January 13, 2020

SDWIS Prime Data Model & Business Rules Engine Analysis

GDIT

Contents

SDWIS Prime Data Model Analysis

Introduction

The Environmental Protection Agency (EPA) has been engaged in an effort to modernize the existing Safe Drinking Water Information System (SDWIS). The current SDWIS State system is composed of standalone systems that are run at the Primacy Agency (state) level. Information from these individual systems is transmitted to the EPA and aggregated into the SDWIS Fed Data Warehouse for reporting and analysis. States transmit basic information about their public water systems, along with violation and enforcement information.

As part of a modernization effort, the EPA has developed a system to replace SDWIS State. The new system, SDWIS Prime, is a complete rewrite of the old system and is hosted in the EPA's National Computer Center (NCC). All users of SDWIS Prime will access this centralized system over the Internet, facilitating easier information exchange among primacy agencies, regulated entities, EPA Regions, and EPA headquarters. States will no longer have to manage manual data entry and transmittal of data to the EPA, improving the availability of information for analytics.

SDWIS Prime Background

The development of SDWIS Prime has been ongoing for years and the EPA decided to put the effort on hold in the spring of 2019 because of issues surrounding the SDWIS Prime/CMDP database. An internal EPA investigation focused on issues with the reference tables and the challenges of migrating data to the new reference table structure. Skylight Consulting also performed an analysis that validated the initial EPA observations concerning the reference tables and the difficulty in migrating data into the system and exporting data back to the primacy agencies. GDIT was brought in by the EPA to perform a deeper analysis of the data model and describe the options that are available for addressing the data model challenges. That is the focus of this document.

Data Model Analysis

The data model analysis focused on the technical viability of the SDWIS Prime schema from the perspective of data migration and on-going maintenance. For context, data models and data samples from both SDWIS Prime and SDWIS State were analyzed. It was assumed that the SDWIS Prime application was functionally acceptable from a business perspective and this analysis did not cover user interface and workflows. It should also be noted that the Compliance Monitoring Data Portal (CDMP) application and the CMDP data model were not part of the scope of this analysis.

Investigation into the data model revealed that the most challenging design aspect in the schema is the KEY_VALUE_REF table. This is a single table that contains reference data for the SDWIS Prime application. The table uses a key-value storage format (also known as entity attribute value or EVA), with the attributes for multiple reference elements stored in the same column of the table, regardless of the attribute's data type. The following is an example taken from the table, with only the first four columns showing for display purposes:

KEY_VALUE_ID	REF_CATEGORY	KEY_DATA	VALUE_DATA
58508	SANITARY_SURVEY_FREQUENCY	1	1
58509	SANITARY_SURVEY_FREQUENCY	2	2
58510	SANITARY_SURVEY_FREQUENCY	3	3
58511	SANITARY_SURVEY_FREQUENCY	4	4
58512	SANITARY_SURVEY_FREQUENCY	5	5
31470	SANITARY_SURVEY_REASON	SNSP	Sanitary Survey, Partial
31474	SANITARY_SURVEY_REASON	SSVF	Sanitary Survey Follow-up
31471	SANITARY_SURVEY_REASON	SNSV	Sanitary Survey, Complete
38393	SANITARY_SURVEY_REASON	SNSF	Sanitary Survey, Partial - Final
58471	SANITARY_SURVEY_SEVERITY	Recommendation Made	Recommendation Made
58472	SANITARY_SURVEY_SEVERITY	Significant	Significant
58470	SANITARY_SURVEY_SEVERITY	Minor	Minor

As illustrated in the above example, the VALUE_DATA column contains all of the descriptive data for all of the reference data elements. Integer and textual data are residing in the same column. The REF_CATEGORY column is analogous to table name in a relational data model. In a relational data model, SANITARY_SURVEY_FREQUENCY, SANITARY_SURVEY_REASON, and SANITARY_SURVEY_SEVERITY would all be separate tables. A notional view of separate tables for these elements is depicted below:

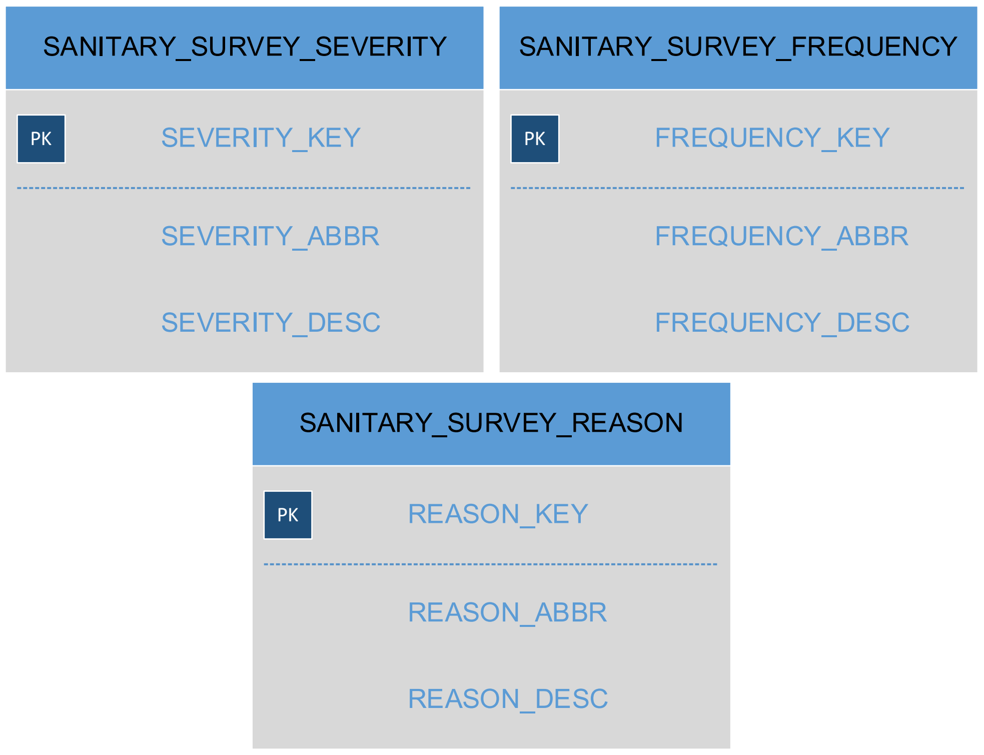

Figure 1- Notional decomposition of key-value data into discrete tables.

The relational database management system (RDBMS) used by SDWID Prime is Oracle 12c. This is the version supported by the EPA National Computer Center. Oracle is an industry-leading enterprise database and is an excellent choice for managing relational data. Unfortunately, the KEY_VALUE_REF table does not conform to the relational design paradigm and as a result, the design of SDWIS Prime forces the Oracle database to do something that it was not designed to do. It appears that this table was designed solely as a storage container for the SDWIS Prime application. Although this table works within the application, from a database perspective it presents challenges to importing and exporting data with standard tools, as well as data management. There are numerous documented reasons to not use key-value table design in a relational database. In general, the most important reasons are:

- The key-value design does not support or conform to rules of database normalization.
- There is no way to make a table column mandatory (the same as NOT NULL)
- There is no way to validate entries by data type since the value column can contain multiple data types.
- There is no way to enforce the correct spelling of attribute names.
- Performing queries against an RDBMS key-value table to return results in a tabular format is complex since getting attributes from multiple rows requires a JOIN to be performed for each attribute.

- A key-value table in an RDBMS cannot follow third normal form normalization rules. Referential integrity cannot be enforced.
- Data migration is challenging because most ETL tools, like Informatica, do not support tabular table to key-value data movement within an RDBMS. EPA has encountered issues trying to migrate SDWIS State data to the SDWID Prime schema.

The typical reason to use a key-value storage format is to give the application flexibility to accommodate new data fields as needed without having to change a table schema. Logically, reference tables in an application are used to store static information that changes infrequently. A reference table, whether in a transactional database or an analytical database is composed of a set of predetermined values. In a transactional system, the reference table insures that users inputting data to a system are all using consistent terminology and spelling. Furthermore, consistency in the reference tables in an analytical system insures that everyone is grouping and filtering on the same values, which engenders trust in the analysis. Under that assumption, reference data does not require storage in a key-value format, and certainly not with the compromises required to use a key-value table in a relational database.

One of the possible reasons for using a key-value format to collapse disparate reference data into a single table is to reduce the number of reference tables in the database schema. This possibility was raised by one the developer of SDWIS Prime. The developer, Attain, supplied a copy of the KEY_VALUE_REF table for analysis. As noted in the above sample from the table, the REF_CATEGORY field is analogous to the table name in a relational schema, which means that the KEY_VALUE_REF table used for this analysis contains the data for 218 individual tables (see Appendix A). Although that can be viewed as a high number of reference tables, managing them is not a challenge since the data in them should logically be fairly static.

It should also be noted that not all of the reference tables used by SDWIS Prime are in KEY_VALUE_REF, which is composed of relatively small reference tables, many of which consist of a single row of data. The sample data supplied by Attain contains 50 reference tables that are of the normal tabular design that is normally seen in a relational database schema. These tables have larger row counts than the ones represented in KEY_VALUE_REF, with the largest, WS_SRCH, having 220,787 rows.

