Database Information:

The table below summarises the basic database information. Detailed information can be found in the Script folder, database files, or by running the backup files.

Database Name:	DigitalXJackySql
Primary Database File and filegroup:	DigitalXJackySql.mdf (Filegroup: Primary)
Database Log File:	DigitalXJackySql_log.ldf
Secondary Database File and filegroups	AddressData1.ndf (Filegroup: FGAddress1), Address Data2.ndf (Filegroup: FGAddress1), CummodityData1.ndf (Filegroup: FGCommodity1), CommodityData2.ndf (Filegroup: FGCommodity1), CustomerData1.ndf (Filegroup: FGCustomer1), CustomerData2.ndf (Filegroup: FGCustomer1), OrderData1.ndf (Filegroup: FGOrder1), OrderData2.ndf (Filegroup: FGOrder1), Mist1Data.ndf (Filegroup: FGMist1)
Schemas and Tables:	Address.AddressDetails, Address.CustomerAddress, Address.OrderAddress, Category.MainCategory, Category.SubCategory, Customer.CreditCard, Customer.CustomerDetails, Order.GSTType, Order.OrderDetail, Order.OrderHeader, Order.OrderStatus, Product.ProductDetails, Src.ProductList, Src.TempProductList, Stg.ProductListDeletes, Stg.ProductListInserts, Stg.ProductListUpdates
Views	dbo. vOrdersOnBackorder, dbo. vProductsOnBackorder
Stored Procedures	dbo. usp_EncryptByAsymKeyCustomerPassword, dbo. usp_DecryptByAsymKeyCustomerPassword, dbo. usp_EncryptByAsymKeyCreditCard, dbo. usp_DecryptByAsymKeyCreditCard
Table-valued Functions	dbo. ufn_GetOrderHistoy

Notes:

For managerial, recovery and performance reasons, a number of secondary data files were created under a number of filegroups. Also, ideally, the data files and log file should be stored

separately at the physical disk level for both performance and recovery reasons. However, for test purpose only, this project put the data and log files in the same physical location.

Table design and relationships:

Please refer to 'DatabaseDiagram.jpg', database files, backup files and scripts enclosed for more information.

The RRP of the Product

The RRP column in Product.ProductDetails table is automatically calculated by using Computed Column technique:

```
USE [DigitalXJackySql]
GO

/****** Object:  Table [Product].[ProductDetails]    Script Date: 24/03/2016 00:20:49
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [Product].[ProductDetails](
	[ProductID] [int] IDENTITY(1,1) NOT NULL,
	[SubCategoryID] [int] NOT NULL,
	[ProductName] [nvarchar](100) NOT NULL,
	[ProductDescription] [nvarchar](max) NULL,
	[Cost] [money] NOT NULL,
	[RRP]  AS ([Cost]*(1.2)) PERSISTED,
	[UnitsInStock] [int] NULL,
	[IsDeleted] [tinyint] NOT NULL CONSTRAINT [DF_ProductDetails_IsAvailable]
DEFAULT ((0)),
 CONSTRAINT [PK_ProductDetails] PRIMARY KEY CLUSTERED
(
	[ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [FGCommodity1]
) ON [FGCommodity1] TEXTIMAGE_ON [FGCommodity1]

GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [Product].[ProductDetails]  WITH CHECK ADD  CONSTRAINT
[FK_ProductDetails_SubCategory] FOREIGN KEY([SubCategoryID])
REFERENCES [Category].[SubCategory] ([SubCategoryID])
GO
```

```
ALTER TABLE [Product].[ProductDetails] CHECK CONSTRAINT
[FK_ProductDetails_SubCategory]
GO
```

The Uniqueness of Customer Email:

