



Alpinist

Course 8

**Globals
MTables**

Agenda



- Globals
 - What are Globals
 - Global types
 - *Cluster Globals*
 - *Session Globals*
 - *Thread Globals*
 - Override Cluster Globals
 - Reset Cluster Globals
 - Using Globals in SQL Statements
- Mtables
 - What Is an Mtable
 - How to create Mtable
 - MTables Storage Settings
 - How to work with Mtable

A vertical photograph on the left side of the slide shows a person wearing a red jacket and dark pants standing on a rocky, snow-covered mountain peak. The person is looking out over a vast, snow-covered mountain range under a clear blue sky with some light clouds. The overall scene is bright and high-contrast.

Globals

Globals refer to variables that are accessible from anywhere within a program. This means that they can be used in any class or method without needing to be explicitly passed as arguments.

Global types

Fabric supports three types of Globals:

1. **Cluster Globals** – defined in the Fabric Studio. Can be accessed by any function or component (unless defined for a specific LUT, and then can be accessed only by functions defined under this specific LUT).
2. **Session Globals** – created on-the-fly at a session level and accessible only within that specific session.
3. **Thread Globals** - created on-the-fly at a GET level and are accessible only within that specific sync process

Cluster Globals

Define Cluster Globals

There are two locations within Fabric Studio to define Cluster Globals:

- 1. SharedGlobals.java:** Found under *Shared Objects/Java/src*. Globals defined here are accessible to all project components, including BW flows, functions, web services, common tables, etc.
- 2. Globals.java:** Located under each *Logical Unit/Java/src* (except for web services). Globals defined in this file are only accessible to components created within that specific Logical Unit.

All Globals defined in either the *SharedGlobals.java* or *Globals.java* are deployed with their default (initial) values as specified on their declaration.

```
public class SharedGlobals {  
  
    @desc("")  
    @category("")  
    public static String globalName = "";  
  
}
```



Note:

1. If a Global is defined at both the Shared Objects level and the Logical Unit level, the definition in the Logical Unit takes precedence within its scope. Other Logical Units will use the Shared Objects definition.
2. Globals have to be of type String

Cluster Globals

When a global is defined in SharedGlobals.java, it is "inherited" by all Logical Unit types. However, when a global is defined in a specific Logical Unit (Globals.java), it is limited to that specific Logical Unit.

Global defined in SharedGlobals.java

```

Implementation > Shared Objects > Java > src > com > k2view > cdbms > usercode > common >
1 // Shared Globals
2 // Shared Globals
3 // Shared Globals
4 // Shared Globals
5 package com.k2view.cdbms.usercode.common;
6
7 import com.k2view.cdbms.shared.utils.UserCodeDescribe.*;
8
9 public class SharedGlobals {
10
11     @desc("CLUSTER_GLOBAL_TEST description")
12     @category("CLUSTER_GLOBAL_TESTCategory")
13     public static String CLUSTER_GLOBAL_TEST = "CLUSTER_GLOBAL_TEST_VALUE";
14
15 }
16
17
    
```

Global defined in Globals.java

```

Implementation > Logical Units > Customer > Java > src > com > k2view > cdbms > usercode
1 // LU Globals
2 // LU Globals
3 // LU Globals
4 // LU Globals
5 package com.k2view.cdbms.usercode.lu.Customer;
6
7 import com.k2view.cdbms.usercode.common.SharedGlobals;
8 import com.k2view.cdbms.shared.utils.UserCodeDescribe.*;
9
10 public class Globals extends SharedGlobals {
11
12     @desc("Lut globals description")
13     @category("LUT_GLOBAL_TESTCategory")
14     public static String LUT_GLOBAL_TEST = "LUT_GLOBAL_TEST_VALUE";
15
16 }
17
    
```

