Solving Load Testing Data Issues With The Virtual Table Server

Otto Strickland
Senior Consultant

ZoneTek, Inc.
Tampa, FL

Dilemma

- Your LoadRunner scripts are complete but issues arise running the scenario
  - Lack of sufficient volumes of data
  - LoadRunner parameterization selection methods do not provide enough data tracking flexibility
  - Concurrency execution issues
- Given the circumstances above, what next?
  - Virtual Table Server (VTS)

Topics Covered This Session

- Virtual Table Server
  - Definition
  - Setup Installation
  - Basic function calls and code examples
- Using unique data across multiple vuser scripts and scenario groups
- Implementing data loads with VTS
- Real time vuser data sharing during load test
- Creating a critical section within a script

Virtual Table Server

- Stores and manipulates data items in dynamic queues
- Operations performed on data items are atomic
- Simultaneous requests are serialized
- Queues (lists) are stored in columns (queue name)

VTS Setup/Installation

- Setup and installation of VTS is not covered in this session
- Setup/Installation documentation and instructions are located in the knowledge base of Mercury Interactive's support website

VTS Basic Operations

- Connect to VTS
- Insert data item to end of column
- Insert unique data item to end column
- Retrieve a data item from beginning of a column
- Empty the contents of a column
- Disconnect from VTS

- Note:
  - load vtclient.dll in scripts to access VTS exported functions

Connect to Server

PCVI vtc_connect(char *pszServer, int iPort, VTP_KEEP_ALIVE)

pszServer – Name of host (ie: LRHost1, 10.192.3.25, Mercury.lrhost1.com)
iPort – Port number (ie: default is 8888, 8887, etc.)

PCVI – VTS descriptor

Code Sample

```
#include "vts2.h"

PVCI pVTS = 0;


vuser_init()
{//Must load vtclient.dll to have access to VTS exported functions
    if (lr_load_dll("vtclient.dll") != LR_PASS)
    {
        lr_error_message("vtserver failed to load");
        return -1;
    }

    pVTS = vtc_connect("10.190.180.78", 8888, VTOPT_KEEP_ALIVE);
```

Insert Data Item

int vtc_send_message(PCVI pVTS, char *pszColumn, char *pszData, unsigned short *uStatus)

