ECM1420 Information and Data

Database Design and Implementation Exercise

Hand-out date: 11th January 2021
Hand-in date: Thursday 25th March 2021, 12 noon
Feedback: 30th April 2021

This assessment is worth 50% of the overall mark for ECM1420. This is an individual assessment. You are reminded of the University's regulations on plagiarism.

Purpose: To reinforce and integrate database design and SQL skills.

Introduction: Nympton Food Hub

Harry and Ashley run a roadside café/restaurant at Nympton in rural Devon. Their business was interrupted in March 2020 when, in response to a national emergency, the government imposed a lockdown on the population and a closure of restaurants.

Faced with a loss of all their income, Harry and Ashley decided to start a local 'food hub'. This would act as a grocery delivery service to local residents who were confined to home. It would use existing suppliers to the restaurant and make additional use of small, local fresh food producers.

The new business had to be up-and-running as soon as possible and solutions to practical, logistical problems initially took priority over administrative issues. However, it soon became clear that data and information issues were very important even to this small, new venture.

Examples of challenging data-related issues:

- Customer satisfaction demanded that orders were satisfied accurately.
- Prices of fresh goods changed almost daily.
- In a rural area it could be very difficult to find a delivery location and the best delivery route.
- Customers' accounts needed to be kept up to date.

Harry and Ashley were adaptable young people who were very comfortable with mobile phone technology and the use of electronic payment systems, but they were new to the idea of managing integrated information systems on a computer. Initially, for instance, their customer contact records came from mobile phone address books – on 3 different phones. In contrast their records of supplier deliveries originated in paper delivery notes. Customer orders could come verbally face-to-face, over the phone, written on scraps of paper, or by email. Customer invoices were at first written by hand.

There were problems with this approach right from the start. It was common for some items of a customer's order to not be available. Keeping track of this and billing the customer for the correct amount could be tricky. It was possible for errors to creep into the process. An error in a customer account could take hours to correct and this was liable to have a damaging impact on the business.

Harry and Ashley gradually built up a series of spreadsheets that held the vital data for the business. This enabled them to improve the accuracy of their data and the efficiency of some operations such as printing price lists. The purpose of our present exercise is to take this progress further by integrating the spreadsheet records in a relational database. As an initial project we are going to concentrate on the 'customer-facing' part of the business which generates most of the business data and where most time-consuming errors can be generated. This means that a detailed bank account and suppliers' records, for example, will not be included at this stage.

Data

You are provided with a series of comma separated values (csv) files that have been exported from the existing spreadsheets.

Details of the purposes of the files and some of their columns is given below. The *id* column in each file is a unique identifier for the record.

csv file name	Purpose of file/table	Fields/columns	Purpose of field/column
StockCategory	Used to split stock items up into useful groups.	Id	Unique identifier
		Category	Descriptive text for the stock category.
		Display Order	Used to control the order of categories in reports and displays.
Contact	Used to hold contact details of a person/organisation	what3words	Used to precisely locate a contact address https://what3words.com/
Unit	Unit of purchase or sale of a stock item.	Unit	Descriptive text. Used as an aid to rapid data entry on the user interface. Not intended to be subject to referential integrity.
PaymentMethod	The method used to transfer funds.	PaymentMethod	Descriptive text. Used as an aid to rapid data entry on the user interface. Not intended to be subject to referential integrity.
Customer	Holds details of a Food Hub customer	Add_Info	Narrative text for additional information
		Deliver_Info	Narrative text for delivery related information
StockItem	Holds data for items available for purchase	Id	Unique identifier
		Item Name	Descriptive text
		Item Unit	Unit of purchase by customer
		Item Price	Sales price in £ for Item Unit
		Available?	Whether this Item currently available for sale
		Item Category	StockCategory to which Item belongs
		ItemAdditionalInfo	Any useful associated text
SalesOrder	Details of Customers' Orders	OrderDate	Date order entered the system.
		Picked?	Whether the component order lines have been assembled ready for delivery (Y/N)
		Complete?	Whether the order has been delivered, payment received and should no longer be live.

