*Author: Courtney Claussen – Sybase IQ Technical Evangelist*
*Contributor: Bruce McManus – Director of Customer Support at Sybase*

# Getting Started with SAP Sybase IQ Column Store Analytics Server

## Lesson 7: Row-Level Versioning

## Table of Contents

# 1. Introduction

In your SAP Sybase IQ database, load, edit or delete data performance is sometimes sacrificed for column storage and increased analytical speed. The row level versioning (RLV) data store can be used to process row-level updates, inserts and deletes in real time, which is useful for improving the performance in some cases.

RLV data storage, introduced in SAP Sybase IQ 16, is optimized for writes. A table can be RLV enabled, which allows multiple transactions to modify different rows of same table concurrently. This minimizes lock contention and allows for low latency writes.

In this lesson, you will learn how the row level versioning data store works, as well as how to monitor locks.  This is a useful store for writing data to tables, but the required memory usage needs to be considered.

**Speed up the server!**

## 2. How RLV Works

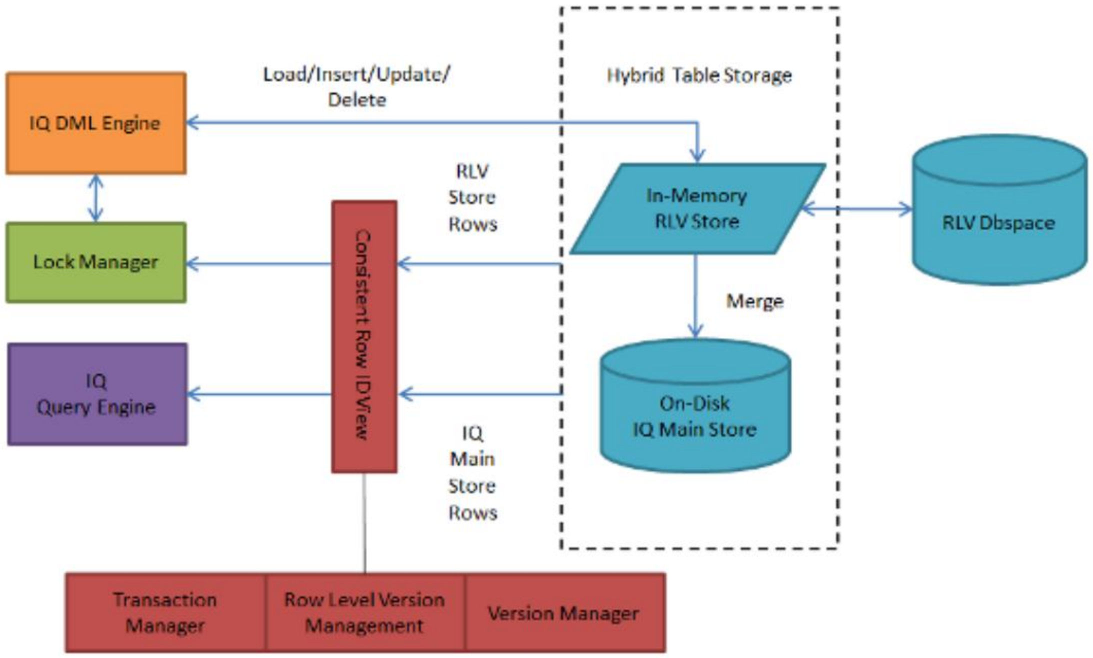
In-memory row-level versioning is a hybrid storage model which optimizes writing data. For any RLV enabled table, when data is loaded or written it is stored in the RLV store in real-time, which has a preconfigured size. The RLV store is later merged into the IQ main store when it reaches a threshold or at a time limit. Then, memory is freed from the RLV store, and the newly written data can be used in the IQ main store for analytics.

*Note: RLV can only be used in simplex configurations, at the current time.*

### *2.1 Architecture*

Row-level storage is enabled in a hybrid table storage, which is made up of RLV and IQ main stores. Logically, the data is stored in the single hybrid table storage, but the server will track the actual data location when querying and manipulating data.

The transaction manager controls the flow of transactions, blocking and allowing them as required. When a single table-level versioning (TLV) transaction is writing, the transaction manager will block all other writes to the table. During row-level versioning transactions, however, only other writes to that row are blocked.



### *2.2 Memory Storage*

When starting the database, memory is set aside for the RLV store. This memory is completely independent of the IQ main cache. Any RLV enabled table is able to write rows to that store, until it is full.

The size of the RLV store should be determined based on the number of tables, load frequency and merge frequency. For most systems, 250-500MB per table marked for each RLV enabled table is sufficient. This can be set by the “-iqrlvmem” startup flag. If this flag is not specified, the value defaults to 2GB.

## 2.3 Lock Management

In SAP Sybase IQ, locks occur automatically. They are used to ensure consistency between concurrent transactions. Without RLV, as in versions of IQ earlier than 16 or tables without RLV enabled, when a table is locked no other users can have write access to the entire table, but they can gain read access. All of a single transaction’s locks are held by the server until a commit or rollback.

In RLV enabled tables, locks are handled differently to allow concurrent writes to the same table, on different rows. These RLV enabled tables have schema locks, row locks and write-intent locks. These locks are defined as follows:

- Schema locks lock the schema of the table, preventing transactions from being disturbed by a difference in columns or other schema changes.
- Write-intent locks are held on tables, if they intend to write to some row in the future. Many requestors can hold a write-intent lock at the same time, as this lock does not imply current writes. A prerequisite to this lock is that the table can be accessed by read-write transactions with row-level snapshot versioning. All write-intent locks must be released before locks on the entire table can be held.
- Row locks allow the holder to write to any column of the row. These locks can only be obtained if the table already has a write-intent lock. This lock is used for updating or deleting rows, as there can only be one holder of the lock on each row at a time.

# 3. RLV Tutorial

In this tutorial, we will configure our environment to allow for row-level versioning. We can then create a table which is flagged for row-level versioning, load data and see the memory usage. We will then look at locks on the database as transactions are executed on RLV-enabled tables.

## 3.1 *Configuration*

Open "tpch.cfg", in the /opt/sybase/TPCHDB directory (or similar directory, as chosen in previous lessons). Add the following line:

```
-iqrlvmem 500
```

This gives the RLV store 500MB of memory. Now, when we run our server, it can store up to 500MB in the RLV store.

Start the IQ server, by running the following command in the terminal:

```
start_iq @tpch.cfg tpch.db
```

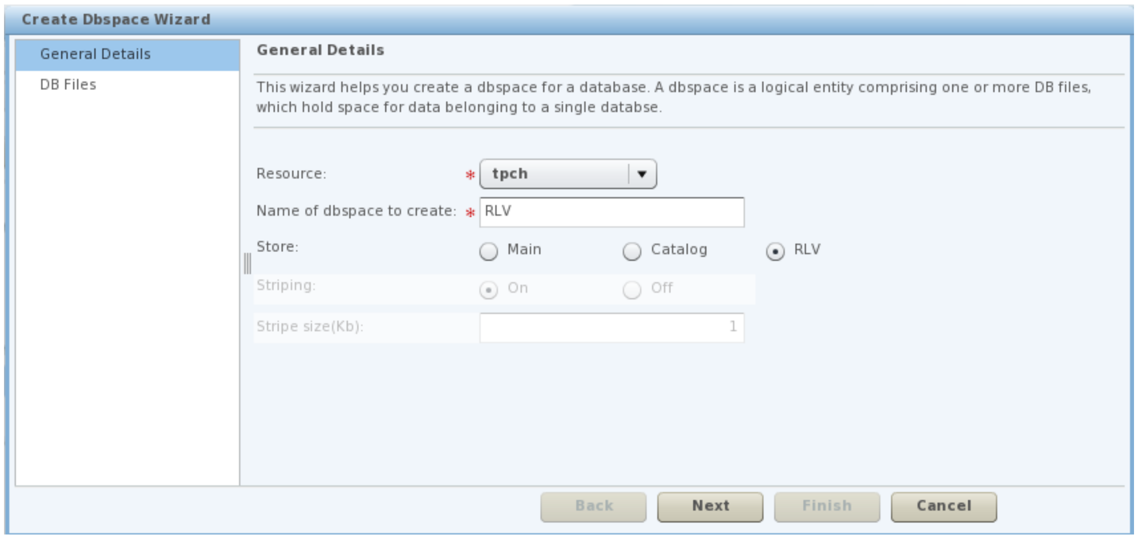
Now open Sybase Control Center:

```
$SYBASE/SCC-3_2/bin/scc.sh
```

From SCC, we will create a RLV dbspace, in a similar way to the dbspace created in Lesson 3, Create Schema and Load Data.

Open the Administration Console for the "tpch" server. Then, expand IQ Servers, followed by Space Management. Click the drop-down for Dbspaces and click "New…".

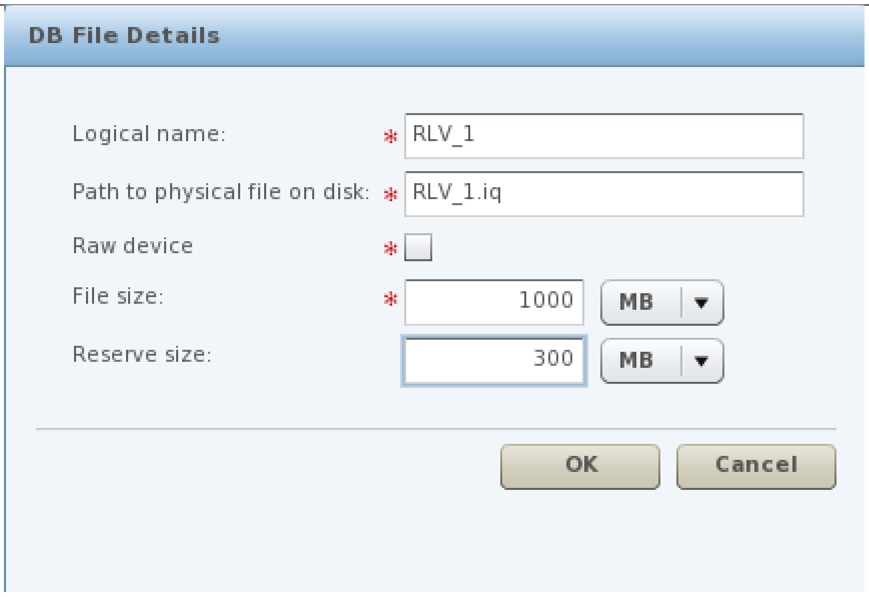
Add a new dbspace in RLV store.



