Scripting With ODBC Protocol

LoadRunner

A white paper from

Shainesh Baheti

Table of Contents

Scripting With ODBC (Open Database Connectivity) Protocol

1 Introduction

A lot of applications use ODBC as an interface to database server. A Load Tester is required to test the ODBC compliant database server. This is essential to develop the LoadRunner script that can connect to the server, submit an SQL query, retrieve and process the information and disconnect from the server. This document helps the beginner to create ODBC scripts to test the server and query response time. The research document provides details on ODBC, Grids and Queries.

It is critical to develop effective methodologies for scripting using ODBC Protocol to measure exact query response time and database server status. In this paper, we propose an approach for scripting with ODBC protocol. It explains the need of ODBC protocol and application that supports it. It includes information how ODBC works. Misys EMR application is used to explain the practical significance of LoadRunner ODBC script.

2 What is ODBC?

ODBC (Open Database Connectivity) is a function library, which provides an open standard API (Application Programming Interface) for ODBC compliant database management systems (DBMS). Using ODBC statements in a program, you can access data from any application like Access, dBase, DB2, Excel, and Text. ODBC is a set of standard function calls based on the SQL Access Group (SAG) for managing a SQL database system (back-end system). ODBC operates as an industry-standard "shim" between applications that utilize databases and the databases itself. ODBC is based on and closely aligned with the Open Group standard Structured Query Language (SQL) Call-Level Interface. ODBC allows programs to use SQL requests that will access databases without having to know the proprietary interfaces to the databases.

3 ODBC Architecture

ODBC provides a robust set of functions to access a database. It is the most widely supported portable database interface available. Though, often accessed through higher-level objects, ODBC provides the appropriate power to utilize databases. The ODBC drivers are developed in sub-components:

- ODBC Driver Manager
- ODBC Drivers

Applications access (submits ODBC calls) the ODBC functions through the ODBC Driver Manager that are dynamically linked to the appropriate ODBC driver. The ODBC driver manager selects the appropriate ODBC driver, loads that driver and sends read/write request using the same driver. ODBC drivers translate ODBC requests to native format for a specific data source. The data source can be a complete RDBMS like FirstSQL or it may be a simple file like Xbase. The ODBC driver processes the ODBC function calls, submits the SQL requests to the database, and returns the results.

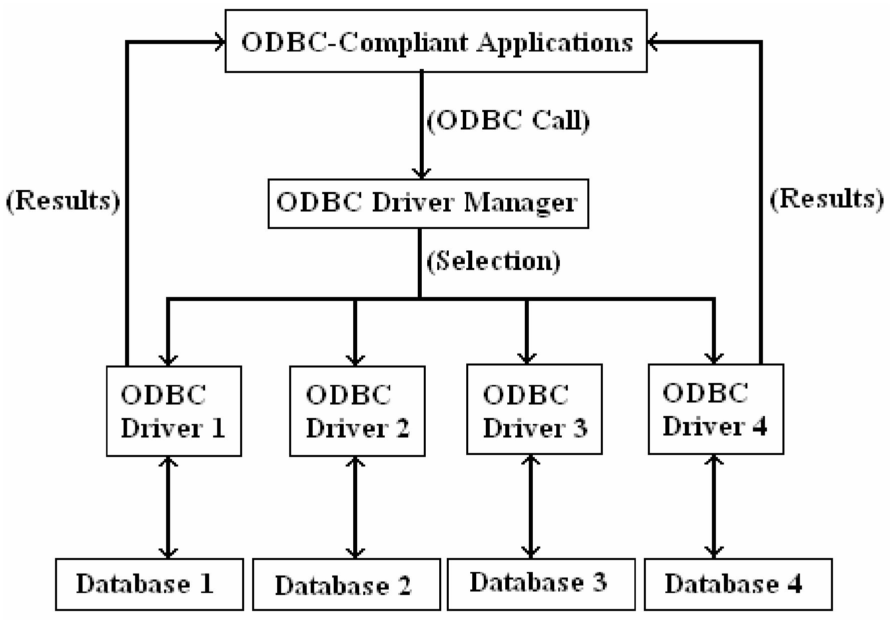


Figure: 1 Hierarchy of ODBC components

When writing programs in the Java language and using the Java Database Connectivity (JDBC) application program interface, a product that includes a JDBC-ODBC "bridge" program to reach ODBC-accessible databases can be used.

4 LoadRunner and ODBC

It is important to know the communication method/protocol type used by the application to communicate to the server. Example, using a Java application, communication can be either developed in Java or may be by using RMI-Java, RMI-CORBA or HTTP. Likewise, for Win32, it could be developed in VB using DCOM, ADO or HTTP and so on. Select an appropriate protocol from the list created in the LoadRunner. RMI-Java use RMI-Java, HTTP use Web (http/html), DCOM uses COM/DCOM, ADO use ODBC (if ODBC connectivity is used). Consult the application development team and understand the communication method used to determine the protocol (usually the Load Tester does not have such knowledge).

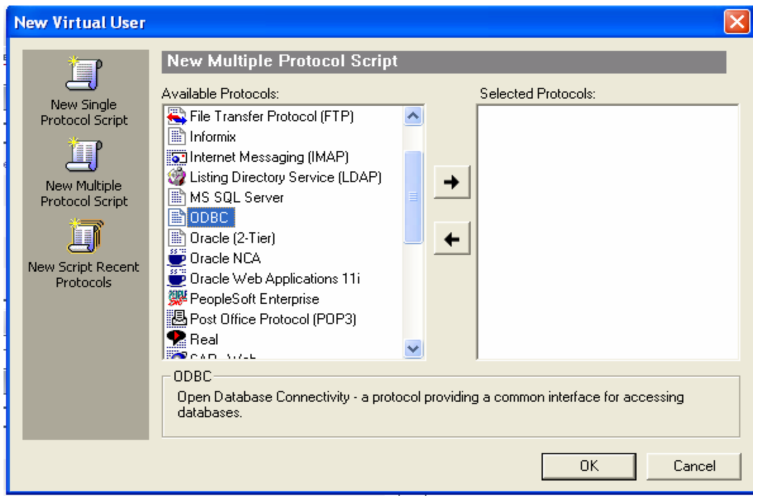
All the applications to be connected to database through ODBC communication have to use ODBC protocol. LoadRunner supports the ODBC protocol. The LoadRunner captures API calls while recording and plays them back. Hence, while creating a LoadRunner ODBC script, Vugen records all the ODBC calls made from the application. The LoadRunner hooks into the API to capture the corresponding API calls.

5 Recording with ODBC

The Vuser script holds the function that measure and record the performance of the server. Create a new Vuser (virtual user) script in LoadRunner every time you record. You are not allowed to record into an existing script.

