

```

// GenMb.cpp - GENERATE MICROBANK LOAN DATA
//
// MODULE INDEX
// NAME                CONTENTS
// ErrFun              Database error handling function
// GenName             Generate executive or customer name
// Main               Main line
//
// MAINTENANCE HISTORY
// DATE                PROGRAMMER AND DETAILS
// 03-11-03           PFT      Original
// 20-08-04           BYG      DB2, Windows2000 and MS VC++ support
//
//-----
#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          "microDb"      // Database name
#define MIN_AMT          100            // Minimum loan amount ($)
#define MAX_AMT          1000          // Maximum loan amount ($)
#define DLR_SCALE        100.0         // Dollar scale
#define N_LOANS          1000000       // Number of loans
#define N_EXEC           1000          // Number of executives
#define DEL_PROB         0.01          // Executive deletion probability
#define NP_LOAN_PROB     0.30          // Non-performing loan probability

int count;

exec sql begin declare section;
    char    dbNam[10];                // Database name
    long    execCode;                 // Executive code
    char    execName[21];             // Executive name
    long    loanId;                   // Loan Id
    char    loanCust[21];             // Customer name
    double  loanAmount;               // Loan amount in cents
    short   loanStatus;               // Loan status
    long    fstExecCode;               // 1st executive code
    short   fstExecAppType;           // 1st executive approver type
    long    secExecCode;              // 2nd executive code
    short   secExecAppType;           // 2nd executive approver type
    long    execCnt;                  // Executive count
    long    loanCnt;                  // Loan count
    long    appsCnt;                  // Approval count
exec sql end declare section;

//-----

// EXECUTIVE DATA STRUCTURE

typedef struct {
    int     execCode;                 // Executive code
    short   execDel;                 // Deleted flag
} Exec_t;

//-----

// 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);

    printf ("LoanId: %d\n", loanId);
    printf ("LoanCust: %s\n", loanCust);
    printf ("LoanAmount: %f\n", loanAmount);
}

```

```

printf ("LoanStatus: %d\n", loanStatus);
printf ("FstExecCode: %d\n", fstExecCode);
printf ("FstExecAppType: %d\n", fstExecAppType);
printf ("SecExecCode: %d\n", secExecCode);
printf ("SecExecAppType: %d\n", secExecAppType);
exit (1);
}

//-----

// GENERATE EXECUTIVE/CUSTOMER NAME

void
GenName (
char          *name)          // Name buffer
{
char          nam[21];        // 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++] = '\0';

// 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++] = '\0';

// 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
Exec_t        execArr[N_EXEC]; // Executive array
short         delFlag;       // Deleted flag
int           fstExecInd;    // 1st executive index
int           secExecInd;    // 2nd executive index
int           i;             // General purpose index
double        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);
    sqlcrea (dbNam, 0, 0, 0, 0, 0, 0, &sqlca);
    if (SQLCODE != 0) goto db_error;
}
}

```

```

        memset (cml, 0, sizeof (cml));
        strcpy (cml, "bind.bat GenMb.bnd ");
        strcat (cml, dbName);
        system (cml);
        printf ("Database creation complete\n");
        exec sql connect to :dbName;
    }

// Unconditionally drop tables

exec sql drop table executives;
exec sql drop table loans;
exec sql drop table approvals;
exec sql commit work;

// Create tables and indexes

exec sql create table executives (
        execCode      integer not null,
        execName      char(20) not null
);
exec sql create table loans (
        loanId        integer not null,
        loanCust      char(20) not null,
        loanAmount    double precision not null,
        loanStatus    smallint not null
);
exec sql create table approvals (
        appLoanId     integer not null,
        appExecCode   integer not null,
        appType       smallint not null
);
exec sql create index appLoanIdInd on approvals (appLoanId);
exec sql commit work;

// Generate executives

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

for (i=0; i<N_EXEC; i++) {
    execArr[i].execCode = execCode = 100000 + i;
    GenName (execName);
    execArr[i].execDel = delFlag = ((rand()%100)/100.00) < DEL_PROB;

    // Store undeleted executives

    if (! delFlag) {
        exec sql insert into executives (
                execCode, execName
            ) values (
                :execCode, :execName
            );
    }
}
printf ("\nCommit executives");
exec sql commit work;
exec sql connect reset;
exec sql connect to :dbName;

// Simulate loans

printf ("\nGenerating loans and approvals...\n");

for (count=0, i=0; i<N_LOANS; count++, i++) {
    loanId = 10000000 + i;
    GenName (loanCust);
    v = MIN_AMT + rand () * (MAX_AMT - MIN_AMT) * DLR_SCALE;
    modf (v, &loanAmount);
    loanStatus = (double)((rand()%100)/100.00)<NP_LOAN_PROB ? 0 : 1;

    // Store the loan data

    exec sql insert into loans (
        loanId, loanCust, loanAmount, loanStatus
    ) values (
        :loanId, :loanCust, :loanAmount, :loanStatus
    );

    // Store approval data for undeleted executives

    fstExecInd = (int)(rand() % N_EXEC);
    fstExecCode = execArr[fstExecInd].execCode;
}

```

```

fstExecAppType = (double)((rand ()%100)/100.00) < 0.5 ? 1 : 0;
if (! execArr[fstExecInd].execDel) {
    exec sql insert into approvals (
        appLoanId, appExecCode, appType
    ) values (
        :loanId, :fstExecCode, :fstExecAppType
    );
}

do {
    secExecInd = (int)(rand() % N_EXEC);
} while (secExecInd == fstExecInd);
secExecCode = execArr[secExecInd].execCode;
secExecAppType = fstExecAppType == 1 ? 0 : 1;
if (! execArr[secExecInd].execDel) {
    exec sql insert into approvals (
        appLoanId, appExecCode, appType
    ) values (
        :loanId, :secExecCode, :secExecAppType
    );
}
if ((i % 100) == 99) printf ("%d records\n", i+1);
if ((i % 5000) == 4999) {
    printf ("Perform partial commit\n");
    exec sql commit work;
    exec sql connect reset;
    exec sql connect to :dbNam;
}

}

exec sql select count(*) into :execCnt from executives;
exec sql select count(*) into :loanCnt from loans;
exec sql select count(*) into :appsCnt from approvals;
printf ("Executives: %d\n", execCnt);
printf ("Loans: %d\n", loanCnt);
printf ("Approvals: %d\n", appsCnt);

// Commit database changes

printf ("\nCommit work...");
exec sql commit work;
exec sql connect reset;

// Exit gracefully

printf ("\nDone!\n");

return 0;

db_error:
    ErrFun();
}

```