```
// TrioGen.cpp - TRIO TRADING DATABASE GENERATOR
//
// MODULE INDEX
// NAME                 CONTENTS
// LeapYear             Test for a leap year
// CalcNextDay          Calculate the date of the next day
// GenRand              Generate a random number in a given range
// main                 Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 17-08-07     JS      Original
//
//-----------------------------------------------------------------------------

#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <cstdlib>              // C-style standard library
#include <cstring>              // C-style string manipulation functions
#include <ctime>                // C-style time functions
#include <cctype>               // C-style character typing functions
using namespace std;            // Expand the standard namespace
exec sql include sqlca;         // Include SQL communications area

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

// DEFINITIONS

static const long       FIRST_TRANS_NO = 85179232;
                                // First transaction number
static const char       START_DATE[] = "2006-01-01";
                                // Start date
static const char       END_DATE[] = "2007-09-22";
                                // End date
static const size_t     DATE_LEN = 10;
                                // Length of a date string
static const size_t     ACC_ID_LEN = 10;
                                // Length of an account identifier
static const size_t     ACC_DESCR_LEN = 60;
                                // Length of an account description
static const long       MIN_TX_PER_DAY = 0;
                                // Minimum transactions per day
static const long       MAX_TX_PER_DAY = 50;
                                // Maximum transactions per day
static const double     MIN_PURCH_VAL = 100.0;
                                // Minimum purchase value (cents)
static const double     MAX_PURCH_VAL = 250000.0;
                                // Maximum purchase value (cents)
static const double     MIN_SALE_VAL = 150.0;
                                // Minimum sale value (cents)
static const double     MAX_SALE_VAL = 275000.0;
                                // Maximum sale value (cents)
static const double     MIN_MORT_VAL = 10000.0;
                                // Minimum mortgage drawdown (cents)
static const double     MAX_MORT_VAL = 100000.0;
                                // Maximum mortgage drawdown (cents)

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

// CHART OF ACCOUNTS

struct AccRow_t {
        char            accId[ACC_ID_LEN+1];            // Account identifier
```

```
        char            accDescr[ACC_DESCR_LEN+1];      // Account description
};
static const AccRow_t ACC_ROW_ARR[] = {
        /*0*/   {"SOH0001000",  "Helical Lamps on Hand"},
        /*1*/   {"SOH0002000",  "Spiral Lamps on Hand"},
        /*2*/   {"SOH0003000",  "Bed Lamps on Hand"},
        /*3*/   {"SOH0004000",  "Chandeliers on Hand"},
        /*4*/   {"SOH0005000",  "Desk Lamps on Hand"},
        /*5*/   {"SOH0006000",  "Power Sockets on Hand"},
        /*6*/   {"SOH0007000",  "Lamp Sockets on Hand"},
        /*7*/   {"SOH0008000",  "Light Bulbs on Hand"},
        /*8*/   {"CGS0001000",  "Cost of Goods Sold"},
        /*9*/   {"VGP0001000",  "Value of Goods Purchased"},
        /*10*/  {"CAB0001000",  "Cash at Bank, Antopolis Commercial Bank"},
