Network Database Applications Assignment 1: Individual Activities

Gary Moore

November 14, 2017

1 Introduction

With the planning completed for the library database system, what remains to be implemented is the database itself. The database is to be implemented using Transactional Structured Query Language which will require a database server, and database management software for writing queries and executing them on the server.

2 Requirements

The following software will be used:

- MSSQL Server 2017 for Docker
- SQLPro management software for macOS

I'll be using the Docker image of MSSQL Server as it can be deployed on any operating system that can run a virtual machine. It also runs from within a sandboxed container, which makes it measurably more secure than running it on the host machine. I personally needed to use a Docker image as MSSQL Server is not software that can run natively on macOS.

Docker will likely only be used for the development of the database. When moved to a production environment, the database will likely be exported and run natively on a Windows server.

Typically, SQL Management Studio by Microsoft is used to manage T-SQL databases, but due to the software only being available on one platform I used SQLPro, which has a layout that matches very closely to the Microsoft software.

3 SQL Development

3.1 tblContact

The first table will be `tblContact` as `tblCustomer` and `tblEmployee` are dependent upon it.

```
CREATE TABLE tblContact
(
        cnContactID INT IDENTITY (1,1) PRIMARY KEY,
        cnFirstName VARCHAR(30) NOT NULL,
        cnSurname VARCHAR(30) NOT NULL,
        cnPhone CHAR(11) NOT NULL,
        cnEmail VARCHAR(30) NOT NULL,
        cnAddress1 VARCHAR(30) NOT NULL,
        cnCity VARCHAR(30) NOT NULL,
        cnCounty VARCHAR(30) NOT NULL,
        cnPostcode VARCHAR(7) NOT NULL
)

INSERT INTO tblContact
(cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty,
    ↪  cnPostcode)
VALUES
('Gary', 'Moore', '07500564955', 'madmangaz@gmail.com', '49␣Woodvale␣
    ↪ Park', 'Dungannon', 'Co␣Tyrone', 'BT716DB'),
('Steven', 'Stove', '02887723114', 'stovesteven@gmail.com', '70␣Stove␣
    ↪ Road', 'Stoveland', 'Co␣Stove', 'BT115DF'),
('Dolly', 'Donko', '07545652412', 'donkodolly@gmail.com', '32␣Frillo␣
    ↪ Road', 'Dollyland', 'Co␣Donko', 'BT323FG'),
('Willy', 'Wonko', '07548895621', 'wonkowilly@gmail.com', '49␣Wonko␣
    ↪ Close', 'Wonkoworld', 'Co␣Willy', 'BT63BN'),
('Henry', 'Hob', '07502314592', 'hobhenry@gmail.com', '21␣Hob␣Lane', '
    ↪ Henrytown', 'Co␣Hob', 'BT125NA'),
('Nobby', 'Noodle', '07504562312', 'noodlenobby@gmail.com', '21␣Noodle␣
    ↪ Way', 'Nobbycity', 'Co␣Noodle', 'BT426FN'),
('Willy', 'Willard', '07501228543', 'willardwilly@gmail.com', '21␣
    ↪ Garfield␣Lane', 'Grungetown', 'Co␣Tables', 'BT124DB'),
('Shellie', 'Shirt', '07542655489', 'shirtshellie@gmail.com', '141␣
    ↪ Dingdong␣Avenue', 'Dingdongcity', 'Co␣Dole', 'BT126DF'),
('Rupert', 'Rung', '02845621453', 'rungrupert@gmail.com', '12␣Rung␣
    ↪ Street', 'Rungland', 'Co␣Ring', 'BT47FG'),
('Berty', 'Balls', '07512545698', 'ballsberty@gmail.com', '5␣Bumble␣Road
    ↪ ', 'Bundo', 'Co␣Herk', 'BT436BN'),
```

```
('Jingo', 'Jango', '07512459821', 'jangojingo@gmail.com', '9␣Honk␣Road',
    ↪  'Linda␣City', 'Co␣Suit', 'BT235HN'),
('Will', 'Wharg', '07512324582', 'whargwill@gmail.com', '99␣Nine␣Lane',
    ↪ 'Niner␣Valley', 'Co␣Number', 'BT654BA')

SELECT * FROM tblContact
```

The `CREATE TABLE` command will create the table `tblCustomer`, with columns then entered in the bracket delimiters. The first common is an `INT`, which is shorthand for integer. It is an auto incrementing number that is also the primary key, with `IDENTITY (1,1)` meaning it increments by one with each new field and `PRIMARY KEY` denoting that this is a fields identifying column. Each primary key is automatically enforced with constraints to ensure that a null value isn't used, and that each field must have a unique primary key.

