

# Network Database Applications Assignment

## 1: Individual Activities

Gary Moore

November 16, 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 SQL 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 SQL 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 Tables

### 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 PhoneNumber,
7     cnEmail Email,
8     cnAddress1 VARCHAR(30) NOT NULL,
9     cnCity VARCHAR(30) NOT NULL,
10    cnCounty VARCHAR(30) NOT NULL,
11    cnPostcode Postcode
12 )
13
14 INSERT INTO tblContact
15 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty,
16  ↪ cnPostcode)
17 VALUES
18 ('Gary', 'Moore', '07500564955', 'madmangaz@gmail.com', '49 Woodvale Park', '
19  ↪ Dungannon', 'Co Tyrone', 'BT716DB'),
20 ('Steven', 'Stove', '02887723114', 'stovesteven@gmail.com', '70 Stove Road', '
21  ↪ Stoveland', 'Co Stove', 'BT115DF'),
22 ('Dolly', 'Donko', '07545652412', 'donkodolly@gmail.com', '32 Frillo Road', '
23  ↪ Dollyland', 'Co Donko', 'BT323FG'),
24 ('Willy', 'Wonko', '07548895621', 'wonkowilly@gmail.com', '49 Wonko Close', '
25  ↪ Wonkoworld', 'Co Willy', 'BT63BN'),
26 ('Henry', 'Hob', '07502314592', 'hobhenry@gmail.com', '21 Hob Lane', 'Henrytown
27  ↪ ', 'Co Hob', 'BT125NA'),
28 ('Nobby', 'Noodle', '07504562312', 'noodlenobby@gmail.com', '21 Noodle Way', '
29  ↪ Nobbycity', 'Co Noodle', 'BT426FN'),
30 ('Willy', 'Willard', '07501228543', 'willardwilly@gmail.com', '21 Garfield Lane
31  ↪ ', 'Grungetown', 'Co Tables', 'BT124DB'),
32 ('Shellie', 'Shirt', '07542655489', 'shirtshellie@gmail.com', '141 Dingdong
33  ↪ Avenue', 'Dingdongcity', 'Co Dole', 'BT126DF'),
34 ('Rupert', 'Rung', '02845621453', 'rungrupert@gmail.com', '12 Rung Street', '
35  ↪ Rungland', 'Co Ring', 'BT47FG'),
36 ('Berty', 'Balls', '07512545698', 'ballsberty@gmail.com', '5 Bumble Road', '
37  ↪ Bundo', 'Co Herk', 'BT436BN'),
38 ('Jingo', 'Jango', '07512459821', 'jangojingo@gmail.com', '9 Honk Road', 'Linda
39  ↪ City', 'Co Suit', 'BT235HN'),
40 ('Will', 'Wharg', '07512324582', 'whargwill@gmail.com', '99 Nine Lane', 'Niner
41  ↪ Valley', 'Co Number', 'BT654BA')
42
43 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.

## Result

	cnContactID	cnFirstName	cnSurname	cnPhone	cnEmail	cnAddress1	cnCity	cnCounty	cnPostcode
1	1	Gary	Moore	07500564955	madmangaz@gmail.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB
2	2	Steven	Stove	02887723114	stovesteven@gmail.com	70 Stove Road	Stoveland	Co Stove	BT115DF
3	3	Dolly	Donko	07545652412	donkodolly@gmail.com	32 Frillo Road	Dollyland	Co Donko	BT323FG
4	4	Willy	Wonko	07548895621	wonkowilly@gmail.com	49 Wonko Close	Wonkoworld	Co Willy	BT63BN
5	5	Henry	Hob	07502314592	hobhenry@gmail.com	21 Hob Lane	Henrytown	Co Hob	BT125NA
6	6	Nobby	Noodle	07504562312	noodlenobby@gmail.com	21 Noodle Way	Nobbycity	Co Noodle	BT426FN
7	7	Willy	Willard	07501228543	willardwilly@gmail.com	21 Garfield Lane	Grungetown	Co Tables	BT124DB
8	8	Shellie	Shirt	07542655489	shirtshellie@gmail.com	141 Dingdong Avenue	Dingdongcity	Co Dole	BT126DF
9	9	Rupert	Rung	02845621453	rungrupert@gmail.com	12 Rung Street	Rungland	Co Ring	BT47FG
10	10	Berty	Balls	07512545698	ballsberty@gmail.com	5 Bumble Road	Bundo	Co Herk	BT436BN
11	11	Jingo	Jango	07512459821	jangojingo@gmail.com	9 Honk Road	Linda City	Co Suit	BT235HN
12	12	Will	Wharg	07512324582	whargwill@gmail.com	99 Nine Lane	Niner Valley	Co Number	BT654BA

