ORACLE

JSON Full-Text Search with Oracle Globally Distributed Database 23ai

With Oracle Cloud Infrastructure Base Database Service

August, 2024, Version [1.0]

Public

Purpose statement

This document provides details of a specific use-case and set of features included in Oracle Database release 23ai. It is intended solely to help you assess the business benefits of upgrading to 23ai and planning for the implementation and upgrade of the product features described.

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement, nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

ORACLE

Table of contents

List of figures and tables

Introduction

This document provides design and deployment guidance for a scalable, distributed full-text search solution for JSON documents certified on an Oracle 23ai Globally Distributed Database environment.

The advantages of using Oracle Globally Distributed Database as your repository and engine for full-text search include:

- Oracle Database can remain the source of record for data that requires textual search capabilities with the use of Oracle Text, without extracting and continually reconciling data with a third-party search platform.
- Oracle customers using third-party JSON-based search indexes can potentially reduce overall costs by transitioning these indexes into Oracle Database.
- Oracle Globally Distributed Database provides a distributed horizontal scalable database platform that can grow with your document collection and application/service performance needs.

All of this can be achieved with the Oracle Database native JSON support combined with Oracle Text for indexing and search capabilities, with Oracle Globally Distributed Database on OCI to achieve horizontal scalability and meet data sovereignty requirements.

Technologies

This solution combines three Oracle technologies: Oracle Globally Distributed Database, Oracle Text and Oracle's native JSON support.

Oracle Database support for JavaScript Object Notation (JSON)

Oracle Database provides the benefits of SQL and relational databases to JSON data, allowing you to store and manipulate your JSON documents in the same ways and with the same confidence as any other type of database data.

Oracle Database supports JSON natively with relational database features, including transactions, indexing, declarative querying, and views. Oracle Database support for JSON provides the best fit between the worlds of relational storage and querying JSON data, allowing relational and JSON queries to work well together. Oracle SQL/JSON support is also closely aligned with the JSON support in the SQL Standard.

Business documents can be stored as JSON documents in a relational table. Each row represents a document with the content inside the document stored in a column with the JSON datatype.

Further Reading:

- 23ai Overview of Sharding JSON Documents
- 23ai documentation: JSON Developer's Guide
- Oracle Blog post on the new JSON datatype
- Oracle Blog post on how JSON documents can be stored in a sharded environment

Oracle Text

External text engines are widely available, but most of them only work with file-based text stores. Oracle Database provides an integrated text indexing and search engine that supports text stores both within and outside the Oracle Database. If the text to search is in the database to start with, using the integrated text engine within the Oracle Database has several advantages. In this case, the developer or application owner has:

- A single repository for all data (text and structured) instead of two, simplifying maintenance, backup, etc.
- Indexes in the same repository, ensuring efficient processing of text and mixed queries.
- A single API for developing applications, leveraging Oracle's converged database and combining text search with any other processing in the database, for example spatial, JSON processing, or graph
- Integration with the Oracle SQL execution engine and query plan optimizer, letting Oracle decide on the most efficient execution plan for the whole statement
- Oracle Text is part of all editions of the Oracle Database. Because text is stored in the database, you get an integrated, holistic view of your data. Additionally, the database chooses the fastest plan to execute queries that involve both text and structure content.

In terms of our use-case, Oracle Text supports JSON documents. This means that Oracle Text understands and can parse the structure of the documents. Oracle Text can use the names and values of the JSON elements within each row's JSON column data, not just the raw text. Oracle Text provides the ability to score and rank search results and render a "snippet" of the document contents in HTML, with term highlighting to provide context and relevance when the results of the query are processed by the application for display. This and other relational aspects of the search queries allow us to optimize the data returned to the application and minimize the application-based processing required.

Further Reading:

- Tech Brief: New User's Guide to Oracle Text in Oracle Database
- 23ai Oracle Text Application Developer's Guide

Oracle Globally Distributed Database

Oracle Globally Distributed Database enables the deployment of hyperscale globally distributed converged databases. It is the Oracle Database feature that supports application requirements for linear scalability, elasticity, fault isolation, and geographic distribution of data for data sovereignty. Oracle Database implements these capabilities by distributing chunks of a data set across independent Oracle databases (shards). Shards can be deployed in the cloud or on-premises and require no specialized hardware or software.

The pool of shards is presented to the application as a single logical database. The single logical database is known as a sharded database. Business solutions can elastically scale their data tier (data, transactions, and users) to any level, on any platform, simply by adding shards to the sharded database. Data and workloads are automatically balanced across the shards transparently to the application tier.

Benefits of sharding with Oracle Globally Distributed Database:

- **Linear Scalability:** Globally Distributed Database eliminates performance bottlenecks and makes it possible to linearly scale performance and capacity by adding shards.
- **High Availability and Fault Containment:** Globally Distributed Database is a shared-nothing hardware infrastructure that provides strong fault isolation and eliminates requirements that can become single points of failure, such as shared disk, SAN, and Clusterware. The failure or performance degradation of one shard does not affect the performance or availability of other shards.
- **Data Sovereignty with Geographical Distribution:** Allows specific data to be stored close to its consumers while also meeting regulatory requirements when data at rest must be stored in a specific jurisdiction.

For in-depth details about the features and benefits of Oracle Globally Distributed Database, visit the Globally Distributed Database page on Oracle.com and 23ai documentation.

The solution offered in this document uses the "system-managed sharding" method to distribute the data evenly across the shards. There are other data distribution methods available to distribute sharded table data across the sharded database to meet various requirements: User-Defined, Directory-Based, and Composite sharding. These methods provide additional features when the data set needs to be sharded in more purposeful ways, such as to

support data residency and other business or regulatory requirements. For additional details see Appendix B: Sharded Data Distribution Methods.

Solution Overview

We will walk through and discuss the database-centric aspects of a JSON full-text search solution, including design and implementation strategy insights, examples, and the critical aspects of each step of the process.

The overall solution is not complex to deploy as it fundamentally involves

1) Deciding on the right mix of JSON and Relational that takes advantage of the power of both
2) Deploying your Oracle Globally Distributed Database Topology
3) Creating the appropriate schemas and indexes and, finally
4) Constructing a template query that can be used for user Searches.

This paper walks through each step in detail to highlight the different decisions that are available and to function as a reference. We walk through the design of a complete search application. We also include many validation steps which can also be used later to help in any diagnosis or simply for informational reasons.

This walk-through includes:

1. Plan
    a. Review application requirements.
    b. Prepare and analyze the JSON data set.
2. Build
    a. Deploy OCI 23ai Base Database services.
    b. Configure the Oracle Globally Distributed Database topology.
    c. Create the schema user, tablespaces, and table family.
3. Load
    a. Load the JSON data into the table family on each shard in parallel.
    b. Collect statistics.
4. Index
    a. Create an index on the primary key.
    b. Create a multi-value index on JSON Array data.
    c. Configure Oracle Text preferences.
    d. Create the Oracle Text index on the JSON column.
5. Search
    a. Review your application use-case scenarios and search form user experience.
    b. Discuss query design factors and requirements.
    c. Review example SQL.

Planning The Solution

For this technical brief, we demonstrate how a JSON data set can be analyzed to inform schema design, loaded into the Oracle Database, indexed, and used as a basis for full-text search queries.

We have opted to use a single table for our schema in which we bulk load the public Wikipedia article data after converting it into JSON format. The WikiCommons schema for Wikipedia has several tables and available data files related to the full Wikipedia application functionality, we only need the corpus of article data, in JSON format, to demonstrate the full-text search indexing and query use-case. Our example table will also include several virtual columns to automatically parse and expose values from the JSON column data for direct use with relational query operators and search predicates.

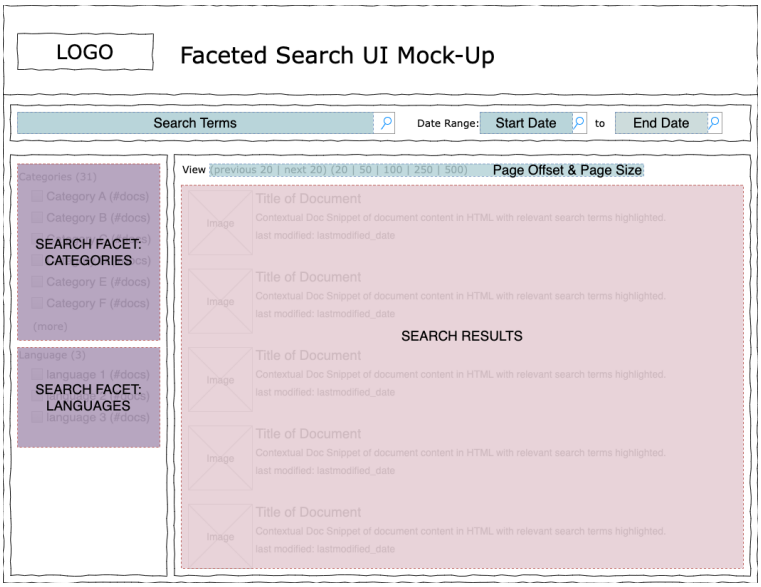
Example Application Requirements

Our example search form user interface differs from that of Wikipedia's advanced search options. For our example, we include fields for terms to search for, and start and end dates to allow the user to constrain the search over time. Once a user performs a search with any of these fields, the search results are populated with the first page of results. The user can then customize the number of results and which page to view. This is all standard fare for the search user experience.

In addition to the search results, there are two search queries performed to aggregate counts of matching documents by specific data dimensions. These are colloquially called "facets" and allow the presentation of values along these dimensions that users can choose to further refine the search results in a parametric fashion.

Our example uses two dimensions or facets of the data: the language the document is written in, and the "categories" the authors have specified for each document. Our user interface displays these two facets in a left-hand navigation panel, side-by-side with the search results. The data for each facet contains all the values found for the dimension, and the count of documents with that value.

Figure 1: Abstract of the Search Form User Interface



For this solution, we have three queries. The first query populates the search results panel. The second and third queries generate the data necessary to present the two facets as groupings of checkboxes for each of the facet's values and the count of documents for each. With relational data, designing these queries is academic. Doing so efficiently with JSON Database features and Oracle Text is why we're here. We will introduce the functions and design necessary to perform these queries.

The JSON Data Set

For our example, we use a JSON representation of the public Wikipedia Articles English-language data set ("enwiki"). We chose Wikipedia because it provides both a publicly available dataset and the precise full-text search use-case we're interested in. Although not originally in JSON format, we converted the enwiki articles data file to JSON for use in our example. The latest "enwiki" data dump files in XML format can be downloaded from https://dumps.wikimedia.org/enwiki/latest/.

Much of the WikiCommons schema and portions of each article's JSON document are not relevant to demonstrate our full-text search solution. The converted JSON data file was altered to remove unnecessary JSON elements from each record.

Below we provide a short introductory example document representative of the JSON elements in our data set. The data file we load for our example consists of documents similar to this one but with actual subject matter and metadata values from Wikipedia.

After that, the next section describes the various elements in the JSON documents, and discusses some examples of how we can leverage the Oracle JSON database features to our best advantage for efficient SQL queries for our full-text search use-cases.

Wikipedia enwiki Article:Synthetic Example (formatting modified for display)

```
{"title": "Oracle Demo Intro Article ",
  "wikiTitle": "ORACLE_DEMO_INTRO_ARTICLE",
  "wid": 999019999,
  "lang": "EN",
  "timestamp": "2022-06-21T05:22:00Z",
  "paragraphs": [
    "This is the first paragraph of the MAA Demo Intro Article test document for database sharding. This is the second sentence of the first paragraph",
    "This is the second paragraph. There are multiple lines each with multiple sentences.\nA second line in the second paragraph.",
    "With Regards to the Data structure of each document:\n",
    "The .wid element is unique per article and extracted for use as the shard key and primary key in the Articles table.",
    "There may be hundreds of strings in the .paragraphs array with up to 4kb per string.",
    "The longest article is over 440kb.",
    "The .categories array .id element contains the actual category term with spaces replaced with underscore and prefixed with the phrase: Category:. \n The .templates array contains various metadata about the article with each .name element being the key and corresponding .description array containing the value.
  ],
  "categories": [
    {
      "id": "Category:Database_Sharding",
      "anchor": "database sharding", "start": 71, "end": 88, "type": "CATEGORY"
    },
    {
      "id": "Category:Oracle_Demo_Article",
      "anchor": "", "start": 0, "end": 0, "type": "CATEGORY"
    },
    {
      "id": "Category:Facet B",
      "anchor": "", "start": 0, "end": 0, "type": "CATEGORY"
    }
  ]
}
```

Analyzing the JSON Data for Schema and Query Design

Each JSON field in the data set should be evaluated for how it might be used in constructing our associated database table. Identify other JSON elements that contain data that would be frequently used in queries where adding indexes or virtual columns could be appropriate for performance reasons or to simplify the application's SQL coding requirements.

The analysis of each JSON element in our example data set is as follows:

wid

This JSON element contains the unique ID for each article consisting of a 1 to10-digit number. We will be using this value as the basis for the Article_ID column which will be the primary key for the Articles table and the sharding key for the table family.

Title

This is the human-readable title of the article. Title is one of the values we want to both search on and present efficiently in the search results. Implementing a virtual column allows us to include this value in the SELECT clause without the complexity of additional JSON_VALUE() function calls to parse the JSON element value in every query. From a search query filter predicate perspective, we will use the JSON_TEXTCONTAINS() function using the JSON column directly to perform the full-text search using the Oracle Text index.

Paragraph[] Array Data

The JSON document's paragraph array contains comma-separated quoted strings. This comprises the bulk of the article content and is used along with Title in the main search results query as the primary source data fields for the requested keywords string search predicates. The paragraph array is searched using the JSON_TEXTCONTAINS() function, and the execution of this function gets optimized to use the available Oracle Text search index. The paragraph JSON element value is also used with an Oracle Text function called CTX_DOC.POLICY_SNIPPET() in the SELECT clause of our main search results query to return a short section of contextually relevant document text with the search terms highlighted in HTML format.

Categories[] Array Data

In Wikipedia, category assignments to articles are a many-to-many relationship.

The categories data serves as a useful example to demonstrate how to perform a faceted search aggregation query using JSON array-based values. In the full-text search user experience, presenting a faceted search capability includes querying for an aggregation, or count, of results based on an associated value that provides a meaningful grouping. These groupings are referred to as "buckets" and the collection of buckets for a given piece of associated data or metadata is called a "facet". Typically, the user is presented with a list of buckets for one or more facets and allowed to refine their search parameters by choosing a subset of buckets to use as a filter for the search results.

Lang

This JSON element contains the ISO 639-1 short code for a 2-char regional language name.

To allow the use of simple relational search predicates and efficient query of these values, we chose to implement a virtual column for this element in our relational schema. This avoids repeated use of the JSON function calls in application code, simplifies the query syntax, and allows the database to efficiently optimize the SQL execution.

Timestamp

This element contains the last-modified date string for each article document. The "timestamp" format in article data does not directly match an Oracle Timestamp format but is easily cast to the timestamp_tz data type.

This is a perfect example of the efficiency provided by virtual columns. The data type conversion can be configured once in the schema definition rather than in every client-side query needing the last-modified timestamp value.

