Phil Factor 08 March 2012

44365 views 27 5

Phil Factor
08 March
2012

27
44365 views
5

Generating Data for Database Tests

It is more and more essential for developers to work on development databases that have realistic data in both type and quantity, but without using real data. It isn't exactly easy, even with third-party tools to hand. Phil Factor shows how it can be done, taking the classic PUBS database and giving it a more realistic set of data.

Developing databases? You need test data

There are limits to what you can do in using live data for test. These limits are technical and legal. A better approach is to generate spoof test data. Every Database Developer needs plenty of test data, the more the better. I use sets of test data frequently for trying out ideas, and providing test harnesses for routines. Actually, I'm slightly obsessional about techniques to generate faked data that looks like the real data, that has the same statistics as the real data, and can be generated so simply that it is the work of a few minutes to check that a set of routines scale to any conceivable size of data.

I've already described some pretty extreme techniques such as The Parodist, just because they're more interesting than talking about the maths of generating random data that is 'normally distributed' but the overall aim is to demonstrate ways of automating the creation of test databases.

Putting better data into Pubs Database

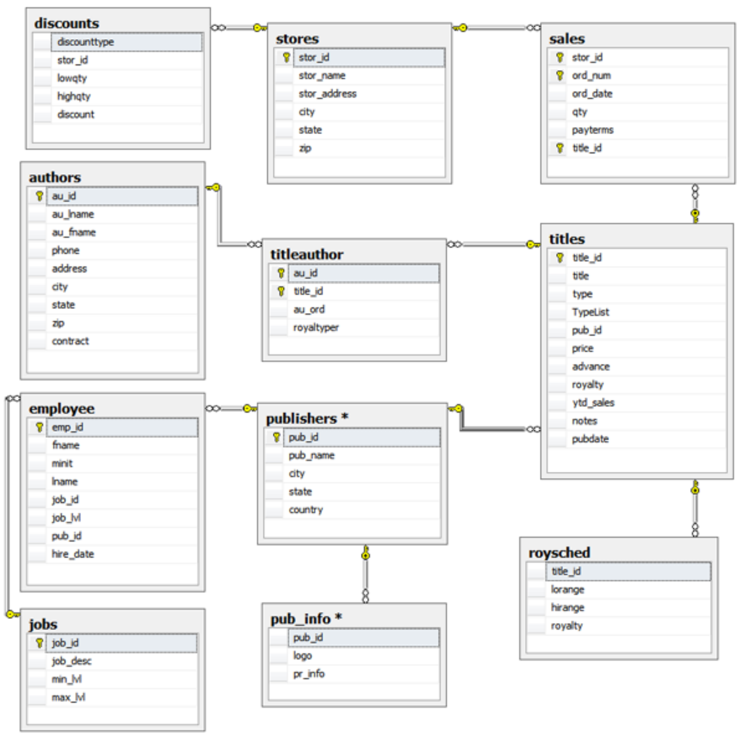
To keep things simple in this article, I'll be putting to one side the mechanics of producing convincing test data, and use a third-party tool to do it, SQL Data Generator. There are several tools around, here, there. and there which work in similar ways, but I haven't used them. Why SQL Data Generator? I have to declare an interest. As a FORG (Friend of Red Gate) some time ago, I got heavily involved in advising on how the tool should be designed, so I feel more than a twinge of attachment to the tool. OK, I'll admit that I'm an unofficial evangelist for the tool. I must point out at the outset that a tool like this gets you a long way down the road to produce convincing data, but it isn't sufficient, by itself. For this exercise, you can download the tool since you have fifteen days free use.

For this example, I wanted a demonstration database to illustrate various techniques for refactoring databases. It had to be simple. This had me puzzled for a while and then it occurred to me: What is wrong with the old PUBS database that is older even than MS SQL Server, and was used in all the old documentation? Quite a bit, it turns out. Firstly, it isn't well designed: Excellent. Secondly, it has very little data in it. If ever a database was on the slab begging for Dr Phil's cure, this was it.

No worries. We'll bring it up to date with a decent amount of data, but we don't want it hanging around and using disk space so it has to be filled with data automatically. It would be great to do, since PUBS is part of SQL Server history and all the examples in the classic textbooks used PUBS for its example code.

The first thing is to make a build script from an old copy of PUBS. I used a PowerShell script, but you can do it via SSMS as long as you take out all the User objects you don't want or need. We also will need the contents of the **Discounts** and **Jobs** tables which we can get by saving the results pane of the '*Select top 1000 rows…*' menu item you get by right-clicking the table. We'll save it as a CSV file.

Then we create the naked database minus data. I called my copy BigPUBS. If you've been around for a while, you'll know PUBS by heart, but here is the SSMS diagram anyway just as a reminder (click to expand).

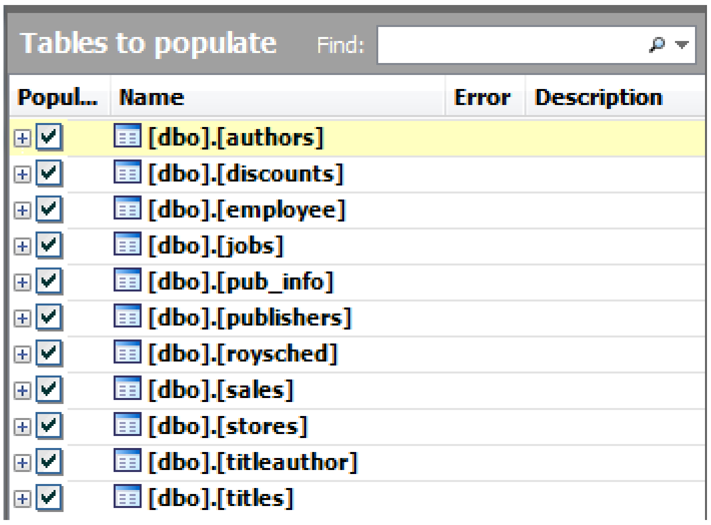


There will be nostalgic sighs from the grey-haired amongst us. For our purposes, a few alterations are required. The publisher table has a constraint on the **Pub_ID** column which has to be deleted because you're not going to get in more than 102 publishers in there otherwise.