A unique key constrain was created for the column 'Email' in the Customer.CustomerDetails table:

```
USE [DigitalXJackySql]
GO

/****** Object:  Table [Customer].[CustomerDetails]    Script Date: 24/03/2016
00:20:16 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [Customer].[CustomerDetails](
	[CustomerID] [int] IDENTITY(1,1) NOT NULL,
	[Email] [nvarchar](100) NOT NULL,
	[Password] [nvarchar](max) NOT NULL,
	[FirstName] [nvarchar](50) NOT NULL,
	[LastName] [nvarchar](50) NOT NULL,
	[DOB] [date] NOT NULL,
	[Telephone] [nvarchar](40) NULL,
	[Mobile] [nvarchar](40) NOT NULL,
 CONSTRAINT [PK_CustomerDetails] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [FGCustomer1],
 CONSTRAINT [IX_CustomerDetails] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [FGCustomer1]
) ON [FGCustomer1] TEXTIMAGE_ON [PRIMARY]

GO
```

Stored Procedure and Security Implementation:

Four stored procedures were created in order to insert and retrieve customer and credit information. Sensitive information including password, credit card number and cvv number are protected by SQL Server Encryption system. Specifically, a database Master Key was created first, and then an Asymmetric key was created with algorithm (RSA_2048). Customer password, credit card number and cvv number are protected (encrypted) by the asymmetric key. The same asymmetric key is used to decrypt aforementioned information. Detailed implementation of security can be found in the scripts in script folder enclosed.

Table-valued Function:

A table-valued user defined function called dbo. ufn_GetOrderHistoy was created in order to support the order history page of the website. The script for this function can be found in the script folder.

Incremental update

An SSIS project was create by Visual Studio for implementing initialising and incremental updating product information. Five Project Connection Managers were created for managing connections with various data sources. Five Project Parameters were created for specifying Expressions in the Properties panels of the connection managers for deployment purpose. Please refer to the SSIS folder enclosed for more detailed information of the SSIS project

The SSIS project was deployed to SQL Server's Integration Services Catalogs under folder *DigitalXProductIncrementalUpdates*. Five variables were created under an Environment called *Production*. The variables are used for mapping corresponding Project Parameters in the SSIS project. These variables are:

Variables:

Name	Type	Description	Value
DBName	String	Database Name	DigitalXJackySql
ServerName	String	Server Name	localhost
Supplier1Path	String	Flat File 1 location	C:\Data\SupplierOne.csv
Supplier2Path	String	Flat File 2 location	C:\Data\SupplierTwo.csv
Supplier3Path	String	Flat File 3 location	C:\Data\SupplierThree.csv

A SQL Server Agent job was then created to perform the Incremental Updates package that is under the SQL Server's SSIS Catalog. The job was scheduled to perform the incremental updating every week on Sunday at 22:01:00 and it has started since 24/03/2016. The scrip of this job can be found in the scrip folder enclosed.

Views

Two Views were created for this database. One view called dbo. vOrdersOnBackorder, which facilitates viewing all orders on backorder. The other view called dbo. vProductsOnBackorder, which is used for viewing all products on backorder. The scripts of these views can be found in script folder enclosed.

Index:

SQL Server automatically created unique and clustered index for all columns that are defined as Primary key column in this database. Since every table in this database has a primary key, hence every table has a clustered index in this database. Some scholars recommended to create clustered index for every table in order to facilitate the performance of an ordinary database. From this point of view, this database thereby meets this requirement.

In addition, SQL Server automatically created unique nonclustered index for the Email column in Customer.CustomerDetails table and the InvoiceNumber column in Order.OrderHeader table since unique constrains were defined on these columns.

Moreover, most foreign key columns in this database are expected to frequently perform JOIN operation, therefore, nonclustered index were created for *most* of the foreign key columns in order to facilitate query performance.

Furthermore, nonclustered index were also created for the columns of FirstName and LastName in Customer.CustomerDetails table and the column of Status in Order.OrderStatus table because these columns are expected to frequently involve in WHERE clause that returns exact match.

All manually created nonclustered index were scripted and can be found in the script folder enclosed.

Backup Implementation

According to the current circumstance of DigitalX including budget and system utilisation information, the following database backup jobs were scheduled and managed by SQL Server Agent. (It is very important to store the database backup files in a reliable and isolated physical disk other than the physical disk that the server and database are currently on.)

1. Full database backup: every week on Monday at 03:00:00.
2. Differential database backup: every day at 18:00:00.
3. Transaction Log backup: every day every one hour between 00:00:00 and 23:59:59.

