Автоматическое генерирование тестовых данных

Теоретическая часть

Чтобы оценить эффективность организации сервера базы данных и приложения доступа к базе данных проводят мониторинг производительности. Основываясь на результатах мониторинга, можно перестроить приложение и настройки системы так, чтобы добиться максимальной эффективности и избежать возникновения *узких мест* (bottleneck). Для проведения анализа производительности необходимо располагать достаточно большим количеством записей в таблицах базы данных (10000 записей на таблицу и более). Создать большое количество правдоподобных данных вручную очень трудно. Для этих целей разрабатывают программы, которые автоматически генерирует относительно правдоподобные данные для каждой базовой таблицы.

В качестве примера рассмотрим относительно простое гипотетическое приложение, которое обрабатывает журнал «транзакций» системы электронной коммерции. Запись журнала будет состоять из следующих полей:

- шифр документа,
- дата,
- название счета,
- результат.

Гипотетический обработчик журнала транзакций ожидает ввода шифров документов в формате **AA9999/aaa** (т. е. две большие буквы, четыре цифры, слеш и три маленькие буквы, например «**BD4392/rjh**»). Для облегчения сортировки предлагается представлять даты в международном формате дат **yyyy/mm/dd** (например, «**2000/06/28**»). Затем требуется указать название счета, (так как это бизнес, то требуются настоящие деловые названия), а также результат транзакции – вещественное число с точностью до второго знака после запятой. Если транзакция состоит в расходовании, то результат должен быть отрицательным. В противном случае результат будет положительным.

При решении поставленной задачи помогут псевдослучайные числа. В языке С значения, которые дает функция **rand()**, основываются на начальной величине, которая запускает внутреннюю реализацию какого-нибудь алгоритма генерирования псевдослучайных чисел. Начальную величину можно выбрать самостоятельно или просто передать ее функции **srand()**. Если передается постоянное значение, то эта функция всегда возвращает тот же результат. Если передается случайное значение, то каждый раз будут получаться разные результаты. (Простой способ получения случайных значений имеет вид: **srand(time(NULL))** .)

Основной механизм для примера генератора тестовых данных передает функции **srand()** постоянное или случайное значение в зависимости от параметра командной строки. Здесь этот механизм не показан, но его можно найти на Web-сайте издательства «ДиаСофт» (www.diasoft.kiev.ua) в файле **datagen.c**.

Применим две функции **RandDbl** и **RandInt** для генерирования случайных чисел **R** двойной точности в диапазоне **0.0 ≤ R < 1.0** и целых чисел в диапазоне **0 ≤ R < N**. Иногда для получения случайных целых чисел в заданном диапазоне используют оператор **%** (например, **srand() % N**). Это приемлемо, если генератор случайных чисел, используемый в этой программе, действительно хороший. Большинство из них посредственны, так как не всегда числа случайны в разрядах битов нижних уровней настолько, насколько это требуется. Технология, показанная ниже, значительно лучше, поскольку располагает более значимые биты в разрядах более высокого уровня; многие коммерческие генераторы случайных чисел делают почти то же самое.

```
double RandDbl( void )
{
      return rand() / ((double)RAND_MAX + 1.0);
}

int RandInt( int n )
{
      return (int)(n * RandDbl());
}
```

Теперь рассмотрим основную функцию генерирования тестовых данных **GenerateData**. Следует обратить внимание на следующие моменты.

- Во-первых, здесь задается пара указателей на алфавит (один для больших и один для маленьких букв). Если система использует код ASCII, то можно получить случайную букву алфавита с использованием простого выражения, такого как **RandInt(26)+'A'**. В противном случае следует работать на системе, которая использует другие коды (например, EBCDIC). Указанная здесь технология применяется для всех кодовых таблиц (но не во всех языках; для простоты используйте обычный 26-буквенный английский алфавит).

- Чтобы ввод исходных данных сделать нагляднее и проще, можно задать пару массивов для хранения двух элементов типичного названия компании. Реальная реализация программы, возможно, будет считывать данные из файла, и они будут иметь сравнительно более широкий диапазон. Технология при этом останется той же – мы комбинируем два этих элемента наугад для получёния приемлемого разброса названий компаний.
- Чтобы получить правдоподобный диапазон варьирования дебетов и кредитов, выбирается случайное число и вычитается из него меньшее число. Таким образом, можно быть абсолютно уверенным, что одни числа будут положительными, а другие отрицательными. Однако за короткий промежуток времени маловероятно получить **0,00**; этот вид исключительного тестирования следует добавить вручную.
- Периодически дату нужно изменять. Выберем увеличение даты на **1** на основе результата подбрасывания монеты **(RandInt(2) == 0)**, но можно и наугад добавить **N** дней. Можно даже для проверки правильности подпрограмм сортировки сделать так, чтобы дата изменялась в обратную сторону.
- Технология изменения даты достаточно проста, но довольно интересна, потому что она позволяет использовать функцию **mktime()**, которая конвертирует **struct tm** в переменную **time_t**, которая имеет очевидный смысл. Интересно то, что при конвертировании функция **mktime()** упорядочивает данные, следовательно, специально это делать не придется. Нужно просто добавить **l** к числу дней и позволить функции **mktime()** рассортировать те случаи, в которых при добавлении единички к числу дней полученное число превышает нормальное количество дней в рассматриваемом месяце.