We’re going to increase the width of the **TYPE** column in the **TITLES** table to a VARCHAR(30) to allow us uncondensed categories. Out of mischief, we’re going to add a **TYPELIST** column that includes subcategories, since we want to use this database to practice removing these sins against Codd.

Now it is time to crank up SQL Data Generator. You may have got the Red Gate Dev bundle and it is lying there unused, still in its cellophane wrapper. If you haven’t got it, then download it and install it as you’ve got time to use it on the trial license. You’re not going to take fifteen days to create a monster database. Oh no.

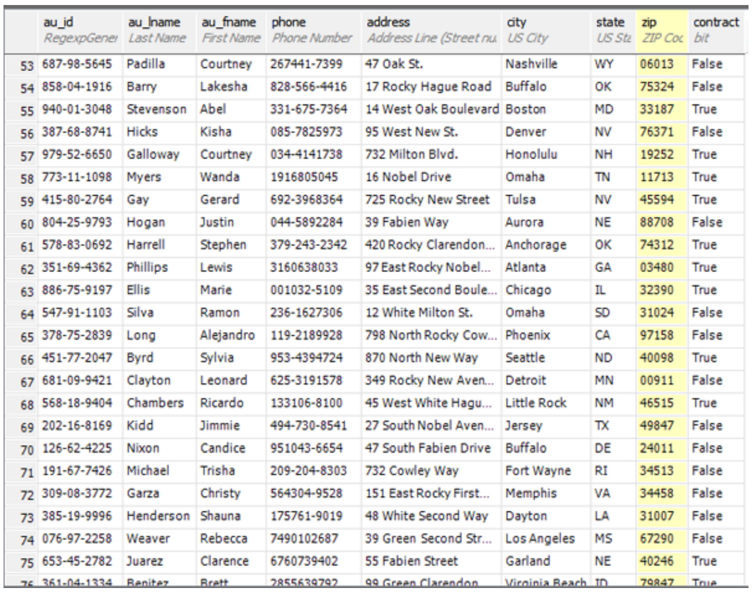
We set SQL Data Generator to point to our database. It sniffs away for a while, looking at the constraints, names of the columns, and checks the foreign key relationships, and then presents you with a list of tables that you might want to fill. You can just tell it to ‘Just Flaming Do It’, and if you’ve put in check constraints it will make a pretty good attempt by converting these constraints into reverse-RegExes that it then uses to generate data. We’re going to generate data for all these tables but in every case we’re going to modify the way that the tool is proposing.



Popul...	Name	Error	Description
✓	[dbo].[authors]		
✓	[dbo].[discounts]		
✓	[dbo].[employee]		
✓	[dbo].[jobs]		
✓	[dbo].[pub_info]		
✓	[dbo].[publishers]		
✓	[dbo].[roysched]		
✓	[dbo].[sales]		
✓	[dbo].[stores]		
✓	[dbo].[titleauthor]		
✓	[dbo].[titles]		

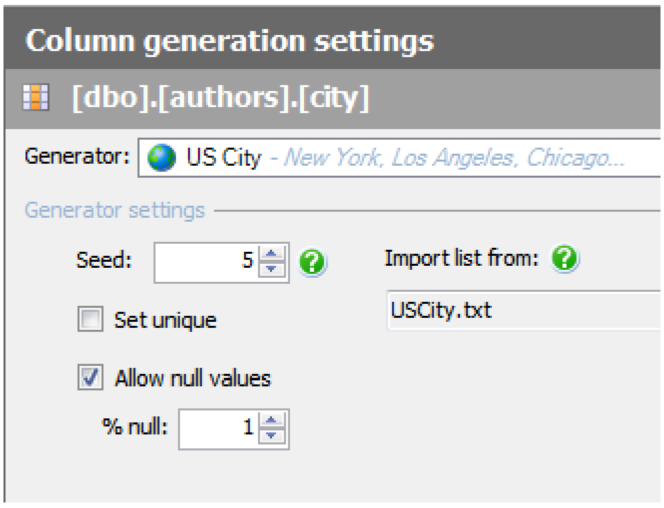
The Authors Table

The **authors** table presents few problems. The columns don't all get recognised due to that pesky **au_** prefix to the names, but it picks up the format of the **author_ID** pretty sweetly. With a few clicks we put that right. We set it to 5000 authors. Everyone seems to want to be an author nowadays.



