Mart Database for Bahmni Analytics

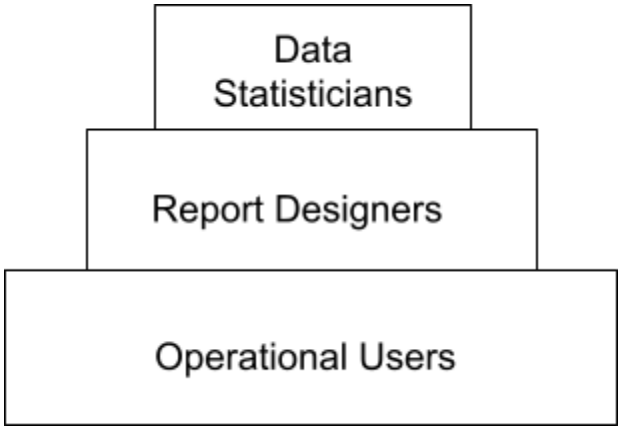
Problem Statement

The database used in Bahmni/OpenMRS has an EAV data model, and hierarchical data (For Ex: Obs table). This makes it difficult to plugin the database to any analytical tools (Tableau, Strata etc) as unstructured hierarchical data is limited and multiplies existing complexity.
In order for implementation engineers to extract data from the existing production databases is said to be very difficult without totally understanding the openmrs data model.
For an implementation to generate indicators or reports, lot of development work has to be done by writing custom reports to generate the data from which the indicators can be extracted from.

Users

1. Operational Users (Operational Teams, Executives)

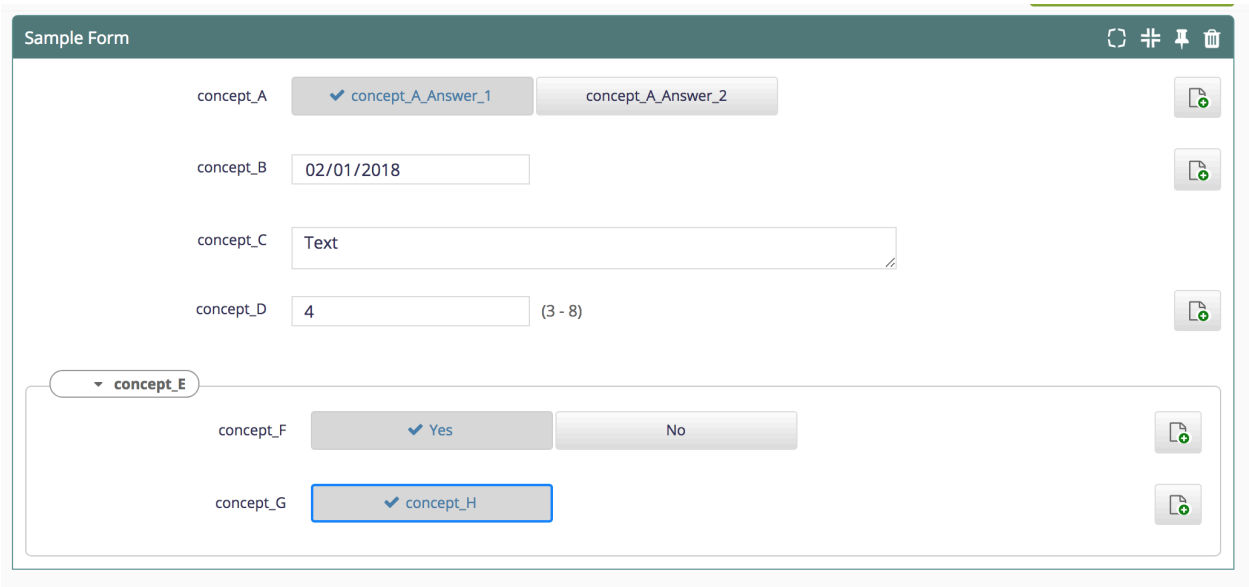
2. Report Designers(Creation of Dashboards & Reports)
3. Data Statisticians (Analysis of Unstructured Data, Statistical Analysis)
Size of User base decreases as complexity of data increases



Solution

One of the solution thought through is to flatten the hierarchical database. This flattened database can be connected to analytical tools (Tableau, Stata etc) for further analysis of data. The generation of the analytical database will be common across all the implementations. We depend on the model of OpenMRS data model to make it as generic as possible. Most of the data used by implementations reside in the obs table which is hierarchical and is dependent on concept and concept_name tables. The concept hierarchy and the dependency on concept tables in the obs data can be shown as below:

If patient with patient_id=100 has the following form filled as below



The Obs table will look like (Omitted the irrelevant data):

Obs_id	Person_id	Concept_id	encounter_id	obs_group_id	value_coded	value_datetime	value_numeric	value_Text
1	100	3682 (concept id of "Sample Form")	49823					
2	100	3686 (concept id of "concept_C")	49823	1				Text
3	100	3683 (concept id of "concept_A")	49823	1	3684 (concept id of "concept_A_Answer_1")			
4	100	3688 (concept id of "concept_E")	49823	1				
5	100	3689 (concept id of "concept_F")	49823	4	1			
6	100	3690 (concept id of "concept_G")	49823	4	3691 (concept id of "concept_H")			
7	100	3687 (concept id of "concept_D")	49823	1			4	
8	100	3685 (concept id of "concept_B")	49823	1		2/1/18 0:00		

As can be seen, all the observations are stored in hierarchical way (Obs_id = 1, is the parent for Obs_ids 2, 3, 4, 7, 8). These observations will be flattened into a single table(per form) by joining Obs table to get the hierarchical data, concept tables (concept, concept_name e.t.c) to get column names for the resultant table in mart database. For ease of understanding, obs data related to only one patient is shown. Obs data related to all the patients captured in a form will be stored in a separate table in mart database. For the above data, the corresponding tables in analytical database will look like:

Table: sample_form

id_sample_form	Person_id	encounter_id	concept_A	concept_B	concept_C	concept_D	concept_F	concept_G
1	100	49823	concept_A_Answer_1	2/1/18 0:00	Text	4	Yes	concept_H

The Bahmni/OpenMRS data model for a form is actually too complex to be flattened into a single table: “add more” sections, and multi-selects don’t fit in a single row. If certain concepts or concept sets has to be extracted as separate tables, a configurability option will be provided to identify those concepts and concept sets.
In the context of Bahmni, all the multi-select and add more concepts/concept sets will be extracted as separate tables automatically. Batch job gets this information from the existing json files of bahmni_config (/var/www/bahmni_config/openmrs/apps/clinical/app.json, /var/www/implementation_config/openmrs/apps/clinical/app.json). In addition to these, if any other concept names are configured in “separate tables” configuration, they will also be extracted as separate tables in mart database.
For non-Bahmni implementations, the application will only consider “separate tables” configuration.
These concept/concepts can be added in that configurable option. If fully specified name of “concept_E” is configured to be extracted as a separate table, the tables generated will be as of below:

Table: sample_form

id_sample_form	Person_id	encounter_id	concept_A	concept_B	concept_C	concept_D
1	100	49823	concept_A_Answer_1	2/1/18 0:00	Text	4

Table: concept_E

id_concept_E	id_sample_form	patient_id	encounter_id	concept_F	concept_G
4	1	100	49823	Yes	concept_H

Approaches

Approach-1

Use Atom Feed data and replay it to write to analytical database into flattened tables.
Pros:

1. Database will be real time, but, this is not a requirement as analytical database is mostly needed once in a month to generate reports or indicators

Cons:

1. If one entry is changed several times, the entry in analytical database should also be altered several times
2. Atom feed is not available for modules like Bed Management, Appointment Scheduling
3. Need a separate solution to build from scratch (if you put the mart on top of an existing install)

Approach-2

Use a Spring batch application to flatten the existing openmrs database. The reader in the batch job will read from current openmrs database (MySQL). Processor in the batch job will flatten the data. Writer in the batch job will write the flattened data to analytical database.

Pros:

1. Complete database in an implementation which includes modules like Bed Management, Appointment Scheduling can be flattened
2. Can schedule the flattening (weekly, monthly etc) or can do on demand flattening also
3. Can create extendable jobs

Cons:

1. Database will not be real time, but, this is not a requirement as analytical database is mostly needed once in a month to generate reports or indicators

Based on the above analysis, Approach-2 is being considered for the flattening of database.

Technical Details

Spring Batch

Spring batch RPM will be used to generate the flattened database. Various jobs will be created to flatten different types of data. The input source for the flattening job will be Openmrs database (MySql) and the output source to store the flattened data will be called as Mart database. Postgres is chosen as the Mart database as it is feature rich and efficient compared to other relational databases like Mysql. Postgres provides inherently pivoting data like crostab, this might be useful feature if we decide to move pivoting operation from spring batch to database in future. NoSql databases were also considered during the analysis but, doing analytics based on NoSQL databases is comparatively complex and implementation heavy compared to Relational/SQL databases. There is extensive support for Relational/SQL databases in all popular analytics/BI tools like Tableau, Superset etc. The simplicity and efficiency of solution based on SQL database was primary reason for not using NoSQL databases. Also, In OpenMRS/Bahmni ecosystem, usually we dont have implementations set up for for multi node / clustered systems which many NoSQL databases usually support by design.