Schema Design

Oracle's converged database allows data to be accessed relationally, indexed, and searched via Oracle Text, and via JSON database functions. In considering the schema design we can take into account all of these access methods. This allows customers to take advantage of consuming and maintaining data already in JSON format as well as the capabilities of a relational database.

JSON Columns

Oracle stores JSON within a JSON datatype column of a database table. The JSON documents within this column are fully indexable and searchable. Standard SQL queries can include filter predicates using JSON functions to perform searches within the JSON documents directly.

In the simplest case, all JSON records are stored in the database as individual rows in a large table. Each table is partitioned across the shards but remains one logical table. Our example schema consists of a single table described in the Schema Structure section below.

Virtual Columns

These are columns with values derived from other columns or fixed expressions. The advantage of a virtual column is that we can create a column in a relational table as a column derived from the evaluation of an expression on the JSON field in our table. Many of the columns in our example table are virtual columns.

For example, the Title of a document may be stored as an element within the JSON document stored in the JSON column. The title data can also appear in a standard relational column, perhaps named “title”, with the value automatically derived from the JSON element value. The advantage is that we can use this virtual column in purely relational expressions with the JSON-specific function call details abstracted at the schema level instead of in every query.

Sharded Data Distribution Methods

There are multiple methods for partitioning the data across the sharded databases in your topology. This is relevant to our data design as it determines how the data will be distributed.

The system-managed data distribution method is used for the examples in this technical brief.

The system-managed method requires only that a column be declared as the “sharding key”. Beyond that, the user does not need to specify the mapping of data to shards. Data is automatically distributed across shards using partitioning by consistent hash of the values of the shard key. The partitioning algorithm evenly and randomly distributes data across shards. The distribution used by the system-managed method is aptly named as it is intended to eliminate hot spots, provide uniform performance across shards, and automatically maintain the balanced distribution of the data when shards are added to or removed from a sharded database.

In our use case:

- The data and use-cases do not include any specific business or technical requirements for correlating and grouping specific records to the same shard.
- Each document record is independent of any other.
- All document records can be re-sharded to any other shard if the sharded database needs to be scaled-out to increase performance.

If there were some correlations, such as a common account with multiple sales invoices, or a requirement for Data Sovereignty records must be sharded together based on a country code or other regional field value, then other sharding methods would be more appropriate.

Sharding Key

Central to sharded databases is the concept of a sharding key. Often this can be naturally mapped to a record identifier although no formal relation needs to exist between the sharding key and the record.

To obtain the benefits of sharding, the schema of a sharded database should be designed in a way that maximizes the number of database requests processed on a single shard. Choosing which field to use as a sharding key is important in organizing your documents across the sharded database.

When using system-managed data distribution, the sharding key does not strictly need to be unique or necessarily the primary key. However, It is recommended that you use the primary key as the sharding key when organizing unrelated records with a unique document ID.

For more information, see the Sharding Keys section of the Oracle Globally Distributed Database documentation.

In online transaction processing scenarios, queries and insert/update/deletion operations are performed with a known primary key and therefore also a known sharding key. These operations can be directed specifically to the appropriate single shard based on the sharding key by the client-side database driver when Oracle drivers are used.

In the document search use-case used in our examples, the sharding key is not known when searches are submitted, only the keywords/full-text to be searched. When a user clicks on a returned document in the search results, the application would then have the document ID as the sharding key data for that unique document request and can retrieve the document directly from the correct shard.

Some suggestions when choosing a sharding key:

- The sharding key can be a text string or number, but we recommend that it be unique and so uniquely identify the document.
- Sharding keys should rarely or never change. It is best to use a fixed ID that will always be associated with the document.

Schema Structure

Related tables that are to be sharded/distributed the same way are called a sharded table family. A sharded database can have multiple table families. There can also be “duplicated tables” where each shard has the same table contents across all shards, typically used for "look-up" tables. Sharded tables are horizontally partitioned across shards, and duplicated tables are replicated entirely to all shards.

See the Sharded Table Family and Duplicated Tables documentation for a full description.

Database Schema Details

Our example implements a single table named "Articles" as the parent table in our sharded table family. The Articles table contains an ID column as the primary key and sharding key, the Article_JSON column, and four virtual columns based on the Articles_JSON column content. The design intent for each column is summarized here.

Article_ID

This is the primary key and used as the sharding key. The data type is NUMBER(10). Article_ID contains the scalar value extracted from the $.wid JSON element. The value must be extracted and inserted verbatim when the data is loaded from the JSON file's external table, and during single-record insert/update operations. The primary key column cannot be a virtual column.

Article_JSON

The Article_JSON column is of datatype JSON and contains the JSON document inserted verbatim as extracted from the JSON file’s external table during data load, or as inserted/updated by the application.

Title

Title is a virtual column with a data type of VARCHAR2(512), evaluating the JSON_VALUE() of $.title from Articles_JSON. This column will be populated dynamically and cannot have a value explicitly inserted.

Lang

This is a virtual column with a VARCHAR2(2) data type, evaluating the JSON_VALUE() of $.lang from Articles_JSON. The values contain an ISO standard 2-character country code. This column will be populated dynamically and cannot have a value explicitly inserted.

Last_Modified

A virtual column querying the JSON_VALUE() of the $.timestamp element from Articles_JSON as a string and converting it to a TIMESTAMP(0) WITH TIMEZONE data type. This column will be populated dynamically and cannot have a value explicitly inserted.

The Article table for our examples, to be created in a later section, is described as follows:

```
SQL> desc Articles
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ARTICLE_JSON                              NOT NULL JSON
 ARTICLE_ID                                NOT NULL VARCHAR2(10)
 DATE_LOADED                               NOT NULL TIMESTAMP(6)
 TITLE                                              VARCHAR2(512)
 WIKI_TITLE                                         VARCHAR2(512)
 LANG                                               VARCHAR2(2
 LAST_MODIFIED                                      TIMESTAMP(0) WITH TIME ZONE
```

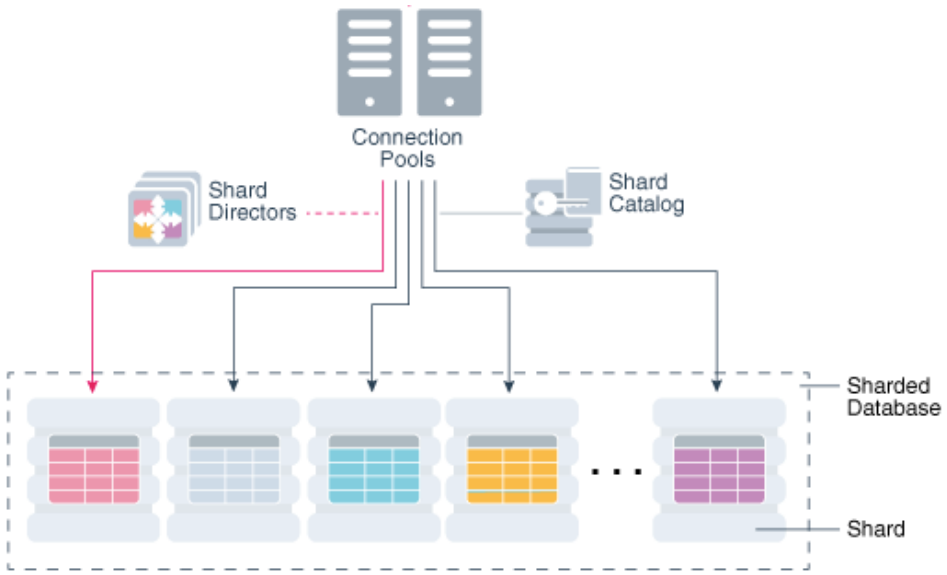
Planning the Sharded Database Topology

Planning the topology and architecture of each sharded database will vary from solution to solution and across environments based on the specific business, technical, and regulatory requirements imposed in the same way as any other database solution. With Oracle Globally Distributed Database, there are just a few more components to factor into the equation.

For the purposes of this technical brief, as a proof-of-concept (POC) focused on the JSON full-text search indexing and query functionality, all high availability and disaster recovery aspects necessary for a rigorous enterprise implementation are out-of-scope. Links to Oracle documentation can be found in the Further Reading appendix at the end of this document.

The simplest topology for deploying a sharded database solution POC environment would consist of one shard catalog database (single-instance), one shard director, and a few shard databases (single-instance) of sufficient capacity to contain a fraction of your data payload and indices.

Figure 2: Physical Infrastructure Overview - System-managed Sharded Database Deployment



Globally Distributed Database Components

Shards

The individual shards each hold a fraction of the records in your data set. The number of shards configured has a direct effect on the overall load that each shard will take. The system-managed data distribution method, recommended for our example use-case, uses a hash of the record identifier (the shard key) to assign each row to a specific shard. This ensures a scalable, even distribution of your data.

Shard Directors

The shard directors are stateless and lightweight components which provide query redirection capabilities to client applications based on a sharding key or record key. Using this sharding key value, the shard director responds to the client application with a connection redirect message for the appropriate shard database. This is used when an application needs to retrieve documents directly, either as the result of a search or at any point when a document needs to be retrieved for reading or updating.

Shard Catalog

The multi-shard query coordinator functionality of the shard catalog database is used for queries that involve a search across more than one shard. The coordinator is responsible for distributing these multi-shard queries to every shard and then retrieving and combining the results. The shard catalog database also serves as the repository for the sharded database configuration metadata.

Building the Environment

Oracle Globally Distributed Database can be deployed on-premises or in the cloud. Currently, the latest version of the database with the features required for the solution in this technical brief can be deployed as OCI Base Database services. The shard directors can be deployed as OCI Compute instances.

When deploying on-premises, the traditional hardware, network, operating system, software requirements, and tuning for Oracle Database apply as usual. There are some tuning specifications to be aware of that apply specifically to Oracle Globally Distributed Database installations. Please review the full specifications in the Provision and Configure Hosts and Operating Systems section of the Oracle Globally Distributed Database Guide.

Note that we have provided here the exact steps for creating a Globally Distributed Database manually, either on premise or using OCI Base Database Service. If automation solutions are used, such as creating an environment using MarketPlace on OCI then feel free to skip down to the Sections on creating the Database Schema on page 28 of this document.

Deploying Database Infrastructure for the Shards and Shard Catalog

Currently, Oracle Database 23ai instances can be deployed using the OCI Base Database Service. Create new Base Database Services for the shard catalog and each shard with the specifications required for your implementation, choosing the 23ai database version. This can be performed using the OCI Console or the OCI Command Line Interface (OCI CLI).

Example OCI CLI command to create a base-database service for a shard catalog:

Note: this example does not include parameters for configuring automatic backups. All parameter values should be tuned to our environment and may not be suitable as-is.

```
[user@laptop ~/dev/oci]$ oci db system launch --display-name "w23catyyz10" \
      --hostname w23catyyz10 \
      --db-name  catcdb \
      --pdb-name catpdb \
      --region ca-toronto-1 \
      --compartment-id ocid1.compartment.oc1..aaaaaaaarc… \
      --subnet-id ocid1.subnet.oc1.ca-toronto-1.aaaaaaaa3q… \
      --availability-domain efXT:CA-TORONTO-1-AD-1 \
      --fault-domains '["FAULT-DOMAIN-1"]' \
      --shape VM.Standard.E4.Flex \
      --node-count 1 \
      --cpu-core-count 4 \
      --initial-data-storage-size-in-gb  2048 \
      --disk-redundancy NORMAL \
      --database-edition ENTERPRISE_EDITION_EXTREME_PERFORMANCE \
      --storage-management LVM \
      --license-model  BRING_YOUR_OWN_LICENSE \
      --db-version 23.4.0.24.05 \
      --character-set AL32UTF8 \
      --ncharacter-set AL16UTF16 \
      --db-workload OLTP \
      --admin-password sys_credential \
      --ssh-authorized-keys-file /full/path/to/.ssh/desired_key.pub \
      --is-incident-logs-enabled true \
      --is-diagnostics-events-enabled true
```

Deploying Compute Infrastructure for the Shard Director

Almost any Linux-based compute instance can be used to host a shard director as long as package, kernel versions, physical storage, and memory requirements are met. See Planning an Installation in the Global Data Services Concepts and Administration Guide.

Installing Global Data Services on the Shard Director Host

Download the Oracle Global Data Services (GDS) global service manager release 21.3.0.0 from Oracle eDelivery and install using the Oracle Universal Installer from the root directory of the software media, following the prompts, or perform a silent install. See the *Oracle Database Global Data Services Concepts and Administration Guide* for information about installing the global service manager (GSM) software.

Open the necessary ports for the GSM in the local OS firewall. This includes the ports that will be specified when creating the GSM instance. The following default values can be customized if necessary.

- Local listener: TCP/1522
- Local ONS: TCP/6123
- Remote ONS: TCP/6234

Example: Oracle Linux firewall lD configuration for GSM ports

```
[opc@w23diryyz10 ~]$ sudo firewall-offline-cmd --port=1522:tcp
[opc@w23diryyz10 ~]$ sudo firewall-offline-cmd --port=6123:tcp
[opc@w23diryyz10 ~]$ sudo firewall-offline-cmd --port=6234:tcp
[opc@w23diryyz10 ~]$ sudo /bin/systemctl restart firewalld
```

Verify Connectivity

Ensure that your network topology and compute instance operating systems allow proper network ingress and egress for the connections required between the GSM instances on the shard director hosts, and the shard and shard catalog database servers.

OCI security lists or network security groups should be correctly defined and applied. Compute OS firewalls for the GSM instances should be configured to allow the required ports. All shard directors and databases will also need the ability to resolve the short host names as well as the fully-qualified domain names.

Testing Database TNS Connectivity

From the shard director, test for TNS ping to the shard and shard catalog databases using the short host names. Also, test TNS ping from the shard catalog to the shards the same way.

Example: Testing connectivity with the tnsping command

```
[oracle@w23diryya10 ~]$ tnsping w23shdyyz11:1521 | egrep 'msec|TNS-|Attempting'
Attempting to contact
(DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=))(ADDRESS=(PROTOCOL=tcp)(HOST=10.8.208.94)(PORT=1521)))
OK (0 msec)
```

Configuring and Tuning the Database Instances

Several configurations must be performed when the generic database services are available.

Parameter Tuning and Event Tracing

Most of the default database parameters set for the OCI Base Database System are sufficient for at least R&D Proof-of-Concept or development environment use; however, some parameter default values need to be tuned specifically due to sharding requirements.

- OPEN_LINKS* – The shard catalog and some direct path load features on the shards require this parameter to be tuned beyond the defaults. Given the complexity of the dependencies, a starting value of 255 or the number of shards in your environment is recommended, whichever is greater.
- GLOBAL_NAMES – Global names should be set to FALSE to allow for the intended interoperability between the shard director (GSM) instances, the shard catalog, and the shards.
- DB_CREATE_FILE_DEST – Must be set to a valid value for your environment
- STANDBY_FILE_MANAGEMENT – Set to AUTO to avoid warnings in GSM configuration
- NLS_NCHAR_CHARACTERSET – Must be identical on all shards and shard catalog databases
- NLS_CHARACTERSET – Must be identical on all shards and shard catalog databases

Setting some trace events is also recommended. Set trace events to allow logging of PL/SQL traces on all shard and shard catalog databases, and for the shards, include events to propagate any parallel query worker errors back to the query coordinator, and disable parallel direct path loading for Oracle Text indexing.