Recording steps are as follows:

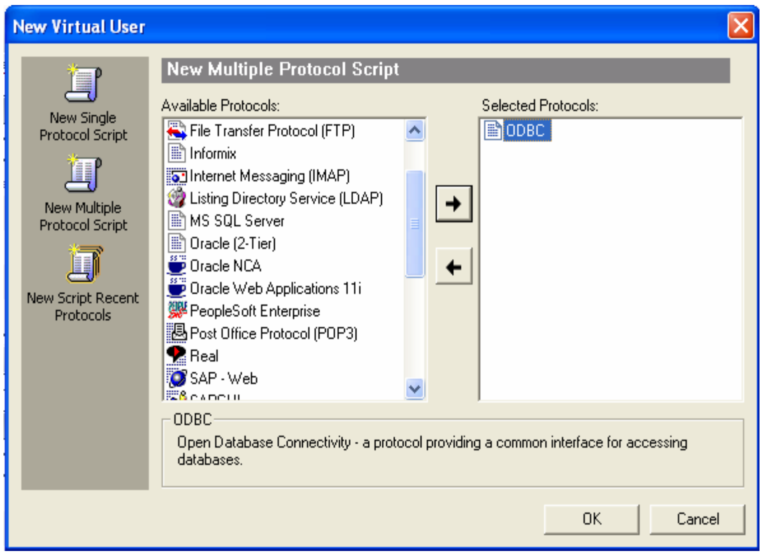
1. Select **ODBC** as the Protocol from the **Available Protocols** list.


New Virtual User
New Single Protocol Script
New Multiple Protocol Script
New Script Recent Protocols
New Multiple Protocol Script
Available Protocols:
File Transfer Protocol (FTP)
Informix
Internet Messaging (IMAP)
Listing Directory Service (LDAP)
MS SQL Server
ODBC
Oracle (2-Tier)
Oracle NCA
Oracle Web Applications 11i
PeopleSoft Enterprise
Post Office Protocol (POP3)
Real
Selected Protocols:
OK
Cancel

ODBC

Open Database Connectivity - a protocol providing a common interface for accessing databases.

2. Click **Ok.**

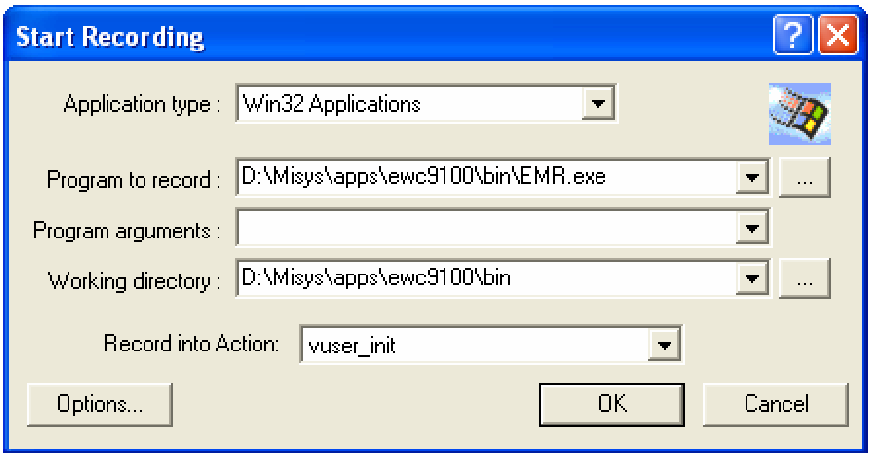

New Virtual User
New Single Protocol Script
New Multiple Protocol Script
New Script Recent Protocols
New Multiple Protocol Script
Available Protocols:
File Transfer Protocol (FTP)
Informix
Internet Messaging (IMAP)
Listing Directory Service (LDAP)
MS SQL Server
Oracle (2-Tier)
Oracle NCA
Oracle Web Applications 11i
PeopleSoft Enterprise
Post Office Protocol (POP3)
Real
SAP - Web
Selected Protocols:
ODBC
OK
Cancel

ODBC

Open Database Connectivity - a protocol providing a common interface for accessing databases.

The **Start Recording** dialog box opens.

3. To Generate the Vuser script, choose the relevant **Application type**. Example, Win32Applications



Start Recording

Application type : Win32 Applications

Program to record : D:\Misys\apps\ewc9100\bin\EMR.exe

Program arguments :

Working directory : D:\Misys\apps\ewc9100\bin

Record into Action: vuser_init

The VuGen launches the application from working directory and begins to record.

In the above scenario **Program to record** path is specified as application (.exe) file and hence .exe file is invoked and records the API calls.

Specify a **Working directory** and in the **Record into Action** list, select the section where you want to record.

4. Click **OK.**

VuGen generates the ODBC Vuser script. The generated Vuser script contains LRD functions that describe the database activity. Each LRD function has lrd prefix with one or more database functions. For example, the **lrd_fetch** function represents a fetch operation. While running a recorded session, the Vuser script communicates directly with the database server, performing the same operations as the original user.

6 ODBC Scripts

In Misys EMR application, suppose there is a database of Patient accessed by Doctors, Nurse, Lab Clerk and Transcription Assistants. The database Vusers emulates this scenario in which the database server services multiple requests for information. A Database Vuser can:

⇨ connect to the server

⇨ submit an SQL query

⇨ retrieve and process the information

⇨ disconnect from the server

You can distribute several hundred Database Vusers among the available hosts, each Vuser accessing the database by using the server API. It measures the performance of the server under the load of multiple users. The program that holds the calls to the server API is called a **Database Vuser script** or ODBC Vuser script in this case. It emulates the client application and all of the actions performed by it. Using the LoadRunner Controller/HP Performance Center, assign the script to multiple Vusers. The Vusers execute the script and emulate user load on the client/server system. LoadRunner generates performance data, which can be analyzed in report and graph format.

VuGen is used to perform the following:

⇨ Create ODBC Vuser scripts by recording all the activity between a database client and the server.

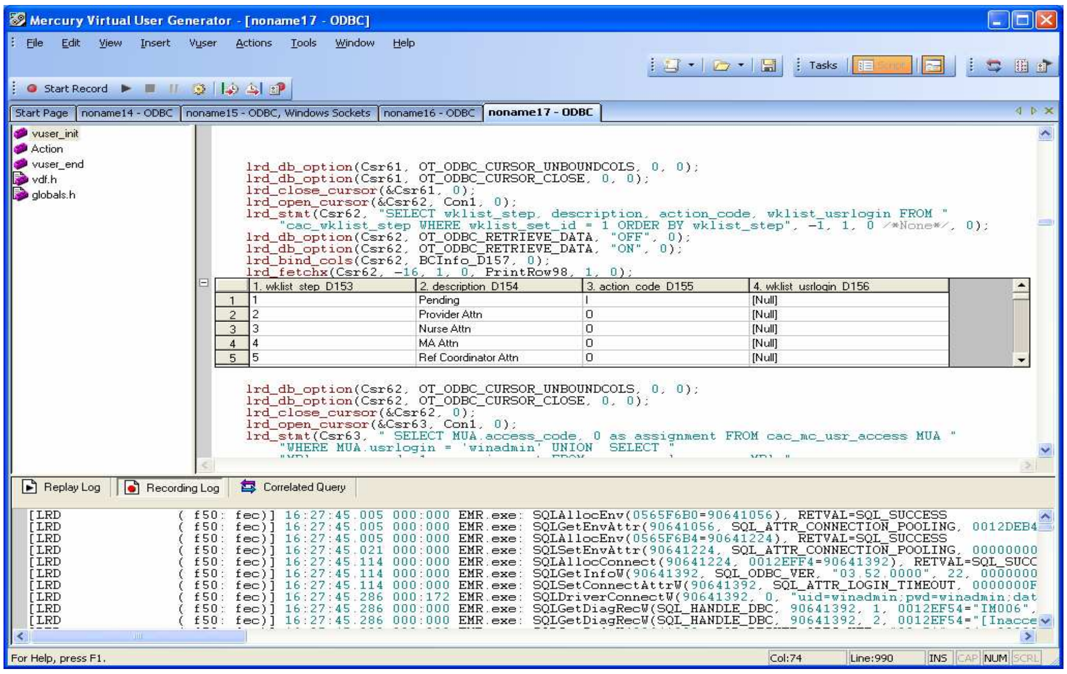
⇨ Monitor the communication between the client end of the database.

