

SQL Fundamentals

LEARN TO DESIGN & BUILD A WINDOWS SQL
DATABASE

STEPHEN O'CONNOR

Database

What is a database?

A file structured for the repository of data. Organized for easy retrieval, sorting, grouping, relating to other data, used to analysis information in numerous ways.

Example: Customer database

To store Customer Data, use a simple text file, not easily manageable. Why not use an Excel spreadsheet file?
No easy way to relate sales data to customer information

Relational Database Management System RDBMS

Relational Database Theory – Organizing data into tables that can be related together, this reducing redundancy and increasing the integrity of the data.

Normalization – the process of determining what information belongs in which tables to minimize redundancy and increase integrity.

Database Objects

- Tables contain
 - Columns
 - Rows or records the value of a single column and a single row is a called a “field”

Data Types

Text varchar(100)

```
Create TABLE person (  
    Name varchar(100)  
);
```

Double pie

Exact number

Int or integer unsigned integer is a positive number.

```
CREATE TABLE person (  
    AGE INTEGER UNSIGNED // expected to be a positive value  
);
```

Decimal

Decimal(precision, scope)

Decimal (10, 0)

```
CREATE TABLE Student (  
    avg_score decimal(6, 2)
```

```
);
```

The is the default (10, 0)

1234567890

(5, 2)

123.45

(8, 7)

9.999999

```
CREATE TABLE student (  
    id integer auto_increment // given to primary keys  
)
```

date 'YYYY-MM-DD' // format of string ``

time '[H]HH:MM:SS

datetime 'YYYY-MM-DD HH:MM:SS'

```
CREATE TABLE order (  
    order_date date  
);
```

Booleans 0 is false non-0 or 1 is true

Boo

Boolean tinyint(1)

```
CREATE TABLE order (  
    fulfilled Boolean // true or false 0 or 1  
);
```

Data Integrity

Keeping data valid, of the correct data type, etc., so that it is usable for its intended purpose. Customers can be moved separate tables

What Needs to be Defined

- The data type for each column in a table
- The maximum size of data that will be stores in the column
- The nullability of a column

In the customer table the credit limit field is a real data type, this is so that the user has to enter a numerical value, so that calculations can be made using the numerical data. The real data type is approximate numerical. The customer since is a date time data type, only dates can be enter into this field. Null fields allow records to be saved to the database, to maintain data integrity.

Primary Key

A field or combination of fields that make a given row unique in the database. A way of differentiating each row in a table when all other fields might be duplicated in other rows in the same table. It can never be null, and must be set when the record is created and never changed.

CONSTRAINT

A rule that is enforced on the table.

CONSTRAINT check_id CHECK (customerID BETWEEN 0 and 10000)

Identity column an attribute that will be automatically increment a field of data in each successive row added to the database. Typically used on the primary key fields to make them unique.

Foreign key

Relate one or more rows in one table to a record in another table that shares the same value in its primary key. To check who made the order use customerID as a foreign key to relate the tables in the database. Data is linked, order is linked to the customer who created the order, i.e. query the database; how much a customer has spent. Adding a FK constraint prevents deletions in the customer table to create orphaned rows in the order table. FK constraints enforce "Referential Integrity".

Customer			
KEY	customerID	int (11)	
	firstName	varchar(50)	
	lastName	varchar(50)	
	address	varchar(200)	Null
	city	varchar(200)	Null
	county	char(10)	Null
	zip	char(10)	Null
	creditLimit	real	
	customerSince	datetime	

Order	
KEY	orderID
	orderDate
	orderAmount
	paymentType
FK	customerID

CRUD

CREATE, Read /SELECT, UPDATE, DELETE

CREATE - Tables are created using the CREATE TABLE statement

CREATE TABLE Customer (inside specify the table details, database schema);

SELECT - Read rows a table

SELECT firstName, lastName FROM Customer;

Or read all rows from a table

SELECT * FROM Customer;

DELETE - Delete rows from table

DELETE FROM Customer - deletes all rows from Customer table

UPDATE - Used to change data in existing rows in the table.

Specific the table after the keyword UPDATE Customer SET firstName = 'Stephen'

Selecting records**WHERE**

Reading data is the main thing you do with a database. WHERE can be used to filter records.

Only select a row where the given value is true.

SELECT * FROM Customer WHERE firstName = 'Stephen' // selects a single row from the database.

