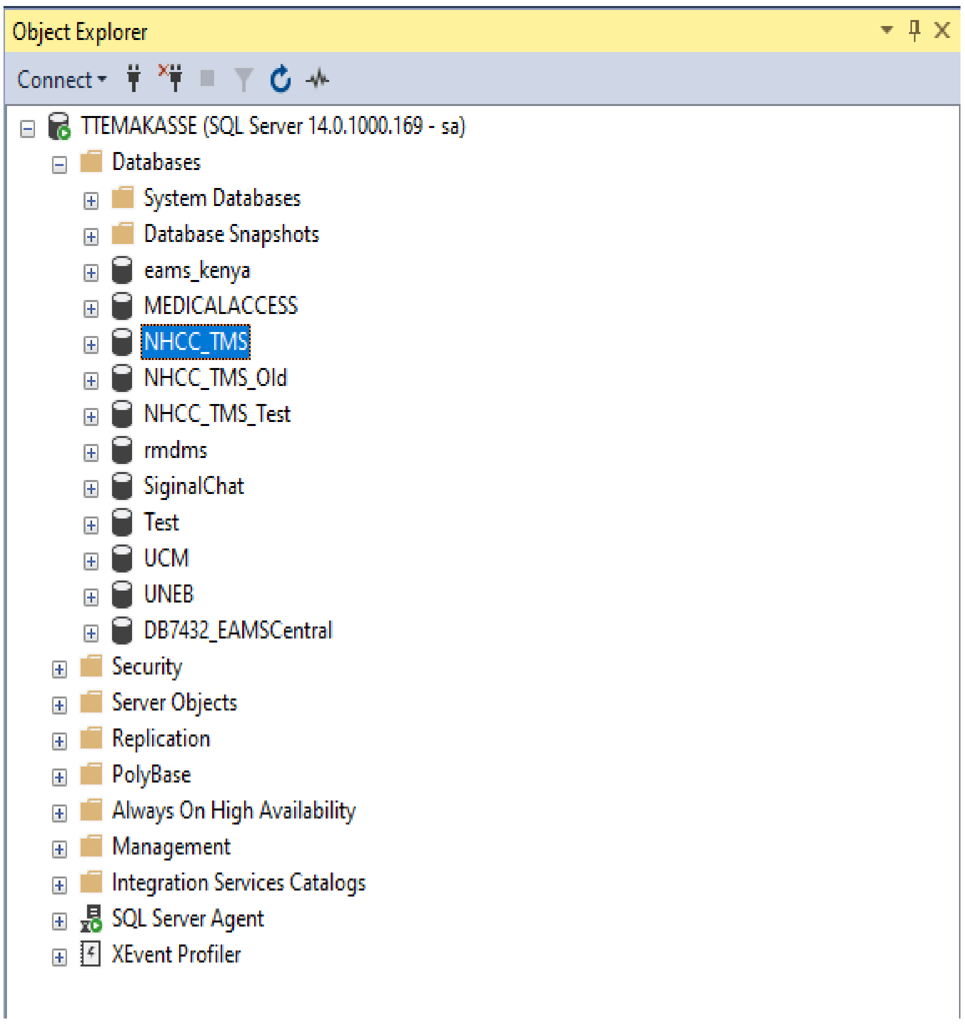
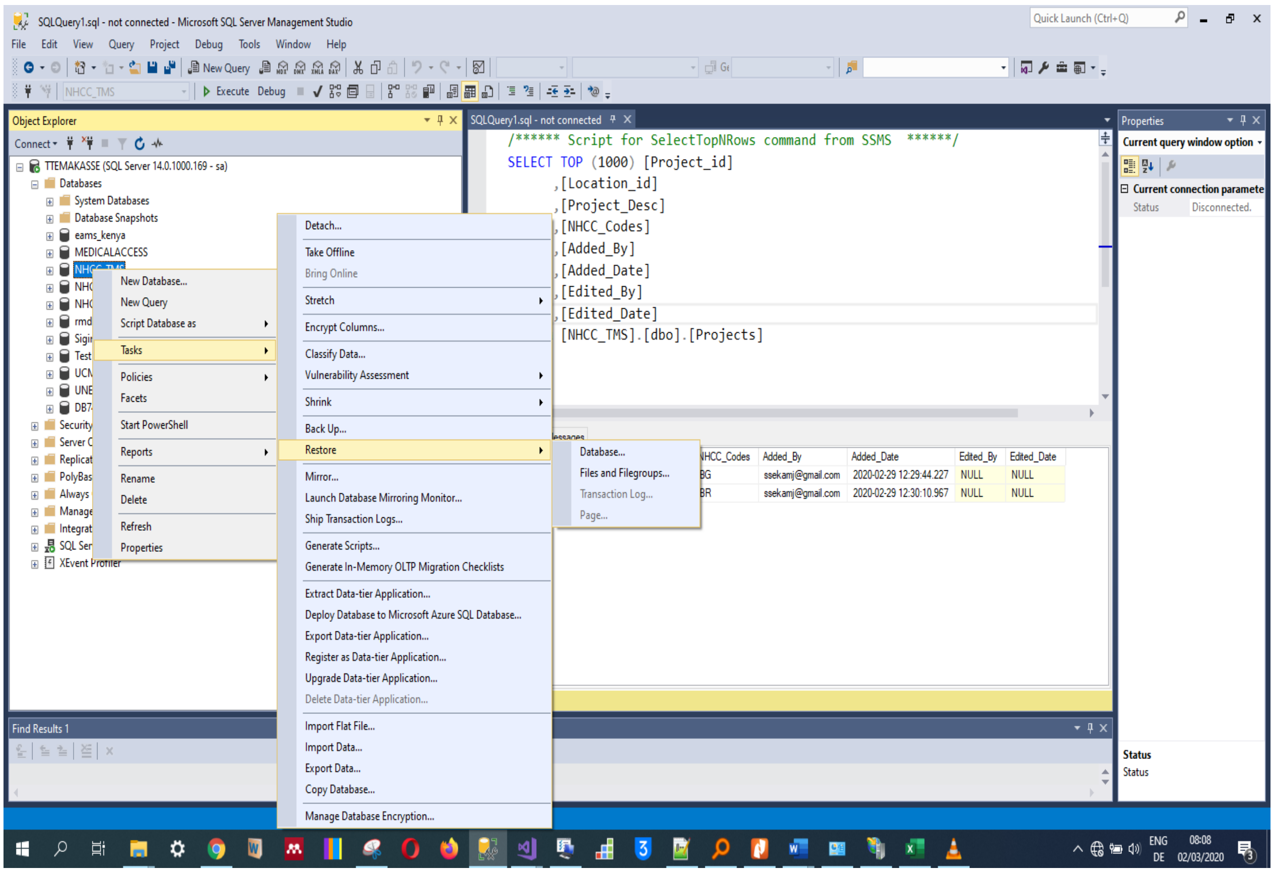


Figure 13:Database to restore to selection

ii. Expand Databases. Depending on the database, either select a user database or expand System Databases, and then select a system database.

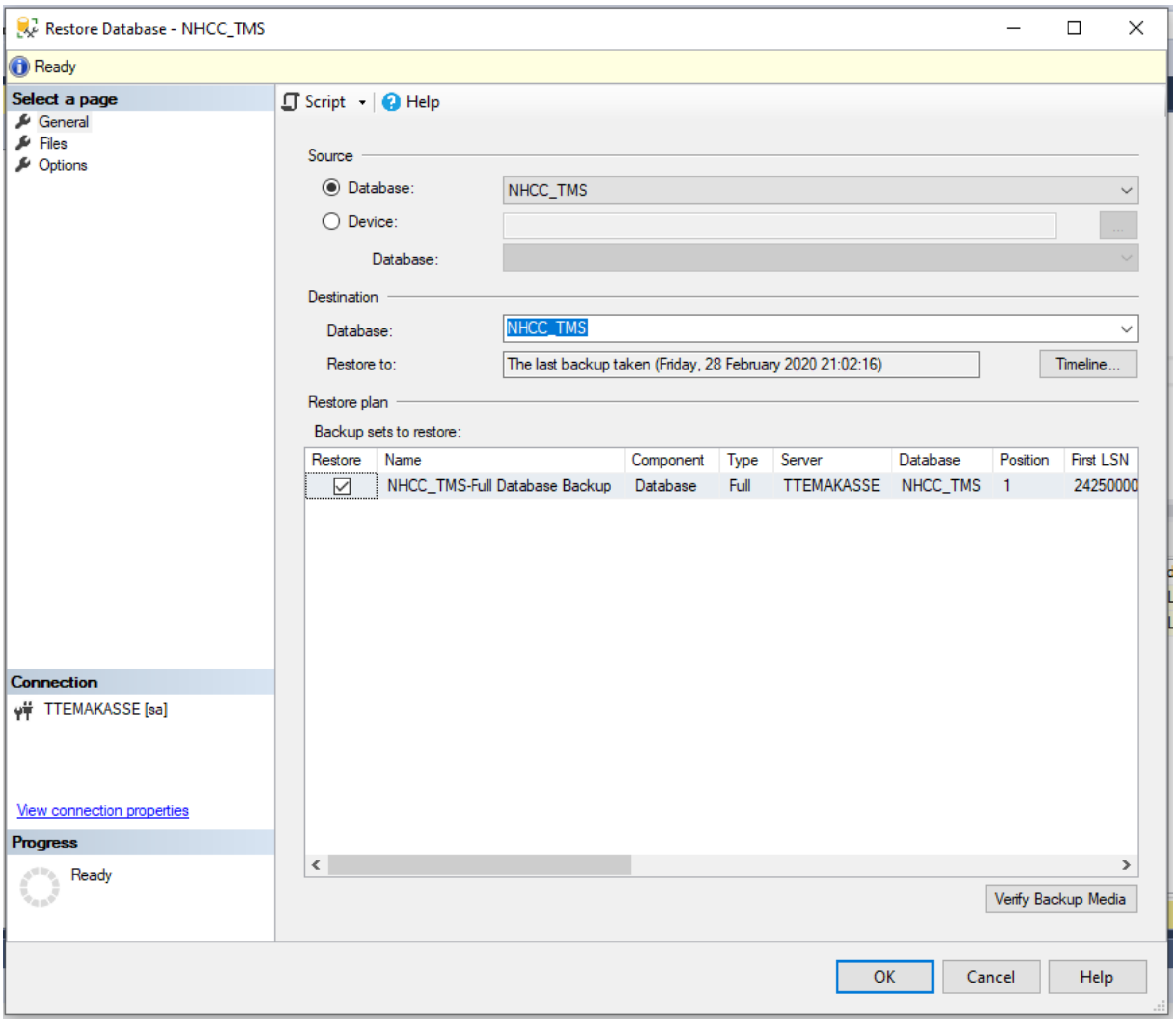
iii. Right-click the database, point to Tasks, point to Restore, and then click Database (***Figure 14***), which opens the Restore Database dialog box (***Figure 15 below***).



```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [Project_id]
      ,[Location_id]
      ,[Project_Desc]
      ,[NHCC_Codes]
      ,[Added_By]
      ,[Added_Date]
      ,[Edited_By]
      ,[Edited_Date]
  [NHCC_TMS].[dbo].[Projects]
```

HCC_Codes	Added_By	Added_Date	Edited_By	Edited_Date
BG	ssekamj@gmail.com	2020-02-29 12:29:44.227	NULL	NULL
BR	ssekamj@gmail.com	2020-02-29 12:30:10.967	NULL	NULL

Figure 14:Database Restore


Restore Database - NHCC_TMS

Ready

Select a page: General, Files, Options

Source
- Database: NHCC_TMS
- Device:
- Database:

Destination
- Database: NHCC_TMS
- Restore to: The last backup taken (Friday, 28 February 2020 21:02:16)

Restore plan

Backup sets to restore:

Restore	Name	Component	Type	Server	Database	Position	First LSN
☑	NHCC_TMS-Full Database Backup	Database	Full	TTEMAKASSE	NHCC_TMS	1	24250000

Connection: TTEMAKASSE [sa]

Progress: Ready

Figure 15:Database restore selection dialog box

iv. On the General page, use the source section to specify the source and location of the backup sets to restore. Select one of the following options:

- **Database**

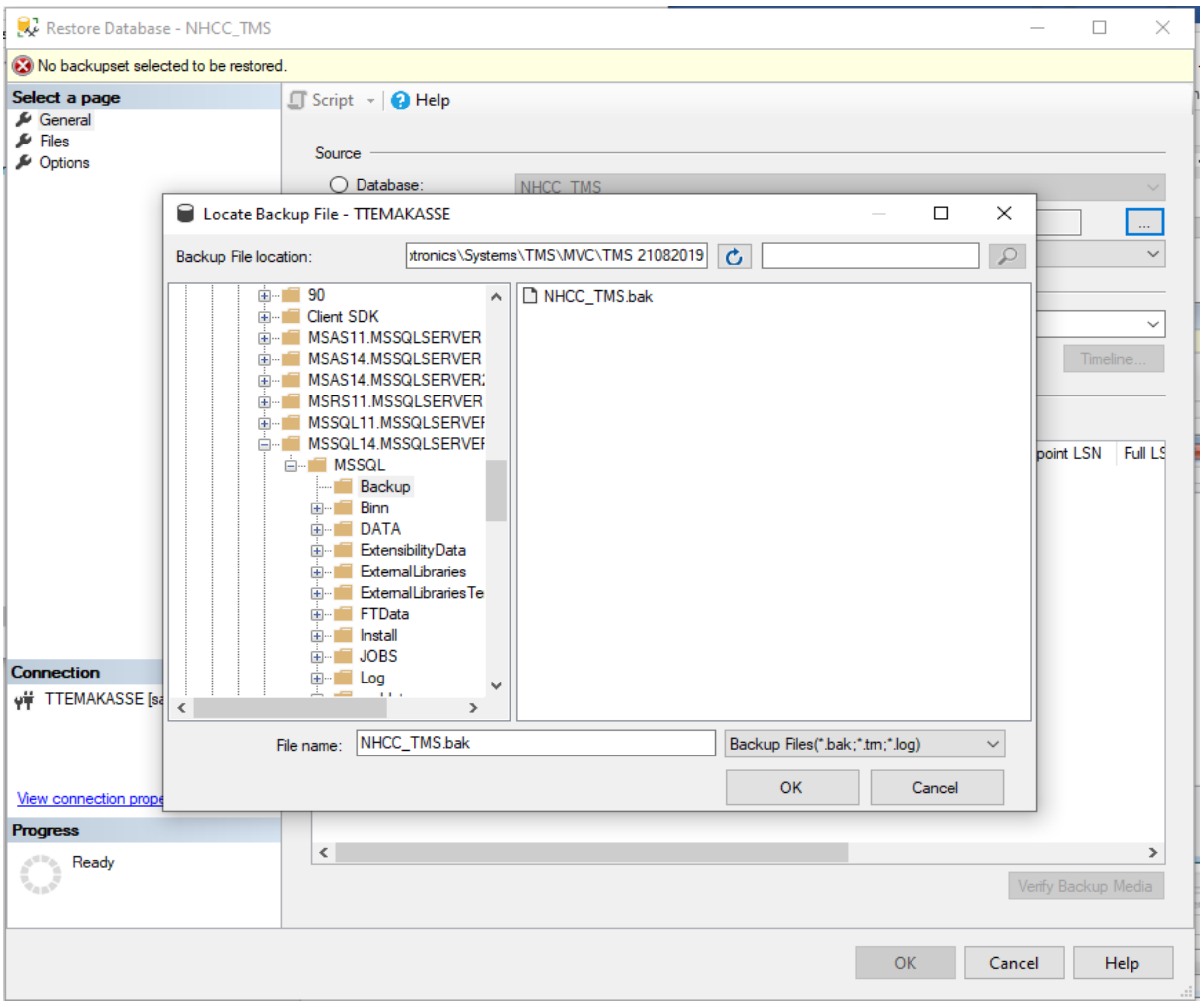
  Select the database to restore from the drop-down list. The list contains only databases that have been backed up according to the **msdb** backup history.

Note

If the backup is taken from a different server, the destination server will not have the backup history information for the specified database. In this case, select **Device** to manually specify the file or device to restore.

❖ Device

- Click the browse (...) button to open the Select backup devices dialog box. In the Backup media type box, select one of the listed device types. To select one or more devices for the Backup media box, click Add.
- Choose the database file to restore from the selected device (***Figure 16***)
- After adding the files, you need to back up from a specific device, click **OK** to get back to the restore general screen.



•

Figure 16:Choosing back up file to restore

Note

This list is only available when Device is selected. Only databases that have backups on the selected device will be available.

Select the medium for the restore operation: File, Tape, URL or Backup Device. The Tape option appears only if a tape drive is mounted on the computer, and the Backup Device option appears, only if at least one backup device exists.

Backup location

View, add, or remove media for the restore operation. The list can contain up to 64 files, tapes, or backup devices.

Add

Adds the location of a backup device to the Backup location list. Depending on the type of media you select in the Backup media field,

Media

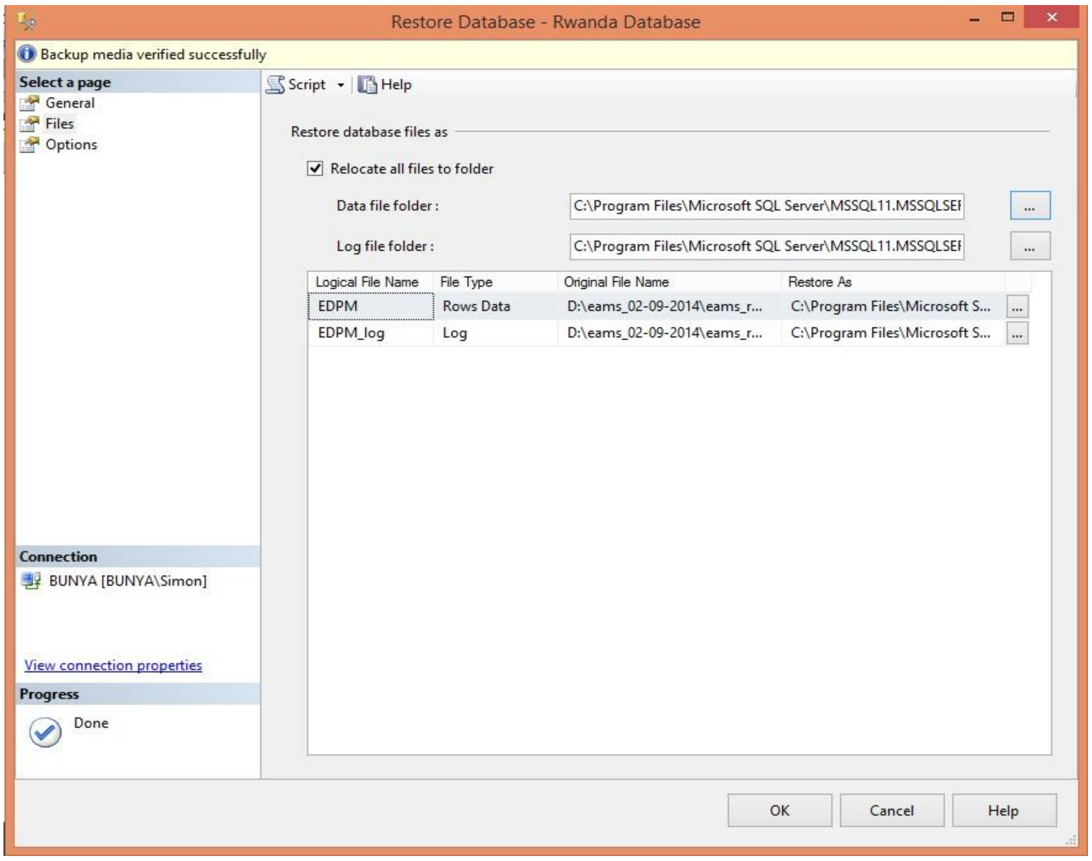
Dialog box	**Description type**
File	Locate the back up file or specify the remote file using its fully qualified universal naming convention (UNC) name
Device Select Backup Device	In this dialog box, you can select from a list of the logical backup devices defined on the server instance.
Select Backup Tape	In this dialog box, you can select from a list of the tape drives that are physically connected to the computer running the instance of the SQL server.

v. In the Destination section, the Database box is automatically populated with the name of the database to be restored. To change the name of the database, enter the new name in the Database box.

vi. In the Restore to box, leave the default as To the last backup taken or click on Timeline to access the Backup Timeline dialog box to manually select a point in time to stop the recovery action. For more information on designating a specific point in time.

vii. In the Backup sets to restore grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. To override the suggested recovery plan, you can change the selections in the grid. Backups that depend on the restoration of an earlier backup are automatically deselected when the earlier backup is deselected.

viii. Alternatively, click Files in the Select a page pane to access the Files dialog box. From here, you can restore the database to a new location by specifying a new restore destination for each file in the Restore the database files as grid.


Restore Database - Rwanda Database

Backup media verified successfully

Select a page
- General
- Files
- Options

Script | Help

Restore database files as

☑ Relocate all files to folder

Data file folder : C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSEI

Log file folder : C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSEI

Logical File Name	File Type	Original File Name	Restore As
EDPM	Rows Data	D:\eams_02-09-2014\eams_r...	C:\Program Files\Microsoft S...
EDPM_log	Log	D:\eams_02-09-2014\eams_r...	C:\Program Files\Microsoft S...

Connection

BUNYA [BUNYA\Simon]

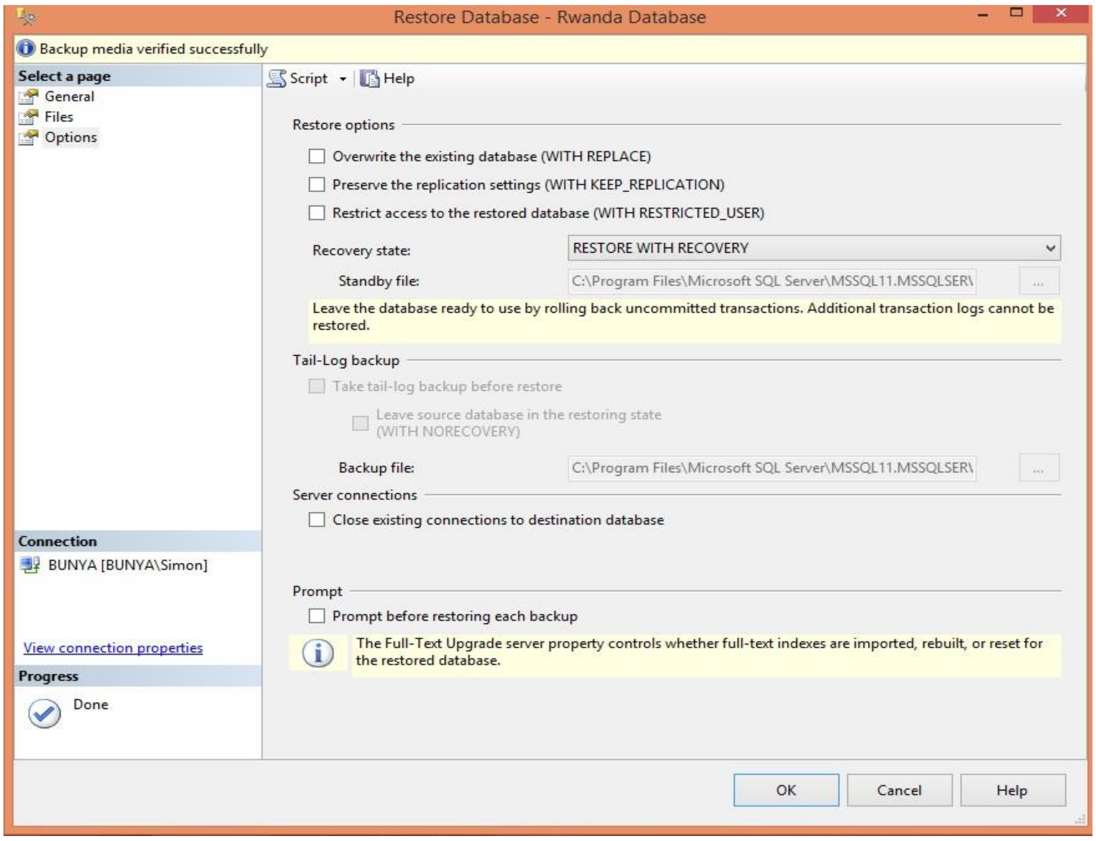
View connection properties

Progress

Done

OK Cancel Help

ix. To view or select the advanced options, on the Options page, in the Restore options panel, you can select any of the following options, if appropriate for your situation:


Restore Database - Rwanda Database

Backup media verified successfully

Restore options

- [ ] Overwrite the existing database (WITH REPLACE)
- [ ] Preserve the replication settings (WITH KEEP_REPLICATION)
- [ ] Restrict access to the restored database (WITH RESTRICTED_USER)

Recovery state: RESTORE WITH RECOVERY

Standby file: C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSER\

Leave the database ready to use by rolling back uncommitted transactions. Additional transaction logs cannot be restored.

