National Archives Catalog (NAC)

**and**

**Description and Authority Services (DAS)**

Software Design Document (SDD)

**For the DAS and NAC Modernization**

**Preliminary Version**

Version 1.0

September 25, 2017

Prepared for:

National Archives and Records Administration (NARA)

Catalog Task NAMA-16-G-005, DME 16 Task 005

Prepared by:



1760 Old Meadow Road

McLean, VA 22102

This document contains Proprietary Information provided by PPC – A DSA Company,

Handle in Accordance with Proprietary and Confidential Restrictions

***Acknowledgements:***

***PPC would like to acknowledge the significant technical contributions made by the following staff:***

* ***Dr. Urmi Majumder***
* ***John Henson***
* ***Edwin Piedmont***
* ***Matthew Mariano***
* ***Fawad Shaikh***

Table of Contents

[Version Control 7](#_Toc494116795)

[1 Introduction 8](#_Toc494116796)

[1.1 Background 8](#_Toc494116797)

[1.1.1 Description and Authority Services 8](#_Toc494116798)

[1.1.2 National Archives Catalog 8](#_Toc494116799)

[1.2 Purpose of this Document 9](#_Toc494116800)

[1.3 Scope and Organization of this Document 11](#_Toc494116801)

[1.4 References 12](#_Toc494116802)

[1.5 Assumptions 12](#_Toc494116803)

[2 DAS REST API Framework Design 13](#_Toc494116804)

[2.1 Overview 13](#_Toc494116805)

[2.2 Resource Overview 13](#_Toc494116806)

[2.3 Collection Resource 14](#_Toc494116807)

[2.3.1 Response Format for Collection Resources 15](#_Toc494116808)

[2.4 Single Resource 18](#_Toc494116809)

[2.4.1 Read Operation 18](#_Toc494116810)

[2.4.2 Update Operation 19](#_Toc494116811)

[2.4.3 Delete Operation 19](#_Toc494116812)

[2.4.4 Create Operation 19](#_Toc494116813)

[2.5 Sub-Resource Collection 21](#_Toc494116814)

[2.6 Modeling Ingestion Operations 21](#_Toc494116815)

[2.6.1 Description 21](#_Toc494116816)

[2.6.2 Authority 22](#_Toc494116817)

[2.7 Modeling Search Operations 22](#_Toc494116818)

[2.7.1 Description 22](#_Toc494116819)

[2.7.2 Authority 22](#_Toc494116820)

[2.8 Modeling Workflow Operations 23](#_Toc494116821)

[2.8.1 Description 23](#_Toc494116822)

[2.8.2 Authority 23](#_Toc494116823)

[2.9 Modeling Reporting Operations 24](#_Toc494116824)

[2.9.1 Description 24](#_Toc494116825)

[2.9.2 Authority 24](#_Toc494116826)

[2.10 Modeling Export Operations 24](#_Toc494116827)

[2.10.1 Description 24](#_Toc494116828)

[2.10.2 Authority 24](#_Toc494116829)

[2.11 Modeling User Operations 24](#_Toc494116830)

[2.12 Implementation 25](#_Toc494116831)

[3 Description Ingest Module Design 26](#_Toc494116832)

[3.1 Overview 26](#_Toc494116833)

[3.2 Detailed Design 28](#_Toc494116834)

[3.2.1 Database Support 28](#_Toc494116835)

[3.2.2 Software Components 29](#_Toc494116836)

[3.2.3 Logical Flow 30](#_Toc494116837)

[3.2.4 Exception Handling 32](#_Toc494116838)

[4 Search Engine Design 33](#_Toc494116839)

[4.1 Overview 33](#_Toc494116840)

[4.2 Search Index Design 34](#_Toc494116841)

[4.3 Search Schema Design 37](#_Toc494116842)

[4.4 Query Builder Design 37](#_Toc494116843)

[4.5 Data Migration 37](#_Toc494116844)

[4.5.1 ETL Design 38](#_Toc494116845)

[4.6 Search Access Layer Design and Implementation 38](#_Toc494116846)

[5 Database Design 39](#_Toc494116847)

[5.1 Overview 39](#_Toc494116848)

[5.2 Conceptual Model 41](#_Toc494116849)

[5.2.1 Authority Model 41](#_Toc494116850)

[5.2.2 Description Model 42](#_Toc494116851)

[5.3 Physical Model 44](#_Toc494116852)

[5.3.1 Authority Model 44](#_Toc494116853)

[5.3.2 Description Model 48](#_Toc494116854)

[5.4 Data Migration from Oracle on EC2 to Amazon PostgreSQL Aurora RDS 51](#_Toc494116855)

[5.4.1 ETL Design 52](#_Toc494116856)

[5.5 Database Access Layer Design and Implementation 54](#_Toc494116857)

[A. Appendix: DAS Data Model 56](#_Toc494116858)

[A.1 Description Ingest Database Table 56](#_Toc494116859)

[A.2 Description and Authority Data Dictionary 58](#_Toc494116860)

[A.3 ETL Script for Authority List Migration from Oracle to Aurora 69](#_Toc494116861)

List of Figures

[Figure 1: Current DAS Ingestion Module Components 26](#_Toc494122018)

[Figure 2: Future DAS Ingestion Module Components and their Integration 28](#_Toc494122019)

[Figure 3: DAS NAC Combined Search Index 34](#_Toc494122020)

[Figure 4: DAS Description Keyword and Advanced Search 35](#_Toc494122021)

[Figure 5: DAS Authority Keyword and Advanced Search 36](#_Toc494122022)

[Figure 6: NAC Keyword Search 37](#_Toc494122023)

[Figure 7: AWS Elasticsearch Resource Domain Architecture 38](#_Toc494122024)

[Figure 8: As-Is Hybrid Data Model 40](#_Toc494122025)

[Figure 9: DAS To-Be Fully Normalized Data Model for Authority 42](#_Toc494122026)

[Figure 10: DAS To-Be Fully Normalized Data Model for Description 44](#_Toc494122027)

[Figure 11: Primary Authority Physical Data Model 48](#_Toc494122028)

[Figure 12: Relationship among Description and Physical Occurrence Tables 50](#_Toc494122029)

[Figure 13: Amazon RDS Aurora DB Cluster 52](#_Toc494122030)

List of Tables

[Table 1: CRUD Functionality and HTTP Verb Correspondence 15](#_Toc494122043)

[Table 2: authority\_list Table Columns 44](#_Toc494122044)

[Table 3: primary\_authority Table Columns 45](#_Toc494122045)

[Table 4: Child Tables 48](#_Toc494122046)

[Table 5: Description Ingest Database Table 55](#_Toc494122047)

[Table 6: Description and Authority Data Dictionary 58](#_Toc494122048)

Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author/Reviewer | Summary Description |
| 0.9 | 9/25/2017 | Urmi Majumder | Initial Draft (Preliminary Version) |
| 1.0 | 9/25/2017 | Aurora Rischak | QC Review |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Background

### Description and Authority Services

The National Archives and Records Association (NARA) Description and Authority Services (DAS) is an archival repository built by Project Performance Corporation (PPC) to replace Archival Research Catalog (ARC). DAS provides data-entry capabilities to NARA’s staff, and this data is made available to the public via the National Archives Catalog (NAC) system. DAS allows NARA’s staff to describe and control archival holdings at appropriate descriptive levels for all media, as well as to create and maintain Authority records that provide access points to the Description data. DAS has about 600 users (as of September, 2017) throughout the United States. As DAS is responsible for the collection of metadata related to the publicly accessible information about NARA’s holdings, this data must go through a rigorous workflow process that ensures data integrity prior to becoming publicly available. The application ensures this integrity through the implementation of business rules that must be met before Descriptions and Authorities can be published into the approved domain. Additionally, DAS supports a comprehensive role- and group-based security model and offers users sophisticated search capabilities, workflow, record editing and global operations. From an implementation perspective, DAS is a three-tier application, hosted in the Amazon Web Services (AWS) Cloud, with a .NET front-end, Java middle tier and an Oracle data store.

### National Archives Catalog

NAC, originally called Online Public Access (OPA), is the online public portal to NARA’s records, and is key to maintaining NARA’s commitment to open government through public access to high-value records. NARA has stated “We will reach beyond the traditional role of making records available for others to discover and will make access happen by providing flexible tools and accessible resources that promote public participation.” NAC is the end-product of this goal to make all records available to the public in digital form to ensure that anyone can explore, discover, and learn from NARA holdings.

The catalog currently provides access to records created or received by organizations of the executive, legislative and judicial branches of the Federal Government. Most of the records are textual, coming from DAS, but NAC also provides access to other forms of media such as microfilm, maps and charts, architectural and engineering plans, aerial photographs, motion picture reels, video recordings, sound recordings, computer data sets, still pictures and others, currently not available elsewhere online. Besides content from DAS, NAC searches all web pages on archives.gov, nara.gov, ofr.gov, presidentialtimeline.org, and many others and presents those pages in the search results along with any catalog records, thus enabling researchers to work more efficiently.

Apart from searching the content via their website, NAC allows the public to download this content, as well as access the same programmatically via application programming interfaces (APIs). Furthermore, NAC allows citizen archivists to transcribe some of the millions of digitized pages of records, thus improving search relevancy and accessibility to historical federal records. Additionally, researchers and the public alike can tag, translate and comment on records, thus allowing NARA to add contributions from external sources to its own repository. The transformation of NAC into a social catalog enables ongoing contributions and scalability of crowdsourcing to all records within the catalog.

## Purpose of this Document

As of September 2017, we have almost 19 million Descriptions in DAS stored as structured Extensible Markup Language (XML) and 37 million links to digitized materials, such as:

* American Standard Code for Information Interchange (ASCII) text
* Image formats, e.g. JPEG, Tagged Image file (TIF), Graphics Interchange Format (GIF), Portable Network Graphics (PNG), bitmap (BMP)
* Audio/visual formats, e.g., Audio Video Interleave (AVI), MOV, Moving Pictures Export Group (MPEG)-4 (MP4), and Windows Media Video (WMV)
* Microsoft (MS) Office formats such as Excel, PowerPoint, and Word
* Sound formats including MP3 and Waveform Audio File Format (WAV)
* Adobe Portable Document Format (PDF) format
* Hypertext Mark-up Language (HTML) pages

NARA is looking to ingest an additional 6 billion Descriptions and 25 billion Digital Objects by the end of this decade. Based on PPC’s long experience supporting NARA and working with DAS and NAC systems, PPC knows that these systems cannot accommodate the significant growth anticipated in Digital Objects and corresponding Descriptions, and meet NARA’s strategic goal of Make Access Happen. The current system design for DAS using Oracle 12c running on single-node first-generation AWS instances as a data store cannot support the anticipated growth in digital images and corresponding descriptions. While current NAC search servers can scale horizontally to support such growth, it is dependent on a weekly export of Description and Authority data as structured XML from DAS to be ingested by its Content Processing module and indexed in its Apache Solr based search engine before the public can discover information about the new and updated NARA’s holdings through keyword, advanced searches, by provenance, and other access points controlled by the DAS Authority files.

The primary business driver for this modernization effort is for PPC to build a system that can keep up with the ever-increasing volume of archival Descriptions that are needed to describe the billions of records generated by the federal government each year. To that effect, PPC will redesign DAS as a scalable application that will allow NARA to not only meet but exceed their existing production quotas for entering records into DAS, and establish tighter integration between DAS and NAC so that records are available for the public to view in NAC as soon as they are approved in DAS. PPC will design a new, horizontally-scalable data store that is shared between DAS and NAC, and a new search cluster that can service both NAC and DAS. The shared data store and search engine approach eliminates the need for the weekly DAS export and the added delays that stem from reprocessing the Descriptions and Authorities in the NAC ingestion server. Instead, the indexing of the data can happen directly in DAS, and the NAC APIs can be modified to present only a subset of DAS data to public users. The combined data store and search cluster for NAC and DAS is expected to provide substantial cost savings from the current cost of Oracle licenses, and the number and type of AWS instances that currently host Oracle (DAS) and Apache Solr (NAC), and perform NAC ingestion. Costs are also reduced by using AWS managed services for both relational database (RDBMS) and search engine (Elasticsearch) instead of individual data store and search engine for each application in a high-availability (HA) configuration. For instance, the Oracle database for DAS replicates in near real-time to a standby Oracle instance using Oracle Data Guard while the NAC search engine is set up in a cluster configuration to increase query capacity and provide failover.

We will apply our in-depth knowledge of the DAS and NAC systems to begin the modernization process, and to define a new system architecture and software design. See Section 1.2.3 of the DAS/NAC Modernization System Architecture Document: Preliminary Version for details on the different modules that currently make up DAS and NAC, Section 1.2.4 of the same document for details of the problem with the existing setup and Section 1.3 addresses how PPC envisions the architecture of the modernized applications to alleviate these problems. Thus, the System Architecture Document describes the different DAS and NAC system components, and how the new system architecture will look upon completion. The Software Design Document, on the other hand, describes the new data model and the search schema that will be used to implement the new architecture. It also highlights the portions of the existing software (such as Description Bulk Processing, DAS Web Services) that need to be changed.

