FACTFILE: GCE SOFTWARE SYSTEMS DEVELOPMENT

DEVELOPING & IMPLEMENTING RDMS

Developing and implementing a desktop solution using a RDMS through an event driven environment

Learning outcomes

Students should be able to:

- apply techniques of relational database management systems to provide solutions to the given problem;
- use an appropriate software tool to connect the required database to the Graphical User Interface (GUI) for the specified problem;
- implement the solution using:
  - multiple forms;
  - appropriate navigation; and
  - connection to the database through data adaptors, data sets, data commands;
- present the solution using appropriate media.

Content

The C# GUI Environment
Multiple forms are required to implement a structured solution to a problem. It is normally advisable to use separate forms for Inserts, Displays, Updates and Deletes. This provides clarity for the user and simplifies maintenance of code.

Inserting data must use Error Trapping and also select relevant data from controls such as, Listboxes, ComboBoxes and GridViews where possible.

To display data, forms should provide several views of the required data in order to satisfy user needs. This can be achieved by use of the Tab Control. Filters should be implemented to facilitate searches by minimising the range of data.

Updating data uses the techniques of searching, displaying and inserting valid data.

Criteria for deleting data must always be considered. An example is, ‘ .. deletion of a customer who has made purchases in the current financial year is illegal’.

Navigation is normally by the use of drop down menus as they are more transparent and the levels are easily viewed within each section. A button menu system is more suited to simple problems as the navigation down through the levels can be confusing for larger systems.

There is a movement of data between the GUI environment and the database. To facilitate this, a range of controls is provided to connect and manipulate data.

Controls provided by Microsoft to connect to the SQL Server Database include:

Component		Features
SqlConnection		Represents an open connection to the SQL Server database
SqlDataAdapter	da	Used to control data movement to/from SQL Server database
SqlCommand	cmd	Represents a Transact-SQL statement or stored procedure to execute against a SQL Server database. Use when parameters are required.
SqlCommandBuilder	cmdB	Generates single table commands that reconcile changes to the dataset with the SQL Server database. Used for Add/update/delete.
DataSet	ds	Set of data from a table or multiple tables from the database.
DataRow	dr	Storage for one row of a table in the data set.
ListView	lsv	Displays data in rows and columns.
DataGridView	dgv	Displays data in rows and columns. Allows View/add/edit/delete.