## 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 avoiding redundant data.

```

1 CREATE TABLE tblCustomer
2 (
3     cuCustomerID INT IDENTITY (1,1) PRIMARY KEY,
```

```

4      cnContactID INT NOT NULL 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`.

## Result

	cuCustomerID	cnContactID
1	1	8
2	2	9
3	3	10
4	4	11
5	5	12

## 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.

## Result

	emEmployeeID	emPassword	emPosition	emPermissions	cnContactID
1	1	password1	andy.admin	root	1
2	2	password2	billy.bob	user	2
3	3	password3	willy.wonka	user	3
4	4	password4	robert.rub	user	4
5	5	password5	henry.hob	user	5
6	6	password6	willard.wong	user	6
7	7	password7	guest	restricted	7

## 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.

## Result

	auAuthorID	auFirstName	auSurname
1	1	Arty	Author
2	2	Steven	Writer
3	3	Henry	Scribe
4	4	Bary	Book
5	5	Teny	Tree
6	6	Sabrina	Scroll

## 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.

## Result

	bkISBN	bkTitle	bkGenre
1	1235785642842	The Big Thing	SciFi
2	4216548789512	The Big One	Romance
3	4562147895444	Dingo the Dog	Childrens
4	4865423157	Back in my Day	Memoir
5	7512354894217	Blast Off from the Crescent Moon	Farty Tale
6	7825123786215	Dirty Dan	Spaghetti Western

### 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.

```
1 CREATE TABLE tblBookAuthor (  
2     auAuthorID INT NOT NULL FOREIGN KEY REFERENCES tblAuthor(auAuthorID),  
3     bkISBN VARCHAR(13) NOT NULL FOREIGN KEY REFERENCES tblBook(bkISBN),  
4     PRIMARY KEY (auAuthorID, bkISBN)  
5 )  
6  
7 INSERT INTO tblBookAuthor  
8 (auAuthorID, bkISBN)  
9 VALUES  
10 (1, '1235785642842'),  
11 (2, '4216548789512'),  
12 (3, '4562147895444'),  
13 (4, '4865423157'),  
14 (5, '7512354894217'),  
15 (6, '7825123786215')  
16  
17 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.

## Result

	auAuthorID	bkISBN
1	1	1235785642842
2	2	4216548789512
3	3	4562147895444
4	4	4865423157
5	5	7512354894217
6	6	7825123786215

### 3.7 tblLeasing

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

```
1 CREATE TABLE tblLeasing (  
2     lsLeasingID INT IDENTITY (1,1) PRIMARY KEY,  
3     cuCustomerID INT NOT NULL FOREIGN KEY REFERENCES tblCustomer(cuCustomerID),  
4     emEmployeeID INT NOT NULL FOREIGN KEY REFERENCES tblEmployee(emEmployeeID)  
5 )  
6  
7 INSERT INTO tblLeasing  
8 (cuCustomerID, emEmployeeID)  
9 VALUES  
10 (1, 4),  
11 (2, 4),  
12 (3, 2),  
13 (4, 3),  
14 (5, 5)  
15  
16 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.

#### Result

	lsLeasingID	cuCustomerID	emEmployeeID
1	1	1	4
2	2	2	4
3	3	3	2
4	4	4	3
5	5	5	5

### 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