Use SET command to see the Globals defined

```

fabric>set;
key                                     value
-----
ENVIRONMENT                             |_dev
EXECUTION_ID                            |cb23f730-3be5-4212-8320-5c6eod8f2917
Global.Customer.CLUSTER_GLOBAL_TEST     |CLUSTER_GLOBAL_TEST_VALUE
Global.Customer.LUT_GLOBAL_TEST         |LUT_GLOBAL_TEST_VALUE
Global.k2_ref.CLUSTER_GLOBAL_TEST       |CLUSTER_GLOBAL_TEST_VALUE
Global.k2_ws.CLUSTER_GLOBAL_TEST        |CLUSTER_GLOBAL_TEST_VALUE
LOG_ID                                   |31040000000000e
PROJECT_NAME                             |Alpinist
SCOPE                                    |empty
SYNC                                     |ON
USERNAME                                 |shani.alpinist@k2view.com
USER_ROLES                               |t-b78d1b60-aa9d-47a9-bf84-703d7a1ce600_k
    
```



Note:

The first "Global" prefix is displayed since the SET command also displays non Globals variables

Cluster Globals

Retrieve Cluster Globals values

- **Using Fabric SET Command:**
Use the following code to fetch the Cluster Global value: `fabric().fetch("set CLUSTER_GLOBAL_TEST").firstValue();`
- **Using BW Actors:**
Use `FabricSetRead` to retrieve the Cluster Global value.
- **Using Java Code:**
To get the Cluster Global value, use one of the following methods:
 - `getGlobal(String globalName, String lu)` - Retrieves the Logical Unit's global value for this session.
Example: `UserCode.getGlobal("CLUSTER_GLOBAL_TEST", "Customer");`
 - `getGlobal(String globalName)` - Retrieves the global value for this session.
Example: `UserCode.getGlobal("CLUSTER_GLOBAL_TEST");`
Note: If there is a conflict in global values between Logical Units, an exception will be thrown.
- **Using the Global name directly:**
Every `Logic.java` file imports the Global declaration files, allowing you to directly use the Global name in your code.

```
// Import shared Globals
import com.k2view.cdbms.shared.Globals;
// Import Globals from the Logical Unit
import static com.k2view.cdbms.usercode.lu.<LU name>.Globals.*;

if(CLUSTER_GLOBAL_TEST.equals("...")) {
    // Perform logic based on the global value
}
```

Session Globals

Globals can also be declared and used at a session level. In this case, the Globals are defined on-the-fly within the session and are terminated once the session ends.

- To create and set (or modify) a session global value, use the following command:
fabric().execute("set GLOBAL_NAME=GLOBAL_VALUE");
- To retrieve the value of a session global, use:
fabric().fetch("set GLOBAL_NAME").firstValue();
- For BW flows, use FabricSetRead and FabricSet BW Actors to read and set the session Globals.



Note:

Specifying a session variable does NOT create it as Shared Object. Therefore, it is not displayed under each LUT (if it is not specifically created for it,) and will not have the Global or LUT prefixes.

```
fabric>set NEW_SESSION_GLOBAL = 100;
(1 row affected)
fabric>set;
|key|value
+-----+-----+
|ENVIRONMENT|dev
|EXECUTION_ID|fd050d4b-94f0-469a-b6e1-4f6
|Global.Customer.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE
|Global.Customer.LUT_GLOBAL_TEST|LUT_GLOBAL_TEST_VALUE
|Global.Customer.RECORDS_LIMIT|7
|Global.Customer.SCHEMA_NAME|public
|Global.k2_ref.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE
|Global.k2_ref.RECORDS_LIMIT|7
|Global.k2_ref.SCHEMA_NAME|public
|Global.k2_ws.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE
|Global.k2_ws.RECORDS_LIMIT|7
|Global.k2_ws.SCHEMA_NAME|public
|LOG_ID|2804000000000000b
|NEW_SESSION_GLOBAL|100
```

Thread Globals

Thread Globals are designed for use exclusively within GET operations, enabling the sharing of values across populations, decision functions and enrichment functions. These Globals are defined dynamically, on-the-fly, within the thread and automatically terminated when the GET operation completes.

Methods to work with Thread Globals:

- **setThreadglobals(String key, Object value)** - create Thread Global
- **getThreadGlobals(String Key)** - get Thread Global value
- **clearThreadGlobals()** - Clear all Thread Globals, created on the thread level

Note:

- Unlike Cluster or Session Globals, Thread Globals also support data types beyond just strings.
- Thread Global cannot override Session Global.