Also can select the column value

SELECT * FROM Customer WHERE CustomerID = 3;

SELECT * FROM Customers WHERE CustomerID != 3;

SELECT * FROM Customer WHERE lastName = CustomerID > 5;

More than one condition AND /OR keywords can chain where clauses

SELECT * FROM Customers WHERE CustomerID = 1 OR CustomerID = 2;

IN To specify multiple possible values for a column or to shorten the above statement

SELECT * FROM Customers WHERE CustomerID IN (1, 2); // a list of value in parenthesis

LIKE Search for a pattern % 0 or more of any other character

SELECT * FROM Customer WHERE firstName LIKE 'S%'

SELECT * FROM Customer WHERE lastName = 'OConnor' AND CustomerID > 5;

SELECT * FROM Customer WHERE lastName = 'OConnor' OR CustomerID > 5;

Comparsion		Logical	
=	Equals	ALL	NOT
>	Greater Than	AND	OR
<	Less Than	ANY	SOME
>=	Greater than or equal	BETWEEN	IN
<=	Less than or equal	EXISTS	LIKE
<>	Not Equal to	BETWEEN Between an inclusive range	
!=	Not Equal to		

Order Customers table sort data by date use ORDER BY. Can be used with ASC or DESC.

customerID	firstName	lastName	address	city	county	zip	creditLimit	customerSince
1	Ste	OConnor	Lucan	Dublin	Dublin	12	100	15/08/2002
2	John	Henry	Limetree	Limerick	Limerick	10	500	11/03/2004
3	Henry	Chinaski	Blossom	Tralee	Kerry	38	1000	02/11/2003
4	Harvey	Pecar	Cherrywood	Galway	Galway	3	75	24/07/2015
5	Brendan	Behan	Ringsend	Dublin	Dublin	11	5000	03/12/2009

SELECT * ORDER BY customerSince

customerID	firstName	lastName	address	city	county	zip	creditLimit	customerSince
1	Ste	OConnor	Lucan	Dublin	Dublin	12	100	15/08/2002
3	Henry	Chinaski	Blossom	Tralee	Kerry	38	1000	02/11/2003
2	John	Henry	Limetree	Limerick	Limerick	10	500	11/03/2004
5	Brendan	Behan	Ringsend	Dublin	Dublin	11	5000	03/12/2009
4	Harvey	Pecar	Cherrywood	Galway	Galway	3	75	24/07/2015

Group SELECT customerID, zip FROM Customers GROUP BY customerID

customerID	zip
1	12
2	10
3	38
4	3
5	11

Aggregate functions

Perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

Useful aggregate functions:

AVG() - Returns the average value.
 COUNT() - Returns the number of rows.
 FIRST() - Returns the first value.
 LAST() - Returns the last value.
 MAX() - Returns the largest value.
 MIN() - Returns the smallest value.
 SUM() - Returns the sum.

HAVING

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

LastName	NumberOfOrders
OConnor	11
Henry	27
Chinaski	14

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
WHERE LastName=OConnor OR LastName=Henry
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

LastName	NumberOfOrders
Henry	27

CAST

Converts an expression of one data type to another in SQL Server.

Syntax for CAST:

```
CAST ( expression AS data_type [ ( length ) ] )
```

Table Student_Score

Column Name	Data Type
StudentID	integer
First_Name	char(20)
Score	float

Table Student_Score Rows

StudentID	First_Name	Score
1	Jenny	85.2
2	Bob	92.5
3	Alice	90
4	James	120.1

Example 1

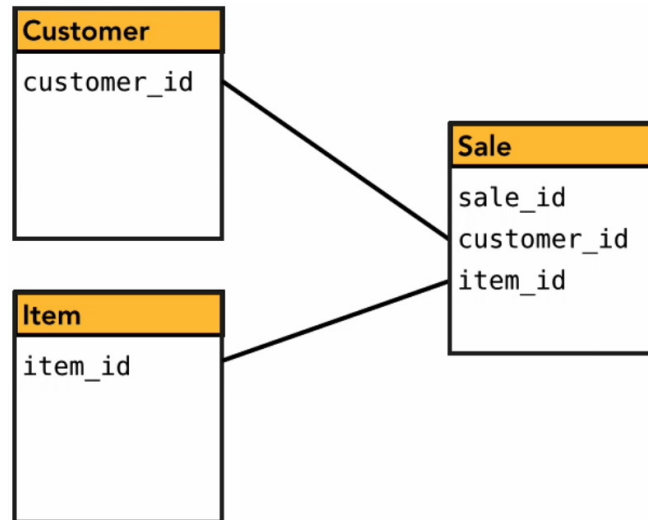
```
SELECT First_Name, CAST(Score AS Integer) Int_Score FROM Student_Score;
```