⇨ Trace all the requests sent to and received from the database and server.

Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.

After creating Database Vuser scripts in a Windows environment using VuGen, It can be assigned to Vusers in both Windows and UNIX environments.

A typical ODBC script looks like this:


Mercury Virtual User Generator - [noname17 - ODBC]
File Edit View Insert Vuser Actions Tools Window Help
Tasks
Start Record
Start Page | noname14 - ODBC | noname15 - ODBC, Windows Sockets | noname16 - ODBC | noname17 - ODBC
vuser_init
Action
vuser_end
vdf.h
globals.h
Replay Log | Recording Log | Correlated Query
For Help, press F1.
Col:74 Line:990 INS CAP NUM SCRL

```
lrd_db_option(Csr61, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr61, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr61, 0);
lrd_open_cursor(&Csr62, Con1, 0);
lrd_stmt(Csr62, "SELECT wklist_step, description, action_code, wklist_usrlogin FROM "
    "cac_wklist_step WHERE wklist_set_id = 1 ORDER BY wklist_step", -1, 1, 0 /*None*/, 0);
lrd_db_option(Csr62, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
lrd_db_option(Csr62, OT_ODBC_RETRIEVE_DATA, "ON", 0);
lrd_bind_cols(Csr62, BCInfo_D157, 0);
lrd_fetchx(Csr62, -16, 1, 0, PrintRow98, 1, 0);
```

	1. wklist_step_D153	2. description_D154	3. action_code_D155	4. wklist_usrlogin_D156
1	1	Pending	I	[Null]
2	2	Provider Attn	0	[Null]
3	3	Nurse Attn	0	[Null]
4	4	MA Attn	0	[Null]
5	5	Ref Coordinator Attn	0	[Null]

```
lrd_db_option(Csr62, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr62, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr62, 0);
lrd_open_cursor(&Csr63, Con1, 0);
lrd_stmt(Csr63, " SELECT MUA.access_code, 0 as assignment FROM cac_mc_usr_access MUA "
    "WHERE MUA.usrlogin = 'winadmin' UNION  SELECT "
```

```
[LRD          ( f50: fec)] 16:27:45.005 000:000 EMR.exe: SQLAllocEnv(0565F6B0=90641056), RETVAL=SQL_SUCCESS
[LRD          ( f50: fec)] 16:27:45.005 000:000 EMR.exe: SQLGetEnvAttr(90641056, SQL_ATTR_CONNECTION_POOLING, 0012DEB4
[LRD          ( f50: fec)] 16:27:45.005 000:000 EMR.exe: SQLAllocEnv(0565F6B4=90641224), RETVAL=SQL_SUCCESS
[LRD          ( f50: fec)] 16:27:45.021 000:000 EMR.exe: SQLSetEnvAttr(90641224, SQL_ATTR_CONNECTION_POOLING, 00000000
[LRD          ( f50: fec)] 16:27:45.114 000:000 EMR.exe: SQLAllocConnect(90641224, 0012EFF4=90641392), RETVAL=SQL_SUCC
[LRD          ( f50: fec)] 16:27:45.114 000:000 EMR.exe: SQLGetInfoW(90641392, SQL_ODBC_VER, "03.52.0000", 22, 0000000
[LRD          ( f50: fec)] 16:27:45.114 000:000 EMR.exe: SQLSetConnectAttrW(90641392, SQL_ATTR_LOGIN_TIMEOUT, 0000000F
[LRD          ( f50: fec)] 16:27:45.286 000:172 EMR.exe: SQLDriverConnectW(90641392, 0, "uid=winadmin;pwd=winadmin;dat
[LRD          ( f50: fec)] 16:27:45.286 000:000 EMR.exe: SQLGetDiagRecW(SQL_HANDLE_DBC, 90641392, 1, 0012EF54="IM006",
[LRD          ( f50: fec)] 16:27:45.286 000:000 EMR.exe: SQLGetDiagRecW(SQL_HANDLE_DBC, 90641392, 2, 0012EF54="[Inacce
```

After recording an ODBC session, you can view the recorded code in VuGen's built-in editor. Use the scroll bar to view the SQL statements that were generated by the application and examine the data returned by the server. The

VuGen window displays the following information about the recorded database session:

- the sequence of functions recorded
- grids displaying the data returned by database queries
- the number of rows fetched during a query

6.1 Function Sequence

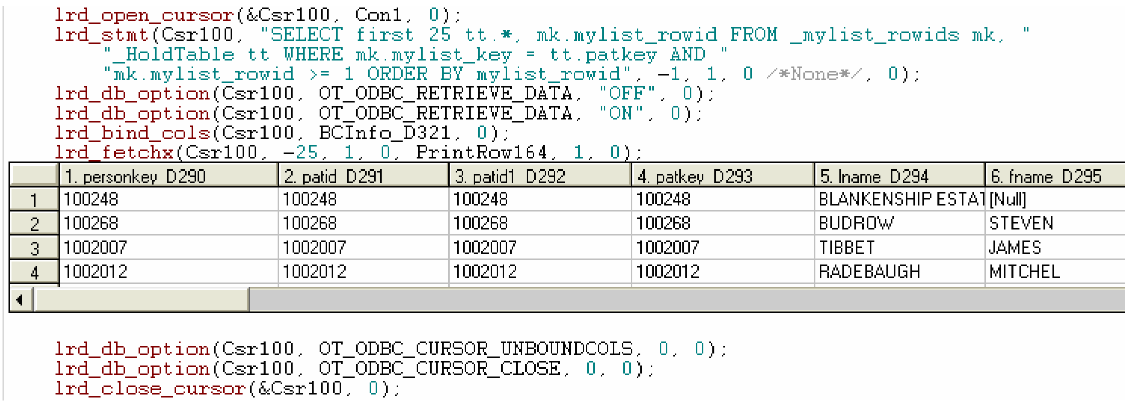
Vuser script in the VuGen window has particular sequence in which VuGen records the application-database activities. Following sequence of functions are recorded during a typical Misys EMR database session:

Function Name	Description
lrd_init	Initializes the LRD environment.
lrd_open_context	Opens a context.
lrd_alloc_connection	Allocates a connection structure.
lrd_open_connection	Connects (logs in) to the database.
lrd_open_cursor	Opens a database cursor.
lrd_stmt	Specifies an SQL statement to be processed
lrd_bind_col	Binds a host variable to an output column.
lrd_exec	Executes the previously specified SQL statement.
lrd_fetchx	Fetches the next row in the result set using an extended fetch.
lrd_close_cursor	Closes a database cursor.
lrd_free_connection	Frees a connection structure.