pVTS – server descriptor

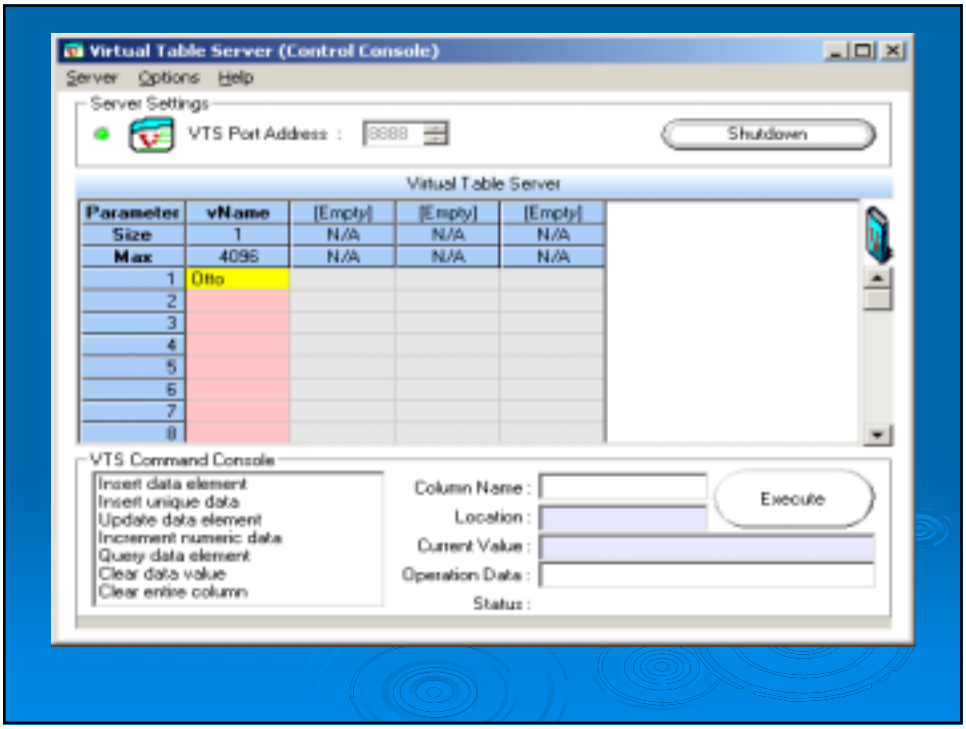
pszColumn – column (queue) name

pszData – data item to be inserted into queue

uStatus – status of operation (1 – successful, 0 – failed)

Code Sample

```
int InsertDataItem(char *pszColumn, char *pszItem)
{
	unsigned short uStatus;
	int iRC;

	iRC = vtc_send_message(pVTS, pszColumn, pszItem, &uStatus);
	if (iRC != VTCERR_OK || uStatus == 0)
	{
		lr_error_message("vtc_send_message error:[%d] Status:[%d]", iRC,
				uStatus);

		return LR_FAIL;
	}

	return LR_PASS;
}
Actions()
{
	if (InsertDataItem("vName", "Otto") != LR_PASS)
	{
		return -1;
	}
```



Insert Unique Data Item

int vtc_send_if_unique(PVCI pVTS, char *pszColumn, char *pszData, unsigned short *uStatus)

pVTS – server descriptor

pszColumn – column (queue) name

pszData – data item to be inserted into queue

uStatus – status of operation (1 – successful, 0 – failed)

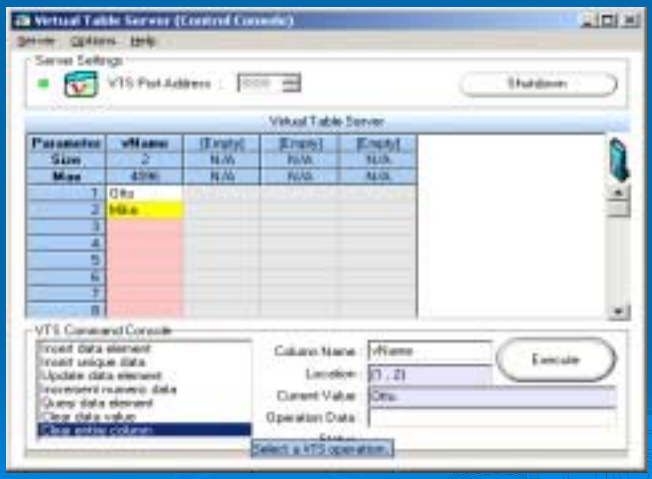
Code Sample

```
#define LR_NOT_UNIQUE                    (LR_FAIL+ 0x0F)

int InsertUniqueItem(char *pszColumn, char *pszItem)
{
      unsigned short uStatus;
      int iRC;

      iRC = vtc_send_if_unique(pVTS, pszColumn, pszItem, &uStatus);
      if (iRC != VTCERR_OK)
      {
            lr_error_message("vtc_send_if_unique error:[%d] Status:[%d]",
                  iRC, uStatus);
            return LR_FAIL;
      }
      if (uStatus == 0)
      {
            return LR_NOT_UNIQUE;
      }
      return LR_PASS;
}
```

Code Sample

```
Actions()
{
    if (InsertUniqueItem("vName", "Mike") != LR_PASS)
    {
          return -1;
    }
    if (InsertUniqueItem("vName", "Otto") != LR_PASS)
    {
          return -1;
    }
```

InsertUniqueItem did not allow Otto to be added twice to the column



Retrieve Data Item

int vtc_retrieve_message(PCVI pVTS, char *pszColumn, char *pszData)

pVTS – server descriptor
pszColumn – column (queue) name
pszData – buffer to store retrieved data item