Result:

```
First_Name Int_Score
Jenny      85
Bob        92
Alice      90
James      120
```

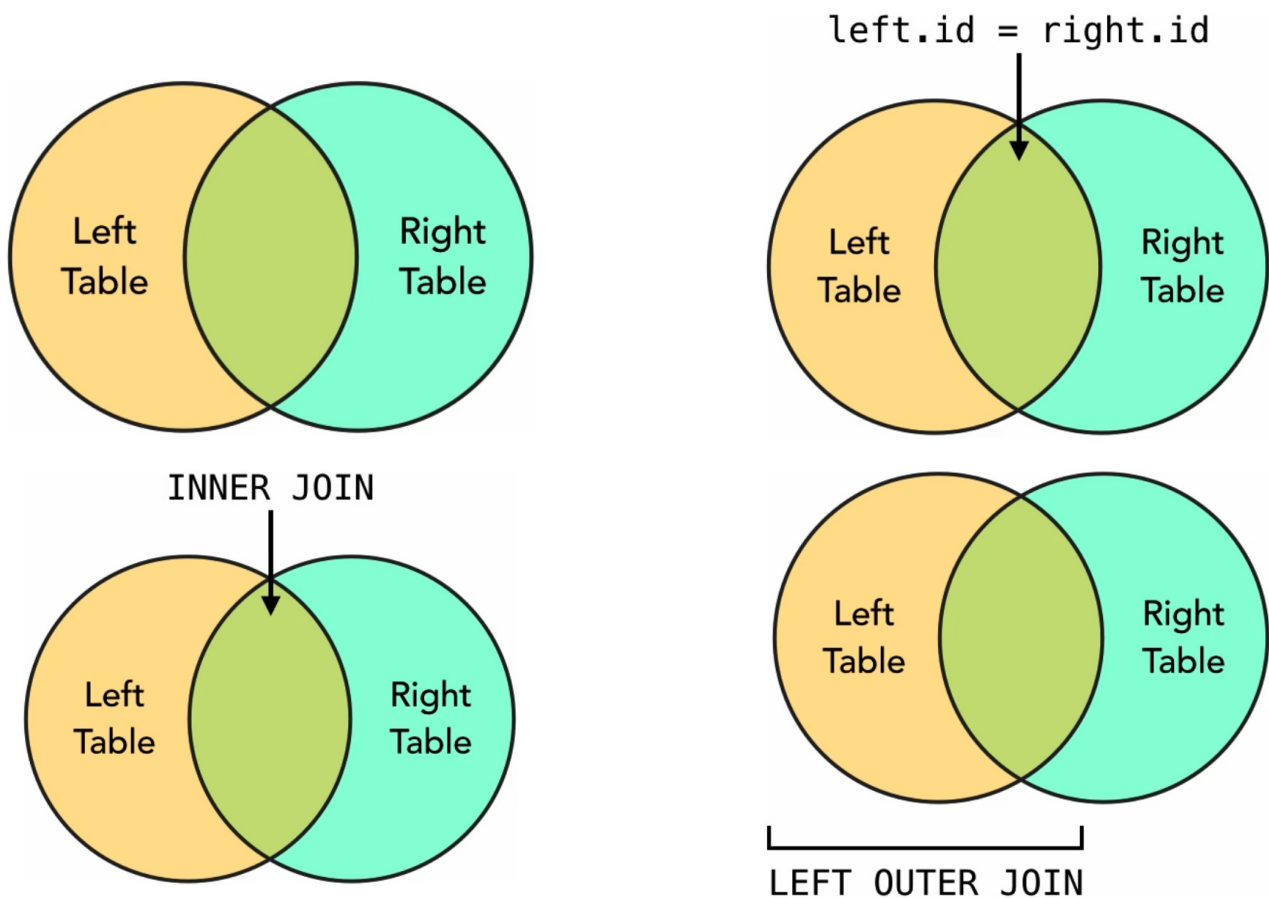
Table Relationships

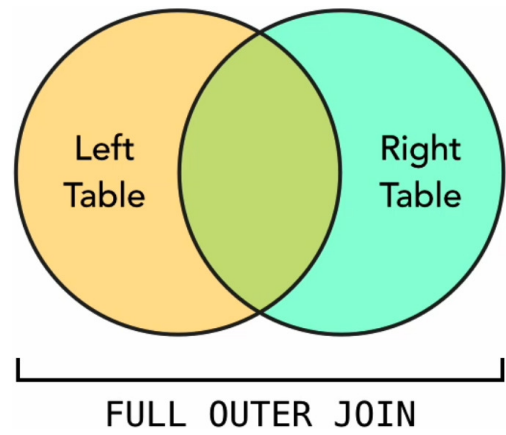
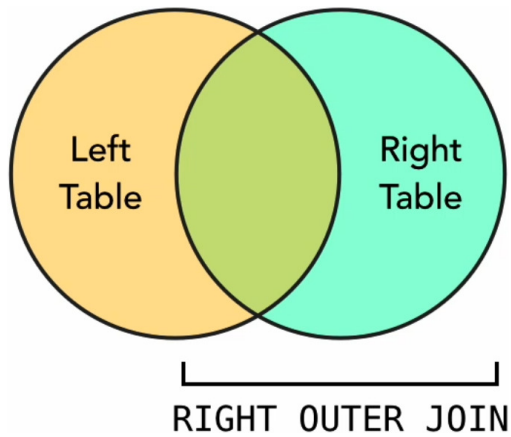
Some tables contain information related to other tables.



Join

Using the join statement, SQL has powerful tools for extracting related data from multiple tables. Typically unique id fields are used to create relationships. Unique id fields work well for creating and managing simple or complex relationships between tables. When a result is needed from multiple rows in multiple tables use a join query. The easiest way to understand joins is by using a venn diagram. Matching ids The simplest most common join is the inner join, this is the default join. Many databases including SQLite do not support a right join or full other join. Inter section of the tables, where the tables overlap a condition is met.





Exercise Joins

Create a new database at the following SQL

The screenshot shows the SQL Server Enterprise Manager interface on the left, displaying a database named "Database1.mdf" with tables "left1" and "right2". The main window shows the SQL Query editor with the following code:

```
1 CREATE TABLE left1 ( id INTEGER, description TEXT);
2 CREATE TABLE right2 ( id INTEGER, description TEXT);
3
4 INSERT INTO left1 VALUES ( 1, 'left1 01' );
5 INSERT INTO left1 VALUES ( 2, 'left1 02' );
6 INSERT INTO left1 VALUES ( 3, 'left1 03' );