This wizard helps you create a dbspace for a database. A dbspace is a logical entity comprising one or more DB files, which hold space for data belonging to a single databse.

Resource: * tpch
Name of dbspace to create: * RLV
Store: Main Catalog [✓ RLV]
Striping: [✓ On] Off
Stripe size(Kb): 1

Click "Next". Create a dbfile, with size 1000MB.



DB File Details

| | |
|---|---|
| Logical name: * | RLV_1 |
| Path to physical file on disk: * | RLV_1.iq |
| Raw device * | ☐ |
| File size: * | 1000 MB |
| Reserve size: | 300 MB |

Click “OK” and then “Finish” on the resulting screen.

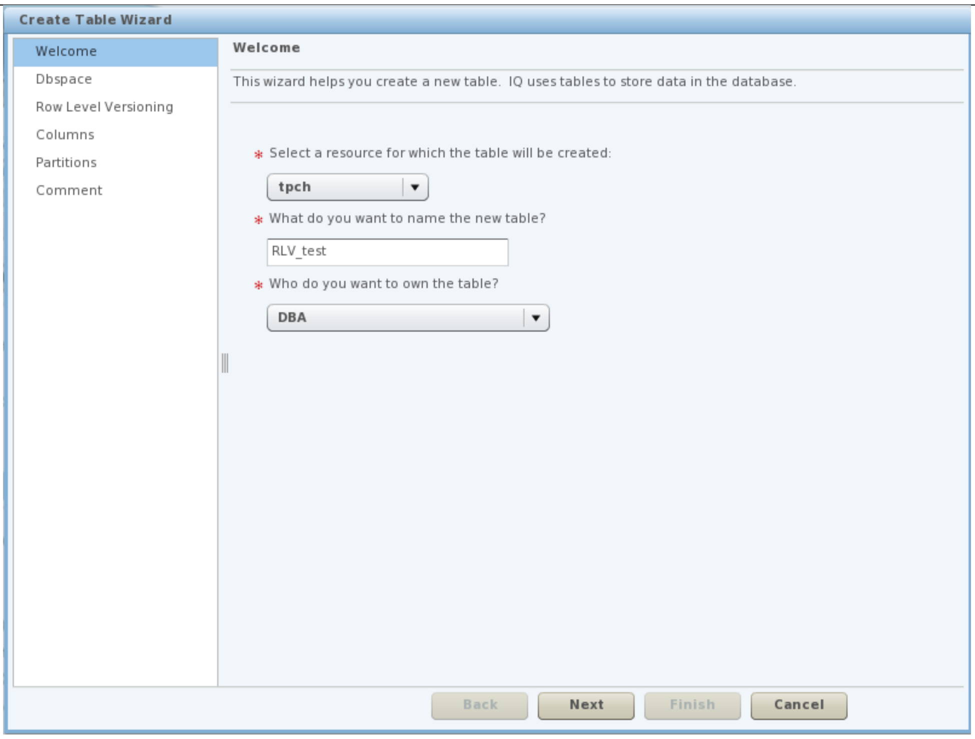
The RLV dbspace, as just created, holds the persistent logs and data from the tables which have RLV storage enabled. The reserve size is used by data structures during table operations, such as load.

## 3.2 *Create Table and Load Data*

Any given table in the database must be flagged for RLV operation for the write-optimized storage to be used.

*Note: RLV cannot be enabled on a table with foreign keys. Catalog, temporary and global temporary tables are also not supported.*

In the Administration Console, expand IQ Servers, followed by Schema Objects and Tables. Then click the drop-down beside “Tables” and click “New…”. Follow the wizard by filling out the following fields:



Welcome

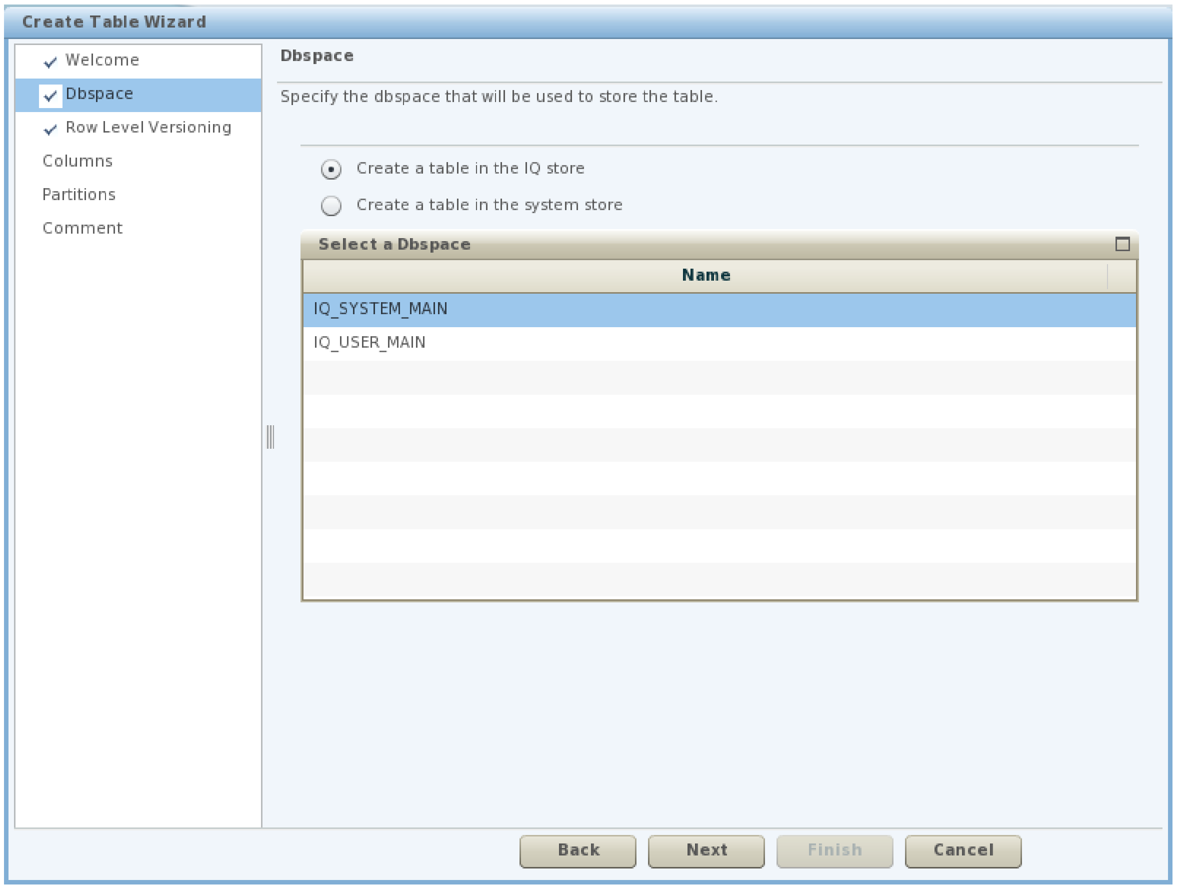
This wizard helps you create a new table. IQ uses tables to store data in the database.

* Select a resource for which the table will be created: tpch

* What do you want to name the new table? RLV_test

* Who do you want to own the table? DBA

Click “Next”. Create the table in the “IQ_SYSTEM_MAIN” dbspace.



Dbspace

Specify the dbspace that will be used to store the table.