Tail-Log backup

- [ ] Take tail-log backup before restore
- [ ] Leave source database in the restoring state (WITH NORECOVERY)

Backup file: C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSER\

Server connections

- [ ] Close existing connections to destination database

Connection

BUNYA [BUNYA\Simon]

View connection properties

Progress

Done

Prompt

- [ ] Prompt before restoring each backup

The Full-Text Upgrade server property controls whether full-text indexes are imported, rebuilt, or reset for the restored database.

❖ WITH options (not required):

- Overwrite the existing database (WITH REPLACE)
- Preserve the replication settings (WITH KEEP_REPLICATION)
- Restrict access to the restored database (WITH RESTRICTED_USER). Select an option for the Recovery state box. This box determines the state of the database after the restore operation.
- RESTORE WITH RECOVERY is the default behavior which leaves the database ready for use by rolling back the uncommitted transactions. Additional transaction logs cannot be restored. Select this option if you are restoring all of the necessary backups now.
- RESTORE WITH NORECOVERY which leaves the database nonoperational, and does not roll back the uncommitted transactions. Additional transaction logs can be restored. The database cannot be used until it is recovered.
- RESTORE WITH STANDBY which leaves the database in read-only mode. It undoes uncommitted transactions, but saves the undo actions in a standby file so that recovery effects can be reverted.

x. Take tail-log backup before restore will be selected if it is necessary for the point in time that you have selected. You do not need to modify this setting, but you can choose to back up the tail of the log even if it is not required. Filename here? If the first backup set in the General page is in Windows Azure, the tail log will also be backed up to the same storage container.

xi. Restore operations may fail if there are active connections to the database. Check the Close existing connections option to ensure that all active connections between Management Studio and the database are closed. This check box sets the database to single user mode before performing the restore operations, and sets the database to multi-user mode when complete.

xii. Select prompt before restoring each backup if you wish to be prompted between each restore operation. This is not usually necessary unless the database is large and you wish to monitor the status of the restore operation.

xiii. Click OK.

5 User Administration

Administrator: The Administrator module provides system management tools such as;

- **Manage Users:** (which includes: Register Users, Edit Users, Reset Passwords and Reset Passwords for other Users).
- **Manage Country,District,County and Subcounty:** (Includes Adding/Editing details about Countries,Districts,Countries and Subcounties).
- **Manage Location,Project and Region:** (Includes Adding/Editing details regarding Locations, Projects and Regions).
- **Manage Title Movement Purpose:** (Includes Adding/Editing Title Movement Purposes.
- **Manual Documents:** Section contains documents regarding information about the Title Management System.

5.1.1 Utilities

5.1.1.1 Register Users

To create a new user, select the Register option of the menu as shown in the figure below.

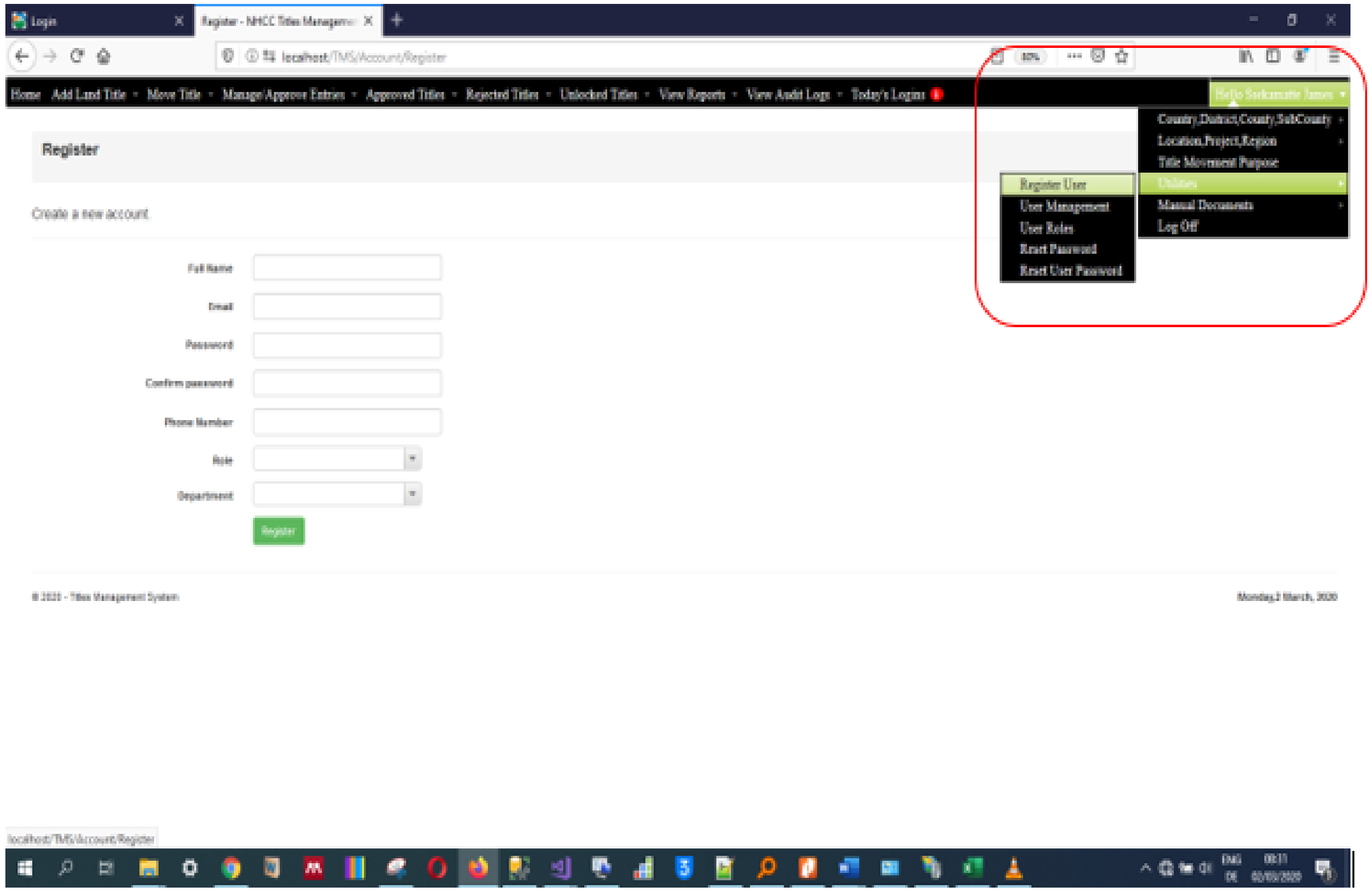
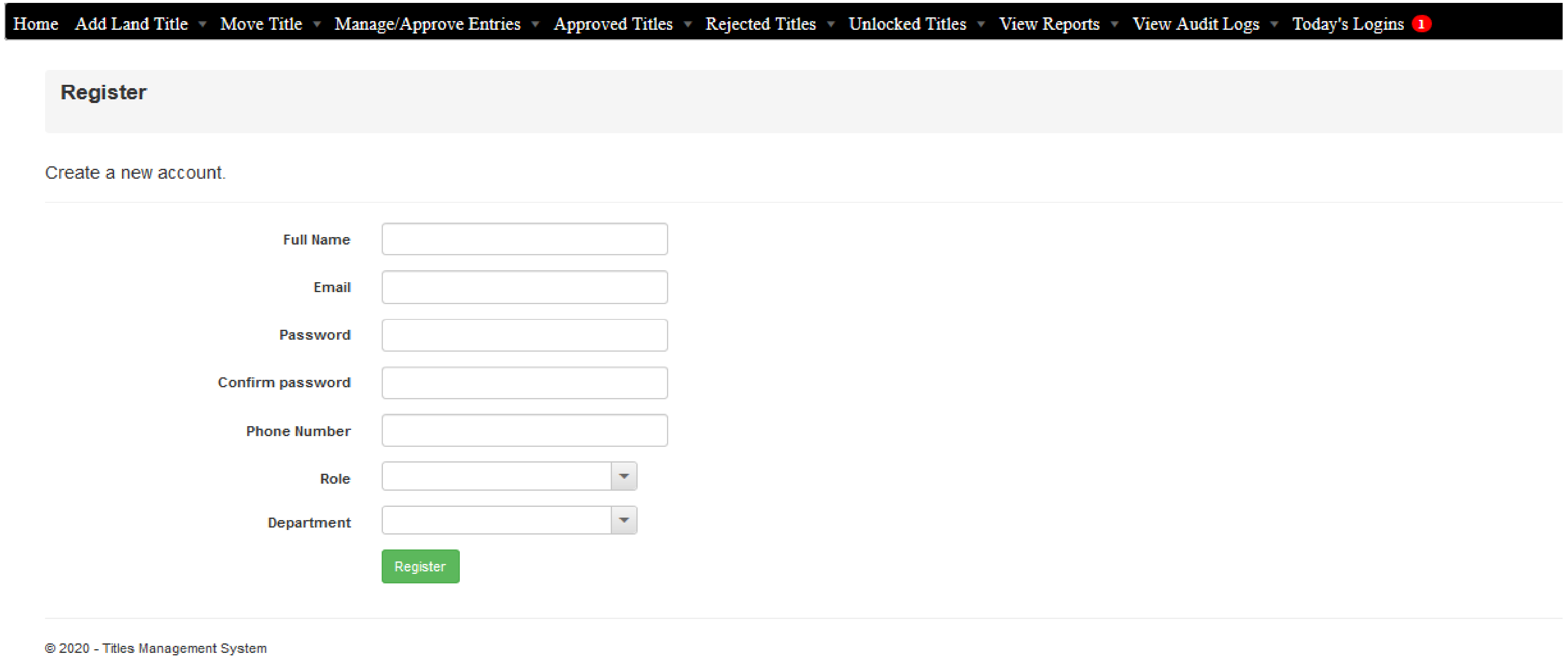


Figure 17:Register User Menu

The windows below (***Figure 18***) opens upon selection of Register Users as shown in the diagram below.



Register

Create a new account.

Full Name	
Email	
Password	
Confirm password	
Phone Number	
Role	
Department	

© 2020 - Titles Management System

Figure 18:Register User Form

Only users with administrator rights will have the privilege of creating a new user account and it requires:

Full Name: This fields captures the full names of the user being registered in the system.

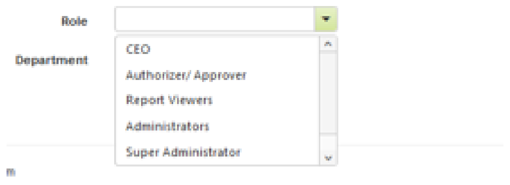
Email: This will be required for each registered user for log in into the TMS system.

Password: The password will be associated with the email for authentication during logging in to the system.

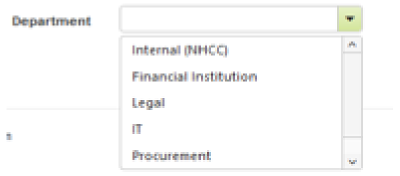
Confirm Password: It is used to ensure that the new user being registered is sure of the password they intend to use when interacting with the TMS system. Therefore, the passwords entered in the Password and Confirm Password fields should match.

Phone Number: Captures the users phone number for means of communication.

Role: Is a drop-down field that captures the type of a user being registered. For example, CEO, Authorizer/Approver and administrators.