Other Data Observations

In a preliminary internal EPA assessment of the challenges of the SDWIS Prime data model, it was noted that the data model makes extensive use of numeric codes to tie data records to the reference tables, which is an additional hurdle for migrating existing state data since the codes are unique to the SDWIS Prime database schema. Our analysis of the reference tables had revealed that the SDWIS Prime database utilizes surrogate integer keys (the codes mentioned above) for the primary keys in the reference tables. A surrogate key uniquely identifies a single record in a table, but it has no natural relationship to the rest of the columns in the table. Typically, surrogate keys are generated during the table load process by the database. In the case of Oracle 12c, the SEQUENCE function would be used to automatically increment a surrogate key. On the other hand, natural keys are composed of one or more columns of real data in a table and have contextual meaning.

There are pros and cons to both approaches (surrogate versus natural keys) and the use of surrogate keys is not considered a design flaw. However, surrogate keys can present challenges, especially during development. For example, consider a scenario that involves loading U.S. state names into a new reference table that uses the Oracle SEQUENCE function to create a surrogate key (STATE_ID). If the source data is sorted alphabetically, Alabama will get a STATE_ID of 101, Alaska 102, and so on. Let's suppose that in the dev environment, that is how the data is loaded, but in the test environment the source data is not sorted, so when the table is loaded, Illinois might get STATE_ID 101 and California 102, and so on. We would have two tables with the same data and identical row counts, but the different STATE_ID surrogate keys would create incorrect records when joining the data. This could be a possible explanation of the discrepancies that have been noted between the three NCC environments in the initial EPA analysis of the SDWIS Prime data model issues.

Because of the centralized nature of SDWIS Prime, the EPA will benefit from managing the reference data within a master data management (MDM) framework. MDM defines and manages an organizations reference data so that there is one single source of authoritative master data, sometimes referred to as the 'golden record'. MDM will help mitigate situations like the example from the previous paragraph. This is important to SDWIS Prime, because the reference data impacts the ability to accurately enter records into the database, as well as the ability to accurately analyze the data coming out of the database. In the NCC example in the previous section, a comprehensive MDM framework would insure that the same 'golden record' reference tables would be used in the three environments.

A comprehensive enterprise information management initiative for SDWIS Prime should be built around the following best practices:

- Understand and define the reference data challenges that a master data management strategy will solve
- Understand all possible sources of reference data and the data quality of the sources
- Engage Primacy Agencies in the ownership of the master data management initiative
- The technology to implement master data management must be reusable and scalable
- Ensure that the MDM framework fosters communication between all stakeholders, Primacy Agencies and IT

- Audit compliance with policies and procedures. This is critical to the success of any master data management initiative.

Recommendations

This analysis focused on study of the data model and communications with EPA subject matter experts to gain a basic understanding of SDWIS State and SDWIS Prime. The recommendations that follow for Options 1 and 2 are based on the assumption that the SDWIS Prime application has acceptable functionality from a user workflow perspective. The technology stack used by the application is solid, with the exception of the older versions of some components (see Appendix B). The major challenge to SDWIS Prime that guided this analysis was the difficulty faced by EPA in migrating SDWIS State data to the new schema, which is directly related to the key-value table structure of the KEY_VALUE_REF table.

Decompose KEY_REF_DATA

GDIT recommends that EPA decompose the current SDWIS Prime KEY_VALUE_REF table into discrete reference tables. As previously noted in this document, KEY_VALUE_REF is key-value table in a relational database platform, Oracle 12c, that is not designed to support that type of structure. Most ETL tools do not support moving data from relational to key-value within a relational database. For those reasons alone, the KEY_VALUE_REF table needs to be decomposed into individual tables, even if it results in many tables. The list of REF_CATEGORY field values in Appendix A was taken from the sample KEY_VALUE_REF table that Attain supplied for this analysis. That list corresponds to the tables that would be created by the decomposition.

The schema for the discrete reference tables can be created by running SQL queries on KEY_VALUE_REF. We recommend that the new discrete tables be analyzed by a data modeler and subject matter experts familiar with both SDWIS State and SDWIS Prime to determine the following:

1. Can the final table count be reduced by merging similar tables?
2. Can tables be eliminated? For example, SANITARY_SURVEY_FREQUENCY only contains count values (1 to 5). Is a table needed for that?
3. Are SDWIS State fields missing from any of the reference tables?
4. There are also reference tables that are not part of KEY_VALUE_REF that should be examined. For example, the MONITORING_PERIOD reference table is just dates and does not appear to deliver functionality that could not be accomplished by date fields within a table record.