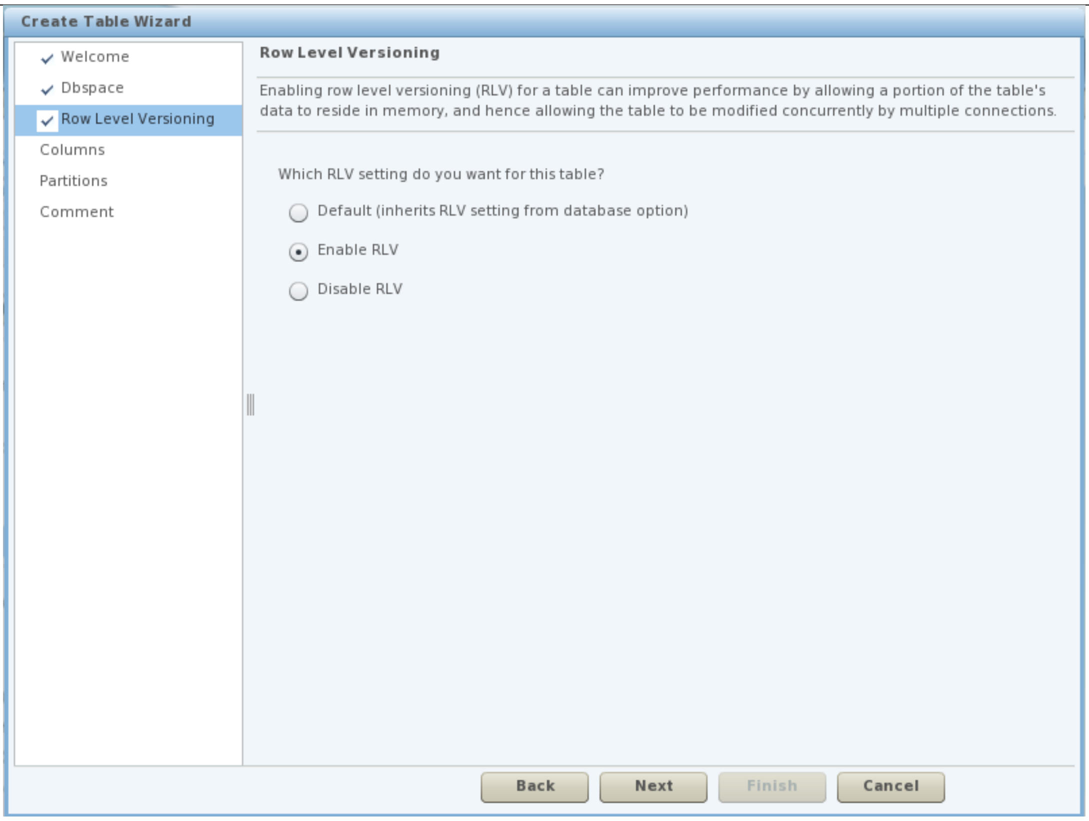
(●) Create a table in the IQ store

( ) Create a table in the system store

Select a Dbspace

| Name |
|---|
| IQ_SYSTEM_MAIN |
| IQ_USER_MAIN |

Click “Next”. Enable RLV.



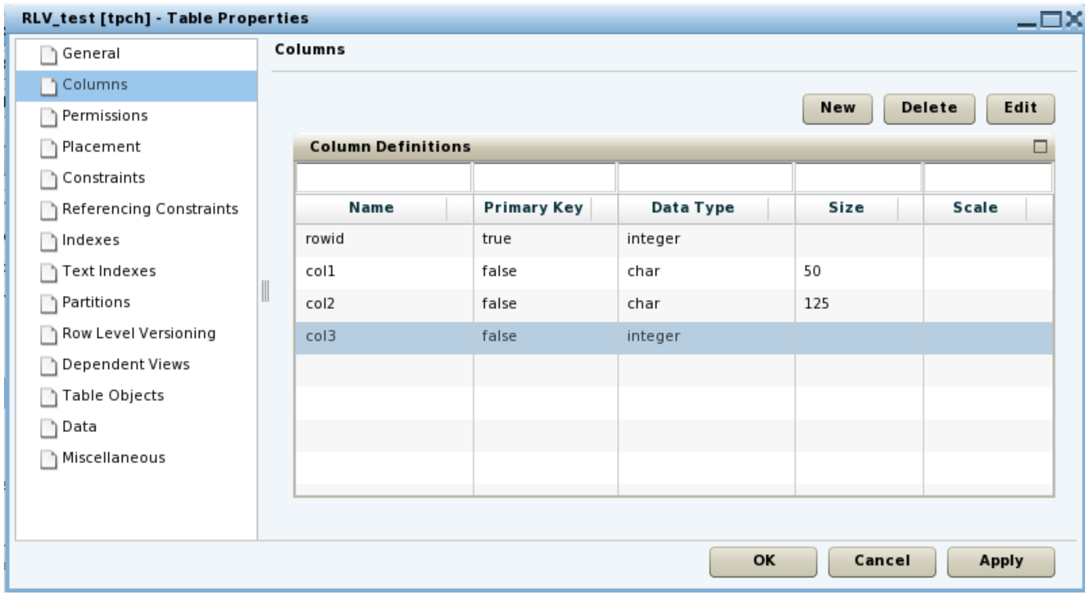
**Row Level Versioning**

Enabling row level versioning (RLV) for a table can improve performance by allowing a portion of the table's data to reside in memory, and hence allowing the table to be modified concurrently by multiple connections.

Which RLV setting do you want for this table?

- ○ Default (inherits RLV setting from database option)
- ● Enable RLV
- ○ Disable RLV

Click “Next”. Use the column wizard to add some test columns, all stored in “IQ_SYSTEM_MAIN”



**Columns**

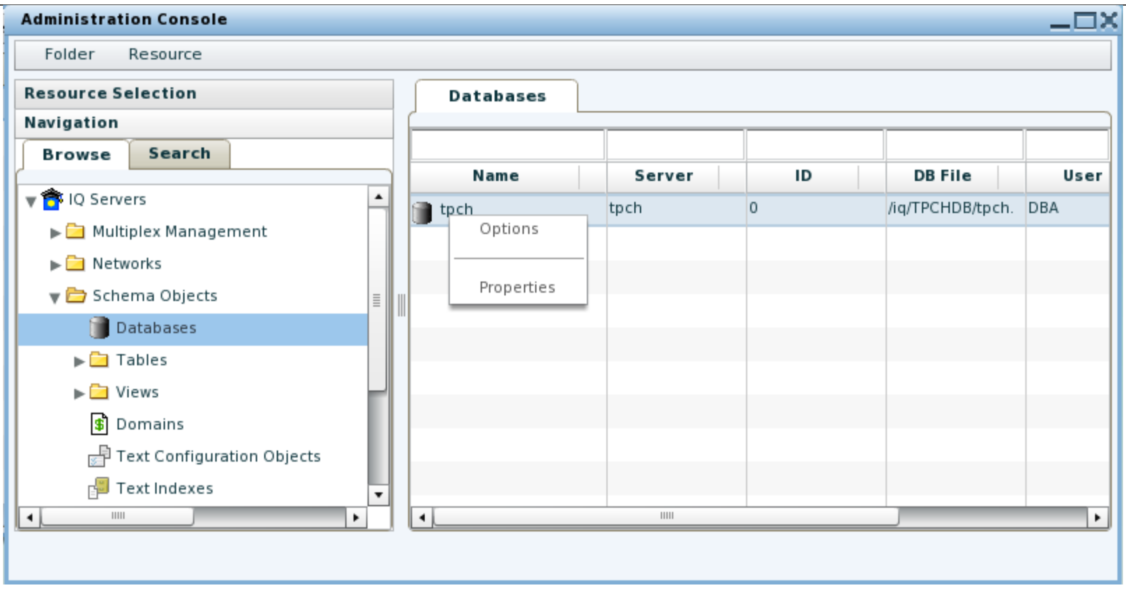
Column Definitions

| Name | Primary Key | Data Type | Size | Scale |
|---|---|---|---|---|
| rowid | true | integer | | |
| col1 | false | char | 50 | |
| col2 | false | char | 125 | |
| col3 | false | integer | | |

Click “Next”, “Next” and “Finish”, not adding any partitions.

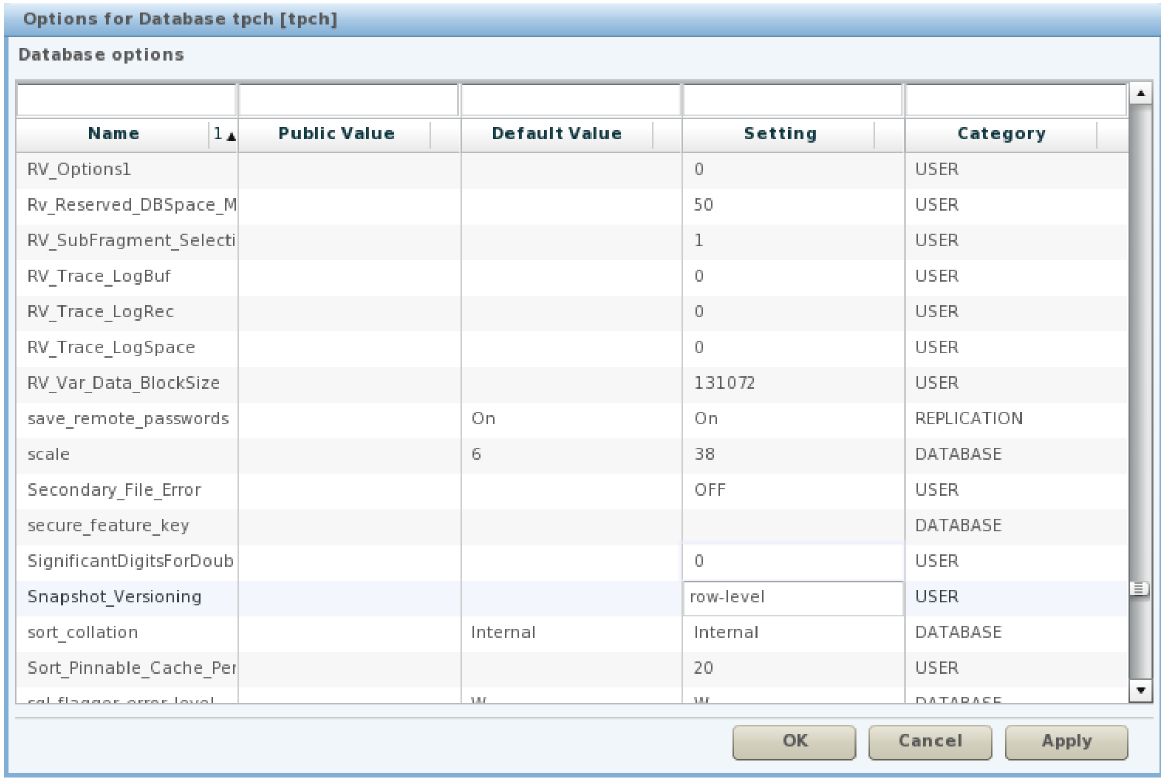
Now, we will set snapshot versioning to row-level.

