# Network Database Applications Assignment 1: Individual Activities

Gary Moore

November 15, 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.

```
1  CREATE TABLE tblContact
2  (
3         cnContactID INT IDENTITY (1,1) PRIMARY KEY,
4         cnFirstName VARCHAR(30) NOT NULL,
5         cnSurname VARCHAR(30) NOT NULL,
6         cnPhone CHAR(11) NOT NULL,
7         cnEmail VARCHAR(30) NOT NULL,
8         cnAddress1 VARCHAR(30) NOT NULL,
9         cnCity VARCHAR(30) NOT NULL,
10        cnCounty VARCHAR(30) NOT NULL,
11        cnPostcode VARCHAR(7) NOT NULL
12 )
13
14 INSERT INTO tblContact
15 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty,
       ↪ cnPostcode)
16 VALUES
17 ('Gary', 'Moore', '07500564955', 'madmangaz@gmail.com', '49 Woodvale Park', '
       ↪ Dungannon', 'Co Tyrone', 'BT716DB'),
18 ('Steven', 'Stove', '02887723114', 'stovesteven@gmail.com', '70 Stove Road', '
       ↪ Stoveland', 'Co Stove', 'BT115DF'),
19 ('Dolly', 'Donko', '07545652412', 'donkodolly@gmail.com', '32 Frillo Road', '
       ↪ Dollyland', 'Co Donko', 'BT323FG'),
20 ('Willy', 'Wonko', '07548895621', 'wonkowilly@gmail.com', '49 Wonko Close', '
       ↪ Wonkoworld', 'Co Willy', 'BT63BN'),
21 ('Henry', 'Hob', '07502314592', 'hobhenry@gmail.com', '21 Hob Lane', 'Henrytown
       ↪ ', 'Co Hob', 'BT125NA'),
22 ('Nobby', 'Noodle', '07504562312', 'noodlenobby@gmail.com', '21 Noodle Way', '
       ↪ Nobbycity', 'Co Noodle', 'BT426FN'),
23 ('Willy', 'Willard', '07501228543', 'willardwilly@gmail.com', '21 Garfield Lane
       ↪ ', 'Grungetown', 'Co Tables', 'BT124DB'),
24 ('Shellie', 'Shirt', '07542655489', 'shirtshellie@gmail.com', '141 Dingdong
       ↪ Avenue', 'Dingdongcity', 'Co Dole', 'BT126DF'),
25 ('Rupert', 'Rung', '02845621453', 'rungrupert@gmail.com', '12 Rung Street', '
       ↪ Rungland', 'Co Ring', 'BT47FG'),
26 ('Berty', 'Balls', '07512545698', 'ballsberty@gmail.com', '5 Bumble Road', '
       ↪ Bundo', 'Co Herk', 'BT436BN'),
27 ('Jingo', 'Jango', '07512459821', 'jangojingo@gmail.com', '9 Honk Road', 'Linda
       ↪  City', 'Co Suit', 'BT235HN'),
28 ('Will', 'Wharg', '07512324582', 'whargwill@gmail.com', '99 Nine Lane', 'Niner
       ↪ Valley', 'Co Number', 'BT654BA')
29
30 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.

```
1  CREATE TABLE tblCustomer
2  (
3         cuCustomerID INT IDENTITY (1,1) PRIMARY KEY,
4         cnContactID INT FOREIGN KEY REFERENCES tblContact(cnContactID)
5  )
6
7  INSERT INTO tblCustomer
8  (cnContactID)
9  VALUES
10 (8),
11 (9),
12 (10),
13 (11),
14 (12)
15
16 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.

```
1  CREATE TABLE tblEmployee
2  (
3         emEmployeeID INT identity(1,1) PRIMARY KEY,
4         emPassword VARCHAR(30) NOT NULL,
5         emPosition VARCHAR(20) NOT NULL,
6         emPermissions VARCHAR(10) NOT NULL,
7         cnContactID INT NOT NULL FOREIGN KEY REFERENCES tblContact(cnContactID)
8  )
9
10 INSERT INTO tblEmployee
11 (emPassword, emPosition, emPermissions,cnContactID)
12 VALUES
13 ('password1', 'andy.admin', 'root', 1),
14 ('password2', 'billy.bob', 'user', 2),
15 ('password3', 'willy.wonka', 'user', 3),
16 ('password4', 'robert.rub', 'user', 4),
17 ('password5', 'henry.hob', 'user', 5),
18 ('password6', 'willard.wong', 'user', 6),
19 ('password7', 'guest', 'restricted', 7)
20
21 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 3.6.