        /*11*/  {"CAB0002000",  "Cash at Bank, Bank of Trioville"},
        /*12*/  {"PUR0001000",  "Purchases"},
        /*13*/  {"SAL0001000",  "Sales"},
        /*14*/  {"TDR0001000",  "Trade Debtors"},
        /*15*/  {"TCR0001000",  "Trade Creditors"},
        /*16*/  {"MOR0001000",  "Mortgage on Warehouse"},
};
static const size_t ACC_ROW_CNT = sizeof(ACC_ROW_ARR) / sizeof(ACC_ROW_ARR[0]);

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

// STOCK ON HAND ACCOUNT INDEXES

static const size_t SOH_ACC_ARR[] = {0, 1, 2, 3, 4, 5, 6, 7};
static const size_t SOH_ACC_CNT = sizeof(SOH_ACC_ARR) / sizeof(SOH_ACC_ARR[0]);

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

// BANK ACCOUNT INDEXES

static const size_t BANK_ACC_ARR[] = {10, 11};
static const size_t BANK_ACC_CNT = sizeof(BANK_ACC_ARR)/sizeof(BANK_ACC_ARR[0]);

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

// OTHER ACCOUNT INDEXES

static const size_t COST_OF_SALES_IND = 8;
                                // Value of goods purchased
static const size_t VAL_GOODS_PURCH_IND = 9;
                                // Value of goods purchased
static const size_t PURCH_IND = 12;
                                // Purchases
static const size_t SALES_IND = 13;
                                // Sales
static const size_t TRADE_DR_IND = 14;
                                // Trade debtors
static const size_t TRADE_CR_IND = 15;
                                // Trade creditors
static const size_t MORT_IND = 16;
                                // Mortgage

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

// TEST FOR A LEAP YEAR

inline bool
LeapYear (
        int             y)              // Year number (e.g. 2004)
{
```

```
        return y % 4 == 0 && (y % 100 != 0 || y % 400 == 0);
}

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

// CALCULATE THE DATE OF THE NEXT DAY

void
CalcNextDay (
        char            *nextDate,      // Next date
        const char      *thisDate)      // Current date
{
        const char      *p;             // General purpose pointer
        int             yr, mo, dy;     // Date components
        ostringstream   oss;            // Output string stream
        string          ostr;           // Output string

        // Days per month

        static const int DAYS_PER_MONTH[] =
                {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        // Decode the current date

        p = thisDate;
        yr = 0;
        while (isdigit(*p)) yr = yr * 10 + *p++ - '0';
        if (*p++ != '-') goto BadDate;
        mo = 0;
        while (isdigit(*p)) mo = mo * 10 + *p++ - '0';
        if (*p++ != '-') goto BadDate;
        dy = 0;
        while (isdigit(*p)) dy = dy * 10 + *p++ - '0';
        if (*p != '\0') goto BadDate;

        // Increment the date

        dy ++ ;
        if (
                dy > DAYS_PER_MONTH[mo-1] &&
                (! LeapYear(yr) || (mo != 2 && dy != 29))
        ) {
                dy = 1;
                mo ++ ;
                if (mo > 12) {
                        mo = 1;
                        yr ++ ;
                }
        }

        // Generate the new date

        oss << setfill('0');
        oss << setw(4) << yr << '-' << setw(2) << mo << '-' << setw(2) << dy;
        ostr = oss.str();
        strcpy (nextDate, ostr.c_str());
        return;

        // Process a bad date
BadDate:
        cerr << "Error: bad date '" << thisDate << "'\n";
        exit (1);
}
```

