



Course 1

LUI Sync Process - Part 1

Agenda

- What happens when we execute GET?
- LU Storage
- System database
- Compression
- Vacuum
- LUI version
- Cache
- Transaction
- Sync timeout



What happens when we execute a GET command?

Get Customer.123:

1. Validate user permissions for the LUI.
2. Check for any attached MDBs in the session that cannot be released due to an ongoing transaction (within the same LU type). If found, raise an error: "Attached LU cannot be detached while in transaction."
3. Extract the SQLite from storage. If not found - start from empty db.
4. Decompress the file.
5. Decrypt the file if encryption is applied.
6. Compare the schema to the LU type definition for any necessary upgrades and implement the schema changes accordingly. If the LUI is new - create the schema.
7. Attach the SQLite file and lock it for write.
If another GET operation is underway for the same instance, on the same node, the current thread will wait MDB_ATTACH_TIMEOUT.
8. If timeout exceeds - throw an error.
9. Begin syncing the LUI by executing table populations.
 1. *Execute all populations and enrichment functions. Each population establishes connections to its source DB as required (utilizing the connection pool).*
 2. *Update k2_objects_info after each population.*
 3. *If LUT sync timeout exceed – throw exception*
 4. *Commit changes to the SQLite.*
10. Encrypt if needed.
 1. Apply compression.
9. Save the SQLite back into the storage.
10. Perform a resource cleanup.
Note: Detach occurs only upon execution of a Release command or when a new instance ID (from the same LU type) needs to be attached.

LUI Storage

The location where to storage the SQLite files (MDBs)

The screenshot displays the k2view Schema (Customer) interface. The main workspace shows a database schema diagram with four tables: **customer**, **activity**, **address**, and **cases**. Each table has a **population.flow** property with a checkmark and a count (2 for customer, 3 for activity, 3 for address, and 3 for cases). The **customer** table has fields: **customer_id**, **first_name**, and **last_name**. The **activity** table has fields: **customer_id**, **activity_id**, **activity_date**, and **activity_note**. The **address** table has fields: **customer_id**, **address_id**, **street_address_1**, and **street_address_2**. The **cases** table has fields: **activity_id**, **case_id**, **case_date**, and **case_type**. The **Schema Properties** panel on the right shows the **Customer** schema. The **Misc** tab is selected, and the **Storage** property is highlighted with a red border. The **Storage** dropdown menu is open, showing the following options: **Default**, **Cassandra**, **S3**, **Azure Blob Store**, **Google Cloud Storage**, and **None**. The **Cache Location Type** property is set to **Default**, and the **Enable Data Encryption** checkbox is unchecked.

LUI Storage

Options:

- **Default** – as defined in `MDB_DEFAULT_SCHEMA_CACHE_STORAGE_TYPE`

- **SYSTEM_DB** - [system_db]
- **AWS S3 storage** - [s3_storage]
- **Azure blob storage** - [azure_blob_storage]
- **GCS – Google Cloud Storage** - [gcs_storage]
- **NFS – shared storage.**
Add `Config.ini[fabricdb].`
`MDB_DEFAULT_STORAGE_PATH`
- **None** – LUIs are not stored

- **Cassandra** - [default_session]

- **S3** - [s3_storage]

- **Azure** blob storage - [azure_blob_storage]

- **GCS** - [gcs_storage]

- **None** – LUIs are not stored

```
[fabricdb]

FABRICDB_ENGINE=postgresql

## Time in millis to wait for a MicroDB to be released (read/write) by other threads
#MDB_ATTACH_TIMEOUT=10000

## Time in millis to wait for a fabricdb context to become available
#MDB_CONTEXT_POOL_GET_TIMEOUT_MILLIS=10000

## The number of concurrent fabricdb sessions
#MDB_CONTEXT_POOL_SIZE=200

## Cache size limit in bytes per Schema for MicroDB instances that are not currently in use
#MDB_DEFAULT_SCHEMA_CACHE_SIZE=10000000

## Defines the default storage for Micro Databases (can be changed for individual schemas from Studio)
## SYSTEM_DB/S3/AZURE_BLOB_STORE/GCS/NFS/NONE
#MDB_DEFAULT_SCHEMA_CACHE_STORAGE_TYPE=SYSTEM_DB
```


LUI Storage

Using cloud storage for LUIs is preferable over Cassandra due to several reasons:

- Frequent upserts of the LUI in Cassandra can lead to a high load due to the frequent creation of SSTables, which in turn requires constant compaction processes to run
- Big LUIs are stored in chunks due to Cassandra limitation, resulting in slower saving and fetching compared to cloud storage, which remains unaffected by blob size.

