# Oracle® Database

# SODA for PL/SQL Developer's Guide

Release 21
F16799-04
August 2021

Oracle Database SODA for PL/SQL Developer's Guide, Release 21

F16799-04



Primary Author: Drew Adams

Contributors: Douglas McMahon, Maxim Orgiyan, Srikrishnan Suresh

# Contents


## Preface

Audience viii
Documentation Accessibility viii
Diversity and Inclusion viii
Related Documents ix
Conventions ix

## 1 SODA for PL/SQL Prerequisites

## 2 SODA for PL/SQL Overview

## 3 Using SODA for PL/SQL

3.1 Getting Started with SODA for PL/SQL 3-3
3.2 Creating a Document Collection with SODA for PL/SQL 3-6
3.3 Opening an Existing Document Collection with SODA for PL/SQL 3-8
3.4 Checking Whether a Given Collection Exists with SODA for PL/SQL 3-8
3.5 Discovering Existing Collections with SODA for PL/SQL 3-9
3.6 Dropping a Document Collection with SODA for PL/SQL 3-10
3.7 Creating Documents with SODA for PL/SQL 3-11
3.8 Inserting Documents into Collections with SODA for PL/SQL 3-17
3.9 Saving Documents Into a Collection with SODA for PL/SQL 3-19
3.10 SODA for PLSQL Read and Write Operations 3-21
3.11 Finding Documents in Collections with SODA for PL/SQL 3-23
3.12 Replacing Documents in a Collection with SODA for PL/SQL 3-32
3.13 Removing Documents from a Collection with SODA for PL/SQL 3-35
3.14 Truncating a Collection (Removing All Documents) with SODA for PL/SQL 3-38
3.15 Indexing the Documents in a Collection with SODA for PL/SQL 3-38
3.16 Getting a Data Guide for a Collection with SODA for PL/SQL 3-42
3.17 Creating a View from a Data Guide with SODA for PL/SQL 3-45

3.18 Handling Transactions with SODA for PL/SQL 3-46

4 SODA Collection Configuration Using Custom Metadata

4.1 Getting the Metadata of an Existing Collection 4-2
4.2 Creating a Collection That Has Custom Metadata 4-2

A Redefining a SODA Collection

Index

# List of Examples


| | | |
|---|---|---|
| 3-1 | Getting Started Run-Through | 3-4 |
| 3-2 | Sample Output for Getting Started Run-Through | 3-5 |
| 3-3 | Creating a Collection That Has the Default Metadata | 3-7 |
| 3-4 | Opening an Existing Document Collection | 3-8 |
| 3-5 | Checking for a Collection with a Given Name | 3-9 |
| 3-6 | Printing the Names of All Existing Collections | 3-9 |
| 3-7 | Dropping a Document Collection | 3-11 |
| 3-8 | Creating a Document with JSON Content | 3-14 |
| 3-9 | Creating a Document with Document Key and JSON Content | 3-15 |
| 3-10 | Inserting a Document into a Collection | 3-18 |
| 3-11 | Inserting a Document into a Collection and Getting the Result Document | 3-18 |
| 3-12 | Saving Documents Into a Collection with SODA for PL/SQL | 3-20 |
| 3-13 | Finding All Documents in a Collection Using SODA For PL/SQL | 3-24 |
| 3-14 | Finding the Unique Document That Has a Given Document Key Using SODA For PL/SQL | 3-25 |
| 3-15 | Finding Multiple Documents with Specified Document Keys Using SODA For PL/SQL | 3-26 |
| 3-16 | Finding Documents with a Filter Specification Using SODA For PL/SQL | 3-26 |
| 3-17 | Specifying Pagination Queries with Methods skip() and limit() Using SODA For PL/SQL | 3-28 |
| 3-18 | Specifying Document Version Using SODA For PL/SQL | 3-28 |
| 3-19 | Counting the Number of Documents Found | 3-29 |
| 3-20 | Retrieving the Documents of a Collection at a Time in the Past (Flashback) Using SODA For PL/SQL | 3-30 |
| 3-21 | Using Full-Text Search To Find Documents in a Heterogeneous Collection Using SODA For PL/SQL | 3-30 |
| 3-22 | Replacing a Document, Given Its Key, and Getting the Result Document Using SODA For PL/SQL | 3-32 |
| 3-23 | Replacing a Particular Version of a Document Using SODA For PL/SQL | 3-33 |
| 3-24 | Locking a Document For Update (Replacement) Using SODA For PL/SQL | 3-34 |
| 3-25 | Removing a Document from a Collection Using a Document Key | 3-36 |
| 3-26 | Removing a Particular Version of a Document | 3-36 |
| 3-27 | Removing Documents from a Collection Using Document Keys | 3-37 |
| 3-28 | Removing JSON Documents from a Collection Using a Filter | 3-37 |
| 3-29 | Truncating a Collection | 3-38 |
| 3-30 | Creating a B-Tree Index for a JSON Field with SODA for PL/SQL | 3-40 |
| 3-31 | JSON Search Indexing with SODA for PL/SQL | 3-40 |
| 3-32 | Dropping an Index with SODA for PL/SQL | 3-41 |

3-33 Getting an Index Specification with SODA for PL/SQL 3-41
3-34 Getting All Index Specifications For a Collection with SODA for PL/SQL 3-42
3-35 Creating a Data Guide Dynamically with SODA for PL/SQL 3-43
3-36 Creating a Data Guide Using a JSON Search Index with SODA for PL/SQL 3-44
3-37 Creating a Relational View from a JSON Data Guide with SODA for PL/SQL 3-45
3-38 Transaction Involving SODA Document Insertion and Replacement 3-47
4-1 Getting the Metadata of a Collection 4-2
4-2 Creating a Collection That Has Custom Metadata 4-3

# List of Tables

3-1 Getter Methods for Documents (SODA_DOCUMENT_T) 3-13

# Preface

This document describes how to use Simple Oracle Document Access (SODA) for C.


- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Related Documents
- Conventions


## Audience

This document is intended for users of Simple Oracle Document Access (SODA) for PL/SQL.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

## Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

# Related Documents

For more information, see these Oracle resources:

- https://docs.oracle.com/en/database/oracle/simple-oracle-document-access/ for complete information about SODA and its implementations
- *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for general information about SODA
- *Oracle as a Document Store* for general information about using JSON data in Oracle Database, including with SODA
- *Oracle Database JSON Developer's Guide* for information about using SQL and PL/SQL with JSON data stored in Oracle Database

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at OTN Registration.

If you already have a user name and password for OTN then you can go directly to the documentation section of the OTN Web site at OTN Documentation.

# Conventions

The following text conventions are used in this document:

| Convention | Meaning |
|---|---|
| **boldface** | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |
| *italic* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `monospace` | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

1

# SODA for PL/SQL Prerequisites

SODA for PL/SQL is an integral part of Oracle Database, starting with Release 18c (18.1). The database is the only prerequisite for using SODA for PL/SQL, but some features are available only starting with particular database releases.

The following features were added to SODA for PL/SQL in Oracle Database Release 18.3. You need that database release or later to use them:

- Data-type `SODA_OPERATION_T`
- Indexing
- JSON data guide

2

# SODA for PL/SQL Overview

**SODA for PL/SQL** is a PL/SQL API that implements **Simple Oracle Document Access** (SODA). You can use it with PL/SQL to perform create, read (retrieve), update, and delete (CRUD) operations on documents of any kind, and you can use it to query JSON documents.

**SODA** is a set of NoSQL-style APIs that let you create and store collections of documents in Oracle Database, retrieve them, and query them, without needing to know Structured Query Language (SQL) or how the data in the documents is stored in the database.

Oracle Database supports storing and querying JSON data. To access this functionality, you need structured query language (SQL) with special JSON SQL operators. SODA for PL/SQL hides the complexities of SQL/JSON programming.

The remaining topics of this document describe various features of SODA for PL/SQL.

**Note:**

- This book provides information about using SODA with PL/SQL applications, and it describes all SODA features currently available for use with PL/SQL. To use SODA for PL/SQL you also need to understand SODA generally. For such general information, please consult *Oracle Database Introduction to Simple Oracle Document Access (SODA)*. Some features described in that book are not yet available with SODA for PL/SQL.
- This book does not provide general information about PL/SQL, including reference information about the SODA for PL/SQL methods and constants. For such information, please consult *Oracle Database PL/SQL Language Reference*.