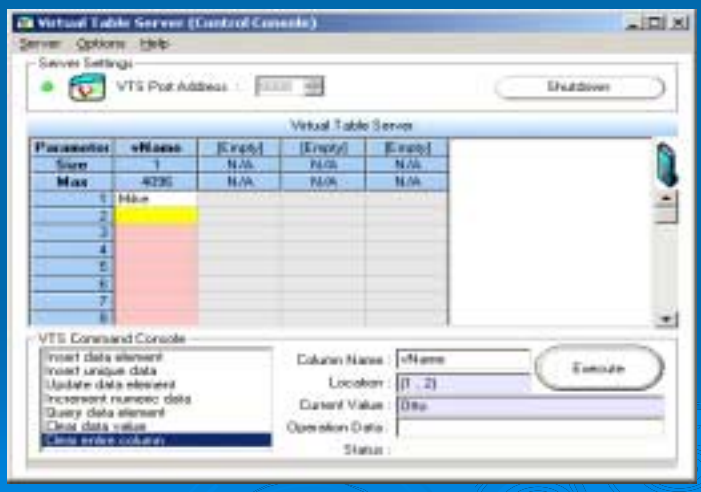
Code Sample

```
int GetItem(char *pszColumn, char *pszItem)
{
	int iRC;
	char  *pszValue = NULL;

	iRC = vtc_retrieve_message(pVTS, pszColumn, &pszValue);
	switch(iRC)
	{
		case VTCERR_OK:
			strcpy(pszItem, pszValue);
			vtc_free(pszValue);
			iRC = LR_PASS;
		case VTCERR_RESPONSE_ARGS_UNMATCH:
			return iRC;
			break;

		default:
			lr_error_message("vtc_retrieve_message error:[%d]", iRC);
			break;
	}
	return LR_FAIL;
}
```

Code Sample

```
Actions()
{
    char szItem[32];

    if (GetItem("vName", szItem) != LR_PASS)
    {
        return -1;
    }
    lr_save_string(szItem, "pParam");

    return 0;
}
```

Actions.c(9): Saving Parameter "pParam = Otto"



Empty Column contents

int vtc_clear_column(PCVI pVTS, char *pszColumn, unsigned short *uStatus)

pVTS – VTS descriptor
pszColumn – column (queue) name
uStatus – status of operation (1 – successful, 0 – failed)

Code Sample

```
#define LR_COLUMN_NOT_EXIST       (LR_FAIL + 0x1F)
int EmptyColumn(char *pszColumn)
{
        int iRC;
        unsigned short uStatus;

        iRC = vtc_clear_column(pVTS, pszColumn, &uStatus);
        if (iRC != VTCERR_OK)
        {
                lr_error_message("vtc_clear_column error:[%d] Status:[%d]", iRC, uStatus);
                return LR_FAIL;
        }
        if (uStatus == 0)
                return LR_COLUMN_NOT_EXIST;

        return LR_PASS;
}

Vuser_init()
{
        :
        if (EmptyColumn("vName") != LR_PASS)
        {
                return -1;
        }
        :
```

Disconnect from VTS

int vtc_disconnect(PCVI pVTS)

pVTS – VTS descriptor

Code Sample

```
vuser_end()
{
   vtc_disconnect(pVTS);
}
```

Using Unique Data Across Multiple Scripts and Scenario Groups

- Technique 1
- Technique 2
- VTS Technique

DB Sample

Action Section

```
    .
    .
lrd_open_cursor(&Csr5, Con1, 0);
  lrd_stmt(Csr5, "UPDATE Orders SET flight_number=<dFlightNo>,
    customer_no=<dCustomerNo>, "
     "agent_no=107,tickets_ordered=<pTickets>, class='<dClass>',
    send_signature_with_order="
     "'<dSignature>', departure_date={d '<dDepartureDate>'} WHERE
    order_number=<pOrderNumber>", -1, 1,
     0, 0);
  lrd_close_cursor(&Csr5, 0);
    .
    .
```