This design effort will result in a draft re-design of DAS and NAC, and will provide actionable guidance for follow-on DAS and NAC modernization efforts to build a system that can keep up with the ever-increasing backlog of federal records that need to be described in DAS and scale reliably (without affecting performance) to support billions of archival descriptions in the system by the end of this decade. We believe that our in-depth experience developing and enhancing both systems since 2010, particularly the enhancement to the architecture and improved performance of interaction between the DAS and NAC systems over the past two Task Orders, will enable us to put forward a design for a horizontally scalable combined data store and search engine for both applications in this Software Design document.

## Scope and Organization of this Document

This Software Design Document is a preliminary document that represents the current plan for the software design as agreed upon by PPC engineers and NARA Systems Engineering leadership at the time of writing. Certain sections are left incomplete, and in those cases PPC has provided as much information as possible with the expectation that those topics will be further developed in later releases of this document. However, as the purpose of this document is to capture the low-level design of the database model, search schema and updated software modules, PPC’s goal was to make this design document as complete and detailed as possible given the information provided. The overall approach taken was to pull in, assemble, and harmonize design elements from the following sources:

* Existing design documents for DAS and NAC
* Information gleaned from own creation of an initial functioning prototype functioning prototype
* PPC’s understanding of how the current systems are designed and operate
* NARA feedback in the daily stand-ups and weekly technical meetings

This document is organized into five main sections:

1. Introduction
2. Representational State Transfer (REST) API (Application Programing Interface) Framework Design for DAS
3. Batch Processing Module Design
4. Search Engine Design
5. Database Design

Each section contains several subsections that have been developed to cover each section in more details. For instance, the introductory section provides a functional overview of the current applications (Section 1.1) and describes the purpose (Section 1.2) and scope (Section 1.3) of this document as well as lists references (Section 1.4) and assumptions (Section 1.5). Section 2 addresses the external interface designs, including design of the primary resources and the associated end points in DAS using REST principles. Section 3 presents the design of the different software modules that make up the batch processing system. Section 4 discusses the logical partitioning of the search engine index (Section 4.1), schema (Section 4.2), the query builder design (Section 4.3), and the data migration plan from Apache Solr to Elasticsearch (Section 4.4). Finally, Section 5 provides a detailed database design, including both conceptual (Section 5.2) and physical (Section 5.3) database models for both Authorities and Descriptions, and a data migration plan for the existing Authorities and Descriptions from Oracle to PostgreSQL compatible Aurora Relational Database Service(RDS) (Section 5.5).

## References

The following documents were referenced in the creation of this design document:

* Development of the NARA Description and Authority Services, Detailed Design Document, Version 2.0
* NARA Catalog Architecture Design – Catalog Perspective, Version 1.6

## Assumptions

This section lists the set of assumptions identified prior to and during Task Order 5 performance:

* DAS API Design: This document assumes that DAS Web Services will be designed using REST API principles.
* Search Engine Design: This document assumes AWS Elasticsearch service can be used to implement the physical search engine for both DAS and NAC.
* Database Design: This document assumes we can design a relational database for the DAS data store and use Amazon Aurora with PostgreSQL compatibility as the physical data store.

# DAS REST API Framework Design

## Overview

REST, introduced and defined in 2000 by Roy Fielding, is an architectural style for designing distributed systems. It is not a standard but a set of useful constraints, such as being stateless, having a client server relationship, and a uniform interface. REST is not strictly related to Hypertext Transfer Protocol (HTTP), but it is most commonly associated with it. For the full modernization efforts, we will develop a set of REST APIs to allow both DAS UI and external consumer applications to authenticate, query and fetch Description and Authority records, as well as create new records based on their role. Specifically, the application tier in DAS will host a set of REST Web Services which, when combined, will implement the DAS System methods and expose data. The services will be designed to stand alone without reference to the Web UI. The UI client as well as the batch processing module and partner agencies will communicate with the application tier via a set of uniform HTTP methods such as GET, POST, PUT, and DELETE on resources such as Descriptions, Authorities, Users and so on using JavaScript Object Notation (JSON) primarily to represent the data objects and their attributes. These interactions will be completely stateless and no client context will be stored on the application server between requests. Instead, the API consumer will hold the session state.

DAS APIs will be designed to support the many functionalities of DAS such as creating, accessing and searching Descriptions and Authorities or creating, accessing and searching user related resources such as work trays, roles and groups in DAS. They will be parameter-based and will support the GET, POST, PUT, and DELETE methods in the following manner:

* GET for performing searches and retrieving Description and Authority metadata
* POST when creating new records (e.g. creating a File Unit Description)
* PUT when updating existing records (e.g. updating person name in Person Authority)
* DELETE when removing records (e.g. deleting an Organization Name Authority)
* All accesses to the API will use Hypertext Transfer Protocol Secure (HTTPS)

## Resource Overview

The fundamental concept in any RESTful API is the *resource*. A resource is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it (corresponding to the standard HTTP GET, POST, PUT and DELETE methods). Examples of resources in DAS will be User, Workflow, Ingest, Search, Authority, Description, as so on. Each resource is identified by one or more Uniform Resource Identifiers (URIs). An URI has three primary components:

* Version
* Namespace
* Resource

**Version:** The URI should include */V{version}* with the major version *{version}* as a prefix. URL-based versioning will be utilized for its simplicity of use for API consumers, versus the more complex header-based approach. For instance, for the reference application that will be designed and implemented as part of this Task Order, all endpoint URLs will begin with */v1/*.

**Namespace**

In any URI, the first noun is considered a “namespace.” Namespaces generally reflect the end-user's perspective on how the application works. For the full modernization efforts, the following namespaces will be used:

* Entity
* Ingest
* Search
* Workflow
* User-profile
* Report
* Export

The combined URI Template with version and namespace information will be */v{version}/{namespace}/*. Using this convention, Description Ingest endpoint will start with */v1/ingest/*.

**Resource**

Resources can be considered either a collection resource or a single resource. They are described in detail in the following sections.

## Collection Resource

Collection resources are data resources that typically support Create, Read, Update, and Delete (CRUD) functionality. As such, CRUD resources should be implemented with adherence to POST/GET/PUT/DELETE HTTP verbs. CRUD functionality and HTTP verb correspondence is listed below:

|  |  |
| --- | --- |
| Table : CRUD Functionality and HTTP Verb Correspondence | |
| HTTP Verb | CRUD Operation |
| POST | Create |
| GET | Read |
| PUT | Update |
| DELETE | Delete |

Collection resource names should be plural nouns. This helps visually disambiguate collections from singletons. For instance, for the reference application and beyond, we will define collection resources such as:

* Descriptions
* Authorities
* DescriptionJobs
* AuthorityJobs
* Users
* Groups
* Roles
* AuthoritySearches
* DescriptionSearches

Note that this is not a complete list.

Hence, the combined URI Template with version, namespace and resource name will be

*/v{version}/{namespace}/{resource}*. Using this convention, the endpoint for retrieving all Descriptions will be */v1/entity/descriptions*.

### Response Format for Collection Resources

For a GET request on a collection resource, an array of resources will be listed in the items field. Fields like total\_items and total\_pages will be used to provide context to paged results. The names of collection resource fields will be consistent across various resources to allow API clients to create generic handling for using the provided data across various resource collections.

#### Paging

Pages of results will be referred to consistently by the query parameters page and page\_size, where page\_size refers to the amount of results per request, and page refers to the requested page. Additionally, responses will include total\_items and total\_pages whenever possible, where total\_items indicates the total items in the requested collection, and total\_pages is the number of pages (calculated from total\_items/page\_size).

#### Sorting

sort\_by and sort\_order will be implemented to allow for collection results to be sorted. sort\_by will be a field in the individual resources, and sort\_order should be asc or desc. An example request for retrieving all Descriptions in DAS is *GET /v1/entity/descriptions appears below*.

Example response is

{

"total\_items": 1,

"total\_pages": 1,

"items": [

{

"item": {

"dataControlGroup": {

"groups": "RDEP",

"groupId": "ou=NWME,ou=groups"

},

"digitalObjectCount": 0,

"isUnderEdit": false,

"naId": 62600935,

"physicalOccurrenceArray": [

{

"copyStatus": {

"termName": "Preservation"

},

"locationArray": [

{

"facility": {

"termName": "Electronic Records Archives (ERA)"

}

}

],

"mediaOccurrenceArray": [

{

"containerId": "0",

"generalMediaTypeArray": [

{

"termName": "Artifacts"

}

],

"specificMediaType": {

"termName": "Networked Storage"

}

}

],

"referenceUnitArray": [

{

"termName": "National Archives at College Park - Electronic Records"

}

]

}

],

"recordHistory": {

"created": {

"dateTime": "2017-08-25T10:15:25",

"userDisplayName": "describer1",

"userDn": "uid=RDTP1,ou=people,ou=dasUI"

},

},

"title": " Test Data",

"Description Type": Item,

}

}

]

}

**HTTP Status:** If the collection is empty (i.e., 0 items in response), instead of returning 404 Not Found, the API will return an empty array and collection metadata fields such as “total\_count”: 0. Invalid query parameter values, however, will return 400 Bad Request. Otherwise, 200 OK will be utilized for a successful response.

## Single Resource

A single resource is typically derived from the parent collection of resources. For instance, the National Archives IDentifier (NAID) will be used to identify one Description or an authority in the collection of all Descriptions and Authorities that exist in DAS today. Hence, the combined URI template with version, namespace, resource name and resource identifier will be */{version}/{namespace}/{resource}/{resource-identifier}*. Using this convention, the endpoint for retrieving a Description with NAID 330 will be */v1/entity/descriptions/330*.

### Read Operation

Example request for retrieving Description with NAID 330 will be

*GET/v1/entity/descriptions/330/*.

**HTTP Status:** If the provided resource identifier is not found, DAS API server will return a 404 Not Found HTTP status. Otherwise, 200 OK HTTP status will be returned when the record is found.

### Update Operation

The URI Template for updating a single resource will be

*PUT /v{version}/{namespace}/{resource}/{resource-identifier}*. A sample PUT request for updating Description with NAID 330 will be *PUT /v1/entity/descriptions/330/*. System-calculated fields, such as sub-fields of record history element, need not be included, as either term names or NAIDs can be provided for all Authority and parent links.

**HTTP Status:** Any failed request validation on a PUT request will be responded to with a 400 Bad Request HTTP status. If an update cannot happen because of some business rule, custom error code and message (in addition to the 400 HTTP status code) will be returned to the consumer, and these application-specific error codes and messages will be implemented as part of the full modernization when all the business rules for updating any resource will be implemented.

After a successful update, however, PUT operations will respond with 204 No Content status, with no response body. We will not be considering partial updates of resource now.

### Delete Operation

The URI Template for deleting a single resource will be

*DELETE /v{version}/{namespace}/{resource}/{resource-identifier}*. Example request for deleting Description with NAID 330 will be *DELETE /v1/entity/descriptions/330/*.

**HTTP Status:** DELETE is treated as idempotent, so it will always respond with a 204 No Content HTTP status. 404 Not Found HTTP status will not be utilized here. Instead, GET can be utilized to verify the affected resource exists prior to DELETE.

### Create Operation

To create a new resource, the following URI template will be used:

*POST /v{version}/{namespace}/{collection\_resource}/*.

For instance, to create a Description, the consumer will send the following request:

*POST /v1/entity/descriptions* with a request body. It will be like that of single resource update except that the request body will not have a record history element or any identifier for the Description resource to be created. Instead, it will only specify either term names or NAIDs for Authority and parent links. Subsequently, the DAS API server will produce an identifier for the resource. Once the POST has successfully completed, a new resource will be created. Hypermedia links will be included in the response body using rel: self to provide an easy way to get the URL of the newly created resource as shown below:

201 Created

{

"naId": 1223345,

"links": [

{

"href": "https://<server-name>/v1/entity/descriptions/1223345",

"rel": "self",

"method": "GET"

},

{

"href": " https://<server-name>/v1/entity/descriptions/1223345",

"rel": "delete",

"method": "DELETE"

}

]

"accessRestriction": {

"status": {

"termName": "Unrestricted"

}

},

"dataControlGroup": {

"groups": "RDEP",

"groupId": "ou=NWME,ou=groups"

},

"title": "Test Data",

},

"DescriptionType": Item,

}

## Sub-Resource Collection

When multiple identifiers are required to identify, a given resource, all behaviors of a collection resource are generally implemented as a subordinate of another resource. With respect to DAS, only Digital Objects can be modeled as a sub resource of the Description resource. The URI template for listing all the sub resources for a specific resource will be

*GET /v{version}/{namespace}/{resource}/{resource-identifier}/{sub-resource}/*, while the URI template for retrieving information about a specific sub-resource will be

*GET/{version}/{namespace}/{resource}/{resource-identifier}/{sub-resource}/{sub-resource-identifier}*.

Hence, example request for retrieving all the objects for Description with NAID 330 will be *GET /v1/entity/descriptions/330/objects/*, while the request for retrieving object with object identifier 1111 for Description with NAID 330 will be

*GET /v1/entity/descriptions/330/objects/1111*.

To add a new digital object to a specific Description resource, the following request will be used: *POST /v1/entity/descriptions/330/objects/*, while to update a specific object in a certain Description, the following request will be used:

*PUT /v1/entity/descriptions/330/objects/111*.

## Modeling Ingestion Operations

### Description

Initial ingestion request with a hyperlink containing information about the input file in the request body will be as follows: *POST /v1/ingest/descriptionJobs/*.

This request will create a Description Ingest Job and return with a 201 *Created* and a Job Identifier that can be used to query the status of the Job.

