```
// MoveGen.sqC - MOVE-IT-QUICK DATA GENERATOR
//
// MODULE INDEX
// NAME                         CONTENTS
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 04-10-04     JS      Original
//
//-----------------------------------------------------------------------------

#include <string.h>             // String manipulation functions
#include <errno.h>              // Operating system error codes
#include <time.h>               // Time declarations
#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <vector>               // C++ vectors
#include <string>               // C++ style strings
#include <functional>           // C++ function objects
using namespace std;            // Expand standard namespace to global scope
exec sql include sqlca;         // Include SQL communications area

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

// DEFINITIONS

const size_t            CUST_CNT = 100;
                                // Number of customers
const size_t            SERV_CNT = 1000;
                                // Number of services
const size_t            SERV_CAT_CNT = 12;
                                // Number of service categories
const size_t            PREM_CUST_PCT = 10;
                                // Premium customer percentage
const size_t            MIN_ORD_TRAN_CNT = 5;
                                // Minimum ordinary customer transaction count
const size_t            MAX_ORD_TRAN_CNT = 15;
                                // Maximum ordinary customer transaction count
const size_t            MIN_PREM_TRAN_CNT = 600;
                                // Minimum premium customer transaction count
const size_t            MAX_PREM_TRAN_CNT = 1200;
                                // Maximum premium customer transaction count
const size_t            MIN_ITEM_PER_TRAN = 1;
                                // Minimum items per transaction
const size_t            MAX_ITEM_PER_TRAN = 5;
                                // Maximum items per transaction
const size_t            MIN_SERV_PER_CUST = 1;
                                // Minimum services per customer
const size_t            MAX_SERV_PER_CUST = 50;
                                // Maximum services per customer
const long              MAX_TRAN_AGE = 5*365;
                                // Maximum transaction age
const long              MIN_AMT_SOLD = 100;
                                // Minimum amount sold
const long              MAX_AMT_SOLD = 10000;
                                // Maximum amount sold
const long              MIN_COST_PCT = 50;
                                // Minimum cost percentage
const long              MAX_COST_PCT = 100;
                                // Maximum cost percentage

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

// STRING FIELD LENGTHS

const size_t            CUST_NAME_LEN = 40;
                                // Customer name
const size_t            CUST_TEL_LEN = 25;
                                // Customer's telephone number
const size_t            SERV_CODE_LEN = 16;
                                // Service code length
const size_t            SERV_CAT_LEN = 8;
                                // Service category length

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

// GLOBAL DATA

string                  servCatArr[SERV_CAT_CNT];
                                // Service categories
string                  servCodeArr[SERV_CNT];
```

```
                                // Service codes
//----------------------------------------------------------------------------

// TEST FOR A LEAP YEAR

inline bool
LeapYear (
    long                y)              // Year number (e.g. 2004)
{
    return y % 4 == 0;
}

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

// CONVERT YEAR NUMBER INTO AN ABSTRACT DAY NUMBER

inline long
YearToDayNo (
    long                y)              // Year number (e.g. 2004)
{
    return ((y-1L)*365L + (y-1)/4 - (y-1)/100 + (y-1)/400);
}

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

// CONVERT DATE COMPONENTS INTO DATABASE DATE FORMAT

long
LoadDate (
        int             year,           // Year number
        int             month,          // Month number
        int             day)            // Day number
{
        long            jd;             // Julian day
        const int       *dayMap;        // Pointer to ordMap or leapMap

        // Map month to number of days before the first of the month

        const static int ordMap[13] =
                {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};
        const static int leapMap[13] =
                {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};

        // Do the conversion

        dayMap = LeapYear(year) ? leapMap : ordMap;
        jd = dayMap[month-1] + day - 1;
        return YearToDayNo(year) + jd - YearToDayNo(1970L);
}

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

int
main ()
{
    size_t              i, j, k;        // General purpose indices
    size_t              nameWordLen;    // Name word length
    size_t              telLen;         // Telephone number length
    vector<string>      custServCodeVec; // Services that the customer uses
    size_t              custServCodeCnt; // Number of services that the customer
                                        // uses
    size_t              tranCnt;        // Number of transactions
    size_t              itemCnt;        // Number of items in the transaction
    time_t              genTime;        // Generation time
    struct tm           genTm;          // Generation time fields
    long                genDate;        // Generation date in database format