6.2 Grids

The data returned by a database query during a recording session is displayed in a grid. By viewing the grid, you can determine how the application generates SQL statements and the efficiency of the client/server system.

For a query executed on a Patient database, VuGen displays the following grid. The query retrieves Person key, Pat id, last name, first name, ID number etc. of all the patients in the records.



```
lrd_open_cursor(&Csr100, Con1, 0);
lrd_stmt(Csr100, "SELECT first 25 tt.*, mk.mylist_rowid FROM _mylist_rowids mk, "
    "_HoldTable tt WHERE mk.mylist_key = tt.patkey AND "
    "mk.mylist_rowid >= 1 ORDER BY mylist_rowid", -1, 1, 0 /*None*/, 0);
lrd_db_option(Csr100, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
lrd_db_option(Csr100, OT_ODBC_RETRIEVE_DATA, "ON", 0);
lrd_bind_cols(Csr100, BCInfo_D321, 0);
lrd_fetchx(Csr100, -25, 1, 0, PrintRow164, 1, 0);
```

	1. personkey D290	2. patid D291	3. patid1 D292	4. patkey D293	5. lname D294	6. fname D295
1	100248	100248	100248	100248	BLANKENSHIP ESTAT	[Null]
2	100268	100268	100268	100268	BUDROW	STEVEN
3	1002007	1002007	1002007	1002007	TIBBET	JAMES
4	1002012	1002012	1002012	1002012	RADEBAUGH	MITCHEL

```
lrd_db_option(Csr100, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr100, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr100, 0);
```

To show or hide the grid select **View** > **Data Grids**. You can adjust the width of the grid columns. Up to 100 rows can be scrolled using the scroll bar.

6.3 Row Information

VuGen generates an **lrd_fetch or lrd_fetchx** function for each SQL query:

lrd_fetch (Csr1, -14, 1, 0, PrintRow24, 0); or

lrd_fetchx (Csr85, -4, 1, 0, PrintRow144, 1, 0) ; (using an extended fetch, SQLExtendedFtech).

The second parameter of the function indicates the number of rows fetched. This can be a positive/negative number.

Positive Row Values

A positive value shows the number of rows fetched during recording, and indicates that not all rows were fetched. (For example, if the operator cancelled the query before it was completed.).

Example,

lrd_fetch (Csr10, 42, 1, 0, PrintRow10, 0); or

lrd_fetchx (Csr85, 42, 1, 0, PrintRow144, 1, 0)

Forty-two rows are retrieved during the database query, but not all of the data is fetched. During execution, the script always retrieves the number of rows indicated by the positive value (provided the rows exist).

Negative Row Values

A negative row value indicates that all available rows were fetched during recording. The absolute value of the negative number is the number of rows fetched.

Example,

lrd_fetch (Csr1, -4, 1, 0, PrintRow7, 0); or

lrd_fetchx (Csr85, - 42, 1, 0, PrintRow144, 1, 0)

All four rows of the result set are retrieved. When you execute an **lrd_fetch** statement containing a negative row value, it retrieves all of the available rows in the table at the time of the run—not necessarily the number at the time of recording. In the above example, all four rows of the table were retrieved during the recording session. However, if more rows are available during script execution, they are all retrieved.

6.4 Error Codes

A return code is generated when LoadRunner executes the LRD function. A return code of 0 indicates successful function. For example, a return code of 0 indicates that another row is available from the result set. If an error occurs, the return code indicates the type of error. For example, a return code of 2014 indicates that an error occurred in the initialization. The error codes are also described in the lrd.h file supplied with VuGen.

Return codes are of four types each represented by a range of numbers.

Type of Return Code	Range
Informational	0 to 999
Warning	1000 to 1999
Error	2000 to 2999
Internal Error	5000 to 5999

7 Creating ODBC Scripts

This section provides an overview of the process of developing ODBC Vuser scripts using VuGen.

Follow the procedure to develop the ODBC Vuser script:

7.1 Record the basic script using VuGen

Invoke VuGen and create a new Vuser script as explained earlier. Specify the type of Vuser as ODBC. Choose an application to record and mention application type (Windows/Internet). Record typical operations on the application.

7.2 Enhancement & Modification of script

Enhance the Vuser script by inserting transactions, rendezvous points, and control-flow structures into the script. Unlike web scripts, which are readable, the steps in ODBC scripts are not too obvious as it records Back-end operations. So it is very important to insert comments, transactions and rendezvous points (if required) into the script while recording. If there is any logic required in the script, insert the logic. For example in Misys EMR application we use some user-defined functions to control number of iteration and delay timings.

7.3 Parameterisation

Define parameters for the fixed-values recorded into Vugen script. By substituting fixed-values with parameters, the same query can be repeated multiple times using different values. In our application we parameterize Patient Id as we have to check for various patients.

7.4 Correlation

As we all know most important part of LoadRunner script is correlation. Here dynamic values are not present in URL, like session ids, in Http/Web Scripts. Dynamic value or value which needs to be correlated is present in grids. Correlating database statements allows using the result of a query in a subsequent one. This feature is useful when working on a database with user constraints. ODBC Vuser scripts can be optimize and simplify by correlating database queries. LoadRunner's Correlated Query feature allows to link database statements by using the results of one statement as input to another.

The two main reasons for correlating database queries are to:

- **simplify or optimize the code**

While performing a series of dependent queries one after another, code may become very long. In order to reduce the size of the code, nest the queries, but then it will lose clarity and the code becomes complex and difficult to understand. Correlating the queries allows to link queries without nesting.

- **Successfully run the script**

For example, record the process of opening a new bank account. Each new account is assigned a unique number that is unknown to the user and this account number is inserted into a table with a unique key constraint during recording. While executing the script as recorded, it will try to create an account with the recorded number, rather than a new unique number and thus will get a database error because the account number already exists. Examine the script at the point where the error occurred. In many cases, a correlated query will solve the problem, as it allows you to use the results of one statement as input to another.

Working procedure for correlating queries:

1. Ascertain the value to correlate

VuGen helps to decide what to correlate. Double-click an error message in the execution log to jump to the problematic statement in the Vugen script and search for possible values to correlate. Specific value can also be selected to correlate directly from the script if it is known where it can be optimize the script or handle a unique constraint error.

2. Save the results

When VuGen scans the script for values to correlate, it provides a list of all matching values from the results of previous queries. Save the value of a query to a variable using the lrd_save_col function.

3. Reference the saved values

VuGen replaces the constants in the query or database statement with the saved variables.

7.4.1 Correlated Query Functions

To correlate statements, modify the recorded script in the VuGen editor using one or more of the following LRD functions. VuGen inserts these functions while using the automatic correlation process:

lrd_save_col

This function saves the result value of a query that appears in a grid (the specified row and column) to a parameter. Use **lrd_save_col** when working with **lrd_fetch** or **lrd_fetchx**. Place the **lrd_save_col** function before fetching of the data. It assigns the value retrieved by the subsequent **lrd_fetch** to the specified parameter.