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

// 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;
}

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

// MAIN LINE

int
main ()
{
        long            nextTxNo;               // Next transaction number
        double          accBal[ACC_ROW_CNT];    // Account balances
        size_t          i;                      // General purpose index
        size_t          txCnt;                  // Number of transactions
        int             typePct;                // Type percentage
        size_t          sohAccInd;              // Stock-on-hand index
        size_t          bankAccInd;             // Bank account index
        exec sql begin declare section;
                char            date[10+1];             // Current date
                long            txNo;                   // Transaction number
                char            txDescr[60+1];          // Transaction descr
                char            fromAccId[10+1];        // From account id
                char            toAccId[10+1];          // To account id
                double          txValue;                // Transaction value
                char            accId[10+1];            // Account identifier
                char            accDescr[60+1];         // Account description
                double          accVal;                 // Account balance
                short           lineNo;                 // Line number
                char            lineDescr[40+1];        // Line description
        exec sql end declare section;

        // Connect to the database

        exec sql connect to triodb;

        // Drop tables

        exec sql whenever sqlerror continue;
        exec sql drop table accounts;
        exec sql drop table transactions;
        exec sql drop table lines;
        exec sql drop table elements;
        exec sql drop table references;
        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 accounts (
        accId           char(10) not null,
        accDescr        char(60) not null,
        accBalance      double precision not null
);
exec sql create table transactions (
        txDate          date not null,
        txNo            integer not null,
        txDescr         char(60) not null,
        txFromAccId     char(10) not null,
        txToAccId       char(10) not null,
        txValue         double precision not null
);
exec sql create table lines (
        lineNo          smallint not null,
        lineDescr       char(40) not null,
        lineType        smallint not null
);
exec sql create table elements (
        eleLineNo       smallint not null,
        eleSign         smallint not null,
        eleAccId        char(10) not null
);
exec sql create table references (
        refThisLineNo   smallint not null,
        refSign         smallint not null,
        refOtherLineNo  smallint not null
);

// Initialise the random number generator

srand (20360L);

// Initialise the next transaction number

nextTxNo = FIRST_TRANS_NO;

// Initialise the account balances

for (i = 0; i < ACC_ROW_CNT; i++) accBal[i] = 0;

// Generate data for each date

for (
        strcpy (date, START_DATE);
        strcmp (date, END_DATE) <= 0;
        CalcNextDay (date, date)
) {
        // Select a stock-on-hand and bank account index

        sohAccInd = SOH_ACC_ARR [ rand() % SOH_ACC_CNT ];
        bankAccInd = BANK_ACC_ARR [ rand() % BANK_ACC_CNT ];

        // Generate individual transactions

        txCnt = static_cast <size_t>
                (GenRand (MIN_TX_PER_DAY, MAX_TX_PER_DAY));
        for (i = 0; i < txCnt; i++) {

                // Select the transaction type

                typePct = rand() % 97;

                // Purchases
```

```
if (typePct < 24) {

        // Transfer trade creditors to purchases

        txNo = nextTxNo++;
        strcpy (txDescr, "Purchase");
        strcpy (fromAccId,
                ACC_ROW_ARR[TRADE_CR_IND].accId);
        strcpy (toAccId,
                ACC_ROW_ARR[PURCH_IND].accId);
        txValue = GenRand(MIN_PURCH_VAL,MAX_PURCH_VAL);
        exec sql insert into transactions (
                txDate, txNo, txDescr,
                txFromAccId, txToAccId, txValue
        ) values (
                :date, :txNo, :txDescr,
                :fromAccId, :toAccId, :txValue
        );
        accBal[TRADE_CR_IND] -= txValue;
        accBal[PURCH_IND] += txValue;

        // Transfer value of goods purchased to
        // stock on hand

        txNo = nextTxNo++;
        strcpy (fromAccId,
                ACC_ROW_ARR[VAL_GOODS_PURCH_IND].accId);
        strcpy (toAccId,
                ACC_ROW_ARR[sohAccInd].accId);
        exec sql insert into transactions (
                txDate, txNo, txDescr,
                txFromAccId, txToAccId, txValue
        ) values (
                :date, :txNo, :txDescr,
                :fromAccId, :toAccId, :txValue
        );
        accBal[VAL_GOODS_PURCH_IND] -= txValue;
        accBal[sohAccInd] += txValue;
}

// Sales

else if (typePct < 48) {

        // Transfer stock on hand to cost of sales

        txNo = nextTxNo++;
        strcpy (txDescr, "Sale");
        strcpy (fromAccId,
                ACC_ROW_ARR[sohAccInd].accId);
        strcpy (toAccId,
                ACC_ROW_ARR[COST_OF_SALES_IND].accId);
        txValue = GenRand(MIN_PURCH_VAL,MAX_PURCH_VAL);
        if (txValue > accBal[sohAccInd])
                txValue = accBal[sohAccInd];
        exec sql insert into transactions (
                txDate, txNo, txDescr,
                txFromAccId, txToAccId, txValue
        ) values (
                :date, :txNo, :txDescr,
                :fromAccId, :toAccId, :txValue
        );
        accBal[sohAccInd] -= txValue;
```

```
                accBal[COST_OF_SALES_IND] += txValue;