```
1 CREATE TABLE tblBookLeasing (  
2     bkISBN VARCHAR (13) NOT NULL FOREIGN KEY REFERENCES tblBook(bkISBN),  
3     lsLeasingID INT NOT NULL FOREIGN KEY REFERENCES tblLeasing(lsLeasingID),  
4     lsLeaseDate DATE NOT NULL,  
5     lsReturnDate DATE NOT NULL,  
6     PRIMARY KEY (lsLeasingID, bkISBN)  
7 )  
8  
9 INSERT INTO tblBookLeasing
```



```

10 (bkISBN, lsLeasingID, lsLeaseDate, lsReturnDate)
11 VALUES
12 ('1235785642842', 1, '2018-04-01', '2018-04-08'),
13 ('7512354894217', 2, '2018-04-02', '2018-04-09'),
14 ('4865423157', 3, '2018-04-03', '2018-04-10'),
15 ('4865423157', 4, '2018-04-04', '2018-04-11'),
16 ('7825123786215', 5, '2018-04-05', '2018-04-12')
17
18 SELECT * FROM tblBookLeasing

```

This table makes use of a composite key, the creation of which is described in 3.6. `tblBookLeasing` is also the first use of the `DATE` data type, which is of course used for storing date information. When a customer decides to check out a book, the transaction will eventually make its way to this table, where the lease and return date are stored in `lsLeaseDate` and `lsReturnDate`.

## Result

	bkISBN	lsLeasingID	lsLeaseDate	lsReturnDate
1	1235785642842	1	2018-04-01	2018-04-08
2	7512354894217	2	2018-04-02	2018-04-09
3	4865423157	3	2018-04-03	2018-04-10
4	4865423157	4	2018-04-04	2018-04-11
5	7825123786215	5	2018-04-05	2018-04-12

## 3.9 Notable Transactions

### Adding a book

When an employee wants to add a book to the database, they will use a form that will inner join `tblAuthor`, `tblBook` and `tblBookAuthor`. This data is inserted into the table through this form.

### Removing a book

The process of removing a book involves some conditional logic. If the book is the only one in the system by that author, the author can be removed too. If the book has other books by the same author, the author will not be deleted. This can be done with an `IF...ELSE` statement.

### Leasing a book

The procedure for checking out a book uses a network of tables to carry out the transaction, again to make sure there's no data redundancy. The tables `tblCustomer`, `tblLeasing`, `tblEmployee`, `tblBook` and `tblBookLeasing` will be used to create a record of a lease with `tblContact` queried to get the customer number from any of the customers details.

## 4 Constraints

I will make use of constraints to ensure that only valid data is entered into the database. These constraints can be created at `CREATE TABLE`, or they can be appended to the column definitions using `ALTER TABLE`.

A constraint is a system in a database that applies limitations to what information can be input into a particular column in a table. An example of a constraint is `NOT NULL`, which doesn't allow the user to enter a null value into their table. Another example is `UNIQUE`, which only allows a unique value compared to other values entered into the same column to exist. Violation of a constraint will result in an error which is displayed in the SQL management software's console.

I've created constraints for `tblContact`; a constraint for email, postcodes and phone numbers.

```
1 ALTER TABLE tblContact
2 ADD
3 CONSTRAINT PhoneNumRule CHECK (cnPhone LIKE '
   ↳ [0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
4 CONSTRAINT EmailRule CHECK (cnEmail LIKE '%@[A-Z][A-Z][A-Z]' AND LEN(cnEmail)
   ↳ > 6),
5 CONSTRAINT PostcodeRule CHECK (cnPostcode LIKE '[A-Z][A-Z][0-9][0-9][0-9][A-Z][
   ↳ A-Z]' OR cnPostcode LIKE '[A-Z][A-Z][0-9][0-9][A-Z][A-Z]')
```

To add these constraints we need to alter the `tblContact` table and then use the `ADD` command followed by the `CONSTRAINT` keyword on a new line with the name that we'll use for the constraint. We use `CHECK` to make sure that input values abide by the defined rule in the parenthesis. A boolean value of true is generated if the value matches the pattern, allowing the database to add that value to the table.

### 4.1 Phone constraint

Check line 3 of the example in Chapter 4. The requirements for the phone constraint are quite simple. Only digits can be accepted, therefore I've used the `[0-9]` wildcard to denote only digits. I've used this 11 times as the wildcard only represents a single character.

#### Testing

In the first image I'm checking to see if the constraint works when I try to add a phone number with less than 11 digits. The constraint successfully prevents this from happening, throwing an error.

