k2view

Alpinist

Non-source Schema tables & MDB JMX Stats

WELCOME

Agenda

- Business Tables
- LU Product Tables
- Additional Table Properties
  - Full-Text Search (FTS)
  - Reference List
  - Columns Collation
- MDB JMX Stats

Business tables

Business tables are independent LU tables that do not have a direct or indirect connection to the LU Root table.
These tables enable users to compute, transform, and store new data within the LU.

Use Cases:

- **Pre-prepare API Responses**: Store an API response in advance when data changes occur in the LUI, ensuring it is ready when needed.
- **Execute Complex Processes**: Run complex/long processes ahead of time, making results immediately available when required.

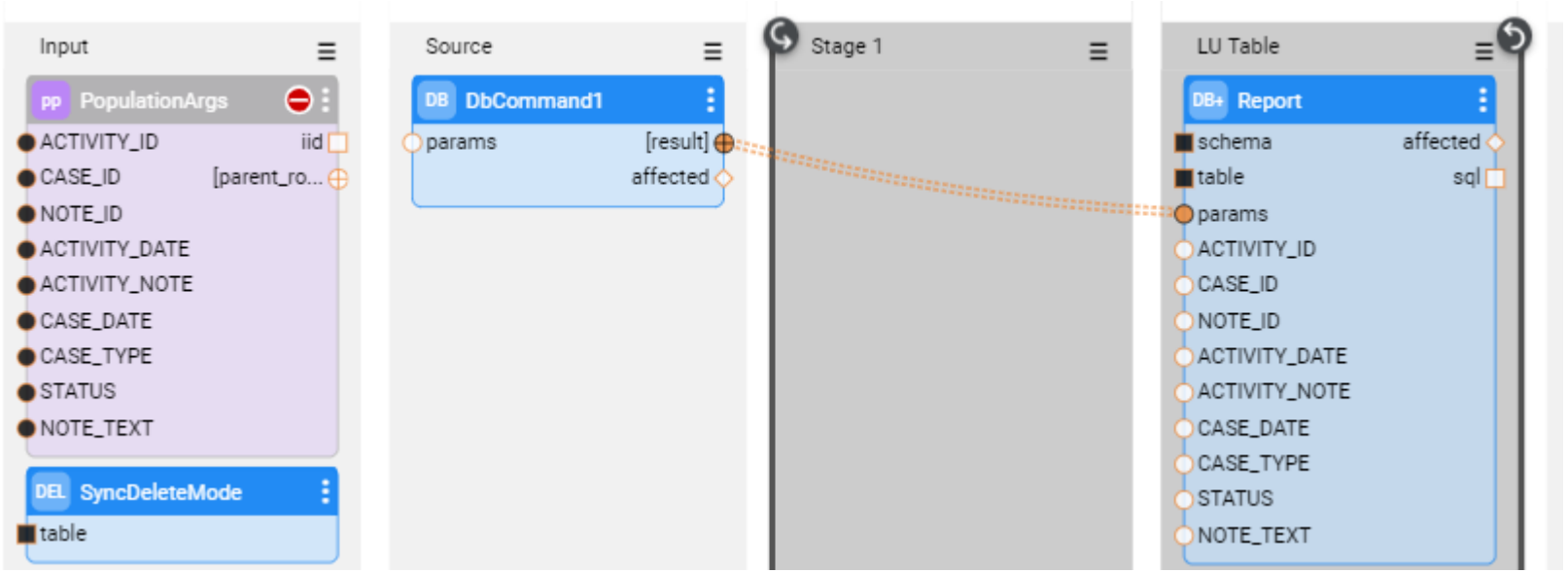
Best practice:

To optimizes performance and efficiency:

- Business table’s population (if exists) should run only when the source tables are updated.
- Use a decision function to determine whether the population should be executed or not.

Business tables

Note:
When creating a population for a business table without a parent, the SourceDBQuery Actor should be replaced with the DbCommand Actor, and it should not be linked to the PopulationArgs Actor.