Note: Order Number must be unique when executing scenario

Technique 1

- Select a parameter type of File with a selection method of unique
- Unselect Advance row each iteration (forces vuser to use only one data item)
- Note: data items count greater than or equal to vuser count

Technique 1

- Advantages
  - Easy (built in functionality)
  - 1 data item per vuser
  - Run unlimited number of iterations (assuming no failures)
- Disadvantages
  - Loss of data item if stopped, failed, or passed vuser requires restart
  - Each vuser uses the same data item every iteration
  - Difficult if using same data across multiple vuser scripts (requires maintaining uniqueness across multiple data files)

Technique 2

- Select parameter type of File with a selection method of unique
- Select Advance row each iteration
- Data block size equals iteration count of the vuser's group in the runtime settings
  - Data item count is greater than or equal to vuser count * iteration count
  - Multiple data block sizes when script is a member of different groups with different iteration counts

Technique 2

Technique 2

- Advantages
  - Easy (built in functionality)
- Disadvantages
  - Difficult when using same script across multiple groups in a scenario when each group has a different iteration setting (data block size) for the script
  - Loss of block of data if stopped, failed, or passed vuser requires restart

VTS Technique

- Convert existing scripts that use technique 1 and 2 to read data from VTS

VTS Technique

```
#include "vts2.h"

PVCI pVTS = 0;

char szOrder[32];
int iProcessed = TRUE;
vuser_init()
{
    if (lr_load_dll("vtclient.dll") != LR_PASS)
    {
        lr_error_message("vtserver failed to load");
        return -1;
    }

    pVTS = vtc_connect("10.192.89.7", 8888, VTOPT_KEEP_ALIVE);
     .
     .
//You can place your business process initialization code here
    return 0;
}
```

VTS Technique

```
Actions()
{//catch failed unprocessed data that does not terminate the script from previous iteration
      if (iProcessed == FALSE && InsertDataItem("vUnprocessed", szOrder) != LR_PASS){
            iProcessed = TRUE;
            return -1;
      }
      //Get order number from the VTS
      if (GetItem("vOrder", szOrder) != LR_PASS)
            return -1;
      iProcessed = FALSE;
      lr_save_string(szOrder, "pOrderNumnber");
            :
// You can replace this update section with code for your business process
      lrd_stmt(Csr5, "UPDATE Orders … WHERE order_number=<pOrderNumber>", -1, 1, 0, 0);
            :
      //Reinsert order number to end of  the list
      if (InsertDataItem("vOrder", szOrder) != LR_PASS)
            return -1;
      iProcessed = TRUE;
      return 0;
}
```

Note: data count greater than or equal to vuser count (reusable data only)

VTS Technique

```
vuser_end()
{
    if (iProcessed == FALSE)
    {
          InsertDataItem("vUnprocessed", szOrder);
    }

    vtc_disconnect(pVTS);
      .
      .
// You can place your business process cleanup code here
    return 0;
}
```

VTS Technique

- Advantages
  - Maintain one list of unique data items for multiple vuser scripts across scenario groups
  - No problem restarting failed vusers
  - Each user can potentially process any piece of data (data not contained within vuser blocks)
  - List of unprocessed data items at end of run
  - Run unlimited number of iterations (assuming no failures)
- Disadvantages
  - Extra level of coding complexity added to script
  - Script or program needed to load initial data into VTS

Implementing the Data Load with LoadRunner

- Good choice for loading large volumes of data
- Most LR protocols provide automatic data validation by forcing the data through the normal processing before insertion into data base
- Note:
  - Improper correlation with protocols that access the database directly can cause database corruption

Normal Parameterization

- Advantages
  - Easy (built in functionality)
- Disadvantages
  - Hard to track unprocessed data when executing multiple vusers
  - If vusers fail, cannot restart immediately without reprocessing same data
  - If vusers are stopped prematurely, hard to determine the remaining data to be processed