Option 1 – Decompose KEY_VALUE_REF and Implement a Shim API

Populating the new reference tables with SDWIS State data will be relatively straightforward and can be done through scripting or with an ETL tool like Informatica. The SDWIS Prime application will need to use the new structures and the implementation of a shim API could possibly be the easiest way to do this since it would involve minimal changes to the SDWIS Prime source code. The shim is a small library that will transparently intercept database calls from Prime to Oracle and change the parameters passed. The shim will reference a mapping table to redirect calls to KEY_VALUE_REF to the appropriate reference table. Note that the shim is only required for the SDWIS Prime application to interact with the database and it will not impact the ability of primacy agencies to import or export data with external applications.

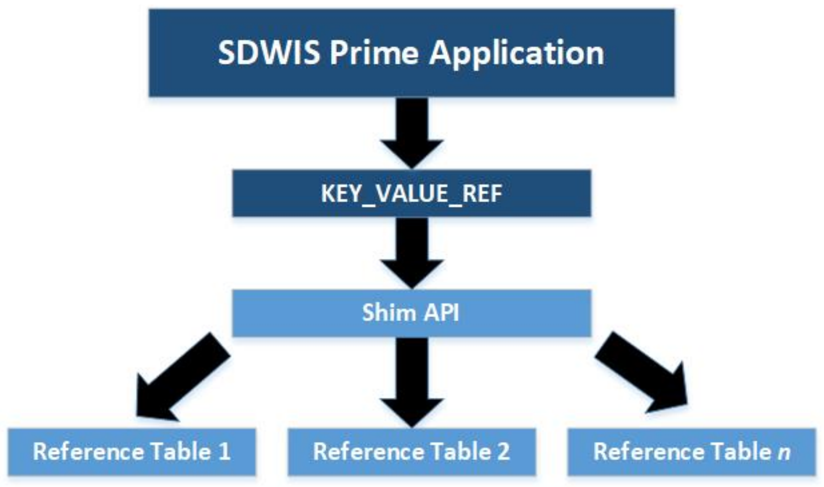

Figure 2- A shim API intercepts and redirects database calls.

Option 2 – Decompose KEY_VALUE_REF and Rewrite SDWIS Application Code

Rewriting the SDWIS Prime application code to use the new discrete reference tables will likely require more work than developing a shim API, but would be the more elegant solution in the long run. During discussions with Attain developers regarding the KEY_VALUE_REF table, it was revealed that the Prime application actually treats the data in KEY_VALUE_REF as separate tables. The Java Persistence API (JPA) layer in SDWIS Prime defines the REF_CATEGORY field in the KEY_VALUE_REF table as a "discriminator" column. The net result is that within the JPA framework, Hibernate (Java object-relational mapping tool) can treat REF_CATEGORY as discrete physical tables. This implies that modifying the source code may require less effort than it otherwise would. GDIT recommends deeper analysis of this to determine the level of effort needed to change the code.

Option 3 – Redesign Entire Database

EPA may want to take a step back and consider a complete redesign of the database based on lessons learned from the SDWIS Prime development. Continuing to use an Oracle RDBMS, with a third normal formal database schema, is certainly a viable option. However, in the years since SDWIS Prime design was started in 2010, there have been considerable advances in NoSQL/document databases, which support schema-less structures. If flexibility in adding new fields to records on-the-fly is needed, NoSQL would be the way to go and platforms like MongoDB would be particularly well-suited to the type of data that SDWIS stores and analyzes. Another database platform to consider is PostgreSQL, an open source database that supports relational and NoSQL structures. The KEY_VALUE_REF table, which is problematic to a relational database, is better suited to this type of database. Document databases typically store weakly-typed JSON (JavaScript Object Notation) that may include arrays and nested documents. Also, scalability, especially horizontal scalability, has historically been better for NoSQL/document databases. The dynamic scalability of a NoSQL database is well-suited to cloud environments. If the EPA's future plans include eventual migration to a public cloud provider like Amazon Web Services (AWS) or Microsoft Azure, then serious consideration should be given to redesigning SDWIS Prime for a NoSQL platform.

Level of Complexity for Options

The order of the three options aligns with the relative level of complexity for each effort. The complexity and predicted timeframe for each option are rough estimates based on information analyzed to date and are subject to change. Appendix C contains the notional cost estimate for each option.