Note: *In general, checks for the problematic statement and then the correlated query tool searches previous statements to find all possible values to correlate or else go straight to the results value of a query (grid) and select the value to save, even if not intend to use it until much later on in the script. For example, suppose the server assigns a session-ID to the client, and will not need the session-ID until later. Then automatically insert an* ***lrd_save_col*** *statement to save it by highlighting the value in the grid and right click to create correlation.*

The correlation functions has the following syntax:

lrd_save_col (cursor, col_number, row_number, option, "parameter_name")**;**

If the row number specified does not exist, **lrd_save_col** retrieves the last row in the result set. To automatically retrieve the final row in the result set, specify 0 for the row_number parameter.

The option parameter for **lrd_save_col** specifies whether to set the specified parameter by the immediately following **lrd_fetch** only, or by all subsequent calls to **lrd_fetch**:

Option	Action	Remarks
0	Following	Set the parameter by the immediately following
1	Any	Set the parameter by all **lrd_fetch** calls for the current statement (advanced users).

Correlating a Query Automatically

When you use the automatic correlation process, VuGen:

- ⇨ Scans for potential correlations
- ⇨ Inserts the appropriate **lrd_save_col** statement to save the results to a parameter
- ⇨ Replaces the statement value with the parameter

Correlate a statement or specific Value:

This procedure correlates an entire statement or the entire script. If specific value to correlate is not known, follow the procedure:

1. View the execution output to check for errors.

Select **View** > **Output** to display the output folders at the bottom of the window. Check for errors in the Execution Log folder.

2. To scan a particular statement, place the cursor on the script statement for correlation.

Double-click an output / error message to jump to the corresponding statement in th script.

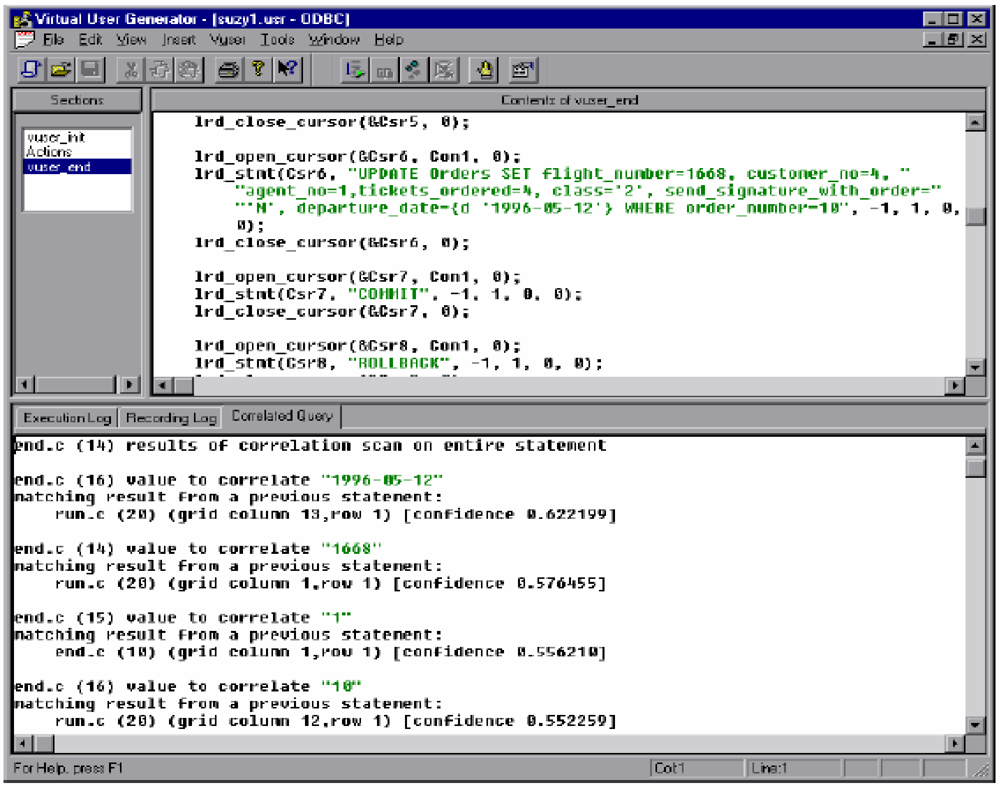
3. Select Vuser > Scan for Correlations (at cursor) (shortcut: Alt+F8), or

Vuser > Scan for Correlations (entire script) (shortcut: Ctrl+F8).

VuGen scans the script and lists all possible values to correlate in the selected statement, or the entire script, along with the matching results from previous statements.

The correlation values are listed in the Correlated Query folder. The possible values for correlation are graded; based on heuristics, select the best candidate for correlation.

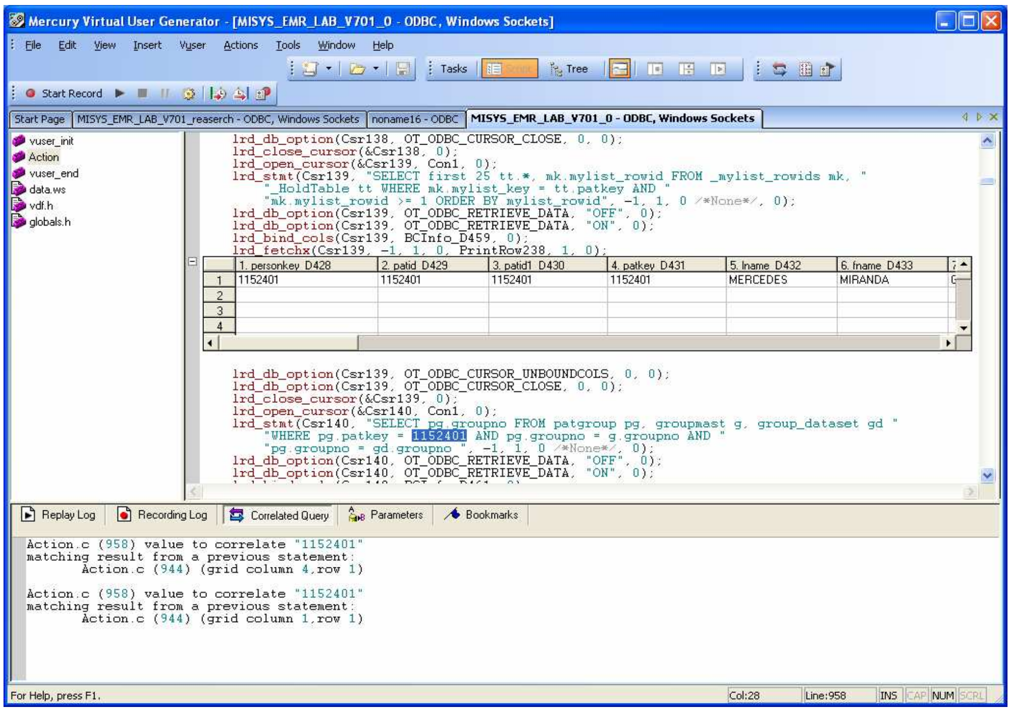
In the example below, VuGen found many possible values to correlate for the **lrd_stmt** (Csr6, 1 “UPDATE...) statement.



```
    lrd_close_cursor(&Csr5, 0);

    lrd_open_cursor(&Csr6, Con1, 0);
    lrd_stmt(Csr6, "UPDATE Orders SET flight_number=1668, customer_no=4, "
        "agent_no=1,tickets_ordered=4, class='2', send_signature_with_order="
        "'N', departure_date={d '1996-05-12'} WHERE order_number=10", -1, 1, 0,
        0);
    lrd_close_cursor(&Csr6, 0);

    lrd_open_cursor(&Csr7, Con1, 0);
    lrd_stmt(Csr7, "COMMIT", -1, 1, 0, 0);
    lrd_close_cursor(&Csr7, 0);

    lrd_open_cursor(&Csr8, Con1, 0);
    lrd_stmt(Csr8, "ROLLBACK", -1, 1, 0, 0);
```

```
end.c (14) results of correlation scan on entire statement

end.c (16) value to correlate "1996-05-12"
matching result from a previous statement:
    run.c (20) (grid column 13,row 1) [confidence 0.622199]

end.c (14) value to correlate "1668"
matching result from a previous statement:
    run.c (20) (grid column 1,row 1) [confidence 0.576455]

end.c (15) value to correlate "1"
matching result from a previous statement:
    end.c (10) (grid column 1,row 1) [confidence 0.556210]

end.c (16) value to correlate "10"
matching result from a previous statement:
    run.c (20) (grid column 12,row 1) [confidence 0.552259]
```