VTS Implementation

```
Actions()
{//catch failed unprocessed data that does not terminate the script from previous iteration
    if (iProcessed == FALSE && InsertDataItem("vUnprocessed", szOrder) != LR_PASS){
            iProcessed = TRUE;
            return -1;
    }
    if (GetItem("vData", szData) != LR_PASS)
            return -1;
    iProcessed = FALSE;
    lr_save_string(szData, "pOrderNumnber");
            :
// Place your data loading business process code here
            :
    iProcessed = TRUE;
    return 0;
}
```

Note: Init and End Sections remain the same as VTS unique technique; just change the variable names

VTS Implementation

- Advantages
  - Easy to track processed/unprocessed data
  - Easy to restart failed or stopped vusers
  - Easy to execute multiple vusers
- Disadvantages
  - Extra level of coding complexity added to script
  - Script or program needed to load initial data into VTS

Simulating User Environments With Real Time Data Sharing Between Vusers

Technique 1

- Process multiple business processes in a single action file
- Action1()
  - Create order
  - Delete order

- Process multiple business processes in multiple action files (if supported by protocol)
- Action1()
  - Create order
- Action2()
  - Delete order

Technique 1

- Advantages
  - Easy to do with normal correlation
- Disadvantages
  - Difficult to correlate when Action files do not have a 1 to 1 relationship

Technique 2

- Pre-create data sets before scenario execution

Note: Pre-create a data set for each planned scenario run between database restores

- Create Order script
  - Generate N data sets
  - Back up database
- Delete Order script
  - Use generated data sets
- Execute Planned Scenarios
- Restore Database

Technique 2

- Advantages
  - Easy to implement with normal parameterization
  - Related scripts can be executed at different ratios
- Disadvantages
  - Data has to be pre-created for each planned scenario execution between database restores
  - If a scenario run fails or is stopped prematurely, a database restore may be required to use the data again
  - Newly created data is not processed during scenario run
  - Could possibly cause initial database seeding to be larger than planned

VTS Technique

- Create two scripts
  - Create Order
  - Delete Order
- Execute Scenarios

VTS Technique

```
// Add order action file
Actions()
{
// Place Add order code here and capture created data
                .
                .
    if (InsertDataItem("vOrder", szOrder) != LR_PASS)
          return -1;
    return 0;
}
```

VTS Technique

```
//Delete order action file
Actions()
{//catch failed unprocessed data that does not terminate the script from previous iteration
      if (iProcessed == FALSE && InsertDataItem("vUnprocessed", szOrder) != LR_PASS){
            iProcessed = TRUE;
            return -1;
      }
      // checks empty column every 5 seconds and times out if column remains empty 120 seconds
      if (GetItemWait("vOrder", szOrder,5, 120) != LR_PASS)
            return -1;
      iProcessed = FALSE;
      lr_save_string(szOrder, "pOrderNumnber");
            :
// Add delete order code here
            :
      // uncomment below if  you want to add processed data back to the column
      // data must be non-exhaustable ( reusable)
      //if (InsertDataItem("vOrder", szOrder) != LR_PASS)   return -1;
      iProcessed = TRUE;
      return 0;
}
```

VTS Technique

int GetItemWait(char *pszColumn, char *pszItem, int iInterval, int iTimeout)

pszColumn – column (queue) name

pszItem – data item to be inserted into queue

iInterval – time interval in seconds for reading VTS

iTimeout – maximum time length a vuser is allowed to reread a previously empty column

This function will allow a vuser to request the next value from the VTS; however, if the column is empty, it will keep trying to request the next value at regular intervals (iInterval) until the timeout (iTimeout) value is exceeded. All times are in seconds.