Override Cluster Globals

If the final keyword is added to a Cluster Global definition, the Global's value becomes immutable and can only be changed by redeploy.

```
@desc("")
@category("")
public final static String RECORDS_LIMIT = "7";
```

When final is not declared, Cluster Global value can be overridden with or without re-deploying the project, at the following levels:

- Implementation level – by updating the SharedGlobals.java or Globals.java files and re-deploying.
- Environment level – by modifying the Environment file in Fabric Studio and re-deploying.
- Cluster level – at run time, using the **set_global global** command.
`set_global global CLUSTER_GLOBAL_TEST =newValue;`
- Session level – at run time, using the **SET** command.
`fabric().execute("set CLUSTER_GLOBAL_TEST =newValue");`

A vertical photograph on the left side of the slide shows a person wearing a red jacket and dark pants standing on a rocky, snow-covered mountain peak. The background features a vast, snow-covered mountain range under a clear blue sky with some light clouds.

Override Cluster Globals

Override Cluster Globals From Studio

In case the value of a Cluster Global was not changed using `set_global` command, you can change the value of the global in the `SharedGlobals.java` or `Globals.java` file, and redeploy

Override Cluster Globals From Environment file

The default values for Cluster Globals, defined in the `SharedGlobals.java` and `Globals.java` files, are used for the `_dev` environment, which is the default environment for each cluster. However, these defaults can be overridden for other environments.

When creating a new environment, a Globals tab is available that allows you to modify the values of the Cluster Globals (as defined in `SharedGlobals.java` and `Globals.java`) specific to that environment.

Override Cluster Globals

Override Cluster Globals per Environment

Once an Environment is applied to a cluster, the Cluster Globals will use the default values defined for that environment.

To switch environments and assign the appropriate global values, use the SET ENVIRONMENT command:

- set environment='UAT' to apply the default values defined for the UAT environment.
- set environment='_dev' to revert to the default values defined in the SharedGlobals.java and Globals.java files for the development environment.

The screenshot shows the k2view interface with the 'Environments' tab active. The 'UAT' environment is selected, and the 'Cluster Globals' table is displayed. The table shows global values for the UAT environment, including 'UAT_CLUSTER_GLOBAL_TEST', 'RECORDS_LIMIT', 'SCHEMA_NAME', and 'UAT_LUT_GLOBAL_TEST_Vi'.

Logical Unit	Category	Name	Value	Comments
	CLUSTER_GLOBAL_TESTsetCategory	CLUSTER_GLOBAL_TEST	UAT_CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST de
		RECORDS_LIMIT	5	
		SCHEMA_NAME	public	
Customer	LUT_GLOBAL_TESTCategory	LUT_GLOBAL_TEST	UAT_LUT_GLOBAL_TEST_Vi	Lut globals description

Override Cluster Globals

Overriding Cluster Globals on a cluster level

Cluster Global value can be overridden for the entire cluster, for all running sessions.

- Use `set_global global` command to change the Cluster Global on runtime.

```
set_global global '*.CLUSTER_GLOBAL_TEST=value1'
```

```
fabric>set_global global '*.CLUSTER_GLOBAL_TEST=CLUSTER_GLOBAL_TEST_VALUE_1';  
(1 row affected)  
fabric>set;  
|key                               |value  
+-----+-----+  
|ENVIRONMENT                       |_dev  
|EXECUTION_ID                      |9d88dd4f-ea58-44eb-9b36-901cf0195e51  
|Global.Customer.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE_1  
|Global.Customer.LUT_GLOBAL_TEST    |LUT_GLOBAL_TEST_VALUE  
|Global.k2_ref.CLUSTER_GLOBAL_TEST  |CLUSTER_GLOBAL_TEST_VALUE_1  
|Global.k2_ws.CLUSTER_GLOBAL_TEST   |CLUSTER_GLOBAL_TEST_VALUE_1
```



Note:

Set_global can be used only for Cluster Globals. You cannot define Cluster Global on-the-fly