In the following example, VuGen found two possible matching result values to correlate to particular value of“1152401”.



```
    lrd_db_option(Csr138, OT_ODBC_CURSOR_CLOSE, 0, 0);
    lrd_close_cursor(&Csr138, 0);
    lrd_open_cursor(&Csr139, Con1, 0);
    lrd_stmt(Csr139, "SELECT first 25 tt.*, mk.mylist_rowid FROM _mylist_rowids mk, "
        "_HoldTable tt WHERE mk.mylist_key = tt.patkey AND "
        "mk.mylist_rowid >= 1 ORDER BY mylist_rowid", -1, 1, 0 /*None*/, 0);
    lrd_db_option(Csr139, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
    lrd_db_option(Csr139, OT_ODBC_RETRIEVE_DATA, "ON", 0);
    lrd_bind_cols(Csr139, BCInfo_D459, 0);
    lrd_fetchx(Csr139, -1, 1, 0, PrintRow238, 1, 0);
```

	1. personkey D428	2. patid D429	3. patid1 D430	4. patkey D431	5. lname D432	6. fname D433
1	1152401	1152401	1152401	1152401	MERCEDES	MIRANDA
2						
3						
4						

```
    lrd_db_option(Csr139, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
    lrd_db_option(Csr139, OT_ODBC_CURSOR_CLOSE, 0, 0);
    lrd_close_cursor(&Csr139, 0);
    lrd_open_cursor(&Csr140, Con1, 0);
    lrd_stmt(Csr140, "SELECT pg.groupno FROM patgroup pg, groupmast g, group_dataset gd "
        "WHERE pg.patkey = 1152401 AND pg.groupno = g.groupno AND "
        "pg.groupno = gd.groupno ", -1, 1, 0 /*None*/, 0);
    lrd_db_option(Csr140, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
    lrd_db_option(Csr140, OT_ODBC_RETRIEVE_DATA, "ON", 0);
```

```
Action.c (958) value to correlate "1152401"
matching result from a previous statement:
        Action.c (944) (grid column 4,row 1)

Action.c (958) value to correlate "1152401"
matching result from a previous statement:
        Action.c (944) (grid column 1,row 1)
```

4. Begin Automatic Correlation.

In the Correlated Query folder, double-click the result value to correlate to the statement value. The value is highlighted in the corresponding grid.

To create a correlation, select **Vuser** > **Create Correlation** or right-click. VuGen prompts to enter a name for the parameter to save the result value.

5. Enter a name for the parameter, or accept the default name and click **OK** to continue. VuGen inserts a statement **lrd_save_col** to save the result value to a parameter.

6. Click **Yes** to confirm the correlation. A message appears asking to replace all occurrences of the value in the script.

7. Click **No** to replace only the value in the selected statement.

8. To search for additional occurrences click **Yes**. A Search and Replace dialog box opens. Confirm any replacements, including the original statement. VuGen replaces the statement value with a reference to the parameter. Note that if cancel the correlation is chosen, VuGen erases the statement created in the previous step.

Correlate Queries to Optimize a Script

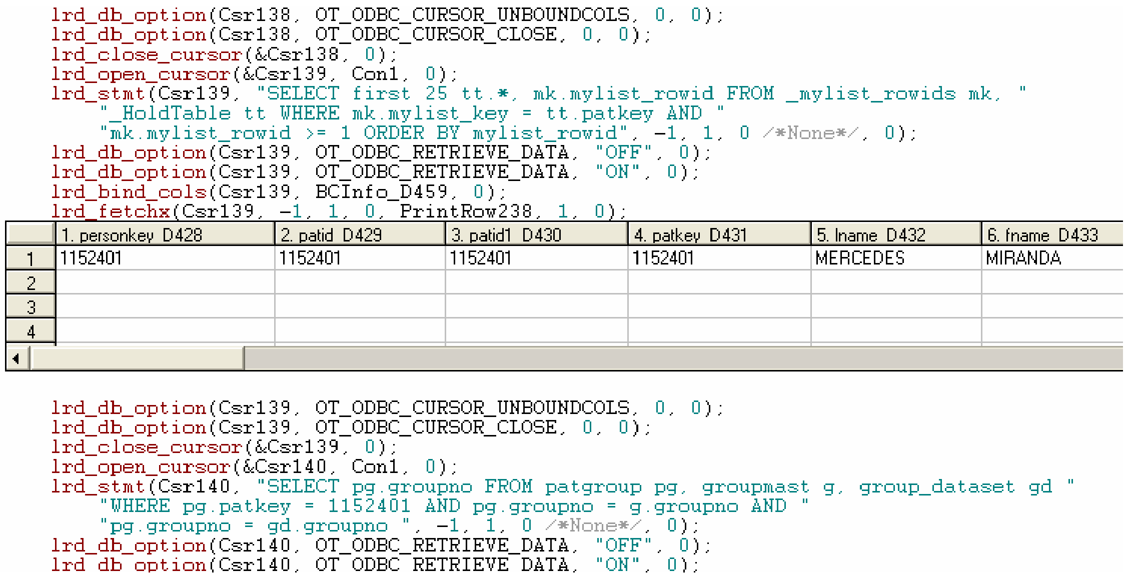
Correlated queries can be used to optimize script while performing multiple, dependent queries. Instead of waiting for the result of one query before performing the second, link between two queries. This feature is particularly useful in load testing, as it performs similar queries several times, using different parameters. The following sections illustrate linked queries for:

- simple queries (without binding)
- queries with binding

Simple Correlated Queries - Sample

A Doctor in a Clinic wants to execute a query to retrieve the Pat key for a patient. However, the table that contains the Pat key only lists the Patient ID. The Doctor does not know Patient’s ID; therefore, he will have to wait for the result of the first query before performing the second query.

The script below illustrates a simple correlated query. The user performed a query to retrieve Patient’s ID of 1152401. A second query on a different table, Pat Key with the above ID. The user waited for the results of the first query, before performing the second query.