- Enable PL/SQL tracing
  '10798 trace name context forever, level 7'
- Propagating parallel query (PQ) worker errors to the query coordinator instead of throwing an ORA-12801
  '10397 trace name context forever, level 1'
- For Oracle Text, disable parallel direct path loading when indexing:
  '30580 trace name context forever, level 256'

Example DDL: Tune 23ai DBS CDB parameters

```
[oracle@w23catyyz10 ~]$ sqlplus / as sysdba
SQL> ALTER SYSTEM SET MAX_DUMP_FILE_SIZE=UNLIMITED       SCOPE=BOTH;
SQL> ALTER SYSTEM SET STANDBY_FILE_MANAGEMENT=AUTO       SCOPE=BOTH;
SQL> ALTER SYSTEM SET GLOBAL_NAMES=FALSE                 SCOPE=BOTH;
SQL> ALTER SYSTEM SET OPEN_LINKS=255                     SCOPE=SPFILE;
SQL> ALTER SYSTEM SET OPEN_LINKS_PER_INSTANCE=255        SCOPE=SPFILE;
```

Example DDL: Tune Shard Catalog 23ai DBS CDB Trace Events for Sharding

```
SQL> ALTER SYSTEM SET EVENT='10798 TRACE NAME CONTEXT FOREVER, LEVEL 7',
                            COMMENT='PL/SQL Tracing for Sharding for Shard Catalog'
                            SCOPE=SPFILE;
```

Example DDL: Tune Shard 23ai DBS CDB Trace Events for Sharding and Oracle Text

```
SQL> ALTER SYSTEM SET EVENT='10397 TRACE NAME CONTEXT    FOREVER, LEVEL 1',
                           '10798 TRACE NAME CONTEXT    FOREVER, LEVEL 7',
                           '30580 TRACE NAME CONTEXT    FOREVER, LEVEL 256',
                           COMMENT='PL/SQL Tracing for Sharding plus: PQ Worker Error
Propagation, and disable Parallel Direct Path Load for Oracle Text Indexing on Shards only'
                           SCOPE=SPFILE;
```

Restart the CDBs after altering them and confirm trace events are set.

```
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP
SQL> CONNECT / AS SYSDBA
SQL> ORADEBUG SETMYPID
SQL> ORADEBUG EVENTDUMP system
10798 trace name context forever, level 7
30582 trace name context forever, level 1048576
```

Database User Accounts and Password Policies

Specific accounts must be enabled on the shard catalog and shard databases so that the shard directors can connect to the databases and manage the sharded database. Follow the instructions in the documentation to set up your databases and the specified user accounts. See Create the Shard Catalog Database and Create the Shard Databases

Validate Each Shard Database

After fully configuring and tuning the shard databases, run the dbms_gsm_fix.validateShard procedure on each shard PDB.

Note: If Oracle Data Guard is not required and the dg_broker_start parameter is set to false, this will throw an error in the validateShard procedure. This error can be ignored when Data Guard is not in use.

Example: Validate shard PDB configuration

```
$ sqlplus / as sysdba
SQL> set serveroutput on
SQL> execute dbms_gsm_fix.validateShard;
INFO: Data Guard shard validation requested.
INFO: Database role is PRIMARY.
INFO: Database name is SHDCDB0.
INFO: Database unique name is shdcdb0_3s2_yyz.
INFO: Database ID is 493557786.
INFO: Database open mode is READ WRITE.
INFO: Database in archivelog mode.
WARNING: Flashback is off.
INFO: Force logging is on.
INFO: Database platform is Linux x86 64-bit.
INFO: Database character set is AL32UTF8. This value must match the character set of the
catalog database.
INFO: 'compatible' initialization parameter validated successfully.
INFO: Database is a multitenant container database.
INFO: Current container is SHDPDB0.
INFO: Database is using a server parameter file (spfile).
INFO: db_create_file_dest set to: '/u02/app/oracle/oradata/shdcdb0_3s2_yyz/'
INFO: db_recovery_file_dest set to: '/u03/app/oracle/fast_recovery_area'
INFO: db_files=200. Must be greater than the number of chunks and/or tablespaces to be
created in the shard.
ERROR: dg_broker_start set to FALSE.
INFO: remote_login_passwordfile set to EXCLUSIVE.
INFO: db_file_name_convert set to: '*, /u01/app/oracle/oradata/'
INFO: standby_file_management is set to AUTO.
INFO: GSMUSER account validated successfully.
INFO: DATA_PUMP_DIR is '/u01/app/oracle/admin/ORCL/dpdump/16E9F79186922679E0631701F40AB0C5'.

PL/SQL procedure successfully completed.
```

Configuring TCPS with TLS for Sharding on OCI

To secure communications between sharded database components, Oracle recommends using Oracle Database native network encryption or the TCPS protocol and Transport Layer Security (TLS) for all connections to, and between, the shard catalog and the shards. When deploying using OCI Base Database Services, enable TLS using the information in Configure TCP/IP with SSL/TLS for Sharding – GSM OCI Mode (Doc ID 2881390.1).

For additional guidance on the use of wallets, configuring Transparent Data Encryption (TDE), and other security-related concepts and recommendations, see the Security in an Oracle Globally Distributed Database Environment documentation.

Configuring the Oracle Globally Distributed Database Topology on the Shard Director

After the databases for the shard catalog and all shards are configured, along with corresponding TNS listeners, you can add the sharding metadata to the shard catalog database using GDSCTL from the shard director. The sharding metadata describes the topology used for the sharded database.

We will run through an example of creating a system-managed sharded database topology without high availability.

These steps are also covered in the Configure the Oracle Globally Distributed Database Topology section of the documentation. There is also an Example Oracle Globally Distributed Database Deployment section that includes high availability with Oracle Active Data Guard standby replica shards in a second regional data center.

The high-level configuration process:

1. Create the shard catalog.
2. Add and start shard directors.
3. Add host metadata.
4. Add a primary shardgroup.
5. Verify the sharding topology.
6. Add the shard CDBs.
7. Add the shard PDBs.
8. Check free DB_FILES.
9. Deploy the sharding configuration.
10. Create and start global services.
11. Verify shard status.

Create the Shard Catalog

Use the `GDSCTL CREATE SHARDCATALOG` command to create metadata describing the sharded database topology in the shard catalog database.

As a best practice, it is recommended that you always set the AUTOVNCR parameter to OFF (default is ON) and specify the required hosts and IP addresses declaratively as "invited nodes" in a later step.

For the shard catalog PDB connection string, the entire string can be provided inline as the parameter value or provide a TNS alias for a connect string that has been added to ORACLE_HOME/network/admin/gsm.ora file.

If using Oracle Active Data Guard to provide an associated standby, this connection string should specify the primary and standby databases. See Configure the Connection String for High Availability in the Oracle Database High Availability Overview and Best Practices documentation.

Our example uses system-managed data distribution within a single region, with single-instance shard catalog and shard databases with no replication. See the Oracle Sharding documentation Create the Shard Catalog for practical examples.

Example: gdsctl create shardcatalog

```
$ gdsctl
GDSCTL> connect
gsmcatuser/gsmcatuser_credential@w23catyyz10.db.maasdbpoc.oraclevcn.com:1521/catpdb.db.maasdb
poc.oraclevcn.com
Catalog connection is established

GDSCTL> create shardcatalog  -user gsmcatuser/gsmcatuser_credential  -database
w23catyyz10.db.maasdbpoc.oraclevcn.com:1521/catpdb.db.maasdbpoc.oraclevcn.com  -region ca-
toronto-1  -configname oradbcloud  -sdb sharddb  -autovncr OFF  -chunks 120  -sharding system

Catalog is created
```

Add and Start a GSM Instance

Shard directors monitor the sharding system and run background tasks in response to `GDSCTL` commands and other events.

Add and start the GSM instance on each shard director host one-by-one, running the commands locally on the shard director host. If adding a shard director to an existing configuration, connect to the shard catalog database

before adding the new GSM. In our example, we are continuing the GDSCTL session started earlier for the initial configuration.

Example: gdsctl create and start GSM

```
GDSCTL> add gsm -gsm w23diryyz10  -catalog
w23catyyz10.db.maasdbpoc.oraclevcn.com:1521/catpdb.db.maasdbpoc.oraclevcn.com -region ca-
toronto-1 -listener 1522 -localons 6123 -remoteons 6234 -pwd gsmcatuser_credential
GSM successfully added

GDSCTL> start gsm -gsm w23diryyz10
GSM is started successfully

GDSCTL> config gsm
Name      Region    ENDPOINT
----      ------    --------
w23diryyz ca-toront (ADDRESS=(HOST=w23diryyz10.db.maasdbpoc.oraclevcn.com)(PORT
10        o-1       =1522)(PROTOCOL=tcp))
```

Add Host Metadata

Add the allowed host names and IP addresses of your shard hosts to the configuration. Oracle recommends creating your shard catalog with the automatic valid node checking for registration functionality (AUTOVNCR) disabled using the "-autovncr off" parameter value as shown in the example above.

As part of the deployment process, the shard director contacts the shards and directs them to register with the shard director's TNS listener process. This listener process only accepts incoming registration requests from trusted sources and rejects registration requests from unknown hosts.

If your shard hosts have multiple host names or network interfaces assigned to them, it is possible that the incoming registration request from the shard to the shard director may come from an address that was not automatically added during the `GDSCTL ADD SHARD` operation. In this case, the registration request is rejected, and the shard will not deploy correctly. The visible symptom of this problem will be that `GDSCTL CONFIG SHARD` shows PENDING for the shard's Availability after `DEPLOY` has completed.

When deploying using Oracle RAC, the IPs for the Public IP and Floating VIP for each RAC instance should be added. The SCAN VIP and FQDN do not need to be included here.

Use the `GDSCTL ADD INVITEDNODE` command to manually add the required entries for your shard hosts to the shard catalog metadata.

See the Add Host Metadata documentation for complete details and examples.

Example: gdsctl add invitednode - Entries for four shards – IP Address and hostnames

```
add invitednode 10.8.208.94
add invitednode w23shdyyz10
add invitednode w23shdyyz10.db.maasdbpoc.oraclevcn.com

add invitednode 10.8.208.78
add invitednode w23shdyyz11
add invitednode w23shdyyz11.db.maasdbpoc.oraclevcn.com

add invitednode 10.8.208.89
add invitednode w23shdyyz12
add invitednode w23shdyyz12.db.maasdbpoc.oraclevcn.com
```

```
add invitednode 10.8.208.72
add invitednode w23shdyyz13
add invitednode w23shdyyz13.db.maasdbpoc.oraclevcn.com
```

After adding everything, run `gdsctl config vncr` to validate all host and IP entries are included.

Add Shardgroups

When using system-defined sharding, configure your desired sharding topology using the `ADD SHARDGROUP` command.

Multiple shardgroup types for primary, standby, and active_standby databases can be declared using the `-deploy_as` parameter. The default is "`standby`".

Ensure that each shard database is opened with the correct status before adding the shards to the new shardgroup. Any shards subsequently added to the shardgroup must be opened in the mode corresponding to the `-deploy_as` setting for the shardgroup. For example, read-write for primary shardgroups, mounted for standby shardgroups, or read-only with apply for active standby shardgroups.

Example: gdsctl create shardgroup - Primary

```
GDSCTL> add shardgroup -shardgroup shardgroup_primary -shardspace shardspaceora -deploy_as
primary -region ca-toronto-1
The operation completed successfully
```

For our example, no replication for HA or disaster-recovery was implemented, so all shards are configured in the default primary shardspace. See the Add Shardspaces documentation for more details and examples.

Verify the Sharding Topology

Before adding information about your shard databases to the shard catalog, verify that your sharding topology is correct before proceeding by using the various `GDSCTL CONFIG` commands.

You can use the various `GDSCTL CONFIG` commands to display more information about shardspaces, shardgroups, and other shard catalog objects. For a complete list of `GDSCTL CONFIG` command variants, see the `GDSCTL` reference documentation or run `GDSCTL HELP`.

Note: there will be no databases or services listed at this point.

Example: Validate the configuration with gdsctl status and config commands

```
GDSCTL> status
Alias                     W23DIRYYZ10
Version                   23.0.0.0.0
Start Date                09-MAY-2024 16:36:04
Trace Level               off
Listener Log File         /u01/app/oracle/diag/gsm/w23diryyz10/w23diryyz10/alert/log.xml
Listener Trace File
/u01/app/oracle/diag/gsm/w23diryyz10/w23diryyz10/trace/ora_949066_139912260905920.trc
Endpoint summary
(ADDRESS=(HOST=w23diryyz10.db.maasdbpoc.oraclevcn.com)(PORT=1522)(PROTOCOL=tcp))
GSMOCI Version            4.0.191114
Mastership                Y
Connected to GDS catalog  Y
Process Id                949068
Number of reconnections   0
Pending tasks.     Total  0
Tasks in  process. Total  0

Regional Mastership       TRUE
Total messages published  0
Time Zone                 +00:00
Orphaned Buddy Regions:
     None
GDS region                ca-toronto-1
```

```
GDSCTL> config
Catalog connection is established

Regions
------------------------
ca-toronto-1

GSMs
------------------------
w23diryyz10

Sharded Database
------------------------
sharddb

Databases
------------------------

Shard Groups
------------------------
shardgroup_primary

Shard spaces
------------------------
shardspaceora

Services
------------------------

GDSCTL pending requests
------------------------
Command                       Object                        Status
-------                       ------                        ------

Global properties
------------------------
Name: oradbcloud
Master GSM: w23diryyz10
DDL sequence #: 0
```

Add Shard CDBs

Add each of the CDBs containing the shard PDBs to the sharding configuration with the `ADD CDB` command. Include the connect string or TNS alias along with the `gsmrootuser` credential. This is a different database schema user credential than what was used earlier for the shard catalog connection. Use the `CONFIG CDB` command when you are done to verify that all CDBs have been added. See the Add the Shard CDBs documentation for complete details and examples.

The connect string for each shard should consist of the simple, direct connection details. GDS deals with each shard CDB individually and does not require a DR-enabled connection string. When using OCI Base Database System databases, the CDB connection strings will use the database unique name fully qualified with the system's domain name.

Example: Add Shard CDBs

```
GDSCTL> add cdb -connect
w23shdyyz10.db.maasdbpoc.oraclevcn.com:1521/shdcdb0_3s2_yyz.db.maasdbpoc.oraclevcn.com -pwd
gsmrootuser_password

GDSCTL> add cdb -connect
w23shdyyz11.db.maasdbpoc.oraclevcn.com:1521/shdcdb1_224_yyz.db.maasdbpoc.oraclevcn.com -pwd
gsmrootuser_password

GDSCTL> add cdb -connect
w23shdyyz12.db.maasdbpoc.oraclevcn.com:1521/shdcdb2_542_yyz.db.maasdbpoc.oraclevcn.com -pwd
gsmrootuser_password

GDSCTL> add cdb -connect
w23shdyyz13.db.maasdbpoc.oraclevcn.com:1521/shdcdb3_pqd_yyz.db.maasdbpoc.oraclevcn.com -pwd
gsmrootuser_password

GDSCTL> config cdb
Catalog connection is established
Name                           Host
----                           ----
shdcdb0_3s2_yyz
shdcdb1_224_yyz
shdcdb2_542_yyz
shdcdb3_pqd_yyz
```

Note that the `config cdb` output may not show a host name in all cases. This is not an issue.