**See Also:**

*Oracle Database JSON Developer's Guide* for information about using SQL and PL/SQL with JSON data stored in Oracle Database

# 3
# Using SODA for PL/SQL

How to access SODA for PL/SQL is described, as well as how to use it to perform create, read (retrieve), update, and delete (CRUD) operations on collections.

(CRUD operations are also called “read and write operations” in this document.)

- Getting Started with SODA for PL/SQL
  How to access SODA for PL/SQL is described, as well as how to use it to create a database collection, insert a document into a collection, and retrieve a document from a collection.
- Creating a Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.
- Opening an Existing Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.open_collection` to open an existing document collection.
- Checking Whether a Given Collection Exists with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.open_collection` to check for the existence of a given collection. It returns a SQL `NULL` value if a collection with the specified name does not exist; otherwise, it returns the collection object.
- Discovering Existing Collections with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.list_collection_names` to discover existing collections.
- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Creating Documents with SODA for PL/SQL
  You use a constructor for PL/SQL object type `SODA_DOCUMENT_T` to create SODA documents.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Saving Documents Into a Collection with SODA for PL/SQL
  You can use `SODA_DOCUMENT_T` method `save()` or `save_and_get()` to save documents into a collection, which means *inserting* them if they are new or *updating* them if they already belong to the collection. (Such an operation is sometimes called "upserting".)
- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.
- Finding Documents in Collections with SODA for PL/SQL
  You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method

`count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.

- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.
- Removing Documents from a Collection with SODA for PL/SQL
  You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.
- Truncating a Collection (Removing All Documents) with SODA for PL/SQL
  You can use `SODA_COLLECTION_T` method `truncate()` to empty, or **truncate**, a collection, which means remove all of its documents.
- Indexing the Documents in a Collection with SODA for PL/SQL
  You index the documents in a SODA collection with `SODA_COLLECTION_T` method `create_index()`. Its input parameter is a textual JSON index specification. This can specify support for B-tree, spatial, full-text, and ad hoc indexing, and it can specify support for a JSON data guide.
- Getting a Data Guide for a Collection with SODA for PL/SQL
  You can use `SODA_COLLECTION_T` method `get_data_guide()` or terminal `SODA_OPERATION_T` method `get_data_guide()` to obtain a data guide for a collection. A **data guide** is a JSON document that summarizes the structural and type information of the JSON documents in the collection. It records metadata about the fields used in those documents.
- Creating a View from a Data Guide with SODA for PL/SQL
  You can use `SODA_COLLECTION_T` method `create_view_from_dg()` to create a database view with relational columns, whose names and values are taken from the scalar JSON fields specified in the data guide. A data guide-enabled JSON search index is *not* required for this; the data guide itself is passed to the method.
- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.

# 3.1 Getting Started with SODA for PL/SQL

How to access SODA for PL/SQL is described, as well as how to use it to create a database collection, insert a document into a collection, and retrieve a document from a collection.

> **Note:**
>
> Don't worry if not everything in this topic is clear to you on first reading. The necessary concepts are developed in detail in other topics. This topic should give you an idea of what is involved overall in using SODA.

Follow these steps to get started with SODA for PL/SQL:

1. Ensure that the prerequisites have been met for using SODA for PL/SQL. See SODA for PL/SQL Prerequisites.
2. Identify the database schema (user account) used to store collections, and grant database role `SODA_APP` to that schema:

   ```
   GRANT SODA_APP TO schemaName;
   ```

3. Use PL/SQL code such as that in Example 3-1 to do the following:
   a. Create and open a collection (an instance of PL/SQL object type `SODA_COLLECTION_T`), using the default collection configuration (metadata).
   b. Create a document with particular JSON content, as an instance of PL/SQL object type `SODA_DOCUMENT_T`.
   c. Insert the document into the collection.
   d. Get the inserted document back. Its other components, besides the content, are generated automatically.
   e. Print the unique document key, which is one of the components generated automatically.
   f. Commit the document insertion.
   g. Find the document in the collection, by providing its key.
   h. Print some of the document components: key, content, creation timestamp, last-modified timestamp, and version.
4. Drop the collection, cleaning up the database table that is used to store the collection and its metadata:

   ```
   SELECT DBMS_SODA.drop_collection('myCollectionName') AS drop_status FROM
   DUAL;
   ```

> **Caution:**
>
> Do *not* use SQL to drop the database *table* that underlies a collection. Dropping a *collection* involves more than just dropping its database table. In addition to the documents that are stored in its table, a collection has *metadata*, which is also persisted in Oracle Database. Dropping the table underlying a collection does *not* also drop the collection metadata.

> **Note:**
>
> - If a PL/SQL subprogram that you write invokes subprograms that are in package `DBMS_SODA`, and if your subprogram has definer (owner) rights, then your subprogram must be granted role `SODA_APP`. For example, this code grants role `SODA_APP` to procedure `my_soda_proc`, which is owned by database schema (user) `my_db_schema`:
>
>   ```
>   GRANT SODA_APP TO PROCEDURE my_db_schema.my_soda_proc;
>   ```
>
> - `DBMS_SODA` subprograms run with *invoker's* right. They require the invoker to have the necessary privileges. For example, procedure `create_collection` needs privilege `CREATE TABLE`. (It is needed to create the table that backs the collection.)
>
>   In general, such privileges can be granted to the invoker either directly or through a database role. However, when a subprogram that is created with *definer's* rights invokes a `DBMS_SODA` subprogram, the relevant privileges must be granted directly, *not* through a role, to the user who defined that definer's-rights subprogram.

> **See Also:**
>
> Predefined Roles in an Oracle Database Installation in *Oracle Database Security Guide* for information about role `SODA_APP`

**Example 3-1 Getting Started Run-Through**

```
DECLARE
    collection      SODA_COLLECTION_T;
    document        SODA_DOCUMENT_T;
    foundDocument   SODA_DOCUMENT_T;
    result_document SODA_DOCUMENT_T;
    docKey          VARCHAR2(100);
    status          NUMBER;
BEGIN
    -- Create a collection.
    collection := DBMS_SODA.create_collection('myCollectionName');
```

```
    -- The default collection has BLOB content, so create a BLOB-based
document.
    document := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" :
"Alexander"}'));

    -- Insert the document and get it back.
    result_document := collection.insert_one_and_get(document);

    -- The result document has auto-generated components, such as key and
version,
    -- in addition to the content.  Print the auto-generated document key.
    docKey := result_document.get_key;
    DBMS_OUTPUT.put_line('Auto-generated key is ' || docKey);

    -- Commit the insert
    COMMIT;

    -- Find the document in the collection by its key
    foundDocument := collection.find_one(docKey);

    -- Get and print some document components: key, content, etc.
    DBMS_OUTPUT.put_line('Document components:');
    DBMS_OUTPUT.put_line('  Key: ' || foundDocument.get_key);
    DBMS_OUTPUT.put_line('  Content: '
                         ||
utl_raw.cast_to_varchar2(foundDocument.get_blob));
    DBMS_OUTPUT.put_line('  Creation timestamp: ' ||
foundDocument.get_created_on);
    DBMS_OUTPUT.put_line('  Last-modified timestamp: '
                         || foundDocument.get_last_modified);
    DBMS_OUTPUT.put_line('  Version: ' || foundDocument.get_version);
END;
/
```

### Example 3-2 Sample Output for Getting Started Run-Through

Example 3-1 results in output similar to this. The values of the auto-generated components will differ in any actual execution.

```
Auto-generated key is 96F35328CD3B4F96BF3CD01BCE9EBDF5
Document components:
  Key: 96F35328CD3B4F96BF3CD01BCE9EBDF5
  Content: {"name" : "Alexander"}
  Creation timestamp: 2017-09-19T01:05:06.160289Z
  Last-modified timestamp: 2017-09-19T01:05:06.160289Z
  Version: FD69FB6ACE73FA735EC7922CA4A02DDE0690462583F9EA2AF754D7E342B3EE78
```

# 3.2 Creating a Document Collection with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.

Example 3-3 uses PL/SQL function `DBMS_SODA.create_collection` to create a collection that has the default metadata.

The *default collection metadata* has the following characteristics.

- Each document in the collection has these document components:
  - Key
  - Content
  - Creation timestamp
  - Last-modified timestamp
  - Version
- The collection can store only JSON documents.
- Document keys are automatically generated for documents that you add to the collection.