Next the consumer can query status of the Description Ingest Job as follows:

*GET /v1/ingest/descriptionJobs/{job identifier}/*.

When the job is completed, the HTTP response code will be 200 *OK* and the response body will contain a list of NAIDs and their Globally Unique IDentifiers (GUIDs) from the input file for Descriptions that were successfully created as well a list of GUIDs and corresponding errors that failed.

While the job is being processed, the API server will return a HTTP response code 202 *Accepted for processing* but the processing is not complete for the same API request. The response body will contain the status of the job to indicate whether it is submitted, in chunking mode, chunking is complete or processing is in progress.

### Authority

Authority Ingestion will be implemented for the full modernization work, but it will follow Description Ingestion closely in terms of API endpoints. In other words, initial ingestion request with a hyperlink containing information about the input file will be:

*POST /v1/ingest/authorityJobs/*, and a Job identifier will be returned in response. This Job identifier can be later used to query the status of the job using the following API endpoint:

*GET /v1/ingest/authorityJobs/{job Identifier} /*. When the Job is completed, the same request will return a list of NAIDs for Authorities that were successfully created and errors for the ones the application failed to create new records.

## Modeling Search Operations

In DAS, a user can not only execute an advanced or keyword search, but also save a search and filter criteria object. A NARA describer can even edit the saved search later. Hence, a search operation needs to be modeled not only as a resource to perform CRUD operations, but also to perform search and return Description or Authority results based on the entity the search request is based on. Modeling such operations is still in progress and, hence, the endpoints for the same will be included in the final version of this document.

### Description

Search operations we will model for Description search are:

* CRUD operations on the Description Search resource
* Perform keyword search
* Perform advanced search

### Authority

Search operations we will model for Authority search are:

* CRUD operations on the Authority Search resource
* Perform keyword search
* Perform advanced search

## Modeling Workflow Operations

### Description

Under-edit Description-related complex operations that we would design as part of follow-on Task Orders are as follows:

* create Task
* advance Task
* create Batch
* advance Batch
* approve
* get User Work Tray
* assign Task
* reassign Task
* assign Batch
* reassign Batch

### Authority

Authority proposal-related complex operations to be designed as part of the full modernization efforts:

* create Proposal
* reject Proposal
* resubmit Proposal
* approve Authority
* close Proposal Workflow
* get User Work Tray

## Modeling Reporting Operations

### Description

Description report-related complex operations to be designed later:

* Adhoc
* Canned

### Authority

Authority report related complex operations to be designed later:

* Canned

## Modeling Export Operations

### Description

Description Export-related complex operations to be designed later:

* XML Export
* Delta Export
* Bulk Search Export (Keyword and Advanced)

### Authority

Authority Export-related complex operations to be designed later:

* XML Export
* Delta Export
* Bulk Search Export (Keyword and Advanced)

## Modeling User Operations

For the modernized application, we intend to serve only authenticated API requests. For instance, if a NARA staff member wants to create a new Description in DAS, he would need to first authenticate against DAS using a POST request with a request body containing his username and the password as below:

{

“username”: “describer1”,

“password”: “$$$$$$”

}

A successful login response will contain a unique string token identifying the user for subsequent requests. This token will be valid for 4 hours with the token validity reset to 4 hours after each subsequent request to the API. Requests with invalid or missing tokens will result in a HTTP 401 – Unauthorized status response from the API. The format of the Request Header key/value pair representing the token will be:

*Header (key): Authorization*

*Value: “token <token>”* where <token> represents the response of a successful Login.

While the login web method may be one of the first one to be implemented as part of the full modernization effort, all the user, group and roles resources will be designed and implemented as part of the follow-on Task Orders and will support CRUD operations for each resource as well as the complex operations listed below:

* Associate User Role
* Associate User Role in Group
* Disassociate User Role
* Disassociate User Role in Group
* Get Role Users

## Implementation

We plan to use Spring boot for implementing the REST APIs discussed in the previous sections. Spring Boot comes with Jackson library out of-the-box which can parse JSON request body to Java objects. For instance, for any POST request on a simple resource such as an Ingest Job or Search object, it suffices to use the @RequestBody Spring model-view-controller (MVC) annotation to unmarshall the incoming JSON string into Java object and annotate the entity member elements with @JsonProperty with corresponding JSON field names. For CRUD requests on Descriptions and Authorities, if the POST request body is in XML format, we will use Java Architecture for XML Binding (JAXB) to convert the POST request body into plain old Java objects (POJOs) that we can use to validate against business rules before persisting into the database.

# Description Ingest Module Design

## Overview

Today, DAS supports bulk upload of XML files to create archival Description and to attach digital objects to existing Descriptions in batch mode. This is accomplished through the DAS Ingest module, which provides a web-based user interface that enables any DAS user with the DAS Import Manager role to upload such XML files for processing. XML records can be imported directly into the Authorized domain or the Import Manager can specify that the records be imported into the Under-edit domain. Once uploaded, XML records are then processed and validated against the DAS business rules by Windows scheduler services that run in the background. Records that meet the business rules are flagged for import into the Authorized domain. Records that fail to meet business rules are flagged for import into the Under-Edit domain. If a file contains records that are missing Data Control Group, Title, or Sequence number, those records are flagged as failed. However, the ingest module runs parallel to DAS UI communicating with the same data store as is indicated in 1.



Figure 1: Current DAS Ingestion Module Components

One of the primary goals of the DAS/NAC modernization efforts is to unify the external interfaces of the DAS application. In other words, as indicated in section 2, the same set of REST Web Services will be used by the UI client and the batch processing module.

We envision the new Description Batch Processing module to operate in the following manner (see Figure 2 for the full process flow):

* User posts to Description Ingest API endpoint with the S3 URL of a file containing, for example, 100,000 Descriptions (Step 1 in Figure 2).
* The Ingestion Web Service processes the incoming request and creates a Simple Queue Service (SQS) message for this request, and enqueues the same into the Job Queue for Description Bulk Ingestion (Step 2 in Figure 2).
* A Chunking service (scheduler task) monitors the job queue and once it sees the new message in this queue, dequeues and parses the message. This service downloads the file from S3, uses a XML parser to chunk the downloaded file into 100,000 SQS messages and enqueues each message into the Single Description queue meant for Description Bulk Ingestion (Steps 3, 4, 5 in Figure 2).
* A Processing service (scheduler task) dequeues one message at a time from the Description queue, and creates a POST request for the Description API endpoint in the application server so that a Description can be created either in the Approved or Under-edit domain based on the result of the validation in the processing service (Steps 6 and 7 in Figure 2).
* The Description Web Service saves the Description to Amazon Aurora and makes a PUT request to the AWS Elasticsearch Index API endpoint for Description search index to add the newly created Description so that it is available for search and access by end-users (Steps 8 and 9 in Figure 2).
* Throughout the lifecycle of an ingest job, the end-user can query the status of the Job he submitted using the Job Identifier he received when he sent the initial request. The API server will respond with the status of his Job and if the Job is completed, the response will include the list of NAIDs for the Descriptions that were successfully created and a list of errors for the ones that could not be created in the application.

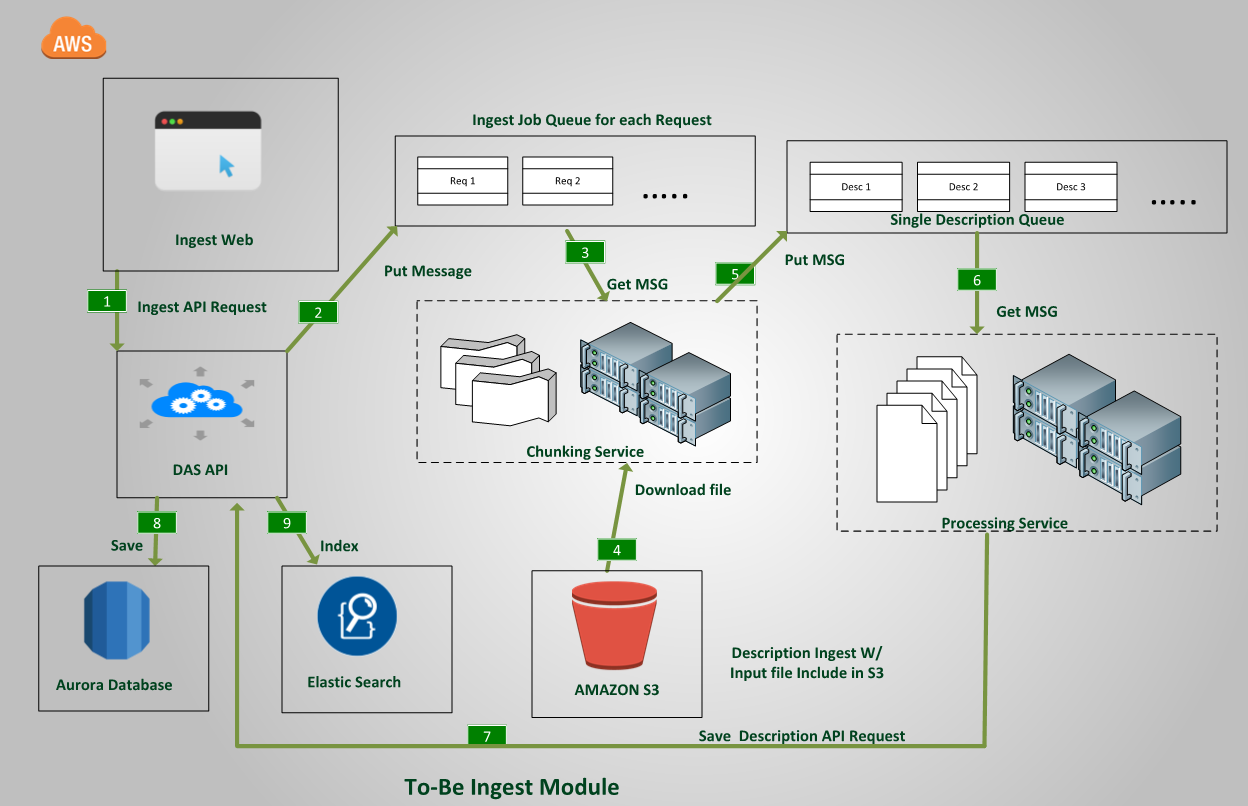


Figure 2: Future DAS Ingestion Module Components and their Integration

## Detailed Design

### Database Support

As the Ingest module is distinct from the primary data model in DAS, the database tables used to track the state of an Ingest Job and maintain relevant information throughout its lifecycle will be created in its own schema in the Aurora instance. The tables that will be used to support Description Ingest are as follows:

* Ingest\_job: Primary Job history table to track current job status, total number of incoming Descriptions, number of Descriptions that were successfully created, number of Descriptions that failed to create and the usual user related audit information for each Job Identifier.
* Job\_action\_types: Reference table that contains the name of all possible states an Ingest Job can be such as:

1. Submitted
2. Chunking in progress
3. Chunking complete
4. Processing complete

* Job\_action\_history: This table maintains the audit history for each state of an Ingest Job.
* Message\_status: The most crucial of all Ingest related tables to track the status of each SQS message that is enqueued into the Single Description Queue. It tracks the message ID of the SQS message, GUID, and the title of the Description to which the message corresponds. This way, if the Description failed to be created and no NAID (which is also tracked in the same table) can be generated for this title, the end-user can easily find out which Description XML from the input collection he needs to correct and send back to the ingest module as a new job. It also tracks the HTTP response code for the POST request that the Processing service makes for each SQS message it dequeues. Once processing is complete, these codes are used to calculate summary totals for successfully created Descriptions and error counts from the ingest\_job table.
* Failure\_types: Reference table for input file and chunking related error.
* Validation\_failure\_types: Reference table for various types of validation errors that may occur in the API server while trying to validate an incoming Description request.
* Message\_status\_types: Reference table for the various states in which an SQS message in the Single Description Queue can be.

**NOTE:** Full table definition for the ingest schema has been included in the Appendix A.1. of this document.

### Software Components

The entire Batch Processing module is highly decoupled and can be thought of being composed of REST API Web Services, three distinct background services and two queues.

The REST API service endpoints were discussed in Section 2.6.1

The two queues are:

* Job Queue: The API server and the chunking services interface with this queue.
* Single Description Queue: A FIFO queue. Both chunking and processing services interface with this queue. It is a FIFO queue to ensure that messages associated with a specific job are mostly processed either sequentially or in parallel and there is minimum interleaving of processing individual Description across multiple ingest jobs.

The three background services are:

* Chunking Service: Converts input XML file into single Description XMLs.
* Processing Service: Takes the single Description XML as input and creates a POST request for Description Web Services.
* Job Management Service: Responsible for managing the status and overall metrics for a job so that, once an ingest request is submitted, the API server can correctly respond to all GET requests on the job resource.

### Logical Flow

The steps below detail the logical flow for ingesting a batch file containing one or more Descriptions into DAS.

1. **API receives POST request to */v1/ingest/descriptionJobs/*.**  
   a. API server enqueues incoming request to Job Queue as a SQS message.

b. API server creates the Job status record in ingest\_job table setting *current\_action\_type\_id=SUBMITTED* (SUBMITTED is the id for the record in the job\_action\_types with action\_name=Submitted). It also creates a record in job\_action\_history table with the *action\_type\_id=SUBMITTED*.

c. It returns HTTP 201 response code and job identifier (referred to as job\_id in the steps below) to the client.