```
1  CREATE TABLE tblAuthor (
2      auAuthorID INT identity(1,1) PRIMARY KEY,
3      auFirstName VARCHAR(30) NOT NULL,
4      auSurname VARCHAR(30) NOT NULL
5  )
```

```
6
7  INSERT INTO tblAuthor
8  (auFirstName, auSurname)
9  VALUES
10 ('Arty', 'Author'),
11 ('Steven', 'Writer'),
12 ('Henry', 'Scribe'),
13 ('Barry', 'Book'),
14 ('Terry', 'Tree'),
15 ('Sabrina', 'Scroll')
16
17 SELECT * FROM tblAuthor
```

For any new books that are added to the database system, the author names will go into a second document to avoid redundant data, as authors can write more than one book.

### 3.5 tblBook

This table has a many to many relationship with `tblAuthor`.

```
1  CREATE TABLE tblBook (
2      bkISBN VARCHAR (13) PRIMARY KEY NOT NULL,
3      bkTitle VARCHAR (50) NOT NULL,
4      bkGenre VARCHAR (30) NOT NULL
5  )
6
7  INSERT INTO tblBook
8  (bkISBN, bkTitle, bkGenre)
9  VALUES
10 (1235785642842, 'The Big Thing', 'SciFi'),
11 (7512354894217, 'Blast Off from the Crescent Moon', 'Farty Tale'),
12 (4216548789512, 'The Big One', 'Romance'),
13 (4865423157, 'Back in my Day', 'Memoir'),
14 (4562147895444, 'Dingo the Dog', 'Childrens'),
15 (7825123786215, 'Dirty Dan', 'Spaghetti Western')
16
17 SELECT * FROM tblBook
```

A unique aspect of this table; it uses the ISBN number of the book as its primary key. Because the number is so big, we use `VARCHAR` instead of the `INT` data type, which can be used as the primary key so long as it's unique. We're using a variable data type because an ISBN can be 10 or 13 digits long.

We also create columns for the title and genre of the book and use `INSERT INTO` to fill out some values.

### 3.6 tblBookAuthor

This is a junction table between `tblBook` and `tblAuthor`. The purpose of this table is to create a link between these two tables, as a many to many relationship cannot exist directly between these two tables.

```
CREATE TABLE tblBookAuthor (
    auAuthorID INT NOT NULL FOREIGN KEY REFERENCES tblAuthor(auAuthorID),
    bkISBN VARCHAR(13) NOT NULL FOREIGN KEY REFERENCES tblBook(bkISBN),
    PRIMARY KEY (auAuthorID, bkISBN)
)

INSERT INTO tblBookAuthor
(auAuthorID, bkISBN)
VALUES
(1, '1235785642842'),
(2, '4216548789512'),
(3, '4562147895444'),
(4, '4865423157'),
(5, '7512354894217'),
(6, '7825123786215')

SELECT * FROM tblBookAuthor
```

This is the first instance where a composite key is created. To create one of these, we create two foreign keys from the tables `tblBook` and `tblAuthor` and then join these together using `PRIMARY KEY (auAuthorID, bkISBN)`. This joins the two together to create a unique key.

### 3.7 tblLeasing

This table establishes three relationships, one with `tblCustomer`, `tblBookLeasing` and a junction table.

```
CREATE TABLE tblLeasing (
    lsLeasingID INT IDENTITY (1,1) PRIMARY KEY,
    cuCustomerID INT NOT NULL FOREIGN KEY REFERENCES tblCustomer(cuCustomerID),
    emEmployeeID INT NOT NULL FOREIGN KEY REFERENCES tblEmployee(emEmployeeID)
)

INSERT INTO tblLeasing
(cuCustomerID, emEmployeeID)
VALUES
()

SELECT * FROM tblLeasing
```

This table has some relatively complex transactions going on. To begin with, it is interacting with transactional and non-transactional tables, with transactions happening in a junction table.

### 3.8 tblBookLeasing

This is a transactional table that is a junction between `tblBook` and `tblLeasing`. It is responsible for documenting the checkout and return date

```
CREATE TABLE tblBookLeasing (
    bkISBN VARCHAR (13) FOREIGN KEY REFERENCES tblBook(bkISBN),
    lsLeasingID INT FOREIGN KEY REFERENCES tblLeasing(lsLeasingID),
    lsLeaseDate DATE NOT NULL,
    lsReturnDate DATE NOT NULL
    PRIMARY KEY (lsLeasingID, bkISBN)
)

INSERT INTO tblBookLeasing
(bkISBN, lsLeasingID, lsLeaseDate, lsReturnDate)
VALUES
(1235785642842, ),
(7512354894217),
(4865423157)
```

This table makes use of a composite key, the creation of which is described in 3.6.