In the left pane of the Administration Console, expand IQ Servers, Schema Objects and click on Database. In the drop-down beside the database name (tpch, in this example), click “Options”.



| Name | Server | ID | DB File | User |
|---|---|---|---|---|
| tpch | tpch | 0 | /iq/TPCHDB/tpch. | DBA |

In the options window, scroll down to find “Snapshot_Versioning” and change the value to “row-level”



| Name | Public Value | Default Value | Setting | Category |
|---|---|---|---|---|
| RV_Options1 | | | 0 | USER |
| Rv_Reserved_DBSpace_M | | | 50 | USER |
| RV_SubFragment_Selecti | | | 1 | USER |
| RV_Trace_LogBuf | | | 0 | USER |
| RV_Trace_LogRec | | | 0 | USER |
| RV_Trace_LogSpace | | | 0 | USER |
| RV_Var_Data_BlockSize | | | 131072 | USER |
| save_remote_passwords | | On | On | REPLICATION |
| scale | | 6 | 38 | DATABASE |
| Secondary_File_Error | | | OFF | USER |
| secure_feature_key | | | | DATABASE |
| SignificantDigitsForDoub | | | 0 | USER |
| Snapshot_Versioning | | | row-level | USER |
| sort_collation | | Internal | Internal | DATABASE |
| Sort_Pinnable_Cache_Per | | | 20 | USER |

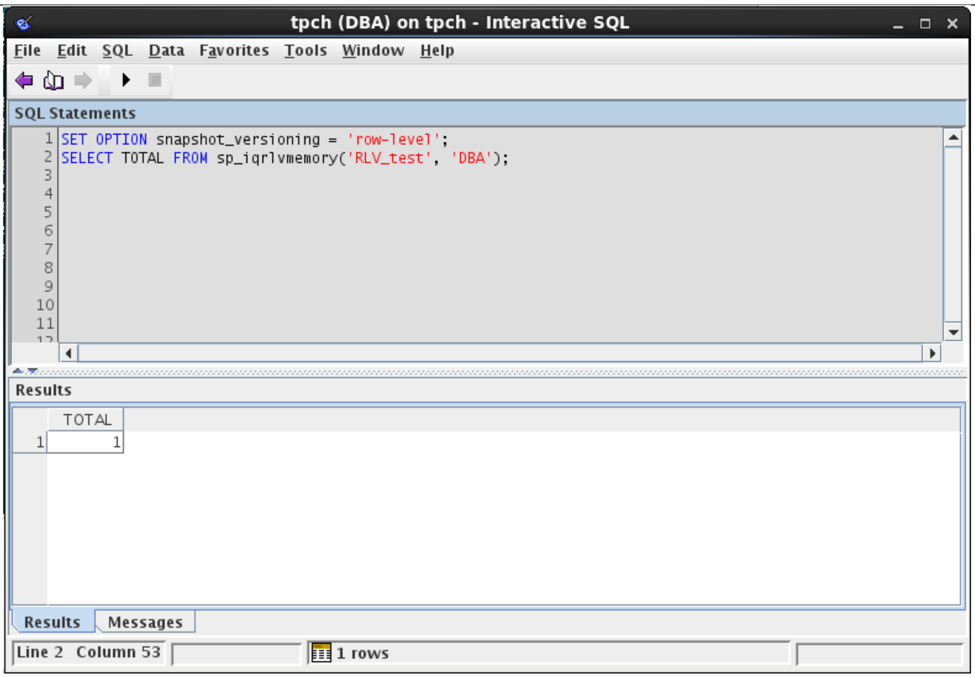
Click “Apply” and “OK”

We will perform the next steps in Interactive SQL. In a terminal window, execute the following command:

```
dbisql -datasource “tpchdb”
```

Check memory usage before data is loaded into the RLV table.

```
SELECT TOTAL FROM sp_iqrlvmemory(‘RLV_test’, ‘DBA’);
```


tpch (DBA) on tpch - Interactive SQL

```
SET OPTION snapshot_versioning = 'row-level';
SELECT TOTAL FROM sp_iqrlvmemory('RLV_test', 'DBA');
```

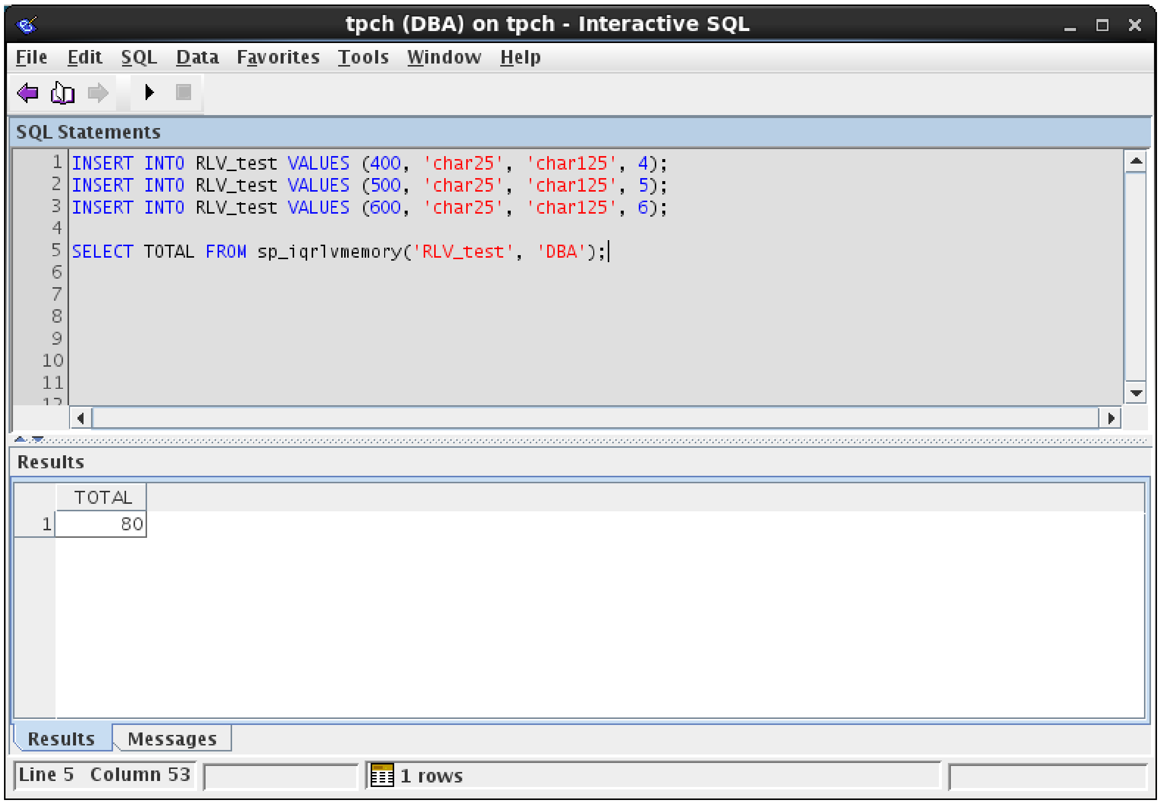
```
   TOTAL
1      1
```

Insert data into your created table, changing values based on created columns.

```
INSERT INTO RLV_test VALUES (1, 'char25', 'char125', 4);
INSERT INTO RLV_test VALUES (2, 'char25', 'char125', 5);
INSERT INTO RLV_test VALUES (3, 'char25', 'char125', 6);
```

Check memory usage again, after the data has been loaded.

```
SELECT TOTAL FROM sp_iqrlvmemory('RLV_test', 'DBA');
```


tpch (DBA) on tpch - Interactive SQL

```
INSERT INTO RLV_test VALUES (400, 'char25', 'char125', 4);
INSERT INTO RLV_test VALUES (500, 'char25', 'char125', 5);
INSERT INTO RLV_test VALUES (600, 'char25', 'char125', 6);

SELECT TOTAL FROM sp_iqrlvmemory('RLV_test', 'DBA');
```

```
   TOTAL
1     80
```

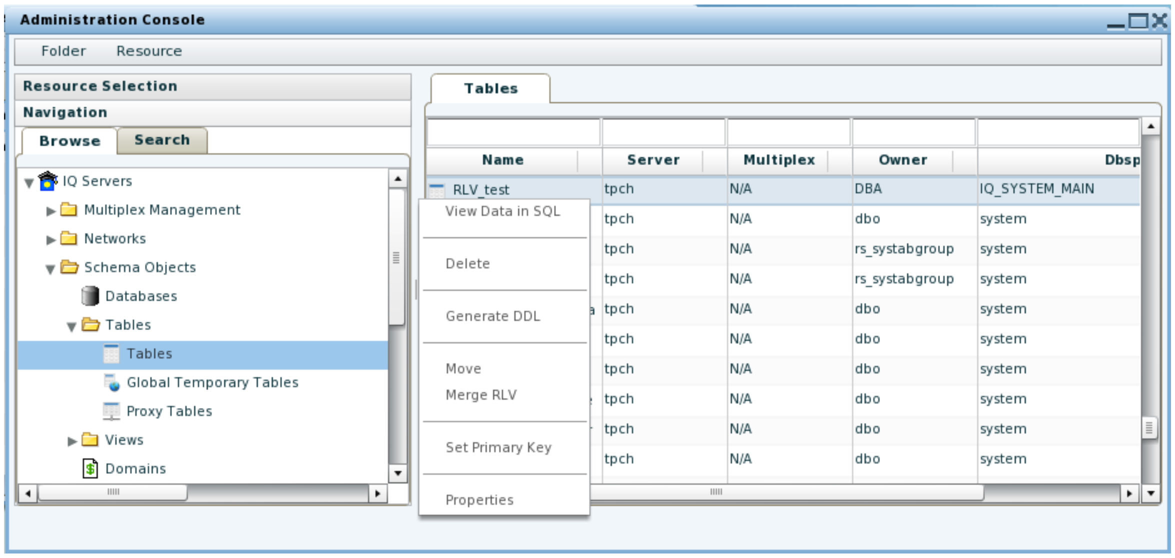
Notice that the total memory in the RLV memory store has increased, as the inserted rows were put in the RLV memory store.