Table 1.1 – Controls for connecting to a SQL Server Database

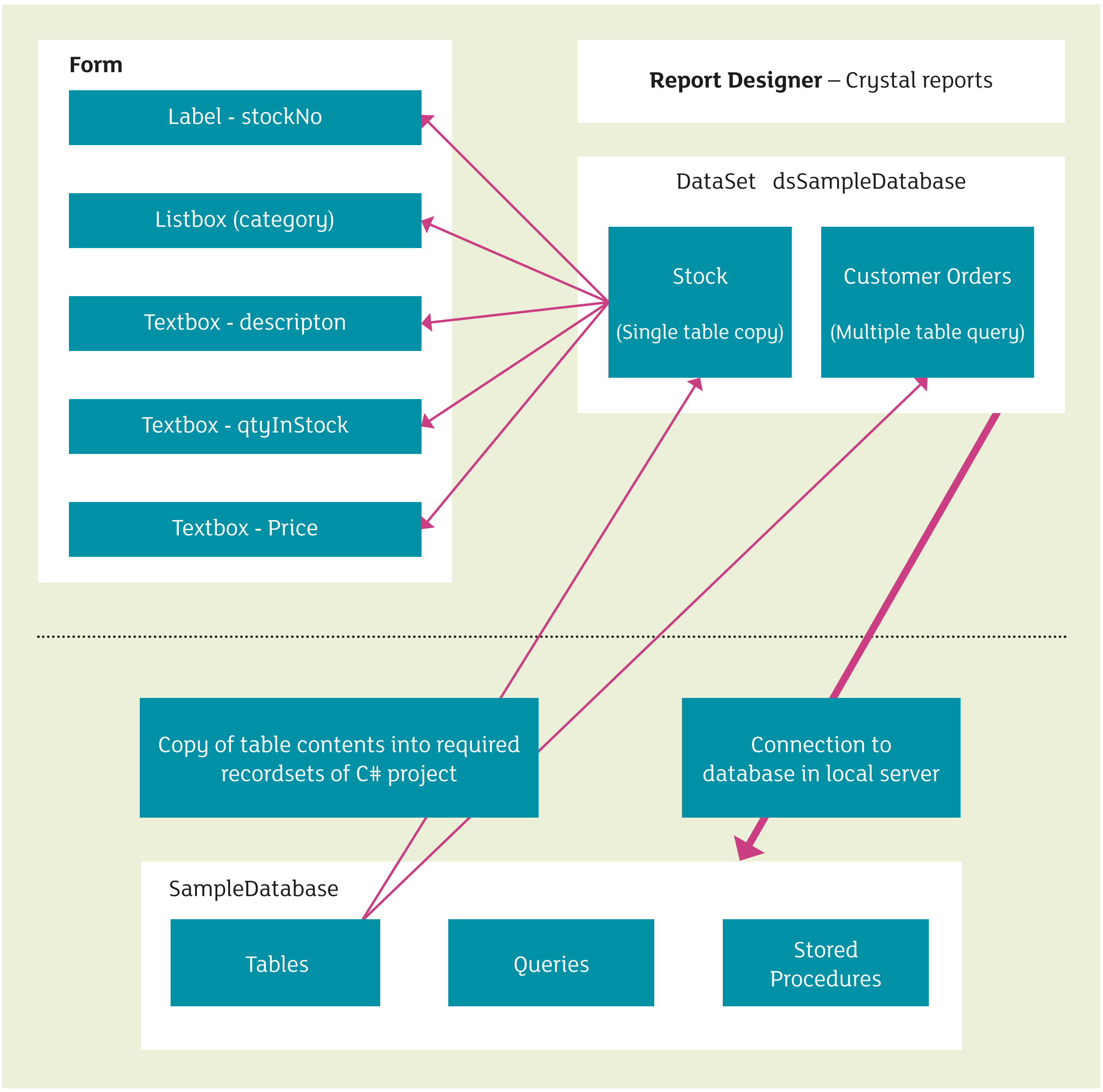


Figure 1.1 – A simple view of the relationship between a form and the database on the server

To reference the stock table in the dataset use **dsSampleDatabase.Tables ["Stock"]**

Connection to a database

To open a database, the server, the name of the database and security settings are specified. The string should be declared once in the Properties/Resources section for efficiency in maintaining the application. This is referenced as required in the relevant forms.

Example of opening a connection to a database

```
String cnstr = Properties.Resources.connectStr;

The string is @"Data Source = localhost\sqlexpress; Initial Catalog = SampleDatabase;
Integrated Security = true";
```

Hold the connection string (connectStr) **once** in Properties and reference on relevant forms

open Properties
open Resources
dblck on resources.resx
fill in value

Example

Name	*data source*
connectStr	Data Source=localhost\sqlexpress; Initial Catalog=SampleDatabase; Integrated Security=true

Implement Single Table Database Processes

ADD/DISPLAY/UPDATE/DELETE

Note: A Datagrid can be used to accomplish any or all of these processes for table data. This is not recommended for casual users.

The use of separate forms is recommended for each process to simplify coding, maintenance of the system and for ease of use.

Multiple forms should be designed with a menu system where the most frequent processes are aligned left to right.

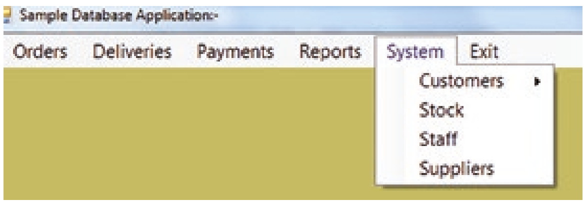


Figure 1.2 – User Interface showing menus

Changes to the database **must** be controlled at form level:

- validate relevant fields as required according to the database constraints;
- ensure foreign key constraints are met by using listbox /combobox selection;
- ensure at least one child exists for a parent after an add or part delete for example an order must have an order detail;
- ensure only relevant data is made available for deletion.
- Consider financial/audit restrictions.