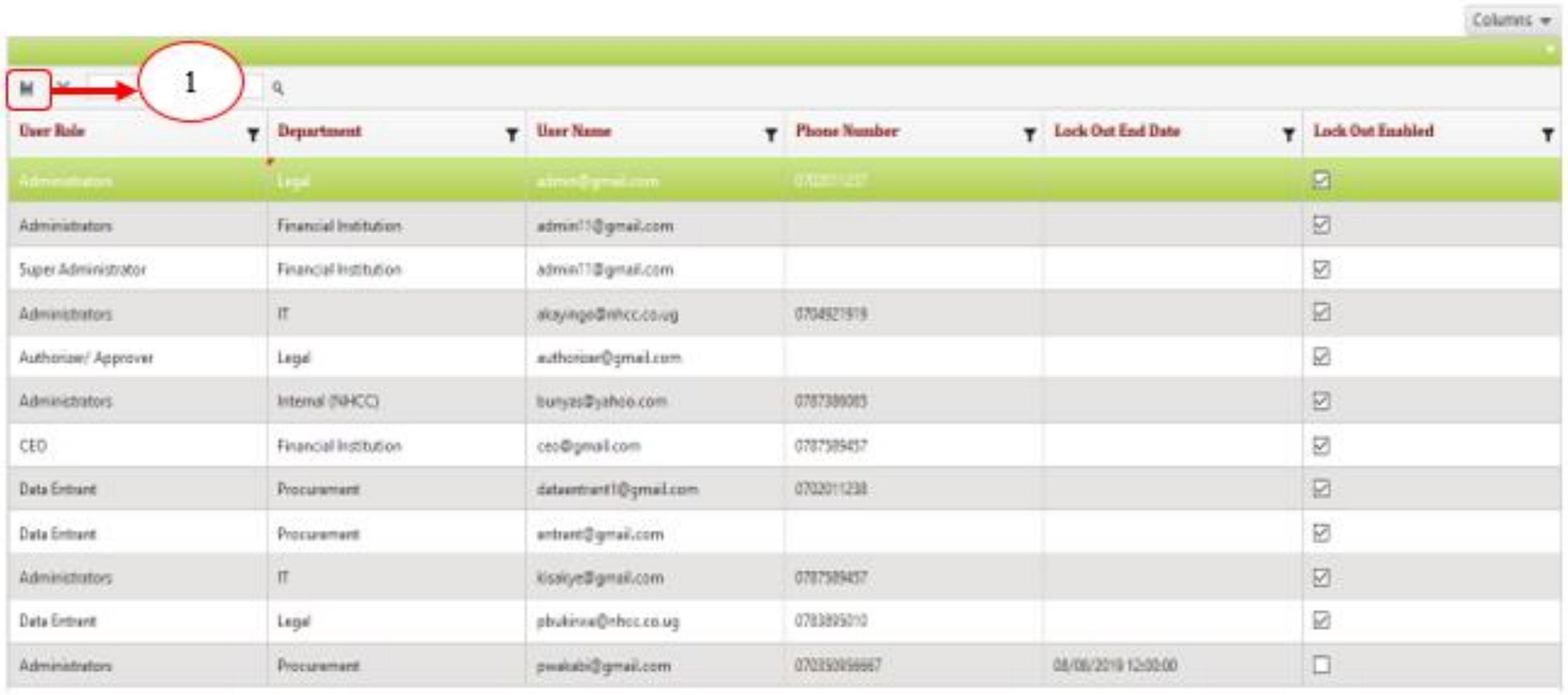


Department: Is a drop-down field that captures the department to which the user being registered is tagged to.



5.1.1.2 User Management

The User Management module shows the list of all users already registered into the system, and this module helps the administrator to edit these users. The administrator can edit user details like User Role, Department, Phone Number, and Locking out a certain user from the system.



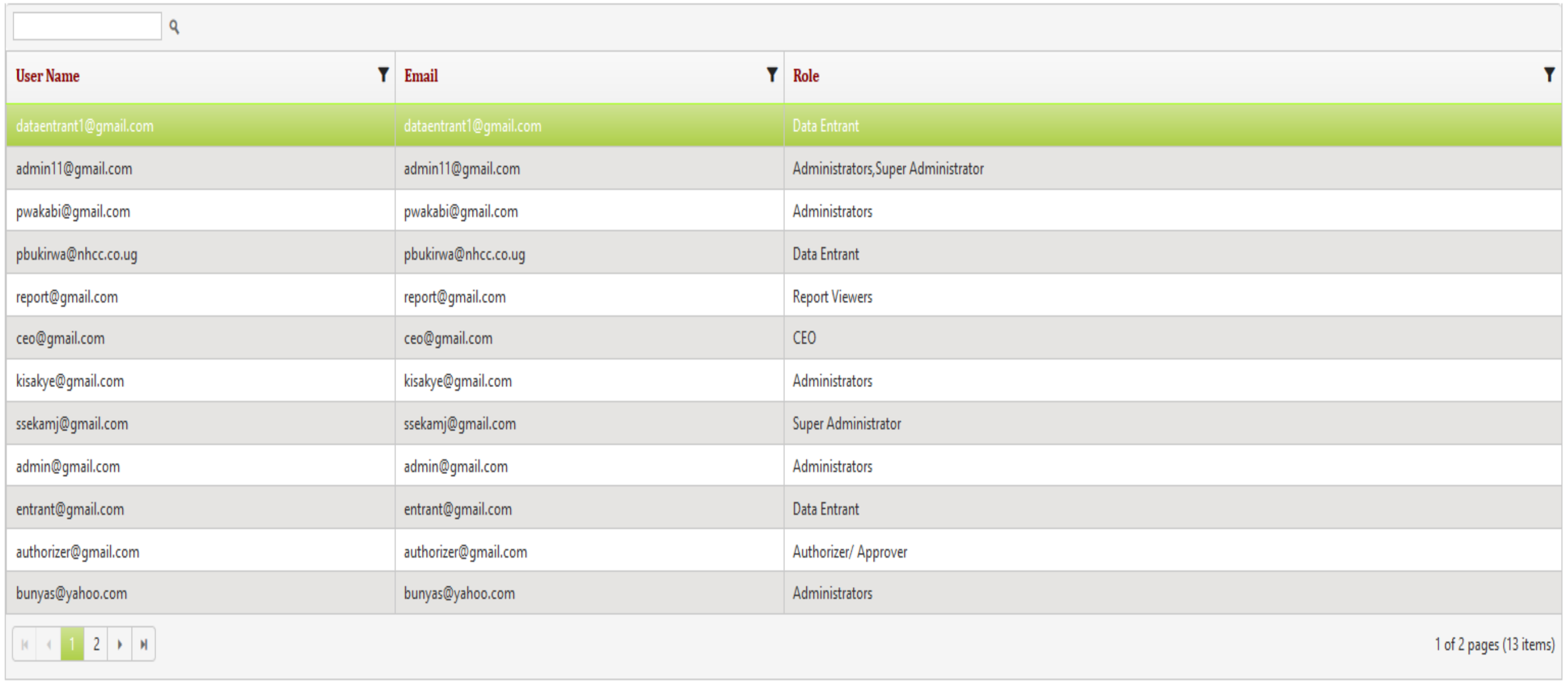
User Role	Department	User Name	Phone Number	Lock Out End Date	Lock Out Enabled
Administrators	Legal	admin@gmail.com	0702011237		✓
Administrators	Financial Institution	admin11@gmail.com			✓
Super Administrator	Financial Institution	admin11@gmail.com			✓
Administrators	IT	akayingo@nhcc.co.ug	0704921919		✓
Authorizer/ Approver	Legal	authorizer@gmail.com			✓
Administrators	Internal (NHCC)	bunyas@yahoo.com	0787388083		✓
CEO	Financial Institution	ceo@gmail.com	0787589457		✓
Data Entrant	Procurement	dataentrant1@gmail.com	0702011238		✓
Data Entrant	Procurement	entrant@gmail.com			✓
Administrators	IT	kisakye@gmail.com	0787589457		✓
Data Entrant	Legal	pbukirwa@nhcc.co.ug	0783895010		✓
Administrators	Procurement	pwakabi@gmail.com	07035095667	08/08/2019 12:00:00	☐

Figure 19:User Management

After making the changes about a user, the administrator can commit the changes to the database by clicking on the save button on the left top corner of the grid as shown by ***label 1*** in ***Figure 19 above***.

5.1.2 User Roles

The module displays details of registered users in the TMS system. The administrator can use this section to edit the user emails and the user roles.



User Name	Email	Role
dataentrant1@gmail.com	dataentrant1@gmail.com	Data Entrant
admin11@gmail.com	admin11@gmail.com	Administrators,Super Administrator
pwakabi@gmail.com	pwakabi@gmail.com	Administrators
pbukirwa@nhcc.co.ug	pbukirwa@nhcc.co.ug	Data Entrant
report@gmail.com	report@gmail.com	Report Viewers
ceo@gmail.com	ceo@gmail.com	CEO
kisakye@gmail.com	kisakye@gmail.com	Administrators
ssekamj@gmail.com	ssekamj@gmail.com	Super Administrator
admin@gmail.com	admin@gmail.com	Administrators
entrant@gmail.com	entrant@gmail.com	Data Entrant
authorizer@gmail.com	authorizer@gmail.com	Authorizer/ Approver
bunyas@yahoo.com	bunyas@yahoo.com	Administrators

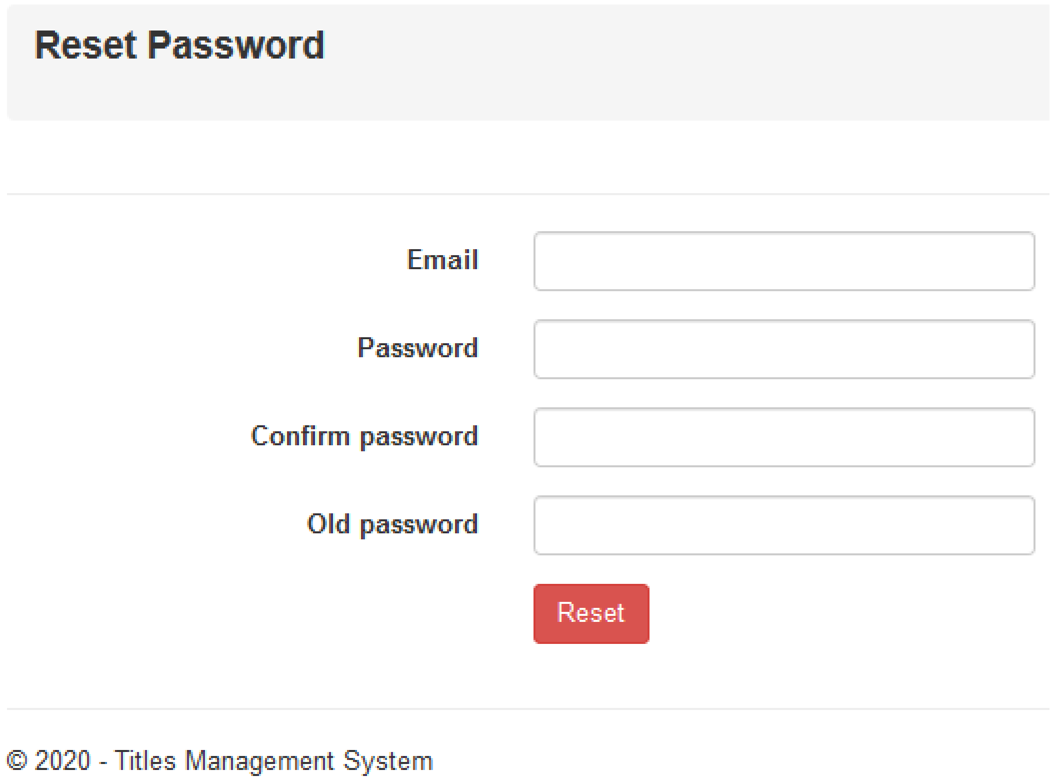
1 2

1 of 2 pages (13 items)

Figure 20:User Roles

5.1.3 Reset Password.

This module is used for changing the password for someone who is currently logged in to the system. However, it requires a user to be knowing the old/previous password she/he was given before changing to a new one.