## 3.3 *Merge*

The RLV memory will be automatically merged into IQ main store based on time or memory threshold. You can also manually force a merge.

Close Interactive SQL. We do this, to stop any transactions on the RLV_test table.

If not still open, open Sybase Control Center and go to the Tables section of the Administration Console. Then, click the drop-down to the right of the RLV table, "RLV_test", in this example. Choose "Merge RLV".



| Name | Server | Multiplex | Owner | Dbsp |
|---|---|---|---|---|
| RLV_test | tpch | N/A | DBA | IQ_SYSTEM_MAIN |
| | tpch | N/A | dbo | system |
| | tpch | N/A | rs_systabgroup | system |
| | tpch | N/A | rs_systabgroup | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |

In the pop-up, choose "Yes" to proceed.

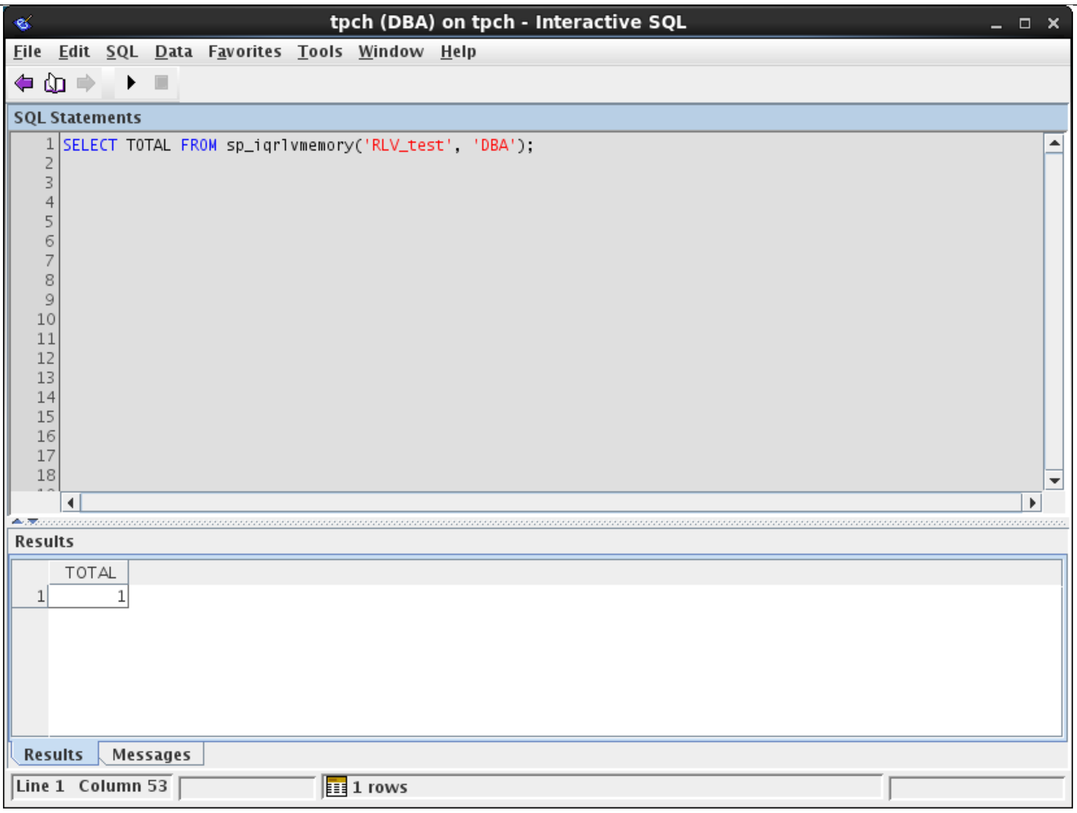
We will now reopen Interactive SQL to check the RLV store memory again.

Execute the following command in a terminal window:

```
dbisql -datasource "tpchdb"
```

Run the following command, to check RLV memory usage again:

```
SELECT TOTAL FROM sp_iqrlvmemory('RLV_test', 'DBA');
```


tpch (DBA) on tpch - Interactive SQL
File Edit SQL Data Favorites Tools Window Help
SQL Statements
Results
Results Messages
Line 1 Column 53
1 rows

```
SELECT TOTAL FROM sp_iqrlvmemory('RLV_test', 'DBA');
```

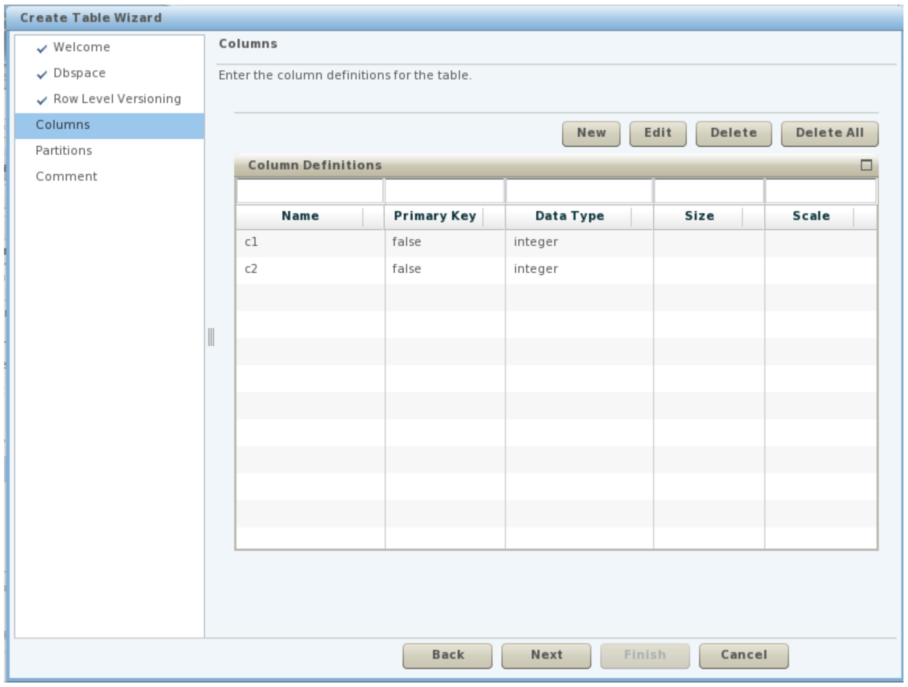
|  | TOTAL |
|---|---|
| 1 | 1 |

Notice the total memory has gone back down.

## 3.4 ***Monitor Write-Intent Locks***

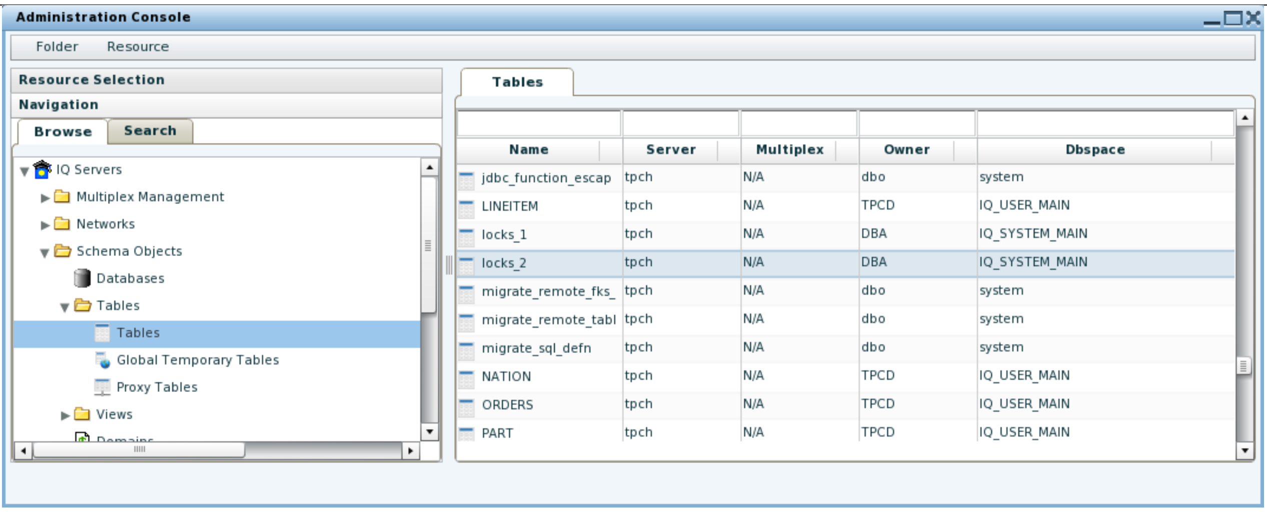
Change “Snapshot_Versioning” to “table_level” in the Administration Console of SCC, as done before, by changing the options of the database.

Create two tables, as earlier, each with RLV enabled. Create columns as you wish.



Enter the column definitions for the table.