LU Built-in Tables

_k2_object_info

Column	Description
table_name	The LU table name
object_name	Name of population/enrichment function
type	6=population, 10=enrichment function
verified_time	Last time of verification as to whether the object should be synced or not (according to the sync policy). Will not be updated if the population was not executed
Start_sync_time	Last run start time
End_sync_time	Last run end time
Start_write_time	Last run start write time (to the Sqlite file)
last_write_time	Last run end write time (to the Sqlite file)
number_of_records	Number of processed records
time_to_populate_in_sec	Total time in seconds to run the object
next_time_to_populate_object	Next time the object should be synced. Calculated based on the sync policy, starting from the beginning of the current object sync.

LU Built-in Tables

_k2_object_info

Column	Description
version	Version of the object's last deployed schema
sync_error	The sync_error is populated in case of a sync raised an error that didn't cause a rollback. For example, if you start the transaction and then perform a GET command which fails - the sync_error will be populated until the transaction is closed.

Note

1. Use this table to investigate LU functionality and sync performance
2. Populations not using **sourceDbQuery**, like business tables, won't update the number_of_records field. Use the **PopulationCount** actor to enable this functionality

LU Built-in Tables

_k2_main_info

Column	Description
lu_name	Each record extracted from the source is inserted into the LU table using the INSERT operation.
version	Version of the last deploy which impacted the LU Schema.
Instance_id	LU IID
Version_timestamp	The timestamp of the last LU schema deployment. This field is retained for backward compatibility but is no longer in use.

Note:

Always use the LUT name when selecting from the built-in tables

```
fabric>get Customer.215;
|luName  |iid|version      |action|notes|
+--------+---+-------------+------+-----+
|Customer|215|1723118036705|UPDATE|     |

(1 row)
fabric>
fabric>select * from Customer._k2_main_info;
|lu_name |version      |instance_id|version_timestamp|
+--------+-------------+-----------+-----------------+
|Customer|1723115249072|215        |1723114555650000 |

(1 row)
```

LU Built-in Tables

_k2_transactions_info

Column	Description
id	Transaction id
ts	timestamp

Note

- Doesn’t exist in Fabric 7.2 and up
- Used to hold the fabric transaction id for the CDC
- This table does not have a cleanup mechanism. It's important to review it in projects using CDC and implement logic for deletion as needed.

Note

1. **k2_read_pos**
   - Not in use. Will be removed in Fabric 8.1
2. **K2_delta_error**
   - Used by iidfinder partitioned delta mode.
   - Holds information on errors, including when each error occurred.

Table Properties

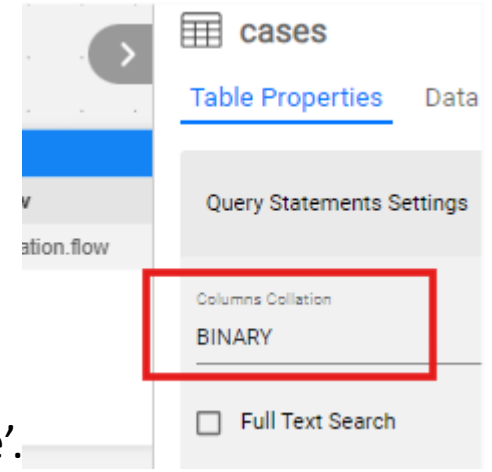
Columns Collation

The COLLATE operator in SQLite defines how string values are compared when using a WHERE clause in queries.

SQLite provides several collation functions to tailor string comparisons:

- **BINARY** – Fabric's default. Providing case-sensitive comparisons based on ASCII values.
- **NOCASE** - enables case-insensitive comparison.

  For example:
  Select TYPE from tblExample where NAME = 'value'
  This query will return records when the NAME field is set to either 'VALUE' or 'value or 'Value'.



- **RTRIM** removes trailing spaces before performing the comparison..

  For example:
  Select TYPE from tblExample where the NAME = 'value'
  This query will return records that match both 'value' and 'value '.

Note:

When modifying this property, if LUIs already exist in Fabric, you must first delete them, deploy the changes, and then resync.
Alternatively, in a development environment, you can drop the LUT before deploying.

