1. **Database creation.**
To create the database "DigitalIX" one runs "CreatingTables.sql" script.

The databse DigitalIX consists of three schemas (“prod”,”cust” and “order”) and the following tables:

1. "Product"
2. "Subcategory"
3. "Category"
4. "Order"
5. "Statuses" (with 5 rows:PROCESSING, BACKORDER, SHIPPING, DELIVERED, CANCELED)
6. "Customers"
7. “ShoppingCart”
8. "Creditcards"

Each table contains a foreign key and a primary key (which is also its clustered index). A foreign key of one table points to a primary key of another table.

There is one more table called "Temp". The table "Temp" stores temporary data from the CSV files and it is used in the SSIS packages to extract data from CSV files and populate “prod” tables in the database.

N.B.:
Table "Customers" has a column "EncryptedPw", which contains encrypted customer passwords. Table "Customers" contains also a column "Email" with "unique" constraint. This ensures that emails are all unique. Table "Creditcards" has a column "EncryptedCardNumber", which contains encrypted credit card used to place an order.

2. **SSIS packages.**
To populate tables associated with products ("Product","Subcategory","Category") one runs SSIS packages in the “SSIS” folder. Package "Step1_ExtractInitialProducts" extracts data from “prouctList_Initial.csv” file into "Temp" table. Package "Step2_LoadInitialProducts" loads data from the "Temp" table into "Product","Subcategory","Category" respectivelly. After that "Temp" is truncated.

There are other two packages used to perform incremental update of the product list. "Step1_ExtractNewProducts" extracts data from “productList_Latest.csv” file into "Temp" table. "Step2_LoadNewProducts" performs incremental update of "Product","Subcategory","Category". After that "Temp" is truncated.

3. **Stored procedures.**
One runs “StoredProcedure.sql” script to create several stored procedures and two asymmetric keys used for encryption and decryption of passwords and credit cards numbers.

Three stored procedures are of great importance:
- “dbo.InsertNewCustomer” is used to insert customers into the database;
- “dbo.authenticateCustomer” is used to authenticate a customer;
- “dbo.PlaceOrder” is used to place orders into the database.

4. **Adding customers**.
The stored procedure "dbo.InsertNewCustomer" is used to add customers. It first checks if an email address of a customer already exists in the "Cusmoters" table. If it exists the procedure raises an error prompting to use another email. If it does not exist the procedure inserts new customer into "Customers" table and encrypt password . “Encryptby...” method is used to perform the task.

One can use stored procedure called "dbo.authenticateCustomer" to check if a customer with provided email and password already exists in the "Customer" table. "Decryptby..." method is used to retrieve the encrypted password.

5. **Shopping cart**.
Once a customer has been added, table "ShoppingCart" can be populated. It contains customer ID, product ID and quantity. This information is used for placing an order.

6. **Placing order**.
The stored procedure “dbo.PlaceOrder” is used to place an order. In particular, it requires a customer ID and credit card number as its arguments.

It first checks if a credit card provided already exists in the “creditcards” table. If it exists, the procedure retrieves its ID. If it does not exist, the procedure inserts an encrypted credit card number into “creditcard” table and retrieves its newly created ID.

The procedure then retrieves one by one all product IDs and their quantities in the “ShoppingCart” table associated with a customer with customer ID provided. The procedure inserts product IDs, their quantities, Customer ID , credit card ID, “getdate()” as date ordered and other information into the “order” table.

After that, the procedure updates product quantities in the “product” table.

7. **Order history and views**.
The script “Func_Orderhistory.sql” creates a table-valued user defined function to support the order history data of the databse. The script “CreatingViews.sql” creates two views allowing DigitalIX employees to view all orders amd the products on backorder.

8. **Backup strategy**.
Full backup is done every Sunday at 3 a.m. Differential backups are done every other day of the week at 3 a.m. Log backups are done every hour. The corresponding three scripts “..._Job.sql” are provided.

9. **High availability plan**.

There are several availability solutions. I have chosen database mirroring, since it supports almost instantaneous failover and works with databases that use the full recovery model (the database "DigitalIX" uses full recovery model).

I need to create a copy of the database "DigitalIX" on a different server instance of SQL Server Database Engine. That server should reside on a computer in a different location than the computer used to store the database "DigitalIX".

Since "DigitalIX" has availability target of 99.7%, I chose high-safety mode supporting synchronous operations between the principal and the mirror databases. I also need a witness server instance which verifies whether the principal server is up and functioning.

I make sure that logins exist on the mirror server for all the database users, so that in the case of failover they can access data in the mirror database.

I perform the following steps to establish a mirroring session:

1. I create the mirror database by restoring the following backups, using **restore with no recovery** on every restore operation:

a) Restore a recent full database backup of the principal database. The mirror database must have the same name as the principal database.
b) Restore most recent differential backup.
c) Restore all the log backups done since the full or differential database backup.

2. I then set up mirroring by using either Transact-SQL or the Database Mirroring Wizard.

3. I reconfigure the session to run in high-safety mode with automatic failover by adding a witness server instance.