                // Transfer sales to trade debtors

                txNo = nextTxNo++;
                strcpy (fromAccId,
                        ACC_ROW_ARR[SALES_IND].accId);
                strcpy (toAccId,
                        ACC_ROW_ARR[TRADE_DR_IND].accId);
                txValue = GenRand(MIN_SALE_VAL,MAX_SALE_VAL);
                exec sql insert into transactions (
                        txDate, txNo, txDescr,
                        txFromAccId, txToAccId, txValue
                ) values (
                        :date, :txNo, :txDescr,
                        :fromAccId, :toAccId, :txValue
                );
                accBal[SALES_IND] -= txValue;
                accBal[TRADE_DR_IND] += txValue;
        }

        // Cash received

        else if (typePct < 72) {

                // Generate transaction value

                txValue = GenRand(MIN_SALE_VAL,MAX_SALE_VAL);

                // If value is more than value of trade
                // debtors, truncate the value

                if (txValue > accBal[TRADE_DR_IND])
                        txValue = accBal[TRADE_DR_IND];

                // Transfer trade debtors to bank account

                txNo = nextTxNo++;
                strcpy (txDescr, "Cash received");
                strcpy (fromAccId,
                        ACC_ROW_ARR[TRADE_DR_IND].accId);
                strcpy (toAccId,
                        ACC_ROW_ARR[bankAccInd].accId);
                exec sql insert into transactions (
                        txDate, txNo, txDescr,
                        txFromAccId, txToAccId, txValue
                ) values (
                        :date, :txNo, :txDescr,
                        :fromAccId, :toAccId, :txValue
                );
                accBal[TRADE_DR_IND] -= txValue;
                accBal[bankAccInd] += txValue;
        }

        // Cash paid

        else if (typePct < 96) {

                // Generate transaction value

                txValue = GenRand(MIN_PURCH_VAL,MAX_PURCH_VAL);

                // If value is more than the value of cash
                // at bank, truncate the value
```

```
                        if (txValue > accBal[bankAccInd])
                                txValue = accBal[bankAccInd];

                        // Transfer cash at bank to trade creditors

                        txNo = nextTxNo++;
                        strcpy (txDescr, "Cash paid");
                        strcpy (fromAccId,
                                ACC_ROW_ARR[bankAccInd].accId);
                        strcpy (toAccId,
                                ACC_ROW_ARR[TRADE_CR_IND].accId);
                        exec sql insert into transactions (
                                txDate, txNo, txDescr,
                                txFromAccId, txToAccId, txValue
                        ) values (
                                :date, :txNo, :txDescr,
                                :fromAccId, :toAccId, :txValue
                        );
                        accBal[bankAccInd] -= txValue;
                        accBal[TRADE_CR_IND] += txValue;
                }

                // Mortgage drawdown

                else if (typePct < 97) {

                        // Generate transaction value

                        txValue = GenRand(MIN_MORT_VAL,MAX_MORT_VAL);

                        // Transfer Mortgage on warehouse to bank
                        // account

                        txNo = nextTxNo++;
                        strcpy (txDescr, "Drawdown on mortgage");
                        strcpy (fromAccId,
                                ACC_ROW_ARR[MORT_IND].accId);
                        strcpy (toAccId,
                                ACC_ROW_ARR[bankAccInd].accId);
                        exec sql insert into transactions (
                                txDate, txNo, txDescr,
                                txFromAccId, txToAccId, txValue
                        ) values (
                                :date, :txNo, :txDescr,
                                :fromAccId, :toAccId, :txValue
                        );
                        accBal[MORT_IND] -= txValue;
                        accBal[bankAccInd] += txValue;
                }
        }
}

// Create the account rows

for (i = 0; i < ACC_ROW_CNT; i++) {
        strcpy (accId, ACC_ROW_ARR[i].accId);
        strcpy (accDescr, ACC_ROW_ARR[i].accDescr);
        accVal = accBal[i];
        exec sql insert into accounts (
                accId, accDescr, accBalance
        ) values (
                :accId, :accDescr, :accVal
        );
```