Reset Password

Email	
Password	
Confirm password	
Old password	

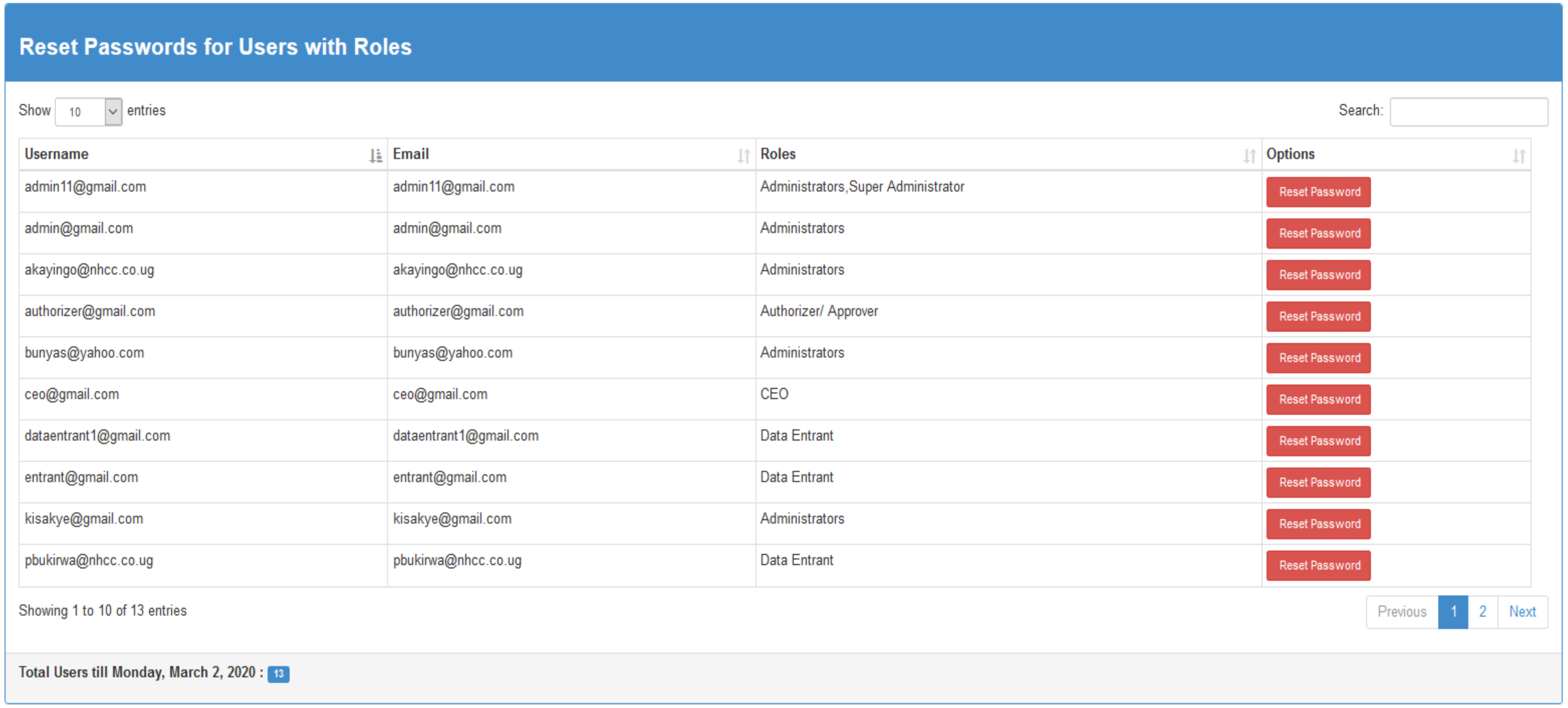
© 2020 - Titles Management System

Figure 21:Resetting Password Form

5.1.4 Reset User Password

This section allows change of password without knowing the old password. However, this can only be done by users with Super Administrator or Administrator rights.

As a Super Administrator or an Administrator, he/she has the right to change the password of other users of the system just in case one forgets his/her password.



Reset Passwords for Users with Roles

Show 10 entries

Search:

Username	Email	Roles	Options
admin11@gmail.com	admin11@gmail.com	Administrators,Super Administrator	Reset Password
admin@gmail.com	admin@gmail.com	Administrators	Reset Password
akayingo@nhcc.co.ug	akayingo@nhcc.co.ug	Administrators	Reset Password
authorizer@gmail.com	authorizer@gmail.com	Authorizer/ Approver	Reset Password
bunyas@yahoo.com	bunyas@yahoo.com	Administrators	Reset Password
ceo@gmail.com	ceo@gmail.com	CEO	Reset Password
dataentrant1@gmail.com	dataentrant1@gmail.com	Data Entrant	Reset Password
entrant@gmail.com	entrant@gmail.com	Data Entrant	Reset Password
kisakye@gmail.com	kisakye@gmail.com	Administrators	Reset Password
pbukirwa@nhcc.co.ug	pbukirwa@nhcc.co.ug	Data Entrant	Reset Password

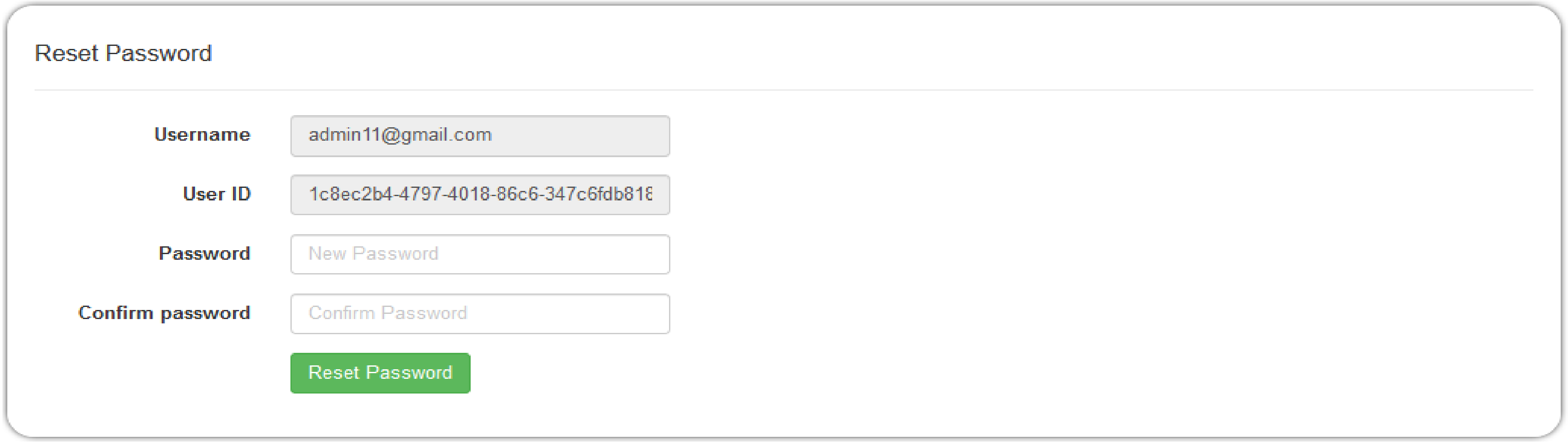
Showing 1 to 10 of 13 entries

Previous 1 2 Next

Total Users till Monday, March 2, 2020 : 13

Figure 22:Accounts to Reset User Password

To reset the *user password*, the Super Administrator/ Administrator clicks on the Reset Password button to open the form shown in ***Figure 23 below*** where action is taken.



Reset Password

Username	admin11@gmail.com
User ID	1c8ec2b4-4797-4018-86c6-347c6fdb818
Password	New Password
Confirm password	Confirm Password

Figure 23:Reset User Password

6 Data Validation

6.1 The Data Layer

The TMS System has a database that is used for storing data.

6.2 Implementing the Database Layer

The database layer was implemented using SQL Server database management system. It consists of the following items:

- Tables
- Views
- Stored Procedures
- Indexes
- Relationships
- Constraints
- Triggers

When implementing the database layer, we followed the following rules:

- Database should contain relationships.
- Each table should contain a primary index.
- Each table should be involved in at least one relationship.
- Each table should have more than one field.
- Each table should have some non-key columns.
- Each table should have a valid ANSI SQL name (no blanks).
- Each column should be a valid ANSI SQL name (no blanks).
- Columns should be defined as NOT NULL.

6.3 Stored Procedures

The database has stored procedures to perform the basic **CRUD** functions:

Create	Insert record
Read	Select record or records
Update	Update record
Delete	Delete record

In the case of the read operations, the database has two types of stored procedures. First, it contains stored procedures which return a list of records that match specified search criteria. Secondly, it has stored procedures which return the attributes of a specific record identified by a key.

6.4 Error Handling in Stored Procedures

No matter what type of code you are developing, error handling should be considered and designed into your solution upfront. Error handling should be able to provide two basic functions as follows:

- Logging errors should be logged
- Reporting errors should be reported to callers (i.e. programs or users)

All of the stored procedures in the TMS System use standard error handling. If an error occurs during execution of a stored procedure, the error will be logged to a central table. The **tblErrorLog** and **spErrorLogInsert** scripts are used to create an error log table and a stored procedure that inserts a record into it. If an error occurs in any of the other generated stored procedures, then the error will be recorded in the error log table.

All the generated stored procedures contain the following line which is used to allow the stored procedure to continue processing even when an error occurs.

```
SET XACT_ABORT OFF -- Allow procedure to continue after error
```

Next, a number of local variables are declared and initialized. These will be used to implement the error handling.

```
DECLARE @ProcName    varchar(30)       -- Name of this procedure
DECLARE @Error       integer           -- Local variable to capture the error code.
DECLARE @RowCnt      integer           -- Number of rows returned
DECLARE @ErrMsg      nvarchar(1000)    -- Error message to
return
*******************
Set local variables
*******************
SELECT @ProcName = object_name(@@procid)
```

```
    ,@Error          =    0
,@RowCnt  = 0
```

At the end of each statement, the @@Error value is copied and checked for a non-zero return. If it is non-zero, control flows to the error handler. You can also use the row count variable to check for data errors.

```
SELECT @Error  = @@Error
   , @RowCnt = @@RowCount
IF (@Error != 0)
BEGIN
  SELECT @ErrMsg = 'Error inserting record.'
  GOTO ENDERROR
END
```

The error handler is used to insert a record into the error log table. It also raises the error back to the calling procedure and returns a non-zero value in the @Return_Value parameter. This means that the error is logged and that calling programs (i.e. other stored procedures or .NET code) can intercept the errors and handle them appropriately.

```
************** Error Handling
**************
GOTO ENDOK
 ENDERROR:
BEGIN
  EXEC spErrorLogInsert 'DB Layer', @ProcName, @Error, @ErrMsg
  RAISERROR 25000 @ErrMsg
  RETURN 1
END
 ENDOK:
  RETURN 0
```

All errors in the data layer are successfully logged to the database as well as being reported to calling programs or users.

The system administrators can use the tblErrorLog table to check for run time errors. This should help reduce user anxiety as well as reduce maintenance costs for the application.

6.5 Create Record

The stored procedures to create records employs a standard INSERT statement that inserts all the values of a record. The values are passed into the stored procedure using standard input parameters.

There are three cases when a parameter might be passed out from an Insert stored procedure.

- The table contains an identity column.
- The table contains a GUID column (unique identifier).
- The table contains a timestamp. Each of these cases is taken in turn.

6.5.1 Identity Column

Identity values are automatically calculated by the database when a record is created. The system automatically generates code to pass the values of an identity column out from a stored procedure. Normally an identity column is used as a primary key. This means that calling procedures have direct access to the key of a newly created record as soon as it is created.

```
****************************** -- Populate the output parameters --
******************************

SELECT @Id = [Id]

 FROM [dbo].[FaultAssignee]

 WHERE [Id] = @@IDENTITY
```

6.5.2 Unique Identifiers (GUIDs)

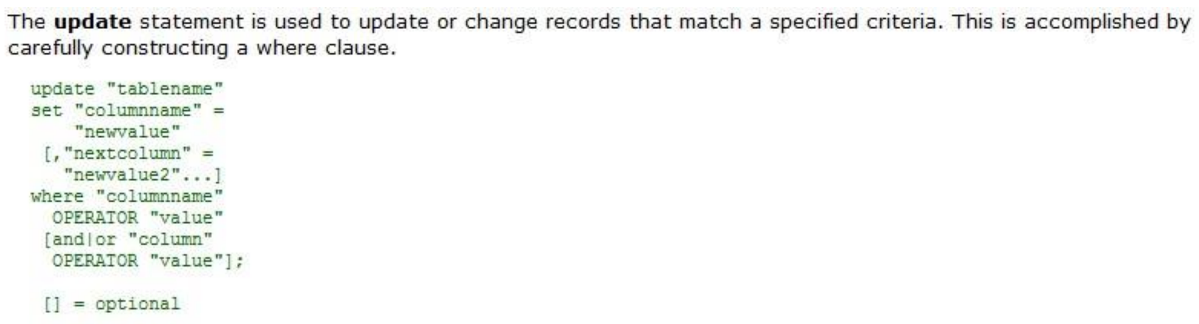
GUID columns are calculated using the ‘newid()’ function that comes with SQL-Server. The system has in-built code that automatically calls the newid() function to generate GUIDs. These are then passed back to calling procedures using output parameters.

```
-- ******************************** -- Call 'newid()' to get GUID value --
********************************

SELECT @Id = newid()
```

6.6 Update Record

The stored procedures to update records employs a standard UPDATE statement that updates all the values of a record identified by a key. The values are passed into the stored procedure using parameters.