However, it's important to note that cloud storage incurs higher costs compared to Cassandra.

```
fabric>list lut storage='Y';  
|LU_NAME |STORAGE|Project Version|  
+-----+-----+-----+  
|Asset   |Default|1.0.0         |  
|Customer|Default|1.0.0         |
```

LUI Storage - Cassandra

Entity table

- When the LU storage is set to Cassandra, the LUIs are stored, as a blob, in a table named “Entity” within the k2view_[LU_name]_[cluster id if exists] keyspace.

Entity table:

- Id = iid
- Batch_id – see Big LUIs section
- Chunks_count - see Big LUIs section
- Data – blob of the LUI SQLite file
- Key_desc_id - see Big LUIs section
- Schema_hash – Schema metadata hash. Used to identify schema change
- Sync_version – the version of the LUI



Entity table best practice:

- To count the number of instances, use Cassandra COPY command.
- In case resync is needed for the entire population, do not use ‘select id from entity’ for the batch command. That will bring the IIDs by partitions and therefore the load on the cluster will not be distributed. Instead – use the ‘reverse migration’ logic

SYSTEM_DB

Operational database for Fabric (cluster info, job mechanism...)

Options:

- NoSQL distributed database, such as Cassandra DB
- Relational database, such as PostgreSQL
- SQLite - development and single-node environments.

```
[system_db]

## System db to use for Fabric internal storage (SQLITE,POSTGRESQL,CASSANDRA). Default: CASSANDRA
SYSTEM_DB_TYPE=POSTGRESQL
SYSTEM_DB_HOST=localhost
SYSTEM_DB_PASSWORD=~encL0~Fg5TJ4AjjfKso6STHjSyMwMeGAB/1JSf+8GKcGUmZk4F7tUJ
SYSTEM_DB_USER=postgres
SYSTEM_DB_PORT=5432
SYSTEM_DB_DATABASE=k2_db
#SYSTEM_DB_CONNECTION_STRING=
```


SYSTEM_DB

NoSQL distributed database

Pros:

- Scalable
- Distributed
- Built-in TTL mechanism on row level.
- If Cassandra is used as a MicroDB storage, there is no need to introduce additional DBs.
- Managed services (such as AWS Keyspaces or Astra) are supported.
- Supported by the iidFinder solution.
- Built-in mechanism for managing parallel threads during a bulk instance loading.

Cons:

- Consistency
- Not easy to operate and maintain.

Relational database

Pros:

- Consistency
- Compliance with services such as Cloud Spanner, AlloyDB.
- Easy to maintain.

Cons:

- Single point of failure
- Not supported by the iidFinder solution.

SYSTEM_DB

Transfer LUIs to a new storage

To move LUIs between storages, use the following config.ini [fabricdb] settings:

- `MDB_DEFAULT_SCHEMA_CACHE_STORAGE_TYPE=TRANSITION`
- `STORAGE_TRANSITION_FROM=CASSANDRA` (the new DB)
- `STORAGE_fTRANSITION_TO=S3` (the old DB)

LUIs transfer process:

1. On GET, Fabric first checks (i.e., reads) `STORAGE_TRANSITION_TO` for data. If Fabric cannot find data there, it then reads `STORAGE_TRANSITION_FROM` for data.
2. On save, Fabric saves data in `STORAGE_TRANSITION_TO`. If the LUI was found in `STORAGE_TRANSITION_FROM` Fabric deletes the data that was found in `STORAGE_TRANSITION_FROM`.

If `STORAGE_TRANSITION_FROM` is set to blank, the system will stop reading `STORAGE_TRANSITION_FROM` (no need for a restart).

LUI Compression

Before storing the LUI in the storage, Fabric compress the MDB file, to have a smaller space and faster save/extract.

The compression can be changed in config.ini.[fabricdb].MDB_DEFAULT_STORAGE_COMPRESSION

```
## The storage compression to use: LZ4/GZIP/NONE  
#MDB_DEFAULT_STORAGE_COMPRESSION=LZ4
```

Options:

- LZ4 (default)
- GZIP
- NONE (no compression).



Best practice:

- GZIP – result is 25% of the original file.
- LZ4 have better compression but a bit is slower.

LUI Vacuum

As an SQLite database, the below scenarios may occur in the LUIs:

- When content is deleted, it is not usually erased but rather the space used to hold the content is marked as available for reuse.

When a large amount of data is deleted the database file might be larger than strictly necessary.

- Frequent inserts, updates, and deletes can cause the database file to become fragmented.