Table Properties

Reference List

The References List property displays all Reference tables defined in the project.

A Reference table should be checked only if:

- It is needed as a lookup function in one of the populations.
- We want to activate the Reference table sync (if needed, according to the sync policy) once the LUI is synced.

Note:

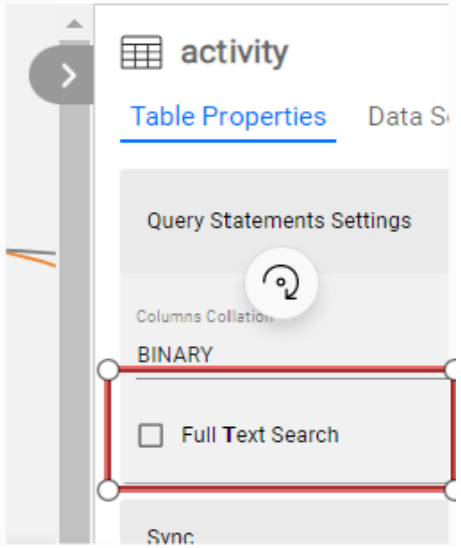
- A Reference table can be accessed from code (e.g., function) also when the table is not checked.
- It is recommended to limit the number of checked Reference tables to avoid a massive sync of the Reference tables when synchronizing an LU instance

Best practice:

To select from Reference use ludb.fetch or fabric.fetch, instead of FabricDB.fetch (fabric local interface defined in the implementation) as this drastically impacts the performance (fabriclocal/fabricremote is over TCP).

Table Properties

Full-Text Search (FTS)



Fabric utilizes the SQLite FTS5 extension module to enable full table search capabilities within an LU table.

The most common use case for FTS tables is wildcard or prefix searches, where queries using the LIKE operator are insufficient (e.g., for fuzzy search).

When the full-text search property is set to TRUE, a virtual table is created along with five additional tables. These tables store all necessary data, tokens, and indexes required to perform a full-text search.

Considerations for Using FTS:

- Full-text search is most advantageous when dealing with large volumes of text data that need frequent searching.
- Each INSERT, UPDATE, or DELETE operation triggers updates to the corresponding tables, which may impact performance, especially with large tables. Ensure that insert operations are not intensive and remain within the SLA.
- Full-text search can be resource-intensive, potentially affecting database performance. Indexing large amounts of text data can increase the database size, so it's recommended to limit the number of columns included to only those necessary for the search functionality.