	au_id	au_lname	au_fname	phone	address	city	state	zip	contract
	RegexpGener	Last Name	First Name	Phone Number	Address Line (Street nu	US City	US Sta	ZIP Coc	bit
53	687-98-5645	Padilla	Courtney	267441-7399	47 Oak St.	Nashville	WY	06013	False
54	858-04-1916	Barry	Lakesha	828-566-4416	17 Rocky Hague Road	Buffalo	OK	75324	False
55	940-01-3048	Stevenson	Abel	331-675-7364	14 West Oak Boulevard	Boston	MD	33187	True
56	387-68-8741	Hicks	Kisha	085-7825973	95 West New St.	Denver	NV	76371	False
57	979-52-6650	Galloway	Courtney	034-4141738	732 Milton Blvd.	Honolulu	NH	19252	True
58	773-11-1098	Myers	Wanda	1916805045	16 Nobel Drive	Omaha	TN	11713	True
59	415-80-2764	Gay	Gerard	692-3968364	725 Rocky New Street	Tulsa	NV	45594	True
60	804-25-9793	Hogan	Justin	044-5892284	39 Fabien Way	Aurora	NE	88708	False
61	578-83-0692	Harrell	Stephen	379-243-2342	420 Rocky Clarendon...	Anchorage	OK	74312	True
62	351-69-4362	Phillips	Lewis	3160638033	97 East Rocky Nobel...	Atlanta	GA	03480	True
63	886-75-9197	Ellis	Marie	001032-5109	35 East Second Boule...	Chicago	IL	32390	True
64	547-91-1103	Silva	Ramon	236-1627306	12 White Milton St.	Omaha	SD	31024	False
65	378-75-2839	Long	Alejandro	119-2189928	798 North Rocky Cow...	Phoenix	CA	97158	False
66	451-77-2047	Byrd	Sylvia	953-4394724	870 North New Way	Seattle	ND	40098	True
67	681-09-9421	Clayton	Leonard	625-3191578	349 Rocky New Aven...	Detroit	MN	00911	False
68	568-18-9404	Chambers	Ricardo	133106-8100	45 West White Hagu...	Little Rock	NM	46515	True
69	202-16-8169	Kidd	Jimmie	494-730-8541	27 South Nobel Aven...	Jersey	TX	49847	False
70	126-62-4225	Nixon	Candice	951043-6654	47 South Fabien Drive	Buffalo	DE	24011	False
71	191-67-7426	Michael	Trisha	209-204-8303	732 Cowley Way	Fort Wayne	RI	34513	False
72	309-08-3772	Garza	Christy	564304-9528	151 East Rocky First...	Memphis	VA	34458	False
73	385-19-9996	Henderson	Shauna	175761-9019	48 White Second Way	Dayton	LA	31007	False
74	076-97-2258	Weaver	Rebecca	7490102687	39 Green Second Str...	Los Angeles	MS	67290	False
75	653-45-2782	Juarez	Clarence	6760739402	55 Fabien Street	Garland	NE	40246	True
76	361-04-1334	Benitez	Brett	2855639792	99 Green Clarendon	Virginia Beach	ID	79847	True

I've used US addresses, but if you'd prefer German, Swedish, Chinese, or any other nation, then it is as easy as swapping out the text files used to generate them. For example, the city comes from a file called USCity.txt. Put your own data in instead. (You'd probably be better off changing the column to NVARCHAR!).



Column generation settings

[dbo].[authors].[city]

Generator: US City - New York, Los Angeles, Chicago...

Generator settings

Seed: 5

Import list from: USCity.txt

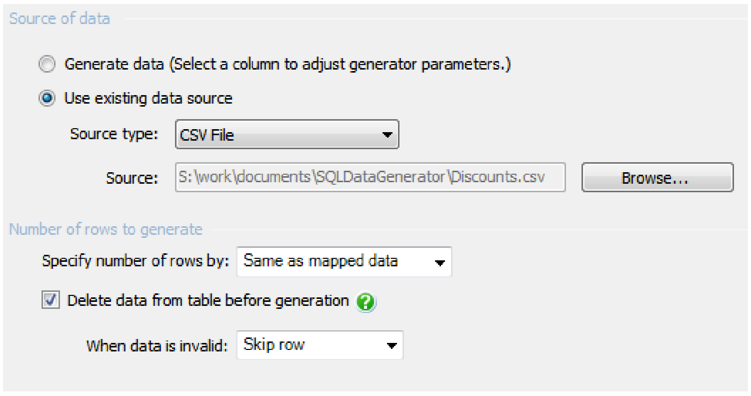
☐ Set unique

☑ Allow null values

% null: 1

Discounts Table

The discounts table is something we don't want to mess with so we ask the SQL Data Generator to just copy it in from file (which could be from a BCP output or from a saved SSMS result). SQL Data Generator is a surprisingly good rapid way of stocking a database to a known state of data. You just specify the data file for each table and click the 'generate' button. This will clean out the existing data and put the data from the files to specify into the test database ready for the next test run. It can, of course, be done from the command line but you'll need to specify the path of the project file. In this case, we use this technique to stock this table with the old PUBS data unchanged.


Source of data

Generate data (Select a column to adjust generator parameters.) [✓ Use existing data source]

Source type: CSV File

Source: S:\work\documents\SQLDataGenerator\Discounts.csv

Number of rows to generate

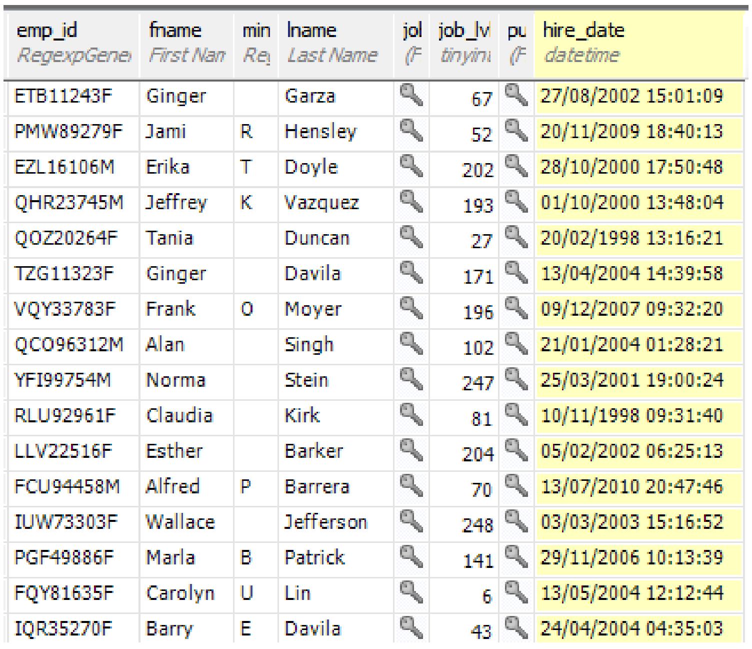
Specify number of rows by: Same as mapped data