Add Shard PDBs

Add each of the shard PDBs to the sharding configuration with the `ADD SHARD` command.

Include parameters for the shard PDB connect string or TNS alias, with the gsmuser credential, the shardgroup name, and cdb name. Use the `CONFIG SHARD` command when you are done to verify that all PDBs have been added.

See the Add the Shard PDBs documentation for complete details and examples.

Example: Add Shard PDBs

```
GDSCTL> add shard -connect
w23shdyyz10.db.maasdbpoc.oraclevcn.com:1521/shdpdb0.db.maasdbpoc.oraclevcn.com -shardgroup
shardgroup_primary -cdb shdcdb0_3s2_yyz -pwd gsmuser_credential

GDSCTL> add shard -connect
w23shdyyz11.db.maasdbpoc.oraclevcn.com:1521/shdpdb1.db.maasdbpoc.oraclevcn.com -shardgroup
shardgroup_primary -cdb shdcdb1_224_yyz -pwd gsmuser_credential

GDSCTL> add shard -connect
w23shdyyz12.db.maasdbpoc.oraclevcn.com:1521/shdpdb2.db.maasdbpoc.oraclevcn.com -shardgroup
shardgroup_primary -cdb shdcdb2_542_yyz -pwd gsmuser_credential

add shard -connect
w23shdyyz13.db.maasdbpoc.oraclevcn.com:1521/shdpdb3.db.maasdbpoc.oraclevcn.com -shardgroup
shardgroup_primary -cdb shdcdb3_pqd_yyz -pwd gsmuser_credential

GDSCTL> config shard
Name                Shard Group         Status    State       Region    Availability
----                -----------         ------    -----       ------    ------------
shdcdb0_3s2_yyz_shd shardgroup_primary  U         none        ca-toront none
pdb0                                                          o-1
shdcdb1_224_yyz_shd shardgroup_primary  U         none        ca-toront none
pdb1                                                          o-1
shdcdb2_542_yyz_shd shardgroup_primary  U         none        ca-toront none
pdb2                                                          o-1
shdcdb3_pqd_yyz_shd shardgroup_primary  U         none        ca-toront none
pdb3                                                          o-1
```

Note that the value for Status is `U` for “undeployed”, and State and Availability are `none` and – until the `DEPLOY` command is successfully run. Also, the column lengths are hardcoded.

Deploy the Sharding Configuration

Once the shards have been added, run the GDSCTL DEPLOY command. This will request the shard catalog to deploy any shards recently added with an undeployed status.

Depending on your topology, the output may vary significantly. See Deploy the Configuration documentation for additional examples and advice about what to do if something is not right.

Example: gdsctl deploy

```
GDSCTL> deploy
deploy: examining configuration...
deploy: requesting Data Guard configuration on shards via GSM
deploy: shards configured successfully
The operation completed successfully
```

```
GDSCTL> config shard
Name                 Shard Group          Status    State       Region    Availability
----                 -----------          ------    -----       ------    ------------
shdcdb0_3s2_yyz_shd shardgroup_primary  Ok        Deployed    ca-toront ONLINE
pdb0                                                           o-1
shdcdb1_224_yyz_shd shardgroup_primary  Ok        Deployed    ca-toront ONLINE
pdb1                                                           o-1
shdcdb2_542_yyz_shd shardgroup_primary  Ok        Deployed    ca-toront ONLINE
pdb2                                                           o-1
shdcdb3_pqd_yyz_shd shardgroup_primary  Ok        Deployed    ca-toront ONLINE
pdb3                                                           o-1
```

Create and Start Global Database Services

After the shards are successfully deployed, and the correct status has been confirmed, create and start global database services on the shards to service incoming connection requests from your application.

The commands in the first example creates read-write services on the primary shards in the configuration. If you have standby shards, read-only services can be created on the standby shards. These service names can then be used in connect strings from your application to appropriately route requests to the correct shards. See Create and Start Global Database Services in the documentation for more details.

Example: Add and start a global service that runs on the primary shards (non-RAC)

```
GDSCTL> add service -service oltp_rw_srvc -role primary
GDSCTL> start service -service oltp_rw_srvc
```

Example: Add and start a global service for the read-only workload to run on the standby shards (non-RAC)

```
GDSCTL> add service -service oltp_ro_srvc -role physical_standby
GDSCTL> start service -service oltp_ro_srvc
```

Note: If you are using an Oracle RAC database, each service must be modified to set the preferred Oracle RAC instances before starting the service. When using Administrator Managed RAC (not Policy Managed), the services must be modified for every RAC node and include all instances the service *might* run on. This includes the primary RAC as well as any Oracle Active Data Guard standbys. Open the standby instances in read-only mode before configuring, modifying, and starting the services. Mounted, non-open standbys exist for disaster recovery and high availability purposes only and cannot service connections.

Note: Database instance names for the `-preferred` parameter are case-sensitive and must match the instance name db parameter value from the '`show parameters instance_name`' SQL statement.

Example: Modify service to set preferred Oracle RAC instances

```
GDSCTL> add service -service oltp_rw_srvc -role primary
GDSCTL> modify service -gdspool poc -service oltp_rw_srvc -database poc_iad1f9_usshard -
modify_instances -preferred POC1,POC2
GDSCTL> modify service -gdspool poc -service oltp_rw_srvc -database poc_yyz173_usshard -
modify_instances -preferred POC1,POC2
GDSCTL> start service -service oltp_rw_srvc
```

Finally, verify the configuration now that the setup is complete.

```
GDSCTL> config
Catalog connection is established

Regions
------------------------
ca-toronto-1

GSMs
------------------------
w23diryyz10

Sharded Database
------------------------
sharddb

Databases
------------------------
shdcdb0_3s2_yyz_shdpdb0
shdcdb1_224_yyz_shdpdb1
shdcdb2_542_yyz_shdpdb2
shdcdb3_pqd_yyz_shdpdb3

Shard Groups
------------------------
shardgroup_primary

Shard spaces
------------------------
shardspaceora

Services
------------------------

GDSCTL pending requests
------------------------
Command                      Object                       Status
-------                      ------                       ------

Global properties
------------------------
Name: oradbcloud
Master GSM: w23diryyz10
DDL sequence #: 0

GDSCTL> config service
Name           Network name                 Pool           Started Preferred all
----           ------------                 ----           ------- -------------
oltp_rw_srvc   oltp_rw_srvc.sharddb.oradbclo sharddb        Yes     Yes
               ud
```

Verify that the databases have all registered correctly and have their services started properly. Databases should show successfully registered and an “OK” state. Services for each database should all be Globally started, Started, Enabled, and Preferred. Validate these parameters have a “Y” value.

```
GDSCTL> databases
Database: "shdcdb0_3s2_yyz_shdpdb0" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances:
1 Region: ca-toronto-1
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     sharddb%1
Database: "shdcdb1_224_yyz_shdpdb1" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances:
1 Region: ca-toronto-1
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     sharddb%11
Database: "shdcdb2_542_yyz_shdpdb2" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances:
1 Region: ca-toronto-1
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     sharddb%21
Database: "shdcdb3_pqd_yyz_shdpdb3" Registered: Y State: Ok ONS: N. Role: PRIMARY Instances:
1 Region: ca-toronto-1
   Service: "oltp_rw_srvc" Globally started: Y Started: Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     sharddb%31
```

Validate the completed sharded database configuration for any warnings or errors.

```
GDSCTL> validate
Validation results:

[Warning] VLD2: Region "ca-toronto-1" does not have buddy region
[Warning] VLD63:  No sharded table family exists.

 Total errors: 0. Total warnings:2
```

The warnings above in the final “validate” output are expected at this point. We have only set up one region, with no plans for cross-region replication in our example, and our next step is to create the sharded table family.

Creating the Sharded Database Schema

There are several steps and specific details critical to creating schema objects in a sharded database. See the Creating Schema Objects section of the Oracle Globally Distributed Database Guide for full specifications.

Our sharded table family consists of a single table named "Articles" with the Wikipedia Articles JSON data stored in a JSON column named "Article_JSON". Our sharded table family is system-managed with a sharding key matching the primary key, both using the Article_ID column. The primary/sharding key value for the Article_ID column is extracted during data load from the Wikipedia article JSON document content $.wid element.

To make some of our SQL queries more efficient with relational operators and reduced complexity, virtual columns are configured to abstract some JSON elements value lookups. Notably, we extract certain elements using JSON_VALUE() queries from Article_JSON and in some cases do the type conversions centrally as part of

the virtual column specification. Virtual columns are created to include the following JSON elements in the Wikipedia article documents: .wid, .title, .wikiTitle, .lang, and .timestamp.

The steps for deploying a new schema with some additional details necessary in a sharded database environment can be summarized as follows:

1. Start a SQL session, connect to the shard catalog PDB as SYSDBA.
2. Alter the session to enable shard DDL so the schema deployment occurs on all shards, not just the shard catalog database.
3. Create the all-shards schema user.
4. Grant roles and permissions.
5. Reconnect to the shard catalog database as the new schema user.
6. Alter the session to enable shard DDL to ensure that the schema deployment occurs on all shards, not just the shard catalog database.
7. Create the tablespace set and set quotas appropriately.
8. Create the sharded table family, specifying the primary key, tablespace set, and partitioning scheme with the table column to use as the sharding key.
9. Create functional indexes needed (the OracleText index will be created in its own chapter after data is loaded).

Before we begin, a note on sharded DDL execution:

When using SQL*Plus to execute DDL on the shard catalog PDB with shard DDL enabled on your session, the SQL execution will return completed once it has been executed on the shard catalog. Two additional steps are required to know if execution on the shards has completed and if there were any errors on any of the shards before continuing with additional DDL statements.

To ultimately know if it is safe to execute your next DDL statement:

1. In between each DDL request, you must execute the gsmadmin_internal.dbms_gsm_utility.wait_for_ddl() function in your catalog PDB session to ensure that execution is complete on all shards before continuing. This function will wait for the execution to complete on all shards before returning control in SQL*Plus. You will not be informed whether or not the SQL completed successfully or had errors on the shards, however.
2. To check for errors during DDL execution across all the shards, use the GDSCTL SHOW DDL command on a shard director. This will not show query completion, only errors.

To avoid this complexity, SQL can be executed using the GDSCTL command line tool from a shard director host. See the DDL Processing in an Oracle Globally Distributed Database section of the documentation.

For example:

```
[oracle@sharddirector1] $ gdsctl
GDSCTL> connect sys@catalog.fqdn:1521/catpdb as sysdba
password:
Catalog connection is established

GDSCTL> sql “GRANT CONNECT, RESOURCE, ALTER SESSION TO wikip”
The operation completed successfully

GDSCTL> show ddl -count 1
id      DDL Text                                 Failed shards
--      --------                                 -------------
113     GRANT CONNECT, RESOURCE, ALTER SESSIO...
```

For the purposes of our examples, we used SQL*Plus.

Example DDL to create a schema user from the shard catalog via SQL*Plus:

DDL: Create Sharded Schema User via Shard Catalog Database

```
REM Connect to Catalog Database with SYS as SYSDBA
SQL> CONNECT sys@catpdb as sysdba;

-- Set the disposition of shard DDL and/or shard operations for the session\
ALTER SESSION ENABLE SHARD DDL;

-- Drop our schema user to be safe.
DROP USER wikip CASCADE;
exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();

-- Create our schema user
CREATE USER wikip IDENTIFIED BY "RedactedCredentialString";
exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();
```

After each use of the wait_for_ddl() function, when executing DDL from the shard catalog database, always confirm that the DDL has not failed on any shard before continuing. Use the GDSCTL command on a shard director to list the DDL executed and indicate whether that DDL failed on any shards.

Example use of GDSCTL to confirm status of DDL completion on all shards:

GSM: Check DDL status for failed shards

```
# Now check DDL status via GDSCTL on Shard Director for success or faulted shards
# Use the -count parameter to reduce the data presented to just the latest DDL as
needed.

[oracle@sharddirector1] $ gdsctl show ddl -count 1

id      DDL Text                                 Failed shards
--      --------                                 -------------
88      CREATE USER wikip IDENTIFIED BY...
```

Continue with granting the necessary roles and permissions. The roles and permissions necessary may differ based on your specific requirements.

DDL: Grant example roles and permissions:

```
SQL>
-- Grant Roles to the new schema user and wait for shards to catch up.
GRANT CONNECT, RESOURCE, ALTER SESSION TO wikip;
GRANT CREATE TABLE, CREATE PROCEDURE, CREATE TABLESPACE, CREATE MATERIALIZED VIEW TO
wikip;
GRANT GSMADMIN_ROLE TO wikip;
GRANT GSM_POOLADMIN_ROLE TO wikip;
GRANT ANALYZE ANY TO wikip;
GRANT SELECT ANY DICTIONARY TO wikip;
GRANT SELECT ANY TABLE TO wikip;
GRANT SELECT_CATALOG_ROLE TO wikip;
GRANT EXECUTE ON DBMS_CRYPTO TO wikip;
GRANT EXECUTE ON DBMS_LOCK TO wikip;
GRANT EXECUTE ON DBMS_SODA TO wikip;
GRANT EXECUTE ON DBMS_SODA_ADMIN TO wikip;
GRANT EXECUTE ON DBMS_SYS_ERROR TO wikip;
GRANT EXECUTE ON EXEC_SHARD_PLSQL TO wikip;
GRANT CTXAPP TO wikip;
-- Grant execute privs on PL/SQL packages for CTX packages
-- These are also granted by ctxapp role, but role perms do not always work in PL/SQL
procedures
-- It is safest to explicitly grant these permissions to the user who already has the
CTXAPP role.
GRANT EXECUTE ON CTXSYS.CTX_ADM TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_CLS TO wikip;
```

```
GRANT EXECUTE ON CTXSYS.CTX_DDL TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_DOC TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_OUTPUT TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_QUERY TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_REPORT TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_THES TO wikip;
GRANT EXECUTE ON CTXSYS.CTX_ULEXER TO wikip;

exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();

GDSCTL> show ddl -count 24
Catalog connection is established
id      DDL Text                                 Failed shards
--      --------                                 -------------
89      GRANT CONNECT, RESOURCE, ALTER SES...
90      GRANT CREATE TABLE, CREATE PROCEDU...
91      GRANT GSMADMIN_ROLE TO wikip;     ...
92      GRANT GSM_POOLADMIN_ROLE TO wikip;...
93      GRANT ANALYZE ANY TO wikip;       ...
94      GRANT SELECT ANY DICTIONARY TO wik...
95      GRANT SELECT ANY TABLE TO wikip;  ...
96      GRANT SELECT_CATALOG_ROLE TO wikip...
97      GRANT EXECUTE ON DBMS_CRYPTO TO wi...
98      GRANT EXECUTE ON DBMS_LOCK TO wiki...
99      GRANT EXECUTE ON DBMS_SODA TO wiki...
100     GRANT EXECUTE ON DBMS_SODA_ADMIN T...
101     GRANT EXECUTE ON DBMS_SYS_ERROR TO...
102     GRANT EXECUTE ON EXEC_SHARD_PLSQL ...
103     GRANT CTXAPP TO wikip;            ...
104     GRANT EXECUTE ON CTXSYS.CTX_ADM TO...
105     GRANT EXECUTE ON CTXSYS.CTX_CLS TO...
106     GRANT EXECUTE ON CTXSYS.CTX_DDL TO...
107     GRANT EXECUTE ON CTXSYS.CTX_DOC TO...
108     GRANT EXECUTE ON CTXSYS.CTX_OUTPUT...
109     GRANT EXECUTE ON CTXSYS.CTX_QUERY ...
110     GRANT EXECUTE ON CTXSYS.CTX_REPORT...
111     GRANT EXECUTE ON CTXSYS.CTX_THES T...
112     GRANT EXECUTE ON CTXSYS.CTX_ULEXER...
```

Once the schema user has been created on all shards successfully, reconnect to the shard catalog database as the new schema user to execute DDL for creating the tablespace set and table family for the sharded database implementation.