```
}

// Create the stock on hand lines

for (i = 0; i < SOH_ACC_CNT; i++) {
        lineNo = 1000 + i * 10;
        strcpy (accId, ACC_ROW_ARR[SOH_ACC_ARR[i]].accId);
        strcpy (lineDescr, ACC_ROW_ARR[SOH_ACC_ARR[i]].accDescr);
        exec sql insert into lines (
                lineNo, lineDescr, lineType
        ) values (
                :lineNo, :lineDescr, 0
        );
        exec sql insert into elements (
                eleLineNo, eleSign, eleAccId
        ) values (
                :lineNo, 1, :accId
        );
}
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        1900, ' ', 2
);
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        1910, 'Total value of stock on hand', 0
);
for (i = 0; i < ACC_ROW_CNT; i++) {
        lineNo = 1000 + i * 10;
        exec sql insert into references (
                refThisLineNo, refSign, refOtherLineNo
        ) values (
                1910, 1, :lineNo
        );
}
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        1920, ' ', 1
);

// Create the bank account lines

for (i = 0; i < BANK_ACC_CNT; i++) {
        lineNo = 2000 + i * 10;
        strcpy (accId, ACC_ROW_ARR[BANK_ACC_ARR[i]].accId);
        strcpy (lineDescr, ACC_ROW_ARR[BANK_ACC_ARR[i]].accDescr);
        exec sql insert into lines (
                lineNo, lineDescr, lineType
        ) values (
                :lineNo, :lineDescr, 0
        );
        exec sql insert into elements (
                eleLineNo, eleSign, eleAccId
        ) values (
                :lineNo, 1, :accId
        );
}
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        2900, ' ', 2
```

```
);
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        2910, 'Total cash at bank', 0
);
for (i = 0; i < ACC_ROW_CNT; i++) {
        lineNo = 2000 + i * 10;
        exec sql insert into references (
                refThisLineNo, refSign, refOtherLineNo
        ) values (
                2910, 1, :lineNo
        );
}
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        2920, ' ', 1
);

// Trade debtors

exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        3910, 'Trade debtors', 0
);
strcpy (accId, ACC_ROW_ARR[TRADE_DR_IND].accId);
exec sql insert into elements (
        eleLineNo, eleSign, eleAccId
) values (
        3910, 1, :accId
);
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        3920, ' ', 1
);

// Trade creditors

exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        4910, 'Trade creditors', 0
);
strcpy (accId, ACC_ROW_ARR[TRADE_CR_IND].accId);
exec sql insert into elements (
        eleLineNo, eleSign, eleAccId
) values (
        4910, -1, :accId
);

// Mortgage on warehouse

exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        5910, 'Mortgage on warehouse', 0
);
strcpy (accId, ACC_ROW_ARR[MORT_IND].accId);
exec sql insert into elements (
        eleLineNo, eleSign, eleAccId
) values (
```

```
        5910, -1, :accId
);
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        5920, ' ', 2
);

// Total liabilities

exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        6910, 'Total liabilities', 0
);
exec sql insert into references (
        refThisLineNo, refSign, refOtherLineNo
) values (
        6910, 1, 4910
);
exec sql insert into references (
        refThisLineNo, refSign, refOtherLineNo
) values (
        6910, 1, 5910
);
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        6920, ' ', 2
);

// Net worth

exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        7910, 'Net worth', 0
);
exec sql insert into references (
        refThisLineNo, refSign, refOtherLineNo
) values (
        7910, 1, 1910
);
exec sql insert into references (
        refThisLineNo, refSign, refOtherLineNo
) values (
        7910, 1, 2910
);
exec sql insert into references (
        refThisLineNo, refSign, refOtherLineNo
) values (
        7910, 1, 3910
);
exec sql insert into references (
        refThisLineNo, refSign, refOtherLineNo
) values (
        7910, -1, 6910
);
exec sql insert into lines (
        lineNo, lineDescr, lineType
) values (
        7920, ' ', 3
);
```

```
        // And that's all

        exec sql commit work;
        return 0;

        // Process database errors

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