1. **Chunking service polls job queue and sees a message.**  
   a. Chunking service reads a message and immediately deletes it.

b. Chunking service downloads the message data from S3.

c. It updates the Job record with *current\_action\_id=CHUNKING\_IN\_PROGESS*.

d. It inserts a record into the job\_action\_history with the same action\_id.

e. It creates a new message using all the data from the message and adds the job id, sqs message id and other data.

1. **Chunking service finishes processing all the messages for the job.**a. It then updates total\_description\_count column in the job record with the count of the messages for the S3 job file. It also updates the current\_job\_action\_id to CHUNKING\_COMPLETE.

b. The Chunking service inserts a record to job\_action\_history with CHUNKING\_COMPELTE as the action\_type\_id.

1. **Processing Service polls the Single Description Queue and sees a message.**  
   a. The service reads a message and immediately deletes it.

b. It creates a payload for a POST request using the message data.

c. It adds the SQS message ID or other unique identifier for the message to the POST request.

d. The service sends the POST request to /v1/entity/descriptions/.

1. **Processing Service has sent a POST request and received a response.**  
   a. It upserts a record into the message\_status table with the job\_id, sqs\_message\_id, http\_post\_request\_time, http\_response\_time, http\_response\_code.
2. **The API receives a POST request to */v1/entity/descriptions/*.**

a. API validates the message.

b. If validation passes, it inserts the Description into Aurora RDS and indexes the same in AWS Elasticsearch service.

c. If the unique message identifier was supplied in the POST request (true when the consumer is Ingest Processing service and not DAS Web Client), it is used to insert new record into the message\_status table, setting insert\_success column=true.

1. **If the API received a POST request to */v1/entity/descriptions/* and validation fails:**

a. If a unique message identifier was supplied in the POST request, it is used to insert new record into the message\_status table, setting insert\_success column=false, insert\_failed=true. It also sets validation\_failure\_id to the appropriate id corresponding to the correct record in validation\_failure\_types.

1. **The Job Management Service periodically reads the job\_status table and finds a record where the *status=CHUNKING\_COMPLETE*, *failed=false* and (current time - *job\_creation\_time) > scan\_threshold* (amount of time after the job was created)**  
   a. It reads all the *message\_status* records for the job id.

b. It counts all the *insert\_success* and *insert\_fail* in the *message\_status* table for the job.

c. It updates the counts in the *ingest\_job* table.

d. If the count of success and fail equals the *total\_description\_count*, the *ingest\_job* record is updated and the *current\_status\_type\_id* is set to COMPLETED

e. If the count of success and fail equals the *total\_description\_count*, a record is inserted into *job\_action\_history* table with *action\_type\_id=COMPLETED*.

f. If the (current time - *job\_creation\_time)* > dead job threshold (amount of time after job creation at which the Job Management Service considers the job to be completed), the job status record is updated with *failed=true*.

### Exception Handling

The following only describes exception handling in the scheduler services. Validation failure is handled in the application server and, hence, briefly covered in the Logical Flow of Description Ingestion section:

1. **Chunking reads message and S3 file not found.** Job status record in ingest\_job table is updated with *failed=true*, *reason\_failed\_id=FILE\_NOT\_FOUND*
2. **Chunking reads message, downloads its S3 file and xml can't be parsed.** Job status record in ingest\_job table is updated with *failed=true*, *reason\_failed\_id=INVALID\_XML\_*FILE
3. **Chunking reads message and encounters issue with xml, like bad tags, bad character data.** Job status record in ingest\_job table is updated with *failed=true*, *reason\_failed\_id=INVALID\_XML\_DURING\_CHUNKING*
4. **Chunking successfully reads message, acquires the job id and then encounters an exception.**

Chunking service updates the job record in ingest\_job table with failed=true, failure\_type\_id=CHUNKING\_UNKNOWN\_EXCEPTION

1. **Processing encounters timeout during POST to API.**

Upserts a record into the *message\_status* table with the *job\_id*, *sqs\_message\_id*, *http\_post\_request\_time*, *http\_response\_time*, *http\_response\_code* and *http\_timeout=true*.

# Search Engine Design

## Overview

Search is central to the Catalog as it is essentially a public search interface for browsing both catalog information and online content. The current architecture of NAC is scalable wherein scalability is achieved by splitting the content into multiple pieces called partitions or shards and resilient by means of replica. However, the current architecture uses proprietary libraries such as the Search Technologies Aspire Content Processing System to ingest records from DAS, and a proprietary Solr plugin library called Query Processing Language (QPL) to make the records searchable and accessible by the public. Additionally, there is significant overhead with managing the Solr cluster deployed on Amazon EC2 instances, especially with the use of Apache Zookeeper to provide centralized management for all search servers in the cluster. In the current architecture, there is one search index that not only stores full Description and Authority metadata records from DAS but also NAC specific annotations such as tags, comments and transcriptions.

Search is also one of the primary capabilities in DAS. It is essential for NARA describers to search existing content so that they can create new descriptions and authorities that are in some way using the existing content in the form of Authority and parent description links. For instance, the Description service allows any user that has been provisioned within DAS to perform primary searches on Approved Descriptions, Under-edit Descriptions or both. The default search is a basic keyword search. An advanced search, on the other hand, includes a user-entered search clause with operators such as wildcards, Boolean Operators, Proximity Operators, exact phrase, or nesting. Additionally, in the advanced search mode, the query-builder option allows a user to stack clauses searching for specific terms across one or more fields using Boolean operators AND, OR, or NOT. Unfortunately, in the current implementation all text searches on Description data in both Approved and Under-edit space as well as proximity and stem searches on Description and Authority data are performed by the Web Services by directly executing SQL queries against the Oracle XML database. This does not scale very well. In fact, current DAS business owners must conduct new user training for the internal NARA staff responsible for creating content in DAS to limit the extent of keyword or text searches they perform to create new Descriptions and Authorities.

One of the primary goals of the DAS/NAC modernization efforts is to unify the search engine for DAS and NAC and eliminate the need for weekly exports from DAS to be ingested into NAC. We envision achieving this goal by using separate indexes for DAS specific entities such as Description, Authorities, User Workflows and NAC specific entities such as Tags, Transcriptions and Comments. Furthermore, Description and Authority metadata will only be stored in DAS instead of being duplicated in DAS database and NAC search engine and NAC will use the DAS Description and Authority Web Services to access records by their NAID.

## Search Index Design

We envision having the following separate indexes in the combined search cluster. See Figure 3:

1. Under-edit Description
2. Approved Description
3. Under-edit Authority
4. Approved Authority
5. DAS Workflow
6. Digital Object
7. NAC Annotation
8. Web Pages



Figure 3: DAS NAC Combined Search Index

Each application will use a subset of these indexes to search and access relevant content. For instance, if the end-user is performing an advanced search for just Descriptions using filters such as created date and Data Control Group and so on, only Approved Description, Under-edit Description and DAS workflow search indexes will be used. If, on the other hand, he is executing a basic Description keyword search, both Approved and Under-edit Description, Approved Authority, Digital Object as well as DAS workflow search indexes will be used as shown in Figure 4.

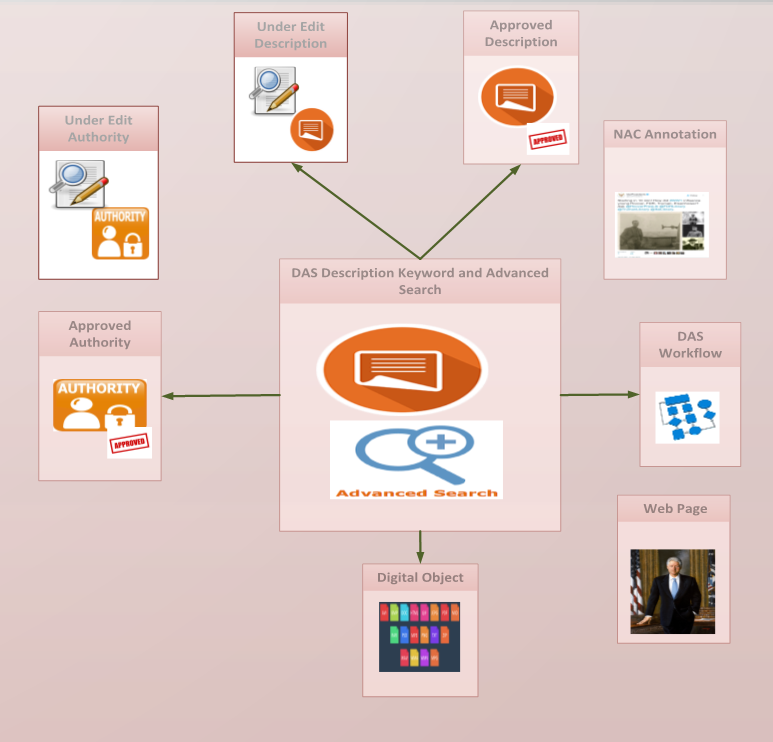


Figure 4: DAS Description Keyword and Advanced Search

Similarly, a basic Authority keyword search will use only the Approved Authority, the Under-edit Authority and the DAS workflow search indexes as shown in Figure 5.

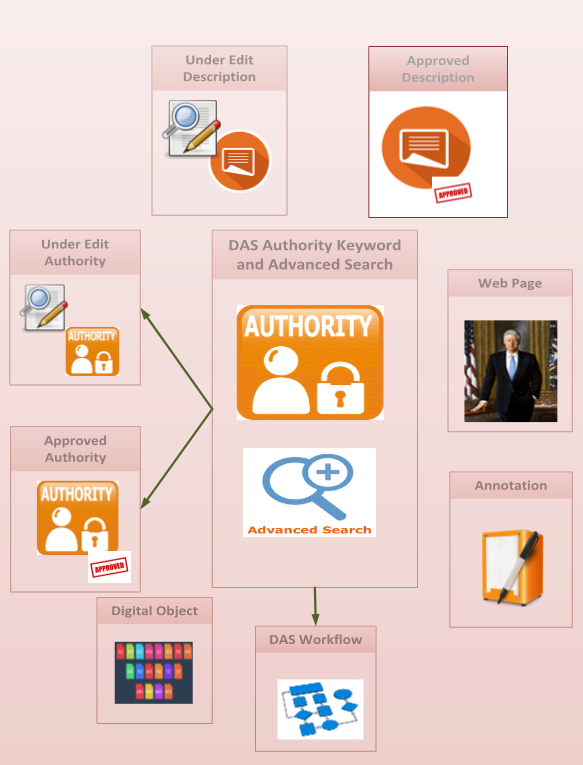


Figure 5: DAS Authority Keyword and Advanced Search

On the contrary, any text searches from NAC will use all indexes except for the Under-edit Description, the Under-edit Authority and the DAS workflow search indexes as shown in Figure 6.

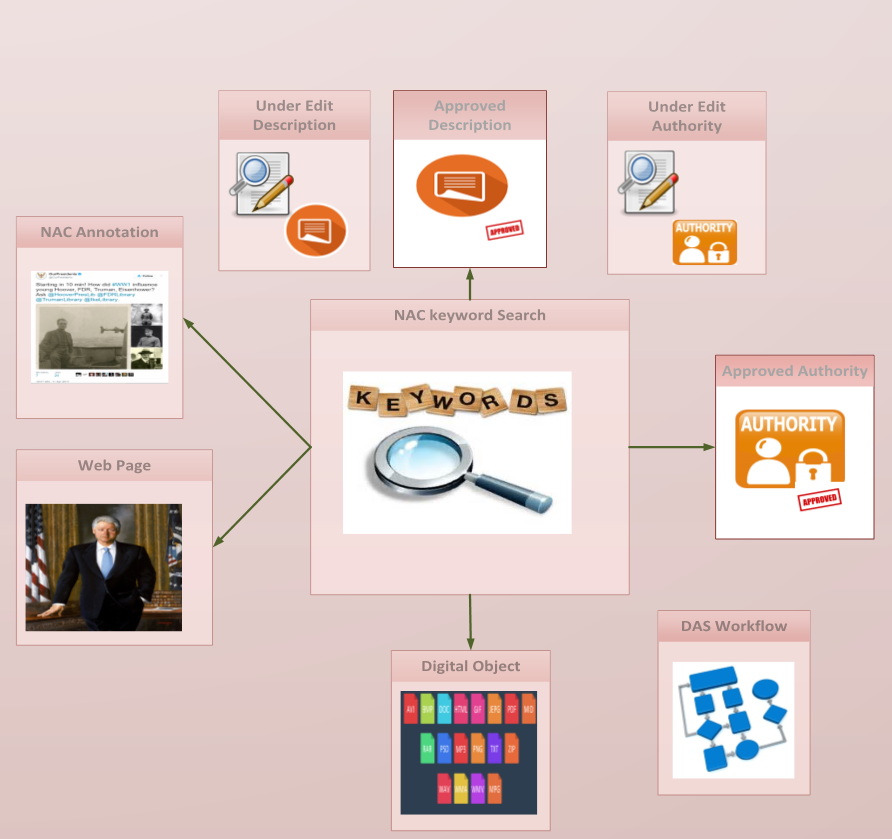


Figure 6: NAC Keyword Search

## Search Schema Design

We have not designed a combined search schema that supports the logical search index partitioning as illustrated in the previous section. This will be worked on in the second phase of this Task Order.