Option 1 – Level of Complexity 4

Option 1 is the least complex of the choices. It involves the creation of new reference tables and an API that maps to those tables, but no changes to the existing SDWIS Prime application code. Although outside of the scope of this analysis, there is a possibility that this change could impact the Compliance Monitoring Data Portal (CMDP) and this needs to be taken into consideration. Option 1 will produce a functioning SDWIS prime application the fastest, but the use of the shim API could Impact future scalability and growth in ways that this analysis cannot predict. On a scale of one to ten, the level of complexity is estimated to be four. The timeframe to complete would be 45 to 60 days, with skilled resources familiar with the SDWIS Prime technical stack and Oracle data modeling and administration.

Option 2 – Level of Complexity 6

Option 2 will be more complex than the first option, since it will involve changes to the source code of the SDWIS Prime Application. This option disposes with KEY_VALUE_REF altogether since the application will interact directly with the reference tables after the code has been modified. Option 2 will take more time to complete than Option 1, but the resulting application will be more robust for meeting the future growth challenges of SDWIS. On a scale of one to ten, the level of complexity is estimated to be six. The timeframe to complete would be 90 to 180 days, with skilled resources familiar with the SDWIS Prime technical stack and Oracle data modeling and administration.

Option 3 – Level of Complexity 9

Obviously, Option 3 is the most complex because it is a clean sheet of paper redesign. As noted above, the technological landscape has changed greatly since the design of SDWIS Prime began almost ten years ago. Although Option 3 will take the longest amount of time to deliver, it could be worthwhile to revalidate the original requirements and make sure that whatever option is chosen today will grow with the EPA in the future. On a scale of one to ten, the level of complexity is estimated to be nine. The timeframe to complete would be nine months to one year, assuming the effort is staffed by resources experienced in NoSQL database technology and cloud architecture.

Conclusions

The KEY_VALUE_REF table is a form of technical debt that must be remedied for SDWIS Prime to be considered an enterprise-level, trusted application. Although the application can programmatically use the table in its present format, the fact that its structure violates the fundamental design philosophy of relational database management systems like Oracle 12c creates too many compromises from a data management perspective. These compromises have already manifested themselves in the data migration challenges that EPA has faced moving the SDWIS State data to SDWIS Prime. There will also be challenges in moving data out of the current schema for analytics and data warehousing. Options 1 and 2 attempt to leverage the EPA's current investment in SDWIS Prime as much as possible. Option 3 is more future-facing and thus requires a greater level of effort in the near term.

Business Rules Engine

BRE Background

As part of the SDWIS Prime development, a Business Rules Engine application (BRE) was integrated into the Prime functionality. The BRE will determine compliance with drinking water rules based on sample data received from the Compliance Monitoring Data Portal (CMDP) and other sources. According to the Association of State Drinking Water Administrators (ASDWA), the business rules were derived from the National Primary Drinking Water Regulations (NPDWR). According to the ADSWA web site, there are six sets of rule logic:

1. Rule Applicability -- Does this NPDWR apply to a particular Public Water Supply System (PWS)?
2. Sample Data Evaluation -- Does this sample result, sample summary, operational summary, satisfy a monitoring schedule, and does/do the result(s) trigger any additional requirements or violations?
3. Monitoring Schedule Evaluation -- Did the PWS collect all the required samples for a particular monitoring schedule, should the frequency of monitoring be increased or decreased, and are there any levels of concern that trigger other requirements or a violation?
4. Returned to Compliance Determination – Does the successful completion of this round of monitoring constitute a return to compliance for an earlier violation?
5. Monitoring Schedule Determination -- What monitoring schedules appear to be appropriate for this new PWS or facility or due to this population or primary source water change?
6. Did the PWS complete this task (e.g., lead consumer notice, DBP operational evaluation) on time?

The SDWIS Prime Business Rules Engine

The BRE as developed for SDWIS Prime is an implementation of the open source Drools business rules management system. It was released under the Apache Software License. It is written in 100% pure Java, runs on any Java virtual machine (JVM). Drools supports the Java Rules Engine API (Java Specification Request 94). This API is an industry standard for the business rule engine and enterprise framework for development and management of business policies in an organization. The SDWIS Prime BRE is not currently using the Java Rules Engine API and communication between the BRE and SDWIS Prime is through the open source Spring Framework, which is fully supported by Drools. The business rules in Drools can be rendered in one of two dialects – Java or MVEL (MVFLEX Expression Language). MVEL is an Apache-licensed Java-based expression language that is similar to Java but supports more concise and expressive code that is more user-friendly. It provides a familiar syntax for Java programmers while also adding syntactic ‘sugar’ for short and concise expressions. SDWIS Prime uses MVEL.