The default collection configuration is recommended in most cases, but collections are highly configurable. When you create a collection you can specify things such as the following:

- Storage details, such as the name of the table that stores the collection and the names and data types of its columns.
- The presence or absence of columns for creation timestamp, last-modified timestamp, and version.
- Whether the collection can store only JSON documents.
- Methods of document key generation, and whether document keys are client-assigned or generated automatically.
- Methods of version generation.

This configurability also lets you map a new collection to an existing database table.

To configure a collection in a nondefault way, supply custom collection metadata, expressed in JSON, as the second argument to `DBMS_SODA.create_collection`.

If you do not care about the details of collection configuration then pass only the collection name to `DBMS_SODA.create_collection` — no second argument. That creates a collection with the default configuration.

If a collection with the same name already exists then it is simply opened and its handle is returned. If custom metadata is provided and it does not match the metadata of the existing collection then the collection is not opened and an error is raised. (To match, all metadata fields must have the same values.)

> **Note:**
>
> Unless otherwise stated, the remainder of this documentation assumes that a collection has the default configuration.

> **See Also:**
>
> - Default Naming of a Collection Table in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for information about the default naming of a collection table
> - CREATE_COLLECTION Function in *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.create_collection`

**Example 3-3 Creating a Collection That Has the Default Metadata**

This example creates collection `myCollectionName` with the default metadata.

```
DECLARE
    collection  SODA_Collection_T;
BEGIN
    collection := DBMS_SODA.create_collection('myCollectionName');
END;
/
```

**Related Topics**

- Getting the Metadata of an Existing Collection
  You use `SODA_COLLECTION_T` method `get_metadata()` to get all of the metadata for a collection, as a JSON document.
- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to PL/SQL function `DBMS_SODA.create_collection`.
- Checking Whether a Given Collection Exists with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.open_collection` to check for the existence of a given collection. It returns a SQL `NULL` value if a collection with the specified name does not exist; otherwise, it returns the collection object.

# 3.3 Opening an Existing Document Collection with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.open_collection` to open an existing document collection.

> **See Also:**
>
> OPEN_COLLECTION Function in *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.open_collection`

**Example 3-4 Opening an Existing Document Collection**

This example uses PL/SQL function `DBMS_SODA.open_collection` to open the collection named `myCollectionName` and returns a `SODA_COLLECTION_T` instance that represents this collection. If the value returned is `NULL` then there is no existing collection named `myCollectionName`.

```
DECLARE
    collection  SODA_COLLECTION_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
END;
/
```

# 3.4 Checking Whether a Given Collection Exists with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.open_collection` to check for the existence of a given collection. It returns a SQL `NULL` value if a collection with the specified name does not exist; otherwise, it returns the collection object.

> **See Also:**
>
> OPEN_COLLECTION Function in *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.open_collection`

**Example 3-5 Checking for a Collection with a Given Name**

This example uses `DBMS_SODA.open_collection` to try to open an existing collection named `myCollectionName`. It prints a message if no such collection exists.

```
DECLARE
    collection SODA_COLLECTION_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    IF collection IS NULL THEN
        DBMS_OUTPUT.put_line('Collection does not exist');
    END IF;
END;
/
```

**Related Topics**

- Creating a Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.

# 3.5 Discovering Existing Collections with SODA for PL/SQL

You can use PL/SQL function `DBMS_SODA.list_collection_names` to discover existing collections.

> **See Also:**
>
> LIST_COLLECTION_NAMES Function in *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.list_collection_names`

**Example 3-6 Printing the Names of All Existing Collections**

This example uses PL/SQL function `DBMS_SODA.list_collection_names` to obtain a list of the collection names. It then iterates over that list, printing out the names.

```
DECLARE
    coll_list  SODA_COLLNAME_LIST_T;
BEGIN
    coll_list := DBMS_SODA.list_collection_names;
    DBMS_OUTPUT.put_line('Number of collections: ' ||
to_char(coll_list.count));
    DBMS_OUTPUT.put_line('Collection List: ');
    IF (coll_list.count > 0) THEN
        -- Loop over the collection name list
        FOR i IN
            coll_list.first .. coll_list.last
        LOOP
            DBMS_OUTPUT.put_line(coll_list(i));
        END LOOP;
    ELSE
```

```
        DBMS_OUTPUT.put_line('No collections found');
    END IF;
END;
/
```

# 3.6 Dropping a Document Collection with SODA for PL/SQL

You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.

> **Caution:**
>
> Do *not* use SQL to drop the database *table* that underlies a collection. Dropping a *collection* involves more than just dropping its database table. In addition to the documents that are stored in its table, a collection has *metadata*, which is also persisted in Oracle Database. Dropping the table underlying a collection does *not* also drop the collection metadata.

> **Note:**
>
> Day-to-day use of a typical application that makes use of SODA does not require that you drop and re-create collections. But if you need to do that for any reason then this guideline applies.
>
> Do *not* drop a collection and then re-create it with *different metadata* if there is any application running that uses the collection in any way. Shut down any such applications before re-creating the collection, so that all live SODA objects are released.
>
> There is no problem just dropping a collection. Any read or write operation on a dropped collection raises an error. And there is no problem dropping a collection and then re-creating it with the same metadata. But if you re-create a collection with different metadata, and if there are any live applications using SODA objects, then there is a risk that a stale collection is accessed, and *no error is raised* in this case.

> **Note:**
>
> Commit all writes to a collection *before* using `DBMS_SODA.drop_collection`. For the drop to succeed, all uncommitted writes to the collection must first be either committed or rolled back — you must explicitly use SQL `COMMIT` or `ROLLBACK`. Otherwise, an exception is raised.

> **See Also:**
>
> DROP_COLLECTION Function in *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.drop_collection`

**Example 3-7 Dropping a Document Collection**

This example uses PL/SQL function `DBMS_SODA.drop_collection` to drop collection `myCollectionName`.

If the collection cannot be dropped because of uncommitted write operations then an exception is thrown. If the collection is dropped successfully, the returned status is 1; otherwise, the status is 0. In particular, if a collection with the specified name does not exist, the returned status is 0 — no exception is thrown.

```
DECLARE
    status  NUMBER := 0;
BEGIN
    status := DBMS_SODA.drop_collection('myCollectionName');
END;
/
```

**Related Topics**

- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.

# 3.7 Creating Documents with SODA for PL/SQL

You use a constructor for PL/SQL object type `SODA_DOCUMENT_T` to create SODA documents.

SODA for PL/SQL represents a document using an instance of PL/SQL object type `SODA_DOCUMENT_T`. This object is a *carrier* of document content and other document components, such as the document key.

Here is an example of the *content* of a JSON document:

```
{ "name" :    "Alexander",
  "address" : "1234 Main Street",
  "city" :    "Anytown",
  "state" :   "CA",
  "zip" :     "12345"
}
```

A document has these **components**:

- Key
- Content
- Creation time stamp
- Last-modified time stamp
- Version
- Media type (`"application/json"` for JSON documents)

You *create a document* by invoking one of the **`SODA_DOCUMENT_T`** constructors. The constructors differ according to the content type of the documents they create: `JSON`, `VARCHAR2`, `CLOB`, or `BLOB`. Documents with content of data type `JSON` can be created only if database initialization parameter `compatible` is at least `20`.

In general, you can write a document of a given content type only to a collection whose *content column* has been defined for documents of that type. For example, you can write (insert or replace) only a document with content type `VARCHAR2` to a collection whose `contentColumn` has a `sqlType` value of `VARCHAR2`.

The only exception to this is that you can write a document of type `BLOB` to a collection with a `JSON` type content column. (The *default* content type for a collection is `JSON` if database initialization parameter `compatible` is at least `20`; otherwise, it is `BLOB`.)

There are different ways to invoke a document constructor:

- You can provide the document *key*, as the first argument.

  In a collection, each document must have a key. You must provide the key when you create the document *only* if you expect to insert the document into a collection that does *not* automatically generate keys for inserted documents. By default, collections are configured to automatically generate document keys.

- You *must* provide the document *content*. If you also provide the document key then the content is the second argument to the constructor.

  If you provide only the content then you must specify both the formal and actual content parameters, separated by the association arrow (`=>`): **`j`**`_content => `*`actual`*, **`v`**`_content => `*`actual`*, **`c`**`_content => `*`actual`*, or **`b`**`_content => `*`actual`*, for content of type **`J`**`SON`, **`V`**`ARCHAR2`, **`C`**`LOB`, or **`B`**`LOB`, respectively.