7 INSERT INTO left1 VALUES ( 4, 'left1 04' );
8 INSERT INTO left1 VALUES ( 5, 'left1 05' );
9 INSERT INTO left1 VALUES ( 6, 'left1 06' );
10 INSERT INTO left1 VALUES ( 7, 'left1 07' );
11 INSERT INTO left1 VALUES ( 8, 'left1 08' );
12 INSERT INTO left1 VALUES ( 9, 'left1 09' );
13
14 INSERT INTO right2 VALUES ( 6, 'right2 06' );
15 INSERT INTO right2 VALUES ( 7, 'right2 07' );
16 INSERT INTO right2 VALUES ( 8, 'right2 08' );
17 INSERT INTO right2 VALUES ( 9, 'right2 09' );
18 INSERT INTO right2 VALUES ( 10, 'right2 10' );
19 INSERT INTO right2 VALUES ( 11, 'right2 11' );
20 INSERT INTO right2 VALUES ( 11, 'right2 12' );
21 INSERT INTO right2 VALUES ( 11, 'right2 13' );
22 INSERT INTO right2 VALUES ( 11, 'right2 14' );
```

Select both tables

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Server Explorer pane displays the database structure for 'Database1.mdf', including tables 'left1' and 'right2'. The main query editor window, titled 'SQLQuery1.sql', contains the following SQL code:

```
1 SELECT * FROM left1;  
2 SELECT * FROM right2;
```

The 'Results' tab is active, displaying two separate result grids. The first grid shows data from 'left1' with columns 'id' and 'description'.

id	description
1	left1 01
2	left1 02
3	left1 03
4	left1 04
5	left1 05
6	left1 06
7	left1 07
8	left1 08

The second grid shows data from 'right2' with columns 'id' and 'description'.

id	description
1	right2 06
2	right2 07
3	right2 08
4	right2 09
5	right2 10
6	right2 11
7	right2 12
8	right2 13
9	right2 14