		GoodsCost	Total cost of the goods in the sales order lines.
		ExtrasCost	Costs attached to this sales order above that of good on order lines
SalesOrderLine	Details of a component line of a Sales Order	ItemCost	Should generally be ItemQty x ItemPrice
		Item_id	Technically a redundant column as Item Name contains the necessary data.
SalesInvoice	Details of a sum due from a customer as the result of the fulfilment of an order	Paid?	Details whether the customer has fully paid the invoice (Y/N).
SalesInvoiceLine	Details of a component line of a Sales Invoice	Item_id	Technically a redundant column as Item Name contains the necessary data.
CustomerPayment	Details funds received from customers to pay for amounts invoiced.	PaymentMethod	The method used to transfer funds.
		Order_id	Technically a redundant column as invoice_id contains the necessary link.
CustomerAccount	Holds details of financial transactions with customers	Narrative	The reason for the transaction
		Adj_id	Redundant at present, but included to allow for adjustments due to errors and omissions.
		DebitAmount	Amount charged to the customer when an invoice is raised against them.
		CreditAmount	Records amounts received from the customer.
Supplier	Holds details of a Food Hub supplier	SupplierName	Trading name of the supplier organisation
SupplierStockItem	Links Suppliers to StockItems	Supplier_id	Link to Supplier
		StockItem_id	Link to StockItem

Actions Required

(Before attempting any database work, you should check that you understand the relevance of the data in the csv files.)

1. Create the SQL Server database for *Nympton Food Hub*.
2. Create suitable database tables using SQL code to receive the data from the csv files. Ensure that the columns have suitable data types, auto-incrementing where necessary, and set the NULL/NOT NULL constraint appropriately. Create the Primary Keys at this stage, but not the Foreign Keys. Save the SQL scripts that you use to create the tables. They can be combined into one file.
3. Import data from the csv files into your database tables.
4. Write and run SQL script to create the foreign keys constraints that are required to maintain referential integrity between your database tables. Save the scripts. They can be combined into one file.
5. Write and run SQL script to create a view named CustomerDebt. This will read the CustomerAccount table and select only those Customers with an outstanding balance (i.e. debits on their account exceed credits on their account). The View should have 2 columns: customerid and balance. Save the code that you use to create this View.
6. Write stored procedures to satisfy the reporting requirements detailed overleaf. Save the code that you use to create the stored procedures.
7. Run each of the stored procedures using the parameters specified overleaf. Copy and save the resulting output (including headers) for submission.
8. Review your stored procedure ReceiveFullPaymentonInvoice. Clearly it would be incorrect to update the SalesInvoice record as Paid = 'Y' if the CustomerPayment record creation had failed. Write a short explanation of how this might be avoided using T-SQL. Indicate the T-SQL commands and the method that you might use to revise the procedure. Full code is not required.
9. Create a database diagram using SQL Server Management Studio (SSMS) (or another suitable tool) to show five of the database tables and the-relationships between them including their cardinality. Also show the columns of the tables. The tables required are SalesOrder, SalesInvoice, Customer, CustomerAccount and CustomerPayment. Arrange the diagram to make it as clear and legible as possible.

Technical Note:

Following the import of data into from the CustomerAccount.CSV file use may need to run code to set all receipt ids of 0 to NULL. This is because of an omission in the import wizard.

Reporting requirements

1. Price List by Category/Item

   The aim of this procedure is to retrieve the data required to print or display a Stock Item price list.

   Details: For each available Stock Item list the Item Name (Item) , Item Unit (Unit) , Item Price (Price) and Stock Category (Category - text of the Category column in StockCategory). The Stock Items should be ordered in Display Order of Stock Categories and then alphabetically within each Category.

2. Picking List

   The aim of this procedure is to provide of list of all the items (and required quantity) that need to be picked from stock to satisfy currently outstanding orders.

   Details: Select the sales order lines for sales orders that haven't been picked. For each Stock Item that appears show the Stock Item Id as Id, item name as Stock Item, itemunit as Unit and total quantity required of the item as Quantity

3. Customer Outstanding Balances

   The aim of this procedure is to identify those customers with an outstanding balance on their account and to display this balance and the customer's contact details.

   You have created a database View named 'CustomerDebt'.