Override Cluster Globals

Overriding Cluster Globals on a cluster level

Shared Global can be overridden for a specific LUT.

```
set_global global 'Customer.CLUSTER_GLOBAL_TEST=value2'
```

```
fabric>set_global global 'Customer.CLUSTER_GLOBAL_TEST=Customer.CLUSTER_GLOBAL_TEST_VALUE_2';
(1 row affected)
fabric>set;
|key|value
-----|-----
|ENVIRONMENT|_dev
|EXECUTION_ID|30b23a89-98fc-4d51-87f2-26c4cb3a730b
|Global.Customer.CLUSTER_GLOBAL_TEST|Customer.CLUSTER_GLOBAL_TEST_VALUE_2
|Global.Customer.LUT_GLOBAL_TEST|LUT_GLOBAL_TEST_VALUE
|Global.k2_ref.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE_1
|Global.k2_ws.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE_1
|LOG_ID|310400000000000f
|PROJECT_NAME|Alpinist
|SCOPE|empty
|SYNC|ON
|USERNAME|shani.alpinist@k2view.com
|USER_ROLES|t-b78d1b60-aa9d-47a9-bf84-703d7a1ce600_k2v_admin,t-b78d1b60-aa9
```



Note:

If the set_global global command is using a specific LUT, when fetching this value later, using SET command or getGlobal function, **Fabric will throw exception in case value is different between the LUTs.**

Override Cluster Globals

Override Cluster Globals per session on runtime

- Use set command to change the Cluster Global on runtime, for a specific session.
- The Cluster Global can be changed for a specific LUT or for all
 - Set `*.CLUSTER_GLOBAL_TEST=value1;`
 - `set Customer.CLUSTER_GLOBAL_TEST=value2;`



Note:

When a Cluster Global is overridden at the session level, it will appear as a "new" global in the SET command, which is a presentation feature indicating that the value has changed.

```
fabric>set Customer.CLUSTER_GLOBAL_TEST=100;
(1 row affected)
fabric>set;
|key|value|
-----|-----|
|CUSTOMER.CLUSTER_GLOBAL_TEST|100|
|ENVIRONMENT|DEV|
|EXECUTION_ID|348-8345-ca09-49dc-8814-9f73aafb1c35|
|Global.Customer.CLUSTER_GLOBAL_TEST|100|
|Global.Customer.LUT_GLOBAL_TEST|LUT_GLOBAL_TEST_VALUE|
|Global.Customer.RECORDS_LIMIT|7|
|Global.Customer.SCHEMA_NAME|public|
|Global.k2_ref.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE|
|Global.k2_ref.RECORDS_LIMIT|7|
|Global.k2_ref.SCHEMA_NAME|public|
|Global.k2_ws.CLUSTER_GLOBAL_TEST|CLUSTER_GLOBAL_TEST_VALUE|
|Global.k2_ws.RECORDS_LIMIT|7|
|Global.k2_ws.SCHEMA_NAME|public|
```

Override Cluster Globals

Cluster Globals Levels and Proprieties

Level	Priority	Where to define	How to override	How to retrieve	How to reset
Global	4	SharedGlobals.java Globals.java	Deploy	GetGlobal function SET command BW – fabricSetRead GLOBAL_NAME	
Environment	3	Environment	Deploy		
Runtime - cluster	2	Runtime	set_global command		set_global global '*.GLOBAL_NAME'
Runtime - session	1	Runtime	SET command BW - fabricSet	GetGlobal function SET command BW - fabricSet	SET GLOBAL_NAME=""

Notes:

- A Global value will always take the value set on the lowest level:
 - First priority – Session level (SET command)
 - Second priority – Cluster level (set_global command)
 - Third priority – Environment file
 - Last priority – SharedGlobals.java / Globals.java

Override Cluster Globals

- When a Cluster Global is overridden at a lower level, subsequent changes made at a higher level will not affect the lower levels. To allow changes at the upper level to propagate to the lower levels, the global must be reset at the lower level.

