TableGen.cpp

```
// TableGen.cpp - TABLE LOAD PROFILE DATABASE GENERATOR
//
// MODULE INDEX
// NAME                    CONTENTS
// GenRand                 Generate a random number in a given range
// GenTableNo              Generate table number
// GenCovers               Generate group size
// GenWaiterCode           Generate waiter code
// GenRefCode              Generate reference code
// ReleaseTables           Release unoccupied tables
// main                    Main line
//
// MAINTENANCE HISTORY
// DATE                    PROGRAMMER AND DETAILS
// 31-08-08         JS     Original
//
//-----------------------------------------------------------------------------

#include <cstring>         // C-style string manipulation functions
#include <iostream>        // C++ input/output streams
#include <iomanip>         // C++ input/output manipulators
#include <sstream>         // C++ string stream declarations
#include <vector>          // C++ vector declarations
#include <map>             // C++ map declarations
using namespace std;       // Expand the standard namespace
#include "dtime.h"         // Double time declarations
exec sql include sqlca;    // Include SQL communications area

//-----------------------------------------------------------------------------

// DEFINITIONS

static const long   MIN_SES_DUR = 15*60;
                           // Minimum session duration (seconds)
static const long   MAX_SES_DUR = 120*60;
                           // Maximum session duration (seconds)
static const char   FIRST_DATE[] = "2008-01-01";
                           // First date
static const char   LAST_DATE[] = "2008-12-31";
                           // Last date
static const double FIRST_HOUR = 11.0*60.0*60.0;
                           // Time of first hour
static const size_t GROUP_SIZE_CNT = 4;

//-----------------------------------------------------------------------------

// GROUP SIZE LIMITS

static const int MIN_GROUP_SIZE[] = {1, 3, 5, 7};
                           // Minimum group sizes
static const int MAX_GROUP_SIZE[] = {2, 5, 6, 12};
                           // Maximum group sizes

//-----------------------------------------------------------------------------

// AVERAGE NUMBER OF SESSIONS STARTED PER GROUP SIZE PER HOURLY SEGMENT

static const double AVG_NO_SES[][GROUP_SIZE_CNT] = {
       {1.0,  0.5,   0.3,   0.2},
       {5.0,  3.0,   1.0,   0.5},
       {12.0, 8.0,   2.0,   0.5},
       {6.0,  3.0,   2.0,   0.7},
       {3.0,  1.0,   0.5,   0.3},
       {2.0,  1.0,   0.3,   0.2},
       {3.0,  2.0,   0.4,   0.3},
       {7.0,  5.0,   2.0,   0.5},
       {18.0, 27.0,  8.0,   2.0},
```

```
        {23.0, 30.0,  10.0,  4.0},
        {17.0, 23.0,  12.0,  3.0},
        {9.0,  12.0,  7.0,   2.0},
        {7.0,  9.0,   3.0,   0.4},
        {5.0,  12.0,  4.0,   0.2},
        {0.7,  3.0,   1.0,   0.1},
};
static const size_t START_HOUR_CNT = sizeof(AVG_NO_SES) / sizeof(AVG_NO_SES[0]);

//------------------------------------------------------------------------------

// OCCUPIED TABLE MAP

typedef map<string,Dtime_t> OccTableMap_t;
                            // Occupied table map type definition
typedef OccTableMap_t::iterator OccTableIter_t;
                            // Occupied table map iterator type

//------------------------------------------------------------------------------

// GENERATE A RANDOM NUMBER IN A GIVEN RANGE

double
GenRand (
        double              minVal,       // Minimum value
        double              maxVal)       // Maximum value
{
        double              r;            // Random value

        modf (minVal + (maxVal - minVal + 1) * static_cast<double>(rand())
                / (static_cast<double>(RAND_MAX) + 1), &r);
        return r;
}

//------------------------------------------------------------------------------

// GENERATE TABLE NUMBER

void
GenTableNo (
        char                *tableNo,     // Table number
        OccTableMap_t       *occTableMap) // Occupied table map
{
        ostringstream       oss;          // Output string stream
        string              os;           // Output string

        do {
                oss.str ("");
                oss << setfill('0') << setw(4) << (rand() % 200);
                os = oss.str();
        } while (occTableMap->find(os) != occTableMap->end());
        strcpy (tableNo, os.c_str());
}

//------------------------------------------------------------------------------

// GENERATE GROUP SIZE

long
GenCovers (
        size_t              j)            // Size group index
{
        return MIN_GROUP_SIZE[j] +
                rand() % (MAX_GROUP_SIZE[j] - MIN_GROUP_SIZE[j] + 1);
}

//------------------------------------------------------------------------------
```

```
// GENERATE WAITER CODE

void
GenWaiterCode (
	char		*waiterCode)	// Waiter code
{
	ostringstream	oss;		// Output string stream
	string		os;		// Output string

	oss.str ("");
	oss << setfill('0') << setw(8) << (rand() % 100000000);
	os = oss.str();
	strcpy (waiterCode, os.c_str());
}

//------------------------------------------------------------------------------

// GENERATE REFERENCE CODE

void
GenRefCode (
	char		*refCode)	// Reference code
{
	ostringstream	oss;		// Output string stream
	string		os;		// Output string

	oss.str ("");
	oss << setfill('0') << setw(8) << (rand() % 100000000);
	os = oss.str();
	strcpy (refCode, os.c_str());
}

//------------------------------------------------------------------------------

// RELEASE UNOCCUPIED TABLES

void
ReleaseTables (
	OccTableMap_t	*occTableMap,	// Occupied table map
	Dtime_t		curTime)	// Current time
{
	OccTableIter_t	occTableIter;	// Occupied table iterator
	vector<string>	freeTables;	// Vector of free tables

	for (
		occTableIter = occTableMap->begin();
		occTableIter != occTableMap->end();
		occTableIter ++
	) {
		if (occTableIter->second < curTime)
			freeTables.push_back (occTableIter->first);
	}

	while (freeTables.begin() != freeTables.end()) {