   Details: Use your View 'CustomerDebt' to show details of Customers who have outstanding debit balances. Show the CustomerId as *Id*, First Name + LastName (separated by a space) as *Name*, phonelandline as *Phone*, phonemobile as *Mobile*, the outstanding balance as *Balance* and the date of the customer's last payment as *Last Paid*.. Display records in descending order of outstanding balance.

4. Recent Demand for Stock Item

   The aim of this procedure is to provide an indication of the recent trend in order quantities of a given stock item.

   Details: This procedure will take the **name** of a stock item and a base date as **input parameters**. The procedure will query the sales order lines for the stock item by date. A one line output will be produced with 4 columns as follows: the input parameter (name of the input stock item) as *Stock Item*, the total quantity of the item ordered in the last 10 days **before** base date (1 to 10 days before) as *Last 10 Days*, total quantity ordered 11-19 days **before** base date as *11-19 days* and total quantity ordered 21-30 days **before** base date as *21-30 days*.

   Run the procedure with input parameters 'Butternut Squash' and '2020-05-01'.

5. Receive Full Payment on Invoice

The aim of this procedure is to update the database when full payment is received on an invoice by the BACS method.

Details: This procedure will take an Invoice Id and a payment date as parameters. The procedure will check the relevant invoice record to see whether it is already paid. If this is the case then the procedure will skip to paragraph (3) below.

(1) If the invoice has not already been paid then the procedure will create a new CustomerPayment record using relevant details from the invoice record and the payment date parameter. The payment method will be recorded as BACS. The procedure will then use a SELECT statement to return the newly created CustomerPayment record.
(2) The procedure will then update the invoice record as 'Paid'.
(3) Return the relevant Sale Invoice record..

Execute the procedure with invoice id parameter as 933 and payment date '2020-05-08'.

Hint: Search for 'SQL Server output inserted' for some sophisticated techniques.

Notes:

All SQL code used should run correctly as T-SQL on SQL Server 2019 or 2017.

Do NOT attempt to further denormalise the database in order to make these reports easier to produce.

Documentation for SQL Server built-in functions for date, string and number manipulation may be found at https://www.w3schools.com/sql/sql_ref_sqlserver.asp

Useful information on the use of sub-queries can be found at https://www.sqlservertutorial.net/sql-server-basics/sql-server-subquery/

Useful information on commitment control:

https://stackoverflow.com/questions/10153648/correct-use-of-transactions-in-sql-server

https://www.mssqltips.com/sqlservertutorial/3305/what-does-begin-tran-rollback-tran-and-commit-tran-mean/

https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-try-catch/

Submission Required

Please submit your work anonymously as a *pdf* document via the supplied link on the ELE Module Information page by 12noon on the due date. The title of your assignment must be as follows:

'Student ID number [insert your student Id number here] Assignment title [insert assignment title here]'

Please include your Student Id number at the top of each page of your assignment and do not include your name.

Please ensure that you use the above title format when you upload and name your assignment within ELE.

Present your results in the form of a single pdf document (A4 page size) containing the following.

1. Cover page to include your student ID number. Omit your name and any other identifying information.

2. The SQL code that you used in action point 2 to create the following database tables **ONLY**, including primary keys; *StockCategory*, *StockItem*, *Customer*, *SalesOrder*, *SalesOrderLine*, *SupplierStockItem*. (Tip: Save your SQL scripts and copy and paste the code into your submission document). (15%)

3. The SQL code that you used in action point 4 to create the Foreign Key constraints in the *SupplierStockItem* and *CustomerPayment* tables **only**. (5%)

4. SQL script that you used to create the View named CustomerDebt. (5%)

5. SQL code for the five stored procedures from action point 6, one for each of the reporting requirements overleaf.

   Copy and paste your SQL code used to create the procedures into your submission document. Run each of the stored procedures in SQL Server Management Studio and then copy the result, with headers, into your submission document. Only a **maximum of 10** data records is required for each report.

   Submit the report output together with the SQL code for each report from action point 7.

   (50%).

6. Your explanation of revisions required to procedure to receive full payment on a invoice. (10%)

7. Your database diagram from action point 9, using the SSMS database diagram tool to show tables and their columns, relationships between the tables and their cardinality.(15%)