For example:

- If a Cluster Global is modified at the session level using the SET command, changing it at the cluster level using `set_global` will not update the session-level value.
 - If a Cluster Global is modified at the cluster level using the `set_global` command, changing it in Studio and redeploying will not affect the value.
- A Cluster Global can be overridden for a specific LUT level. After being overridden, the global behaves independently:
 - Changing the Cluster Global with `set_global`, without specifying a specific LUT, will not alter its value.
 - To reconnect the global to its higher levels, it must first be reset using the specific LUT.
 - Cluster Globals can be modified or overridden at the cluster or session level. `getGlobal`, the SET command, and BW Actors will return the session-level Global value, while using the global name directly will always return the cluster-level value.

Reset Cluster Global value

Resetting a Session Global to the Cluster Global Value

To reset a Cluster Global to its default value, use the `set_global` command **without specifying** `=<PARAM_VALUE>`.

Resetting will revert the value to its default as defined in the Environment file, or if no environment is in use, to its default in `SharedGlobals.java` or `Globals.java`.

```
fabric>set_global global 'Customer.CLUSTER_GLOBAL_TEST';
(1 row affected)
fabric>set;
```

key	value
ENVIRONMENT	_dev
EXECUTION_ID	96f4fe5b-fe73-408a-9fd4-d40380b64037
Global.Customer.CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST_VALUE
Global.Customer.LUT_GLOBAL_TEST	LUT_GLOBAL_TEST_VALUE
Global.k2_ref.CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST_VALUE_1
Global.k2_ws.CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST_VALUE_1

```
fabric>set_global global '*.CLUSTER_GLOBAL_TEST';
(1 row affected)
fabric>set;
```

key	value
ENVIRONMENT	_dev
EXECUTION_ID	96f4fe5b-fe73-408a-9fd4-d40380b64037
Global.Customer.CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST_VALUE
Global.Customer.LUT_GLOBAL_TEST	LUT_GLOBAL_TEST_VALUE
Global.k2_ref.CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST_VALUE
Global.k2_ws.CLUSTER_GLOBAL_TEST	CLUSTER_GLOBAL_TEST_VALUE

Note: If the Global was modified for a specific LUT, you must reset that LUT's Global explicitly. Resetting for *.Global will not reset the specific LUT's Global.

Reset Cluster Global value

Resetting a Session Global to the Cluster Global Value:

Running **set <PARAM_NAME> = ""** will reset the session-level global value to its original value based on the following priority:

1. The value set using the `get_global` command.
2. The value defined in the Environment file.
3. The value defined in `SharedGlobals.java` or `Globals.java`.



Note:

- If the Cluster Global was modified for a specific LUT, you must also reset it specifically.
- The SET command will always display the values for the specific session, which may differ from their current values in the Cluster.

How it works?

Fabric uses the **k2system.global_settings** table to store Globals data and any overridden values, including those set by the **set_global** command and **Environment** settings.

This ensures that the Globals data is retained when Fabric restarts.

When the **set_global** command is executed, Fabric updates the **global_settings** table and uses fabric-jdbc (TCP) to notify other nodes about the change.

Session Globals Integration in Job Execution

Jobs automatically receive the `session_scope` as part of their arguments, which includes Session Globals.

This means that when a Session Global is set and a job is run within the same session, the job will inherit these global values and operate accordingly.

For example:

```
{"session_scope":{"scope":{"EXECUTION_ID":"8f5dcec4-0125-4915-88c8daf8003519eb","GLOBAL_TEST":"10","LOG_ID":"b4050000000000bd"}}
```

The same behavior applies to migration commands.

```
{"FABRIC_COMMAND":"sync_instance TestLU?","JOB_UID":"","session_scope":{"scope":{"EXECUTION_ID":"8f5dcec4-0125-4915-88c8-daf8003519eb","GLOBAL_TEST":"10","LATEST_BATCH":"45d294f5-bf9a-47af-8e9f-070826397321","IS_IN_BATCH_PROCESS_PROCESS":"true","LOG_ID":"b4050000000000bd"},"user":{"type":"Authenticated UserByCredentials","username":"..."
```



Note:

The BroadwayJob actor currently does not receive the `session_scope`. This will be addressed in upcoming releases

Using Globals in SQL Statement

A global can be used in an SQL statement in an LU function. The syntax is: '@[global_name]@'.