For the solution to be generic a provision to configure jobs is provided. Every job in Batch, has a reader, writer and processor. All these elements are also configurable based on the type of the jobs. Every job will be associated with a ‘type’ based on the type of job it performs.

Types of Jobs

1. “Obs”:

Input data source: Openmrs datasource
Output data source: Mart datasource

To flatten the observations a job is created. This job takes all the obs and flattens it to every form as a separate table as described in section Solution. Every form will be created as a separate table in Mart database. Any add more or multiselect concepts will also be created as separate tables. Any modifications to the forms will be considered only when the batch job runs next time after the change. For eg, if a new concept is added to a form, the new column corresponding to the concept will be added to the form table only after the batch job is run after adding the concept. The obs can be flattened in various ways based on the requirements of implementation. Some of the ways are listed below:

a. Flatten it based on the forms. Every form becomes a separate table. If configuration is provided with concept names to create separate tables, those will be extracted as separate tables. The output tables will contain fully specified names in place of concept ids.
b. Flatten it based on the forms. Every form becomes a separate table. If configuration is provided with concept names to create separate tables, those will be extracted as separate tables. The output tables will contain implementation specific codes (based on concept_reference_term_map_view table).
c. Flatten it based on the concept sets. Every concept set becomes a separate table. If configuration is provided with concept names to create separate tables, those will be extracted as separate tables. The output tables can contain fully specified names or implementation specific codes.

All the above extractions should be achievable and also to make it generic to support any other possibilities, the processor of this job will be pluggable. Thus, implementers can write their own processor and plug it in as processor for this job. The processor class will be configurable.

2. “Generic”:

Input datasource: Openmrs datasource
Output datasource: Mart datasource

A generic type job will read a “readersql” from configuration, executes that sql in openmrs database and write the corresponding data as table in mart database. This gives flexibility to add any required data to analytical database. In case new modules are added in future, the data from those modules can be flattened and added to mart database by adding a new job in the configuration with the corresponding readersql.

3. “View”:
Input datasource: Mart datasource
Even though all the data is available in the Mart database, in order to generate reports/indicators, business views have to be created based on the implementation requirement. The views can be configured in the “readersql”.
For example, if an implementation requires an indicator to know number of patients under the age of 18 who are enrolled in HIV programme, this information can’t be obtained directly from a single table in the mart database. For this we can create a view with all the patient attributes (that includes age) and patient programme information. Using this view, the indicator can be generated.

Configuration File

The configuration file will be json format. The path of the json file can be provided as an input while running the batch jar/rpm. We are planning to place the json in “bahmni_config” folder.

Configuration Name	Description	Supported in job type
separate_tables	List of concept names that need to be extracted as separate tables	obs
jobs	List of jobs to be executed	
readersql	Reader sql to be executed on input database	Generic, view
name	Name of the job	Obs, generic, view
type	Type of the job which is one of ‘obs’, ‘generic’, ‘view’	Obs, generic, view
transformerClass	Processor class to be run	obs
tableName	Output table name	Generic
viewName	Output view name	view
chunkSizeToRead	Number of chunks to be read and write while performing the job	Obs, generic, view

Sample config file:

```
{

  "jobs" :
  [
          {
          "name" : "job1",
          "type" : "<type of job>",
          "separate_tables" : [<List of concept names>],
          "readersql": "<reader sql>",
          "transformerClass" : "<pluggable processor>",
          "tableName": "<output table name>",
          "viewName": "<output view name>",
          "chunkSizeToRead": "<number of chunks to read>"
          },
          {
          "name" : "job2",
          "type" : "<type of job>",
          "readersql": "<reader sql>",
          "transformerClass" : "<pluggable processor>",
          "tableName": "<output table name>",
          "chunkSizeToRead": "<number of chunks to read>"
          }
  ]
}
```