- You can provide the document media type, which defaults to `"application/json"`. Unless the content type is `JSON` or you provide all of the parameters (key, content, and media type) you must specify both the formal and actual media-type parameters, , separated by the association arrow (`=>`): **`media_type =>`** *`actual`*. If the content type is `JSON` then the media type is always `"application/json"` — you

need not specify it as such, and you cannot specify it as something other than `"application/json"` without raising an error.

Parameters that you do not provide explicitly default to `NULL`.

Providing only the content parameter can be useful for creating documents that you insert into a collection that automatically generates document keys. Providing only the key and content can be useful for creating documents that you insert into a collection that has client-assigned keys. Providing (the content and) the media type can be useful for creating *non*-JSON documents (using a media type other than `"application/json"`).

However you invoke a `SODA_DOCUMENT_T` constructor, doing so sets the components that you provide (the content, possibly the key, and possibly the media type) to the values you provide for them. And it sets the values of the creation time stamp, last-modified time stamp, and version to a SQL `NULL` value.

Object type `SODA_DOCUMENT_T` provides **getter methods** (also known as *getters*), which each retrieve a particular component from a document. (Getter `get_data_type()` actually returns information about the content component, rather than the component itself.)

**Table 3-1 Getter Methods for Documents (SODA_DOCUMENT_T)**

| Getter Method | Description |
|---|---|
| **`get_created_on()`** | Get the *creation time stamp* for the document, as a `VARCHAR2` value. |
| **`get_key()`** | Get the unique *key* for the document, as a `VARCHAR2` value. |
| **`get_last_modified()`** | Get the *last-modified time stamp* for the document, as a `VARCHAR2` value. |
| **`get_media_type()`** | Get the *media type* for the document, as a `VARCHAR2` value. |
| **`get_version()`** | Get the document *version*, as a `VARCHAR2` value. |
| **`get_json()`** | Get the document *content*, as a `JSON` value.<br>The document content must be `JSON` data, or else an error is raised. |
| **`get_blob()`** | Get the document *content*, as a `BLOB` value.<br>The document content must be `BLOB` data, or else an error is raised. |
| **`get_clob()`** | Get the document *content*, as a `CLOB` value.<br>The document content must be `CLOB` data, or else an error is raised. |
| **`get_varchar2()`** | Get the document *content*, as a `VARCHAR2` value.<br>The document content must be `VARCHAR2` data, or else an error is raised. |
| **`get_data_type()`** | Get the data type of the document content, as a `PLS_INTEGER` value. The value is **`DBMS_SODA.DOC_VARCHAR2`** for `VARCHAR2` content, **`DBMS_SODA.DOC_BLOB`** for `BLOB` content, and **`DBMS_SODA.DOC_CLOB`** for `CLOB` content. |

Immediately after you create a document, the getter methods return these values:

- Values provided to the constructor
- `"application/json"`, for method `get_media_type()`, if the media type was not provided
- `NULL` for other components

Each content storage data type has an associated content-component getter method. You must use the getter method that is appropriate to each content storage type: `get_json()` for `JSON` type storage, `get_varchar2()` for `VARCHAR2` storage, `get_clob()` for `CLOB` storage, and `get_blob()` for `BLOB` storage. Otherwise, an error is raised.

Example 3-8 creates a `SODA_DOCUMENT_T` instance, providing only content. The media type defaults to `"application/json"`, and the other document components default to `NULL`.

Example 3-9 creates a `SODA_DOCUMENT_T` instance, providing the document key and content. The media type defaults to `"application/json"`, and the other document components default to `NULL`.

**See Also:**

- Overview of SODA Documents in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for an overview of SODA documents
- SODA Restrictions (Reference) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for restrictions that apply for SODA documents
- SODA_COLLECTION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about object type `SODA_DOCUMENT_T` constructors and getter methods

**Example 3-8 Creating a Document with JSON Content**

This example uses `SODA_DOCUMENT_T` constructors to create three documents, one of each content type. The example provides only the document content (which is the same for each).

The content parameter is different in each case; it specifies the SQL data type to use to store the content. The first document creation here uses content parameter **`j`**`_content`, which specifies `JSON` type content; the second uses **`v`**`_content`, which specifies `VARCHAR2` content; the third uses parameter **`c`**`_content`, which specifies `CLOB` content; the fourth uses parameter **`b`**`_content`, which specifies `BLOB` content.

Note that for the document of data type `JSON`, the literal `VARCHAR2` string input is wrapped in the `JSON` constructor. And to print the document content it is serialized to text using Oracle SQL function `json_serialize`.

After creating each document, the example uses getter methods to get the document content. Note that the getter method that is appropriate for each content storage type is used: `get_json()` for `JSON` content, and so on.

If database initialization parameter `compatible` is at least `20`, then the document with content type `JSON` is appropriate for writing to the collection created in Example 3-3,

because that collection has the *default* metadata. If `compatible` is less than 20 then the document with content type `BLOB` is appropriate for writing to that collection.

The default metadata indicates `JSON` document content (only `JSON` type content is accepted) in the first case and `BLOB` content in the second case (only `BLOB` type content is accepted). Trying to insert JSON data of the wrong content type into a collection raises an error.

However, as an exception, you can insert a `BLOB` JSON document into a `JSON` type collection. Other than this exception, only a document of the same SQL type can be inserted into a collection. For example, only a `VARCHAR2` JSON document can be inserted into a collection whose content column is of type `VARCHAR2`.

```
DECLARE
    j_doc  SODA_DOCUMENT_T;
    v_doc  SODA_DOCUMENT_T;
    b_doc  SODA_DOCUMENT_T;
    c_doc  SODA_DOCUMENT_T;
BEGIN
    -- Create JSON type document
    j_doc := SODA_DOCUMENT_T(j_content => JSON('{"name" : "Alexander"}'));
    DBMS_OUTPUT.put_line('JSON type doc content: ' ||
JSON_SERIALIZE(j_doc.get_json));

    -- Create VARCHAR2 document
    v_doc := SODA_DOCUMENT_T(v_content => '{"name" : "Alexander"}');
    DBMS_OUTPUT.put_line('VARCHAR2 doc content: ' || v_doc.get_varchar2);

    -- Create BLOB document
    b_doc := SODA_DOCUMENT_T(
               b_content => utl_raw.cast_to_raw('{"name" : "Alexander"}'));
    DBMS_OUTPUT.put_line('BLOB doc content: ' ||
                         utl_raw.cast_to_varchar2(b_doc.get_blob));

    -- Create CLOB document
    c_doc := SODA_DOCUMENT_T(c_content => '{"name" : "Alexander"}');
    DBMS_OUTPUT.put_line('CLOB doc content: ' || c_doc.get_clob);
END;
/
```

### Example 3-9 Creating a Document with Document Key and JSON Content

This example is similar to Example 3-8, but it provides the document key (`myKey`) as well as the document content.

```
DECLARE
    j_doc  SODA_DOCUMENT_T;
    v_doc  SODA_DOCUMENT_T;
    b_doc  SODA_DOCUMENT_T;
    c_doc  SODA_DOCUMENT_T;
BEGIN

    -- Create JSON type document
    j_doc := SODA_DOCUMENT_T('myKey' , j_content => JSON('{"name" :
"Alexander"}'));
    DBMS_OUTPUT.put_line('JSON type doc key: ' || j_doc.get_key);
```

```
    DBMS_OUTPUT.put_line('JSON doc content: ' ||
JSON_SERIALIZE(j_doc.get_json));

    -- Create VARCHAR2 document
    v_doc := SODA_DOCUMENT_T('myKey' , v_content => '{"name" :
"Alexander"}');
    DBMS_OUTPUT.put_line('VARCHAR2 doc key: ' || v_doc.get_key);
    DBMS_OUTPUT.put_line('VARCHAR2 doc content: ' ||
v_doc.get_varchar2);

    -- Create BLOB document
    b_doc := SODA_DOCUMENT_T('myKey' ,
                             b_content =>
utl_raw.cast_to_raw('{"name" : "Alexander"}'));
    DBMS_OUTPUT.put_line('BLOB doc key: ' || b_doc.get_key);
    DBMS_OUTPUT.put_line('BLOB doc content: ' ||
                         utl_raw.cast_to_varchar2(b_doc.get_blob));

    -- Create CLOB document
    c_doc := SODA_DOCUMENT_T('myKey' , c_content => '{"name" :
"Alexander"}');
    DBMS_OUTPUT.put_line('CLOB doc key: ' || c_doc.get_key);
    DBMS_OUTPUT.put_line('CLOB doc content: ' || c_doc.get_clob);
END;
/
```

### Related Topics

- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Finding Documents in Collections with SODA for PL/SQL
  You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method `count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.
- Removing Documents from a Collection with SODA for PL/SQL
  You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.

