

```

// RestRep.cpp - GENERATE THE REST BREAK PROFILE
//
// USAGE
// RestRep -D from-date to-date -C cashier-code
//
// -D from-date to-date    defines the reporting period as between
//                          from-date and to-date inclusive.
// -C cashier-code        sets the cashier code.
//
// MODULE INDEX
// NAME                    CONTENTS
// main                    Main line
//
// MAINTENANCE HISTORY
// DATE                    PROGRAMMER AND DETAILS
// 01-09-08 JS            Original
//
//-----

#include <cstdlib>          // C-style standard library
#include <cstring>          // C-style string manipulation functions
#include <cctype>           // C-style character typing functions
#include <iostream>         // C++ input/output streams
using namespace std;       // Expand the standard namespace
#include "dttime.h"        // Double time declarations
exec sql include sqlca;    // Include SQL communications area

//-----

// DEFINITIONS

static const size_t VERT_CNT = 20;
                        // Number of vertical graduations

//-----

// REST BREAK INTERVALS (MINUTES)

static const double INTER_ARR[] = {
    10.0, 14.0, 16.0, 18.0, 20.0, 22.0, 24.0, 26.0, 30.0, 60.0
};
static const size_t INTER_CNT = sizeof(INTER_ARR) / sizeof(INTER_ARR[0]);

//-----

// SCALES

static const size_t SCALE_ARR[] = {1, 2, 5};
static const size_t SCALE_CNT = sizeof(SCALE_ARR) / sizeof(SCALE_ARR[0]);

//-----

// MAIN LINE

int
main (
    int    argc,          // Argument count
    char   *argv[]       // Argument value pointers
)
{

```

```

Dtime_t    fromDate;    // From date
Dtime_t    toDate;     // To date
size_t     i, j;       // General purpose indices
int        c;          // Argument character
bool       periodDefined; // Reporting period defined flag
bool       cashierDefined; // Cashier defined flag
Dtime_t    prevCloseTime; // Previous shift close time
Dtime_t    openTime;    // Shift open time
Dtime_t    closeTime;  // Shift close time
Dtime_t    restBreakTime; // Rest break time
size_t     freq[INTER_CNT]; // Rest break frequencies
size_t     maxFreq;    // Maximum frequency
size_t     vertScale;  // Vertical scale
size_t     scaleMult;  // Scale multiplier
size_t     height;     // Height of current column
exec sql begin declare section;
    char    fromSqlDate[10+1]; // From SQL date
    char    toSqlDate[10+1];  // To SQL date
    char    openSqlTime[26+1]; // Shift open time
    char    closeSqlTime[26+1]; // Shift close time
    char    cashierCode[8+1];  // Cashier code
exec sql end declare section;

// Connect to the database

exec sql connect to restdb;

// Jump to DbError whenever an SQL error occurs

exec sql whenever sqlerror goto DbError;

// Decode arguments

periodDefined = 0;
cashierDefined = 0;
while ((c = getopt (argc, argv, "D:C:")) != -1) {
    switch (c) {
        case 'D':
            fromDate = DtimeFromUserDate (optarg);
            if (fromDate == NULL_DTIME) {
                cerr << "Error: bad from-date\n";
                exit (1);
            }
            SqlDateFromDtime (fromSqlDate, fromDate);
            if (optind >= argc) {
                cerr << "Error: missing to-date\n";
                exit (1);
            }
            toDate = DtimeFromUserDate (argv[optind]);
            if (toDate == NULL_DTIME) {
                cerr << "Error: bad to-date\n";
                exit (1);
            }
            SqlDateFromDtime (toSqlDate, toDate);
            optind ++ ;
            periodDefined = 1;
            break;
        case 'C':
            if (strlen(optarg) > 8) {

```

```

        cerr << "Error: bad cashier code\n";
        exit (1);
    }
    strcpy (cashierCode, optarg);
    cashierDefined = 1;
    break;
default:
    cerr << "Error: invalid option\n";
    exit (1);
    //NOTREACHED
}
}
if (optind < argc) {
    cerr << "Error: superfluous arguments\n";
    exit (1);
}

// If no reporting period was defined, it's a problem

if ( ! periodDefined) {
    cerr << "Error: no reporting period\n";
    exit (1);
}

// If no cashier was defined, it's a problem too

if ( ! cashierDefined) {
    cerr << "Error: no cashier code\n";
    exit (1);
}

// Reset the frequencies

for (i = 0; i < INTER_CNT; i++) freq[i] = 0;

// Select and order the applicable data

exec sql declare shiftCur cursor for
select  shiftOpenTime, shiftCloseTime
from    shifts
where   shiftDate between :fromSqlDate and :toSqlDate and
        cashierCode = :cashierCode
order  by shiftOpenTime;

// Initialise the previous shift close time

exec sql open shiftCur;
exec sql fetch  shiftCur
into    :openSqlTime, :closeSqlTime;
if (SQLCODE == 0) {
    prevCloseTime = DtimeFromSql (closeSqlTime);

    // Determine the rest breaks

    for (;;) {
        exec sql fetch  shiftCur
        into    :openSqlTime, :closeSqlTime;
        if (SQLCODE != 0) break;
    }
}