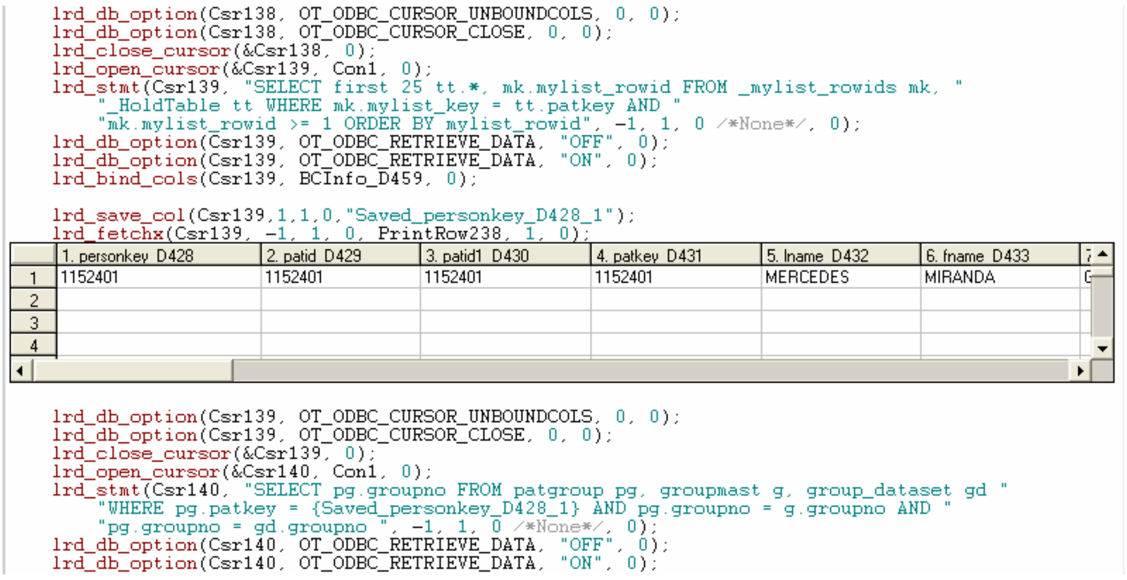


```
lrd_db_option(Csr138, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr138, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr138, 0);
lrd_open_cursor(&Csr139, Con1, 0);
lrd_stmt(Csr139, "SELECT first 25 tt.*, mk.mylist_rowid FROM _mylist_rowids mk, "
    "_HoldTable tt WHERE mk.mylist_key = tt.patkey AND "
    "mk.mylist_rowid >= 1 ORDER BY mylist_rowid", -1, 1, 0 /*None*/, 0);
lrd_db_option(Csr139, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
lrd_db_option(Csr139, OT_ODBC_RETRIEVE_DATA, "ON", 0);
lrd_bind_cols(Csr139, BCInfo_D459, 0);
lrd_fetchx(Csr139, -1, 1, 0, PrintRow238, 1, 0);
```

	1. personkey D428	2. patid D429	3. patid1 D430	4. patkey D431	5. lname D432	6. fname D433
1	1152401	1152401	1152401	1152401	MERCEDES	MIRANDA
2						
3						
4						

```
lrd_db_option(Csr139, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr139, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr139, 0);
lrd_open_cursor(&Csr140, Con1, 0);
lrd_stmt(Csr140, "SELECT pg.groupno FROM patgroup pg, groupmast g, group_dataset gd "
    "WHERE pg.patkey = 1152401 AND pg.groupno = g.groupno AND "
    "pg.groupno = gd.groupno ", -1, 1, 0 /*None*/, 0);
lrd_db_option(Csr140, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
lrd db option(Csr140, OT ODBC RETRIEVE DATA, "ON", 0);
```

To correlate the queries, save the value of a query with the **lrd_save_col** function. In this example, **lrd_save_col** saves the value of the first column and first row in the returned table (Patient ID of 1152401) and assigns it to a newly defined parameter, “**Saved_personkey_D428_1**”. This value is used in the second query.



```
lrd_db_option(Csr138, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr138, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr138, 0);
lrd_open_cursor(&Csr139, Con1, 0);
lrd_stmt(Csr139, "SELECT first 25 tt.*, mk.mylist_rowid FROM _mylist_rowids mk, "
    "_HoldTable tt WHERE mk.mylist_key = tt.patkey AND "
    "mk.mylist_rowid >= 1 ORDER BY mylist_rowid", -1, 1, 0 /*None*/, 0);
lrd_db_option(Csr139, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
lrd_db_option(Csr139, OT_ODBC_RETRIEVE_DATA, "ON", 0);
lrd_bind_cols(Csr139, BCInfo_D459, 0);

lrd_save_col(Csr139,1,1,0,"Saved_personkey_D428_1");
lrd_fetchx(Csr139, -1, 1, 0, PrintRow238, 1, 0);
```

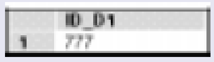
	1. personkey D428	2. patid D429	3. patid1 D430	4. patkey D431	5. lname D432	6. fname D433	7
1	1152401	1152401	1152401	1152401	MERCEDES	MIRANDA	G
2							
3							
4							

```
lrd_db_option(Csr139, OT_ODBC_CURSOR_UNBOUNDCOLS, 0, 0);
lrd_db_option(Csr139, OT_ODBC_CURSOR_CLOSE, 0, 0);
lrd_close_cursor(&Csr139, 0);
lrd_open_cursor(&Csr140, Con1, 0);
lrd_stmt(Csr140, "SELECT pg.groupno FROM patgroup pg, groupmast g, group_dataset gd "
    "WHERE pg.patkey = {Saved_personkey_D428_1} AND pg.groupno = g.groupno AND "
    "pg.groupno = gd.groupno ", -1, 1, 0 /*None*/, 0);
lrd_db_option(Csr140, OT_ODBC_RETRIEVE_DATA, "OFF", 0);
lrd_db_option(Csr140, OT_ODBC_RETRIEVE_DATA, "ON", 0);
```

Correlate Queries with Binding—Example

The following example illustrates a correlated query that uses binding. The user performs the same query as above, but instead of using a constant, he specifies a placeholder name, id. The **lrd_assign_bind** statement assigns the retrieved value to a host variable.

```
lrd_stmt(Csr1, "select id from employees where name='John'", ...);
lrd_bind_col(Csr1,...);
lrd_exec(Csr1, ...);
lrd_fetch(Csr1, 1,...);
```

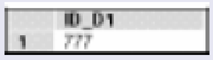


	ID_D1
1	777

```
lrd_stmt(Csr1, "select salary from payment where year='1996' and
id=:id",...);
lrd_assign_bind(Csr1, "id", "777", &id_D4, ...);
lrd_exec(Csr1, ...);
```

To correlate the queries, save the result of the first query to a new variable, emp_id using **lrd_save_col**. The constant 777 is replaced with the new variable name. The saved value is assigned to the placeholder by **lrd_assign_bind**.

```
lrd_stmt(Csr1, "select id from employees where name='John'", ...);
lrd_bind_col(Csr1,...);
lrd_exec(Csr1, ...);
lrd_save_col(Csr1, 1, 1, 0, "emp_id");
lrd_fetch(Csr1, 1, ...);
```



	ID_D1
1	777

```
lrd_stmt(Csr1, "select salary from payment where year='1996' and
id=:id",...);
lrd_assign_bind(Csr1, "id", "<emp_id>", &id_D4...);
lrd_exec(Csr1, ...);
```

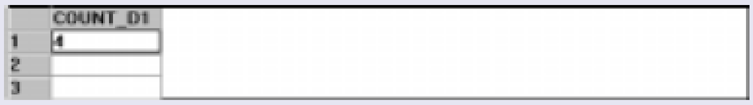
Correlating Queries for Tables with Constraints

In instances where table columns have constraints such as Unique, correlating the script’s queries is the only way to run the script. For example, while recording, if a value is inserted into a table with a Unique key constraint, then it is not possible to insert the same value again. A database error is thrown while executing the script as recorded.