Running VACUUM reduces the size of the database file:

- Reclaims the “free” space
- Fix the fragmentations



Best practice:

Usually there is no need to change the default vacuum behavior.

In case of an issue, and after consulting with R&D, you can force reclaim free space before storing the mdb using config.ini.[fabricdb].

MDB_VACUUM_THRESHOLD_KB: -1 is off, 0 always

LUI Version

Each time the LUI is saved back to the storage, it is assigned with a new version number.

The version number is constructed from the timestamp when the file is saved back

- When the storage is Cassandra, the version is kept in the Entity table.
- When it is stored in cloud, it is kept as property of the file (tag)

The version is used by Fabric in the below cases:

1. Validate LUI override
2. Validate cache is up to date

```
fabric>get Customer.1;
|luName  |iid|version          |action|notes|
+-----+---+-----+-----+-----+
|Customer|1  |1713091350631|ADD   |      |

(1 row)
fabric>select version("Customer");
|version("Customer")|
+-----+
|1713091350631      |
```

LUI Cache

To optimize the LUI retrieval process, Fabric uses a cache mechanism, which enables a faster loading of an instance into the memory.

The cache location is defined in the Cache Location property of the LUI Schema.

The screenshot displays the k2view interface for the 'Schema (Customer)'.

Schema (Customer) x
Implementation > Logical Units > Customer > Schema (Customer)

The main workspace shows a diagram with several entities and their relationships:

- customer** (population.flow 2):
 - customer_id
 - st_name
 - st_name
- activity** (population.flow 3):
 - customer_id
 - activity_id
 - activity_date
 - activity note
- address** (population.flow 3):
 - customer_id
 - address_id
 - street_address_1
 - street address 2
- cases** (population.flow):
 - activity_id
 - case_id
 - case_date
 - case type

The **Schema Properties** panel is open, showing the **Population Order View** for the **Customer** schema. The **Cache Location Type** property is highlighted with a red box, and a dropdown menu is open, showing the options **Default** and **Storage**.

Schema Properties Population Order View

Customer

Misc

Storage

Default

Cache Location Type

Default

Storage

☐ Enable Data Encryption

LUI Cache

Cache location property | Available options:

1. Default

- As configured in config.ini.[fabricdb].
MDB_DEFAULT_CACHE_PATH

```
## This is where fabricdb stores it's MDB cache.  
MDB_DEFAULT_CACHE_PATH=${FABRIC_HOME}/storage/fdb_cache
```

- Default is /dev/shm/fdb_cache.
If this path does not exist,
\${FABRIC_HOME}/storage/fdb_cache/ will be used.
- Under the default path, a folder is created per each LUT.
- The /dev/shm directory is a special directory on Linux systems that is used for storing temporary files.
The files in /dev/shm are stored in memory, rather than on disk, which makes them much faster to access (disk is mounted directly to the Memory)

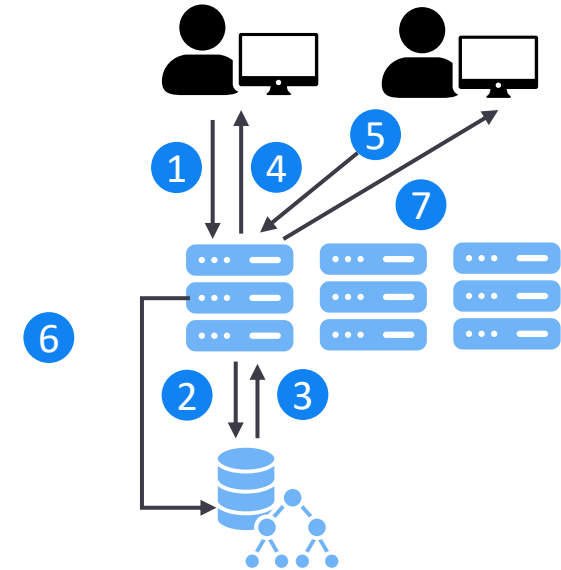
2. Storage

- MDB files will be stored in
\${FABRIC_STORAGE}/storage/fdb_cache/ .
If this location does not exist, store the cache in \${FABRIC_HOME}/storage/fdb_cache/.
- Note: changing the default path in config.ini will result in ignoring this setting for all the LUTs.

LUI Cache

Fabric usage of the LUI cache

1. User 1 execute GET Customer.215 on node 1
2. Fabric is fetching the LUI (MDB) from Storage
3. Fabric save the MDB file on node 1 cache directory
4. User 1 can query the LUI
5. User 2 execute GET Customer.215 on node 1
6. Prior to utilizing the cached file, Fabric validates whether it is the most recent version by querying the storage for MDB with the same IID but with version that is greater than cached one. If MDB is found – the cached file will be replaced with the newer version.
7. If up-to-date, user 2 can query the file



LUI cache best practice:

1. Always prioritize using the default path (/dev/shm) to optimize performance.
2. For very large LUIs that exceed the cache size, consider configuring the cache to utilize another disk space.