Замечание. На некоторых системах переменная **time_t** представлена внутренне как l**ong int**, при этом различие между двумя последующими значениями **time_t** составляет одну секунду и это значение составляет время с даты примерно 30-летней давности (вполне возможно, с 1 января 1970 года). (Стандарт ANSI не утверждает, что это так, но и не запрещает этого.) На таких системах представляемые переменной **time_t** дни буквально сочтены. Такого рода реализация с использованием начальной даты 1/1/1970 будет приводить к переполнению переменной **time_t** в январе 2038 года. При использовании такой системы, если делать достаточно записей с помощью функции **GenerateData**, можно получить довольно странные результаты. Этот момент надо уточнить для используемого Вами компилятора.

```
int GenerateData( FILE *fp, size_t MaxRecs )
{
      char *Lower = "abcdefghijklmnopqrstuvwxyz";
      char *Upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

      char *Name[] =
      {
            "Arrow", "Bizzare",
            "Complete", "Drastic",
            "Eagle", "Fiddleyew",
            "Gilbert", "Havago",
            "Ingenious", "J Random Loser",
            "Kludgit & Runn", "Lightheart",
            "Mouse", "Neurosis",
            "Objective", "Paradigm",
            "Quality", "Runaway",
            "Systemic", "Terrible",
            "Underwater", "Value",
            "Wannabee", "YesWeWill"
      };

      char *Business[] =
      {
            "Advertising", "Building",
            "Computers", "Deliveries",
            "Engineering", "Foam Packing",
            "Garage", "Hotels",
            "Industries", "Janitorial",
            "Knitwear", "Lazer Printers",
            "Mills", "Notaries",
            "Office Cleaning", "Printers",
            "Questionnaires", "Radio",
            "Systems", "Talismans",
            "Upholstery", "Van Hire",
            "Waste Disposal", "Yo-yos"
      };

      size_t NumNames = sizeof Name / sizeof Name[0];
      size_t NumBuses = sizeof Business / sizeof Business[0];
```

```
	time_t date_time_t;
	struct tm date = {0};
	struct tm *pd;
	size_t ThisRec;

	date_time_t = time( NULL );
	pd = localtime( &date_time_t );
	if ( pd != NULL )
	{
		memcpy( &date, pd, sizeof date );
	}
	else
	{
		date.tm_mday = 1;
		date.tm_mon = 0;
		date.tm_year = 100;
	}

	fprintf( fp, "Reference,Date,Account,Amount\n" );

	for( ThisRec = 0; ThisRec < MaxRecs; ThisRec++ )
	{
		fprintf( fp, "%c%c%04d/%c%c%c,",
			Upper[RandInt(26)],
			Upper[RandInt(26)],
			RandInt(10000),
			Lower[RandInt(26)],
			Lower[RandInt(26)],
			Lower[RandInt(26)] );

		fprintf( fp, "%d/%02d/%02d,",
			date.tm_year + 1900, /* NB: NOT a Y2K bug! */
			date.tm_mon + 1,
			date.tm_mday );

		fprintf( fp, "%s ",
			Name[RandInt(NumNames)] );

		fprintf( fp, "%s Ltd,",
			Business[RandInt(NumBuses)] );

		fprintf( fp, "%.2f\n",
			(RandDbl() * 5000.0) - 2250.0 );

		if ( RandInt(2) == 0 )
		{
			++date.tm_mday;
			date_time_t = mktime( &date );
			if ( date_time_t != (time_t) -1 )
			{
				pd = localtime( &date_time_t );
				if ( pd != NULL )
				{
					memcpy( &date, pd, sizeof date );
				}
			}
		}
	}
	return 0;
}
```

Замечание. Функция будет успешно компилироваться при наличии следующих директив включения:

```
#include <stdio.h>
#include <time.h>
```

```
#include <stdlib.h>
#include <memory.h>
```

Практическая часть

Для предложенного варианта объявления базовой таблицы написать программу, которая автоматически генерирует относительно правдоподобные данные. Программа может быть написана на языках C, C++, Java, C#.
Программа должна представлять собой консольное приложение и запускаться с командной строки. Результаты программы выводятся в выходной файл `1.out`. Допускается чтение вспомогательных данных из входного файла `1.in`. Значения полей записей в выходном файле должны отделяться друг от друга символами, которые допустимы в качестве разделителей полей при импорте данных. (В рассмотренном примере в качестве разделителя полей использовалась запятая. Следует отдать предпочтение символу табуляции.) Показать, что разработанная программа может генерировать 10000 записей.

Литература

1. Искусство программирования на C. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: Пер. с англ./Ричард Хэзфилд, Лоуренс Кирби и др. – К.: Издательство «ДиаСофт», 2001.