## Query Builder Design

It will follow search schema design.

## Data Migration

We plan to use Amazon Elasticsearch Service for the physical implementation of the NAC/DAS combined search engine. Because it is a managed service, Amazon Elasticsearch Service automatically detects and replaces failed Elasticsearch nodes, reducing the overhead associated with self-managed infrastructure that we have today with Apache Solr Cloud in NAC. Amazon Elasticsearch Service will also allow us to easily scale our cluster via a single API call or a few clicks in the console, unlike having to provision new EC2 instances for new shards and replicas. Furthermore, Amazon Elasticsearch Service has tight integration with Amazon Identity and Access Management (IAM) for security, Amazon CloudWatch for monitoring and Amazon CloudTrail for auditing as shown in Figure 7.

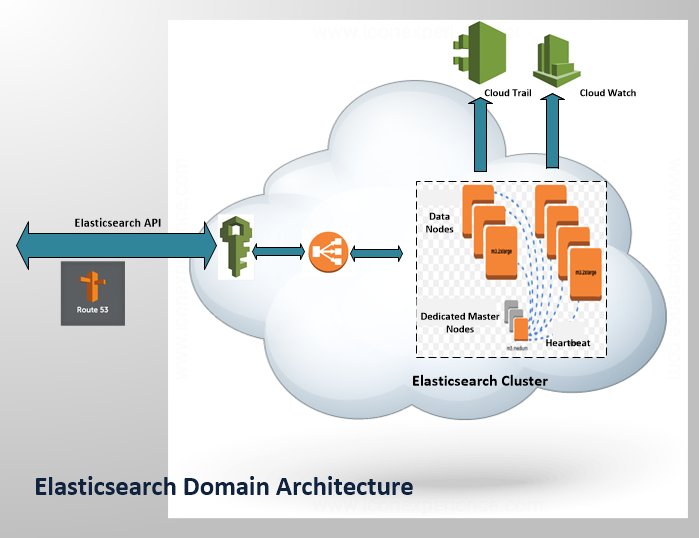


Figure 7: AWS Elasticsearch Resource Domain Architecture

### ETL Design

To be implemented once search schema is designed.

## Search Access Layer Design and Implementation

To be implemented once search schema is designed.

# Database Design

## Overview

The basic utilization of the DAS system pertains to querying and obtaining information to create new information. These capabilities depend on the organization of the data in the database. The current DAS database design ensures the following:

* Flexibility in the design to adapt to the needs of different users.
* Controlled and standardized approach to input data and update data.
* System of validation checks to maintain data integrity and consistency.

Data related to both Description and Authorities is stored in the Oracle 12c XML database. Tables in the DAS database are grouped together in separate schemas depending on their functionality such as bulk ingestion, workflow, auditing and so on. The primary schema is called das\_desc schema. This schema holds all the Under-edit and Approved Authorities and Descriptions along with the other stored procedures and indexes required to support the Description and Authority Web Services. It uses Oracle XML Database (XMLDB) and persists Description and Authority data in native XML format. Description and Authority data are stored across multiple XML Type tables with selected attributes like NAID, Description Type (Authority Type for Authorities), Title (Term Name for Authorities), Created Date, Created User Name and so on extracted as separate columns during the pre-processing stage to facilitate faster processing while serving end-user queries.

Oracle XML DB has all the advantages of relational database technology plus the advantages of XML. It provides high-performance, native XML storage and retrieval technology. It has standard access methods for navigating and querying XML. Oracle 12c XML Database offers a complete and efficient content management solution that is a viable platform for meeting DAS storage and search requirements up to a certain data volume. Traditional relational databases seek to achieve a design pattern called “normalization.” This pattern breaks up database content and arranges it within the database structure in such a way that the same content is never repeated. On the other hand, XML databases store data in XML as single entities within the database. With XML databases, the concern is not so much eliminating repetition of content, but rather the emphasis is on keeping the content together. This method of storage allows for the easy implementation of some search scenarios, like proximity, and full text searches. Primary tables with such a hybrid model are listed below and their relationship is shown in Figure 8:

* Description: Contains metadata for Approved Descriptions and partitioned based on NAID range for efficient record retrieval.
* Desc\_Assc: Contains Authority and parent references for Approved Descriptions and partitioned based on NAID range for efficient record retrieval.
* Authority: Contains metadata for Approved Authorities; partitioned based on NAID range for efficient record retrieval.

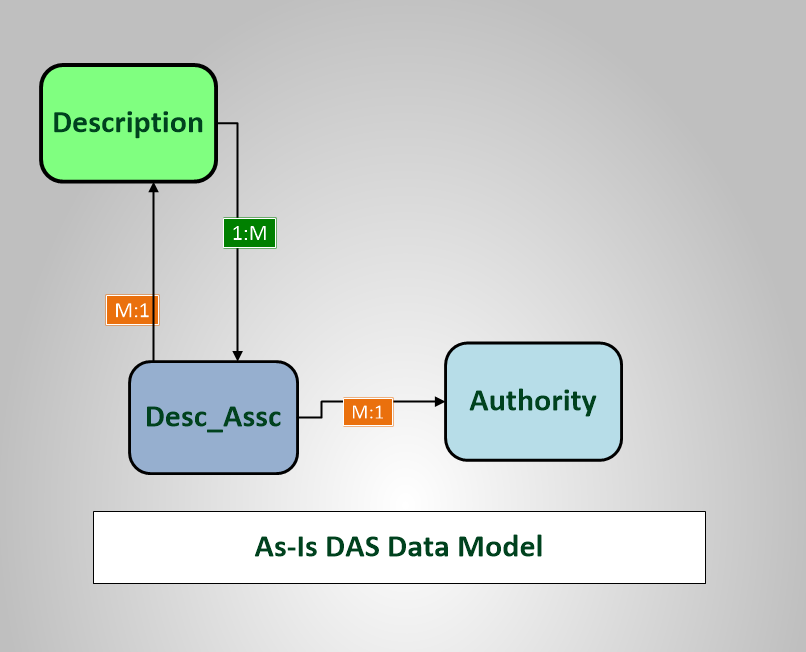


Figure 8: As-Is Hybrid Data Model

However, as indicated in Section 1, with the sharp increase in the size of the database, the performance degradation, especially for full text searches and the regular maintenance overhead required to keep the read and write response times at a user-agreeable level far surpasses the benefits of having an XML database with a flexible schema definition and its ability to perform context based searches. Hence, as part of the modernization efforts, the goal is to move to a fully relational data model with a fixed schema. Since DAS went into production with Oracle XML database in May 2014, schema changes to core Description and Authority tables have been few and far between. On the other hand, repartitioning required frequent table index changes. This was required to maintain the search ability as data grew.

## Conceptual Model

We envision the new DAS data model to be comprised of three top-level tables: Description, Primary\_Authority and Authority\_List.

### Authority Model

Authority\_List table will contain all the information pertaining to Authority List values, such as Color, Base, Access Restriction, Finding Aid URL and so on. Primary\_Authority, on the other hand, will contain all the information pertaining to six primary types of Authority: Person, Organization, Geographic Place Name, Topical Subject, Program Area and Specific Records Type. Similarly, Description table will contain all the information pertaining to all seven types of Descriptions: Collection, Record Group, Series, File Unit, Item, Item AV and Preliminary Description.

Figure 9 demonstrates the relationship between Primary\_Authority and Authority\_List tables. Each of the above-mentioned Primary Authority references a Record Source and a Reference Unit Authorities. However, no crosswalk tables are needed for Authority List types Reference Unit or Record Source, because any instance of a Primary Authority can have only one reference of record source or reference unit and can directly reference Authority\_List table for those records. Similarly, each of these Authorities can be linked to one or more Special Project Authorities. Consequently, we need to use a crosswalk table for Special Project linking Primary Authorities with Authority List of type Special Project.

The conceptual model below also has number of crosswalk tables to describe the relationship among Primary Authorities. For instance, it uses the See Also table to link a Person Authority to a collection of Person Authorities, the Jurisdiction table to link an Organization to a collection of Geographic Place Name, the Organization Name Reference table to link an Organization to its Organization Name children, the Organizational Reference table to link a Person, Geographic Place Name, Program Area, Topical Subject or Specific Records Type to a collection of Organization Authorities, the Predecessor Successor table to establish links between a specific Organization Name and a collection of Organization Names that will either serve as a Predecessor or a Successor, the Personal Reference table to link an Organization to a collection of Person Authorities, the Program Area Reference table to link an Organization to a collection of Program Area Authorities and finally, the Broader Narrower Related Term table to link a Geographical Place Name, Program Area, Specific Records Type or a Topical Subject to a collection of Primary Authorities of the same type that will either serve as a Broader Term, Narrower Term or a Related Term. The Use For table is used to link preferred Geographic Place Name, Program Area, Specific Records Type or a Topical Subject Authority with a collection of its non-preferred variant.

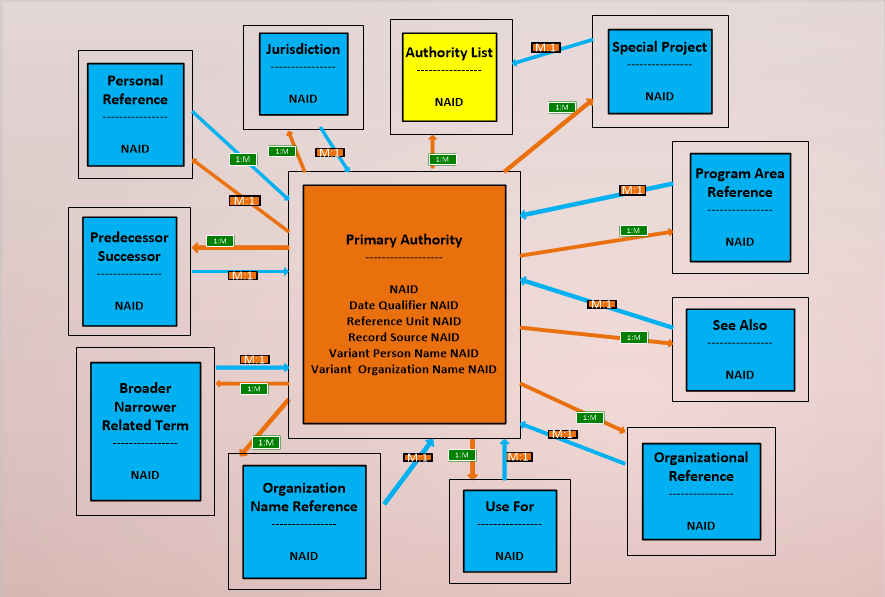


Figure 9: DAS To-Be Fully Normalized Data Model for Authority

### Description Model

Figure 10 demonstrates the relationship among the Description, Primary\_Authority and Authority\_List tables using several crosswalk tables. These tables contain not only the Description and the Authority references, but also some Description-specific metadata. For instance, Access Restriction crosswalk table stores Access Restriction note information, in addition to storing Description and Authority NAIDs. The Description table is the central table in this model and it not only contains attributes that are common across all seven types of Descriptions such as NAID, Title, Data Control Group, Staff-only note, Record History but also Description type specific attributes such as Record Group Number (applicable to only Record Group Descriptions), Production Date (applicable to only Item and Item AV Descriptions), Collection Identifier (applicable to only Collection Descriptions). Additionally, it stores references to Authority List types such as Begin Congress, End Congress as well as all the Date Qualifiers associated with Inclusive and Coverage Dates.

With respect to the crosswalk tables shown in Figure 10, not all are applicable to all types of Description. For instance, Finding Aid applies to all types of Description and it stores all Finding Aid related information that a Description may contain. Note that Finding Aid table has three Authority List references viz. Finding Aid Type, Finding aid URL and Object Type.

Series Former Parent table, on the other hand, is only relevant for Series Description. While we need not track Former Parents for any Description below the Series level, there is a business requirement to track Former Parents for Series Descriptions and, hence, Series Former Parent is used to manage the references to Former Collection and Record Group Parents. Similarly, Creating Organization and Individual table is relevant only for Series Descriptions.

On the contrary, Variant Control Number, Digital Object, Access Restriction, Use Restriction, General Records Type, Organizational and Personal Contributor, Special Project, Microform Publication, Physical Occurrence, Primary Authority Reference and Donor and Language cross walk tables are relevant only for Series, File Unit, Item and Item AV Descriptions. Some of these tables such as Primary Authority Reference and Donor as well as Organizational and Personal Contributor contain multiple types of references and have a discriminator column to indicate the type of reference.

Physical Occurrence has its own set of child tables such as Media Occurrence, Holding Measurement, Reference Unit and Location. Media Occurrence, in turn, uses crosswalk tables such as General Media Type and Other Preservation Characteristics to represent a collection of General Media Type Authority List references or Other Preservation Characteristic references, a Description may have in an instance of Media Occurrence.

**Note**: In Figure 10 even though Authority List appears thrice to minimize crisscrossing one-to-one, many-to-one, one-to-many relationship arrows, there is only one instance of Authority List table being referenced in the Description data model.