Add records to a database table

It is important to validate the fields at form level to minimise internet flow and prevent errors occurring at the database level. The design of the form should indicate the required fields clearly. Where possible determine and set values such as the key field value and provide selection from listboxes.

Adding records to the database will require use of a **CommandBuilder** with the **DataAdapter.**

See Appendix 1:

- an example of an add to a customer table in the SampleDatabase with basic validation;
- an example using validation within the **class** Customer and try – catch feature.

Display information from the database tables

Information should be displayed in a format that is user friendly. Consider the different reasons why a user might want to view the information.

For example:
- A telephone inquiry may have the customer name available but require the Customer No. Therefore a search by customer name is necessary;
- A manager may need to contact a customer and require the address or Tel No. Therefore a search by Customer No or customer name is necessary;
- Marketing may need all the customer details for a particular Type of customer. Therefore a search by customer type is necessary.

A TabControl is useful as it allows several 'views' of the information to be displayed.

The realistic size of a table must be considered. There may be hundreds or thousands of customers or stock items. **Filter** mechanisms should be used to reduce the search time where relevant.
Examples;
- Selection of stock category from a ListBox to reduce the number of stock items to be displayed;
- Selection of an alphabet button to display customers whose surnames start with that letter.

Filtering records from the database will require the use of a **SqlCommand** with the **SqlDataAdapter** as a parameter for the first letter of CustomerSurname.

See Appendix 2:

- display of customers by customerNo; (Tab1 of TabControl)
- display of an individual customer's full details after selection from a listbox (name held alphabetically) or by an alphabetic filter. (Tab2 of TabControl)

The Update form has similarities to the Display form as the record must first be selected for editing. It also resembles the Add form as validation must be carried out before allowing the update to be committed. The key field should not be editable if it has been automatically created in the Add form.

Note - It is recommended that the name of the table in the dataset should indicate an update e.g. CustomerUpd. SqlCommandBuilder is required for this table.

Example of Update code

```
int no = int.Parse( txtCustomerNo.Text);
drCustomer = dsSampleCode.Tables[“CustomerUpd”].Rows.Find(no);

drCustomer.BeginEdit();

drCustomer[“CustomerNo”] = txtCustomerNo.Text;            // use if not automatic field
drCustomer[“Title”] = lstTitle.SelectedItem();            // listBox of valid titles
drCustomer[“CustomerSurname”] = txtSurname.Text.Trim();
drCustomer[“CustomerForename”] = txtForename.Text.Trim();
drCustomer[“Address1”] = txtAddress1.Text.Trim();
drCustomer[“Address2”] = txtAddress2.Text.Trim();
drCustomer[“Address3”] = txtAddress3.Text.Trim();
drCustomer[“PostCode”] = txtPostcode.Text.Trim();         // maskedTextBox
drCustomer[“TelNo”] = txtTelNo.Text.Trim();        // maskedTextBox

drCustomer.EndEdit();
daCustomerUpd.Update (dsSampleDatabase, “CustomerUpd”);

// refresh selection ListBox to mirror changes - if any - to name
```

The Delete Form has similarities to the Display form as the record must first be selected for deletion.

Select and display the record for deletion:

Set up sqlCustomerDel, cmdBCustomerDel, daCustomerDel and table with fillSchema and
fill to handle deletion;
Fill dataset table –CustomerDisplay, with customers (**apply criteria** – e.g. customers with no orders);
Fill listbox/listview from CustomerDisplay table;
Select from listbox/ listview;
Find and display details;
Use a MessageBox to ask for confirmation of delete.