Note that in a sharded database, each individual shard database will contain a fraction (1/shards) of the data set payload. Partitions and data files will be created for each "chunk" as declared when the shardspace is defined. The default number of chunks is 120. In a four-shard environment, there will be 120 partitions per database instance with 30 of them populated with data and the other 90 partition's data files empty.

When planning and configuring the tablespace for the sharded table family, the data file size specified in the DDL is used for each (populated) partition's data file. With 4 shards and 120 chunks, and a data file maxsize of 20G, the new tablespace set will consume up to 620GB of storage. Size this sufficiently for the expected data import and indexing payloads.

Example DDL and GSM commands to create the tablespace set and table family is as follows:

DDL: Create Sharded Tablespace Set

```
REM Connect to Catalog and Shard Databases with the WIKIP schema user
CONNECT wikip@catpdb;

-- Set the disposition of shard DDL and/or shard operations for the session
```

```
ALTER SESSION ENABLE SHARD DDL;

-- Create our Tablespace Set and wait for shards to catch up.
-- 31 chunks per shard @ MAXSIZE 20G ~= 620GB
CREATE BIGFILE TABLESPACE SET TSP_SET_WIKIP
  USING TEMPLATE (
    DATAFILE SIZE 10G
    AUTOEXTEND ON NEXT 250M maxsize 20G
    EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO
  );

exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();
```

Verify that the create tablespace set DDL has completed on all shards.

GSM: Check DDL status for failed shards

```
# Now check DDL status via GDSCTL on Shard Director for success or faulted shards
# Use the -count parameter to reduce the data presented to just the latest DDL as
needed.

[oracle@sharddirector1] $ gdsctl show ddl -count 1

id      DDL Text                                 Failed shards
--      --------                                 -------------
113     CREATE BIGFILE TABLESPACE SET...
```

Alter the new schema user's quota on the new tablespace set to be unlimited, connecting as sysdba.

DDL: Alter User to set Unlimited Quota on the tablespace family

```
REM Connect to Catalog and Shard Databases with the WIKIP schema user
CONNECT sys@catpdb as sysdba;

-- Set the disposition of shard DDL and/or shard operations for the session
ALTER SESSION ENABLE SHARD DDL;

ALTER USER wikip QUOTA UNLIMITED ON TSP_SET_WIKIP;
exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();
```

Verify that the Alter User DDL has completed on all shards.

GSM: Check DDL status for failed shards

```
# Now check DDL status via GDSCTL on Shard Director for success or faulted shards
# Use the -count parameter to reduce the data presented to just the latest DDL as
needed.

[oracle@sharddirector1] $ gdsctl show ddl -count 1

id      DDL Text                                 Failed shards
--      --------                                 -------------
90       ALTER USER wikip QUOTA UNLIMITED ON...
```

Create the Table Family

In our example we use the Article_ID column as both the primary key and the sharding key. The sharding key and use of system-managed data distribution is specified in the PARTITION BY CONSISTENT HASH clause. We have

also included several virtual columns to support the use of specific JSON attribute values in queries with relational query operators.

DDL: Create Sharded Table Family

```
REM Connect to Catalog and Shard Databases with the WIKIP schema user
CONNECT wikip@catpdb;

-- Set the disposition of shard DDL and/or shard operations for the session
ALTER SESSION ENABLE SHARD DDL;
-- Create the table family, dropping if it already exists
-- Wait for the drop and create executions to complete on all shards before continuing.
DROP TABLE ARTICLES CASCADE CONSTRAINTS PURGE;
exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();
CREATE SHARDED TABLE Articles
(
  Article_JSON           JSON            NOT NULL,
  Article_ID             VARCHAR(10)     NOT NULL,
  Date_Loaded            TIMESTAMP(6)    DEFAULT SYS_EXTRACT_UTC(SYSTIMESTAMP) NOT NULL,
  Title                  VARCHAR(512)    AS ( JSON_VALUE(Article_JSON, '$.title'
                                              returning varchar(512) null on error
                                              null on empty) ),
  Wiki_Title             VARCHAR(512)    AS ( JSON_VALUE(Article_JSON, '$.wikiTitle'
                                              returning varchar(512) null on error
                                              null on empty) ),
  Lang                   VARCHAR(2)      AS ( JSON_VALUE(Article_JSON, '$.lang'
                                              returning varchar(2) null on error
                                              null on empty) ),
  Last_Modified          TIMESTAMP(0) WITH TIME ZONE AS ( TO_TIMESTAMP_TZ(
                         JSON_VALUE(Article_JSON, '$.timestamp' null on error null on empty),
                         'YYYY-MM-DD"T"HH24:MI:SS TZH:TZM') ),
  PRIMARY KEY (Article_ID)
)
TABLESPACE SET tsp_set_wikip
PARTITION BY CONSISTENT HASH (Article_ID) PARTITIONS AUTO;

SQL> exec gsmadmin_internal.dbms_gsm_utility.wait_for_ddl();
```

GSM: Check DDL status for shard status

```
# Now check DDL status via GDSCTL on Shard Director for success
# Use the -count parameter to reduce the data presented to just the latest DDL as
needed.

[oracle@sharddirector1] $ gdsctl show ddl -count 2

id      DDL Text                                 Failed shards
--      --------                                 -------------
91      DROP TABLE ARTICLES CASCADE CONSTRAIN...
92      CREATE SHARDED TABLE Articles (Articl...
```

With the table family created, describe the table and verify the virtual columns using the same SQL session.

```
SQL> desc articles

Name                          Null?    Type
```

```
-------------------------  -------- ------------------------------
 ARTICLE_JSON              NOT NULL JSON
 ARTICLE_ID                NOT NULL VARCHAR2(10)
 DATE_LOADED               NOT NULL TIMESTAMP(6)
 TITLE                              VARCHAR2(512)
 WIKI_TITLE                         VARCHAR2(512)
 LANG                               VARCHAR2(2)
 LAST_MODIFIED                      TIMESTAMP(0) WITH TIME ZONE

SQL> -- Query for virtual column details
SQL> col column_name  format a48;
SQL> col data_default format a80;
SQL> SELECT
       column_name,
       virtual_column,
       data_default
  FROM
       all_tab_cols
 WHERE owner = 'WIKIP'
   AND column_name not like 'SYS%';


COLUMN_NAME       VIR DATA_DEFAULT
----------------- --- --------------------------------------------------------------------
ARTICLE_JSON       NO
ARTICLE_ID         NO
DATE_LOADED        NO SYS_EXTRACT_UTC(SYSTIMESTAMP)
TITLE             YES JSON_VALUE("ARTICLE_JSON" FORMAT OSON , '$.title' RETURNING VARCH
WIKI_TITLE        YES JSON_VALUE("ARTICLE_JSON" FORMAT OSON , '$.wikiTitle' RETURNING V
LANG              YES JSON_VALUE("ARTICLE_JSON" FORMAT OSON , '$.lang' RETURNING VARCHA
LAST_MODIFIED     YES TO_TIMESTAMP_TZ(JSON_VALUE("ARTICLE_JSON" FORMAT OSON , '$.timest
```

Bulk Loading the JSON Data

Loading JSON documents into the schema in bulk from existing JSON data files involves staging the required data file on each shard, then declaring an external table and using Oracle SQL*Loader to extract the primary/sharding key values from the JSON records, and then insert the data as selected from the external table into the sharded table family. This can be performed in parallel on all shards simultaneously. Only the JSON documents with sharding keys having hashed values within the range for each shard will be loaded on that shard.

Note that when loading data, the only columns that must be inserted include the JSON column and the Primary Key/Sharding key. All other columns in our example table are virtual columns and their values will be derived from the JSON column based on the virtual column declaration.

Deploy the JSON Data File

Data can be made available for loading from a central JSON file into each shard database individually using an external table. The JSON data file can be loaded directly on to each shard's local file system, accessed from a shared storage location or OCI Object Storage.

In our example, we have simply copied the data file to each of the shards temporarily and exposed it as an external table. This is performed on all shards. We load the data file `enwiki.json` to the `/u01/wikip/data` folder on each shard database server. This folder can be local or on a shared file system mounted to all shards. Once the folder exists, it can be made available to the shard databases as follows while connected as sys with sysdba privileges at the shard PDBs.

Example code: Registering the shard data directory

```
SQL> ALTER SESSION DISABLE SHARD DDL;
SQL> DROP DIRECTORY wikip_data_dir;
SQL> CREATE OR REPLACE DIRECTORY wikip_data_dir AS '/u01/wikip/data';
SQL> GRANT ALL on DIRECTORY wikip_data_dir TO wikip;
```

Once the directory is configured, we can reference the data file in that directory using Oracle Loader with an external table while connected as the all-shards user, in our case “wikip”.

The length in bytes of the longest individual JSON document records within the data file must be used to tune the READSIZE access parameter in the following statement, so that the Oracle Loader can accurately parse long JSON document records.

```
SQL> CONNECT WIKIP@SHDPDB0
SQL> ALTER SESSION DISABLE SHARD DDL;
SQL> DROP   TABLE Articles_Ext;
SQL> CREATE TABLE Articles_Ext
            (Article_JSON JSON NOT NULL)
     ORGANIZATION EXTERNAL(
        TYPE ORACLE_LOADER
        DEFAULT DIRECTORY wikip_data_dir
        ACCESS PARAMETERS (
           RECORDS DELIMITED BY 0x'0A'
           READSIZE 444100
           BADFILE wikip_data_dir: 'JSON_DUMPFILE_CONTENTS.bad'
           LOGFILE wikip_data_dir: 'JSON_DUMPFILE_CONTENTS.log'
           FIELDS(Article_JSON char(444100))
         )
     LOCATION (wikip_data_dir:'enwiki.json')
)
PARALLEL
REJECT LIMIT UNLIMITED;

-- Count the rows to verify the entire data set is available
select count(*) from articles_ext;
```

Load the JSON Documents

Once the JSON data is available as a JSON column in an external table, we can query for insert into the sharded table family. This query can be executed in parallel directly on each shard. The insert statement populates the primary key "article_id" and "article_json" columns only. Both the "article_id" column value and the sharding key value are set using the JSON_VALUE() function to extract the $.wid element value from each JSON record. This concept applies to any inserted record during application runtime as well. The submission of a new JSON document must also submit a value for the “article_id” primary key column and can specify the same value as the sharding key.

Note: Once this section is complete and data load is validated, the external table can be dropped.

Execute this in parallel on all shards as the all-shards user.

```
SQL> CONNECT WIKIP@SHDPDB0
SQL> ALTER SESSION DISABLE SHARD DDL;
SQL> ALTER SESSION ENABLE SHARD OPERATIONS;
SQL> ALTER SESSION ENABLE PARALLEL DML;
SQL> INSERT /*+ NOLOGGING APPEND PARALLEL (6) */ INTO ARTICLES (article_id, article_json)
  (SELECT json_value(ext.article_json,'$.wid'), ext.article_json
     FROM articles_ext ext
     WHERE SHARD_CHUNK_ID('WIKIP.ARTICLES',
                          json_value(ext.article_json,'$.wid') ) IS NOT NULL
  );
SQL> COMMIT;
```

The INSERT AS SELECT statement can utilize parallel execution based on the number of CPU threads available to the database by enabling parallel DML and including the appropriate PARALLEL(x) hint on the INSERT statement.

It may not be ideal to use all CPU cores for parallel processing of this query, and it may not be ideal to leave the default value of "1" either. Tune the degree of parallelism (DOP) of this insert as necessary based on your database instance's hardware resource availability. It is also a best practice to disable logging during bulk inserts for performance reasons.

Finally, to ensure only the correct rows are inserted for each shard, it is necessary to include a WHERE clause using the SHARD_CHUNK_ID() function to evaluate the system-managed hash of the shard key value for each row, and decide if that row belongs in a chunk on this shard. The SHARD_CHUNK_ID() function requires two values: the name of the table family, and the sharding key value. For details about the SHARD_CHUNK_ID() function, see the Oracle 23ai SQL Language Reference Guide.

Currently, it is recommended that when implementing a sharded database with system-managed data distribution to have the entire data file available to every shard to allow processing of all rows via the external table consistently. It is up to the SHARD_CHUNK_ID function to distinguish which rows have a shard ID value that is valid to insert for the chunks assigned to the current shard, based on the value of the Oracle consistent hash model. See System-Managed Sharding in the Oracle Globally Distributed Database documentation for details.

Collect Statistics

Once the data has been inserted completely on all shards, collect schema statistics for the sharded schema user.

There are several steps required to accurately collect statistics in a sharded database. Statistics need to be gathered at each shard and "pulled" up to the coordinator (shard catalog database). To allow the coordinator to pull the statistics gathered on the shards, the statistic preference parameter `COORDINATOR_TRIGGER_SHARD` must be set to true on all shard databases.

Note: When deploying into a multi-tenant database environment, these steps should be performed against the shard catalog and shard pluggable databases (PDB).

The steps for manual statistics gathering on sharded tables include:

1. Set COORDINATOR_TRIGGER_SHARD to TRUE on all shard PDBs as sys on behalf of your all-shards schema user. This only needs to be done once per shard.