VTS Technique

```
int GetItemWait(char *pszColumn, char *pszItem, int iInterval, int iTimeout)
{
	int iTime = time(0); //assign unix time in seconds
	int iRC;

	while ((time(0) - iTime) < iTimeout)
	{
		iRC = GetItem(pszColumn, pszItem);
		switch(iRC)
		{
			case VTCERR_OK:
				return LR_PASS;
				break;

			case VTCERR_RESPONSE_ARGS_UNMATCH:
				sleep(iInterval * 1000);
				continue;
				break;

			default:
				return LR_FAIL;
				break;
		}

	}
	lr_error_message("forced VTS timeout; waited %d sec.", iTimeout);
	return LR_FAIL;
}
```

VTS Technique

- Advantages
  - Related scripts can be executed at different ratios
  - Data pre-creation not needed
  - More accurately simulates a user environment
- Disadvantages
  - Extra level of coding complexity added to script
  - Data must be created at a greater rate than consumed if created data is not reusable

Creating Critical Section

What is a critical section?

- A section of code that does an atomic task in multiple steps (ie. Updating a row-id table)
- A section of code that will cause a conflict or error if two users execute the section of code simultaneously
  - Usually involves correlating values that should be unique when used later in the script

Database Example

Sequence table that generates key for other tables

- Non-atomic code example
  - Fetch (correlate value)
  - Increment (correlated value = correlated value + N)
  - Update (correlated value + N)
  - Insert (Original correlated value into another record as key value)

Note:

- It is possible for two or more concurrent vusers to fetch the same value before it is updated

DB Code Segment

```
lrd_stmt(Csr3, "SELECT MAX(order_number)+1 FROM Orders", -1, 1, 0, 0);

lrd_bind_cols(Csr3, BCInfo_D17, 0);
lrd_save_col(Csr3,1,1,0,"dOrderCounter");
lrd_fetch(Csr3, 1, 1, 0, PrintRow3, 0);
 lrd_close_cursor(&Csr3, 0);

lrd_open_cursor(&Csr4, Con1, 0);
lrd_stmt(Csr4,
    "UPDATE Counters SET counter_value=<dOrderCounter> WHERE
    table_name='Orders'", -1,  1, 0, 0);
              :
 lrd_stmt(Csr10, "INSERT INTO Orders (order_number,agent_no,customer_no,"
      "flight_number,departure_date,tickets_ordered,class,"
      "send_signature_with_order) VALUES (<dOrderCounter>, 107,
    <dCustomerCounter>, 1490, {d "
      "'2002-09-01'}, 1, '1', 'N')", -1, 1, 0, 0);
```

Traditional Solution 1

- Modify Code – make update an atomic operation

Update(value = value + 1)…correlate value

Fetch (New value)

Insert (New Value)

- Note:
  - changes actual SQL and ordering of steps
  - difficult for complex steps

Solution 1 Code Segment

```
lrd_stmt(Csr3, "UPDATE Counters SET counter_value=counter_value+1 WHERE "
        "table_name='Orders'", -1, 1, 0, 0);
l5d_close_cursor(&Csr3, 0);

lrd_open_cursor(&Csr4, Con1, 0);
lrd_stmt(Csr4,
   "SELECT counter_value FROM Counters WHERE table_name='Orders'", -1, 1,  0, 0);
lrd_bind_cols(Csr4, BCInfo_D17, 0);
lrd_save_col(Csr4,1,1,0,"dOrderCounter");
lrd_fetch(Csr4, 1, 1, 0, PrintRow3, 0);
                :
 lrd_stmt(Csr7, "INSERT INTO Orders (order_number,agent_no,customer_no,"
    "flight_number,departure_date,tickets_ordered,class,"
    "send_signature_with_order) VALUES (<dOrderCounter>, 107, 703, <dFlightNo>, {d "
    "'<pCurrentDate>'}, 1, '3', 'N')", -1, 1, 0, 0);
```

Traditional Solution 2

- Modify Code – with parameters

Fetch(value)
Update(Unique Parameter)
Insert (Unique Parameter)

