```
// TicketGen.cpp - TICKET DATABASE GENERATOR
//
// MODULE INDEX
// NAME             CONTENTS
// main             Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 24-09-12 JS  Original
//
//-----------------------------------------------------------------------

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

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

// DEFINITIONS

static const char   FROM_DATE[] = "2011-01-01";
                // From date
static const char   TO_DATE[] = "2012-10-30";
                // Last date

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

// TICKET STATUS CODES

exec sql begin declare section;
    static const short TICKET_STATUS_ISSUED = 1;
                // The ticket has been issued by a ride,
                // but not yet redeemed
    static const short TICKET_STATUS_ACCEPTED = 2;
                // The ticket has been accepted by a ride
    static const short TICKET_STATUS_CASHED = 3;
                // The ticket has been cashed at an ATM
    static const short TICKET_STATUS_CANCELLED = 4;
                // The ticket has been cancelled
    static const short TICKET_STATUS_EXPIRED = 5;
                // The ticket has expired
exec sql end declare section;

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

// TRANSACTION TYPE CODES

exec sql begin declare section;
    static const short TX_TYPE_ISSUE = 1;
                // Issue by a ride
    static const short TX_TYPE_ACCEPT = 2;
                // Acceptance by a ride
    static const short TX_TYPE_CASH = 3;
```

```
                // Encashment at an ATM
    static const short TX_TYPE_CANCEL = 4;
                // Cancellation
    static const short TX_TYPE_EXPIRE = 5;
                // Expiry
exec sql end declare section;

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

// ACTIVE TICKET STRUCTURE

struct Ticket_t {
    long        ticketNo;   // Ticket number
    Dtime_t     issueDate;  // Issue date
};

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

// MAIN LINE

int
main ()
{
    size_t      i;          // General purpose index
    Dtime_t     tradingDate;    // Current trading day date
    Dtime_t     fromDate;   // From-date
    Dtime_t     toDate;     // To-date
    Dtime_t     txTime;     // Transaction time
    Ticket_t    ticket;     // Ticket structure
    vector<Ticket_t> activeTickets; // Active ticket set
    long        txTypeRand; // Transaction type random number
    Dtime_t     expiryThreshold; // Expiry threshold
    long        txCnt;      // Transaction count

    exec sql begin declare section;
        char        sqlTradingDate[10+1]; // Trading day date
        char        sqlTxTime[26+1]; // Transaction time
        long        nextTicketNo;   // Next ticket number
        long        ticketNo;   // Ticket number
        long        nextTxNo;   // Next transaction number
        long        txNo;       // Transaction number
        double      ticketValue;    // Ticket value
        long        locationId; // Location identifier
    exec sql end declare section;

    // Connect to the database

    exec sql connect to ticketdb;

    // Drop tables

    exec sql whenever sqlerror continue;
    exec sql drop table dayStatus;
    exec sql drop table tickets;
    exec sql drop table transactions;
    exec sql commit work;
```

```
    // Jump to DbError whenever an SQL error occurs

    exec sql whenever sqlerror goto DbError;

    // Create the database tables

    exec sql create table dayStatus (
        tradingDate date not null,
        dayOpen     smallint not null
    );
    exec sql create table tickets (
        ticketNo    integer not null,
        ticketStatus    smallint not null,
        ticketValue double precision not null
    );
    exec sql create unique index ticketNoInd on
        tickets (ticketNo);
    exec sql create table transactions (
        txNo        integer not null,
        ticketNo    integer not null,
        locationId  integer not null,
        txTime      timestamp not null,
        txDate      date not null,
        txType      smallint not null
    );
    exec sql create index txDateInd on
        transactions (txDate);

    // Initialise the random number generator

    srand48 (20360L);

    // Initialise the from and to-dates

    fromDate = DtimeFromSql (FROM_DATE);
    toDate = DtimeFromSql (TO_DATE);

    // Initialise the next ticket and transaction numbers

    nextTicketNo = 1;
    nextTxNo = 1;

    // Process each trading day date in the sequence

    for (
        tradingDate = fromDate;
        tradingDate <= toDate;
        tradingDate += 24.0*60.0*60.0
    ) {
        // Load the SQL date

        SqlDateFromDtime (sqlTradingDate, tradingDate);

        // Expire out-of-date tickets

        expiryThreshold = tradingDate - 183.0*24.0*60.0*60.0;
        i = 0;
```

```
        while (i < activeTickets.size()) {
            if (activeTickets[i].issueDate < expiryThreshold) {
                ticketNo = activeTickets[i].ticketNo;
                activeTickets.erase (activeTickets.begin()+i);
                txNo = nextTxNo ++ ;
                locationId = lrand48() % 200;
                ticketValue = lrand48() % 10000 + 5;
                txTime = tradingDate+10.0*60.0*60.0;