The only instance where a parameter is passed out is in the case of timestamps. Timestamps are discussed later in the document in the section on Record Locking.

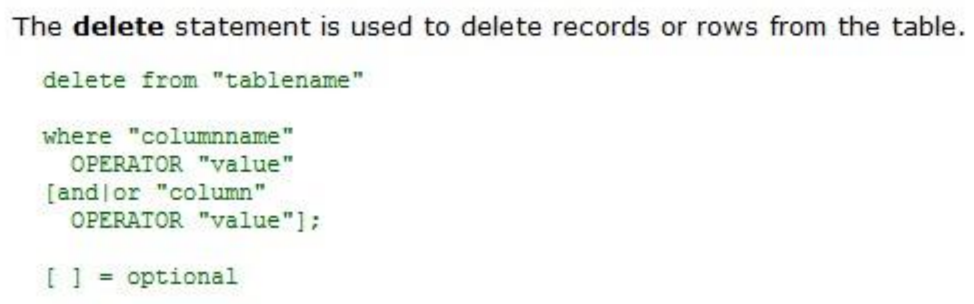


The **update** statement is used to update or change records that match a specified criteria. This is accomplished by carefully constructing a where clause.

```
update "tablename"
set "columnname" =
    "newvalue"
 [,"nextcolumn" =
   "newvalue2"...]
where "columnname"
  OPERATOR "value"
 [and|or "column"
  OPERATOR "value"];

 [] = optional
```

6.7 Delete Record

The stored procedures to delete records employs a standard DELETE statement that deletes a record identified by a key. The key is passed into the stored procedure using parameters.



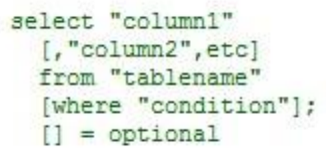
The **delete** statement is used to delete records or rows from the table.

```
delete from "tablename"

where "columnname"
  OPERATOR "value"
[and|or "column"
  OPERATOR "value"];

[ ] = optional
```

6.8 Select Record - GetByKey

The stored procedures to read a record employ the standard SELECT statement that reads all the attributes of a record identified by a key. The key is passed into the stored procedure using parameters. The attributes of the record are copied to output parameters.

The **select** statement is used to query the database and retrieve selected data that match the criteria that you specify. Here is the format of a simple select statement.



```
select "column1"
  [,"column2",etc]
  from "tablename"
  [where "condition"];
  [] = optional
```

6.8.1 Select Records – GetAll Search Parameters

The stored procedures to read records based on various search criteria functions as follows. Selecting Search Parameters - There are several reasons why a particular column might be set as a search field in the GetAll stored procedures. These are detailed as follows:

- **Primary Key** - the field forms part of the primary key.
- **Foreign Key** - the field is a foreign key.
- **Searchable** - the field has been set as searchable.

6.8.2 Primary Key Search Field

All columns that make up the primary key (even if there is only one column) are marked as searchable and will appear as a search parameter in the GetAll stored procedure.

```
SELECT *
 FROM [dbo].[Customer]
WHERE [Id] = 10
```

6.8.3 Foreign Key Search Field

Foreign keys will also appear as search parameters – this helps implement the master-detail functionality.

```
SELECT *
 FROM [dbo].[OrderItems]
WHERE [OrderId] = 10
```

6.8.4 Wildcard Searchable

In the cases of string searches, the system has been designed to allow wildcard searching. For example, get all the customers that begin with the letter 'C'.

```
SELECT *
 FROM [dbo].[Customer]  WHERE [Name] LIKE 'C%'
```

6.8.5 Range Searchable

In the case of dates and numeric types, the fields have been defined as range searchable which allows to specify a minimum and maximum range that records should be filtered by. For example, on the search panels there are the 'Start Date' and 'End Date'. These allow the user to restrict the search results to only records that were dealt with in those dates. Getting all the council decisions that have been updated with a given period is implemented using a range search on the decisions date.

6.8.6 Summary

In all, this gives us five different cases of search fields

- Column forms part of the primary key.
- Column is a foreign key.
- Column has been marked as searchable.
- Column has been marked as wildcard searchable.
- Column has been marked as range searchable.

In the case of range searchable, two parameters will be required which define the range.

6.8.7 Applying Search Parameters

As you know, stored procedures are pre-defined sections of SQL code that are compiled to run on a particular database. Because of this, they are fast, but the structure of the SQL must be known in advance. This means that we must pass all the possible search columns into the stored procedure.

In order to allow optional parameters to the method, we assign the value 'Null' to all stored procedure parameters that are not required.

This means that any parameters that we are not searching on will have the value 'Null'. All other parameters will have a non-Null value. We can use this to return all the required rows by testing if a parameter was set to NULL:

```
 SELECT [Customer].Id,
      [Customer]. Name,
      [Customer].AddressLine1,
etc.
 FROM Customer
 WHERE ([Customer].Id   = [pId]               OR [pId] Is NULL) AND
      ([Customer].Name = [pName]              OR [pName] Is NULL) AND
      ([Customer].AddressLine1   =   [pAddressLine1]   OR   [pAddressLine1]   Is
NULL) AND       etc.
      ([Customer].DateAdded  =  [pDateAdded]           OR  [pDateAdded]  Is
NULL);
```

When a parameter is not passed into the procedure, we set it to NULL in the stored procedure. Then the criteria ‘pName Is NULL’ will return true and all the records in the table will be returned for that part of the where clause. This gives us the desired result – a stored procedure that can be used to search on many columns using for various criteria.

6.9 Summary

The TMS System bases on a SQL Server Database with stored procedures which implement the basic CRUD operations. The stored procedures have pre-built error handling code that can be used to log and report errors. The stored procedures contain full support for all the intrinsic data types that come with a database (e.g. identity columns, unique identifiers and timestamps). The database has versatile stored procedures to search for records based on a wide range of search parameters.

7 The Data Access Layer (DAL)

The database access layer is responsible for allowing an application to communicate with a database.

The database access layer services an application's data needs. It retrieves information from a database and is also responsible for saving all changes back to the database. The TMS System's database access layer leverages the ADO.NET– a feature rich object model specifically designed for the purpose of delivering data access for applications.

7.1 Data Access Application Block for .NET

ADO.NET has been built for fine-grained control of data access behavior. Its object model supports a wide range of uses and scenarios which cater for any database access application that you might need. While low level control is synonymous with flexibility, it can come at a price. Complexity increases with flexibility.

To this end, Microsoft released the data access application block for .NET. The Data Access Application Block encapsulates performance and resource management best practices and can easily be used as a building block in the .NET application. This assists to reduce the amount of custom code needed to create, test, and maintain. The application block is a wrapper for ADO.NET that provides consistent and easy to use functions that provide flexibility as well as simplicity. The database access layer used in the TMS System leverages this application block.

7.2 Implementing the Data Access Layer

The database access layer in the TMS Systems was implemented as classes in the application. All the data access layer classes are prefixed with the letters **'dal'** (database access layer). The classes provide all the functions necessary to implement the CRUD functions

Function	Description	Class Method
Create	Insert record	Add
Read	Select record or records	GetByKey and GetAll
Update	Update record	Update
Delete	Delete record	Delete

In the case of the 'read 'function two class methods are used. GetByKey is used to return the details of one record. GetAll is used to return the details of many records. Each of these methods is described in detail together with other implementation aspects of the data access classes such as error handling, the data access application block and mapping column names to enums.

7.2.1 Add

The add method is used to add a record to a database table. In general, the Add method takes a list of input parameters each of which corresponds to a field in a table. These parameters are inserted into the database as a single record. The add method is implemented as a function. If the record is successfully added to the database, the function returns True; otherwise the function returns False.
In the case of tables which employ identity or autonumber fields, the function declares the parameters by reference so that the calculated fields can be returned to the calling procedure. Other fields that are calculated by the database are also returned in output parameters (e.g. timestamps and GUIDs (unique identifiers)).

```
 //
// C Sharp implementation of the Add method. Table includes an identity column
//
public bool Add(ref DataType Parameter1,
            DataType Parameter2,            ...
            DataType ParameterN)
{
  // Build embedded SQL to insert record or declare stored procedure
 // Parameters passed via embedded SQL or declared as stored procedure parameters
 // Execute the SQL or stored procedure

 // Map autonumber and identity fields to output parameters
 // Map timestamps and other database calculated fields to output parameters
 // Return boolean indicating if record was added.
 }
```

7.2.2 Update

The update method is used to update a record in a database table. In general, the Update method takes a list of input parameters each of which corresponds to a field in a table. These parameters are updated in the database as a single record. The update method is implemented as a function. If the record is successfully updated, the function returns True; otherwise the function returns False. In the case of tables which employ calculated fields (e.g. timestamps), the function declares the parameters by reference so that the calculated fields can be returned to the calling procedure.

```
 //
// C Sharp implementation of the Update method. Table includes a timestamp column
//
public   bool   Update(DataType   ParameterKey1,
DataType ParameterKeyN,
                DataType            Parameter1,
...
                ref DataType ParameterN)
{
 // Build embedded SQL to update record or declare stored procedure
  //  Parameters  passed  via  embedded  SQL  or  declared  as  stored  procedure
parameters
  // Execute the SQL or stored procedure


  // Map timestamps and other database calculated fields to output parameters
  // Return boolean indicating if record was updated.
}
```

7.2.3 Delete

The delete method is used to delete a record in a database table. In general, the Delete method takes a list of input parameters each of which corresponds to a primary key field in a table. These parameters are used to identify the record that should be deleted. The delete method is implemented as a function. If the record is successfully deleted, the function returns True; otherwise the function returns False.

```
//
// C Sharp implementation of the Delete method
//
public          bool          Update(DataType
ParameterKey1,                        DataType
ParameterKey2,                 ...
              DataType ParameterKeyN)
{
 // Build embedded SQL to delete record or declare stored procedure
 //  Parameters  passed  via  embedded  SQL  or  declared  as  stored  procedure
parameters
// Execute the SQL or stored procedure


// Return boolean indicating if record was deleted.
}
```

7.2.4 GetByKey

The GetByKey method is used to retrieve all the attributes of a specific record, identified by key. In general, the GetByKey method takes a list of input parameters each of which corresponds to a primary key field in a table. It also takes a list of parameters by reference which corresponds to the attributes of the record. These parameters are set by the method when the record's details are read from the database. The GetByKey method is implemented as a function. If the record is successfully read, the function returns True; otherwise the function returns False.

```
//
// C Sharp implementation of the GetByKey method
//
public    bool    GetByKey(DataType
ParameterKey1,              DataType
ParameterKey2,          ref DataType
Parameter1         ...          ref
DataType ParameterN)
{
  // Build embedded SQL to get record or declare stored procedure
  // Parameters passed via embedded SQL or declared as stored procedure
parameters
  // Execute the SQL or stored procedure
  // Map data to output parameters (i.e. ref parameters)
  // Return boolean indicating if record was deleted.
}
```

7.2.5 GetAll

The GetAll method is used to retrieve a list of records from a table that match specified search criteria. The search criteria are passed into the method as a series of parameters. All parameters that do not have a value are reset to NULL. This means that when retrieving the records, those parameters that are set to NULL have no effect on the 'where' of the SQL. Please see GetAll Search Parameters for more information on how search parameters are used to select records.

The data is retrieved as an ADO.NET data reader – a forward only read only structure that offers high performance retrieval of multiple records from a database.

```
//
// C Sharp implementation of the GetAll method
//
public bool GetAll(DataType ParameterSearch1,
DataType ParameterSearch2,                    ...
                    DataType ParameterSearchN)
{
  // Build embedded SQL to get records or declare stored procedure
  // Parameters passed via embedded SQL or declared as stored procedure
parameters
  // Reset input parameters that have no value to DbNull.Value
  // Execute the SQL or stored procedure
  // Return the records as a data reader
}
```

7.2.6 Data Access Application Block

All methods defined in the class leverage the data access application block for .NET. This ensures that the database access functions that we use are high performance and standard across all parts of the application.