The purpose of this analysis of the BRE is to determine if the BRE implementation can be decoupled from the SDWIS Prime application. Attain, the developer of the BRE has documented the following points that describe its functionality:

- Contains rules based on the NPDWR as decision tables
- Analyzes data from SDWIS Prime to determine compliance with those rules (data includes PWS inventories, samples data, and schedules)
- Evaluates compliance with schedules and creates candidate violations when there are failures
- Executes the rules in a runtime production environment separate from the application code

Results of BRE Analysis and Conclusions

To determine the feasibility of decoupling the BRE from SDWIS Prime, an analysis of Drools documentation was conducted, as well as meetings with Attain developers. The key findings from this analysis are:

- The current BRE is an ‘out of the box’ Drools implementation. The source code has not been modified.
- The BRE and SDWIS Prime do not share any tables in the database.
- All BRE interactions are triggered by Java Message Service (JMS) messages. This is part of the Spring Framework.
- The BRE is not directly invoked by SDWIS Prime users, so all interaction between the applications is behind the scenes.
- The data that the BRE needs for functioning is stored in Excel spreadsheet files (decision tables - Drools supports managing rules in a spreadsheet format) and XML files (Spring-to-Drools interface).

Based on the above observations, decoupling the BRE from SDWIS Prime should be a straightforward effort since it is a basic Drools implementation with no source code modifications. Moving the BRE to the existing SDWIS State application should require minimal effort. The lack of modifications should also make integration with other primacy agency systems feasible. This is basically a reconfiguration initiative.

There are, however, a couple of things to consider for this effort. The version of Drools in use by the BRE is 5.5.0, which was released in 2012. The latest version of Drools is 7.29. It would be advisable to use the latest version of the application if the EPA decides to move forward with the decoupling. Secondly, the BRE will need to be reconfigured to communicate with whatever application it will be coupled with. As previously noted, the BRE communicates to SDWIS Prime through the Spring Framework and as of Drools version 6, which was released in 2013, the Drools Spring integration underwent a complete makeover. Assuming that the BRE decoupling will include updating to a new version of Drools, this is where the bulk of the effort will expended in reconfiguring the BRE.

GDIT

Appendix

Appendix A – Listing of New Reference Tables

The following list of notional table names was derived from the REF_CATEGORY column of the KEY_VALUE_REF table that was supplied to GDIT for this analysis. GDIT recommends that these be used as a starting point for decomposing the KEY_VALUE_REF table. Deeper inspection with a SDWIS subject matter expert may reveal that some of these table can be consolidated and some eliminated to reduce the total number of reference tables needed.

NOTIONAL TABLE NAME
90TH_CALCULATION_METHOD
ACTIVITY_DATE
ACTIVITY_DATE_TYPE
ACTIVITY_REASON
ACTIVITY_STATUS
AERATOR_TYPE
AK_REMOTE_VLG
ANALYTE_CLASS
ANALYTE_TYPE
CASING_TOP_MEASURE_RELATIVE
CENSUS_TRACT
CFG_CDX_AUTH_CRED
CFG_CDX_AUTH_ID
CFG_CDX_AUTH_NAMESPACE
CFG_CDX_AUTH_URL
CFG_CDX_USER_AUTH_NSPACE
CFG_CDX_USER_AUTH_URL
CFG_MSG_CATEGORY
CFG_MSG_PRIORITY
CFG_PRIME_AUTH
CFG_PRIME_KEY
CFG_PRIME_KEY_TTL_HOURS
COMPLIANCE_VALUE_TYPE
CONGRESS_DIST
CONTACT_TYPE
COORDINATE_SOURCE
COUNTRY
CUSTOM_DATA_CATEGORY
CUSTOM_DATA_TYPE

NOTIONAL TABLE NAME
CUSTOM_FIELD_DATE_FORMAT
CUSTOM_FIELD_PAGE
CUSTOM_FIELD_PICKLIST_TYPE
CUSTOM_FIELD_TABLE
CUSTOM_FIELD_TEXT_LENGTH
CUSTOM_FIELD_TYPE
DEFICIENCY_PLAN_TASK
DEFICIENCY_SEVERITY
DEFICIENCY_TYPE
DISTRICT
DVALVER_RULE_LANGUAGE
DVALVER_RULE_TYPE
EPA_REGION
FAC_FLOW_RATE_TYPE
FAC_FLOW_RATE_UOM
FAC_FLOW_UOM
FAC_INDICATOR_TYPE
FAC_INDICATOR_VALUE
FAC_MEASURE_TYPE
FAC_MEASURE_UOM
FAC_PUMP_TYPE
FACILITY_ACTIVITY_STATUS
FACILITY_AVAILABILITY
FACILITY_CATEGORY
FACILITY_FLOW_CONNECTION_TYPE
FACILITY_FLOW_PROCESS_WATER_TYPE
FACILITY_QUADRANT
FACILITY_SOURCE_TREATMENT_STATUS
FACILITY_TYPE
FEDERAL_WATER_SOURCE_TYPE
FILE_VALIDATION_STATUS
FILTER_MEDIA_TYPE
FRANCH_AREA
GEO_STATE
GEOGRAPHIC_AREA_TYPE
GEOMETRIC_TYPE
GWR_TRG_BEGIN_DT_DAYS
GWR_TRG_END_DT_DAYS

