```
// ProfRep.sqC - CUSTOMER PROFILE REPORT
//
// 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
#include <map>                  // C++ map object declarations
using namespace std;            // Expand standard namespace to global scope
exec sql include sqlca;         // Include SQL communications area

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

// DEFINITIONS

const long              FULL_HISTORY = 0;
                                        // Symbolic constant for full customer
                                        // history
const static long       PERIOD_SIZE_ARR[] = {1,30,182,365,730,FULL_HISTORY};
                                        // Number of days in each period
const size_t            PERIOD_CNT = sizeof(PERIOD_SIZE_ARR)
                            / sizeof(PERIOD_SIZE_ARR[0]);
                                        // Number of analysis periods
const int               LINES_PER_PAGE = 55;
                                        // Lines per page

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

// INDIVIDUAL SERVICE CATEGORY DATA STRUCTURE

struct CatData_t {
        long            catTxCnt;       // Transaction count
        double          catAmtSold;     // Sale amounts
        double          catCost;        // Cost of sales
        CatData_t () { catTxCnt = 0; catAmtSold = 0; catCost = 0; }
                                        // Constructor
};

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

// INDIVIDUAL ANALYSIS PERIOD STRUCTURE

struct Period_t {
        long            prdSize;        // Number of days in the period
        map<string,CatData_t> prdCatData; // Category map
        CatData_t       prdTotal;       // Analysis period totals
};

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

// GLOBAL DATA

int                     pageNo;         // Page number
int                     linesRem;       // Lines remaining on the current page
struct tm               genTm;          // Generation time fields

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

// 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);
}

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

// FORMAT A NUMBER

string
FormatNum (
        double          val,            // Number
        size_t          wid,            // Field width
        size_t          dec)            // Decimal places
{
        char            stack[20];      // Digit stack
        char            *p;             // General purpose pointer
        bool            negative;       // Number is negative
        double          x;              // A temporary variable
        size_t          i;              // General purpose index
        size_t          groupSize;      // Digit group size

        // Initialise the digit stack and append a null terminator

        p = stack + sizeof(stack);
        *--p = '\0';

        // Make the number positive and remove any decimal places

        negative = val < 0;
        if (negative) val = -val;
        modf (val, &val);

        // Push the digits to the right of the decimal point

        if (dec != 0) {
                for (i = 0; i < dec; i++) {
                        if (p <= stack) goto TooLong;
                        modf (val/10, &x);
                        *--p = static_cast<char>(val - x*10) + '0';
                        val = x;
                }
                if (p <= stack) goto TooLong;
                *--p = '.';
        }

        // Push the digits to the left of the decimal point

        groupSize = 0;
        do {
                if (groupSize >= 3) {
```

```
                        if (p <= stack) goto TooLong;
                        *--p = ',';
                        groupSize = 0;
                }
                if (p <= stack) goto TooLong;
                modf (val/10, &x);
                *--p = static_cast<char>(val - x*10) + '0';
                val = x;
                groupSize ++ ;
        } while (val != 0);

        // Push the sign

        if (negative) {
                if (p <= stack) goto TooLong;
                *--p = '-';
        }

        // Fill with spaces to the required width

        while (stack+sizeof(stack)-1 - p < wid) {
                if (p <= stack) goto TooLong;
                *--p = ' ';
        }

        // Convert the formatted number into a string and return it

        return string (p);

        // Process string too long
TooLong:
        cerr << "Error: number too long\n";
        exit (1);
}

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

// DISPLAY THE PAGE TITLE

void
ShowPageTitle ()
{
        cout << '\f';
        cout << right << setfill('0');
        cout << setw(2) << genTm.tm_year % 100;
        cout << '-';
        cout << setw(2) << genTm.tm_mon + 1;
        cout << '-';
        cout << setw(2) << genTm.tm_mday;
        cout << ' ';
        cout << setw(2) << genTm.tm_hour;
        cout << ':';
        cout << setw(2) << genTm.tm_min;
        cout << setfill(' ');
        cout << setw(48-15) << "CUSTOMER PROFILE";
        cout << setw(79-48) << "PAGE";
        cout << setw(2) << ++pageNo;
        cout << '\n';
        linesRem = LINES_PER_PAGE - 1;
}

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

// DISPLAY CUSTOMER INFORMATION

void
ShowCustInfo (
        long            custId,         // Customer identifier
        const char      *custName,      // Customer name
        const char      *custTel)       // Customer's telephone number
{
        cout << '\n';
        cout << left << setfill(' ');
        cout << setw(19) << "Customer id:";
        cout << setfill('0') << setw(8) << custId << '\n';
        cout << left << setfill(' ');
        cout << setw(19) << "Customer name:" << custName << '\n';
        cout << setw(19) << "Telephone number:" << custTel << '\n' << '\n';
        linesRem -= 4;
}

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

```
// DISPLAY PROFILE TITLE

void
ShowProfTitle ()
{
        cout << right;
        cout << '\n';
        cout << setw(10) << "Number";
        cout << setw(10) << "Service ";
        cout << setw(9) << "Number";
        cout << setw(14) << "Amount";
        cout << setw(14) << "Cost of";
        cout << setw(14) << "Gross";
        cout << '\n';
        cout << setw(10) << "of days";
        cout << setw(10) << "category";
        cout << setw(9) << "of trans";
        cout << setw(14) << "sold";
        cout << setw(14) << "sales";
        cout << setw(14) << "revenue";
        cout << '\n';
        linesRem -= 3;
}

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

// DISPLAY CATEGORY DATA

void
ShowCatData (
        const string    &lineTitle,     // Line title
        const CatData_t *catData)       // Category data structure
{
        cout << left << "  " << setw(8) << lineTitle;
        cout << FormatNum (catData->catTxCnt, 9, 0);
        cout << FormatNum (catData->catAmtSold, 14, 2);
        cout << FormatNum (catData->catCost, 14, 2);
        cout << FormatNum (catData->catAmtSold - catData->catCost, 14, 2);
}

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

// DISPLAY THE PROFILE FOR AN INDIVIDUAL PERIOD

void
ShowPeriod (
        const Period_t  *prd)           // Period data
{
        int             linesRequired;  // Number of lines required
        map<string,CatData_t>::const_iterator catDataIt;
                                        // Category data iterator

        // Process a new page if necessary

        if (prd->prdCatData.size() == 0)
                linesRequired = 2;
        else
                linesRequired = prd->prdCatData.size() + 3;
        if (linesRem < linesRequired) {
                ShowPageTitle ();
                ShowProfTitle ();
        }

        // Display the number of days

        cout << FormatNum (prd->prdSize, 10, 0);

        // Display the values for each service category

        if (prd->prdCatData.size() != 0) {
                for (
                        catDataIt = prd->prdCatData.begin();
                        catDataIt != prd->prdCatData.end();
                        catDataIt ++
                ) {
                        ShowCatData (catDataIt->first, &catDataIt->second);
                        cout << '\n' << setw(10) << ' ';
                        linesRem -- ;
                }
                cout << setw(12) << ' ' << "-------";
                cout << "  --------------";
```

```
                cout << "  --------------";
                cout << "  --------------";
                cout << '\n' << setw(10) << ' ';
                linesRem -- ;
        }

        // Display the totals

        ShowCatData ("Total", &prd->prdTotal);
        cout << '\n';
        cout << '\n';
        linesRem -= 2;
}

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

int
main (
    int                 argc,           // Argument count
    char                *argv[])        // Argument value pointers
{
        Period_t        periodArr[PERIOD_CNT]; // Period data array
        Period_t        *prd;           // Period element pointer
        CatData_t       *catData;       // Category data pointer
        typedef class {} Usage_c;       // Usage exception class
        time_t          genTime;        // Generation time
        long            genDate;        // Generation date in database format
        map<string,string> catMap;      // Category map
        map<string,string>::iterator catMapIt; // Category map iterator
        long            tranAge;        // Transaction age
        vector<string>  tranCats;       // Transaction categories
        vector<string>::iterator tranCatsIt; // Transaction categories iterator
        size_t          i, j;           // General purpose index
        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
                double          amtSold;        // Amount sold
                double          cost;           // Cost
                long            firstDate;      // First date for the customer
                short           firstDateInd;   // Indicator for firstDate
        exec sql end declare section;