Almost every key after the primary key uses the `VARCHAR` datatype, which can contain a string, and is of variable length. A column of this data type that's notable is `cnPostcode`. A post code is typically considered to be fixed length, however there's a fairly common edge case of a 6-character long post code, for example BT7 5NA. This is why `VARCHAR` is used instead of `CHAR`. Another notable aspect of these columns is most of them use the constraint `NOT NULL`, which means that a null value cannot be entered into this column.

`CHAR` of length 11 is used for the phone number as an integer isn't appropriate here because we don't want to perform any calculations on a phone number, and it is possible to enter a value in excess of the 32bit integer limit. `CHAR` is used instead of `VARCHAR` because all UK phone numbers are of length 11.

To insert fields into this table we first use the `INSERT INTO` command followed by the table name `tblContact`. Next we declare what columns we wish to input information into in delimiters separated by commas. On a new line you write the `VALUES` keyword followed by delimited entries on a new line with strings surrounded by single quotes and numbers entered without quotes separated with a comma.

The command `SELECT * FROM tblContact` will display all fields in the table.

3.2 tblCustomer

This is a small table as most of the details are contained in `tblContact`. The two tables use a one to many relationship to avoid redundant data.

```
CREATE TABLE tblCustomer
(
       cuCustomerID INT IDENTITY (1,1) PRIMARY KEY,
       cnContactID INT FOREIGN KEY REFERENCES tblContact(cnContactID)
)

INSERT INTO tblCustomer
(cnContactID)
VALUES
```

```
(8),
(9),
(10),
(11),
(12)

SELECT * FROM tblCustomer
```

This table is the first case where a foreign key is used, and a different syntax is used to create a foreign key compared to a primary key. Instead of declaring it with `PRIMARY KEY`, we write `FOREIGN KEY REFERENCES` *`table`*`(`*`column`*`)`. The purpose of a foreign key is to link this table with another table that it references data from.

We must associate each customer with its corresponding entry in tblContact, and to do this we insert the corresponding foreign key value into tblCustomer.

3.3 tblEmployee

Similarly to tblCustomer we must initialise a foreign key, however there are extra fields for this table with employee specific information.

```
CREATE TABLE tblEmployee
(
        emEmployeeID INT identity(1,1) PRIMARY KEY,
        emPassword VARCHAR(30) NOT NULL,
        emPosition VARCHAR(20) NOT NULL,
        emPermissions VARCHAR(10) NOT NULL,
        cnContactID INT NOT NULL FOREIGN KEY REFERENCES tblContact(
            ↪ cnContactID)
)

INSERT INTO tblEmployee
(emPassword, emPosition, emPermissions,cnContactID)
VALUES
('password1', 'andy.admin', 'root', 1),
('password2', 'billy.bob', 'user', 2),
('password3', 'willy.wonka', 'user', 3),
('password4', 'robert.rub', 'user', 4),
('password5', 'henry.hob', 'user', 5),
('password6', 'willard.wong', 'user', 6),
('password7', 'guest', 'restricted', 7)

SELECT * FROM tblEmployee
```

Every employee will need a password to get access to the database system and appropriate permissions that will define what that employee can change in the database. For

example an administrator will be able to drop and add tables, where as a user may only be able to alter or add data to the database.

3.4 tblAuthor

This table will have a many to many relationship with `tblBook`. A junction table will be used between the two tables, more information will be in section 3.5.

```
CREATE TABLE tblAuthor (
    auAuthorID INT identity(1,1) PRIMARY KEY,
    auFirstName VARCHAR(30) NOT NULL,
    auSurname VARCHAR(30) NOT NULL
)

INSERT INTO tblAuthor
(auFirstName, auSurname)
VALUES
('Arty', 'Author'),
('Steven', 'Writer'),
('Henry', 'Scribe'),
('Barry', 'Book'),
('Terry', 'Tree'),
('Sabrina', 'Scroll')

SELECT * FROM tblAuthor
```

3.5 tblBookAuthor