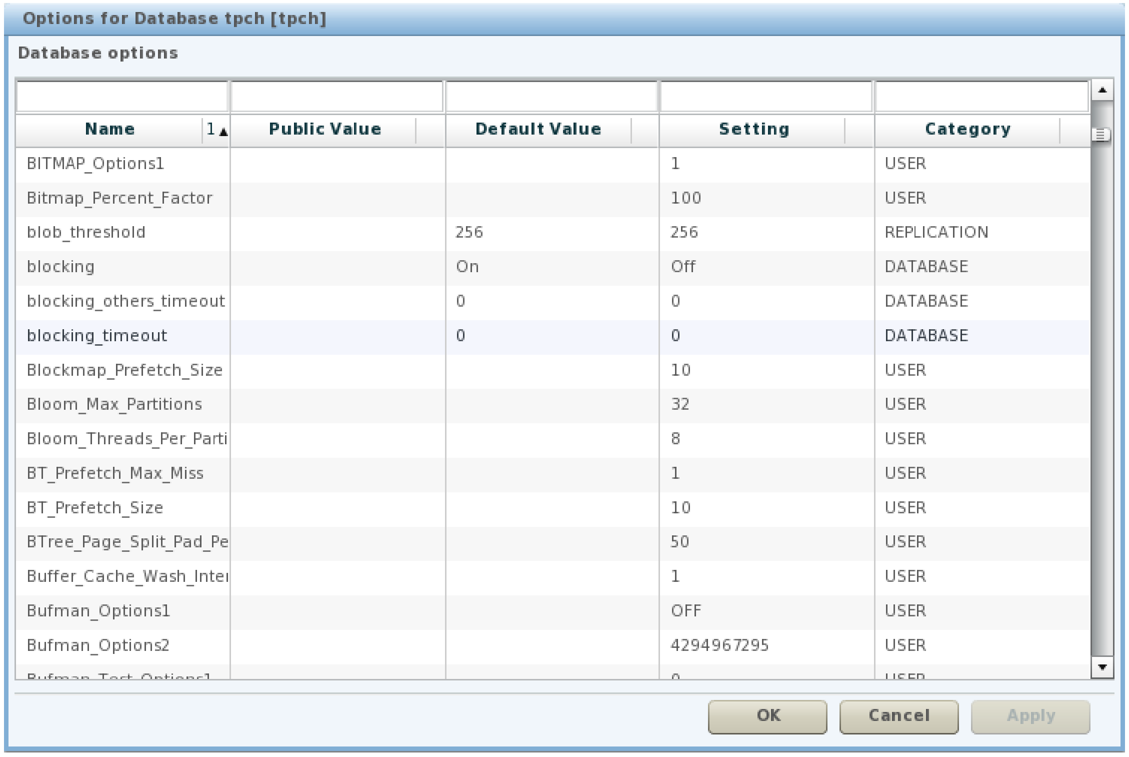
| Name | Primary Key | Data Type | Size | Scale |
|---|---|---|---|---|
| c1 | false | integer |  |  |
| c2 | false | integer |  |  |



| Name | Server | Multiplex | Owner | Dbspace |
|---|---|---|---|---|
| jdbc_function_escap | tpch | N/A | dbo | system |
| LINEITEM | tpch | N/A | TPCD | IQ_USER_MAIN |
| locks_1 | tpch | N/A | DBA | IQ_SYSTEM_MAIN |
| locks_2 | tpch | N/A | DBA | IQ_SYSTEM_MAIN |
| migrate_remote_fks_ | tpch | N/A | dbo | system |
| migrate_remote_tabl | tpch | N/A | dbo | system |
| migrate_sql_defn | tpch | N/A | dbo | system |
| NATION | tpch | N/A | TPCD | IQ_USER_MAIN |
| ORDERS | tpch | N/A | TPCD | IQ_USER_MAIN |
| PART | tpch | N/A | TPCD | IQ_USER_MAIN |

Check the database properties as before, ensuring that "blocking" is on and "blocking_timeout" is 0. Blocking controls the behavior in response to locking conflicts.

When blocking is enabled, blocks occur once a transaction has obtained a lock. This means that once a transaction obtains a lock any other transaction that attempts to obtain that lock must wait, until the original transaction has released the lock or the blocking timeout has been reached. When blocking is disabled, errors occur if a transaction attempts to obtain an already obtained lock.



| Name | Public Value | Default Value | Setting | Category |
|---|---|---|---|---|
| BITMAP_Options1 | | | 1 | USER |
| Bitmap_Percent_Factor | | | 100 | USER |
| blob_threshold | | 256 | 256 | REPLICATION |
| blocking | | On | Off | DATABASE |
| blocking_others_timeout | | 0 | 0 | DATABASE |
| blocking_timeout | | 0 | 0 | DATABASE |
| Blockmap_Prefetch_Size | | | 10 | USER |
| Bloom_Max_Partitions | | | 32 | USER |
| Bloom_Threads_Per_Parti | | | 8 | USER |
| BT_Prefetch_Max_Miss | | | 1 | USER |
| BT_Prefetch_Size | | | 10 | USER |
| BTree_Page_Split_Pad_Pe | | | 50 | USER |
| Buffer_Cache_Wash_Inte | | | 1 | USER |
| Bufman_Options1 | | | OFF | USER |
| Bufman_Options2 | | | 4294967295 | USER |

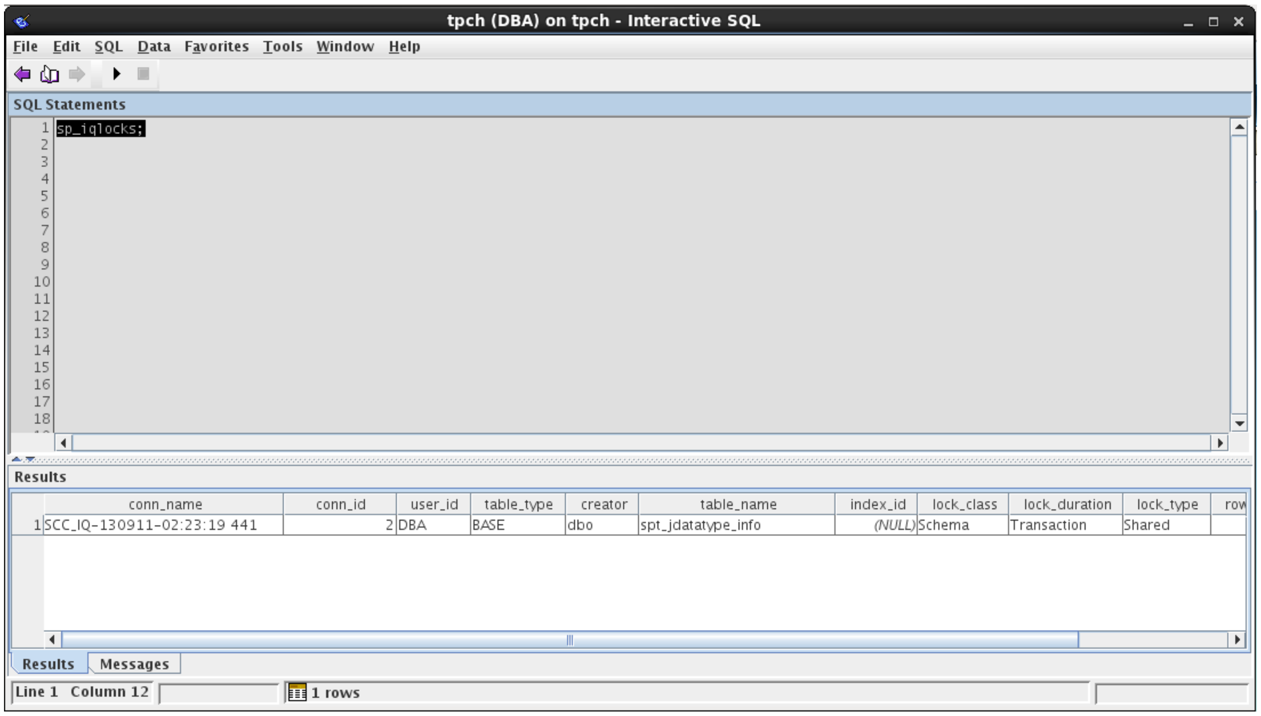
We will now view the current set of database locks, ensuring that no write-intent locks are obtained yet (and thus, are not displayed).

Now, open Interactive SQL, as earlier, running the following command in a terminal window.

```
dbisql -datasource "tpchdb"
```

Run the following SQL statement and click "Execute".

```
sp_iqlocks;
```


tpch (DBA) on tpch - Interactive SQL
File Edit SQL Data Favorites Tools Window Help
SQL Statements
Results
Results Messages
Line 1 Column 12
1 rows

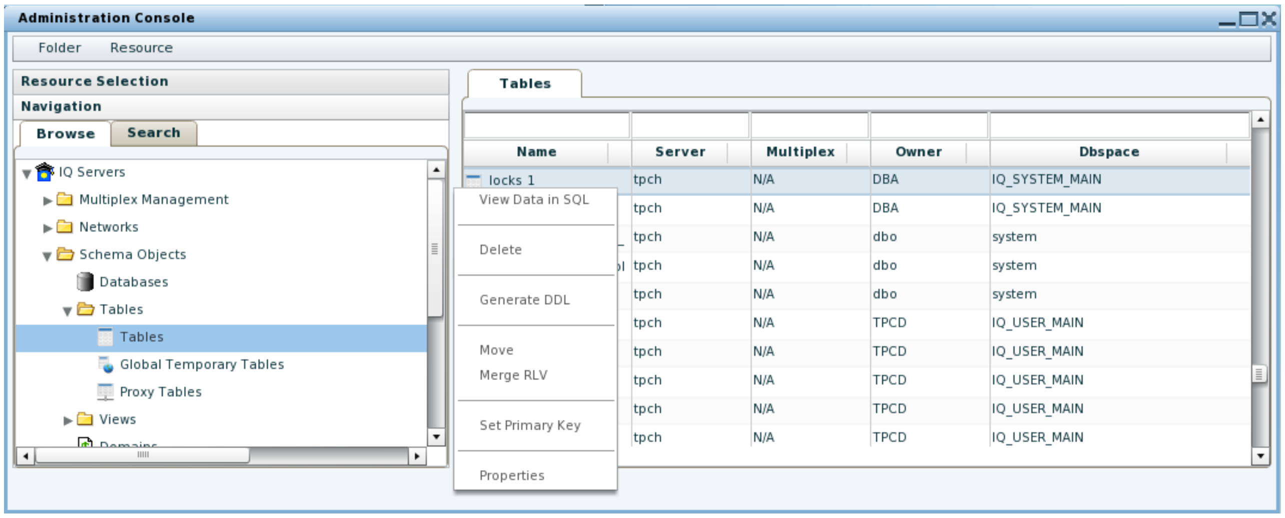
```
sp_iqlocks;
```

| | conn_name | conn_id | user_id | table_type | creator | table_name | index_id | lock_class | lock_duration | lock_type | row |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | dbo | spt_jdatatype_info | (NULL) | Schema | Transaction | Shared | |

Notice that there are no write-intent locks or locks on the newly created lock_1 or lock_2 table.

