

cqctest.cpp

```

// cqctest.cpp - TEST THE CUSTOMER QUERY CACHE
//
// MODULE INDEX
// NAME                CONTENTS
// InsertCust          Insert a pseudo-random record into the cache
// RemoveCust          Remove a pseudo-randomly selected record from the cache
// Verify              Verify the contents of the control cache
// main                Main line
//
// MAINTENANCE HISTORY
// DATE                PROGRAMMER AND DETAILS
// 25-09-10           JS          Original
//
//-----

#include <cstdlib>           // C-style standard library
#include <cstring>          // C-style string manipulation functions
#include <string>            // C++ string declarations
#include <set>               // C++ set declarations
#include <map>               // C++ map declarations
#include <vector>            // C++ vector declarations
#include <iostream>         // C++ I/O stream declarations
using namespace std;       // Expand the standard namespace
#include "Cqc.h"            // Customer query cache declarations

//-----

// CONTROL MAP DECLARATION

typedef map<long,string> CtrlMap_t;           // Control map type definition
typedef CtrlMap_t::iterator CtrlIter_t;     // Control iterator type definition

//-----

// GLOBAL DATA

Cqc_c          cqc;           // Customer query cache instance
CtrlMap_t      ctrlMap;      // Control map

//-----

// INSERT A PSEUDO-RANDOM RECORD INTO THE CACHE

void
InsertCust ()
{
    Cust_t      cust;         // Customer record
    size_t      nameLen;     // Name length
    long        randVal;     // Pseudo-random value
    char        nameChar;    // Customer name character

    // Select a customer identifier that is not already in the cache

    do {
        cust.custId = lrand48();
    } while (ctrlMap.count(cust.custId) != 0);

    // Generate a pseudo-random customer name

    nameLen = lrand48() % 40 + 1;
    while (cust.custName.length() < nameLen) {

```

cqctest.cpp

```
        randVal = lrand48() % ('Z' - 'A' + 2);
        if (randVal < 'Z' - 'A' + 1)
            nameChar = randVal + 'A';
        else
            nameChar = ' ';
        cust.custName += nameChar;
    }

    // Add the customer to the cache

    cq.c.CqcInsert (&cust);

    // Add the customer to the control map

    ctrlMap[cust.custId] = cust.custName;
}

//-----

// REMOVE A PSEUDO-RANDOMLY SELECTED RECORD FROM THE CACHE

void
RemoveCust ()
{
    size_t          targetRecNo;    // Deletion target record number
    size_t          i;              // General purpose index
    CtrlIter_t      targetIter;     // Deletion target iterator

    // If the cache is empty, we cannot delete anything

    if (ctrlMap.size() == 0) return;

    // Select the deletion target

    targetRecNo = lrand48() % ctrlMap.size();
    targetIter = ctrlMap.begin ();
    for (i = 0; i < targetRecNo; i++) targetIter ++ ;

    // Remove the record from the cache

    cq.c.CqcRemove (targetIter->first);

    // Delete the record from the control map

    ctrlMap.erase (targetIter);
}

//-----

// VERIFY THE CONTENTS OF THE CONTROL CACHE

void
Verify ()
{
    const Cust_t    *cust;          // Customer pointer
    CtrlMap_t       contMap;        // Cache contents map
    CtrlIter_t      contIter;       // Contents iterator
    CtrlIter_t      ctrlIter;      // Control iterator

    // Load the contents of the cache into a contents map

    cq.c.CqcSelect ("*");
    while ((cust = cq.c.CqcNextCust()) != 0) {
```

cqctest.cpp

```
// Check that the customer identifier is not already
// in the contents map

if (contMap.find (cust->custId) != contMap.end()) {
    cerr << "Verify: error: custId already extracted\n";
    exit (1);
}

// Add the customer to the map

contMap[cust->custId] = cust->custName;
}

// Check that the contents map contains the same information
// as the control map

contIter = contMap.begin ();
ctrlIter = ctrlMap.begin ();
while (contIter != contMap.end() && ctrlIter != ctrlMap.end()) {
    if (contIter->first != ctrlIter->first) {
        cerr << "Verify: error: custId mismatch\n";
        exit (1);
    }
    if (contIter->second != ctrlIter->second) {
        cerr << "Verify: error: custName mismatch\n";
        exit (1);
    }
    contIter++;
    ctrlIter++;
}
if ((contIter != contMap.end()) != (ctrlIter != ctrlMap.end())) {
    cerr << "Verify: error: end-of-file mismatch\n";
    exit (1);
}
}

//-----

// MAIN LINE

int
main ()
{
    size_t          i;                // General purpose index

    // Execute verification cycles indefinitely until it finds the fault

    for (;;) {

        // Execute 1,000 update cycles

        for (i = 0; i < 1000; i++) {

            // 60% of the time insert a row and 40% of the
            // time remove a row

            if (lrand48() % 100 < 60)
                InsertCust ();
            else
                RemoveCust ();

        }
    }
}
```

```
        // Verify the contents of the cache  
        Verify ();  
    }  
    return 0;  
}
```