- Note:
  - Can possibly corrupt database

Solution 2 Code Segment

```
lrd_stmt(Csr3, "SELECT MAX(order_number)+1 FROM Orders", -1, 1, 0, 0);

lrd_bind_cols(Csr3, BCInfo_D17, 0);
lrd_fetch(Csr3, 1, 1, 0, PrintRow3, 0);
 lrd_close_cursor(&Csr3, 0);

lrd_open_cursor(&Csr4, Con1, 0);
lrd_stmt(Csr4,
    "UPDATE Counters SET counter_value=<pOrderCounter> WHERE
     table_name='Orders'", -1,  1, 0, 0);
              :
  lrd_stmt(Csr10, "INSERT INTO Orders (order_number,agent_no,customer_no,"
       "flight_number,departure_date,tickets_ordered,class,"
       "send_signature_with_order) VALUES (<pOrderCounter>, 107,
     <dCustomerCounter>, 1490, {d "
       "'2002-09-01'}, 1, '1', 'N')", -1, 1, 0, 0);
```

VTS Solution

- Execute Blocking Setup Script
  - Run setup script before scenario execution to place VTS in proper state for creating a virtual blocking mechanism
- Modify Code – add VTS blocking code

  Allow one vuser inside critical section and block other vusers out

  Fetch (correlated value)

  Increment

  Update

  Insert (Correlated value into record as key value)

  Vuser unblocks when finished executing critical section

- Note:
  - Each critical section requires a separate column in VTS
  - No modification of recorded statements
  - Could mask concurrency runtime issues if no other application mechanisms guarantee serial execution of the critical section

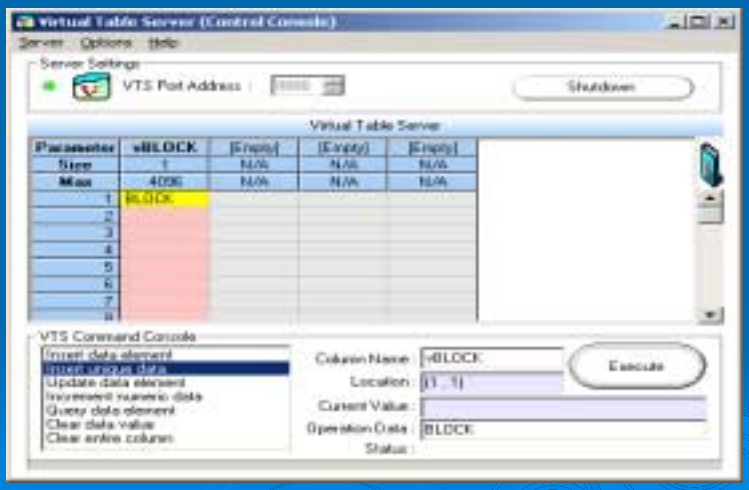
VTS Solution Code Segment

```
#include "vts2.h"

PVCI pVTS = 0;

char szBlock[32];
int iBlock = FALSE;
vuser_init()
{
    if (lr_load_dll("vtclient.dll") != LR_PASS)
    {
        lr_error_message("vtserver failed to load");
        return -1;
    }

    pVTS = vtc_connect("localhost", 8888, VTOPT_KEEP_ALIVE);

// Add initialization code here
```

VTS Solution Setup Script

```
Actions()
{
	int iRC;

	if (lr_load_dll("vtclient.dll") != LR_PASS)
	{
		lr_error_message("vtserver failed to load");
		return -1;
	}

	pVTS = vtc_connect("galaleo7", 8888, VTOPT_KEEP_ALIVE);
	if (EmptyColumn("vBLOCK") == LR_FAIL)
	{
		return -1;
	}

	iRC = InsertUniqueItem("vBLOCK", "BLOCK");
	if (iRC != LR_PASS)
	{
		return-1;
	}

	vtc_disconnect(pVTS);
	return 0;
}
```