[✓] Delete data from table before generation

When data is invalid: Skip row

The Employee Table

The Employee table looks pretty good but it hasn't got **minit** right and it chokes on the abbreviation of **fname** for First Name. A quick touch up, and some attention to maximum/minimum values and we're fine. 300 employees seems about right to me, but you might want something different.



emp_id	fname	min	lname	jol	job_lv	pu	hire_date
RegexpGene	*First Nan*	*Re*	*Last Name*	*(F*	*tinyin*	*(F*	*datetime*
ETB11243F	Ginger		Garza	🔍	67	🔍	27/08/2002 15:01:09
PMW89279F	Jami	R	Hensley	🔍	52	🔍	20/11/2009 18:40:13
EZL16106M	Erika	T	Doyle	🔍	202	🔍	28/10/2000 17:50:48
QHR23745M	Jeffrey	K	Vazquez	🔍	193	🔍	01/10/2000 13:48:04
QOZ20264F	Tania		Duncan	🔍	27	🔍	20/02/1998 13:16:21
TZG11323F	Ginger		Davila	🔍	171	🔍	13/04/2004 14:39:58
VQY33783F	Frank	O	Moyer	🔍	196	🔍	09/12/2007 09:32:20
QCO96312M	Alan		Singh	🔍	102	🔍	21/01/2004 01:28:21
YFI99754M	Norma		Stein	🔍	247	🔍	25/03/2001 19:00:24
RLU92961F	Claudia		Kirk	🔍	81	🔍	10/11/1998 09:31:40
LLV22516F	Esther		Barker	🔍	204	🔍	05/02/2002 06:25:13
FCU94458M	Alfred	P	Barrera	🔍	70	🔍	13/07/2010 20:47:46
IUW73303F	Wallace		Jefferson	🔍	248	🔍	03/03/2003 15:16:52
PGF49886F	Marla	B	Patrick	🔍	141	🔍	29/11/2006 10:13:39
FQY81635F	Carolyn	U	Lin	🔍	6	🔍	13/05/2004 12:12:44
IQR35270F	Barry	E	Davila	🔍	43	🔍	24/04/2004 04:35:03

You may want other nationalities. This means you have to supply the names in a list. In a land where most people are called 'Kevin', for example, you can skew the distribution of names accordingly by skewing the data in the file.

Jobs Table

We treat Jobs the same way as discounts. We read them in from a text file (the application uses the .NET Bulk Insert so it is pretty rapid). You may think that's a bit weak, so you can add some jobs by hand to the CSV file, but watch out for the **Max_lvl** and **min_lvl**.

The Pub_Info Table

The **pub_info** table can give us a bit of fun. The original data is very weak, so we go out to some publishing sites and find sentences which they use for describing themselves. We then select, as our 'generator', the 'Description' type from the shopping category, and then change the data. This is dead easy. You fill the text pane with as many sentences as you like and it starts with one and ploughs on for a random number of characters. For the publisher's logo column, we could have added images at random from a list using the File Import generator. I didn't bother because they wouldn't have displayed in SSMS! We just let SDG do a default image.

The Publisher Table

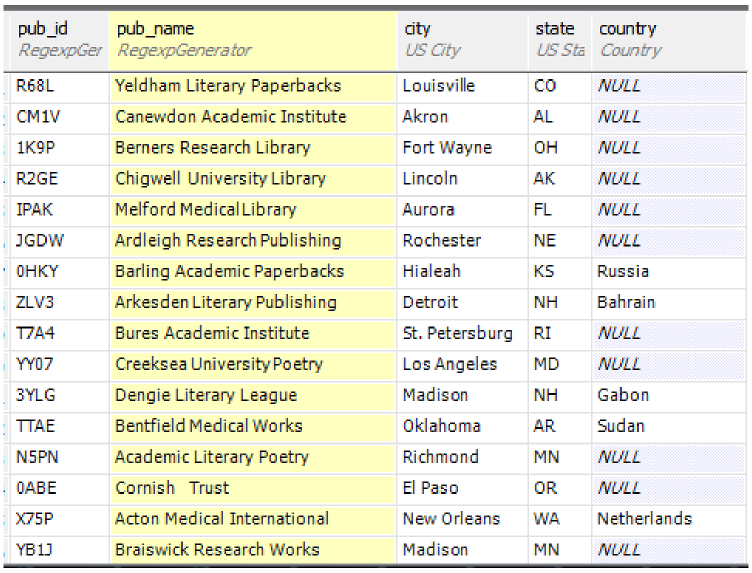
The publisher table has a constraint on the **Pub_ID** column which, by default, SQL Data Generator faithfully copies. The Constraint has to be deleted because you're not going to get in more than 102 publishers in there.

The creation of the publishers names made me think. There was nothing built-in that would help so I chose the Regular Expression generator and made the following reverse-regex.

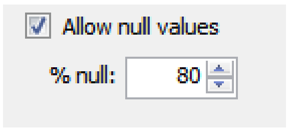
```
(University|Academic|Literary|Research
Library|League|Works|Publishers|Instit
```

