# EIDSS7 Database design approach

It’s critical that we take the right approach on the backend design to ensure delivery of EIDSS7 on time with no compromise on data quality. Data is the key in any project and should be preserved and stored with integrity. As we are giving the freedom to redesign EIDSS, our initial though was to do a complete redesign of the database. This would be the best approach to do it right if we have the time and support to do it. However, after looking at the new requirements (200+) for V7, it became imperative that we relook at the initial though on complete redesign.

This document outlines the options taken into account and the decision taken on the approach based on the factors that need to be taken into account for the EIDSS project.

**Option 1**

* + Redesign the database objects
  + Redesign the data access layer

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Flexibility to defining new objects and relationships to confirm data integrity and quality | Takes a lot of time to deliver the requirements |
| Flexibility to design the data access layer to meet the need of the new redesigned front end. | Data Migration could be at risk due to inaccessibility of real country data. |
| Define database objects to follow DB standards to support better sustainability | Extensive work on data migration which can lead to failure if not tested thoroughly using data to cover all use cases for all the modules. |
| Define new data dictionary to support the new design |  |
| Opportunity to address the current DB design that includes system generated and user defined IDs that limit the easy access of data |  |
| Address the multiple layer issue (that exist in V6) to access data from DB. The multiple layer makes debugging a challenge |  |

**Option 2**

* + Update EIDSS6.1 database objects to meet V7 requirements
  + Redesign data access layer to meet V6 and V7 requirements

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Use the existing database objects. | Continue to use the ID schemes in V6 |
| Spend time on the data access layer and develop a sturdy data access layer to support the requirement | Data integrity might be compromised. Need to continue to carry any existing data issue from V6 to V7 |
| Clean the data access layer to include only objects needed to support the need. | Database objects do follow best practice standards |
| Address performance issue from the get-go |  |
| Data Migration would be easier compared to redesign |  |

**Option 3**

* + Update EIDSS6.1 database objects to meet the V7 requirements and add Optimized and Simplified data structures
  + Optimize EIDSS6.1 data access layer to address performance issue and V7 requirements

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Use the existing database objects. | Continue to use the ID schemes in V6 |
| Spend time on the data access layer to optimize performance | Data integrity might be compromised. In other words, need to continue to carry any existing data issue from V6 to V7 |
| Reuse existing Data access layer objects to address existing functionality that work as expected | Database objects do not follow best practice standards |
| Data Migration would be easier compared to redesign |  |
| The Optimized and Simplified data structures will address all identified performance issues and external system integration and data export for analysis. |  |

**Conclusion**

Option 3 is considered the best approach given the timeline and limitation on availability of real data to verify complete and accurate data migration. Lack of documentation on the existing database, limits us to get the warm and comfy feeling on the existing structure to know it all and make sure the new structure will meet all the needs. It’s imperative that we understand the current structure to ensure accurate and complete data migration. Data migration is on the critical path of this project. Keeping this in mind, though Option 1 is the preferred option, we will go with Option 3 which will address one of the most important pain point in V6 that is performance. Option 3 will leave us with time to address the performance issue.

In addition, we will be creating a **EIDSSV7 ONE DB** that will support all EIDSS country. This will reduce the maintenance overhead on supporting multiple DBs. This approach will promote better sustainability on V7 DB for the future. The existing Database structure is intended for Data Capture and not for analysis. In the new EIDSSV7 ONE DB, to address some of existing performance related issues, new ‘Optimized and Simplified Data Structures’ will be put in place. The New ‘Optimized and Simplified Data Structures’ are intended to support the data extraction to third party analytical tools and external system integration needs.

# EIDSS V7 ONE DB approach

Use the CORE DB as the baseline

Compare the Core with the country (GG, AJ, IQ, AM, KZ) DBs one at a time. The compare will be incremental compare. In other words, after comparing the CORE with a country and applying the necessary action as identified below in the scenarios, the new DB with merged objects from the country with become the CORE for the next country compare.

Compare tables

* + - Table structure (Column definition, Triggers, Indexes, Keys)

Compare Stored Procedure

* Name and Content

Compare Functions (Scalar and Tabular)

* Name and Content

## Table Compare Scenarios and action

|  |  |  |
| --- | --- | --- |
| **#** | **Scenario** | **Action** |
|  | Same name and table structure | 1. Keep the core table structure as is. 2. Do not import the Table from the country DB |
|  | Same name but different structure | 1. Keep the core table structure as is 2. Rename the country table by adding the country code as suffix (\_GG, \_AJ, \_AM, \_IQ, \_KZ) to the table 3. Import the new Suffixed table to the CORE 4. Review the differences in all country suffixed tables for the current table and decide on the MERGE rule to create ONE table with no suffix |
|  | Table exist in CORE but not in Country DB | 1. No action |
|  | Table do not exist in CORE but in country DB | 1. Rename the country table by adding the country code as suffix (\_GG, \_AJ, \_AM, \_IQ, \_KZ) to the table 2. Import the new Suffixed table to the CORE 3. Review the differences in all country suffixed tables for the current table and decide on the MERGE rule to create ONE table with no suffix.    1. NOTE: We might end up with only country suffixed Tables |

## Stored proc Compare Scenarios and Action

|  |  |  |
| --- | --- | --- |
| **#** | **Scenario** | **Action** |
|  | Same name and content | 1. Keep the core SP as is. 2. Do not import the SP from the country DB |
|  | Same name but different content | 1. Keep the core SP Content as is 2. Rename the country SP by adding the country code as suffix (\_GG, \_AJ, \_AM, \_IQ, \_KZ) to the SP 3. Import the new Suffixed SP to the CORE 4. Review the differences in all country suffixed SPs for the current SP and decide on the MERGE rule to create ONE SP with no suffix |
|  | SP exist in CORE but not in Country DB | 1. No action |
|  | SP do not exist in CORE but in country DB | 1. Rename the country SP by adding the country code as suffix (\_GG, \_AJ, \_AM, \_IQ, \_KZ) to the SP 2. Import the new Suffixed SP to the CORE 3. Review the differences in all country suffixed SPs for the current SP and decide on the MERGE rule to create ONE SP with no suffix    1. NOTE: We might end up with only country suffixed SPs |

## Function Compare Scenarios and Action

|  |  |  |
| --- | --- | --- |
| **#** | **Scenario** | **Action** |
|  | Same name and content | 1. Keep the core Function as is. 2. Do not import the FUNCTION from the country DB |
|  | Same name but different content | 1. Keep the core FUNCTION Content as is 2. Rename the country FUNCTION by adding the country code as suffix (\_GG, \_AJ, \_AM, \_IQ, \_KZ) to the FUNCTION 3. Import the new Suffixed FUNCTION to the CORE 4. Review the differences in all country suffixed FUNCTIONs for the current FUNCTION and decide on the MERGE rule to create ONE FUNCTION with no suffix |
|  | Function exists in CORE but not in Country DB | 1. No action |
|  | Function does not exist in CORE but in country DB | 1. Rename the country FUNCTION by adding the country code as suffix (\_GG, \_AJ, \_AM, \_IQ, \_KZ) to the SP 2. Import the new Suffixed SP to the CORE 3. Review the differences in all country suffixed FUNCTIONs for the current FUNCTION and decide on the MERGE rule to create ONE FUNCTION with no suffix    1. NOTE: We might end up with only country suffixed FUNCTIONs |