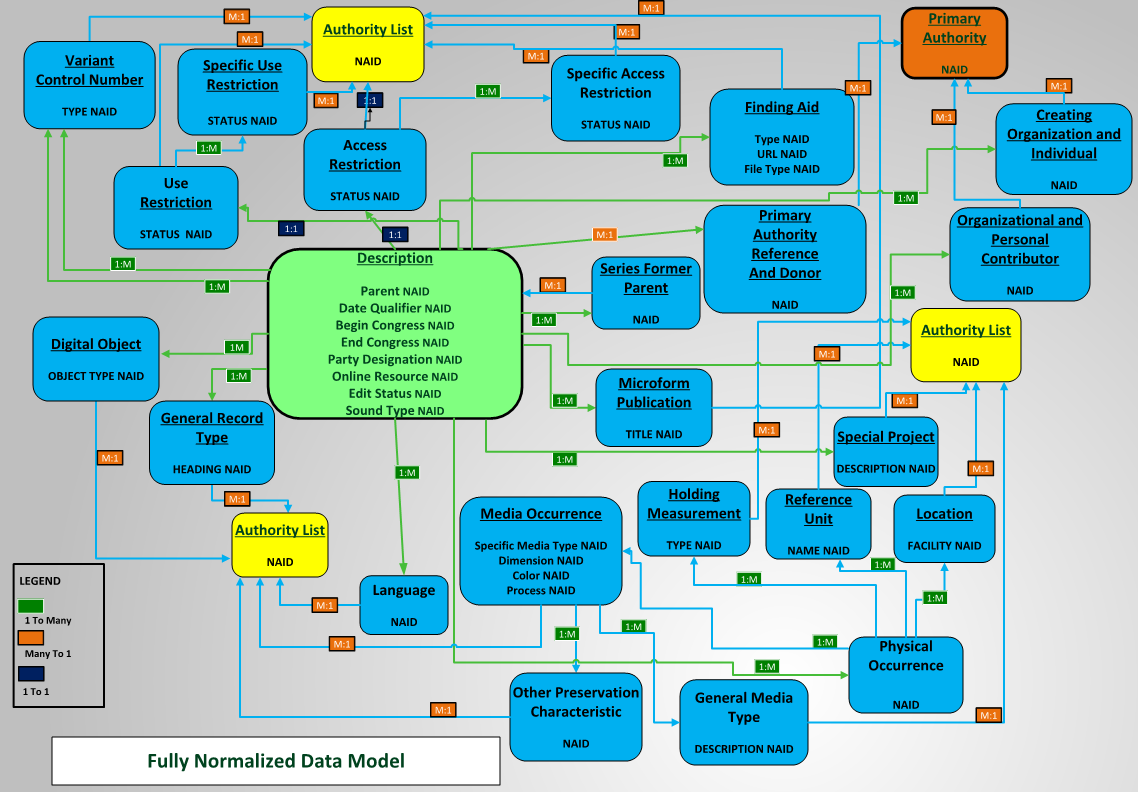


Figure 10: DAS To-Be Fully Normalized Data Model for Description

## Physical Model

### Authority Model

See Table 2 for a list of the columns that will be included in the PostgreSQL implementation of the Authority\_List table.

| Table : authority\_list Table Columns | |
| --- | --- |
| **Column Name** | **Column Type** |
| auth\_list\_naid | bigint |
| term\_name | text |
| auth\_type | text |
| auth\_details | jsonb |
| created\_date | timestamp |
| brought\_under\_edit\_history | json |
| changed\_history | json |
| approval\_history | json |
| last\_brought\_under\_edit | timestamp |
| last\_approved\_date | timestamp |
| last\_changed\_date | timestamp |
| imported\_date | timestamp |

See Table 3 for a list of the columns that will be included in the PostgreSQL implementation of the Primary\_Authority table.

| Table : primary\_authority Table Columns | |
| --- | --- |
| **Column Name** | **Column Type** |
| prim\_auth\_naid | bigint |
| name\_heading | text |
| full\_name | text |
| numerator | text |
| personal\_title | text |
| birth\_date\_qualifier\_naid | bigint |
| death\_date\_qualifier\_naid | bigint |
| birth\_date | jsonb |
| biographical\_note | text |
| source\_note | text [] |
| proposer\_name | text |
| proposal\_date | timestamp |
| reference\_unit\_naid | bigint |
| record\_source\_naid | bigint |
| naco\_submitted | Boolean |
| import\_rec\_ctl\_no | text |
| notes | text |
| scope\_note | text |
| saco\_submitted | Boolean |
| lat\_long | real |
| admin\_hist\_note | text |
| establish\_date | jsonb |
| abolish\_date | jsonb |
| establish\_date\_qualifier\_naid | bigint |
| abolish\_date\_qualifier\_naid | bigint |
| death\_date json | jsonb |
| term\_name | text |
| auth\_type | text |
| auth\_details | jsonb |
| created\_date | timestamp |
| Imported\_date | timestamp |
| brought\_under\_edit\_history | json |
| changed\_history | json |
| approval\_history | json |
| last\_brought\_under\_edit | timestamp |
| last\_approved\_date | timestamp |
| last\_changed\_date | timestamp |

As explained in the conceptual model section, a primary authority can be linked to one or more primary authorities, and its corresponding physical model is represented in Figure 11.

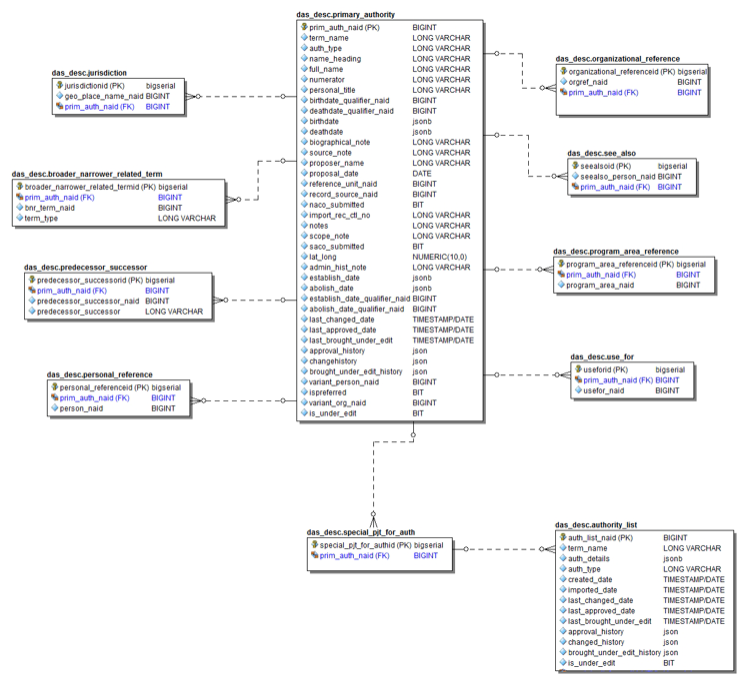


Figure 11: Primary Authority Physical Data Model

### Description Model

The physical model for description is a bit more involved than that for authority. The primary table is called Description and the child tables with an identifying relationship with Description are listed in Table 4.

| Table : Child Tables |
| --- |
| **Child Tables** |
| access\_restriction |
| creating\_org\_indv |
| Digital\_object |
| finding\_aid |
| general\_media\_type |
| general\_record\_type |
| holding\_measurment |
| language |
| location |
| media\_occurence |
| microform\_pub |
| org\_personal\_contributor |
| other\_preservation\_characteristic |
| pers\_org\_topsub\_geo\_sppr\_ref |
| physical\_occurence |
| reference\_unit |
| series\_former\_parent |
| special\_pjt\_for\_desc |
| specific\_access\_restriction |
| specific\_use\_restriction |
| use\_restriction |
| variant\_control\_number |

As the full physical model is too large to include as a graphic in this document, Figure 12 describes only the relationship between the Description table and Physical Occurrence related child tables.

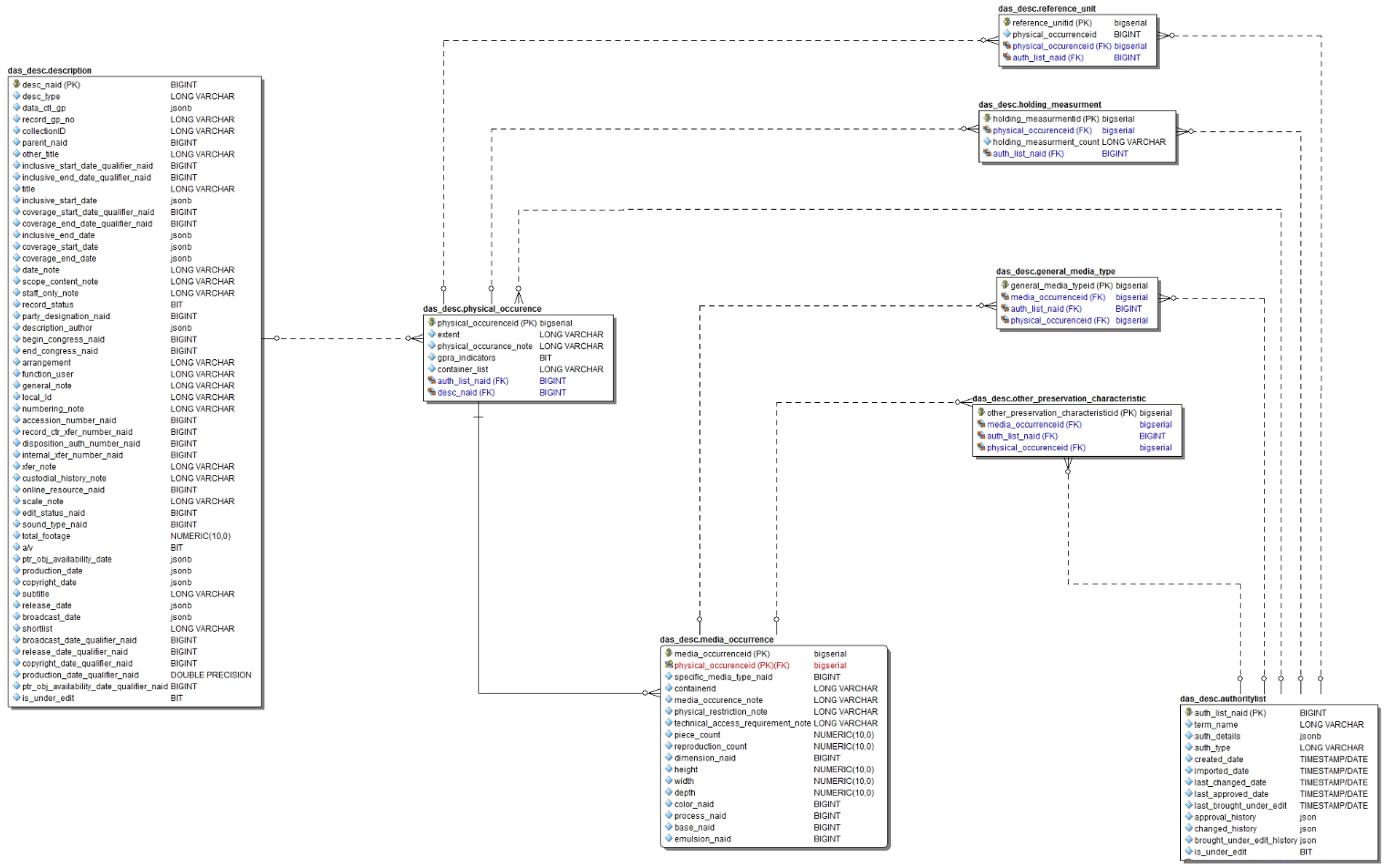


Figure 12: Relationship among Description and Physical Occurrence Tables

A full physical model for the Description table can be found as part of the Data Dictionary in Appendix A.2. Note that not all tables are used for all types of Descriptions. For instance, a Record Group cannot have any Digital Object. Hence, a set of insert SQL statements for Record Group persistence will not include any inserts to the Digital\_Object table. A sample select query for a Record Group Description using this physical model is described below:

select d.desc\_naid, desc\_type, data\_ctl\_gp, record\_gp\_no, other\_title, date\_note, scope\_content\_note, title, d.is\_under\_edit,

begin\_congress\_naid, bcAL.term\_name,

end\_congress\_naid, ecAL.term\_name,

coverage\_start\_date, coverage\_start\_date\_qualifier\_naid, covSDAL.term\_name,

coverage\_end\_date, coverage\_end\_date\_qualifier\_naid, covEDAL.term\_name,

FA.finding\_aid\_Note, FA.finding\_aid\_source, FA.finding\_aid\_type\_naid, fatAL.term\_name, FA.finding\_aid\_url\_naid, fauAL.term\_name, FA.object\_type\_naid, faotAL.term\_name,

inclusive\_start\_date, inclusive\_start\_date\_qualifier\_naid, inclSDAL.term\_name,

inclusive\_end\_date, inclusive\_end\_date\_qualifier\_naid, inclEDAL.term\_name,

SP\_desc.auth\_list\_naid, spAL.term\_name

from description d

LEFT OUTER JOIN authoritylist bcAL ON (d.begin\_congress\_naid = bcAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist ecAL ON (d.end\_congress\_naid = ecAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist covSDAL ON (d.coverage\_start\_date\_qualifier\_naid = covSDAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist covEDAL ON (d.coverage\_end\_date\_qualifier\_naid = covEDAL.auth\_list\_naid)

LEFT OUTER JOIN finding\_Aid FA ON (d.desc\_naid = FA.desc\_naid)

LEFT OUTER JOIN authoritylist fatAL ON (FA.finding\_aid\_type\_naid = fatAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist fauAL ON (FA.finding\_aid\_url\_naid = fauAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist faotAL ON (FA.object\_type\_naid = faotAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist inclSDAL ON (d.inclusive\_start\_date\_qualifier\_naid = inclSDAL.auth\_list\_naid)

LEFT OUTER JOIN authoritylist inclEDAL ON (d.inclusive\_end\_date\_qualifier\_naid = inclEDAL.auth\_list\_naid)

LEFT OUTER JOIN special\_pjt\_for\_desc SP\_desc ON (d.desc\_naid = SP\_desc.desc\_naid)

LEFT OUTER JOIN authoritylist spAL ON (SP\_desc.auth\_list\_naid = spAL.auth\_list\_naid)

## Data Migration from Oracle on EC2 to Amazon PostgreSQL Aurora RDS

We envision using PostgreSQL-compatible Amazon Aurora database service as our physical data store for DAS. Like Amazon Elasticsearch service, Amazon Relational Database Service (RDS) is a fully managed database service in the Amazon Cloud that is easy to setup, operate and scale. It automates common database administration tasks such as hardware provisioning, database setup, patching and backups. Amazon RDS is available on several database instance types - optimized for memory, performance or I/O as well as six distinct database engines. For this modernization effort, we will use PostgreSQL compatible Amazon Aurora DB engine that combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of the open source PostgreSQL database. It is built on a fully distributed and self-healing storage system and provides enterprise-level capabilities including database monitoring, database cloning, cross-region copying and replication as well as integration with common AWS services such as Identity and Access Management, CloudWatch and so on. In other words, an Aurora instance corresponds to a DB cluster that consists of one or more instances, and a cluster volume that manages the data for those instances. An Aurora *cluster volume* is a virtual database storage volume that spans multiple Availability Zones, with each Availability Zone having a copy of the cluster data as shown in Figure 13.