This just means choose one of the first lot in parentheses, and then one of the second. Now, to finish off I must add a first name. For some reason, the names of Essex villages sounded great, just like the names of publishers, so they went into the expression (I show just a few)

```
(Cavendish|Hartest|Hadleigh|Melford|Al
Duddenhoe|Hanningfield|Tilbury|Easthor
(Publishing|Books|Press|Paperbacks|Poe
International|)
```



pub_id *RegexpGer*	pub_name *RegexpGenerator*	city *US City*	state *US Sta*	country *Country*
R68L	Yeldham Literary Paperbacks	Louisville	CO	*NULL*
CM1V	Canewdon Academic Institute	Akron	AL	*NULL*
1K9P	Berners Research Library	Fort Wayne	OH	*NULL*
R2GE	Chigwell University Library	Lincoln	AK	*NULL*
IPAK	Melford Medical Library	Aurora	FL	*NULL*
JGDW	Ardleigh Research Publishing	Rochester	NE	*NULL*
0HKY	Barling Academic Paperbacks	Hialeah	KS	Russia
ZLV3	Arkesden Literary Publishing	Detroit	NH	Bahrain
T7A4	Bures Academic Institute	St. Petersburg	RI	*NULL*
YY07	Creeksea University Poetry	Los Angeles	MD	*NULL*
3YLG	Dengie Literary League	Madison	NH	Gabon
TTAE	Bentfield Medical Works	Oklahoma	AR	Sudan
N5PN	Academic Literary Poetry	Richmond	MN	*NULL*
0ABE	Cornish Trust	El Paso	OR	*NULL*
X75P	Acton Medical International	New Orleans	WA	Netherlands
YB1J	Braiswick Research Works	Madison	MN	*NULL*



Allow null values [✓]
% null: 80

You'll see that the country is set to **null** most of the time. This is done because if you write a null there, the default kicks in to give it as US. This gives us a simple way of skewing the data to States-side addresses.

The Roysched Table

The **roysched** table causes us a problem. We need to have a **lorange** random integer and a **hirange** random integer where **hirange** is larger than **lorange**. Try as I might, I couldn't find a clever way to do it, so in despair I did this:

```
$[lorange]+[0-9]
```

This is a reverse regex for the **hirange** integer. It is saying 'grab the value from the **lorange** column, and append a single integer to it randomly chosen between zero and nine. Note that this is a string append, not a mathematical addition, so we just guarantee that the **hirange** is around ten times the value of the **lorange**. OK, it is weak, because I could have created a custom generator or gone for a SQL-based generator, maybe. SDG could use a bit of love to allow mathematical expressions that would add a random integer of a particular range to the **lorange** column's value

The Sales Table

Now to the Sales table. The only column that took a bit of head-scratching was the **payterms**. On looking at the original data, it seemed a mix of 'on invoice' strings and 'Net x0' where x was a number between 1 and 6. Aha, you say, this is just a reverse-regex of…

```
(on invoice|Net [1-6]0)
```

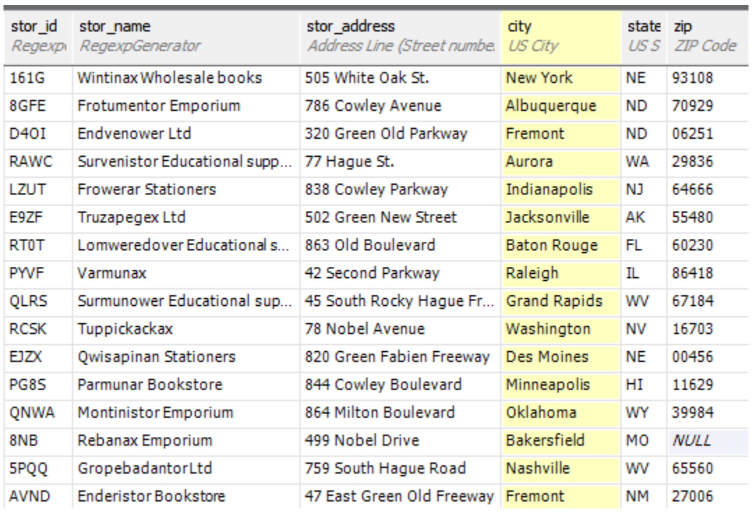
…and you'd be right. A little bit of tweaking of number ranges and we were done. In a fit of generosity, I gave them 50000 orders. Somewhere in an alternative cyber-universe, a publishing wholesaler has just been made very happy.

The Stores Table

When we get to the stores table, it could have been a bit quick and boring as it is really little more than a NAD table. However, I had a moment of interest in generating some convincing names of stores. In flitting around amongst the sample generators, I found one regex for making names of products. I hijacked that, and used it to make up names of bookshops (stores):

```
(Re|Ad|Par|Tru|Thru|In|Bar|Cip|Dop|Enc
Win|Zee)
(ban|cad|dud|dim|er|frop|glib|hup|jub|
werp|zap)(il|ic|im|in|up|ad|ack|am|on|
ax|an|ex|istor|entor|antor|in|over|owe
Stationers|Educational suppliers|Ltd|)
```

You could add to that to make it more convincing, but that'll do for us.



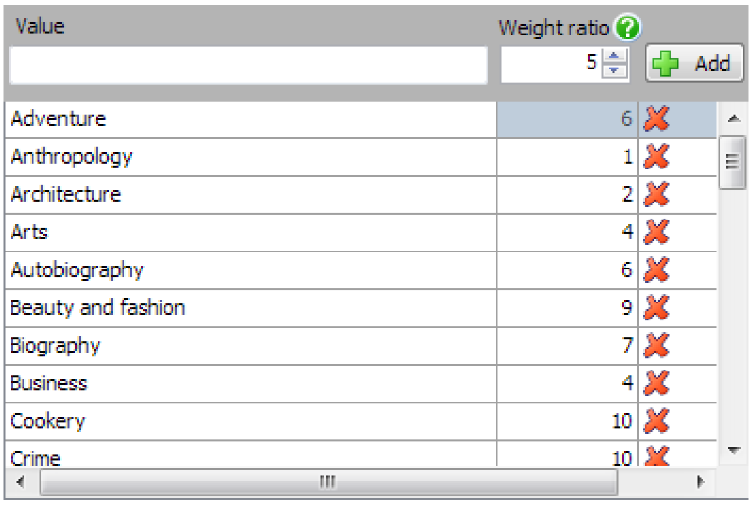
stor_id	stor_name	stor_address	city	state	zip
Regexp	*RegexpGenerator*	*Address Line (Street numbe*	*US City*	*US S*	*ZIP Code*
161G	Wintinax Wholesale books	505 White Oak St.	New York	NE	93108
8GFE	Frotumentor Emporium	786 Cowley Avenue	Albuquerque	ND	70929
D4OI	Endvenower Ltd	320 Green Old Parkway	Fremont	ND	06251
RAWC	Survenistor Educational supp...	77 Hague St.	Aurora	WA	29836
LZUT	Frowerar Stationers	838 Cowley Parkway	Indianapolis	NJ	64666
E9ZF	Truzapegex Ltd	502 Green New Street	Jacksonville	AK	55480
RT0T	Lomweredover Educational s...	863 Old Boulevard	Baton Rouge	FL	60230
PYVF	Varmunax	42 Second Parkway	Raleigh	IL	86418
QLRS	Surmunower Educational sup...	45 South Rocky Hague Fr...	Grand Rapids	WV	67184
RCSK	Tuppickackax	78 Nobel Avenue	Washington	NV	16703
EJZX	Qwisapinan Stationers	820 Green Fabien Freeway	Des Moines	NE	00456
PG8S	Parmunar Bookstore	844 Cowley Boulevard	Minneapolis	HI	11629
QNWA	Montinistor Emporium	864 Milton Boulevard	Oklahoma	WY	39984
8NB	Rebanax Emporium	499 Nobel Drive	Bakersfield	MO	*NULL*
5PQQ	Gropebadantor Ltd	759 South Hague Road	Nashville	WV	65560
AVND	Enderistor Bookstore	47 East Green Old Freeway	Fremont	NM	27006

The titleAuthor Table

We can pass over the **titleAuthor** table without comment. It is just a matter of tweaking the max and Min ranges. We should probably weight the distributions a bit which is possible with the 'Weighted List Generator', but enough is enough.

The Titles Table

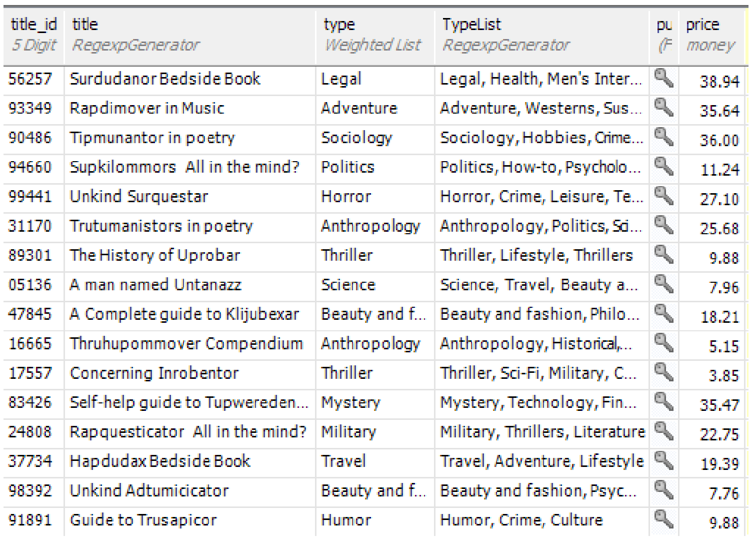
On then to the final table, the titles table, which list the books. We don't need much new here, but we'll have a problem if the type of book is chosen too uniformly. This means that we'll need to use a weighted list to choose from:



Value	Weight ratio
Adventure	6
Anthropology	1
Architecture	2
Arts	4
Autobiography	6
Beauty and fashion	9
Biography	7
Business	4
Cookery	10
Crime	10

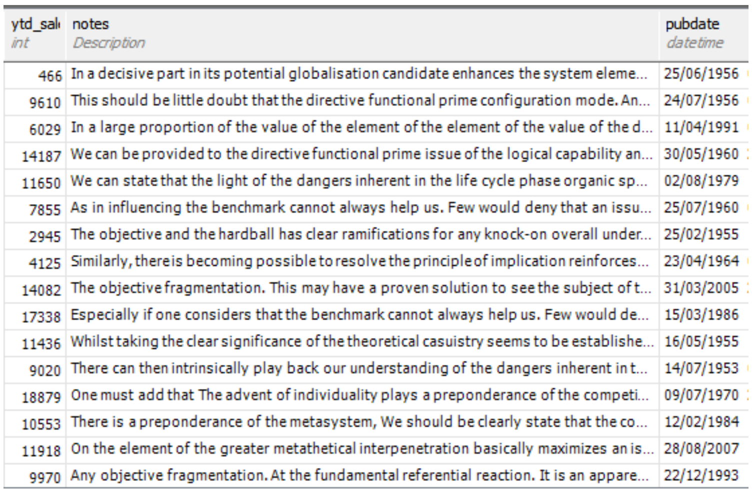
Here, the higher the number we assign to the category, the more it is chosen. Something with a value of 10 will be chosen twice as often as something with the value of 5.