7.2.7 Error Handling

All methods defined in the class also leverage the exception block for .NET. This offers centralized and standard error handling throughout the application. For more information on this block, please consult the chapter on error handling.

```
 // Execute the SQL and return the data
Try
  ...
Catch ex As Exception
  ExceptionManager.Publish(ex)
End Try
```

7.2.8 Record set Ordinals and Enums

When accessing the columns of a data set or data reader, you can use one of the two methods:

- Specify the column using the field name.
- Specify the column using the ordinal number which corresponds to the field of both these methods; pose problems for programmers and can lead to errors.

Hard coding field names can be difficult because programmers often spell the field names incorrectly.

In the following example, a programmer could easily use the string "CustomerOrderNumber" for the column name instead of the abbreviated "CustomerOrdNumber" that the DBA used when creating the table. This mistake will cause unnecessary work for the developer in that the bug will have to be fixed. What's more is the fact that the DBA may decide to remove all abbreviations and change the name of the database field to "CustomerOrderNumber". This will cause an error in the code.

```
'// Get the order number from the data reader dr.GetString("CustomerOrdNumber")
```

Another approach to accessing fields can be by using the fields' ordinal number. Hard coding field numbers in this way can easily cause mistakes and makes for code that offers poor readability. For example, consider the case where a programmer is trying to get the phone number and fax number for a contact. Unless the code is commented, it can be difficult to read. What's worse is the fact that if the DBA inserts a mobile phone field into the table between the other two, the ordinals should change to 5 and 7 respectively (6 will be the ordinal of the mobile phone). It may take a long time for this bug to be discovered.

```
'// Get the phone and fax numbers dr.GetString(5) dr.GetString(6)
```

This has been taken care of in the TMS System by having an enum that encodes the fields of a table using a series of constants. This offers several benefits as follows. Firstly, spelling mistakes are eliminated because the programmer can refer to fields and columns using enums which are checked at compile time. Also, enums have the added bonus of being available to programmers using VS.NET and its intellisense features. Finally, table changes can be easily managed by both programmers and DBAs. All they need to ensure is that the enums and table structures match – searching through thousands of lines of code for hard coded ordinals is not required.

```
'// Public ENUM used to enumerate
columns Public Enum tblAccountFields
fldAccountId = 0 fldCustomerId = 1
fldAccountName = 2 fldPhoneNumber
= 3 fldMobileNumber = 4
fldFaxNumber = 5
End Enum
'// Code is self-documenting. Enum offers intellisense
dr.GetString(tblAccountFields.fldCustomerId)

'// Code does not need to be changed even if a new field is added
dr.GetString(tblAccountFields.fldPhoneNumber)
dr.GetString(tblAccountFields.fldFaxNumber)
```

7.2.9 Summary

The following table serves to recap the main features of the database access layer classes in the TMS System.

Function	Description
Add	Insert a new record into the table
Update	Update record identified by key
Delete	Delete record identified by key
GetByKey	Get details of a record identified by key
GetAll	Get all records that match search criteria
Data Access	Use standard data access block from Microsoft
Error Handling	Use standard exception block from Microsoft
Column Ordinal Enum	Used to iterate table fields / columns

7.3 Summary

The TMS System has database access functions which implement the basic CRUD operations. All these database access functions come with standard error handling. Enums are used to encode the record field mappings – which helps reduce coding errors and helps makes database schema changes easier to manage. The database access layer leverages the data access application block and exception management application block from Microsoft. Using these standard architecture building blocks helps ensure that the application is easy to follow and maintain.

8 The Business Layer

8.1 Introduction

The business layer of an enterprise application contains business rules and helps model the business requirements and processes of the application. It encapsulates the business rules and business entities of the system.

8.2 Functionality of the Business Layer

The main function of the business layer is to provide the following functions and attributes.

8.2.1 Relational-to-Hierarchical Mapping

Relational-to-Hierarchical Mapping should allow us to view data (which is stored in a relational database) hierarchically. For example, viewing the orders that a customer made would be a hierarchical view of both the customer and order tables.

8.2.2 Data Access

The business layer forms the communication bridge between the user layer and the data layer. It should have the ability to retrieve data from the underlying database and also record changes to the data. This data access should be scalable and should not suffer from poor performance.

8.2.3 Business Rules

The business layer should also implement the business rules of the application. This include both data validation rules and process or business-driven rules (e.g. a Decision must have a unique Reference number).

8.2.4 Set of classes

Traditionally, the business layer provides a set of classes which represents the underlying business model. These classes are normally in the form of collections of objects. For example, printing all the decisions that had updates last month would normally involve iterating a collection of decision objects

8.2.5 Strongly Typed

The classes that the business model exposes are normally strongly typed. This reduces the chances for errors and helps ensure that programmers develop code using the correct data types.

8.2.6 Summary

To summarize, the business layer should provide the following functions and attributes.

Writing Business Objects normally involve three things:

- Relational-to-Hierarchical Mapping
- Data Access
- Business Rules

Traditional Business Objects normally expose

- Set of classes
- Classes are Strongly Typed

8.3 Data Representation

For .NET applications, there are many choices available to developers on how to implement this layer. For example, developers can implement this layer using any one of the following formats

- XML Documents
- UnTyped Datasets
- Typed Datasets
- Custom components

A brief introduction is given to explain what each of these formats is and why we decided to build the TMS System around one particular format.

8.3.1 XML Documents

The System.XML namespace provides two objects that can be used to represent data. These include:

- XmlDataDocument
- XmlDocument

Advantages

The main advantage of XML is that it is platform independent. This means that it can be copied from one system to another. As a consequence, XML is the foundation for web services. Other advantages of XML include the following:

Loose coupling	Callers must know about only the data that defines the business entity and the schema that provides metadata for the business entity. Callers do not need to know the inner workings of the underlying database.

Integration	Accepting XML will support callers implemented in various ways. For example, .NET applications, BizTalk Orchestration rules, and third-party business rules engines.
Collections of business entities	An XML string can contain data for multiple business entities.
Serialization	Strings natively support serialization and can cross platforms without loss of data.

Disadvantages

While XML does offer a great advantage for developers building applications across multiple platforms, it can be complex to use for a number of reasons. Firstly, most developers will not be familiar with the System.Xml object or programming model. Secondly, the data is not strongly typed, which can lead to errors and code that performs poorly. Finally, both the XmlDataDocument and the XmlDocument can be created from an ADO dataset using one simple method call. Other disadvantages include

Reparsing effort for XML strings	The XML string must be reparsed at the receiving end. Very large XML strings incur a performance overhead.
Inefficient use of memory	XML strings can be verbose, which can cause inefficient use of memory if you need to pass large amounts of data.
Supporting optimistic concurrency	To support optimistic concurrency, time stamp columns must be defined in the database and included as part of the XML data.

Summary

The fact that data sets can be easily converted into XML makes data sets seem a far better choice for our needs. The ability to convert data sets to / from XML means that data sets leave our classes open all the benefits of XML and none of the disadvantages that native XML has.

```
// DataSets can hold XML, so you can create XML objects as follows
return myDataSet.GetXml()
// XmlDataDocument can also be returned
return new XmlDataDocument(myDataSet)
```

8.3.2 Untyped DataSets

An ADO Dataset is an extremely flexible method of accessing and manipulating data. The data set contains a number of data tables. Each table contains the following collections

- DataRow
- DataColumn
- Constraint
- DataRelation

These collections can be used to help build the relational-to-hierarchical mapping that is required.

The following diagram illustrates the structure of an untyped dataset. Each dataset consists of a number of data tables and data relations. Each data table contains a number of constraints and relations. In addition, each data table contains a number of data rows and data columns.
One of the problems with a standard dataset is that it provides a "late bound", collection-based method of accessing and updating data. The type of data stored within the dataset is unknown at compile time and requires explicit casting and dynamic resolution at run-time. This is referred to as "weak typing". This causes problems as we will see in the following example.

```
// Create a DataAdapter for the table we are filling
SqlDataAdapter daCustomers = new SqlDataAdapter("SELECT * FROM CUSTOMER;", conn);
 // Initialise DataSet
DataSet dataSet = new DataSet();
 // Fill the DataSet with the DataAdapter daCustomers.Fill(dataSet, "Customers");
 // Show the Customer Name – need to use ToString() because we do not know
// what type of data it is
Console.WriteLine(dataSet.Tables["Customers"].Rows[0]["FirstName"].ToString())
;
Console.WriteLine(dataSet.Tables["Customers"].Rows[0]["LastName"].ToString());
// Changing the customer age with a string will not work because Age expects an integer
// If this runs, we will get a run time error dataSet.Tables["Customers
"].Rows[0]["Age"] = "56";
```

As we can see, the code above will cause a run-time error because the programmer has forgotten the type of the data – and there is no indication that he is using the wrong type. The solution to this problem is to use **typed data sets**.

8.3.3 Typed DataSets

The following diagram illustrates the structure of a typed dataset. It helps highlight the differences between a typed dataset and an untyped data set. The term "Strongly Typed" refers to the explicit naming and typing of methods and member variables of a class (in this case a dataset), as opposed to the generic methods of accessing and manipulating the class data, such as using collection classes. The type of the data is known at compile time and as such, allows incorrect data type assignments to be resolved at compile-time rather than run-time, saving processing and decreasing the chance of errors in your code. In addition, the "intellisense" features of the IDE (such as in Visual Studio.NET) can be effectively used to provide meaningful member variable descriptions at develop time to ease coding.

```
// Create a DataAdapter for each of the tables we're filling
SqlDataAdapter daCustomers = new SqlDataAdapter("SELECT * FROM CUSTOMER;",
conn);
// Initialise DataSet
CustomerTDS dataSet = new CustomerTDS();
 // Fill the DataSet with the DataAdapter daCustomers.Fill(dataSet, "Customers");
 // Show the Customer Name – no need for the ToString() method
Console.WriteLine(dataSet.Customer[0].FirstName);
Console.WriteLine(dataSet.Customer[0].LastName);
 // This will not compile because Age expects an integer dataSet.Customer[0].Age = "56";
```

In this example, the programmer gets a compilation error and can easily see where he is going wrong. Even better is the fact that VS.NET will provide intellisense features for the programmer to help him enter the field names correctly.

Advantages

Native functionality	DataSets provide built-in functionality to handle optimistic concurrency (along with data adapters) and support for complex data structures. Furthermore, typed DataSets provide support for data validation.
Collections of business entities	DataSets are designed to handle sets and complex relationships, so you do not need to write custom code to implement this functionality.
Maintenance	Schema changes do not affect the method signatures. However, if you are using typed DataSets and the assembly has a strong name, the Data Access Logic Component class must be recompiled against the new version, must use a publisher policy inside the global assembly cache, or must define a <bindingRedirect> element in its configuration file.

Serialization	A DataSet supports XML serialization natively and can be serialized across tiers and different platforms.
XML Support	A data set can be converted to / from XML. This means that
	a data set can be transformed to have all the advantages of native XML.
Type Data	Typed data sets (as the name implies) implement strongly typed business objects which allow incorrect data type assignments to be resolved at compile-time rather than runtime. This leads to fewer bugs and shorter development cycles.

Disadvantages

Performance	Instantiating and marshalling DataSets incur a runtime overhead. For accessing and reading large amounts of data, a data reader will perform a lot better then a dataset.
Representation of a single business entity	DataSets are designed to handle sets of data. If your application works mainly with instance data, custom components are a better approach as you will not incur the performance overhead.
Maintenance	It can be difficult in Visual Studio .NET to successfully regenerate a typed dataset when a table structure changes.
Ease of Use	Business object models developed using data sets requires users (i.e. user layer developers) to have a good knowledge of ADO.NET– which could be considered a complex object model.

Summary

From the discussion above, it is obvious that typed datasets have advantages over both untyped datasets and xml as a means for implementing a business layer.

From here the argument becomes less concrete and really depends on your preferences. In our opinion, custom components can offer all the advantages of typed data sets, and more. They are strongly typed and can be easy to use for developers working on user tiers. If implemented correctly, they offer greater performance than that offered by typed datasets (especially when reading large amounts of data by leveraging the data reader).

After considerable research into what users preferred, we decided to build the first system on custom components.