Create a basic join. `ON l.id = r.id; // join on the rows that the condition is met`

The screenshot shows the same SQL Server Enterprise Manager interface. The query editor now contains the following SQL code:

```
1 SELECT l.description AS lefty, r.description AS righty  
2 FROM left1 AS l  
3 JOIN right2 AS r  
4 ON l.id = r.id;  
5  
6 SELECT * FROM left1;  
7 SELECT * FROM right2;
```

The 'Results' tab displays three result grids. The first grid shows the joined data with columns 'lefty' and 'righty'.

	lefty	righty
1	left1 06	right2 06
2	left1 07	right2 07
3	left1 08	right2 08
4	left1 09	right2 09

The second grid shows data from 'left1' with columns 'id' and 'description'.

id	description
1	left1 01
2	left1 02
3	left1 03
4	left1 04
5	left1 05
6	left1 06
7	left1 07
8	left1 08
9	left1 09

The third grid shows data from 'right2' with columns 'id' and 'description'.

id	description
1	right2 06
2	right2 07
3	right2 08
4	right2 09
5	right2 10
6	right2 11
7	right2 12
8	right2 13
9	right2 14

All the results from the left column and the inner join.

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Server Explorer' with a tree view of a database named 'Database1.mdf'. The tree view includes 'Tables', 'Views', 'Stored Procedures', 'Functions', 'Synonyms', 'Types', and 'Assemblies'. The 'Tables' folder is expanded, showing 'left1' and 'right2'. The 'right2' table is selected. The right pane shows the 'SQLQuery1.sql' file with the following query:

```
1 SELECT l.description AS lefty, r.description AS righty
2 FROM left1 AS l
3 LEFT JOIN right2 AS r
4 ON l.id = r.id;
5
6 SELECT * FROM left1;
7 SELECT * FROM right2;
```

The query is executed, and the results are displayed in the 'Results' pane. The results are organized into three sections:

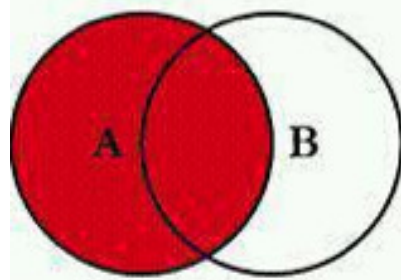
	lefty	righty
1	left1 01	NULL
2	left1 02	NULL
3	left1 03	NULL
4	left1 04	NULL
5	left1 05	NULL
6	left1 06	right2 06
7	left1 07	right2 07
8	left1 08	right2 08
9	left1 09	right2 09

	id	description
1	1	left1 01
2	2	left1 02
3	3	left1 03
4	4	left1 04
5	5	left1 05
6	6	left1 06
7	7	left1 07
8	8	left1 08
9	9	left1 09

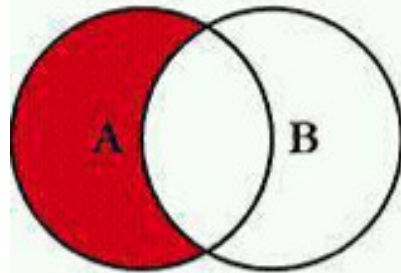
	id	description
1	6	right2 06
2	7	right2 07
3	8	right2 08
4	9	right2 09
5	10	right2 10
6	11	right2 11
7	11	right2 12
8	11	right2 13
9	11	right2 14

Join SELECT * FROM Orders INNER JOIN Customers;
SELECT firstName, lastName, orderAmount FROM Customers INNER JOIN Orders ON CustomerID = CustomerID;

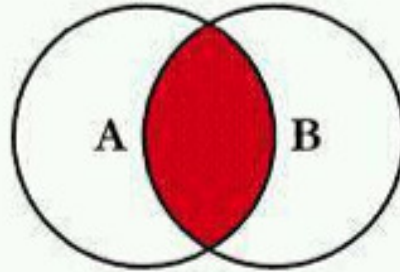
SQL JOINS



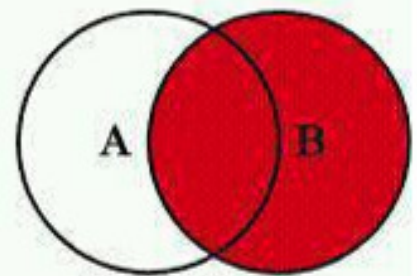
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



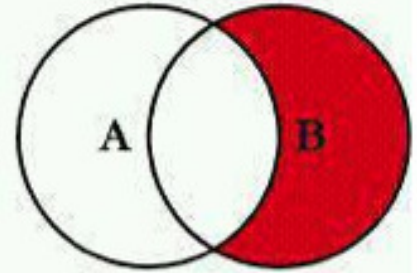
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL.
```



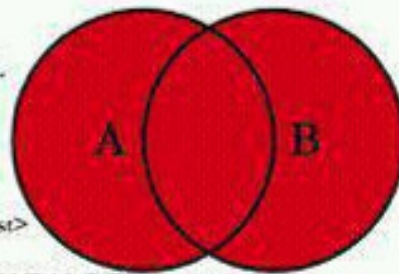
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



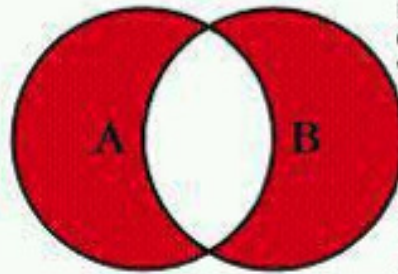
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL.
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL.
```

© C.L. McFatt, 2008

View

A view stores the select statement as a table.

```
CREATE VIEW employeeName AS
```

```
SELECT firstName, lastName, FROM employees
```

Select all from the view.

```
SELECT * FROM employeeName
```

GRANT statement

Use the GRANT statement to give privileges to a specific user or role, or to all users, to perform actions on database objects. You can also use the GRANT statement to grant a role to a user, to PUBLIC, or to another role.

Syntax

```
GRANT privilege-type ON [TABLE] { table-Name | view-Name } TO grantees
```

Example

To grant the SELECT privilege on table Managers to all users, use the following syntax:

```
GRANT SELECT ON TABLE Managers to PUBLIC
```

Download and PowerShell, run SQLite3.exe create new database.

```
sqlite>../sqlite3.exe PatsClothesShop.db
```

```
Customer      Order
```

```
sqlite> CREATE TABLE Customer(  
    costumerID INT PRIMARY KEY    NOT NULL,  
    firstName      CHAR(50) NOT NULL,  
    lastName       CHAR(50) NOT NULL,  
    address        VARCHAR(200),  
    city           CHAR(10),  
    county         CHAR(10),  
    creditLimit..... REAL,  
    costomerSince  DATETIME  
);
```

```
sqlite> CREATE TABLE Order(  
    orderID INT PRIMARY KEY    NOT NULL,  
    orderDate    DATETIME NOT NULL,  
    orderAmount  REAL,  
    paymentType  INT,  
    customerID   INT      NOT NULL  
);
```

```
sqlite>.tables
```

```
Customer      Order
```

```
sqlite>.header on  
sqlite>.mode column  
sqlite>.timer on
```

Insert 5 customers like below

```
INSERT INTO Customers (firstName, lastName, address, city, county, creditLimit,  
costomerSince)  
VALUES (1, 'Paul', 'Murphy' 32, 'Apt 1', 'Dublin', 15000.00, '2007-01-01 10:00:00' );
```

XML eXtensible Markup Language

Similar to HTML in it's use of the tags <></>

XML contains self-describing structured data but omits presentation (formatting) information.

XML is used to store or transfer data between disparate computing platforms, operating systems, software application, etc.

XML is an open standard, as opposed to a proprietary data format.

Common easily readable format for computer and humans, very portable.

XML used extensively in the .NET Framework.

XML's relationship to ADO.NET Datasets

Allows for a disconnected means of working with data, then allows for easy synchronization of changes back to the original data store.

- XML is used for syndicated RSS feeds
- XML is used to store or transfer data between disparate computing
- XML contains self-describing

Create an XML file called cars.

Create a root element /node **carcollection**, all XML documents have to have a root element.

Inside the **carcollection** element /node create two elements called car these are the sub elements /children of **carcollection**.

Inside the car elements create a sub-element called make.

It can one or the other but not both or it can contain an attribute.

Tag name are created by the programmer.

Pretty much free form, however follow a pattern to make it readable for others and yourself.

A need for conscience valid documents the structure of the XML and follows the rules of XML a well formed XML document.