More about the FTS MATCH command: https://www.sqlite.org/fts5.html

MDB JMX Stats

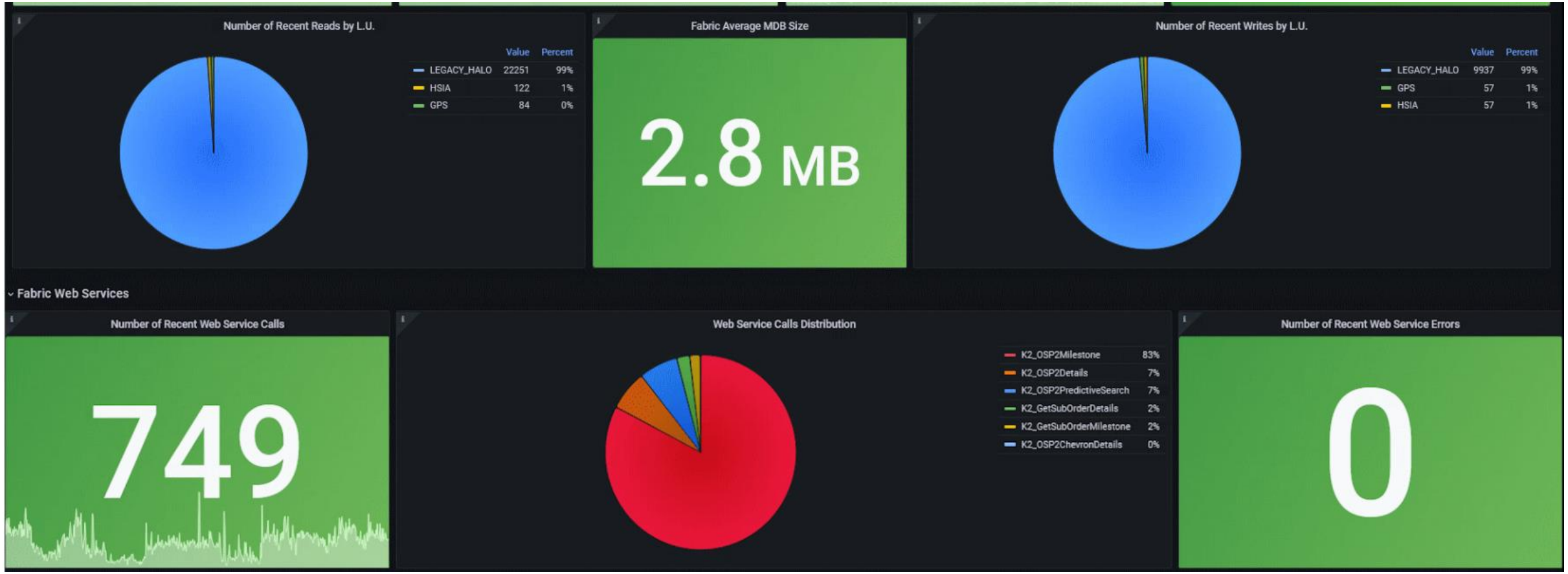
Fabric arrives tightly pre-integrated with JMX (Java Management eXtensions) - technology to enable comprehensive and low-resolution monitoring and management of applications.

Note:

1. JMX stats are per Fabric node (and not per cluster)
2. JMX stats are restarted once Fabric node is restarted

JMX Stats in Grafana

JMX Stats in Grafana



MDB JMX Stats

GET

1. **getDuration** – total duration (and count) of GET performed

MDB Save

1. **mdbSaveErrors** - duration and count of mdb saved to storage, resulting in an exception, per schema.
2. **mdbSaveDuration** - duration and count of mdb saved to storage, per schema.
3. **mdbSaveBytes** - Bytes (uncompressed) and count of mdb saved to storage, per schema.

MDB Fetch

1. **MdbFetchDuration** - Duration and count of mdb read from storage, per schema.
2. **mdbFetchNoInstance** Duration and count of mdb fetch from storage, where no instance was available, per schema.
3. **mdbFetchErrors** Duration and count of mdb fetch from storage, resulting in exception, per schema.
4. **mdbFetchBytes** Bytes (uncompressed) and count of mdb read from storage, per schema.

MDB Attach

1. **mdbAttachError** Duration and count of mdb attach that resulted in an error, per schema.
2. **mdbAttachDuration** Duration and count of successful mdb attach, per schema.

MDB JMX Stats

MDB Cache

1. **mdbCacheCount** Count of LUI micro databases cached (not in use), per schema.
2. **mdbFetchNoNewVersion** Duration and count of mdb fetch from storage, where the cache was up to date, per schema (fetched from cache).
3. **mdbCacheBytes** Bytes of LUI micro databases cached (not in use), per schema.

Statements

1. **mdbActivePreparedStatements** The number of active prepared statements, including the ones in the prepared statement cache. (MDB_PREPARED_STATEMENT_CACHE_LIMIT pool).
2. **mdbActiveStatements** The number of active statements (non-prepared).
3. **mdbActiveResultSets** The number of active result sets.

Fabric Pool

1. **mdbSessionFromPoolDuration** Time spent successfully waiting for a session from the mdb session pool (MDB_CONTEXT_POOL_SIZE pool).

Lock

1. **mdbWriteLockDuration** Time spent in write locking the mdb.
   Time spent on waiting to lock mdb file for write (until previous lock is released)

Vacuum

1. **mdbVacuumReclaim** Number of bytes reclaimed by the vacuum operations.
2. **mdbVacuumDuration** Duration of vacuum done on mdb above thresholds.
3. **mdbVacuumErrors** Number of errors in mdb Vacuum.