   ```
   SQL> CONNECT sys/shdpdb0 AS SYSDBA
   SQL> ALTER SESSION DISABLE SHARD DDL;
   SQL> EXECUTE DBMS_STATS.SET_SCHEMA_PREFS('WIKIP','COORDINATOR_TRIGGER_SHARD','TRUE');
   ```

2. Collect schema statistics for your all-shards user on all shard PDBs connected as your all-shards schema user. The `SYS.DBMS_STATS.GATHER_SCHEMA_STATS()` procedure can be executed directly on each shard, or on all shards at once from the shard catalog using the `SYS.EXEC_SHARD_PLSQL()` function with `SHARD DDL` enabled on your session.

```
SQL> CONNECT wikip@catpdb
SQL> ALTER SESSION ENABLE SHARD DDL;
SQL> EXEC SYS.EXEC_SHARD_PLSQL('SYS.DBMS_STATS.GATHER_SCHEMA_STATS(
                                OWNNAME => ''WIKIP'', OPTIONS => ''GATHER'')');
```

3. Gather statistics on the shard catalog PDB for your schema after statistics have been gathered on all of the shards.

```
SQL> CONNECT wikip@catpdb
SQL> ALTER SESSION DISABLE SHARD DDL;
SQL> EXEC SYS.DBMS_STATS.GATHER_SCHEMA_STATS(OWNNAME => 'WIKIP', OPTIONS => 'GATHER');
```

4. Check statistics on each shard.

```
SQL> CONNECT wikip@shdcdb0
SQL> ALTER SESSION DISABLE SHARD DDL;
SQL> SELECT t.table_name AS "Table Name",
      t.num_rows AS "Rows",
       t.avg_row_len AS "Avg Row Len",
       Trunc((t.blocks * p.value)/1024) AS "Size KB",
       sharded,
       duplicated,
       to_char(t.last_analyzed,'DD/MM/YYYY HH24:MM:SS') AS "Last Analyzed"
FROM   user_tables t,
       v$parameter p
WHERE table_name not like 'MLOG%'
  AND table_name not like 'RUPD%'
  AND table_name not like 'USLOG%'
  AND table_name not like 'DR$%'
  AND p.name = 'db_block_size'
ORDER by t.table_name
/

Table Name                     Rows Avg Row Len    Size KB S D Last Analyzed
------------------------ ---------- ----------- ---------- - - -------------------
ARTICLES                    1524331        2860    5952480 Y N 30/05/2024 10:05:27
ARTICLES_EXT                                               N N
```

Once statistics have been gathered and validated, automatic statistics gathering can be configured. For full details, including how to configure automatic statistics gathering on the sharded database, see Gathering Optimizer Statistics on Sharded Tables. Any additional tuning of statistics gathering parameters and the related optimizations and tuning of maintenance plans is beyond the scope of this example and can be integrated as needed.

Indexing Sharded JSON Data

To optimize the access to our data, several indexes are created. We will demonstrate the creation of functional indexes on the primary key and a multivalue index of the array of categories for each JSON document. Then we will create an Oracle Text search index on the full text of the values within each JSON document. The multivalue index will demonstrate the use of the JSON database features and help optimize the execution of search results

queries that filter based on category facet values. The Oracle Text search index will support contextual full text search of the element values from the JSON document rather than matching against the JSON code.

These indexes are created from the shard catalog, or from the shard director, using the `GDSCTL SQL` command, ensuring that the index is created on every shard. Verify each DDL execution using `GDSCTL SHOW DDL` on the shard director to detect any failures. When PL/SQL is needed, the SYS.EXEC_SHARD_PLSQL() function can be used from the shard catalog to propagate the execution to all shards.

Executing DDL via SQL at the shard catalog will return a prompt as soon as the index is created on the shard catalog; however, the population of the index on each of the shards may take several minutes, or possibly much longer.

To monitor the status of index creation, execute the SQL in the *Validating Oracle Text Index Status* section (below) on each shard periodically to determine the status of the index creation. Note that verifying each DDL execution using `GDSCTL SHOW DDL` will now show successful completion on each shard.

For full documentation on indexing JSON data, see Indexes for JSON Data.

Creating an Index on the Primary Key

We have added an index for the primary key/sharding key column. This is a standard relational database operation and is included to provide a complete representation of the deployment.

```
SQL> CREATE INDEX article_id_idx ON articles(article_id) LOCAL;
```

Creating a Multivalue Index on JSON Array Values

The primary use of a multivalue index (MVI) is to index scalar values within an array, and can also include scalar field values of object array elements.

For queries that target multiple values, a multivalue index can be more performant than a B-tree or bitmap index. If a single value is targeted, a B-tree or bitmap index may out perform the MVI. The SQL Optimizer will only use the MVI when the WHERE clause of a SELECT statement includes the use of the JSON_EXISTS() function. In a sharded database environment, the MVI must be created as a local index.

In our example data, each JSON document stored in the Articles_JSON column has a $.categories[] array with several array elements. Each array element has an `.id` value containing the name of the category. These $.categories[].id values are used in the category facet aggregated data query to display document counts for each category. If a user selects one or more categories to refine their search, the search results query must include WHERE clause filter predicates using JSON_EXISTS() for each selected category. This is where the MVI becomes important.

Example: DDL to create the multivalue index for the categories array ID element value

```
SQL> CREATE MULTIVALUE INDEX mvi_article_categories ON Articles a
           (a.article_json.categories.id.string()) local;
```

Creating an Oracle Text Index

To facilitate full text searching of the JSON content, we create an Oracle Text search index on our JSON column. First, we create the text preferences as desired, before creating the index. Oracle Text indexes can be tuned to meet your specific needs by setting any necessary preference attributes. Oracle Text preferences are defined once and applied consistently across all shards.

For full documentation, see the JSON Search Index for Ad Hoc Queries and Full-Text Search section of the JSON Developers Guide, and the Creating Oracle Text Indexes section of the Oracle Text Application Developer's Guide.

Example: SQL for creating the Oracle Text preferences

```
SQL> begin
  sys.exec_shard_plsql('ctxsys.ctx_ddl.create_preference(''wl'', ''BASIC_WORDLIST'')');
  sys.exec_shard_plsql('ctxsys.ctx_ddl.set_attribute(''wl'', ''FUZZY_MATCH'',       ''ENGLISH'')');
  sys.exec_shard_plsql('ctxsys.ctx_ddl.set_attribute(''wl'', ''FUZZY_SCORE'',       ''60'')');
  sys.exec_shard_plsql('ctxsys.ctx_ddl.set_attribute(''wl'', ''FUZZY_NUMRESULTS'',  ''5000'')');
  sys.exec_shard_plsql('ctxsys.ctx_ddl.set_attribute(''wl'', ''WILDCARD_INDEX'',    ''TRUE'')');
end;
/
```

Once the Oracle Text wordlist preferences are set, create the Oracle Text search index.

Example SQL for creating an Oracle Text Index:

```
SQL> create search index articles_idx on Articles(article_json) for json
parameters (' wordlist wl
              search_on text_value_string
              memory 170M
              maintenance auto
              optimize (auto_daily parallel 1)
           ')
           parallel 6
           local;
```

Several parameters in addition to the wordlist preferences are set directly in the CREATE SEARCH INDEX call. These should be analyzed and tuned appropriately for the specific needs of each implementation. See the Creating Oracle Text Indexes documentation for full details.

For the parameters and values shown:

- SEARCH_ON TEXT_VALUE_STRING - Causes the index to be created on the JSON element values rather than the full JSON code.
- MEMORY 170M – A moderate allocation on our hardware. Test and adjust with your hardware and data set keeping in mind any degree of parallelism specified.
- MAINTENANCE AUTO – Provides automatic synchronization of the index across the partitions for the sharded data rather than the traditional “manual” approach with scheduled jobs on a discrete schedule. This is new in Oracle Database 23ai.
- OPTIMIZE (auto_daily parallell 1) – A baseline configuration is more than sufficient for our test scenario considering our data set is not getting updates from end-users.
- PARALLEL 6 – Our test environment databases have 8 CPU cores, and no other utilization. Sparing a few from the indexing allows other database and system processes to be scheduled without CPU contention. Assess your free CPU resources and tune accordingly.

Validating Oracle Text Index Status

Several queries can be helpful to run on the shards to help inspect and verify that the index partitions, tables, and jobs have been created and are functioning correctly.

Keep in mind that the default sharded database configuration creates 120 “chunks” for your sharded data that are allocated to however many shards are added to the topology. When the tablespace set was created, every shard had a tablespace and partition created for every chunk, though only the allocated chunks for that shard will be populated with data. The Oracle Text index will still need to process every partition for every tablespace in the tablespace set.

In our examples, with four shards and the default 120 chunks, the tablespace set will only contain 25% of the data on a given shard, and the Oracle Text index will only populate 30 of 120 local index partitions.

When running these SQL statements on each shard, alter the SQL session to disable shard DDL.

```
SQL> ALTER SESSION DISABLE SHARD DDL;
```

Query for Tablespace Size

In addition to the default system tablespaces, there will be a row displayed for the explicitly declared tablespace set and several additional tablespaces for each “chunk” assigned to the shard.

In our case, for the example shard, this adds 1 for the `TSP_SET_WIKIP` and 30 tablespaces for each chunk (`C001..C00UTSP_SET_WIKIP`). The second shard will also have `TSP_SET_WIKIP`, and 30 more tablespaces for each chunk, starting with tablespace `C00VTSP_SET_WIKIP`.

```
SQL> column "Tablespace" format a24
SQL> column "Used MB"    format 99,999,999
SQL> column "Free MB"    format 99,999,999
SQL> column "Total MB"   format 99,999,999
SQL> select
   fs.tablespace_name                          "Tablespace",
   (df.totalspace - fs.freespace)              "Used MB",
   fs.freespace                                "Free MB",
   df.totalspace                               "Total MB",
   round(100 * (fs.freespace / df.totalspace)) "Pct. Free"
from
   (select tablespace_name, (sum(bytes) / 1048576) TotalSpace
      from dba_data_files
     group by tablespace_name
   ) df,
   (select tablespace_name, round(sum(bytes) / 1048576) FreeSpace
      from dba_free_space
     group by tablespace_name
   ) fs
where
   df.tablespace_name = fs.tablespace_name
order by df.tablespace_name;


Tablespace                Used MB     Free MB    Total MB  Pct. Free
------------------------ ----------- ----------- ----------- ----------
C001TSP_SET_WIKIP             1,616         158       1,774          9
C002TSP_SET_WIKIP             1,616         158       1,774          9
C003TSP_SET_WIKIP             1,540         234       1,774         13
(24 rows omitted…)
C00STSP_SET_WIKIP             1,532         242       1,774         14
C00TTSP_SET_WIKIP             1,608         166       1,774          9
C00UTSP_SET_WIKIP             1,608         166       1,774          9
SYSAUX                        1,015          75       1,090          7
SYSTEM                          328           2         330          1
TSP_SET_WIKIP                13,118       7,362      20,480         36
UNDOTBS1                        277      17,117      17,394         98
USERS                             1           4           5         80
```

Query for List of Tables, Status, and Number of Rows for Our Schema User

There should be rows returned for the tables in the table family, the external table used for data load if it hasn't been dropped, and several tables for the Oracle Text index depending on the preferences specified for the Index.

These "dollar-sign" tables contain various types of index data, the details of which are beyond the scope of this technical brief. If they are created with a "valid" status, and the other queries in this section show no faults, all is well.

```
SQL> COL OWNER FORMAT A16
SQL> COL TABLE_NAME FORMAT A32
SQL> SELECT
            OWNER, TABLE_NAME, IOT_TYPE, STATUS, NUM_ROWS
       FROM DBA_TABLES
      WHERE OWNER='WIKIP'
   ORDER BY TABLE_NAME;


OWNER            TABLE_NAME                       IOT_TYPE     STATUS     NUM_ROWS
---------------- -------------------------------- ------------ -------- ----------
WIKIP            ARTICLES                                      VALID       1524331
WIKIP            ARTICLES_EXT                                  VALID
WIKIP            DR$ARTICLES_IDX$B                             VALID
WIKIP            DR$ARTICLES_IDX$C                             VALID
WIKIP            DR$ARTICLES_IDX$G                             VALID
WIKIP            DR$ARTICLES_IDX$I                             VALID
WIKIP            DR$ARTICLES_IDX$K                             VALID
WIKIP            DR$ARTICLES_IDX$KG                            VALID
WIKIP            DR$ARTICLES_IDX$N                             VALID
WIKIP            DR$ARTICLES_IDX$Q                             VALID
WIKIP            DR$ARTICLES_IDX$SN                            VALID
WIKIP            DR$ARTICLES_IDX$ST                            VALID
WIKIP            DR$ARTICLES_IDX$SV                            VALID
WIKIP            DR$ARTICLES_IDX$U                             VALID

14 rows selected.
```

Query for the Status of the Index

Verify that none of the status values report back as FAILED.

```
SQL> SELECT
            OWNER, INDEX_NAME, INDEX_TYPE, TABLE_NAME,
            STATUS, DOMIDX_STATUS, DOMIDX_OPSTATUS, FUNCIDX_STATUS,
            DROPPED, INDEXING, DEGREE, INSTANCES, PARTITIONED, BUFFER_POOL
       FROM SYS.DBA_INDEXES
      WHERE INDEX_NAME = 'ARTICLES_IDX'
   ORDER BY 1, 2;
```

```
OWNER    INDEX_NAME     INDEX_TYPE TABLE_NAME   STATUS
-------- -------------- ---------- ------------ --------
WIKIP    ARTICLES_IDX   DOMAIN     ARTICLES     VALID

DOMIDX_STATUS DOMIDX_OPSTATUS FUNCIDX_STATUS
------------- --------------- --------------
VALID         VALID

DROPPED INDEXING DEGREE INST PAR BUFFER_POOL
------- -------- ------ ---- --- -----------
NO      FULL     6      1    YES DEFAULT
```

Query for Partition Status

This query will provide a count of partitions grouped by status column values.

All partitions should report status of USABLE and DOMIDX_OPSTATUS of VALID or INPROCESS. The total number of partitions should match the number of chunks configured for your sharded database.

```
SQL> SELECT INDEX_NAME, STATUS, DOMIDX_OPSTATUS, COUNT(*)
       FROM DBA_IND_PARTITIONS
      WHERE INDEX_NAME IN ('ARTICLES_IDX')
      GROUP BY INDEX_NAME, STATUS, DOMIDX_OPSTATUS
      ORDER BY 1,2,3;

INDEX_NAME     STATUS   DOMIDX_OPSTATUS   COUNT(*)
-------------- -------- --------------- ----------
ARTICLES_IDX   USABLE   VALID                  120
```

Query for a List of Index Partitions and Details.

This query will list all 120 index partitions, their status, indexed document count, and synchronization details.

When using “MAINTENANCE AUTO”, the columns pertaining to manually scheduled sync jobs can be omitted. These include `IXP_SYNC_PARA_DEGREE, IXP_SYNC_INTERVAL`, and `IXP_SYNC_JOB_NAME`.