The scrips of all above jobs can be found in script folder enclosed.

Moreover, given DigitalX expects to have significant increase in customer base as well as the daily order transaction will reach 2500 over the next six months and double again with 12 months, we may expect that both the database utilisation and size will be increased significantly in the future. Therefore, the above database backup strategy may become inefficient one day. In such circumstance, DigitalX may consider additional backup configurations in order to facilitate the performance of the backup. For instance, the database file can be compressed by SQL Server in order to reduce size.

In addition, Partial Backup may also be a viable option if the database includes some read-only filegroups, so that only the data in the primary filegroup and every read/write filefroup are targeted by the routine backup jobs. In terms of DigitalX, potential candidate data files that can be placed into the read-only filegroups includes, for example, the historical order information in the OrderData secondary data files. The historical order information (i.e. inactive data) can be placed in read-only filegroups and only need to be badked up once. Subsequent backups only target those read/write filegroups and primary filegroup so that the full and differential backup time taken may be reduced significantly. Another advantage of partial backup strategy is that it enables a piecemeal restore. In a piecemeal restore, the read/write filegroups can be recovered and made them available to users before the recovery of read-only filegroups is complete.

High Availability Consideration

Maintaining high availability of a database is vital for a 24/7 online shopping website. According to the current circumstance of DigitalX, following options for maintaining high availability are discussed here with related recommendations for DigitalX:

Windows Azure

It may look like Windows Azure should not be listed as a high availability solution. However, if DigitalX currently does not want to invest in additional software and hardware infrastructure, does not want to upgrading existing equipment, and does not have enough human resource to manage physical servers, then efficiently utilising Windows Azure Cloud services could be a possible option temporarily – at least in the beginning stage of the new online store database system. Certainly, DigitalX may not want to always rely on third party services to maintain database availability and it should recognise that Windows Azure is not guaranteed for 100% availability. Eventually, DigitalX may want to have full control over its data, servers and databases for performance, security and reliability reasons – all of these require investment in reliable and up-to-date software and hardware infrastructure.

Database Mirroring

While this is a well-known and conventional high availability solution, the Database Mirroring will be removed in future version of SQL Server, therefore, it is not recommended to use it for maintaining high availability for DigitalX (https://msdn.microsoft.com/en-us/library/ms190202.aspx).

Log Shipping

Another conventional method can be recommended to DigitalX is Log Shipping. It operates at the database level. One or more secondary databases (warm standby databases) are utilised for maintaining high availability of the primary database (production database). By utilising SQL Server Log Shipping, the transaction log backups of DigitalX's primary database on the

primary server instance are automatically send to one or more secondary databases on separate secondary server instances.

The Log Shipping solution therefore requires DigitalX having minimum one secondary database on a separate secondary server. If the budget permits, DigitalX may also consider to set up another (or even more) secondary database on another (or even more) separate secondary server instance for maintaining high availability in the event of failure on primary database and server.

A significant drawback of Log Shipping is that it does not automatically fail over from the primary server to the secondary server. If the primary database becomes unavailable, the secondary databases must be brought online manually.

AlwaysOn Availability Groups

Similar to Log Shipping, the AlwaysOn Availability Groups solution operates at the database level. Yet, unlike the Log Shipping, AlwaysOne Availability Groups solution supports automatic failover.

AlwaysOn Availability Groups supports a set of read-write primary databases and one to eight sets of corresponding secondary databases. It supports flexible failover environment for a set of availability databases that fail over together. There are three forms of failover exist. The first two (automatic failover and planned manual failover) will not result in data loss, therefore are recommended for ordinary usage. The third form (forced manual failover) will result in data loss and is typically used for disaster recovery only.

If DigitalX's budget permits and would like to pursue this high availability solution, then the SQL Server instances must reside on Windows Server Failover Clustering (WSFC) nodes.

Furthermore, there is a more advanced high availability solution called AlwaysOn Failover Cluster Instances, which provides high availability through redundancy at the server-instance level. This option is so far not necessary according to DigitalX's current and near future circumstance. However, should DisitalX's business requirement in the future needs such high availability solution, it can be implemented.