                SqlTimestampFromDtime (sqlTxTime, txTime);
                exec sql insert into tickets (
                    ticketNo, ticketStatus,
                    ticketValue
                ) values (
                    :ticketNo, :TICKET_STATUS_EXPIRED,
                    :ticketValue
                );
                exec sql insert into transactions (
                    txNo, ticketNo, locationId,
                    txTime, txDate,
                    txType
                ) values (
                    :txNo, :ticketNo, :locationId,
                    :sqlTxTime, :sqlTradingDate,
                    :TX_TYPE_EXPIRE
                );
            } else {
                i ++ ;
            }
        }

        // Process transaction times

        txCnt = 0;
        for (
            txTime = tradingDate+10.0*60.0*60.0 + lrand48()%30 + 1;
            txTime < tradingDate+26.0*30.0*60.0;
            txTime += lrand48() % 30 + 1
        ) {
            // Load the SQL time

            SqlTimestampFromDtime (sqlTxTime, txTime);

            // Load the location identifier

            locationId = lrand48() % 200;

            // 50-50 chance of a ticket issue

            txTypeRand = lrand48() % 200;
            if (txTypeRand < 100 || activeTickets.size() == 0) {
                ticketNo = nextTicketNo ++ ;
                txNo = nextTxNo ++ ;
                ticket.ticketNo = ticketNo;
                ticket.issueDate = tradingDate;
                activeTickets.push_back (ticket);
                exec sql insert into transactions (
                    txNo, ticketNo, locationId,
```

```
            txTime, txDate,
            txType
        ) values (
            :txNo, :ticketNo, :locationId,
            :sqlTxTime, :sqlTradingDate,
            :TX_TYPE_ISSUE
        );
    }

    // Of the remainder, 75% of the transactions
    // are ticket redemptions at rides

    else if (txTypeRand < 175) {
        i = lrand48() % activeTickets.size();
        ticketNo = activeTickets[i].ticketNo;
        activeTickets.erase (activeTickets.begin()+i);
        txNo = nextTxNo ++ ;
        ticketValue = lrand48() % 10000 + 5;
        exec sql insert into tickets (
            ticketNo, ticketStatus,
            ticketValue
        ) values (
            :ticketNo, :TICKET_STATUS_ACCEPTED,
            :ticketValue
        );
        exec sql insert into transactions (
            txNo, ticketNo, locationId,
            txTime, txDate,
            txType
        ) values (
            :txNo, :ticketNo, :locationId,
            :sqlTxTime, :sqlTradingDate,
            :TX_TYPE_ACCEPT
        );
    }

    // 15% are cashed

    else if (txTypeRand < 190) {
        i = lrand48() % activeTickets.size();
        ticketNo = activeTickets[i].ticketNo;
        activeTickets.erase (activeTickets.begin()+i);
        txNo = nextTxNo ++ ;
        ticketValue = lrand48() % 10000 + 5;
        exec sql insert into tickets (
            ticketNo, ticketStatus,
            ticketValue
        ) values (
            :ticketNo, :TICKET_STATUS_CASHED,
            :ticketValue
        );
        exec sql insert into transactions (
            txNo, ticketNo, locationId,
            txTime, txDate,
            txType
        ) values (
            :txNo, :ticketNo, :locationId,
```

```
                :sqlTxTime, :sqlTradingDate,
                :TX_TYPE_CASH
            );
        }

        // 1% are cancelled

        else if (txTypeRand < 191) {
            i = lrand48() % activeTickets.size();
            ticketNo = activeTickets[i].ticketNo;
            activeTickets.erase (activeTickets.begin()+i);
            txNo = nextTxNo ++ ;
            ticketValue = lrand48() % 10000 + 5;
            exec sql insert into tickets (
                ticketNo, ticketStatus,
                ticketValue
            ) values (
                :ticketNo, :TICKET_STATUS_CANCELLED,
                :ticketValue
            );
            exec sql insert into transactions (
                txNo, ticketNo, locationId,
                txTime, txDate,
                txType
            ) values (
                :txNo, :ticketNo, :locationId,
                :sqlTxTime, :sqlTradingDate,
                :TX_TYPE_CANCEL
            );
        }

        // Commit work every 1000 transactions

        txCnt ++ ;
        if (txCnt >= 1000) {
            exec sql commit work;
            txCnt = 0;
        }
    }

    // Execute a commit at the end of each trading day

    if (txCnt != 0) exec sql commit work;
}

// Write the active tickets

for (i = 0; i < activeTickets.size(); i++) {
    ticketNo = activeTickets[i].ticketNo;
    ticketValue = lrand48() % 10000 + 5;
    exec sql insert into tickets (
        ticketNo, ticketStatus,
        ticketValue
    ) values (
        :ticketNo, :TICKET_STATUS_ISSUED,
        :ticketValue
    );
```

```
    }

    // Write the day status table

    exec sql insert into dayStatus (
        tradingDate, dayOpen
    ) values (
        :sqlTradingDate, 1
    );

    // Commit the changes

    exec sql commit work;

    // And that's all

    return 0;

    // Process database errors

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