For example:

```
public static String SCHEMA_NAME = "public";  
public static Integer RECORDS_LIMIT = 5;
```

- Table population, sourceDbQuery Actor:
select * From **@SCHEMA_NAME@**.activity limit **@RECORDS_LIMIT@**
- Java code:
String sql = "SELECT * From ACTIVITY limit **@RECORDS_LIMIT@**"
ludb().fetch(sql, input1, input2).each(row->{
 yield(row.cells());
});
- Same structure for each BW DB Actors

Globals Best Practice

- Using `set_global` affects the entire cluster. Instead, use Thread globals within your schema functions (such as populations or other LU functions).

For example, if a root function doesn't use the linking field as input, use a session global to add logic that ensures it executes only once.

- Always clear session or thread globals at the end of your function or GET process. This ensures that if another GET runs later in the same session, the session values will be reset.

MTables

What Is an MTable?

An MTable is an object created in Fabric memory from a CSV file. It stores reference data as part of the Fabric project, allowing for fast in-memory lookups during runtime.

MTables are best suited for small, static lists of reference data.



Note:

MTable replaces the Translation tables in Cloud Studio.

MTables

How to create MTable?

1. Using a CSV File in the MTable Folder:

- When deployed, an MTable is created in Fabric memory based on the CSV file's structure and data and made available on all Fabric nodes (other files in the folder are ignored).
- On a Fabric restart, the memory is released, and the MTable is reloaded.
- Each MTable can be accessed from any LU, regardless of where its CSV file is located in the project.

2. At Runtime with MTableLoad Actor:

- A new MTable created at runtime is available on a single node.
- To distribute it across nodes, use `SET CLUSTER_DISTRIBUTE_AFFINITY = ALL`.
- Note: MTables created or updated at runtime are lost after a Fabric restart.



Note:

SET CLUSTER_DISTRIBUTE_AFFINITY = is a new command, for distributing the subsequent command to the specified affinity. Use ALL to distribute the subsequent command to all live nodes (working only on Fabric commands)

MTables

Notes

- Reloading an MTable deletes all existing records.
- If an MTable is created with an existing name, it replaces the previous one in memory when deployed.
- Data lookup can be done using one or more keys. Search indices are created during the first lookup, based on the search keys.



How to work with MTable?

Broadway Actors:

- **MTableLookup** - fetching data from an MTable by the given key(s).
If no keys are provided, the entire MTable dataset is returned (array of objects).

MTableRandom - for fetching a random row from an MTable.
The random selection can be limited by providing an input key(s).
This Actor returns one object only.
- **MTableLoad** - creating a new MTable dataset or replacing an existing one in the Fabric memory.
The MTable is then created on one node and must be distributed to other nodes.



Note:

Search indices for an MTable are created on-the-fly during the first lookup

How to work with MTable?

Java Code:

Use com.k2view.fabric.common.mtable package.
For example:

1. Create MTable:

```
ArrayList<Object[]> rows = new ArrayList<>();  
Object[] row1 = new Object[] { "value1", "value2" };  
Object[] row2 = new Object[] { "value3", "value4" };  
rows.add(row1);  
rows.add(row2);  
MTables.create("mtable_name", new String[]{"col1", "col2"}, rows);
```

2. Read MTable based on keys:

```
MTable mtable = MTables.get("mtable_name");  
Map<String, Object> keys = new HashMap();  
keys.put("lu_name", getLuType().luName);  
List<Map<String, Object>> mtResults = mtable.mapsByKey(keys, MTable.Feature.caseInsensitive);
```

3. Read all records from MTable:

```
AllRows<Map<String, Object>> mtableRows = MTables.get("mtable_name").allMaps();
```

MTables

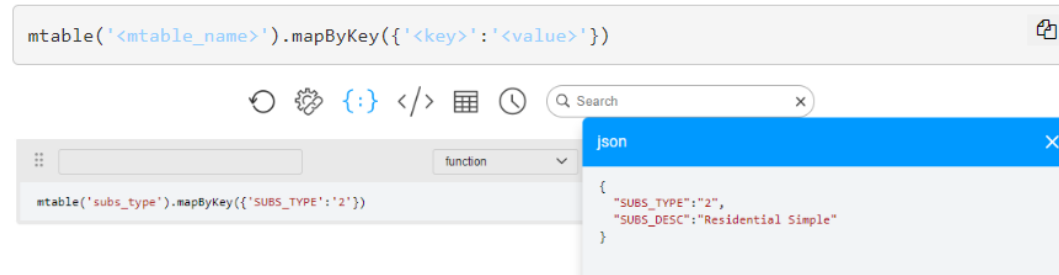
How to work with MTable?