```
1 INSERT INTO tblContact
2 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3 VALUES
4 ('Gary', 'Moore', '07955', 'placeholder@placeholder.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone', 'BT716DB')
5
6 SELECT * FROM tblContact
7
```

100 %

Results Messages

Msg 547, Level 16, State 0, Line 1  
The INSERT statement conflicted with the CHECK constraint "PhoneNumRule". The conflict occurred in database "dbSqlAssign".  
The statement has been terminated.

(13 rows affected)

In the second image, I am trying to use letters as a phone number, however the constraint catches this, and throws out an error.

```
1 INSERT INTO tblContact
2 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3 VALUES
4 ('Gary', 'Moore', 'qwertyuiop[]', 'placeholder@placeholder.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone', 'BT716DB')
5
6 SELECT * FROM tblContact
7
```

100 %

Results Messages

Msg 547, Level 16, State 0, Line 1  
The INSERT statement conflicted with the CHECK constraint "PhoneNumRule". The conflict occurred in database "dbSqlAssign".  
The statement has been terminated.

In the third image, I pass a valid phone number to test if the constraint will allow valid data to be inserted into the table. The constraint passes the test.

```

1 INSERT INTO tblContact
2 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3 VALUES
4 ('Gary', 'Moore', '12345678901', 'placeholder@placeholder.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone',
5  'BT716DB')
6
7 SELECT * FROM tblContact

```

	cnContactID	cnFirstName	cnSurname	cnPhone	cnEmail	cnAddress1	cnCity	cnCounty	cnPostcode
1	1	Gary	Moore	07500564955	madmangaz@gmail.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB
2	2	Steven	Stove	02887723114	stovesteven@gmail.com	70 Stove Road	Stoveland	Co Stove	BT115DF
3	3	Dolly	Donko	07545652412	donkodolly@gmail.com	32 Frillo Road	Dollyland	Co Donko	BT323FG
4	4	Willy	Wonko	07548895621	wonkowilly@gmail.com	49 Wonko Close	Wonkoworld	Co Willy	BT63BN
5	5	Henry	Hob	07502314592	hobhenry@gmail.com	21 Hob Lane	Henrytown	Co Hob	BT125NA
6	6	Nobby	Noodle	07504562312	noodlenobby@gmail.com	21 Noodle Way	Nobbycity	Co Noodle	BT426FN
7	7	Willy	Willard	07501228543	willardwilly@gmail.com	21 Garfield Lane	Grungetown	Co Tables	BT124DB
8	8	Shellie	Shirt	07542655489	shirtshellie@gmail.com	141 Dingdong Avenue	Dingdongcity	Co Dole	BT126DF
9	9	Rupert	Rung	02845621453	rungrupert@gmail.com	12 Rung Street	Rungland	Co Ring	BT47FG
10	10	Berty	Balls	07512545698	ballsberty@gmail.com	5 Bumble Road	Bundo	Co Herk	BT436BN
11	11	Jingo	Jango	07512459821	jangojingo@gmail.com	9 Honk Road	Linda City	Co Suit	BT235HN
12	12	Will	Wharg	07512324582	whargwill@gmail.com	99 Nine Lane	Niner Valley	Co Number	BT654BA
13	15	Gary	Moore	07500564955	placeholder@placeholder.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB
14	18	Gary	Moore	12345678901	placeholder@placeholder.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB

## 4.2 Email constraint

Check line 4 of the example in Chapter 4. This check is slightly more complicated as i'm using the AND statement, which means values must pass two checks in order to be inserted into the table. First it will check if the correct email formatting is used, that is the requirement of `&`, `period`, `username`, `domain name` and `tld`. I don't want an email to look like `'&.com'`, and the `%` wildcard will allow any string with no characters up to exist there, so I have to do a second check that requires the length of the value to be greater than 6, which is the minimum for there to be information in the username and domain name. This isn't perfect however, as it could still look like this `'aa@.com'`. This constraint should however weed out the most troublesome data validation errors.

Another way to do data validation that I haven't looked at is to use a Regular Expression, however this requires integrating a CLR<sup>1</sup> function into the database solution.

## Testing

In the first image I am testing to see if validation works when the input is not long enough to pass. An error message is displayed and the data is not inserted into the table as it did not pass the test.

<sup>1</sup>Common Language Runtime