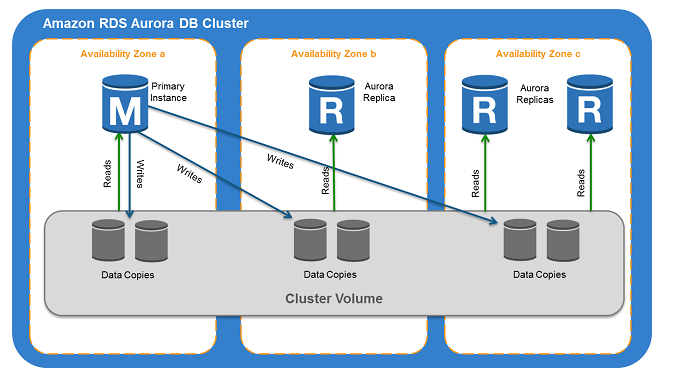


Figure 13: Amazon RDS Aurora DB Cluster

### ETL Design

We must migrate Authority Lists and Primary Authorities before we can migrate Descriptions via reference application. Our goal for this Task Order is to only migrate Record Groups and Series Descriptions from Oracle to Aurora instance for projecting storage usage in the target database.

For creating Record Groups, we will need to migrate the following Authorities:

1. Begin Congress
2. End Congress
3. Date Qualifier
4. Finding Aid File Type
5. Object Type
6. Finding Aid URL
7. Special Project

The above set of Authorities are of type Authority List. Authority\_List table in Aurora is a parent table with no dependency on either Primary Authority or Description tables and, hence, can be migrated independent of Primary\_Authority and Description tables. Authority\_List uses a PostgreSQL specific data type called jsonb for the auth\_details column to store all the specific information about that Authority type in a key-value format. For instance, when the Authority type is Begin Congress, the auth\_details column will have the following keys and sample values:

{

 "beginCongressValue":"114",  
   "startDate":{    
      "day":6,  
      "month":1,  
      "year":2015,  
      "logicalDate":"06-JAN-15"  
   },  
   "endDate":{    
      "day":3,  
      "month":1,  
      "year":2017,  
      "logicalDate":"03-JAN-17"  
   }

}

While when the Authority type is End Congress, the auth\_details column will have the following keys and sample values:

{

 "endCongressValue":"114",  
   "startDate":{    
      "day":6,  
      "month":1,  
      "year":2015,  
      "logicalDate":"06-JAN-15"  
   },  
   "endDate":{    
      "day":3,  
      "month":1,  
      "year":2017,  
      "logicalDate":"03-JAN-17"  
   }

}

And when the Authority type is Accession Number, the auth\_details column will have the following keys and sample values:

{

 "number":"114"

}

We will use Oracle XQuery to extract such fields from the xml\_data column in the Authority table in Oracle and then use PL-JSON, an open source generic JSON object written in PL/SQL to populate the auth\_details column in Authority\_List table in Aurora.

See sample ETL script in Appendix A.3.

## Database Access Layer Design and Implementation

We plan to use Java Database Interface (JDBI), an open source SQL convenience library for Java instead of just Java Database Connectivity API (popularly known as JDBC framework) for implementing the database access layer in the reference application. JDBI is an abstraction layer on top of JDBC much like Spring JDBCTemplate but works at a lower level than Hibernate or JPA. Furthermore, it shares most if not all the functionality JDBCTemplate provides (e.g. named and indexed paramaters). It supports a SQL Object API that simplifies the common idiom of creating Data Access Object (DAO) where a single method maps to a single statement. An SQL object definition is an annotated interface from which we can obtain an instance of the DBI instance and then call methods on it.

For the reference application, we will define SQL object definition for POJOs whose member elements will closely follow the table attributes on the corresponding database objects. The detailed design of such POJOs and DAOs have not been finalized yet and, hence, they will be included in the final version of this document.

1. Appendix: DAS Data Model
2. Description Ingest Database Table

Description ingest database table details are listed in Table 5.

| Table : Description Ingest Database Table | | | |
| --- | --- | --- | --- |
| **Table** | **Column Name** | **Type** | **Comment** |
| Ingest\_job | job\_id | integer |  |
| total\_description\_count | integer | set by Chunking Service |
| completed | bool | set by JobProcessManager Service |
| success\_count | integer | set by JobProcessManager Service |
| failure\_count | integer | set by JobProcessManager Service |
| failed | bool | set by JobProcessManager to true when current time minus job\_creation\_time > job obsolete threshold |
| failure\_type\_id | integer | fk to failure\_types |
| chunking\_error\_id | text | the guid generated and logged when an error occurs in Chunking |
| last\_chunked\_guid | UIUD | the guid of the last message chunked. This is set only if an XML error occurs while chunking messages |
| job\_creation\_time | timestamp |  |
| job\_created\_user | text |  |
| current\_action\_type\_id | integer | corresponds to action\_type\_id in the job\_actions table |
| last\_update\_date | timestamp |  |
| job\_action\_types | action\_type\_id | integer | unique id |
| action\_type | text | one of  submitted,  chunking in progress,  chunking completed,  processing completed |
| job\_action\_history | id | bigserial |  |
| job\_id | integer |  |
| action\_type\_id | integer | Foreign key to job\_action\_types |
| action\_time | timestamp |  |
|  |  |  |
| failure\_types | failure\_type\_id | integer | unique id |
| failure\_type | text | one of  s3 file not found,  invalid xml file for job,  invalid xml while chunking,  unknown exeption in chunking |
| Validation\_failure\_types | failure\_type\_id | integer | unique id |
| failure\_type | text | TBD |
| Message\_status\_types | Message\_status\_type\_id | integer | unique id |
| Message\_status\_type | text | TBD |
| message\_status | sqs\_message\_id | GUID |  |
| title | text |  |
| job\_id | integer |  |
| message\_status\_type\_id | integer | Foreign key to message\_status\_types |
| insert\_success | bool | set by API to true on success |
| insert\_failed | bool | set by API to true if insert fails |
| validation\_failure\_id | integer | Foreing key to validation\_failure\_types. This is set by API |
| http\_post\_request\_time | timestamp | the time the POST request was sent to API. This is set by Processing Service |
| http\_response\_time | timestamp | the time the API responded to the insertPOST request. This is set by the Processing Service |
| http\_response\_code |  | The http response code is received by Processing Service which sets it here |
| http\_timeout | bool | set to true by Processing Service if http timeout occurs |

1. Description and Authority Data Dictionary

Click on the following embedded image to see the fully-relational DAS data model:



All supporting table definition for authorities and descriptions for the full physical model has been listed in the following table:

|  |
| --- |
| Table : Description and Authority Data Dictionary |

All supporting table definition for Authorities and Descriptions for the above physical model has been listed in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Column Name** | **Column Type** | **Precision** |
| access\_restriction | access\_restrictionid | bigserial | 19 |
| access\_restriction | desc\_naid | int8 | 19 |
| access\_restriction | auth\_list\_naid | int8 | 19 |
| access\_restriction | access\_restriction\_note | text | 2147483647 |
| authoritylist | auth\_list\_naid | int8 | 19 |
| authoritylist | term\_name | text | 2147483647 |
| authoritylist | auth\_details | jsonb | 2147483647 |
| authoritylist | auth\_type | text | 2147483647 |
| authoritylist | created\_date | timestamp | 29 |
| authoritylist | imported\_date | timestamp | 29 |
| authoritylist | last\_changed\_date | timestamp | 29 |
| authoritylist | last\_approved\_date | timestamp | 29 |
| authoritylist | last\_brought\_under\_edit | timestamptz | 35 |
| authoritylist | approval\_history | json | 2147483647 |
| authoritylist | changed\_history | json | 2147483647 |
| authoritylist | brought\_under\_edit\_history | json | 2147483647 |
| authoritylist | is\_under\_edit | bool | 1 |
| broader\_narrower\_related\_term | broader\_narrower\_related\_termid | bigserial | 19 |
| broader\_narrower\_related\_term | prim\_auth\_naid | int8 | 19 |
| broader\_narrower\_related\_term | bnr\_term\_naid | int8 | 19 |
| broader\_narrower\_related\_term | term\_type | text | 2147483647 |
| creating\_org\_indv | creating\_org\_indvid | bigserial | 19 |
| creating\_org\_indv | desc\_naid | int8 | 19 |
| creating\_org\_indv | prim\_auth\_naid | int8 | 19 |
| creating\_org\_indv | auth\_list\_naid | int8 | 19 |
| creating\_org\_indv | creating\_ind\_creating\_org | bool | 1 |
| description | desc\_naid | int8 | 19 |
| description | desc\_type | text | 2147483647 |
| description | data\_ctl\_gp | jsonb | 2147483647 |
| description | record\_gp\_no | text | 2147483647 |
| description | collection\_identifier | text | 2147483647 |
| description | parent\_naid | int8 | 19 |
| description | other\_title | text | 2147483647 |
| description | inclusive\_start\_date\_qualifier\_naid | int8 | 19 |
| description | inclusive\_end\_date\_qualifier\_naid | int8 | 19 |
| description | title | text | 2147483647 |
| description | inclusive\_start\_date | jsonb | 2147483647 |
| description | coverage\_start\_date\_qualifier\_naid | int8 | 19 |
| description | coverage\_end\_date\_qualifier\_naid | int8 | 19 |
| description | inclusive\_end\_date | jsonb | 2147483647 |
| description | coverage\_start\_date | jsonb | 2147483647 |
| description | coverage\_end\_date | jsonb | 2147483647 |
| description | date\_note | text | 2147483647 |
| description | scope\_content\_note | text | 2147483647 |
| description | staff\_only\_note | text | 2147483647 |
| description | record\_status | bool | 1 |
| description | party\_designation\_naid | int8 | 19 |
| description | description\_author | jsonb | 2147483647 |
| description | begin\_congress\_naid | int8 | 19 |
| description | end\_congress\_naid | int8 | 19 |
| description | arrangement | text | 2147483647 |
| description | function\_user | text | 2147483647 |
| description | general\_note | text | 2147483647 |
| description | local\_identifier | text | 2147483647 |
| description | numbering\_note | text | 2147483647 |
| description | accession\_number\_naid | int8 | 19 |
| description | record\_ctr\_xfer\_number\_naid | int8 | 19 |
| description | disposition\_auth\_number\_naid | int8 | 19 |
| description | internal\_xfer\_number\_naid | int8 | 19 |
| description | xfer\_note | text | 2147483647 |
| description | custodial\_history\_note | text | 2147483647 |
| description | online\_resource\_naid | int8 | 19 |
| description | scale\_note | text | 2147483647 |
| description | edit\_status\_naid | int8 | 19 |
| description | sound\_type\_naid | int8 | 19 |
| description | total\_footage | float4 | 8 |
| description | is\_av | bool | 1 |
| description | ptr\_obj\_availability\_date | jsonb | 2147483647 |
| description | production\_date | jsonb | 2147483647 |
| description | copyright\_date | jsonb | 2147483647 |
| description | subtitle | text | 2147483647 |
| description | release\_date | jsonb | 2147483647 |
| description | broadcast\_date | jsonb | 2147483647 |
| description | shotlist | text | 2147483647 |
| description | broadcast\_date\_qualifier\_naid | int8 | 19 |
| description | release\_date\_qualifier\_naid | int8 | 19 |
| description | copyright\_date\_qualifier\_naid | int8 | 19 |
| description | production\_date\_qualifier\_naid | int8 | 19 |
| description | ptr\_obj\_availability\_date\_qualifier\_naid | int8 | 19 |
| description | is\_under\_edit | bool | 1 |
| description | created\_date | \_timestamp | 29 |
| description | imported\_date | timestamp | 29 |
| description | last\_changed\_date | timestamp | 29 |
| description | last\_approved\_date | timestamp | 29 |
| description | last\_brought\_under\_edit\_date | timestamp | 29 |
| description | approval\_history | \_json | 2147483647 |
| description | change\_history | \_json | 2147483647 |
| description | brought\_under\_edit\_history | \_json | 2147483647 |
| description | created\_user | text | 2147483647 |
| description | imported\_user | text | 2147483647 |
| description | last\_changed\_user | text | 2147483647 |
| description | last\_approved\_user | text | 2147483647 |
| description | last\_brought\_under\_edit\_user | text | 2147483647 |
| digital\_object | digital\_objectid | bigserial | 19 |
| digital\_object | desc\_naid | int8 | 19 |
| digital\_object | auth\_list\_naid | int8 | 19 |
| digital\_object | object\_description | text | 2147483647 |
| digital\_object | object\_designator | text | 2147483647 |
| digital\_object | label\_flag | text | 2147483647 |
| digital\_object | locate\_by | text | 2147483647 |
| digital\_object | access\_filename | text | 2147483647 |
| digital\_object | access\_file\_size | text | 2147483647 |
| digital\_object | thumbnail\_filename | text | 2147483647 |
| digital\_object | thumbnail\_file\_size | numeric | 10 |
| digital\_object | projectID | text | 2147483647 |
| digital\_object | imported | date | 13 |
| digital\_object | status | text | 2147483647 |
| digital\_object | display | text | 2147483647 |
| digital\_object | in\_database | text | 2147483647 |
| digital\_object | original\_process | text | 2147483647 |
| digital\_object | scanning\_color | text | 2147483647 |
| digital\_object | original\_width | numeric | 10 |
| digital\_object | scanning\_dimensions | numeric | 10 |
| digital\_object | server\_name | text | 2147483647 |
| digital\_object | version | numeric | 10 |
| digital\_object | scanning\_process | text | 2147483647 |
| digital\_object | scanning\_medium | text | 2147483647 |
| digital\_object | scanning\_medium\_category | text | 2147483647 |
| digital\_object | original\_orientation | text | 2147483647 |
| digital\_object | masterfile\_size | numeric | 10 |
| digital\_object | master\_media\_backup | text | 2147483647 |
| digital\_object | master\_media\_primary | text | 2147483647 |
| digital\_object | digital\_object\_translation | text | 2147483647 |
| digital\_object | master\_derivation\_file\_media | text | 2147483647 |
| digital\_object | master\_filename | text | 2147483647 |
| digital\_object | digital\_object\_transcript | text | 2147483647 |
| digital\_object | original\_dimension | numeric | 10 |
| digital\_object | original\_height | numeric | 10 |
| digital\_object | original\_medium | text | 2147483647 |
| digital\_object | batch\_number | numeric | 10 |
| digital\_object | batch\_date | date | 13 |
| digital\_object | original\_color | text | 2147483647 |
| finding\_aid | finding\_aidid | bigserial | 19 |
| finding\_aid | desc\_naid | int8 | 19 |
| finding\_aid | finding\_aid\_type\_naid | int8 | 19 |
| finding\_aid | finding\_aid\_note | text | 2147483647 |
| finding\_aid | finding\_aid\_source | text | 2147483647 |
| finding\_aid | finding\_aid\_url\_naid | int8 | 19 |
| finding\_aid | object\_type\_naid | int8 | 19 |
| general\_media\_type | general\_media\_typeid | bigserial | 19 |
| general\_media\_type | auth\_list\_naid | int8 | 19 |
| general\_media\_type | media\_occurrenceid | int8 | 19 |
| general\_record\_type | general\_record\_typeid | bigserial | 19 |
| general\_record\_type | desc\_naid | int8 | 19 |
| general\_record\_type | auth\_list\_naid | int8 | 19 |
| holding\_measurment | holding\_measurmentid | bigserial | 19 |
| holding\_measurment | physical\_occurenceid | int8 | 19 |
| holding\_measurment | auth\_list\_naid | int8 | 19 |
| holding\_measurment | holding\_measurment\_count | text | 2147483647 |
| jurisdiction | jurisdictionid | bigserial | 19 |
| jurisdiction | prim\_auth\_naid | int8 | 19 |
| jurisdiction | geo\_place\_name\_naid | int8 | 19 |
| language | languageid | bigserial | 19 |
| language | desc\_naid | int8 | 19 |
| language | language\_naid | int8 | 19 |
| location | locationid | bigserial | 19 |
| location | auth\_list\_naid | int8 | 19 |
| location | note | text | 2147483647 |
| location | physical\_occurrenceid | int8 | 19 |
| media\_occurrence | media\_occurrenceid | bigserial | 19 |
| media\_occurrence | physical\_occurenceid | int8 | 19 |
| media\_occurrence | specific\_media\_type\_naid | int8 | 19 |
| media\_occurrence | containerid | text | 2147483647 |
| media\_occurrence | media\_occurence\_note | text | 2147483647 |
| media\_occurrence | physical\_restriction\_note | text | 2147483647 |
| media\_occurrence | technical\_access\_requirement\_note | text | 2147483647 |
| media\_occurrence | piece\_count | numeric | 10 |
| media\_occurrence | reproduction\_count | numeric | 10 |
| media\_occurrence | dimension\_naid | int8 | 19 |
| media\_occurrence | height | numeric | 10 |
| media\_occurrence | width | numeric | 10 |
| media\_occurrence | depth | numeric | 10 |
| media\_occurrence | color\_naid | int8 | 19 |
| media\_occurrence | process\_naid | int8 | 19 |
| media\_occurrence | base\_naid | int8 | 19 |
| media\_occurrence | emulsion\_naid | int8 | 19 |
| microform\_pub | microform\_pubid | bigserial | 19 |
| microform\_pub | desc\_naid | int8 | 19 |
| microform\_pub | auth\_list\_naid | int8 | 19 |
| microform\_pub | microform\_pub\_note | text | 2147483647 |
| org\_name\_ref | org\_name\_refid | bigserial | 19 |
| org\_name\_ref | prim\_auth\_naid | int8 | 19 |
| org\_name\_ref | org\_name\_naid | int8 | 19 |
| org\_personal\_contributor | org\_personal\_contributorid | bigserial | 19 |
| org\_personal\_contributor | desc\_naid | int8 | 19 |
| org\_personal\_contributor | prim\_auth\_naid | int8 | 19 |
| org\_personal\_contributor | auth\_list\_naid | int8 | 19 |
| org\_personal\_contributor | personal\_or\_organizational\_contributor | bool | 1 |
| organizational\_reference | organizational\_referenceid | bigserial | 19 |
| organizational\_reference | prim\_auth\_naid | int8 | 19 |
| organizational\_reference | orgref\_naid | int8 | 19 |
| other\_preservation\_characteristic | other\_preservation\_characteristicid | bigserial | 19 |
| other\_preservation\_characteristic | auth\_list\_naid | int8 | 19 |
| other\_preservation\_characteristic | media\_occurrenceid | int8 | 19 |
| pers\_org\_topsub\_geo\_sppr\_ref | pers\_org\_topsub\_geo\_sppr\_refid | bigserial | 19 |
| pers\_org\_topsub\_geo\_sppr\_ref | desc\_naid | int8 | 19 |
| pers\_org\_topsub\_geo\_sppr\_ref | prim\_auth\_naid | int8 | 19 |
| pers\_org\_topsub\_geo\_sppr\_ref | auth\_reference\_type\_donor | text | 2147483647 |
| personal\_reference | personal\_referenceid | bigserial | 19 |
| personal\_reference | prim\_auth\_naid | int8 | 19 |
| personal\_reference | person\_naid | int8 | 19 |
| physical\_occurence | physical\_occurenceid | bigserial | 19 |
| physical\_occurence | desc\_naid | int8 | 19 |
| physical\_occurence | auth\_list\_naid | int8 | 19 |
| physical\_occurence | extent | text | 2147483647 |
| physical\_occurence | physical\_occurance\_note | text | 2147483647 |
| physical\_occurence | gpra\_indicators | bool | 1 |
| physical\_occurence | container\_list | text | 2147483647 |
| predecessor\_successor | predecessor\_successorid | bigserial | 19 |
| predecessor\_successor | prim\_auth\_naid | int8 | 19 |
| predecessor\_successor | predecessor\_successor\_naid | int8 | 19 |
| predecessor\_successor | predecessor\_successor | text | 2147483647 |
| primauth | prim\_auth\_naid | int8 | 19 |
| primauth | term\_name | text | 2147483647 |
| primauth | auth\_type | text | 2147483647 |
| primauth | name\_heading | text | 2147483647 |
| primauth | full\_name | text | 2147483647 |
| primauth | numerator | text | 2147483647 |
| primauth | personal\_title | text | 2147483647 |
| primauth | birthdate\_qualifier\_naid | int8 | 19 |
| primauth | deathdate\_qualifier\_naid | int8 | 19 |
| primauth | birthdate | jsonb | 2147483647 |
| primauth | deathdate | jsonb | 2147483647 |
| primauth | biographical\_note | text | 2147483647 |
| primauth | source\_note | text | 2147483647 |
| primauth | proposer\_name | text | 2147483647 |
| primauth | proposal\_date | date | 13 |
| primauth | reference\_unit\_naid | int8 | 19 |
| primauth | record\_source\_naid | int8 | 19 |
| primauth | naco\_submitted | bool | 1 |
| primauth | import\_rec\_ctl\_no | text | 2147483647 |
| primauth | notes | text | 2147483647 |
| primauth | scope\_note | text | 2147483647 |
| primauth | saco\_submitted | bool | 1 |
| primauth | lat\_long | numeric | 10 |
| primauth | admin\_hist\_note | text | 2147483647 |
| primauth | establish\_date | jsonb | 2147483647 |
| primauth | abolish\_date | jsonb | 2147483647 |
| primauth | establish\_date\_qualifier\_naid | int8 | 19 |
| primauth | abolish\_date\_qualifier\_naid | int8 | 19 |
| primauth | last\_changed\_date | timestamp | 29 |
| primauth | last\_approved\_date | timestamp | 29 |
| primauth | last\_brought\_under\_edit | timestamp | 29 |
| primauth | approval\_history | json | 2147483647 |
| primauth | changehistory | json | 2147483647 |
| primauth | brought\_under\_edit\_history | json | 2147483647 |
| primauth | variant\_person\_naid | int8 | 19 |
| primauth | ispreferred | bool | 1 |
| primauth | variant\_org\_naid | int8 | 19 |
| primauth | is\_under\_edit | bool | 1 |
| program\_area\_reference | program\_area\_referenceid | bigserial | 19 |
| program\_area\_reference | prim\_auth\_naid | int8 | 19 |
| program\_area\_reference | program\_area\_naid | int8 | 19 |
| reference\_unit | reference\_unitid | bigserial | 19 |
| reference\_unit | auth\_list\_naid | int8 | 19 |
| reference\_unit | physical\_occurrenceid | int8 | 19 |
| seealso | seealsoid | bigserial | 19 |
| seealso | prim\_auth\_naid | int8 | 19 |
| seealso | seealso\_person\_naid | int8 | 19 |
| series\_former\_parent | series\_former\_parentid | bigserial | 19 |
| series\_former\_parent | desc\_naid | int8 | 19 |
| series\_former\_parent | former\_parent\_naid | int8 | 19 |
| special\_pjt\_for\_auth | special\_pjt\_for\_authid | bigserial | 19 |
| special\_pjt\_for\_auth | prim\_auth\_naid | int8 | 19 |
| special\_pjt\_for\_auth | auth\_list\_naid | int8 | 19 |
| special\_pjt\_for\_desc | special\_pjt\_for\_descid | bigserial | 19 |
| special\_pjt\_for\_desc | desc\_naid | int8 | 19 |
| special\_pjt\_for\_desc | auth\_list\_naid | int8 | 19 |
| specific\_access\_restriction | specific\_access\_restrictionid | bigserial | 19 |
| specific\_access\_restriction | specific\_access\_restriction\_naid | int8 | 19 |
| specific\_access\_restriction | access\_restrictionid | int8 | 19 |
| specific\_access\_restriction | security\_classification\_naid | int8 | 19 |
| specific\_use\_restriction | specific\_use\_restrictionid | bigserial | 19 |
| specific\_use\_restriction | user\_restrictionid | int8 | 19 |
| specific\_use\_restriction | auth\_list\_naid | int8 | 19 |
| use\_restriction | use\_restrictionid | bigserial | 19 |
| use\_restriction | desc\_naid | int8 | 19 |
| use\_restriction | auth\_list\_naid | int8 | 19 |
| use\_restriction | note | text | 2147483647 |
| usefor | useforid | bigserial | 19 |
| usefor | prim\_auth\_naid | int8 | 19 |
| usefor | usefor\_naid | int8 | 19 |
| variant\_control\_number | variant\_control\_numberid | bigserial | 19 |
| variant\_control\_number | desc\_naid | int8 | 19 |
| variant\_control\_number | auth\_list\_naid | int8 | 19 |
| variant\_control\_number | variant\_ctl\_no | text | 2147483647 |
| variant\_control\_number | variant\_ctl\_no\_note | text | 2147483647 |

1. ETL Script for Authority List Migration from Oracle to Aurora

Will be included in the final version of this document.