NOTIONAL TABLE NAME
HORIZONTAL_COLLECTION_METHOD
HORIZONTAL_REFERENCE_DATUM
HYDRO_UNIT
INACTIVE_REASON
JOB_DETAIL_STATUS
JOB_STATUS
JOB_TYPE
LAB_CERTIFICATION_LEVEL
LE_ADDRESS_TYPE
LE_CATEGORY
LE_CERT_AGENCY_TYPE
LE_EMAIL_TYPE
LE_GOV_AGENCY_TYPE
LE_ORG_TYPE
LE_PHONE_TYPE
LE_SALUTATION
LE_STATUS
METERING_STATUS
METRO_STAT_AREA
MNTRG_SCH_PACKAGE_ROLE
MNTRG_WAIVER_STATUS
MNTRG_WAIVER_TYPE
MP_SMF_TYPE
MS_DESIGNATED_STATE_PERIOD
MS_DESIGNATED_STATE_PERIOD_PARENT
MS_INTERVAL_UNIT
MS_OVERRIDE_REASON
MS_SAMPLE_TYPE
MS_SCHEDULE_TYPE
MS_STATUS
NAT_RSRC_DIST
OD_CHLORAMINE_DS_RESIDUAL_REPORTING_TYPE
OD_CHLORAMINE_OPERATIONAL_STATUS
OD_CHLORINE_RESIDUAL_MEASURE
OD_CHLORINE_WATER_SOURCE
OD_COMBINED_POP_SERVED
OD_IFE_INDIVIDUAL_EVENT_GREATER_THAN_10000
OD_IFE_INDIVIDUAL_EVENT_LESS_THAN_10000

NOTIONAL TABLE NAME
OD_LCRWPQ_MEASURE_UOM
OD_MNTRG_PERIOD_QUARTERLY
OD_TTHMHAA5_MEASURE_UOM
OPERATIONAL_SAMPLE_TYPE
OWNER_TYPE
PA_DETERMINATION_REASON_CODE
POPULATION_TYPE
PRIMACY_TYPE
PROFILE_CHANGE_REQUEST_ACTION
PROFILE_CHANGE_REQUEST_LAB_MODULE
PROFILE_CHANGE_REQUEST_PWS_MODULE
PROFILE_CHANGE_REQUEST_STATUS
PROFILE_CHANGE_REQUEST_TYPE
REFERENCE_POINT
REGION
REGULATING_AGENCY
REGULATORY_LEVEL_TYPE
REGULATORY_LEVEL_UOM
REPORTED_FILTRATION_STATUS
RESULT_UOM_CATEGORY
RESULT_UOM_TYPE
RTCR_ADOPTED_REDUCED_MON_PROVISIONS
RTCR_RPT_BEGIN_DT_DAYS
RTCR_RPT_END_DT_DAYS
SA_REJECT_REASON
SAMPLE_CATEGORY
SAMPLE_FIELD_RESULT_PARAM
SAMPLE_FIELD_RESULT_PARAM_UOM
SAMPLE_PURPOSE
SAMPLE_REJECT_REASON
SAMPLE_REPEAT_LOCATION
SAMPLE_RESULT_CHEM_UOM
SAMPLE_RESULT_DATA_QUALITY
SAMPLE_RESULT_INTERFERENCE
SAMPLE_RESULT_MEASURE
SAMPLE_RESULT_MEASURE_UOM
SAMPLE_RESULT_MICRO_SOURCE_TYPE
SAMPLE_RESULT_REJECT_REASON

