

```

// GenSample.cpp - GENERATE SAMPLE DATABASE
//
// MODULE INDEX
// NAME                CONTENTS
// ErrFun              Database error handling function
// GenName             Generate customer name
// Main               Main line
//
// MAINTENANCE HISTORY
// DATE              PROGRAMMER AND DETAILS
// 02-09-04        BYG          Original (Adapted from GenMb.cpp)
//
//-----

#include <stdio.h>           // Standard input/output declarations
#include <string.h>         // String manipulation functions
#include <stdlib.h>         // Standard library
#include <math.h>           // Mathematical functions
#include <sqlca.h>          // SQLCA structure definition
#include <sqlenv.h>         // DB2 environment definition
#include <sqlutil.h>        // DB2 sql utilities

exec sql include sqlca;

//-----

// GLOBAL DEFINITIONS

#define DB_NAME          "custDb"           // Database name
#define N_PURCHASES     10000             // Number of purchases
#define N_CUST          1000              // Number of customers

int count;

exec sql begin declare section;
    char    dbNam[10];           // Database name
    long    custId;             // Customer ID
    char    custName[41];       // Customer name
    long    purId;              // Purchase ID
    long    purNo;              // Purchase number
    long    purVal;             // Purchase value
    long    custCnt;            // Customer count
    long    purCnt;             // Purchase count
exec sql end declare section;

//-----

// DATABASE ERROR HANDLING FUNCTION

void
ErrFun ()
{
    exec sql rollback work;
    printf ("\nDB error SQLCODE = %d\t", SQLCODE);
    if (sqlca.sqlerrml != 0)
        printf ("/%s\n", sqlca.sqlerrmc);
    exit (1);
}

//-----

// GENERATE CUSTOMER NAME

void
GenName (
    char    *name)           // Name buffer
{
    char    nam[41];         // Name
    int     namLen;          // Name length
    int     i, j;            // General purpose indices

    // Generate last name

    namLen = (int)(4 + rand() % 3);
    for (i=0; i<namLen; i++)
        nam[i] = (char)('A' + rand() % ('Z'-'A'+1));
    nam[i++] = ' ';

    // Generate middle name

    namLen = (int)(4 + rand() % 3);
    for (j=i; j<i+namLen; j++)

```

```

        nam[j] = (char)('A' + rand() % ('Z'-'A'+1));
        nam[j++] = ' ';

        // Generate last name

        namLen = (int)(4 + rand() % 3);
        for (i=j; i<j+namLen; i++)
            nam[i] = (char)('A' + rand() % ('Z'-'A'+1));
        nam[i] = '\0';

        // Load the name buffer

        sprintf (name, "%s", nam);
    }

//-----

// MAIN LINE

int
main ()
{
    char    cml[128];           // Command line
    long    custIdArr[N_CUST];  // Customer ID array
    int     i;                 // General purpose index
    long    v;                 // General purpose variable

    // Initiate database error handling

    // exec sql whenever sqlerror goto :db_error;

    // Seed the random number generator

    srand (121);

    // Create Database and Tables
    // Create using script

    // Connect to database

    strcpy (dbNam, DB_NAME);
    exec sql connect to :dbNam;
    if (SQLCODE != 0) {
        // Create database

        printf ("Creating database: %s\n", dbNam);
        sqlecrea (dbNam, 0, 0, 0, 0, 0, 0, &sqlca);
        if (SQLCODE != 0) goto db_error;
        memset (cml, 0, sizeof (cml));
        strcpy (cml, "bind.bat GenSample.bnd ");
        strcat (cml, dbNam);
        system (cml);
        printf ("Database creation complete\n");
        exec sql connect to :dbNam;
    }

    // Unconditionally drop tables

    exec sql drop table custFile;
    exec sql drop table purFile;
    exec sql commit work;

    // Create tables and indexes

    exec sql create table custFile (
        custId          integer not null,
        custName        char(40) not null
    );
    exec sql create table purFile (
        purId           integer not null,
        purNo           integer not null,
        purVal          integer not null
    );
    exec sql commit work;

    // Generate customers

    printf ("\nGenerating customers...");

    for (i=0; i<N_CUST; i++) {
        custIdArr[i] = custId = 100000 + i;
        GenName (custName);
    }
}

```

```

        // Store customer
        exec sql insert into custFile (
            custId, custName
        ) values (
            :custId, :custName
        );
    }
    printf ("\nCommit customers");
    exec sql commit work;
    exec sql connect reset;
    exec sql connect to :dbNam;

    // Simulate purchases
    printf ("\nGenerating purchases...\n");

    for (count=0, i=0; i<N_PURCHASES; count++, i++) {
        purNo = 10000000 + i;
        purId = custIdArr[(long)(rand() % N_CUST)];
        purVal = (long)(rand ());

        // Store the purchase data
        exec sql insert into purFile (
            purId, purNo, purVal
        ) values (
            :purId, :purNo, :purVal
        );

        if ((i % 100) == 99) printf ("%d records generated\n", i+1);
        if ((i % 5000) == 4999) {
            printf ("Performing partial commit\n");
            exec sql commit work;
            exec sql connect reset;
            exec sql connect to :dbNam;
        }
    }

    exec sql select count(*) into :custCnt from custFile;
    exec sql select count(*) into :purCnt from purFile;
    printf ("Customers: %d\n", custCnt);
    printf ("Purchases: %d\n", purCnt);

    // Commit final database changes
    printf ("\nFinal commit...");
    exec sql commit work;
    exec sql connect reset;

    // Exit gracefully
    printf ("\nDone!\n");

    return 0;

db_error:
    ErrFun();
}

```