# 3.8 Inserting Documents into Collections with SODA for PL/SQL

To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.

Both method `insert_one()` and method `insert_one_and_get()` insert a document into a collection and automatically set the values of the creation time stamp, last-modified time stamp, and version (if the collection is configured to include these components and to generate the version automatically, as is the case by default).

When you insert a document, any document components that currently have `NULL` values (as a result of creating the document without providing those component values) are updated to have appropriate, automatically generated values. Thereafter, other SODA operations on a document can automatically update the last-modified timestamp and version components.

In addition to inserting the document, `insert_one_and_get` returns a result document, which contains the generated document components, such as the key, and which does not contain the content of the inserted document.

> **Note:**
>
> If the collection is configured with client-assigned document keys (which is not the default case), and the input document provides a key that identifies an existing document in the collection, then these methods throw an exception.

Method `insert_one_and_get()` accepts an optional second argument, `hint`, whose value is passed as a hint to the SQL code that underlies SODA. The `VARCHAR2` value for the argument uses the SQL hint syntax (that is, the hint text, without the enclosing SQL comment syntax `/*+...*/`). Use *only* hint `MONITOR` (turn on monitoring) or `NO_MONITOR` (turn off monitoring).

(You can use this to pass any SQL hints, but `MONITOR` and `NO_MONITOR` are the useful ones for SODA, and an inappropriate hint can cause the optimizer to produce a suboptimal query plan.)

**See Also:**

- INSERT_ONE Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `insert_one()`
- SODA_COLLECTION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `insert_one_and_get()`
- SODA_DOCUMENT_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods
- Monitoring Database Operations in *Oracle Database SQL Tuning Guide* for complete information about monitoring database operations
- MONITOR and NO_MONITOR Hints in *Oracle Database SQL Tuning Guide* for information about the syntax and behavior of SQL hints `MONITOR` and `NO_MONITOR`

**Example 3-10 Inserting a Document into a Collection**

This example creates a document and inserts it into a collection using `SODA_COLLECTION_T` method `insert_one()` .

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');
    document :=
      SODA_DOCUMENT_T(
        b_content => utl_raw.cast_to_raw('{"name" : "Alexander"}'));

    -- Insert a document
    status := collection.insert_one(document);
END;
/
```

**Example 3-11 Inserting a Document into a Collection and Getting the Result Document**

This example creates a document and inserts it into a collection using method `insert_one_and_get()`. It then gets (and prints) each of the generated components from the result document (which contains them). To obtain the components it uses `SODA_DOCUMENT_T` methods `get_key()`, `get_created_on()`, `get_last_modified()`, and `get_version()`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    ins_doc     SODA_DOCUMENT_T;
```

```
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');
    document :=
      SODA_DOCUMENT_T(
        b_content => utl_raw.cast_to_raw('{"name" : "Alexander"}'));
    ins_doc := collection.insert_one_and_get(document);

    -- Insert the document and get its components
    IF ins_doc IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Inserted document components:');
        DBMS_OUTPUT.put_line('Key: ' || ins_doc.get_key);
        DBMS_OUTPUT.put_line('Creation timestamp: '
                             || ins_doc.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: '
                             || ins_doc.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || ins_doc.get_version);
    END IF;
END;
/
```

**Related Topics**

- Saving Documents Into a Collection with SODA for PL/SQL
  You can use `SODA_DOCUMENT_T` method `save()` or `save_and_get()` to save documents into a collection, which means *inserting* them if they are new or *updating* them if they already belong to the collection. (Such an operation is sometimes called "upserting".)

**Related Topics**

- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.
- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.

# 3.9 Saving Documents Into a Collection with SODA for PL/SQL

You can use `SODA_DOCUMENT_T` method `save()` or `save_and_get()` to save documents into a collection, which means *inserting* them if they are new or *updating* them if they already belong to the collection. (Such an operation is sometimes called "upserting".)

Method `save_and_get()` is equivalent to `insert()`, and `save_and_get()` is equivalent to `insert_one_and_get()`, with this difference: If client-assigned keys are used, and if the document with the specified key already belongs to the collection, that document is *replaced* with the input document.

When inserting, these methods create the key automatically, unless the collection is configured with client-assigned keys and the key is provided in the input document.

Method `save_and_get()` accepts an optional second argument, `hint`, whose value is passed as a hint to the SQL code that underlies SODA. The `VARCHAR2` value for the argument uses the SQL hint syntax (that is, the hint text, without the enclosing SQL comment syntax `/*+...*/`). Use *only* hint `MONITOR` (turn on monitoring) or `NO_MONITOR` (turn off monitoring).

(You can use this to pass any SQL hints, but `MONITOR` and `NO_MONITOR` are the useful ones for SODA, and an inappropriate hint can cause the optimizer to produce a suboptimal query plan.)

> **See Also:**
>
> - Monitoring Database Operations in *Oracle Database SQL Tuning Guide* for complete information about monitoring database operations
> - MONITOR and NO_MONITOR Hints in *Oracle Database SQL Tuning Guide* for information about the syntax and behavior of SQL hints `MONITOR` and `NO_MONITOR`

**Example 3-12 Saving Documents Into a Collection with SODA for PL/SQL**

This example creates a collection and two documents, and saves the documents to the collection using method `save()`, *inserting* them. The example then changes the content of the documents and saves them again, which *replaces* the existing documents.

```
DECLARE
    coll  SODA_COLLECTION_T;
    md    VARCHAR2(4000);
    doca  SODA_DOCUMENT_T;
    docb  SODA_DOCUMENT_T;
    n     NUMBER;
BEGIN
    -- Create a collection and print its metadata
    md := '{"keyColumn":{"assignmentMethod":"CLIENT"}}';
    coll := DBMS_SODA.create_collection('SODAPLS_SAVE01', md);
    DBMS_OUTPUT.put_line('Coll: ' ||
                         json_query(coll.get_metadata, '$' pretty));

    -- Create two documents.
    doca := SODA_DOCUMENT_T('a', b_content =>
                                  utl_raw.cast_to_raw('{"a" : "value A" }'));
    docb := SODA_DOCUMENT_T('b', b_content =>
                                  utl_raw.cast_to_raw('{"b" : "value B" }'));

    -- Save the documents.  They are new, so this inserts them.
    n := coll.save(doca);
    DBMS_OUTPUT.put_line('Status: ' || n);
    n := coll.save(docb);
    DBMS_OUTPUT.put_line('Status: ' || n);
```

```
    -- Rewrite the content of the documents
    doca := SODA_DOCUMENT_T('a', b_content =>
                                  utl_raw.cast_to_raw('{"a" : "new value A" }'));
    docb := SODA_DOCUMENT_T('b', b_content =>
                                  utl_raw.cast_to_raw('{"b" : "new value B" }'));

    -- Save the existing documents, replacing them.
    n := coll.save(doca);
    DBMS_OUTPUT.put_line('Status: ' || n);
    n := coll.save(docb);
    DBMS_OUTPUT.put_line('Status: ' || n);
END;
/
```

**Related Topics**

- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.

# 3.10 SODA for PLSQL Read and Write Operations

A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

You typically use `SODA_OPERATION_T` to specify all SODA read operations, and all write operations other than document insertions and saves into a collection. You chain together `SODA_OPERATION_T` nonterminal methods to narrow the scope or otherwise condition or qualify a read or write operation.

**Nonterminal** methods return the same `SODA_OPERATION_T` instance on which they are invoked, which allows you to chain them together. The nonterminal methods are these:

- `acquire_lock()` — Lock documents (pessimistic locking).
- `as_of_scn()` — Access documents as of a given System Change Number (SCN). This uses Oracle Flashback Query: `SELECT AS OF`.

- `as_of_timestamp()` — Access documents as of a given date and time. This uses Oracle Flashback Query: `SELECT AS OF`.
- `filter()` — Filter documents using a query-by-example (QBE, also called a filter specification).
- `hint()` — Provide a hint, to turn real-time SQL monitoring of queries on and off.