A write intent lock will have “lock_type” set to “Write”. Each lock has a specified “table_name”, which tells you what table the lock is on.

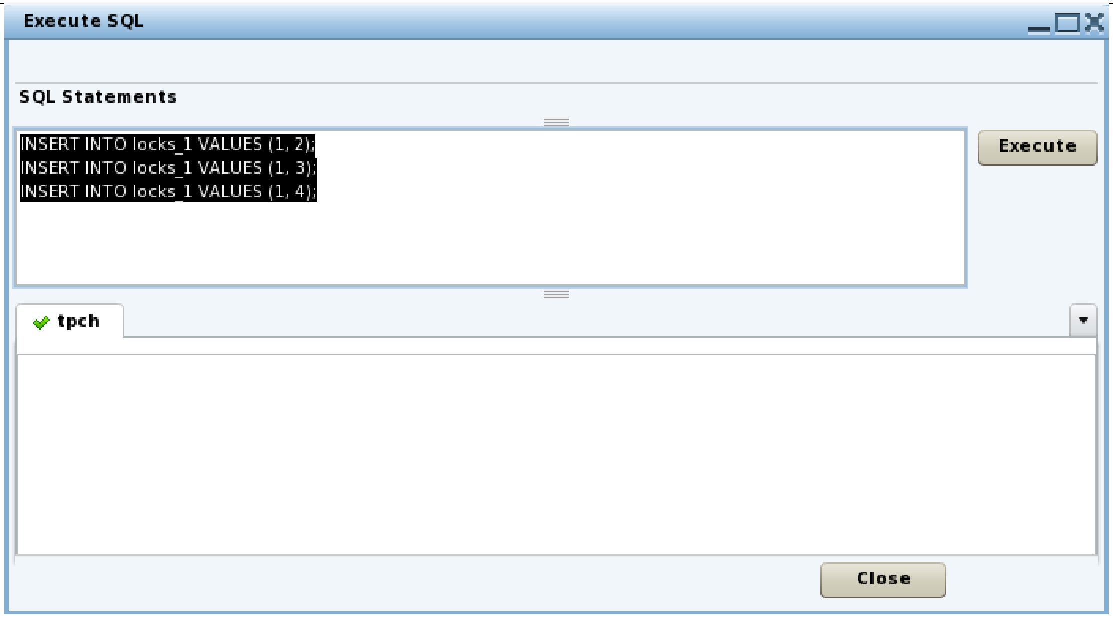
Now, close this window and go back to the Administration Console. View the options for the database, as done in “3.2 Create Table and Load Data” above. Change the “Snapshot_Versioning” value to “row-level”. Click the drop-down arrow and choose “View Data in SQL”



| Name | Server | Multiplex | Owner | Dbspace |
|---|---|---|---|---|
| locks 1 | tpch | N/A | DBA | IQ_SYSTEM_MAIN |
| | tpch | N/A | DBA | IQ_SYSTEM_MAIN |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | dbo | system |
| | tpch | N/A | TPCD | IQ_USER_MAIN |
| | tpch | N/A | TPCD | IQ_USER_MAIN |
| | tpch | N/A | TPCD | IQ_USER_MAIN |
| | tpch | N/A | TPCD | IQ_USER_MAIN |
| | tpch | N/A | TPCD | IQ_USER_MAIN |

Change the values in the SQL Statements textbox to the following, and execute.

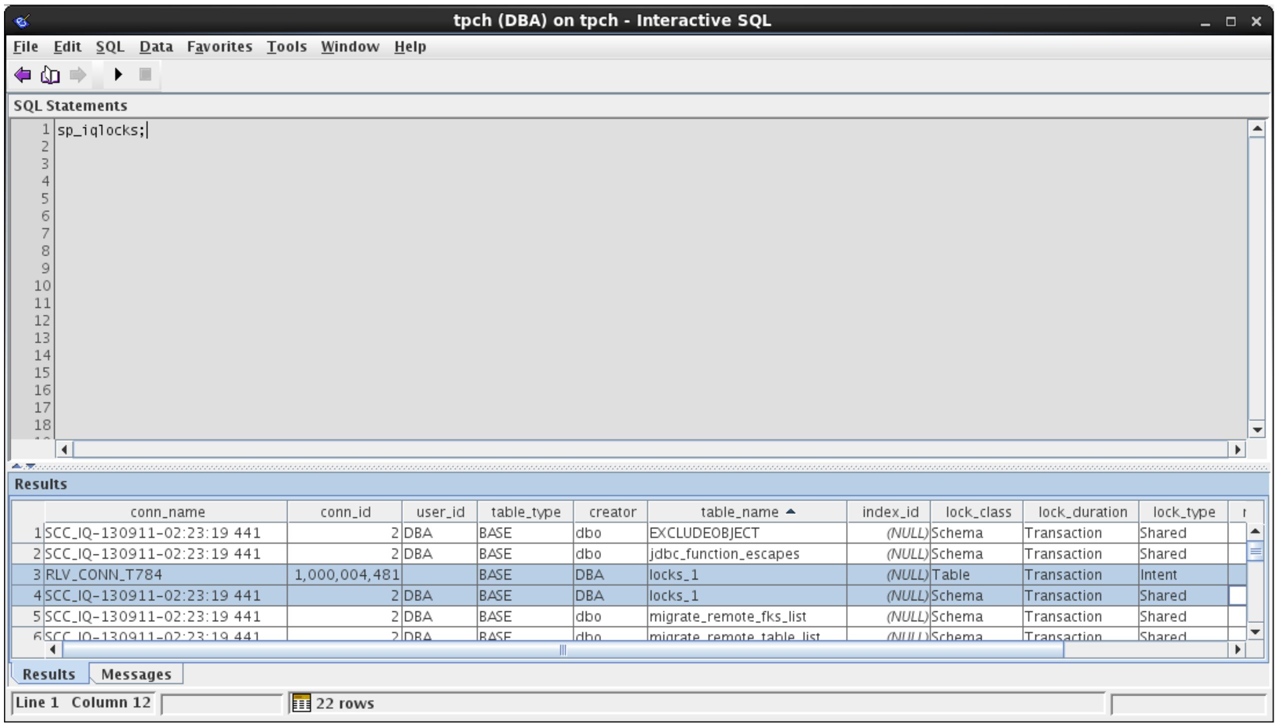
```
INSERT INTO locks_1 VALUES (1, 2);
INSERT INTO locks_1 VALUES (2, 2);
INSERT INTO locks_1 VALUES (3, 2);
```



```
INSERT INTO locks_1 VALUES (1, 2);
INSERT INTO locks_1 VALUES (1, 3);
INSERT INTO locks_1 VALUES (1, 4);
```

This creates the RLV-enabled portion of the table in memory.

Run "sp_iqlocks" in Interactive SQL again.

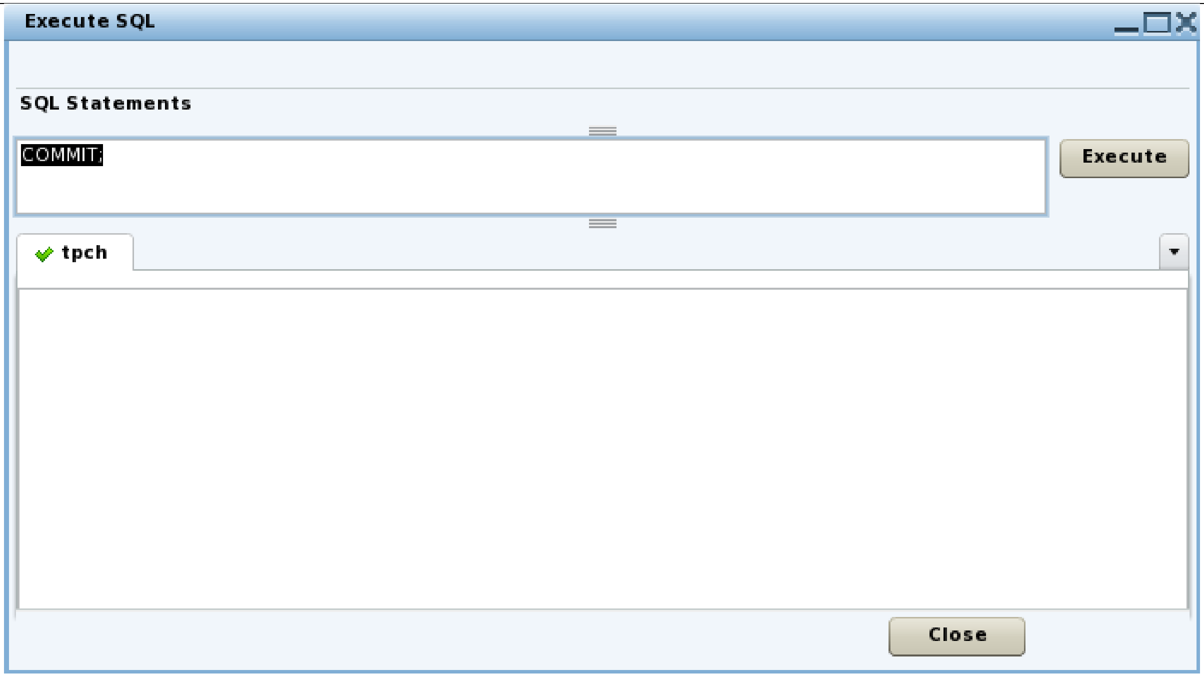

tpch (DBA) on tpch - Interactive SQL
File Edit SQL Data Favorites Tools Window Help
SQL Statements
Results
Results Messages
Line 1 Column 12
22 rows

```
sp_iqlocks;
```