Note that for this shard, partitions 1-30 are populated with documents. This corresponds with the chunks assigned to the shard in the sharded database configuration. These assigned chunks can be seen with `GDSCTL CONFIG CHUNKS` from the shard director.

```
SQL> SELECT IXP_ID, IXP_INDEX_OWNER, IXP_INDEX_NAME, IXP_INDEX_PARTITION_NAME,
            IXP_STATUS, IXP_DOCID_COUNT, IXP_SYNC_TYPE, IXP_SYNC_MEMORY
       FROM CTXSYS.CTX_INDEX_PARTITIONS
      WHERE IXP_INDEX_NAME IN ('ARTICLES_IDX')
   ORDER BY 2, 3, 1;

    IXP_ID IXP_INDEX_OWNER  IXP_INDEX_NAME   IXP_INDEX_PARTIT IXP_STATUS   IXP_DOCID_COUNT IXP_SYNC_TYPE IXP_SYNC_MEM
---------- ---------------- ---------------- ---------------- ------------ --------------- ------------- -------------
         1 WIKIP            ARTICLES_IDX     ARTICLES_P4      INDEXED                50999 MANUAL        178257920
         2 WIKIP            ARTICLES_IDX     ARTICLES_P3      INDEXED                50746 MANUAL        178257920
         3 WIKIP            ARTICLES_IDX     ARTICLES_P6      INDEXED                51040 MANUAL        178257920
(rows 4-28 omitted)
        29 WIKIP            ARTICLES_IDX     ARTICLES_P29     INDEXED                50833 MANUAL        178257920
        30 WIKIP            ARTICLES_IDX     ARTICLES_P30     INDEXED                50687 MANUAL        178257920
        31 WIKIP            ARTICLES_IDX     ARTICLES_P31     INDEXED                    0 MANUAL        178257920
(rows 32-119 omitted)
       120 WIKIP            ARTICLES_IDX     ARTICLES_P120    INDEXED                    0 MANUAL        178257920
```

Search Query Design and Examples

A fully realized solution for text search might include several capabilities for filtering and refining search requests.

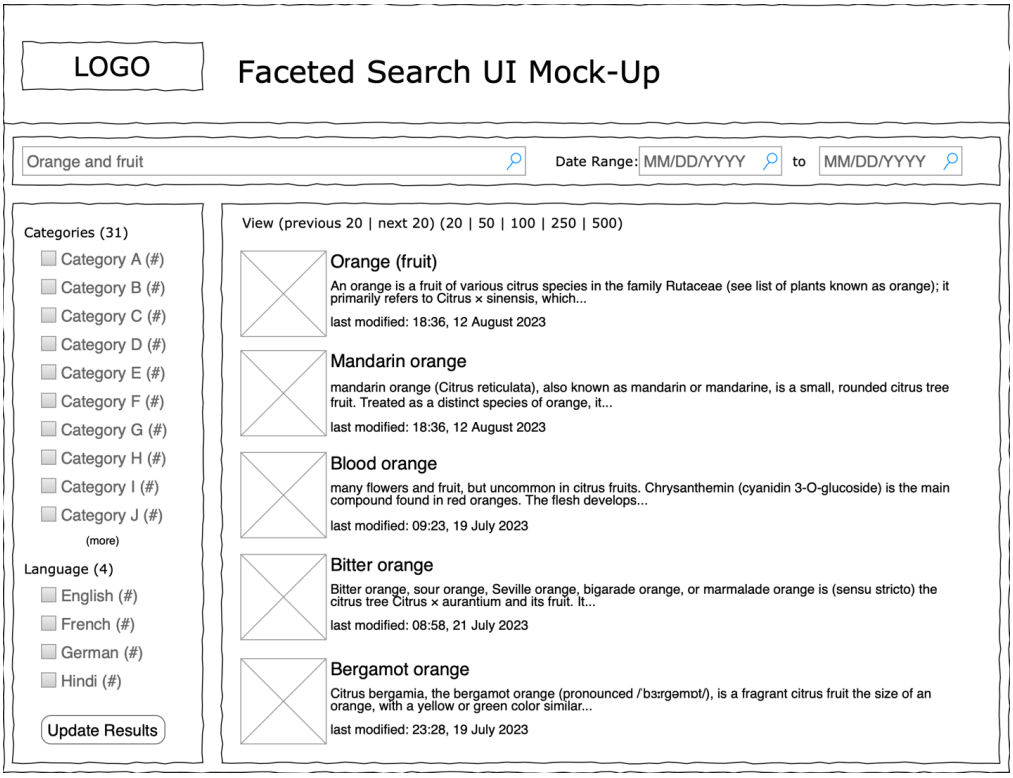
For our example, we outline several use-cases, present a mock-up of a UI design, and discuss some of the user-experience as a basis for informing the design of the SQL queries necessary for implementation. Examples of the queries and how they change to meet the requirements for the various use cases are provided, including recommended best practices.

Example Search Form

The abstract search form mock-up shown earlier is provided here in detail. This design includes basic text search functionality, date filtering, and faceted search, based on the specific categories and language metadata from our document set.

The aggregated facet selection lists should only populate after an initial search by keyword is submitted. Any dates also submitted would serve as additional filter predicates on queries. Pagination parameters are also provided for customization. The page size and current page number are used in the search results query to limit the data retrieved per request.

Figure 3 Detailed View of a Faceted Search Form UI Mock-Up



Search Form Layout and Parameters:

The UI contains three panels with several form parameters relevant to our query design. The mechanisms to submit the forms are left to the application development team.

1. Horizontal search bar with three form elements:
    a. “search_term” text box for free-form input
    b. “start_date” element for providing a date (text/calendar pop-up, etc…)
    c. “end_date” element for providing a date (text/calendar pop-up, etc…)
2. Left-hand search filter panel, containing form elements for two ‘faceted’ search filters:
    a. Result Count Aggregated By Categories
        i. displaying a header with and total count of categories found
        ii. checkboxes for each category, displayed with an aggregated count of articles for each.
    b. Result Count Aggregated By Language
        i. displaying a header with and total count of languages found
        ii. checkboxes for each language, displayed with an aggregated count of articles for each.
3. Right-hand search results panel
    a. Displays a partial list of search results sorted by the Oracle Text “score” ranking
    Each search result may display:
        i. Title
        ii. Text Snippet containing the search term(s)
        iii. Last Modified date
        iv. The search results score (optional, not shown here, use for sorting only)

Use Cases

The example application text search use case scenarios shown here highlight features common to many search interfaces. These capabilities help inform the distinct design aspects of our queries.

1. Search using basic keywords or phrases
   Submit a simple text search phrase without additional filters.
   Initial Facets and Search Results data should be presented.
2. Search with date filters
   Configure one or both start and end date filters in addition to the search term and submit the form.
   Updated Facets and Search Results data should be presented.
3. Faceted Search
   Select one or more values from the Facets and submit the form.
   The application should update the Search Results.
4. Paging through results
   Click the search results page navigation links to change the page of results displayed.
   The application should update the Search Results with the requested page of results.
5. Changing the number of results
   Update the preferred page size in the search results panel.
   The application should update the Search Results with the requested number of results starting from the same point.
6. Document View
   Click on a search result link for a specific document to be presented with a rendered view of the JSON document.
   Page should refresh, replacing the search form with the document view.

Query Design

The application's search UI form element submissions drive application logic to alter the WHERE clause filter predicates based on the parameters provided. The SQL queries are designed with bind variables based on the user inputs.

Our search queries are "multi-shard queries" that must search for results across the entire sharded database. This is appropriate for a system-managed sharded database. All multi-shard queries are submitted to the coordinator at the shard catalog database.

The exception is the sixth use-case scenario where a user clicks on a specific document link that will be handled differently. The final search result click-through query to retrieve a single article based on its shard key can use the read-write or read-only OLTP service configured through the shard director.

For each of our first five use-cases, different combinations of the queries will need to be submitted with specific combinations of filter predicates. The syntax for filter predicates will be the same for all queries. For efficiency, the SQL query examples are shown with all filter predicates in-place as if the user has iterated through use-cases 1, 2, and 3.

We provide a short description of each query, example SQL code, sample results, and discussion of the functions and techniques demonstrated. The SQL queries will show all filter predicates with comments discussing the permutations for the different use cases where various predicates are required in the WHERE clause.

Note: Use-cases 4 and 5 for managing the search results in view pagination. The implementation of the queries to derive the total documents count and paging details for the UI is left to the reader.

As the user updates the search terms, selects facet values or dates, and resubmits the faceted search page forms, the application must dynamically derive the search results query with the appropriate clauses specifically matching the input values submitted.

SQL Query Overview

There are three primary queries, one for each panel to be displayed. Each query must be adapted to include the appropriate filter predicates based on the properties of each incoming search request.

We will start with a comprehensive overview of each of the queries, along with their respective filter clauses. Once the full context is presented, the individual clauses and SQL features and functions used will be discussed in detail along with best practice recommendations.

The primary queries include:

1. Search results query - Including scored relevance rankings and document snippets
2. Faceted search query - JSON category array values with aggregated document counts
3. Faceted search query - JSON language element values with aggregated document counts

When additional filtering options are provided, the queries must include additional filters on a request-by-request basis within the application code. These filter predicates are included in the primary query examples and discussed separately. These include filtering by:

A. Dates
B. Category terms
C. Languages

Bind Variables

Using bind variables to declare the values used in query filter predicates allows for a single cursor for each SQL statement per use-case, rather than having to allocate memory for and spend time optimizing what is essentially the same SQL statement but with n-combinations with hardcoded data for parameter values. Imagine the number of combinations for every search term and date range the user population could feasibly submit. Removing these values from the literal queries has a drastic effect on the number of cursors, memory and processing overhead, and ultimately performance and scalability. For reference, see the Designing Applications for Oracle Real World Performance documentation.

Note: Applications should do strict field data validations before setting bind variable values for SQL execution.

For our example solution, the optimization of the search queries benefits from multiple bind variable declarations. Some are simple values parsed directly from the form data submitted by the users, and others are more complex values composed into specific SQL function parameter clauses or calculated values.

- The search term and date-based variables are straightforward string values of explicit input data.
- The pagination parameters that inform the search results query may be simple submitted values as well, or calculated depending on requirement. For example, the offset value needed in the query would be calculated from the rows-per-page value times the page number.
- Facet parameters may be multi-value, and therefore require some formatting to be used within a single bind variable. It is advisable in these cases for the application to build the complete filter clause parameter or path expression statement based on the SQL required.
    - In the case of the language virtual column, we can use an “IN” clause that needs a simple comma- separated list of quoted string values.
    - For the Categories facet values, use the submitted form category_id values to generate a fully qualified path expression with filters.
    - Example: SQL*Plus bind variable declarations

```
REM Set bind variables for our search queries based on user request form inputs
REM
REM Main Search Header Form data with freeform search term text entry, and selectors for
start and end dates.

define s_searchterm = 'Orange and fruit';
define s_datestart  = '2021-11-10';
define s_dateend    = '2021-11-20';

REM Categories Facet Values
REM  Form Element: "FacetCategories" checkboxes contains:
REM     "Taxonomy_articles_created_by_Polbot"
REM     and
REM     "Taxa_named_by_Carl_Linnaeus"
REM  get exact matches for each individual values (exclusive OR) to match the categories[]
array .id fields
REM  format as a path expression containing a filter for JSON_EXISTS()

define s_filtercat  = '$.categories[*]?(@.id == "Taxonomy_articles_created_by_Polbot" || @.id
== "Taxa_named_by_Carl_Linnaeus")';

REM
REM Language Facet Values
REM  Form Element: "FacetLang" checkboxes contains:
REM      "EN"
REM      and
REM      "HI"
REM  format the bind variable value for multiple values to use with the "IN" filter clause

define s_filterlang = '''EN'',''HI''';

REM
REM Paging Parameters
REM  The number of rows to display
REM  The number of rows to offset (calculate in-app as rows to display * page number)

define n_pgrows    = 3;
define n_pgoffset  = 0;
```

Search Results Query

Our example search query selects data from both the virtual columns and from the JSON column, using Oracle Text to retrieve the required results. In addition to the basic selection of the article ID and title (virtual column) data, the query returns ranked search scores and a relevant short snippet of HTML of each document's contents for each search result.

After submission of any updates on the application's search form, the application will need to build the search results query with the appropriate clauses based on the input form element values provided, execute the query, and format the results for display in the results view. The query will need to be dynamically assembled to include conditional clauses and set the bind variables based on the specific combination of inputs provided.

There are two sections to the query below:

1) The inner SELECT statement returns all matching rows, sets the search rank scores, and constructs the document snippet content. These “inner” query operations are sent from the coordinator in parallel to each shard for processing.
2) The outermost SELECT statement operations perform the final sorting and filtering for paging purposes on the result set, including the combined data returned from all shards after the inner query operations are completed and data returned to the coordinator.

Example: Scored search results query – all parameters

```
SELECT /*+ QB_NAME(qb_paging)*/
    article_id,
    title,
    last_modified,
    search_score,
    doc_snippet
FROM
    (
     SELECT /*+ QB_NAME(search_result_uc3) DOMAIN_INDEX_SORT FIRST_ROWS(100) PARALLEL(4) */
            a.article_id,
            a.title,
            a.last_modified,
            greatest(score(1),score(2)) AS search_score,
            ctx_doc.policy_snippet('articles_idx',
                                   json_serialize(a.article_json.paragraphs RETURNING clob),
                                   '&s_searchterm') AS doc_snippet
        FROM
             articles a
        WHERE
              (
                JSON_TEXTCONTAINS(a.article_json, '$.title',      '&s_searchterm', 1)
             OR JSON_TEXTCONTAINS(a.article_json, '$.paragraphs', '&s_searchterm', 2)
              )
            AND a.last_modified BETWEEN to_utc_timestamp_tz('&s_datestart')
                                    AND to_utc_timestamp_tz('&s_dateend')
            AND JSON_EXISTS ( a.article_json, '&s_filtercat' )
            AND a.lang IN ( &s_filterlang )
       ORDER BY
                greatest(score(1),score(2)) DESC
    )
OFFSET &n_pgoffset ROWS FETCH NEXT &n_pgrows ROWS ONLY;
```

Example: Scored Search Results Data Returned

```
ARTICLE_ID TITLE                    DOC_SNIPPET                                          SEARCH_SCORE
---------- ------------------------ ---------------------------------------------------- ------------

191214     Mandarin orange          Short_description, Small citrus <b>fruit</b>]\nTEMPL          100
                                    ATE[speciesbox, name = Mandarin <b>orange</b>, image
                                     = Citrus reticulata April<b>...</b>mandarine, is a
                                    small citrus tree <b>fruit</b>. Treated as a distinc
                                    t species of <b>orange</b>, it is usually eaten plai
                                    n or in <b>fruit</b> salads.TEMPLATE[cite

58875      Maclura pomifera         Osage <b>orange</b>, image<b>...</b>caption = Foliag          100
                                    e and [[multiple <b>fruit</b>]], genus = Maclura, sp
                                    ecies<b>...</b>commonly known as the Osage <b>orange
                                    </b>, horse apple, hedge, or hedge<b>...</b>15, m, f
                                    t, -1] tall. The distinctive <b>fruit</b>, a multipl
                                    e <b>fruit</b>, is roughly spherical, bumpy

68763241   Worldwide breakfast      and fresh <b>fruit</b>, including<b>...</b>accompani          100
                                    ed by coffee, tea and <b>orange</b> juice. A typical
                                     Israeli meal<b>...</b>and drink milk, hot chocolate
                                     or <b>fruit</b> juice. Japanese adults (especially
                                    younger<b>...</b>They often drink coffee or <b>orang
                                    e</b> juice. Traditional Japanese inns
```

Returned Data

This query returns display elements for each document in the search results according to business requirements, as reflected in the JSON data and schema design. For our example, this includes:

1. The document title text and article ID
   These are used to present a hyperlink to each document. For efficient retrieval of the article data from the correct shard, the request must include the shard key (article_ID) as a parameter. In this example, our sharding key matches the primary key for the data.
   Note: In some cases, the sharding key may not be the primary key. In that case, both the primary key and the sharding key column values would need to be returned by this query.
2. The last modified date of the document
3. The search results rank/score for the document to be used for sorting purposes
4. A "snippet" of the document body text from the $.paragraphs[] array
   This displays a short section of the document content with the search terms highlighted.