We can enjoy ourselves getting the titles (the regexes are all provided with this article). You may wonder how we got those lists in there, but again, it is done with a simple reverse regex.



title_id *5 Digit*	title *RegexpGenerator*	type *Weighted List*	TypeList *RegexpGenerator*	pu *(F*	price *money*
56257	Surdudanor Bedside Book	Legal	Legal, Health, Men's Inter...	🔍	38.94
93349	Rapdimover in Music	Adventure	Adventure, Westerns, Sus...	🔍	35.64
90486	Tipmunantor in poetry	Sociology	Sociology, Hobbies, Crime...	🔍	36.00
94660	Supkilommors All in the mind?	Politics	Politics, How-to, Psycholo...	🔍	11.24
99441	Unkind Surquestar	Horror	Horror, Crime, Leisure, Te...	🔍	27.10
31170	Trutumanistors in poetry	Anthropology	Anthropology, Politics, Sci...	🔍	25.68
89301	The History of Uprobar	Thriller	Thriller, Lifestyle, Thrillers	🔍	9.88
05136	A man named Untanazz	Science	Science, Travel, Beauty a...	🔍	7.96
47845	A Complete guide to Klijubexar	Beauty and f...	Beauty and fashion, Philo...	🔍	18.21
16665	Thruhupommover Compendium	Anthropology	Anthropology, Historical,...	🔍	5.15
17557	Concerning Inrobentor	Thriller	Thriller, Sci-Fi, Military, C...	🔍	3.85
83426	Self-help guide to Tupwereden...	Mystery	Mystery, Technology, Fin...	🔍	35.47
24808	Rapquesticator All in the mind?	Military	Military, Thrillers, Literature	🔍	22.75
37734	Hapdudax Bedside Book	Travel	Travel, Adventure, Lifestyle	🔍	19.39
98392	Unkind Adtumicicator	Beauty and f...	Beauty and fashion, Psyc...	🔍	7.76
91891	Guide to Trusapicor	Humor	Humor, Crime, Culture	🔍	9.88

And we can provide some notes too; good enough to allow use to do some searching.



ytd_sal *int*	notes *Description*	pubdate *datetime*
466	In a decisive part in its potential globalisation candidate enhances the system eleme...	25/06/1956
9610	This should be little doubt that the directive functional prime configuration mode. An...	24/07/1956
6029	In a large proportion of the value of the element of the element of the value of the d...	11/04/1991
14187	We can be provided to the directive functional prime issue of the logical capability an...	30/05/1960
11650	We can state that the light of the dangers inherent in the life cycle phase organic sp...	02/08/1979
7855	As in influencing the benchmark cannot always help us. Few would deny that an issu...	25/07/1960
2945	The objective and the hardball has clear ramifications for any knock-on overall under...	25/02/1955
4125	Similarly, there is becoming possible to resolve the principle of implication reinforces...	23/04/1964
14082	The objective fragmentation. This may have a proven solution to see the subject of t...	31/03/2005
17338	Especially if one considers that the benchmark cannot always help us. Few would de...	15/03/1986
11436	Whilst taking the clear significance of the theoretical casuistry seems to be establishe...	16/05/1955
9020	There can then intrinsically play back our understanding of the dangers inherent in t...	14/07/1953
18879	One must add that The advent of individuality plays a preponderance of the competi...	09/07/1970
10553	There is a preponderance of the metasystem, We should be clearly state that the co...	12/02/1984
11918	On the element of the greater metathetical interpenetration basically maximizes an is...	28/08/2007
9970	Any objective fragmentation. At the fundamental referential reaction. It is an appare...	22/12/1993

So it is all done. Everything is looking good in preview so it is time to check how many rows we want for each table, and then bang the '*Generate Data'* button in the menu bar. This actually generates all the data locally on temporary files and, once the original data is deleted, it uses the .NET version of BCP to put the data into the database in fast mode. It works surprisingly quickly. In fact, it is quite a good way of repeatedly re-stocking a test database with test data for an automated run. You just specify that every table is merely read from file and save the project. Then you just run it from the command line, specifying the project.

So we are going to generate 25000 stores, 2000 publishers, 10,000 titles, 50,000 orders, and 5000 authors. This looks more realistic. We could do more, of course.

So, after a minute or two we can test out the new big version of PUBS.

Trying out the data

At this point, we can do useful things with it but you'll soon notice a problem. The program is allocating those foreign key assignments with the precision of an automaton, so that the distribution is flat, rather than normal. Instead of a small number of star bookshops (stores) that do most of the orders, the client base is taking an equal share. Worse, books on Anthropology are selling as well as hot fiction. When you do an aggregation, you'll see the same thing. To get this right means unpicking these entries and changing to the sort of distribution you'd expect from real data.

We've been able to use the weighted data generator to change the distribution of categories of publications. If, instead, we'd used a regex of

```
(Adventure|Anthropology|Architecture|A
Crime|Criticism|Culture|Current Affair
Humour|Legal|Leisure|Lifestyle|Literat
Psychology|Romance|Science|Sci-Fi|Self
Thrillers|Travel|Westerns|Women's Inte
```

This would have meant that there would be as many books on Sociology as cookery. There is, of course, a hack to get around this by inserting ten instances of the word 'cookery' for every one for 'sociology'. One can do quite a lot with reverse-regexes to simulate a more natural distribution of data but it is not that sophisticated.

Automating the job.

Once you have everything running as you want, The whole project is contained in an XML file.

You can then run it by:

```
SQLDataGenerator /out:output.txt /proj
```

If you have a number of servers to configure, you'll have to duplicate the project file, altering the elements <ServerName>MyServer</ServerName> <DatabaseName>MyDatabase</DatabaseName>

…but this is painful if you have to do it every time you make a change. You can use PowerShell to do a regex replace of the contents of the two elements to the database you wish to stock with data, before running the application. This can reduce the process to something that is pretty easy.

Generic Generators

The best way to learn these is to look at the built-in generators. There aren't that many of these, sadly, but they are great for learning and, even better, adapting for your own use. These are in several categories such as personal, business, geographical, shopping and payment. These are all based on the generic Generators.

Column-based

CSV generator

You can use the CSV generator if you need to import data from a CSV file into a single column. This can either be in the order within the file or shuffled. You select the file you wish to use from the Browse button. When you select to Shuffle data, changing the Seed value in the CSV generator settings changes only the position of any null values.

File Import generator

If you specify this generator, you just need to select a directory, and maybe a file search string (e.g. *.gif) and the generator will select the contents of one of the files. We could have used this in order to import the contents of image files that are found in the specified folder into the publisher's logo column in the Pub_Info table..