Cache storage size

- The catch storage size is restricted and is set per LUT in config.ini.[fabricdb].
MDB_DEFAULT_SCHEMA_CACHE_SIZE.
- Once the cache storage reaches the specified size:
 - Inactive files are removed from the cache by LRU (Least Recently Used) order.
 - If the size of the LUI file exceeds the remaining space it will be retained in the cache, potentially exceeding the maximum size defined, but it will not be stored in the directory once detached.
 - If the file size exceeds the total memory, Fabric will crash.



Note:

- *The cache is stateless - cached files are deleted upon Fabric restart.*
- *The cache is utilized best when the same LUI is accessed multiple times within a short period. Otherwise, it is possible for the file to be removed from the cache due to other files utilizing it.*



LUI transaction

The Sync process is managed as a single transaction that starts at the beginning of the Sync process and finishes at its end.

- If the Sync is completed successfully, the data is committed to the Fabric database.
- If an error occurs at any point during the Sync process, the transaction is rolled back.

The sync transaction can be managed from outside the sync process, when opening it by the calling session:

```
Db ci = db("fabric");  
ci.beginTransaction();  
ci.execute("get Customer.'" + IID + "'");  
String SQL = "INSERT into CONTRACT_COPY  
(CUSTOMER_ID,CONTRACT_ID,CONTRACT_REF_ID) values (?, ?, ?)";  
Object[] params = new Object[]{IID, contrID, contrRefID};  
ci.execute(SQL, params);  
ci.execute("commit");
```

LUI transaction

LUI transaction

In case the source system cannot be accessed, you may prefer to roll back the Sync without getting an error (exception).

In that case, use the `ignore_source_exception` command set to 'true' on the session level.

```
fabric>set ignore_source_exception = true;  
(1 row affected)  
fabric>get Customer.215;
```



Note:

If the instance is not yet in Fabric, the GET command will throw an exception.

LUI sync timeout

Timeout for the Sync process

The screenshot displays the k2view application interface. At the top, the breadcrumb navigation shows 'Implementation > Logical Units > Customer > Schema (Customer)'. Below this is a toolbar with various icons for navigation and editing. The main workspace shows a schema diagram with four entities: 'customer', 'activity', 'address', and 'cases'. Each entity has a 'population.flow' property with a checkmark and a count (2 for customer, 3 for activity, 3 for address, and 1 for cases). Lines connect the 'customer_id' attribute of the 'customer' entity to the 'customer_id' attributes of the 'activity', 'address', and 'cases' entities. On the right side, the 'Schema Properties' panel is open, showing the 'Population Order View' for the 'Customer' entity. The 'Sync' section is expanded, and the 'Sync Timeout' field is highlighted with a red border, showing a value of '0'. Below it, the 'Sync Method' is set to 'None', and there is a checkbox for 'Delete Instance If Not Exists' which is currently unchecked.

Schema (Customer) ×

Implementation > Logical Units > Customer > Schema (Customer)

Schema Properties Population Order View

Customer

Sync

Sync Timeout

0

Sync Method

None

☐ Delete Instance If Not Exists

LUI sync timeout

By default, LUI sync time is not limited by time. Nevertheless, it is recommended to limit the sync time to avoid bottlenecks and stuck instances.

If a timeout is set and the sync exceeds the predefined timeout, Fabric rolls back the changes and throws the following exception: Timeout occurred.

A sync timeout can be defined either on the LUT level or per session:

- LUT Schema level – set the timeout for all the instances of the LUT.
- Session level – override the LUT setting on a session level, using **“set timeout”** command.

```
fabric>set sync_timeout = 10000;  
(1 row affected)  
fabric>get Customer.215;
```

LUI sync timeout



Design best practice:

- Always set the timeout on the LUT schema level.
- For instances that exceed the timeout, it is recommended to have a retry process (log the failure and create a retry process). The retry process can dynamically adjust the sync timeout at the session level using the `set sync_timeout` command.

Note: when creating a different process, consider the limitations described in the Parallel GETs section.

Course assignment

1. Add functionality inside the Customer LU, to update the zip code in address table.
2. Create new BW flow that execute GET on the Customer LU. After the GET is completed:
 - Check if the zip code valid in the state provided in the address record, using Open AI library
 - If the zip code is not matching – rollback the changes in the address table