Following is the example that illustrates a query with an Unique constraint using **lrd_save_col** described in the above section. If a new record has to be inserted to the employee database by the personnel department, the new employee should be assigned with next available ID. Two employees cannot have the same ID, so the operator runs a query for the number of records in the employee table. Based on the result, the operator inserts the new employee with the next available number or increment the count by 1.

In the example below, the query returned a count of 4 records and therefore assigned the new employee an ID of 5, or newemp=5. The columns in the table are Name, ID, DOB, Dept, and DeptNo.

```
 lrd_stmt(Csr2, "select count(*) from employees", -1, 1 /*Deferred*/,
       2 /*Ora V7*/, 0);
lrd_bind_col(Csr2, 1, &COUNT_D1, 0, 0);
lrd_exec(Csr2, 0, 0, 0, 0, 0);
lrd_fetch(Csr2, 1, 1, 0, PrintRow2, 0);
```



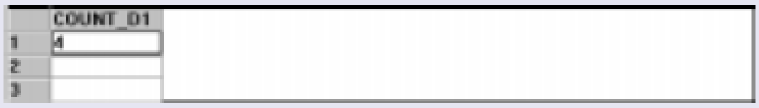
	COUNT_D1
1	4
2	
3	

```
lrd_stmt(Csr2, "\t\tdeclare\n    \tPROCEDURE new_emp (empno in integer, "
        "newemp out integer) is\n\t\tbegin\n\t\t\tnewemp:=empno+1;\n\t\tend "
        "new_emp;\n\t    begin\n\t\t\tnew_emp(:empno,:newemp);\n\t\tend;", -
1, -1
        /*Deferred*/, 2 /*Ora V7*/, 0);
lrd_assign_bind(Csr2, "empno", "4", &empno_D2, 0, 0, 0);
lrd_assign_bind(Csr2, "newemp", "0", &newemp_D3, 0, 0, 0);
lrd_exec(Csr2, 0, 0, 0, 0, 0);
lrd_stmt(Csr2, "insert into employees values ('TOM JONES',5,"
       "'1-JAN-97','R&D',22)", -1, 0 /*Non deferred*/, 2 /*Ora V7*/, 0);
lrd_assign_bind(Csr2, "newemp", "5", &newemp_D4, 0, 0, 0);
lrd_exec(Csr2, 0, 0, 0, 0, 0);
```

If the recorded query is repeated in its current form, the script will attempt to insert the new employee with the same employee number, 5. The script fails since the empno column has a unique constraint. To enable the script to run, replace the constant with a variable. This variable, saved dynamically during script execution, is used for the next query.

Now, after getting the row count during execution, using select count (*), the current row count is returned, not the count generated during execution. Save the variable dynamically by adding **lrd_save_col** before the fetch statement. The sample script displays the changes that are marked in bold.

```
/* Retrieve the number of records in the employees table */
lrd_stmt(Csr1, "select count(*) from employees", -1, 1 /*Deferred*/, 2 /*Ora
V7*/, 0);
lrd_bind_col(Csr1, 1, &COUNT_D1, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_save_col(Csr1, 1, 1, 0, "row_cnt");
lrd_fetch(Csr1, 1, 1, 0, PrintRow2, 0);
```



	COUNT_D1
1	4
2	
3	

After saving the count value from the first fetch, the script derives the next value with a stored procedure, using the formula newemp=empno+1. The placeholder descriptor, newemp, receives this new value when the stored procedure is executed.

7.5 Handling Errors

The Controller and VuGen handle errors while running the ODBC Vuser script. By default, if an error occurs during script execution, the script execution is terminated. To change the default behavior, instruct LoadRunner to continue when an error occurs. Apply this behavior:

- **Globally—**to the entire script, or to a segment of the script
- **Locally—**to a specific function only

Modify Error Handling Globally

To handle errors during script execution can be specified. By default, when the script detects an error, it exits. To continue script execution, even when errors occur, select the **Continue on Error** check box in VuGen's General run-time settings.

Modifying Error Handling Locally

To control error handling for a specific segment of the script, insert an **lr_continue_on_error** statement before and after the desired segment. The new setting is used until the end of the script execution or until another **lr_continue_on_error** statement is issued.

For example, the Continue on Error run-time setting is On and an error is encountered during replay of the following segment, execution.

lrd_stmt (Csr1, "select..."...);

lrd_exec (...);

To continue on error for the entire script except for one segment, enable the Continue on Error setting, and then enclose the segment with lr_continue_on_error statements.

lr_continue_on_error (0);
lrd_stmt (Csr1, "select..."...);
lrd_exec (...);
lr_continue_on_error (1);

In addition to the **lr_continue_on_error** statements, error handling can be controlled using Error Severity Level. **lr_continue_on_error** statements detect all types of errors-database related, invalid parameters, etc. To terminate execution only when a database operation error occurs (Error Code 2009), set a function's severity level. All functions that perform a database operation use severity levels, indicated by the function's final parameter, *miDBErrorSeverity*.

Note: The *miDBErrorSeverity* parameter tells the Vuser whether to continue the script execution on encountering an error. The default 0 indicates that script should be aborted. To instruct the Vuser to continue script execution even after an error, change the severity level from 0 to 1.

7.6 Configure the run-time settings

The run-time settings control the Vuser behavior during script execution. These settings are stored in the file default.cfg, of the Vuser script directory. These settings include loop, log, and timing information.

7.7 Execute the script from VuGen

Save and run the script from VuGen to verify that it runs correctly. After creating the ODBC Vuser script, integrate it into a LoadRunner scenario on either a Windows or UNIX platform.

8 Summary

Many companies are utilizing ODBC compliant databases to deploy in their applications hence performance testing these environments has become more common. Successfully writing script to imitate a user's action and to check query response time is made fairly easy when using LoadRunner. In order to write a good script, planning must be done to ensure that proper correlations and parameterizations are in place.

Load tester has world class consultants with intimate knowledge of ODBC communication from an administrator perspective, as well as a performance engineer view. Load tester helps companies ensure their applications deployed using ODBC compliant database are scalable and meet the performance objectives required for service level agreements.

Happy ODBC!!!

About Author

Shainesh Baheti is a Performance Engineer with Misys. Shainesh has over 30 months of experience in Performance Engineering and Software Quality Assurance services. Shainesh has extensive experience in Performance testing tool LoadRunner and sound knowledge in Financial and Healthcare domain.

Contact
Shainesh.baheti@misys.com, shaineshbaheti@gmail.com