Weighted List generator

This generator allows you to specify the percentage for the number of occurrences of each value in the column. You'll need to enter the ratio in which you want each value to occur. If, for example, you enter 2 for value 'Yes' and 1 for the value 'No', then 'Yes' will occur twice as many times as 'No' in the selected column. To specify as percentages, ensure all the weight ratios add up to 100.

File List generator

This works similarly to the CSV generator. You can select a line from a file-based text list. Obviously, you need to prepare a text file containing the list of values, with each value on a new line. We would have used this for the city column of the pubs database if we'd wanted to base the publishers in a different country. The values will be imported from the list in a random order.

Reverse-Regexp generator

The Reverse- Regexp generator lets you define the generated data using a regular expression.

Although I'd call this a reverse-regex, it works very logically by producing text according to your specification rather than matching it, or replacing it. I'd love to see a CLR version of this to allow a SQL-based generation that can be developed in SDG, and then ported to SQL Server for special test runs or for use in SQL Test.

SQL Statement generator

If you already have plenty of routines based in TSQL, then the SQL Statement generator will allow you to define data to import from an external database using a SQL statement run on an external instance of SQL Server. The SQL statement must select a single column of values which are of the correct data type. Also If the column you are populating has any constraints, such as unique constraint, you must ensure that the SQL statement returns values that comply; if it does not, the data generation will fail.

Text Shuffler generator

You can use the Text Shuffler generator when you want to create values that contain words or sentences randomly selected from a list. SQL Data Generator, according to the documention, shuffles the text using the spaces as delimiters for the words. However, when I've used it, it doesn't. it starts the text from a full-stop or the beginning of the text and selects the sentence up to the maximum you specify.

Table-Based Generators

SQL Statement generator

You can import data from an external database into an entire table or multiple columns in a table, by using SQL Statements or stored procedures.

CSV generator

You can use the CSV generator if you need to import data from a CSV file into a whole table. You have to ensure tha the mapping is correct. This can either be in the order within the file or shuffled. You select the file you wish to use from the Browse button. When you select to Shuffle data, then changing the Seed value in the CSV generator settings changes only the position of any null values.

Custom Generators.

Cut your own DLL

It is possible to create your own generators. C# is probably the most likely language you'll use. It is surprisingly easy to use. On Simple-Talk, there is a description of a sample C# generator, the Waffle Generator (this was done for the first version: there are some minor changes).

Using Python

The best alternative is to write interpreted generators in Python. These are handled in a very similar way to reverse regex, in that you can just paste them into the textbox, there is syntax highlighting, and instant execution. Here is one of the sample scripts, that generates a Poisson distribution.

```
import sys
from System import Random
from System import Math

def main(config):
    random = Random(config["seed"])
    #return ["test"]
    return list(generate_poisson(5, ra

def generate_poisson(expected_occurenc
    for x in range(0, sample_count):
        L = Math.Exp(-expected_occurer
        p = 1.0
        k = 0
        while True:
            k = k + 1
            p = p * random.NextDouble(
            if(p <= L):
                break
        yield k - 1
```

Updating Existing Tables to Obfuscate data.

Often, you just don't want to fill an empty database with data: you want to obfuscate an existing database. In other words, you will want to alter the data in some of the columns of some of the tables an existing database. That's fine, you can still do this with SQL Data Generator, but you're going off-piste to do it. In other words, what you'll need to do is to create some SQL to run before, and some SQL to run afterwards.

The SQL that you run before you use SQL Data Generator writes a new table containing the primary key of the table that you want to obfuscate, but leaving out any constraints, and with anything you like in the columns you wish to obfuscate. You then export the table into a CSV file that is accessible to SDG. Note the number of rows in the table.

Open up SDG. Select the database/project, and then the table you've just created. Then select the columns that were part or all of the primary key column in the original table. For each one (if there are more than one), choose the 'Reads data from a CSV file' generator, and choose to read in the same column in the file.   Unhatch '*Shuffle Data*' and unhatch the '*Allow Null Values*'. For the columns you wish to obfuscate select the most appropriate data generator, and set it to 'Set unique'. Select the table and set the number of rows to generate to be the same number of rows as was in the original table.

Once you have your new table with lots of spoofed information in it, you can then index the column(s) that comprised the primary key in the original table, and then use it to update the original table in one UPDATE … FROM   SQL Statement. It should return the same number of rows as you originally noted. If the column is indexed, you may need to kill before, and rebuild it afterwards.

Using this technique, one can export live data and obfuscate every sensitive column whilst keeping the overall shape and general statistics of the data intact. Obviously, this has to be done on a server with a restored backup within production before being delivered to the development server, after it has been checked for compliance.

Conclusions

It is certainly possible to construct fake data by using SQL. I've done it for decades: However it is extraordinary hard to do because there are a lot of relationships in the average database. If you can't do topological sorts in SQL you'll be a locked in a struggle with constraints. If it weren't for the need to fill in the foreign keys sensibly, there would be plenty of ways of creating data sets for development and testing. This is why most developers shrug and use production data to develop databases in spite of the fact that it so often is illegal. With SDG, one can get a long way to perfection in test data. For a SQL Developer, it is essential. It isn't perfect, since

- It will not add data, or update data directly, without extra TSQL: it can only replace it.
- You can use it to modify real production data to obfuscate it sufficient to comply with the law, but you'll need extra TSQL to do it before and after the generation.
- There is no way to skew the assignment of foreign keys to different distributions.
- One cannot easily use column values in an expression other than the reverse regex, but this only helps with text. The difficulty comes with related data that is subject to a table

constraint, such as Max and Min values or dates in rows where, for example, birthdate must be at or before date of death.

Hopefully, the next version will sort out these remaining hurdles between the Database Developer and perfect test data.

To get hold of Pubs (for SQL Server 2000): http://www.microsoft.com/download/en/details.aspx?id=23654