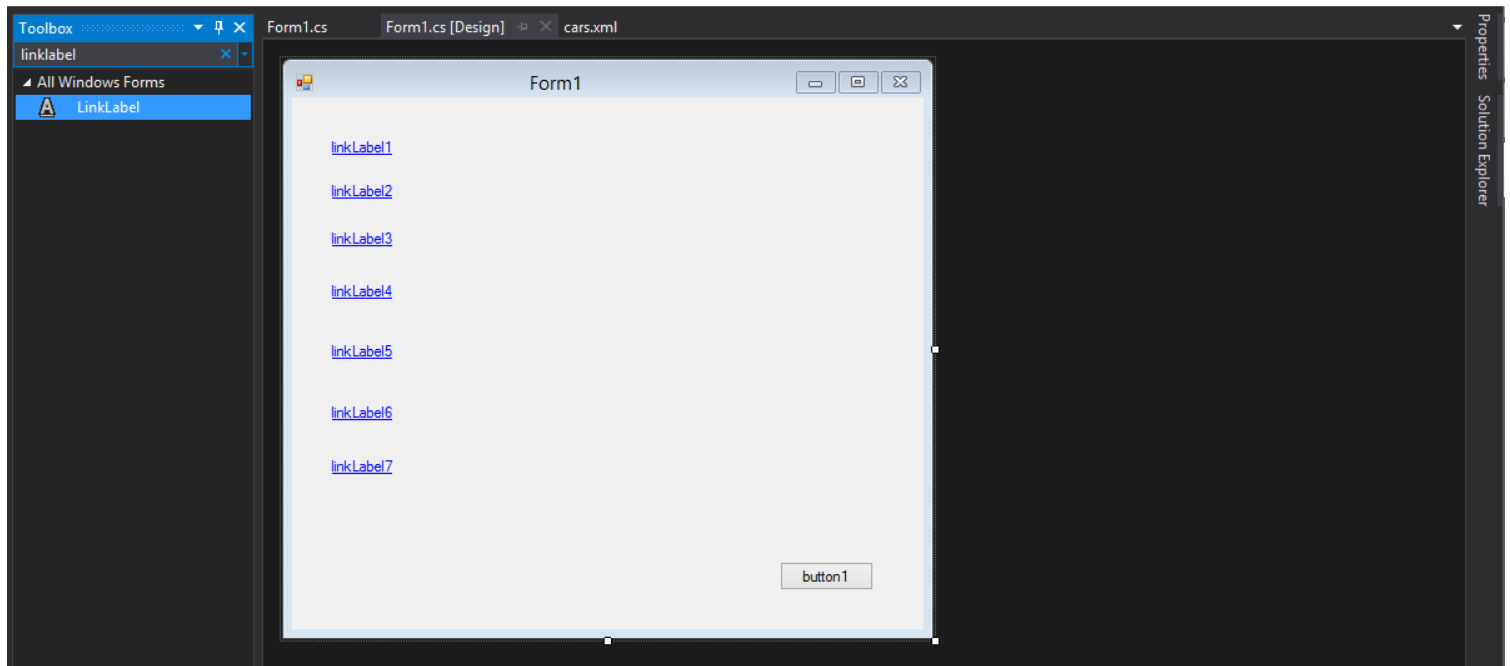
A valid document and well formed.

```
<?xml version="1.0" encoding="utf-8" ?>
<carcollection>
  <car>
    <make>DeLorean</make>
    <model>Time Machine</model>
    <year>1981</year>
  </car>
  <car>
    <make>Cadillac</make>
    <model>Ecto-1</model>
    <year>1984</year>
  </car>
</carcollection>
```