  The `VARCHAR2` value for the argument uses the SQL hint syntax (that is, the hint text, without the enclosing SQL comment syntax `/*+...*/`). Use *only* hint `MONITOR` (turn on monitoring) or `NO_MONITOR` (turn off monitoring). The hint is simply passed down to the SQL code that underlies SODA.

  (You can use this to pass any SQL hints, but `MONITOR` and `NO_MONITOR` are the useful ones for SODA, and an inappropriate hint can cause the optimizer to produce a suboptimal query plan.)
- `key()` — Specify a particular document by its unique key.
- `keys()` — Specify particular documents by their unique keys.

  The maximum number of keys passed as argument must not exceed 1000, or else a runtime error is raised.
- `limit()` — Limit how many documents a read operation can return.
- `skip()` — Specify how many documents to skip when reading, before returning others.
- `version()` — Specify a particular version of a specified document.

A `SODA_OPERATION_T` **terminal** method at the end of the chain carries out the actual read or write operation. The terminal methods for *read* operations are these.

- `count()` — Count the documents found by the read operation.
- `get_cursor()` — Retrieve multiple documents. (Get a cursor over read operation results.)
- `get_data_guide()` — Obtain a data guide for the documents found by the read operation.
- `get_one()` — Retrieve a single document.

The terminal methods for *write* operations are these:

- `remove()` — Remove documents from a collection.
- `replace_one()` — Replace one document in a collection.
- `replace_one_and_get()` — Replace one document and return the new (result) document.

Unless documentation states otherwise, you can chain together any nonterminal methods, and you can end the chain with any terminal method. However, not all combinations make sense. For example, it does not make sense to chain method `version()` together with methods that do not uniquely identify the document, such as `keys()`.

**Related Topics**

- Finding Documents in Collections with SODA for PL/SQL
  You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal

method `count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.

- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.
- Removing Documents from a Collection with SODA for PL/SQL
  You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.

**See Also:**

- Using Oracle Flashback Query (SELECT AS OF) in *Oracle Database SQL Language Reference* for information about Oracle Flashback Query
- SODA_OPERATION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T`, including each of its methods
- Monitoring Database Operations in *Oracle Database SQL Tuning Guide* for complete information about monitoring database operations
- MONITOR and NO_MONITOR Hints in *Oracle Database SQL Tuning Guide* for information about the syntax and behavior of SQL hints `MONITOR` and `NO_MONITOR`
- SODA Restrictions (Reference) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for information about SODA restrictions

# 3.11 Finding Documents in Collections with SODA for PL/SQL

You can use `SODA_OPERATION_T` terminal method `get_one()` or `get_cursor()` to find one or multiple documents in a collection, respectively. You can use terminal method `count()` to count the documents in a collection. You can use nonterminal methods, such as `key()`, `keys()`, and `filter()`, to specify conditions for a find operation.

You can use nonterminal `SODA_OPERATION_T` method `hint()` to provide a SQL hint to turn SQL monitoring on or off. You can use nonterminal methods `as_of_scn()` and `as_of_timestamp()` to access documents as of a given system change number (SCN) or a given date and time.

**Note:**

Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

**See Also:**

- FIND Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `find()`
- SODA_OPERATION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about data type `SODA_OPERATION_T` and its methods
- SODA_DOCUMENT_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods
- JSON_QUERY in *Oracle Database SQL Language Reference* for information about SQL/JSON function `json_query`
- Monitoring Database Operations in *Oracle Database SQL Tuning Guide* for complete information about monitoring database operations
- MONITOR and NO_MONITOR Hints in *Oracle Database SQL Tuning Guide* for information about the syntax and behavior of SQL hints `MONITOR` and `NO_MONITOR`

**Example 3-13 Finding All Documents in a Collection Using SODA For PL/SQL**

This example uses `SODA_COLLECTION_T` method `find()` and `SODA_OPERATION_T` method `getCursor()` to obtain a cursor for a query result list that contains each document in a collection. It then uses the cursor in a `WHILE` statement to get and print the content of each document in the result list, as a string. Finally, it closes the cursor.

It uses `SODA_DOCUMENT_T` methods `get_key()`, `get_blob()`, `get_created_on()`, `get_last_modified()`, and `get_version()`, to get the document components, which it prints. It passes the document content to SQL/JSON function `json_query` to pretty-print (using keyword `PRETTY`).

**Note:**

To avoid resource leaks, *close* any cursor that you no longer need.

```
DECLARE
    collection    SODA_COLLECTION_T;
    document      SODA_DOCUMENT_T;
    cur           SODA_CURSOR_T;
    status        BOOLEAN;
BEGIN
    -- Open the collection to be queried
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Open the cursor to fetch the documents.
    cur := collection.find().get_cursor();

    -- Loop through the cursor
    WHILE cur.has_next
```

```
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: '
            || json_query(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: '
            || document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: '
            || document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;

    -- IMPORTANT: You must close the cursor, to release resources.
    status := cur.close;
END;
/
```

**Example 3-14 Finding the Unique Document That Has a Given Document Key Using SODA For PL/SQL**

This example uses `SODA_COLLECTION_T` methods `find()`, `key()`, and `get_one()` to find the unique document whose key is `"key1"`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Find a document using a key
    document := collection.find().key('key1').get_one;

    IF document IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || document.get_key);
        DBMS_OUTPUT.put_line('Content: '
          || JSON_QUERY(document.get_blob, '$' PRETTY));
        DBMS_OUTPUT.put_line('Creation timestamp: '
          || document.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: '
          || document.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || document.get_version);
    END IF;
END;
/
```

### Example 3-15 Finding Multiple Documents with Specified Document Keys Using SODA For PL/SQL

This example defines key list **myKeys**, with (string) keys "key1", "key2", and "key3". It then finds the documents that have those keys, and it prints their components. SODA_COLLECTION_T method keys() specifies the documents with the given keys.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_CURSOR_T;
    status      BOOLEAN;
    myKeys      SODA_KEY_LIST_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Set the keys list
    myKeys := SODA_KEY_LIST_T('key1', 'key2', 'key3');

    -- Find documents using keys
    cur := collection.find().keys(myKeys).get_cursor;

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: '
            || json_query(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: '
            || document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: '
            || document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;
    status := cur.close;
END;
/
```

### Example 3-16 Finding Documents with a Filter Specification Using SODA For PL/SQL

SODA_OPERATION_T method filter() provides a powerful way to filter JSON documents in a collection. Its parameter is a JSON query-by-example (QBE, also called a filter specification).

The syntax of filter specifications is an expressive pattern-matching language for JSON documents. This example uses only a very simple QBE, just to indicate how you make use of one in SODA for PL/SQL.

This example does the following:

1. Creates a filter specification that looks for all JSON documents whose `name` field has value `"Alexander"`.
2. Uses the filter specification to find the matching documents.
3. Prints the components of each document.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_CURSOR_T;
    status      BOOLEAN;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the filter specification (QBE)
    qbe := '{"name" : "alexander"}';

    -- Open a cursor for the filtered documents
    cur := collection.find().filter(qbe).get_cursor;

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: '
            || JSON_QUERY(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: '
            || document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: '
            || document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;
    status := cur.close;
END;
/
```

**See Also:**

- Overview of SODA Filter Specifications (QBEs) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for an introduction to SODA filter specifications
- SODA Filter Specifications (Reference) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for reference information about SODA filter specifications

**Example 3-17 Specifying Pagination Queries with Methods skip() and limit() Using SODA For PL/SQL**

This example uses `SODA_OPERATION_T` methods `filter()`, `skip()` and `limit()` in a pagination query.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_Cursor_T;
    status      BOOLEAN;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the filter
    qbe := '{"name" : "Alexander"}';

    -- Find all documents that match the QBE, skip over the first 1000
    -- of them, limit the number of returned documents to 100
    cur := collection.find().filter(qbe).skip(1000).limit(100).get_cursor;

    -- Loop through the cursor
    WHILE cur.has_next
    LOOP
      document := cur.next;
      IF document IS NOT NULL THEN
          DBMS_OUTPUT.put_line('Document components:');
          DBMS_OUTPUT.put_line('Key: ' || document.get_key);
          DBMS_OUTPUT.put_line('Content: ' ||
                               JSON_QUERY(document.get_blob, '$' PRETTY));
          DBMS_OUTPUT.put_line('Creation timestamp: ' ||
                               document.get_created_on);
          DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                               document.get_last_modified);
          DBMS_OUTPUT.put_line('Version: ' || document.get_version);
      END IF;
    END LOOP;
    status := cur.close;
END;
/
```

**Example 3-18 Specifying Document Version Using SODA For PL/SQL**

This example uses `SODA_OPERATION_T` method `version()` to specify the document version. This is useful for implementing optimistic locking, when used with the terminal methods for write operations.