Graphit:

The below syntax returns the first matching MTable row:

`mtable('<mtable_name>').mapByKey({'<key>':'<value>'})`

```
mtable('<mtable_name>').mapByKey({'<key>':'<value>'})
```



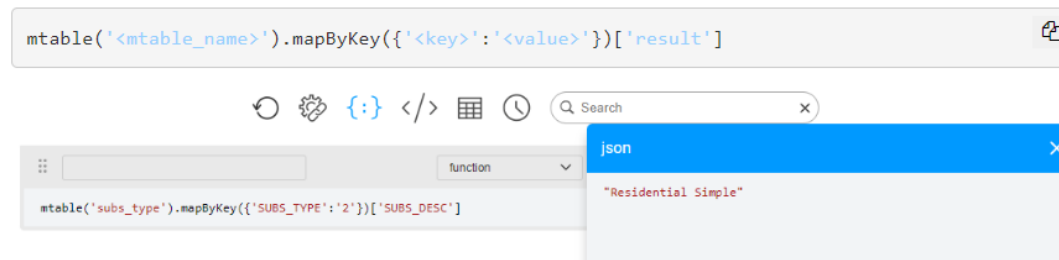
```
mtable('subs_type').mapByKey({'SUBS_TYPE':'2'})
```

```
{  
  "SUBS_TYPE": "2",  
  "SUBS_DESC": "Residential Simple"  
}
```

The below syntax returns the value of a specific MTable column

`mtable('<mtable_name>').mapByKey({'<key>':'<value>'})[col_name]`

```
mtable('<mtable_name>').mapByKey({'<key>':'<value>'})['result']
```



```
mtable('subs_type').mapByKey({'SUBS_TYPE':'2'})['SUBS_DESC']
```

```
"Residential Simple"
```

MTables

MTables Storage Settings

By default, MTables are stored in Fabric memory for fast data lookup. However, they can be stored in FabricDB under certain conditions:

1. When a joint query between MTable data and LU data is needed.
2. Due to the size of the MTable.

To change MTable storage to FabricDB, modify the `config.ini.[fabricdb].FABRICDB_MTABLE_LIMIT` parameter:

- **-1**: Keep all tables in memory (default).
- **0**: Store all tables in FabricDB.
- **>1**: Any MTable exceeding the specified row limit is stored in FabricDB; smaller tables remain in memory.



MTables

MTable in FabricDB

MTables that are stored in FabricDB, are kept in mtables.db SQLite DB.

```
fabric@dev-fabric-deployment-848b7d9f89-6qwqc:~/workspace/storage/common$ ll
total 4536
drwxrwsr-x 2 fabric fabric  4096 Oct  9 09:50 ./
drwxrwsr-x 4 fabric fabric  4096 Jun 26 08:49 ../
-rw-rw-r-- 1 fabric fabric 1093632 Aug 31 14:57 common.db
-rw-rw-r-- 1 fabric fabric  32768 Oct  9 09:50 common.db-shm
-rw-rw-r-- 1 fabric fabric     0 Oct  9 09:50 common.db-wal
-rw-r--r-- 1 fabric fabric 237568 Oct  9 09:52 mtables.db
-rw-r--r-- 1 fabric fabric  32768 Oct  9 09:51 mtables.db-shm
-rw-r--r-- 1 fabric fabric 1013552 Oct  9 09:51 mtables.db-wal
```

Indexes are created on-the-fly when querying the MTable using the lookup functionality (actor or mtable package functions).

The MTable can be queried like regular SQLite table.