```

```

// Convert the SQL times into the internal format

openTime = DtimeFromSql (openSqlTime);
closeTime = DtimeFromSql (closeSqlTime);

// Calculate the duration of the rest break

restBreakTime = openTime - prevCloseTime;

// Determine the rest break interval

i = INTER_CNT;
while (i != 0 && restBreakTime <= INTER_ARR[i-1]*60.0)
    i -- ;

// Increment the frequency for the rest break interval

if (i < INTER_CNT) freq[i] ++ ;

// Update the previous close time

prevCloseTime = closeTime;
}
}
exec sql close shiftCur;

// Determine the vertical scale

maxFreq = 0;
for (i = 0; i < INTER_CNT; i++)
    if (maxFreq < freq[i]) maxFreq = freq[i];
i = 0;
scaleMult = 1;
for (;;) {
    vertScale = SCALE_ARR[i] * scaleMult;
    if ((maxFreq + vertScale/2)/vertScale <= VERT_CNT) break;
    i ++ ;
    if (i >= SCALE_CNT) {
        i = 0;
        scaleMult *= 10;
    }
}

// Emit the HTML document

cout << "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 3.0//EN\">";
cout << "<html>\n";
cout << "<head>\n";
cout << "<title>Rest Break Profile</title>\n";
cout << "</head>\n";
cout << "<body>\n";

// Define the style sheet for the table

cout << "<style type=\"text/css\">\n";
cout << "table.profile {\n";
cout << "\tbody>\n";
cout << "\tbody>\n";
cout << "\tbody>\n";

```

```

cout << "\tbodyborder-color: black;\n";
cout << "\tbodyborder-collapse: collapse;\n";
cout << "\tbodybackground-color: white;\n";
cout << "}\n";
cout << "table.profile td {\n";
cout << "\tbodyborder-width: 1pt;\n";
cout << "\tbodypadding: 1pt;\n";
cout << "\tbodyborder-style: inset;\n";
cout << "\tbodyborder-color: black;\n";
cout << "\tbodybackground-color: white;\n";
cout << "}\n";
cout << "table.profile td.blank {\n";
cout << "\tbodyborder-width: 1pt;\n";
cout << "\tbodypadding: 1pt;\n";
cout << "\tbodyborder-style: inset;\n";
cout << "\tbodyborder-color: black;\n";
cout << "\tbodybackground-color: white;\n";
cout << "\tbodywidth: 12pt;\n";
cout << "\tbodyheight: 12pt;\n";
cout << "}\n";
cout << "table.profile td.filled {\n";
cout << "\tbodyborder-width: 1pt;\n";
cout << "\tbodypadding: 1pt;\n";
cout << "\tbodyborder-style: inset;\n";
cout << "\tbodyborder-color: black;\n";
cout << "\tbodybackground-color: lightgray;\n";
cout << "\tbodywidth: 12pt;\n";
cout << "\tbodyheight: 12pt;\n";
cout << "}\n";
cout << "</style>\n";

// Generate the table

cout << "<table class=\"profile\">\n";
for (i = 0; i < VERT_CNT+1; i++) {
    cout << "<tr>\n";
    if (i == 0) {
        cout << "<td rowspan=\"" << (VERT_CNT+4)
            << "\">&nbsp;Frequency&nbsp;</td>\n";
    }
    if (i % 2 == 0) {
        cout << "<td rowspan=\"2\" align=\"right\">&nbsp;\"
            << ((VERT_CNT-i)*vertScale) << "&nbsp;</td>\n";
    }
    cout << "<td class=\"blank\"/>\n";
    for (j = 0; j < INTER_CNT; j++) {
        height = (freq[j] + vertScale/2)/vertScale;
        if (height >= VERT_CNT+1-i) {
            cout << "<td class=\"filled\"/>\n";
            cout << "<td class=\"filled\"/>\n";
        } else {
            cout << "<td class=\"blank\"/>\n";
            cout << "<td class=\"blank\"/>\n";
        }
    }
    cout << "<td class=\"blank\"/>\n";
    cout << "</tr>\n";
}
cout << "<tr>\n";

```

```

cout << "<td class=\"blank\"/>\n";
for (j = 0; j < INTER_CNT; j++) {
    cout << "<td class=\"blank\"/>\n";
    cout << "<td class=\"blank\"/>\n";
}
cout << "<td class=\"blank\"/>\n";
cout << "</tr>\n";
cout << "<tr>\n";
cout << "<td/>\n";
cout << "<td align=\"center\" colspan=\"2\">0</td>\n";
for (j = 0; j < INTER_CNT; j++) {
    cout << "<td align=\"center\" colspan=\"2\">"
        << INTER_ARR[j] << "</td>\n";
}
cout << "</tr>\n";
cout << "<tr>\n";
cout << "<td/>\n";
cout << "<td align=\"center\" colspan=\"" << (INTER_CNT*2+2)
    << "\">Rest Break Duration (minutes)</td>\n";
cout << "</tr>\n";
cout << "</table>\n";
cout << "</body>\n";
cout << "</html>\n";
return 0;

```

```

// Process database errors

```

```

DbError:

```

```

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

```

```

}

```