You typically use `version()` together with method `key()`, which specifies the document. You can also use `version()` with methods `keyLike()` and `filter()`, provided they identify at most one document.

```
DECLARE
    collection  SODA_COLLECTION_T;
```

```
    document    SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Find a particular version of the document that has a given key
    document := collection.find().key('key1').version('version1').get_one;

    IF document IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || document.get_key);
        DBMS_OUTPUT.put_line('Content: ' ||
          JSON_QUERY(document.get_blob, '$' PRETTY));
        DBMS_OUTPUT.put_line('Creation timestamp: '
          || document.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: '
          || document.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || document.get_version);
    END IF;
END;
/
```

### Example 3-19 Counting the Number of Documents Found

This example uses `SODA_OPERATION_T` method `count()` to get a count of all of the documents in the collection. It then gets a count of all of the documents that are returned by a filter specification (QBE).

```
DECLARE
    collection  SODA_COLLECTION_T;
    num_docs    NUMBER;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Count of all documents in the collection
    num_docs := collection.find().count;
    DBMS_OUTPUT.put_line('Count (all): ' || num_docs);

    -- Set the filter
    qbe := '{"name" : "Alexander"}';

    -- Count of all documents in the collection that match
    -- a filter spec
    num_docs := collection.find().filter(qbe).count;
    DBMS_OUTPUT.put_line('Count (filtered): ' || num_docs);
/
```

### Example 3-20 Retrieving the Documents of a Collection at a Time in the Past (Flashback) Using SODA For PL/SQL

This code uses `SODA_OPERATION_T` method `as_of_timestamp()` to open a cursor for the documents that were in collection `myCollectionName` on April 27th, 2021 at UTC time 5:00, that is, the time represented by ISO 8601 date-time string `2021-04-27T05:00:00Z`.

```
DECLARE
  coll    SODA_COLLECTION_T;
  cur     SODA_CURSOR_T;
  b       BOOLEAN;
BEGIN
  -- Open the collection to be queried
  coll := DBMS_SODA.open_collection('myCollectionName');

  -- Specify SCN to retrieve documents as it existed then
  cur := coll.find().as_of_timestamp('2021-04-27T05:00:00Z').get_cursor;
  b := cur.close;
END;
/
```

Similarly, this code uses `SODA_OPERATION_T` method `as_of_scn()` to access the documents present at a particular time using an Oracle Database **system change number** (SCN), which is a logical, internal time stamp.

```
DECLARE
  coll    SODA_COLLECTION_T;
  cur     SODA_CURSOR_T;
  b       BOOLEAN;
BEGIN
  -- Open the collection to be queried
  coll := DBMS_SODA.open_collection('myCollectionName');

  -- Specify SCN to retrieve documents as it existed then
  cur := coll.find().as_of_scn(2068287).get_cursor;
  b := cur.close;
END;
/
```

### Example 3-21 Using Full-Text Search To Find Documents in a Heterogeneous Collection Using SODA For PL/SQL

This example uses QBE operator `$textContains` to perform a full-text search of a **heterogeneous** collection, which is one that has the media type column. For example, Microsoft Word, Portable Document Format (PDF), and plain-text documents can all be searched using `$textContains`.

(You use QBE operator `$contains`, not `$textContains`, to perform full-text search of a collection of *JSON* documents.)

The search pattern in this example is `Rogers`, which means search for that literal text anywhere in a document of collection `myTextCollection`.

```
DECLARE
  coll    SODA_COLLECTION_T;
  cur     SODA_CURSOR_T;
  qbe     VARCHAR2(100);
  b       BOOLEAN;
BEGIN
  -- Open the collection to be queried
  coll := DBMS_SODA.open_collection('myTextCollection');

  -- Use $textContains operator to specify the subtring
  qbe := '{"$textContains" : "Rogers"}';
  cur := coll.find().filter(qbe).get_cursor;
  b := cur.close;
END;
/
```

The syntax of the search-pattern value for `$textContains` is the same as that for SQL function `contains`, and the resulting behavior is the same. This means, for instance, that you can query for text that is near some other text, or query use fuzzy pattern-matching. (If the search-pattern argument contains a character or a word that is reserved with respect to Oracle Text search then you must escape that character or word.)

In order to use operator `$textContains` to search a collection, you must first have defined an Oracle Text search index on the content column of the collection, using SQL. This SQL code does that; it creates index `mySearchIndex` on content column `myContentColumn` of collection `myTextCollection`.

```
CREATE SEARCH INDEX mySearchIndex ON
  myTextCollection(myContentColumn)
```

### Related Topics

- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

> **See Also:**
>
> - Overview of SODA Document Collections in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*
> - Media Type Column Name in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*
> - CREATE SEARCH INDEX in *Oracle Text Reference*

# 3.12 Replacing Documents in a Collection with SODA for PL/SQL

You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

In addition to replacing the content, methods `replace_one()` and `replace_one_and_get()` update the values of the last-modified timestamp and the version. Replacement does *not* change the document key or the creation timestamp.

> **See Also:**
>
> - FIND Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `find()`
> - SODA_OPERATION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about data type `SODA_OPERATION_T` and its methods
> - REPLACE_ONE Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T` method `replace_one()`
> - REPLACE_ONE_AND_GET Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T` method `replace_one_and_get()`
> - ACQUIRE_LOCK Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T` method `acquire_lock()`
> - SODA_DOCUMENT_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods

**Example 3-22 Replacing a Document, Given Its Key, and Getting the Result Document Using SODA For PL/SQL**

This example replaces a document in a collection, given its key. It then gets (and prints) the key and the generated components from the result document. To obtain the

components it uses `SODA_DOCUMENT_T` methods `get_key()`, `get_created_on()`, `get_last_modified()`, and `get_version()`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    new_doc     SODA_DOCUMENT_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    document := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" : "Sriky"}'));
    new_doc := collection.find().key('key1').replace_one_and_get(document);

    IF new_doc IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || new_doc.get_key);
        DBMS_OUTPUT.put_line('Creation timestamp: ' || new_doc.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                             new_doc.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || new_doc.get_version);
    END IF;
END;
/
```

### Example 3-23 Replacing a Particular Version of a Document Using SODA For PL/SQL

To implement **optimistic locking** when replacing a document, you can chain together methods `key()` and **`version`**`()`, as in this example. The write operation (`replace_one_and_get`) optimistically tries to modify the latest version known (`version1`, here).

If the write were to fail (returning `NULL`) because some other transaction modified the document since we last read it, then we would need to repeatedly try again until writing succeeds: reread the document, get its new version, and specify that version in a new write attempt. This example shows only a single write attempt.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    new_doc     SODA_DOCUMENT_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Replace content of version 'version1' of the document that has key 'key1'
    new_doc := SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"name" : "Sriky"}'));
    document :=
collection.find().key('key1').version('version1').replace_one_and_get(new_doc);

    IF document IS NOT NULL THEN
        DBMS_OUTPUT.put_line('Document components:');
        DBMS_OUTPUT.put_line('Key: ' || document.get_key);
        DBMS_OUTPUT.put_line('Content: ' ||
                             JSON_QUERY(document.get_blob, '$' PRETTY));
```

```
        DBMS_OUTPUT.put_line('Creation timestamp: ' || document.get_created_on);
        DBMS_OUTPUT.put_line('Last modified timestamp: ' ||
                             document.get_last_modified);
        DBMS_OUTPUT.put_line('Version: ' || document.get_version);
    END IF;
END;
/
```

**Example 3-24 Locking a Document For Update (Replacement) Using SODA For PL/SQL**

This example uses nonterminal method `acquire_lock()` to lock a document while replacing it. The document is selected by its key. Method `acquire_lock()` provides **pessimistic locking**, which prevents other users from interfering with the update operation. A commit or a rollback releases the lock. The example rolls back the transaction for the operation if any error was raised.

```
DECLARE
  coll    SODA_COLLECTION_T;
  doc1    SODA_DOCUMENT_T;
  doc2    SODA_DOCUMENT_T;
  k       VARCHAR2(255) := 'key-0';
  n       NUMBER;
BEGIN
  coll := DBMS_SODA.open_collection('myCollectionName');

  -- Get the document with a lock, using its key.
  doc1 := coll.find().key(k).acquire_lock().get_One;

  -- Construct a new, replacement document.
  doc2 := SODA_DOCUMENT_T(
    key => k,
    b_content => utl_raw.cast_to_raw('{"name" : "Scott", "age" : 35}'));

  -- Replace the document, specifying its key.
  n := coll.replace_one(k, doc2);

  -- Commit the transaction, releasing the lock.
  COMMIT;
  DBMS_OUTPUT.put_line('Transaction is committed');

-- Catch exceptions and roll back if an error was raised.
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.put_line (SQLERRM);
        ROLLBACK;
        DBMS_OUTPUT.put_line('Transaction has been rolled back');
END;
/
```

**Related Topics**

- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.
- Saving Documents Into a Collection with SODA for PL/SQL
  You can use `SODA_DOCUMENT_T` method `save()` or `save_and_get()` to save documents into a collection, which means *inserting* them if they are new or *updating* them if they already belong to the collection. (Such an operation is sometimes called "upserting".)
- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.