Noteworthy SQL Functionality

As various search form element parameters are included in the request, corresponding SQL clauses need to be present in the query to refine the results accordingly.

Search Term Text

The application user may input the search terms with or without conditional operators in free-form text. This will require careful validation at the application tier to avoid SQL injection attacks before inserting the value into the SQL queries.

Our example data has two elements within the JSON data that contain content-relevant text to search on: the title element and the paragraph element. The paragraph element is an array of string elements containing the body content of each article, and the title element as a single string.

The text string of search terms submitted from the application should be used for the search term bind parameter value specified with the JSON_TEXTCONTAINS() function. The execution plan operation for the JSON_TEXTCONTAINS() filter predicate will be optimized to use the Oracle Text search index.

Note that Oracle also provides the CONTAINS() function. When evaluating a JSON typed column it is a best practice to use the JSON_TEXTCONTAINS() function specifically optimized for this use-case.

When an evaluation of the JSON_TEXTCONTAINS() function returns a non-zero result indicating a match, the row is added to the result set. The parameters for the JSON_TEXTCONTAINS() function include the JSON field to process, a logic statement called a “path expression”, and the numeric scoring object to attribute the ranked score of these matches. It is a best practice to include separate JSON_TEXTCONTAINS() functions for each JSON element required in your WHERE clause. It is possible to write a path expression that includes the logic for multiple element values, however that may not allow the SQL engine to optimize the query to your best advantage.

Example: Multiple JSON_TEXTCONTAINS() clauses

```
WHERE
      (
        JSON_TEXTCONTAINS(a.article_json, '$.title',      '&s_searchterm', 1)
     OR JSON_TEXTCONTAINS(a.article_json, '$.paragraphs', '&s_searchterm', 2)
      )
```

With multiple JSON_TEXTCONTAINS() functions, the results rank scoring is done on a per-element basis. It is then necessary to account for all the scores when sorting the results. Using the GREATEST function with the multiple scores provides the expected results.

Example: SQL SELECT and ORDER BY clauses obtaining the actual greatest score

```
SELECT   greatest(score(1),score(2)) AS search_score
ORDER BY greatest(score(1),score(2)) DESC
```

WHERE clause order and precedence must be tested with care. When including JSON_TEXTCONTAINS() for multiple JSON elements for full-text search and using other clauses with the AND operator, the JSON_TEXTCONTAINS() should be grouped with parenthesis and use the OR operator so the search result for a given row will be successful for either match. The parenthesis wrapping the JSON_TEXT_CONTAINS() clauses are necessary when additional search predicates are included with the AND operator. The AND operator takes precedence over the OR. For example, consider these abstracted WHERE clauses:

1. WHERE A or B and C and D
   This is evaluated as: WHERE A or (B and C and D)
2. WHERE (A or B) and C and D
   This is what we want, where A & B are our JSON_TEXTCONTAINS() always being evaluated together before filtering on other fields.

Search Results HTML "Snippet" With Search Term Highlighting

Displaying a relevant HTML "snippet" of the article content containing the search term can be accomplished by returning the results of the Oracle Text CTX_DOC.POLICY_SNIPPET() function. CTX_DOC.POLICY_SNIPPET produces a 'concordance' of the document, in which occurrences of the query term are returned with their surrounding text. See the Oracle Text Application Developer's Guide 23ai

The POLICY_SNIPPET() function can take many parameters based on required functionality. For our basic example, we require three parameters: the index to search, the data to display, and the search term. When returning JSON data to display, we are choosing in this example to return the snippet from the paragraphs array with the article body content in it. This JSON array data is returned as a CLOB and processed with JSON_SERIALIZE() to give reliably displayable results. See the CTX_DOC section of the Oracle Text Reference 23ai documentation.

Example: How to take an HTML snippet of document content with search term highlighting

```
ctx_doc.policy_snippet('articles_idx',
                       json_serialize(a.article_json.paragraphs RETURNING clob),
                       '&s_searchterm') AS doc_snippet
```

Start and/or End Date Filters

All date form field values should be validated before query submission and cast to a date format. If no dates are submitted, the `WHERE` clause filter predicates for date comparison should of course be omitted from the query before submission.

For our example schema, we have implemented an `Articles.last_modified` virtual column that automatically converts the JSON string value to a proper date format.

If only one date is submitted, the `WHERE` clause filter predicate would simply include a greater-than or less-than comparison. When both values are required, the `BETWEEN()` function can be more efficient.

Use one of these three filter clauses per use-case scenario:

```
AND a.last_modified > TO_UTC_TIMESTAMP_TZ('&s_datestart')
```

```
AND a.last_modified < TO_UTC_TIMESTAMP_TZ('&s_dateend')
```

```
AND a.last_modified BETWEEN TO_UTC_TIMESTAMP_TZ('&s_datestart')
                        AND TO_UTC_TIMESTAMP_TZ('&s_dateend')
```

Facet Checkbox Selected Values

Once the user completes the first search, and the facet queries have been run to populate the facet bucket values and document counts, the user is able to select any combination of facet bucket values to further refine the search results. After selecting the facet value check boxes and resubmitting the search form, the search results documents query will need to be executed with additional WHERE clauses for the facets with returned bucket values.

In our example, there are two facets on the form: facetCategories, and facetLang. After the first search, these will be populated and contain select box elements/values for the categories and languages for the documents from the earlier search results. Let's say the user for our example selects "Category:A" and "Category:B" to limit the search context and then selects two languages, English (EN) and Hindi (HI), before submitting the search form again.

In this case, the search results document query will need to include additional SQL WHERE clauses, and the JSON path expressions required should be pre-assembled by the application and set in the bind variable values.

```
define s_filtercat  = '$.categories[*]?(@.id == "Category:A" || @.id == "Category:B")';
define s_filterlang = '''EN'',''HI''';
```

```
            AND JSON_EXISTS ( a.article_json, '&s_filtercat' )
            AND a.lang IN ( &s_filterlang )
```

For the Categories facet checkbox values, each document will be filtered by the JSON value of each $.categories[].id value matching the either of the two values submitted. To do this, we construct a filter to use with the `JSON_EXISTS()` function. The filter can be interpreted as "for every `$.categories` array element, match the document if the contained `@.id` element equals one of the requested values". The addition of a Multivalue Index on the `articles_json.categories.id` element values help optimize performance.

For the Language facet, because the bind variable values are compared to a simple scalar JSON element rather than a JSON array, we can use a virtual column and a simpler relational clause. We avoid having to explicitly use a `JSON_VALUE()` function in the query with the implementation of the `Articles.lang` virtual column.

Pagination

The user may choose the number of results to display or change the 'page' of results they want to see. The sorting and pagination of the complete merged row set returned from all shards is performed at the shard catalog using a “wrapper” `SELECT` statement around the main query. Use the “offset” function to provide pagination.

Note that the example inner query has an optimizer hint of `/* FIRST_ROWS(100) */`. This value should probably be modified dynamically using its own bind variable set to a value of `(&n_pgoffset+1)*&n_pgrows` if the remote shard execution of the query is not optimizing well.

Category Facet Query

The query to aggregate data for the Category Facet returns distinct `$category[].id` values for all matching documents ordered by document count and then the category. To retrieve a simple list of distinct category IDs requires declaring a JSON Table with one column for the `$category[].id` values.

When using `JSON_TABLE()` we provide the JSON column to work with, the JSON array we want to parse data from, and column definitions that use the JSON path relative to the array. This will distinctly populate a row in the JSON Table for every `$.category[].id` value found in the JSON data for every row returned by this query. To get the `article_id` counts we group by the JSON Table's `catid` column.

The `WHERE` clauses for this query follow the same logic and functionality as the Search Results query but should only include the search terms and any date values. The category and language facet data presented are independent in our example.

Example: Category Facet SQL query

```
SELECT /*+ QB_NAME(facet_cat_uc3) FIRST_ROWS(10) PARALLEL(4) */
      jt.catid,
      COUNT(a.article_id) AS doc_count
 FROM Articles a,
      JSON_TABLE(Article_JSON,
                 '$.categories[*]' COLUMNS (catid varchar2(64) PATH '$.id'
                 null on error)) jt
WHERE
     (
       JSON_TEXTCONTAINS(a.article_json, '$.title',      '&s_searchterm')
    OR JSON_TEXTCONTAINS(a.article_json, '$.paragraphs', '&s_searchterm')
     )
   AND a.last_modified BETWEEN to_utc_timestamp_tz('&s_datestart')
                           AND to_utc_timestamp_tz('&s_dateend')
GROUP BY jt.catid
ORDER BY doc_count DESC, jt.catid;
```

Example: Category Facet data set

```
CATID                                                            DOC_COUNT
---------------------------------------------------------------- ----------
Taxa_named_by_Carl_Linnaeus                                             246
Flora_of_New_South_Wales                                                200
Flora_of_Queensland                                                     179
Fungi_of_North_America                                                  148
Taxonomy_articles_created_by_Polbot                                     144
```

Language Facet Query

The design for the language facet query is identical to that of the Category facet query, however the `JSON_TABLE` is built directly on the `$.lang` `JSON` element containing a single string value per document. Optionally, the language facet query could utilize the `lang` Virtual Column we created earlier for relational queries. This eliminates the complexity of calling and parsing a `JSON_TABLE()` function evaluation simplifying client-side code. Both approaches are provided.

Example: Language Facet SQL Query with JSON_TABLE()

```
SELECT /*+ QB_NAME(facet_lang_uc3a) PARALLEL(4) */
       jt.lang,
       COUNT(a.article_id) AS doc_count
  FROM Articles a,
       JSON_TABLE(Article_JSON,'$.lang' COLUMNS (lang PATH '$' null on error)) jt
 WHERE
       (
           JSON_TEXTCONTAINS(a.article_json, '$.title',      '&s_searchterm')
        OR JSON_TEXTCONTAINS(a.article_json, '$.paragraphs', '&s_searchterm')
       )
       AND a.last_modified BETWEEN to_utc_timestamp_tz('&s_datestart')
                               AND to_utc_timestamp_tz('&s_dateend')
  GROUP BY jt.lang
  ORDER BY doc_count desc jt.lang;
```

Example: Category Facet data set

```
LANG  DOC_COUNT
---- ----------
EN          851
HI            1
```

Example: Language Facet SQL query using a virtual column

```
SELECT /*+ QB_NAME(facet_lang_uc3b) PARALLEL(4) */
       a.lang,
       COUNT(a.article_id) AS doc_count
  FROM Articles a,
 WHERE
       (
           JSON_TEXTCONTAINS(a.article_json, '$.title',      '&s_searchterm')
        OR JSON_TEXTCONTAINS(a.article_json, '$.paragraphs', '&s_searchterm')
       )
       AND a.last_modified BETWEEN to_utc_timestamp_tz('&s_datestart')
                               AND to_utc_timestamp_tz('&s_dateend')
  GROUP BY a.lang
  ORDER BY doc_count desc, a.lang;
```

Example: Language Facet data set

```
LANG  DOC_COUNT
---- ----------
EN          851
HI            1
```

Conclusion

We have demonstrated how to keep your data in the Oracle database and combine the power of OracleText Indexes, JSON Search capabilities, and Oracle's Globally Distributed Database architecture to provide full-text search at a massive scale using Oracle's native query capabilities.

This provides the ability to:

- Reduce application complexity with a single connection model for data and search requests.
- Eliminate requirements for other third-party document stores, reducing operational costs.
- Scale-out to support massive datasets and extreme workloads.
- Leverage the high availability inherent to Oracle Database solutions.

Appendix A: High Availability and Disaster Recovery

Oracle Maximum Availability Architecture best practices apply to Globally Distributed Databases.

In terms of replication technologies, Oracle Globally Distributed Database includes a built-in Raft replication technology using Raft consensus algorithms. Oracle Data Guard is also tightly integrated. Either replication technology can be chosen for the shard databases. Oracle Data Guard can be used for replicating the shard catalog database.

References:

- Replication in Oracle Globally Distributed Database
- Introduction to Sharded Database Deployment
- Raft Replication Configuration and Management
- Oracle MAA Best Practices for Oracle Database

Appendix B: Sharded Data Distribution Methods

Oracle Globally Distributed Database uses a scaling technique based on horizontal partitioning of data across multiple independent physical databases, called “shards”. The distribution of the data is transparent to the application.

Tables that are deployed as sharded tables are partitioned in a manner similar to the standard Oracle Database partitioning features with the addition of distributing the data into specific partitions across distinctly separate shard databases. The way data is distributed across the shards can be chosen to align with your design goals and requirements. For a comprehensive introduction, see the Overview section of the Oracle Globally Distributed Database Guide documentation.

Because Oracle Globally Distributed Database is based on table partitioning, all of the sub-partitioning methods provided by Oracle Database are also supported by Oracle Globally Distributed Database.

A data distribution method controls the placement of the data on the shards. Oracle Globally Distributed Database supports system-managed, user-defined, directory-based, and composite sharding methods.

- **System-managed:** Does not require you to map data to shards. The data is automatically distributed across shards using partitioning by consistent hash. The partitioning algorithm uniformly and randomly distributes data across shards.

  See System-Managed Sharding in the Oracle Globally Distributed Database Guide documentation for more information.

- **User-defined:** Lets you explicitly specify the mapping of data to individual shards. It is used when, because of performance, regulatory, or other reasons, certain data needs to be stored on a particular shard, and the administrator needs to have full control over moving data between shards.

  See User-Defined Sharding in the Oracle Globally Distributed Database Guide documentation for more information.

- **Composite:** Allows you to use two levels of partitioning. First the data is partitioned by range or list and then it is partitioned further by consistent hash.

  In many use cases, especially for data sovereignty and data proximity requirements, the composite method offers the best of both system-managed and user-defined methods, giving you the automation you want and the control over data placement you need.

  See Composite Sharding in the Oracle Globally Distributed Database Guide documentation for more information.

- **Directory-Based:** An enhancement of the user-defined method, where the location of data records associated with any sharding key is specified dynamically at runtime based on user preferences. The key location information is stored in a directory, which can hold a large set of key values in the hundreds of thousands.

  You have the freedom to move individual key values from one location to another, or make bulk movements to scale up or down, or for data and load balancing. The location information can include the shard database information and partition information.

  See Directory-Based Sharding in the Oracle Globally Distributed Database Guide documentation for more information.

ORACLE

Further Reading:

Oracle's JSON Capabilities:

- 23ai Overview of Sharding JSON Documents
- 23ai Doc Reference: JSON Developer's Guide
- Oracle Blog post on the new JSON datatype:
- Oracle Blog post on how JSON documents can be stored in a Sharded environment:

Oracle Text:

- Tech Brief: New User's Guide to Oracle Text in Oracle Database
- 23ai Oracle Text Application Developer's Guide

Globally Distributed Database:

- 23ai Globally Distributed Database Documentation
- Configure TCPS with TLS for Sharding
- Oracle Globally Distributed Database Deployment
- Command Reference

Global Data Services:

- Global Data Services Concepts and Administration Guide

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle