**Non-replicated Objects**

For now, this concept will apply to Animals, Devices, Personas.

The concept (and all the logic derived from it) stems from the replication management by Marmot, with its eventual consistency.

Due to eventual consistency, there is no guarantee that a certain object in any given node will have its original record replicated to all nodes so that subsequent operations (including Baja) will be executed using the original record for the object.

*In short, any object may come into existence in a given node, have operations/activities performed on it and eventually be removed from the system (Baja) while never getting its original creation record replicated from the node where it was created.*

The logic for non-replicated objects comes to address this for all objects that require and support replication (Note: **objects only**. Activities don’t go through this).

*Unreplicated objects (07Jul24). Initial notes*

*Unreplicated animals (or objects in general) are objects that are read into the node (scanning its identifiers), but the object is not yet defined in the local database (object was created in another node and is not yet replicated to this node).*

*Properties.*

* *Those objects give a count of 1: They physically exist (object’s been physically scanned) but the local node doesn’t have a record of them.*
* *All operations and activities on the object are deemed as valid while they remain unreplicated.*
* *When the original object is found via replication, the replication routines must run consistency checks and cleanups on all the records associated to the object.*
* *After consistency checks are passed, the object is re-classified to a new mode (regular, substitute, etc), based on present condition.*

With this, Tag and Device should also implement the unreplicated attribute. -> See into this.

**1st Attempt to logic**:

* The logic will be handled by the getObject() methods of each object class.
* A ***Update\_Checksum*** column will be implemented in every object table (Animales, Dispositivos, Personas for now).

That checksum will be written on and read from the Earliest Duplicate Record (EDR) for each object. The ERD can also be considered a temporary original (By definition, the Original Record for an object is the one with the earliest date of all records referring to the same object).

* The checksum will be a hex number resulting from combining all the values in the DataFrame that results from pulling all the duplicate records for an object existing in the node. The code to create the checksum is as follows.

chksum = hashlib.sha256(frame.to\_json().encode()).hexdigest()

* When getObject() reads the records for an object ID, creates a dataframe with all the records read and generates a checksum value from the data read.
* If the checksum value matches the checksum stored in the EDR -> there have been no changes in the involved record, all the data available at the time is stored in the EDR.
* If the check sum differs from the value in the EDR, new data has been added to 1 or more of the duplicate records. 🡺 the logic picks the 1st non-NaN value from the data frame (for each column of the df) and stores the value in the ERD row.

Updates the check sum, writes that check sum value in the EDR field and re-writes the EDR back to database

* Finally, getObject() creates an object with the values found in EDR and returns the object.
* Activities must define the mandatory parameters they require from the Objects Table (tblAnimales, tblCaravanas, etc) in order to run its code. For example:
  + AgeActivity requires fldDOB as mandatory.
  + Most of Sanitation Activities will require Category (from table Categoria)
  + CategoryActivity, CastrationActivity, ParturitionActivity will require Category (same as Sanitation) -> This is pulled from tblCategorias when the u,id of the EDR is known. Doesn’t affect or is involved in any way with the check sum done with the Objects table records.
  + LocalizationActivity, InventoryActivity will probably require no mandatory data from Objects table.
  + And so on…
* When an Activity detects a missing mandatory parameter must make a UI call to request parameter entry by the user. In this way, the Objects table on gets populated, on each node, by the subsequent entries on 1 or more duplicate records on a on-demand basis: field values get entered as they are needed, or filled by replication as replication runs its course across all nodes.

**1st Attempt to.**

sdfsdf