NOTIONAL TABLE NAME
SAMPLE_RESULT_TYPE
SAMPLE_RESULT_VOL_ASSAYED
SAMPLE_RESULT_VOLUME
SAMPLE_SUMM_CATEGORY
SAMPLE_SUMM_LC_UOM
SAMPLE_TYPE
SAMPLE_TYPE_CRYPTO
SAMPLE_VOLUME
SAMPLING_POINT_ACTIVITY_STATUS
SAMPLING_POINT_TYPE
SAMPLING_POINT_WATERTREAT_STATUS
SANITARY_SURVEY_ASSESSMENT
SANITARY_SURVEY_CATEGORY
SANITARY_SURVEY_EVALUATION
SANITARY_SURVEY_FREQUENCY
SANITARY_SURVEY_REASON
SANITARY_SURVEY_SEVERITY
SCHEDULE_ACTIVITY_CATEGORY
SCHEDULE_ACTIVITY_ENFORCEMENT_STATUS
SCHEDULE_ACTIVITY_EVENT
SCHEDULE_ACTIVITY_STATUS
SEC_RESOURCE
SEC_RESOURCE_GROUP
SEC_ROLE
SELLER_TREATMENT_STATUS
SERVICE_CHARACTERISTIC_TYPE
SERVICE_CONNECTION_STATUS
SERVICE_CONNECTION_TYPE
SITE_VISIT_PRIMARY_REASON
SITE_VISIT_STATUS
SLUDGE_REMOVAL_TYPE
SMP_PNT_WATERTREAT_STAT
SOURCE_MAP_SCALE
ST_HOUSE_DIST
ST_SENATE_DIST
STANDARD_RESPONSE_APPLIED
STANDARD_RESPONSE_BASE_DATE
STANDARD_TYPE

NOTIONAL TABLE NAME
STATE_PROVINCE
SUBDIVISION
SWTR_CATEGORY
TASK_CATEGORY
TOWNSHIP
TREATMENT_UNIT_FLOW_CONN_TYPE
TREATMENT_UNIT_SUBTYPE
TREATMENT_UNIT_TYPE
TRIBAL_AREA
US_CITY
US_COUNTY
USER_PA_STATUS
VALIDATION_LOG_SAMPLE_SUB_CATEGORY
VALIDATION_LOG_TYPE
VERIFICATION_METHOD
VERTICAL_COLLECTION_METHOD
VERTICAL_REFERENCE_DATUM
VIOLATION_CV_UOM
VIOLATION_MCL_UOM
VIOLATION_REJECT_REASON
VIOLATION_RESCINDED_CODE
VIOLATION_STATUS
VIOLATION_TYPE
VISIT_TYPE
WATER_SOURCE_TYPE
WATER_SYSTEM_ACTIVITY_STATUS
WATER_SYSTEM_TYPE
WELL_AQUIFER_TYPE
WELL_CASING_UOM
WS_FLOW_RATE_TYPE
WS_FLOW_RATE_UOM
WS_INDICATOR_TYPE
WS_INDICATOR_VALUE
WS_MEASURE_TYPE
WS_MEASURE_UOM
WS_STATE_TYPE
WS_STATE_WATER_SRC_TYPE

Appendix B – SDWIS Prime/BRE Technology Stack

SDWIS Prime/BRE Technology Stack
Spring Framework 4.3.12
Spring Security 4.2.5
Oracle 12c RDBMS
Oracle JDBC
Hibernate 5.2.2
Java 8
Jersey REST 1.19.3
Drools 5.5.0
ActiveMQ 5.15.x
NodeJS, AngularJS 1.7.3, Grunt, Karma
AngularJS 1.7.3
Grunt - JavaScript Task Runner
Karma - JavaScript Test Runner
CMDP 1.x (Some components)
Central Data Exchange (CDX)

GDIT

Appendix C – Notional Estimates for Options 1, 2, and 3

The estimates below are notional and should only be considered as a rough estimate of the costs involved. A notional blended rate of $180 per hour was used for each resource. Real rates will vary by experience and skill level.

Option 1 – New Reference Tables and Shim API – 12 Weeks

Resource	Total Hours	Rate	Amount
Technical Lead	480		
Senior Software Engineer	480		
Junior Software Engineer	480		
Database Architect/DBA	480		
QA Analyst	480		
	2,400	**$180**	**$432,000**

Option 2 – New Reference Tables and Rewrite Application Code – 36 Weeks

Resource	Total Hours	Rate	Amount
Technical Lead	1440		
Senior Software Engineer	1440		
Junior Software Engineer	1440		
Database Architect/DBA	1440		
QA Analyst	1440		
	7,200	**$180**	**$1,296,000**

Option 3 – Rewrite Application with new Database – 1 Year

Resource	Total Hours	Rate	Amount
Technical Lead	1,960		
Senior Software Engineer	1,960		
Junior Software Engineer	1,960		
Database Architect	980		
NoSQL Database Engineer	1,960		
Senior UI Engineer	1,960		
Junior UI Engineer	1,960		
QA Analyst	1,960		
	14,700	**$180**	**$2,646,000**

The Excel file with the complete worksheets for these tables is provided as a separate attachment. A baseline of 1,960 hours per year is used for this calculation.