VTS Solution Pre-scenario state



VTS Solution Code Segment

```
// Action section segment
// Set Block with GetItemWait
if (GetItemWait("vBLOCK", szBlock, 1, 60) != LR_PASS)   return -1;
iBlock = TRUE;
lrd_stmt(Csr3, "SELECT MAX(order_number)+1 FROM Orders", -1, 1, 0, 0);

lrd_bind_cols(Csr3, BCInfo_D17, 0);
lrd_save_col(Csr3,1,1,0,"dOrderCounter");
lrd_fetch(Csr3, 1, 1, 0, PrintRow3, 0);
 lrd_close_cursor(&Csr3, 0);

lrd_open_cursor(&Csr4, Con1, 0);
lrd_stmt(Csr4,
    "UPDATE Counters SET counter_value=<dOrderCounter> WHERE table_name='Orders'", -1,  1,
      0, 0);
// Release Block
if (InsertUniqueItem("vBLOCK", "BLOCK") == LR_FAIL)    return -1;
iBlock = FALSE;
                :
 lrd_stmt(Csr10, "INSERT INTO Orders (order_number,agent_no,customer_no,"
      "flight_number,departure_date,tickets_ordered,class,"
      "send_signature_with_order) VALUES (<dOrderCounter>, 107, <dCustomerCounter>, 1490, {d "
      "'2002-09-01'}, 1, '1', 'N')", -1, 1, 0, 0);
```

VTS Solution Code Segment

```
vuser_end()
{ // Releases block when vusers fails within critical
   section
   if (iBlock == TRUE)
   {
        InsertUniqueItem("vBLOCK", "BLOCK");
   }
   vtc_disconnect(pVTS);

// Add cleanup code here
```

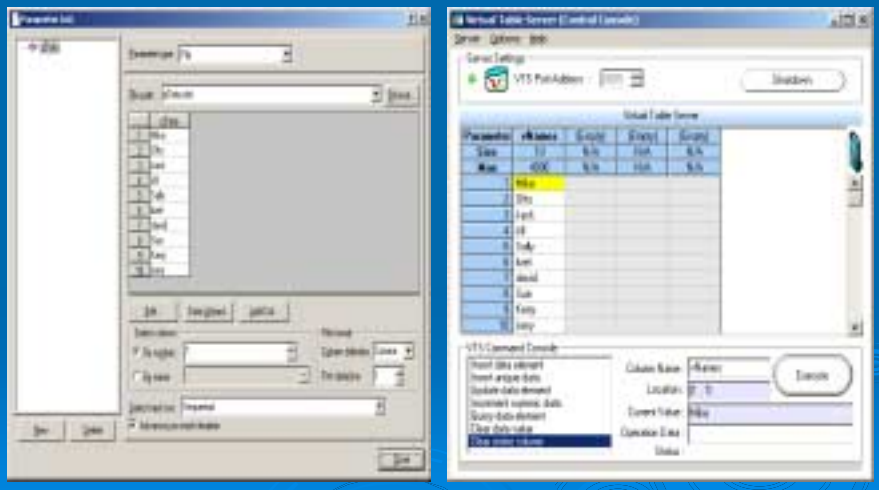
Conclusion

- With minor script enhancements the Virtual Table Server can provide solutions for many load testing data issues
  - Real time data sharing between vusers during scenario execution
  - More thorough data tracking during and after scenario execution
  - Increased flexibility when data issues are outside the scope of the LoadRunner built-in parameterization constraints

Sample VTS Loading Script

```
Actions()
{
	int iRC, iCount=0;

	if (lr_load_dll("vtclient.dll") != LR_PASS)
	{
		lr_error_message("vtserver failed to load");
		return -1;
	}

	pVTS = vtc_connect("galaleo7", 8888, VTOPT_KEEP_ALIVE);
	if (EmptyColumn("vNames") == LR_FAIL)
	{
		return -1;
	}

	for(iCount = 0; iCount < 10; iCount++, lr_advance_param("pData"))
	{
		iRC = InsertUniqueItem("vNames", lr_eval_string("<pData>"));
		if (iRC != LR_PASS)
			return -1;
	}
	vtc_disconnect(pVTS);
	return 0;
}
```

Sample VTS Loading Script Results



Questions?