    long                costPct;        // Cost percentage
    exec sql begin declare section;
        char            servCode[16+1]; // Service code
        char            servCat[8+1];   // Service category
        long            custId;         // Customer identifier
        char            custName[40+1]; // Customer name
        char            custTel[25+1];  // Customer's telephone number
        short           cnt;            // Row count
        long            tranNo;         // Transaction number
        long            tranDate;       // Transaction date
        long            lAmtSold;       // Long amount sold
        long            lCost;          // Long cost of sale
        double          dAmtSold;       // Double amount sold
        double          dCost;          // Double cost
    exec sql end declare section;
```

```
exec sql whenever sqlerror goto DbError;

// Connect to the Customer Database

exec sql connect to cust;

// Create the database tables

// Customer File

exec sql create table custFile (
    custId          integer not null,
    custName        char(40) not null,
    custTel         char(25) not null
);
exec sql create unique index custIdInd on custFile(custId);

// Service File

exec sql create table servFile (
    servCode        char(16) not null,
    servCat         char(8) not null
);
exec sql create unique index servCodeInd on servFile(servCode);

// Transaction File

exec sql create table tranFile (
    tranNo          integer not null,
    tranCustId      integer not null,
    tranDate        integer not null
);
exec sql create unique index tranNoInd on tranFile(tranNo);
exec sql create index tranCustDateInd on tranFile(tranCustId, tranDate);

// Item File

exec sql create table itemFile (
    itemTranNo      integer not null,
    itemServCode    char(16) not null,
    itemAmtSold     double precision not null,
    itemCost        double precision not null
);
exec sql create unique index itemTranServInd on
    itemFile (itemTranNo, itemServCode);

// Generate the service categories

for (i = 0; i < SERV_CAT_CNT; i++) {
    ostringstream servCatOst;
    servCatOst << "CAT" << setfill('0') << setw(5) << i+1;
    servCatArr[i] = servCatOst.str();
}

// Generate the services

for (i = 0; i < SERV_CNT; i++) {
    ostringstream servCodeOst;
    servCodeOst << "SELLITFAST-" << setfill('0') << setw(5) << i+1;
    servCodeArr[i] = servCodeOst.str();
    strcpy (servCode, servCodeOst.str().c_str());
    j = lrand48() % SERV_CAT_CNT;
    strcpy (servCat, servCatArr[j].c_str());
    exec sql insert into servFile (
        servCode, servCat
    ) values (
        :servCode, :servCat
    );
}

// Work out the generation date in database format

genTime = time(0);
localtime_r (&genTime, &genTm);
genDate = LoadDate (genTm.tm_year+1900, genTm.tm_mon+1, genTm.tm_mday);

// Generate the customers and transactions

for (custId = 0; custId < CUST_CNT; custId++) {

    // Generate the Customer File row
```

```
ostringstream custNameOst;
ostringstream custTelOst;
for (j = 0; j < 3; j++) {
    if (j != 0) custNameOst << ' ';
    nameWordLen = lrand48() % 5 + 2;
    for (k = 0; k < nameWordLen; k++)
        custNameOst << char(lrand48() % ('Z' - 'A' + 1) + 'A');
}
telLen = lrand48() % 14 + 8;
for (j = 0; j < telLen; j++)
    custTelOst << lrand48() % 10;
strcpy (custName, custNameOst.str().c_str());
strcpy (custTel, custTelOst.str().c_str());
exec sql insert into custFile (
    custId, custName, custTel
) values (
    :custId, :custName, :custTel
);

// Select the services that the customer purchases

custServCodeCnt = lrand48() % (MAX_SERV_PER_CUST-MIN_SERV_PER_CUST+1)
    + MIN_SERV_PER_CUST;
while (custServCodeVec.size() < custServCodeCnt) {
    j = lrand48() % SERV_CNT;
    if (
        find(custServCodeVec.begin(), custServCodeVec.end(),
            servCodeArr[j]) == custServCodeVec.end()
    )
        custServCodeVec.push_back(servCodeArr[j]);
}

// Select the number of transactions for the customer

if (lrand48() % 100 < PREM_CUST_PCT)
    tranCnt = lrand48() % (MAX_PREM_TRAN_CNT-MIN_PREM_TRAN_CNT+1)
        + MIN_PREM_TRAN_CNT;
else
    tranCnt = lrand48() % (MAX_ORD_TRAN_CNT-MIN_ORD_TRAN_CNT+1)
        + MIN_ORD_TRAN_CNT;

// Generate the transactions

for (i = 0; i < tranCnt; i++) {
    do {
        tranNo = lrand48() % 1000000000;
        exec sql select count(*)
                into    :cnt
                from    tranFile
                where   tranNo = :tranNo;
    } while (cnt != 0);
    tranDate = genDate - lrand48() % MAX_TRAN_AGE;
    exec sql insert into tranFile (
        tranNo, tranCustId, tranDate
    ) values (
        :tranNo, :custId, :tranDate
    );

    // Generate the items

    itemCnt = lrand48() % (MAX_ITEM_PER_TRAN-MIN_ITEM_PER_TRAN+1)
        + MIN_ITEM_PER_TRAN;
    vector<string> custServCodeRem = custServCodeVec;
    for (j = 0; j < itemCnt && j < custServCodeRem.size() != 0; j++) {
        k = lrand48() % custServCodeRem.size();
        strcpy (servCode, custServCodeRem[k].c_str());
        custServCodeRem.erase (custServCodeRem.begin()+k);
        lAmtSold = lrand48() % (MAX_AMT_SOLD-MIN_AMT_SOLD+1)
            + MIN_AMT_SOLD;
        costPct = lrand48() % (MAX_COST_PCT-MIN_COST_PCT+1)
            + MIN_COST_PCT;
        lCost = lAmtSold * costPct / 100;
        dAmtSold = lAmtSold;
        dCost = lCost;
        exec sql insert into itemFile (
            itemTranNo, itemServCode, itemAmtSold, itemCost
        ) values (
            :tranNo, :servCode, :dAmtSold, :dCost
        );
    }
}
```

```
    }

    // And that's all

    exec sql commit work;
    return 0;

    // Process database errors

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