Integration with Spring Cloud

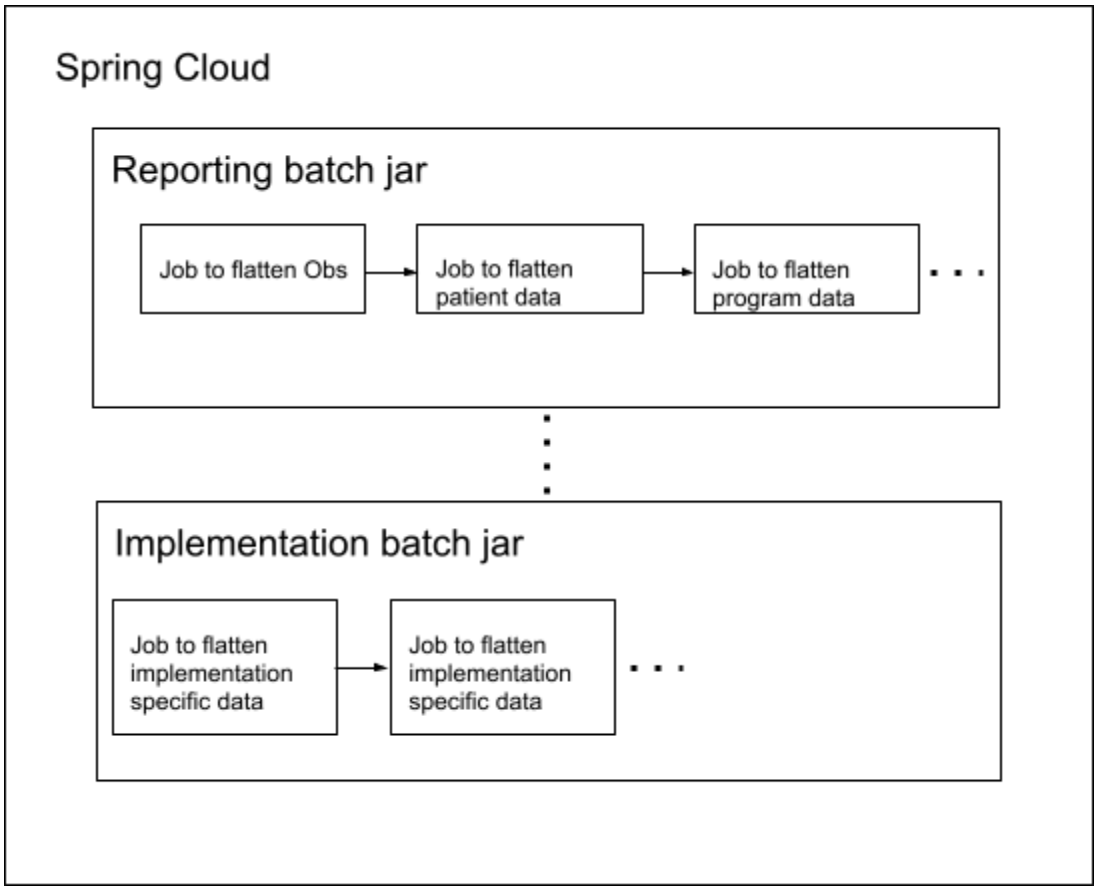
The reporting application bahmni-mart is integrated with Spring Cloud Data Flow (https://cloud.spring.io/spring-cloud-dataflow/). Bahmni mart is a Spring Boot application developed using the Spring Cloud Task microservice framework. (https://cloud.spring.io/spring-cloud-task/), so that data integration pipelines can be built with bahmni-mart in Spring Cloud Data Flow. This provides a very good UI to the user to run on demand batch jobs, selective jobs from various batch jars.
This will be hosting the bahmni-mart batch application. Implementation will be installing Spring cloud through which batch application is executed.

Extensibility

Jobs are made configurable in such a way that implementations can flatten the data as desired. In case implementations would like to write their own batch jobs for any of the modules

developed in future, implementations can generate their batch jar and can run it after running the Reporting batch jar. In this way the Reporting batch jar can be reused and can be extended for any future use.

Architecture



Future Scope

Incremental Flattening

Currently the batch application flattens the entire database every time it runs. This can lead to performance issues once the database grows to a considerable size in an implementation. To avoid this, we need to look into how incremental flattening can be done.
One of the possible approaches could be using atom feed to get the information of patients for whom any changes were done and then updating the mart database only for those patients.

Business Views

The data flattened may not be used directly to generate all the reports or indicators in any implementation. Common business views across the implementations can be thought through so that these views can be generated by the application. Few pointers could be:

- ➔ Identify some basic concepts common across the forms like weight/BP and have them in all the form related tables
- ➔ Have patient related information like patient state and program in all the form related tables
- ➔ Have a single table for all the form related data

Storing of data in JSON

Postgres allows data to be stored in JSON format. The application can make use of it to store all the relevant data in JSON format in a single column.