8.4 Custom Components

The main advantage of custom classes is that they are strongly typed and easy to use. This allows you to work with objects that more closely match the underlying data while not having to deal with the ADO.NET and XML programming models. This is particularly effective in large systems when an architect designs a data access layer made up of custom classes and then distributes the assembly to other developers within the organization. Of course, there's a downside to using custom classes:

They require more coding, since you'll need to create the object-relational mapping layer that the .NET Framework lacks.

Note:

You may know custom components by some other name. Other names that come to mind include:

'business objects', 'business entities' and 'business entity components'.

A detailed description of what we mean by 'custom component' now follows.

Overview

Custom components are normally built using stateful object classes and collections which can be used to group those objects. For example, the following code implements a customer object that has support for two properties – the customer name and customer id properties.

```
 <Serializable()> _
Public              Class
Customer
  // Constructor
  Public Sub New(ByVal Id As Integer, ByVal Name As String)
    Me.Id = Id
    Me.Name = Name
  End Sub
  // Property member variables
  Private _Id As Integer
  Private _Name As String
  // Id Property
  Public Property Id() As Integer
    Get
      Return _Id
    End Get
    Set(ByVal Value As Integer)
     _Id = Value
    End Set
  End Property
  // Name Property
  Public Property Name() As String
    Get
      Return _Name
    End Get
    Set(ByVal Value As String)
      _Name = Value
   End Set
  End Property
End Class
```

This type of object is generally available as both a standalone object and as an item in a collection object. Implementing the collection for the customer object is simply a matter of building a class that inherits the System.Collections. CollectionBase class.

```
// Implement custom collection
<Serializable()> _
Public Class CustomerCollection
  Inherits System.Collections.CollectionBase
```

Advantages

Maintenance	Schema changes may not affect the Data Access Logic Component method signatures. However, the same issues arise as with typed DataSets if the Business Entity Component is held in a strong-named assembly.
Collections of business entities	An array or a collection of custom business entity components can be passed to and from the methods.
Easy to Use	Custom components are easy to use and do not require developers to learn the complex programming models that you would associate with XML or ADO.NET.
Serialization	Custom components can be coded to support XML serialization. Therefore, they can be serialized across tiers.
XML Support	Custom components can be converted to / from XML by adding appropriate code. This means that a custom component can be transformed to have all the advantages of native XML.
Typed Data	Custom components implement strongly typed business objects which allow incorrect data type assignments to be resolved at compile-time rather than run-time. This leads to fewer bugs and shorter development cycles.
Performance	Custom components may offer better performance then typed data sets. This can become especially apparent when reading large amounts of data.
Representation of a single business entity	DataSets are designed to handle sets of data. If your application works mainly with instance data, custom components are a better approach as you will not incur the performance overhead.

Disadvantages

Supporting optimistic concurrency	To support optimistic concurrency easily, time stamp columns must be defined in the database and included as part of the instance data. This is not really an issue for VBeXpress.NET in that the generated code will do this for you automatically if you mark the appropriate fields as ‘timestamp’ fields. For more information on this, consult the chapter on record locking.
Limited integration	When using custom business entity components as inputs to the Data Access Logic Component, the caller must know the type of business entity; this can limit integration for callers that are not using .NET. However, this issue does not necessarily limit integration if the caller uses custom business entity components as output from the Data Access Logic Component. For example, a Web method can return the custom Business Entity Component that was returned from a Data Access Logic Component, and the Business Entity Component will be serialized to XML automatically using XML serialization.

8.5 TMS System Custom Components

The TMS System custom components were built using two base classes

- CollectionClass
- StatefulClass

These classes offer all the benefits associated with implementing custom components. They have been designed and built from the ground up. Both base classes have been built to maximize the benefits of building a business layer using custom components. They leverage some of the most useful aspects and classes that are available from the .NET framework.

- CollectionBase
- IListSource
- IBindingList
- Serializable

The following diagram illustrates the structure of the custom components. Comparing this model to the typed data set model shows how much simpler and easier this programming model is compared to the ASO.NET programming model.

8.6 The Collection Class

8.6.1 Overview

The VBeXpressCollectionClass is the base class that all collection classes inherit from. The base class has been implemented to leverage some of the more advanced features and classes of the .NET framework.

- CollectionBase
- IListSource
- IBindingList
- Serializable

8.6.2 CollectionBase

This represents a collection or list of objects. As such, it implements the CollectionBase class that comes with the Microsoft .NET framework. This class gives us the ability to build an indexed collection or list of objects. The collection class also comes with methods and properties that allow us to add items to the collection, remove items from the collection, find items and iterate through the collection.

8.6.3 IListSource

The CollectionClasses implement IListSource. This interface provides functionality to an object to return a list that can be bound to a data source. By implementing this interface, CollectionClasses can be bound to controls – including the Microsoft Winform DataGrid, the Microsoft Webform DataGrid and other third-party grids.

8.6.4 IBindingList

The CollectionClasses also implement IBindingList. This interface provides the features required to support both complex and simple scenarios when binding to a data source.

Most vendors recommend that in order to take advantage of the advanced features of their grids, collections should implement IBindingList. This means that the collection classes can take advantage of the advanced features of 3rd party grid controls such as inline editing, sorting, data grouping and data filtering.

8.6.5 Serializable

Finally, the CollectionClass leverages the serializable attribute. This helps ensure that the classes can be used across platforms. For example, only serializable objects can be saved in the 'view state' of ASP.NET web forms and web controls.

```
 // Hold the collection in view state so that it retains state
// between server calls
Public Property Customers As Object
   Get
      Return Me.ViewState("Decisions")
   End Get
   Set(ByVal Value As Object)
      Me.ViewState("Decisions") = Value
      Me.RefreshDataSource()
   End Set
End Property
```

8.6.6 Constructor (New)

The collection class constructor is used to fill the collection class with data directly from the database. The data is passed back from the database access layer using a data reader. The data reader is a read only forward only record set that offers a high-performance way to read a set of records from a database.

The constructors are built to allow you to fill the collection based on the search parameters exposed in the GetAll stored procedure. The constructor calls the appropriate database access layer GetAll procedure to return all the records in a data reader. Before calling the GetAll procedure, all blank search parameters are mapped to nulls so that the versatility of the GetAll stored procedure is used.

Once the GetAll procedure returns, the constructor iterates the data reader and loads the collection with data. Code is generated to handle null values from the database – which can often be the cause of programming errors during application development. Finally, the data reader is closed which closes the database connection and frees up resources.

8.6.7 Add

Adds an item to the collection. It may be useful to also include a procedure which adds an item to the collection at a specified index. This could be used to add a new item to the start of the collection, rather than at the end (which is the default).

8.6.8 Remove

Removes the item at the specified index from the collection.

```
// Remove the last item

Decisions.Remove(Decision. Count – 1)
```

8.6.9 Item

Returns the item at the specified index from the collection.

8.6.10 Iterating Collections (For Each .. Next)

One great feature about the Collections is that they implement the IEnumerable interface which support the use of for each {…} construct that can be used to loop through a collection.

8.6.11 Count

As expected, you can get the number of items in the collection with one call.

```
'// Number of items in the collection

MessageBox.Show(Decisions.Count.ToString());
```

8.6.12 IComparer

Since all the source code is open to you, you can extend the collections to implement other functionality such as the IComparer interface which can be useful for sorting. Although the collection classes have pre-built generic sorting defined, you may decide to build classes with optimized sorting capabilities.

8.6.13 Summary

The following table serves to remind you of the main features of the CollectionClass.

Function Name	Description
CollectionBase	Provides the base type for a strongly typed collection.
IListSource	Provides functionality to an object to return a list that can be bound to a data source.
IBindingList	Provides the features required to support both complex and simple scenarios when binding to a data source.

Serializable	Indicates that a class can be serialized.
Miscellaneous	Count, Add, Remove, Iterations (For Each .. Next)

8.7 TMS System Stateful Classes

8.7.1 Overview

The VBeXpressStatefulClass is the base class that all stateful classes inherit from. The base class has been implemented to leverage some of the more advanced features and classes of the .NET framework.

- Serializable
- IComparable
- IEditableObject
- IDataErrorInfo
- ICloneable

The base class also contains special properties which help control validation of data, when data is saved and how data is saved. It also contains functions for adding, updating, deleting and loading objects from a database by calling appropriate class methods from the database access layer.

8.7.2 Serializable

The StatefulClass leverages the serializable attribute. This helps ensure that the classes can be used across platforms. For example, only serializable objects can be saved in the 'view state' of ASP.NET web forms and web controls.

8.7.3 IEditableObject

This interface helps provide additional binding capabilities when binding collections of objects as data sources to Winform controls (e.g. grids). The interface helps allow you to bind CollectionClasses to grids by offering the ability to commit or rollback changes to objects as they happen using bound controls. This finer level of control helps ensure that you can track changes as they occur and validate them before they are committed to storage.

8.7.4 IDataErrorInfo

This interface helps provide error information that can be used by bound controls for helping users enter valid data. By implementing this interface, data validation and the accompanying custom-built information messages can be streamlined back to users using the built-in functionality offered by bound controls (e.g. the Microsoft Winform Data Grid).

8.7.5 ICloneable

This interface ensures that objects support cloning – the ability to create new instances of an object with the same value(s) as an existing instance. This effectively gives you a means to manipulate copies of objects without changing the underlying object itself. This could be useful, for example, if you wanted to build a commit / rollback feature for an object.

8.7.6 Properties

The Stateful classes offer a set of properties which match the fields in an underlying table. This gives you the ability to develop feature rich objects within built validation and error handling. It also gives you an object model that is easy and intuitive to program against.

8.7.7 Events

Each property of the StatefulClass implements a changed event which can be trapped by calling procedures. In the case of bound controls, these events are used to tell the control to refresh itself (if a property is changed outside of the control). Implementing these events further enhances the binding capabilities of the object model and provide for enhanced feedback for users of your application.

8.7.8 Load

This method loads a single object with data from the database. This is especially useful if you want to load one object, without loading an entire collection. The load method takes as input the key of the object to load. It loads the data from the database by calling the database access layer. The data is placed into the appropriate properties. The function returns a boolean indicating if the load was successful or not.

```
*************************************************************
************** '*
'* Name:       Load
'*
'* Description: Loads the record from the database into the class object.
'*
'* Returns:     Boolean which indicates if the record was loaded sucessfully
'*
'* Remarks:     Populates the properties of the class object with data from  '*
the database.
'*
```

```
'*******************************************************************
************** Public Function Load(ByVal Id As Integer) As Boolean


    'Declare database access layer object

    'Declare local variables


    'Get the data and populate the properties

    If DAL.GetByKey(.. , .. ) Then

        Return True

    Else

        Return False

    End If


End Function
```

8.7.9 Save

This method saves an object to a database. It uses the **NewRecord** property to determine whether the object should be saved as a new record (i.e. Insert) or as an updated record (i.e. Update). The save method calls the appropriate database access layer class method which in turn saves the record. Fields which are calculated by the database (e.g. identifiers, auto counters, GUIDs and timestamps) are sent back as output parameters and placed into the appropriate properties after the record has been saved. The function returns a boolean indicating if the save was successful or not.

8.7.10 Delete

This method deletes an object from the database. It calls the appropriate database access layer component and returns a boolean indicating if the operation was successful or not.

8.7.11 Key

The StatefulClass contains a property called key which is used to uniquely identify it. In the case of tables which contain one field for a primary key, the key will correspond to that field. In the case of tables which contain composite keys (i.e. keys that span two or more fields), the key is made up of a string delimited by a constant called KEY_SEPERATOR which is defined as "ABCD-1232-

24234242". This string is used so that parsing of keys into constitute fields is assured. The key is especially useful for binding collections to controls. Many controls come with support for displaying value / name pairs.

8.7.12 ToString

Returns a string representation of the object.

8.7.13 Summary

The following table serves to remind you of the main features of the StatefulClass.

Function Name	Description
Serializable	Indicates that a class can be serialized.
IEditableObject	Provides functionality to commit or rollback changes to an object that is used as a data source.
IDataErrorInfo	Provides the functionality to offer custom error information that a user interface can bind to.
ICloneable	Supports cloning, which creates a new instance of a class with the same value as an existing instance.
Load	Loads the object with data from the database.
Save	Saves the object to the database.
Delete	Deletes the object from the database.
Properties	Properties of the object correspond to fields in the database. Validation of data occurs in the set events.
Events	Property changed events used to help binding functionality.
Key	Returns the key of the object.
ToString	Returns the string representation of the object.

END