Code for delete:

```
if(okToDelete)
{
drCustomer = dsSampleDatabase.Tables[“CustomersDel”].Rows.Find(customerNo);
drCustomer.Delete();
daCustomerDel.Update(dsKennelKare, “CustomersDel”);
// use MessageBox to give feedback of successful deletion
}
```

Processes involving Multiple Tables in the Database

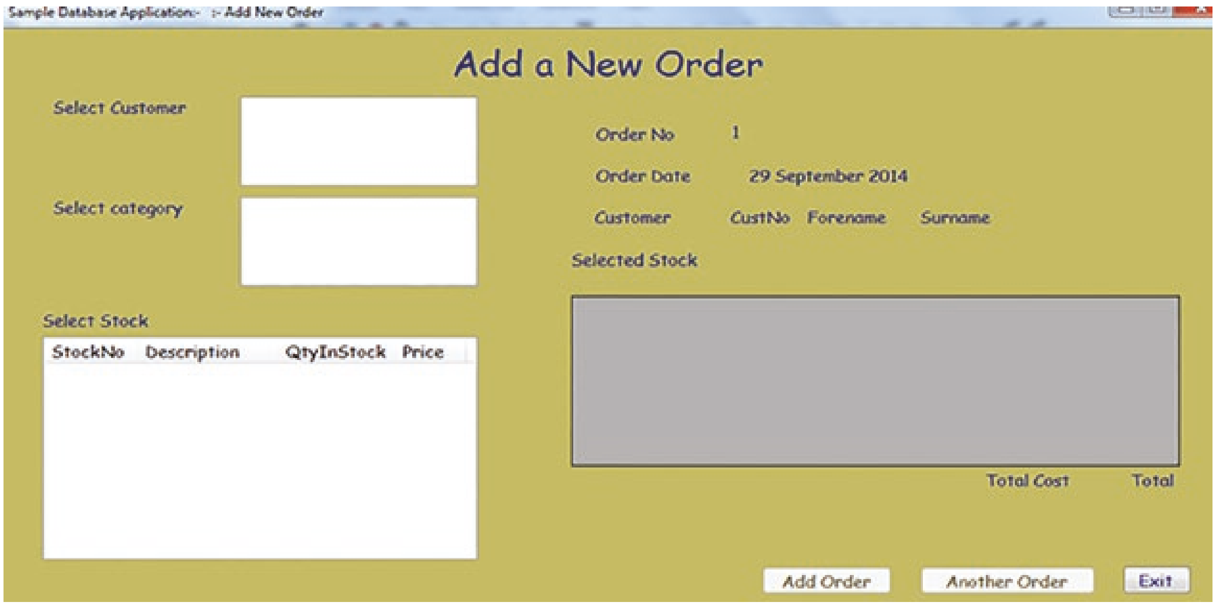
ADD / DISPLAY / UPDATE / DELETE

More complex code is required for processes relating to multiple tables. For example,

Add an order	Adds order header to the Order table and order items to theOrderDetails table. This must ensure at least one order item detail is added with the order header.
Display Orders	View all orders for a customer View individual customer order View orders by date.
Update Orders	Add another order item detail Remove an order item (must leave at least one item) Change orderQty for an order item **Note** • Only orders not yet delivered or paid for may be updated. • Partial delete is covered by removing an item in **Update Orders**
Delete Order	Order item details and order header are deleted. **Note** Only orders **not** yet delivered or not paid for may be deleted.

Table 1.2 – Processes involving Multiple Tables in the Database

ListViews and **DataGridViews** Controls are very useful to ensure database integrity, for example:


Sample Database Application:- :- Add New Order
Add a New Order
Select Customer
Select category
Select Stock
StockNo Description QtyInStock Price
Order No 1
Order Date 29 September 2014
Customer CustNo Forename Surname
Selected Stock
Total Cost Total
Add Order
Another Order
Exit

listView of Stock for selection

dataGridView of selected stock details + columns orderQty (editable), cost(calculated)

Figure 1.3 – Add a new order form