        // Decode the customer identifier

        try {
                if (argc != 2) throw Usage_c();
                istringstream iss(argv[1]);
                iss >> custId;
                if ( ! iss.eof()) throw Usage_c();
        }
        catch (Usage_c) {
                cerr << "Usage: MoveRep custId\n";
                exit (1);
        }

        // Catch database errors

        exec sql whenever sqlerror goto DbError;

        // Connect to the Customer Database

        exec sql connect to cust;

        // Validate the customer identifier and read the customer's
        // particulars.

        exec sql select custName, custTel
                into    :custName, :custTel
                from    custFile
                where   custId = :custId;
        if (SQLCODE != 0) {
                cerr << "Error: invalid customer identifier\n";
                exit (1);
        }

        // 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);

// Determine the date of the customer's first transaction
// on the database.

exec sql select min(tranDate)
        into    :firstDate indicator :firstDateInd
        from    tranFile
        where   tranCustId = :custId;
if (firstDateInd < 0) firstDate = genDate + 1;

// Initialise the period structures

for (i = 0; i < PERIOD_CNT; i++) {
        if (PERIOD_SIZE_ARR[i] != FULL_HISTORY)
                periodArr[i].prdSize = PERIOD_SIZE_ARR[i];
        else
                periodArr[i].prdSize = genDate - firstDate + 1;
}

// Process each of the customer's transactions

exec sql declare tranCur cursor for
        select  tranNo, tranDate
        from    tranFile
        where   tranCustId = :custId;
exec sql open tranCur;
for (;;) {
        exec sql fetch  tranCur
                into    :tranNo, :tranDate;
        if (SQLCODE != 0) break;

        // Calculate the transaction age

        tranAge = genDate - tranDate;

        // Initialise the vector of transaction categories

        tranCats.clear ();

        // Process each line item in the transaction

        exec sql declare itemCur cursor for
                select  itemServCode, itemAmtSold, itemCost
                from    itemFile
                where   itemTranNo = :tranNo;
        exec sql open itemCur;
        for (;;) {
                exec sql fetch  itemCur
                        into    :servCode, :amtSold, :cost;
                if (SQLCODE != 0) break;

                // Endeavour to find the service category of the
                // item in the service category map.  If the
                // service code is not in the map, load it from
                // the database.

                catMapIt = catMap.find(servCode);
                if (catMapIt != catMap.end()) {
                        strcpy (servCat, catMapIt->second.c_str());
                } else {
                        exec sql select servCat
                                into    :servCat
                                from    servFile
                                where   servCode = :servCode;
                        if (SQLCODE != 0) {
                                cerr << "Error: service code "
                                        << servCode << " not found\n";
                                exit (1);
                        }
                        catMap[servCode] = servCat;
                }

                // Add the category to the vector of transaction
                // categories.

                if (
                        find (tranCats.begin(), tranCats.end(),
                                servCat) == tranCats.end()
```

```
                        ) tranCats.push_back (servCat);

                        // Post the item to the periods

                        for (i = 0; i < PERIOD_CNT; i++) {
                                prd = periodArr + i;
                                if (tranAge < prd->prdSize) {
                                        catData = &prd->prdCatData[servCat];
                                        catData->catAmtSold += amtSold;
                                        catData->catCost += cost;
                                        prd->prdTotal.catAmtSold += amtSold;
                                        prd->prdTotal.catCost += cost;
                                }
                        }
                }
                exec sql close itemCur;

                // Increment the number of transactions

                for (i = 0; i < PERIOD_CNT; i++) {
                        prd = periodArr + i;
                        if (tranAge < prd->prdSize) {
                                for (j = 0; j < tranCats.size(); j++) {
                                        catData = &prd->prdCatData[tranCats[j]];
                                        catData->catTxCnt ++ ;
                                }
                                prd->prdTotal.catTxCnt ++ ;
                        }
                }
        }
        exec sql close tranCur;

        // Generate the report

        pageNo = 0;
        ShowPageTitle ();
        ShowCustInfo (custId, custName, custTel);
        ShowProfTitle ();
        for (i = 0; i < PERIOD_CNT; i++)
                ShowPeriod (periodArr + i);
        return 0;

        // Process database errors

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