| | conn_name | conn_id | user_id | table_type | creator | table_name | index_id | lock_class | lock_duration | lock_type |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | dbo | EXCLUDEOBJECT | (NULL) | Schema | Transaction | Shared |
| 2 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | dbo | jdbc_function_escapes | (NULL) | Schema | Transaction | Shared |
| 3 | RLV_CONN_T784 | 1,000,004,481 | | BASE | DBA | locks_1 | (NULL) | Table | Transaction | Intent |
| 4 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | DBA | locks_1 | (NULL) | Schema | Transaction | Shared |
| 5 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | dbo | migrate_remote_fks_list | (NULL) | Schema | Transaction | Shared |
| 6 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | dbo | migrate_remote_table_list | (NULL) | Schema | Transaction | Shared |

You will see a schema lock and a table lock for the locks_1 table, with the uncommitted insert statements.

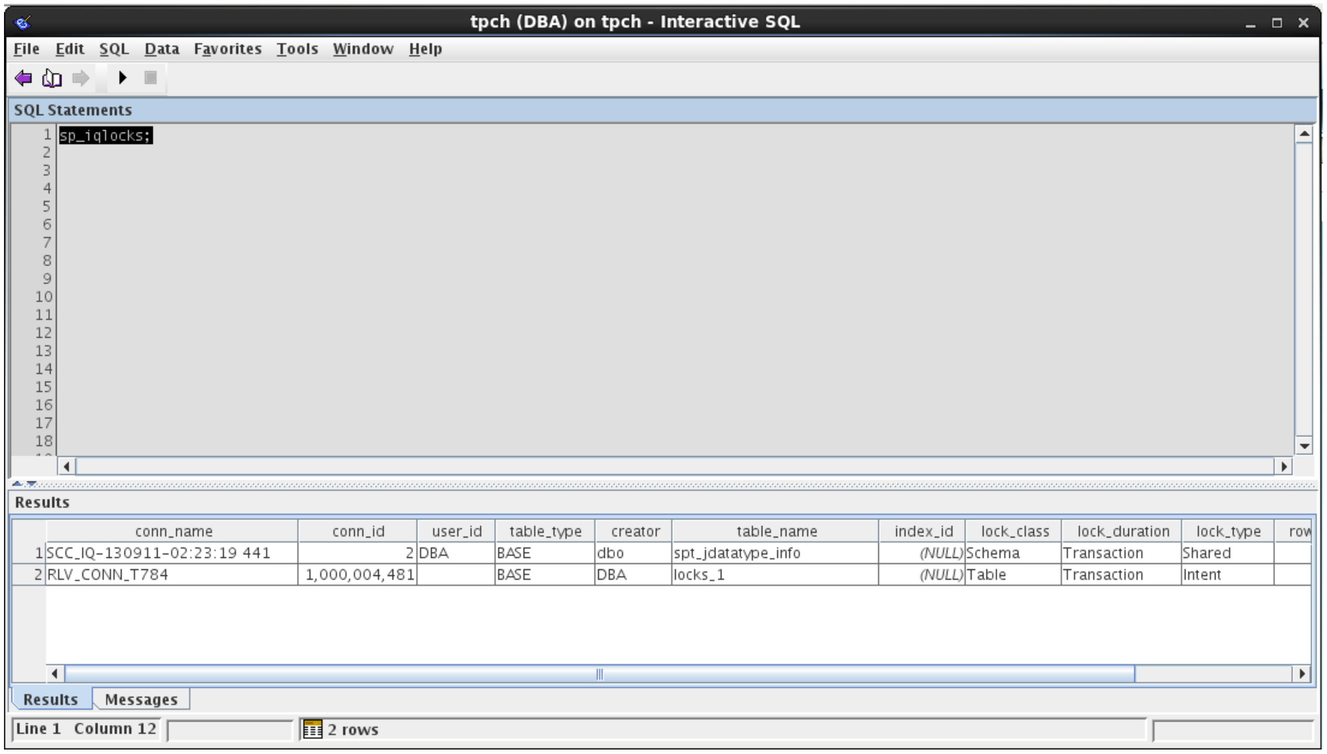
Go back to the Administration Console in SCC, and in the "View Data in SQL" pop-up of the "locks_1" table, run "COMMIT;" Commit releases the locks held by the transaction, in this case, it will be the schema lock.



```
COMMIT;
```

Now, run “sp_iqlocks” in Interactive SQL again.

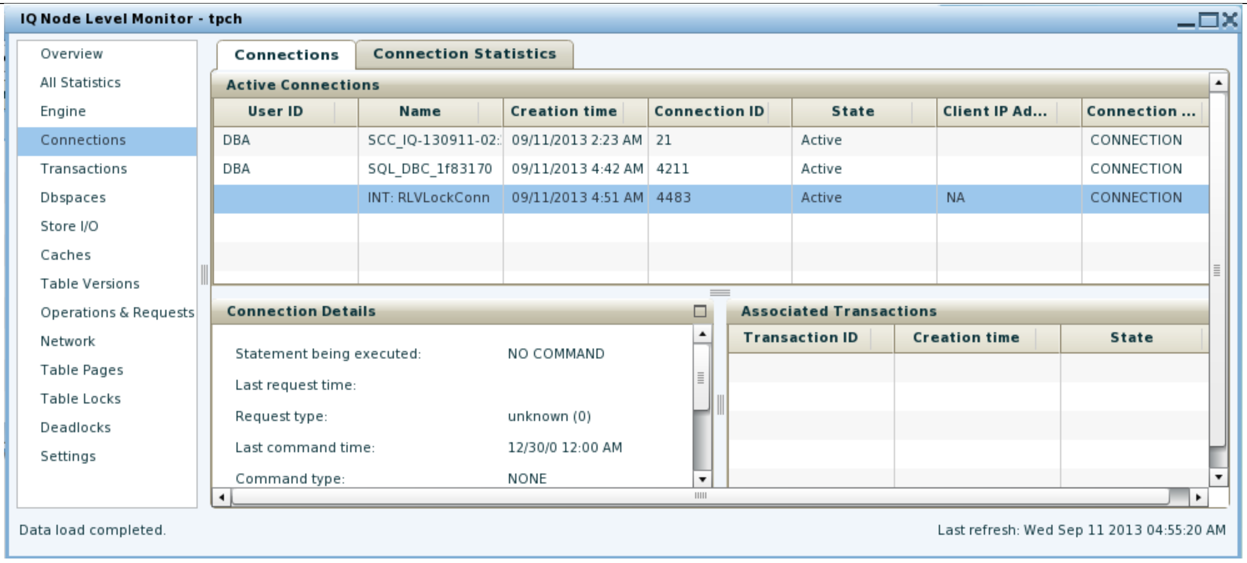
You will now see that the schema lock is gone. The rows inserted in the table are now in the RLV-enabled portion of the table, and this write-intent lock is held by that part of the table.


tpch (DBA) on tpch - Interactive SQL
File Edit SQL Data Favorites Tools Window Help
SQL Statements
Results
Results Messages
Line 1 Column 12
2 rows

```
sp_iqlocks;
```

| | conn_name | conn_id | user_id | table_type | creator | table_name | index_id | lock_class | lock_duration | lock_type | row |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SCC_IQ-130911-02:23:19 441 | 2 | DBA | BASE | dbo | spt_jdatatype_info | (NULL) | Schema | Transaction | Shared | |
| 2 | RLV_CONN_T784 | 1,000,004,481 | | BASE | DBA | locks_1 | (NULL) | Table | Transaction | Intent | |

These locks can also be viewed, in less detail, in the “Connections” tab of the “Monitor Node” of the “tpch” server in SCC.



Active Connections

| User ID | Name | Creation time | Connection ID | State | Client IP Ad... | Connection ... |
|---|---|---|---|---|---|---|
| DBA | SCC_IQ-130911-02: | 09/11/2013 2:23 AM | 21 | Active | | CONNECTION |
| DBA | SQL_DBC_1f83170 | 09/11/2013 4:42 AM | 4211 | Active | | CONNECTION |
| | INT: RLVLockConn | 09/11/2013 4:51 AM | 4483 | Active | NA | CONNECTION |

Connection Details

| | |
|---|---|
| Statement being executed: | NO COMMAND |
| Last request time: | |
| Request type: | unknown (0) |
| Last command time: | 12/30/0 12:00 AM |
| Command type: | NONE |

Associated Transactions

| Transaction ID | Creation time | State |
|---|---|---|

Data load completed.

Last refresh: Wed Sep 11 2013 04:55:20 AM

## 4. Summary

In this chapter you have learned about the new row-level versioning and how it works with SAP Sybase IQ. You have done a lesson, in which you created a table which performs write operations in the RLV memory store and merges the data to IQ main store. You can now use in-memory row-level versioning to optimize writes, when applicable.

More information and tutorials can be found at:
http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01840.1601/doc/pdf/iqrlv.pdf

www.sap.com