```
1 INSERT INTO tblContact
2 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3 VALUES
4 ('Gary', 'Moore', '07500564955', '@.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone', 'BT716DB')
5
6 SELECT * FROM tblContact
7
```

100 %

Results Messages

Msg 547, Level 16, State 0, Line 1  
The INSERT statement conflicted with the CHECK constraint "EmailRule". The conflict occurred in database "dbSqlAssignment".  
The statement has been terminated.

(12 rows affected)

In the second image I am testing to see if validation works when I only input a username without the , period, tld or domain name. The data fails the test and isn't inserted into the table.

```
1 INSERT INTO tblContact
2 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3 VALUES
4 ('Gary', 'Moore', '07500564955', 'placeholder', '49 Woodvale Park', 'Dungannon', 'Co Tyrone', 'BT716DB')
5
6 SELECT * FROM tblContact
7
```

0 %

Results Messages

Msg 547, Level 16, State 0, Line 1  
The INSERT statement conflicted with the CHECK constraint "EmailRule". The conflict occurred in database "dbSqlAssignment".  
The statement has been terminated.

(12 rows affected)

In the third image, I am testing valid data to ensure that it passes the test. It does, and the data is inserted into the table.

```

1 INSERT INTO tblContact
2 (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3 VALUES
4 ('Gary', 'Moore', '07500564955', 'placeholder@placeholder.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone',
5  'BT716DB')
6
7 SELECT * FROM tblContact

```

	cnContactID	cnFirstName	cnSurname	cnPhone	cnEmail	cnAddress1	cnCity	cnCounty	cnPostcode
1	1	Gary	Moore	07500564955	madmangaz@gmail.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB
2	2	Steven	Stove	02887723114	stovesteven@gmail.com	70 Stove Road	Stoveland	Co Stove	BT115DF
3	3	Dolly	Donko	07545652412	donkodolly@gmail.com	32 Frillo Road	Dollyland	Co Donko	BT323FG
4	4	Willy	Wonko	07548895621	wonkowilly@gmail.com	49 Wonko Close	Wonkoworld	Co Willy	BT63BN
5	5	Henry	Hob	07502314592	hobhenry@gmail.com	21 Hob Lane	Henrytown	Co Hob	BT125NA
6	6	Nobby	Noodle	07504562312	noodlenobby@gmail.com	21 Noodle Way	Nobbycity	Co Noodle	BT426FN
7	7	Willy	Willard	07501228543	willardwilly@gmail.com	21 Garfield Lane	Grungetown	Co Tables	BT124DB
8	8	Shellie	Shirt	07542655489	shirtshellie@gmail.com	141 Dingdong Avenue	Dingdongcity	Co Dole	BT126DF
9	9	Rupert	Rung	02845621453	rungrupert@gmail.com	12 Rung Street	Rungland	Co Ring	BT47FG
10	10	Berty	Balls	07512545698	ballsberty@gmail.com	5 Bumble Road	Bundo	Co Herk	BT436BN
11	11	Jingo	Jango	07512459821	jangojingo@gmail.com	9 Honk Road	Linda City	Co Suit	BT235HN
12	12	Will	Wharg	07512324582	whargwill@gmail.com	99 Nine Lane	Niner Valley	Co Number	BT654BA
13	15	Gary	Moore	07500564955	placeholder@placeholder.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB

### 4.3 Postcode constraint

Check line 5 of the example in Chapter 4. A postcode can be 6 or 7 characters long, and the typical format is [Char, Char, Num, Num, Num, Char, Char]. The constraint needs to ensure that this formatting is used, and also that it is of length 6 or 7.

In order to do this I used the wildcards [A-Z] and [0-9] where appropriate. I also used an OR statement that results in true if either one of the conditional statements is true.

### Testing

In the first image I am testing to ensure that the constraint rejects data that is too short to be a valid postcode. The constraint works as intended, and an error message is displayed with the data not being inserted into the table.