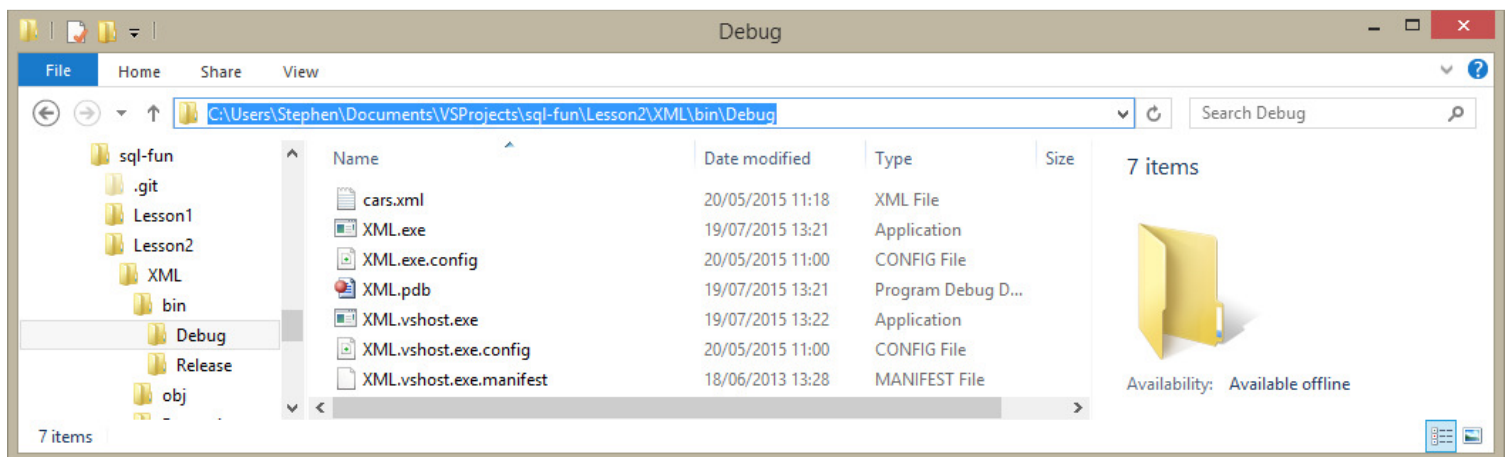
XML Schema Document

Using C# and .NET Framework Class Library to open and navigate through an XML file

Add seven linkLabels and a button



Save the XML document in the bin /debug folder



@"cars.xml" is the location of the XML file

```
// StreamReader will retrieve the file from the source
// and will convert it into a stream ready to be processed
System.IO.StreamReader sr = new System.IO.StreamReader(@"cars.xml");
```

```
// XmlTextReader
System.Xml.XmlTextReader xr = new System.Xml.XmlTextReader(sr);

// XmlDocument
System.Xml.XmlDocument carCollectionDoc = new System.Xml.XmlDocument();

carCollectionDoc.Load(xr);

// using the InnerText property will give us just the data
// ... since we are at the entire Document level, it will
// give us *all* the values (no delimiter).
linkLabel1.Text = carCollectionDoc.InnerText;
```

SQL Fundamentals

Useful links

[View Constraint](#)

[Aggregate Functions \(Transact-SQL\)](#)

[GRANT statement](#)

[SQL HAVING Clause](#)

[SQL Cast](#)

[SQL JOINS using SETS](#)

My Set-up

The image shows a developer's GitHub profile for Stephen J O'Connor (Stevo5o) and a PowerShell terminal window. The GitHub profile displays various statistics and repository information.

GitHub Profile: Stephen J O'Connor (Stevo5o)

- Freelance**, **Dublin**
- stev5o.joc@gmail.com**
- Joined on Jan 3, 2013**
- 21 Followers**, **25 Starred**, **13 Following**

Popular repositories:

- Software-Design-Fundamentals** (5 stars)
- D3-Charts** (4 stars)
- sql-fun** (3 stars)
- StephCakeLiveSearch** (2 stars)
- Angular-Fundamentals** (1 star)

Repositories contributed to:

- github/ghignore** (25,362 stars)
- atom/atom** (18,535 stars)
- marcpalmieri/CSharp-Basics** (0 stars)
- Glavin001/atom-beauty** (156 stars)
- tapio/live-server** (260 stars)

Contributions:

- Contributions in the last year:** 1,117 total (Jul 14, 2014 – Jul 14, 2015)
- Longest streak:** 184 days (November 22 – May 24)
- Current streak:** 0 days (Last contributed 14 hours ago)

Contribution activity: 12 commits (Period: 1 week)

Pushed 8 commits to Stevo5o/sql-fun Jul 6 – Jul 14

PowerShell Terminal:

```
You are now entering PowerShell : Stephen
PS C:\Users\Stephen\Documents\VSProjects\sql-fun> sql-fun master $ git add .
sql-fun master $ git commit -am 'Lesson Plan clean up'
[master b1489aa] Lesson Plan clean up
1 file changed, 0 insertions(+), 0 deletions(-)
sql-fun master $ git push origin master
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), 269.37 KiB | 0 bytes/s, done.
Writing objects: 100% (3/3), 269.37 KiB | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To https://github.com/Stevo5o/sql-fun.git
   beb0412..b1489aa master -> master
sql-fun master $
```