		occTableMap->erase (freeTables.back());
		freeTables.pop_back();
	}
}

//------------------------------------------------------------------------------

// MAIN LINE

int
main ()
{
	Dtime_t		firstDate;	// Begin date
	Dtime_t		lastDate;	// End date
	Dtime_t		curDate;	// Current date
```

```
unsigned long       curSec;      // Current second
Dtime_t             curTime;     // Current time
size_t              i, j;        // General purpose indices
double              pArrival;    // Prob of a group of guests arriving
double              sesDur;      // Session duration
OccTableMap_t       occTableMap; // Occupied table map

exec sql begin declare section;
       long         orderNo;            // Order number
       char         orderDate[10+1];    // Order date
       char         orderTime[26+1];    // Order time
       char         tableNo[4+1];       // Table number
       long         covers;             // Number of guests
       char         waiterCode[8+1];    // Waiter code
       char         settleDate[10+1];   // Settlement date
       char         settleTime[26+1];   // Settlement time
       short        settleMode;         // Settlement mode
       double       billAmount;         // Bill amount
       double       tipAmount;          // Tip amount
       char         refCode[32+1];      // Reference code
       short        refCodeInd;         // Indicator for refCode
exec sql end declare section;

// Connect to the database

exec sql connect to tabledb;

// Drop tables

exec sql whenever sqlerror continue;
exec sql drop table orders;
exec sql drop table settlements;
exec sql commit work;

// Jump to DbError whenever an SQL error occurs

exec sql whenever sqlerror goto DbError;

// Create the database tables

exec sql create table orders (
       orderNo      integer not null,
       orderDate    date not null,
       orderTime    timestamp not null,
       tableNo      char(4) not null,
       covers       integer not null,
       waiterCode   char(8) not null
);
exec sql create table settlements (
       tableNo      char(4) not null,
       settleDate   date not null,
       settleTime   timestamp not null,
       settleMode   smallint not null,
       billAmount   double precision not null,
       tipAmount    double precision not null,
       refCode      char(32)
);

// Initialise the random number generator

srand (20360L);
```

```
	// Process each trading day

	orderNo = 0;
	firstDate = DtimeFromSql (FIRST_DATE);
	lastDate = DtimeFromSql (LAST_DATE);
	for (
		curDate = firstDate;
		curDate <= lastDate;
		curDate += 24.0*60.0*60.0
	) {
		// Process each second

		for (curSec = 0; curSec < START_HOUR_CNT*3600; curSec++) {

			// Calculate the hourly segment

			i = curSec / 3600;

			// Process each group size segment

			for (j = 0; j < GROUP_SIZE_CNT; j++) {

				// Determine the probability of a group
				// arriving in any second in the hourly
				// segment.

				pArrival = AVG_NO_SES[i][j] / 3600.0;

				// Determine if a guest arrived in this
				// one-second interval.

				if (rand() < pArrival*RAND_MAX) {

					// Generate the order time

					curTime = curDate + FIRST_HOUR + curSec;

					// Release any vacated tables

					ReleaseTables (&occTableMap, curTime);

					// Create an orders table row for the
					// arriving group of guests

					orderNo ++ ;
					SqlDateFromDtime (orderDate, curDate);
					SqlTimestampFromDtime (orderTime,
						curTime);
					GenTableNo (tableNo, &occTableMap);
					covers = GenCovers(j);
					GenWaiterCode (waiterCode);
					exec sql insert into orders (
						orderNo, orderDate,
						orderTime, tableNo,
						covers, waiterCode
					) values (
						:orderNo, :orderDate,
						:orderTime, :tableNo,
						:covers, :waiterCode
					);

					// Determine the duration of the session

					sesDur = GenRand (MIN_SES_DUR,
						MAX_SES_DUR);
```

```
                                    // Create a settlements table
                                    // row

                                    SqlDateFromDtime (settleDate, curDate);
                                    SqlTimestampFromDtime (settleTime,
                                           curTime + sesDur);
                                    settleMode = rand() % 4;
                                    billAmount = GenRand (1000.0, 100000.0);
                                    tipAmount = GenRand (10.0, 2000.0);
                                    if (rand() % 2 == 0) {
                                           GenRefCode (refCode);
                                           refCodeInd = 0;
                                    } else {
                                           refCode[0] = '\0';
                                           refCodeInd = -1;
                                    }
                                    exec sql insert into settlements (
                                           tableNo, settleDate,
                                           settleTime, settleMode,
                                           billAmount, tipAmount,
                                           refCode
                                    ) values (
                                           :tableNo, :settleDate,
                                           :settleTime,:settleMode,
                                           :billAmount, :tipAmount,
                                           :refCode
                                    );

                                    // Update the occupied tables map

                                    occTableMap[tableNo] = curTime + sesDur;
                             }
                      }
               }

               // Commit one day at a time to avoid filling the
               // rollback log

               exec sql commit work;
        }

        // And that's all

        return 0;

        // Process database errors

DbError:
        cerr << "Error: SQLCODE=" << SQLCODE << '\n';
        return 1;
}
```