```

1  INSERT INTO tblContact
2  (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3  VALUES
4  ('Gary', 'Moore', '12345678901', 'placeholder@placeholder.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone',
5  'BT71')
6
7  SELECT * FROM tblContact

```

100 %

Results Messages

Msg 547, Level 16, State 0, Line 1  
The INSERT statement conflicted with the CHECK constraint "PostcodeRule". The conflict occurred in database "dbSqlAssign".  
The statement has been terminated.

(14 rows affected)

In the second image I am testing to ensure that a 6 character postcode get successfully inserted into the table. The test shows that the validation passed, and the data goes into the table.

```

1  INSERT INTO tblContact
2  (cnFirstName, cnSurname, cnPhone, cnEmail, cnAddress1, cnCity, cnCounty, cnPostcode)
3  VALUES
4  ('Gary', 'Moore', '12345678901', 'placeholder@placeholder.com', '49 Woodvale Park', 'Dungannon', 'Co Tyrone',
5  'BT76DB')
6
7  SELECT * FROM tblContact

```

100 %

Results Messages

	cnContactID	cnFirstName	cnSurname	cnPhone	cnEmail	cnAddress1	cnCity	cnCounty	cnPostcode
2	2	Steven	Stove	02887723114	stovesteven@gmail.com	70 Stove Road	Stoveland	Co Stove	BT115DF
3	3	Dolly	Donko	07545652412	donkodolly@gmail.com	32 Frillo Road	Dollyland	Co Donko	BT323FG
4	4	Willy	Wonko	07548895621	wonkowilly@gmail.com	49 Wonko Close	Wonkoworld	Co Willy	BT63BN
5	5	Henry	Hob	07502314592	hobhenry@gmail.com	21 Hob Lane	Henrytown	Co Hob	BT125NA
6	6	Nobby	Noodle	07504562312	noodlenobby@gmail.com	21 Noodle Way	Nobbycity	Co Noodle	BT426FN
7	7	Willy	Willard	07501228543	willardwilly@gmail.com	21 Garfield Lane	Grungetown	Co Tables	BT124DB
8	8	Shellie	Shirt	07542655489	shirtshellie@gmail.com	141 Dingdong Avenue	Dingdongcity	Co Dole	BT126DF
9	9	Rupert	Rung	02845621453	rungrupert@gmail.com	12 Rung Street	Rungland	Co Ring	BT47FG
10	10	Berty	Balls	07512545698	ballsberty@gmail.com	5 Bumble Road	Bundo	Co Herk	BT436BN
11	11	Jingo	Jango	07512459821	jangojingo@gmail.com	9 Honk Road	Linda City	Co Suit	BT235HN
12	12	Wharg	Wharg	07512324582	whargwill@gmail.com	99 Nine Lane	Niner Valley	Co Number	BT654BA
13	15	Gary	Moore	07500564955	placeholder@placeholder.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB
14	18	Gary	Moore	12345678901	placeholder@placeholder.com	49 Woodvale Park	Dungannon	Co Tyrone	BT716DB
15	20	Gary	Moore	12345678901	placeholder@placeholder.com	49 Woodvale Park	Dungannon	Co Tyrone	BT76DB

## 5 Locking

We use locking in databases to prevent multiple users from accessing data simultaneously. The reason this is a desirable feature is because if data is accessed at the same time it can have negative side effects such as data mutation. If someone alters information in a column, and another person does the same thing, one of these transactions may get overwritten, and it would be as if one of these transactions never happened. This would be very bad in for example a bank account, where real money could be lost as a result.