**Related Topics**

- Handling Transactions with SODA for PL/SQL
  As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.
- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.

# 3.13 Removing Documents from a Collection with SODA for PL/SQL

You can remove documents from a collection by chaining together `SODA_OPERATION_T` method `remove()` with nonterminal method `key()`, `keys()`, or `filter()` to identify documents to be removed. You can optionally make use of additional nonterminal methods such as `version()`.

> **Note:**
>
> Data type `SODA_OPERATION_T` was added to SODA for PL/SQL in Oracle Database 18.3. You need that database release or later to use it.

**See Also:**

- FIND Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `find()`
- SODA_OPERATION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about data type `SODA_OPERATION_T` and its methods
- REMOVE Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_OPERATION_T` method `remove()`
- REMOVE_ONE Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `remove_one()`
- SODA_DOCUMENT_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_DOCUMENT_T` getter methods

**Example 3-25 Removing a Document from a Collection Using a Document Key**

This example removes the document whose document key is `"key1"`. The removal status (1 if the document was removed; 0 if not) is returned and printed.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Remove document that has key 'key1'
    status := collection.find().key('key1').remove;

    -- Count is 1 if  document was found
    IF status = 1 THEN
        DBMS_OUTPUT.put_line('Document was removed!');
    END IF;
END;
/
```

**Example 3-26 Removing a Particular Version of a Document**

To implement optimistic locking when removing a document, you can chain together methods `key()` and `version()`, as in this example.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');
```

```
    -- Remove version 'version1' of the document that has key 'key1'.
    status := collection.find().key('key1').version('version1').remove;

    -- Count is 1, if specified version of document with key 'key1' is found
    IF status = 1 THEN
        DBMS_OUTPUT.put_line('Document was removed!');
    END IF;
END;
/
```

### Example 3-27 Removing Documents from a Collection Using Document Keys

This example removes the documents whose keys are `key1`, `key2`, and `key3`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    document    SODA_DOCUMENT_T;
    cur         SODA_CURSOR_T;
    num_docs    NUMBER;
    myKeys      SODA_KEY_LIST_T;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the keys list
    myKeys := SODA_KEY_LIST_T('key1','key2','key3');

    -- Remove documents using keys
    num_docs := collection.find().keys(myKeys).remove;

    DBMS_OUTPUT.put_line('Number of documents removed: ' || num_docs);
END;
/
```

### Example 3-28 Removing JSON Documents from a Collection Using a Filter

This example uses a filter to remove the JSON documents whose `greeting` field has value `"hello"`. It then prints the number of documents removed.

```
DECLARE
    collection  SODA_COLLECTION_T;
    num_docs    NUMBER;
    qbe         VARCHAR2(128);
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the filter specification
    qbe := '{ "greeting" : "hello" }';

    -- Get a count of all documents in the collection that match the QBE
    num_docs := collection.find().filter(qbe).remove;
    DBMS_OUTPUT.put_line('Number of documents removed: ' || num_docs);
```

```
END;
/
```

**Related Topics**

- SODA for PLSQL Read and Write Operations
  A `SODA_OPERATION_T` instance is returned by method `find()` of `SODA_COLLECTION_T`. You can chain together `SODA_OPERATION_T` methods, to specify read and write operations against a collection.

# 3.14 Truncating a Collection (Removing All Documents) with SODA for PL/SQL

You can use `SODA_COLLECTION_T` method `truncate()` to empty, or **truncate**, a collection, which means remove all of its documents.

**Example 3-29 Truncating a Collection**

This example uses `SODA_COLLECTION_T` method `truncate()` to remove all documents from `collection`.

```
DECLARE
    collection SODA_COLLECTION_T;
    document   SODA_DOCUMENT_T;
    status     NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Truncate the collection
    status := collection.truncate;

    -- Count is 1 if  document was found
    IF status = 1 THEN
        DBMS_OUTPUT.put_line('Collection was truncated!');
    END IF;
END;
```

# 3.15 Indexing the Documents in a Collection with SODA for PL/SQL

You index the documents in a SODA collection with `SODA_COLLECTION_T` method `create_index()`. Its input parameter is a textual JSON index specification. This can

specify support for B-tree, spatial, full-text, and ad hoc indexing, and it can specify support for a JSON data guide.

> **Note:**
>
> SODA for PL/SQL support for indexing was added in Oracle Database 18.3. You need that database release or later to use this SODA feature.

A JSON search index is used for full-text and ad hoc structural queries, and for persistent recording and automatic updating of JSON data-guide information.

An Oracle Spatial and Graph index is used for GeoJSON (spatial) data.

You can drop an index on a SODA collection with `SODA_COLLECTION_T` method `drop_Index()`.

You can obtain an index specification or all index specifications for a collection, using `SODA_COLLECTION_T` method `get_index()` or `list_indexes()`, respectively. The value returned by method `list_indexes()` is an instance of data type `SODA_INDEX_LIST_T`, which is a PL/SQL collection of `VARCHAR2` index specifications.

For method `get_index()` you provide the index name, and optionally the relevant database schema name, as arguments. (The values used for the schema and index names are identifiers in the data dictionary. In particular, they must follow the same letter case, so if they were created in SQL without using any double quotation marks then they must be uppercase.)

> **See Also:**
>
> - Overview of SODA Indexing in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for an overview of using SODA indexing
> - SODA Index Specifications (Reference) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for information about SODA index specifications
> - JSON Search Index for Ad Hoc Queries and Full-Text Search in *Oracle Database JSON Developer's Guide* for information about JSON search indexes
> - Persistent Data-Guide Information: Part of a JSON Search Index in *Oracle Database JSON Developer's Guide* for information about persistent data-guide information as part of a JSON search index
> - Using GeoJSON Geographic Data in *Oracle Database JSON Developer's Guide* for information about spatial indexing of GeoJSON data
> - Database Object Naming Rules in *Oracle Database SQL Language Reference* for information about database identifier syntax

### Example 3-30 Creating a B-Tree Index for a JSON Field with SODA for PL/SQL

This example creates a B-tree non-unique index for numeric field `address.zip` of the JSON documents in collection `myCollectionName`.

```
DECLARE
    collection  SODA_COLLECTION_T;
    spec        VARCHAR2(700);
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the index specification
    spec := '{"name"   : "ZIPCODE_IDX",
              "fields" : [{"path"     : "address.zip",
                           "datatype" : "number",
                           "order"    : "asc"}]}';
    -- Create the index
    status := collection.create_index(spec);
    DBMS_OUTPUT.put_Line('Status: ' || status);
END;
/
```

### Example 3-31 JSON Search Indexing with SODA for PL/SQL

This example indexes the documents in collection `myCollectionName` for ad hoc queries and full-text search (queries using QBE operator `$contains`), and it automatically accumulates and updates data-guide information about your JSON documents (aggregate structural and type information). The index specification has only field `name` (no field `fields`).

```
DECLARE
    collection  SODA_COLLECTION_T;
    spec        VARCHAR2(700);
    status      NUMBER;
BEGIN
    -- Open the collection
    collection := DBMS_SODA.open_collection('myCollectionName');

    -- Define the index specification
    indexSpec := '{"name" : "SEARCH_AND_DATA_GUIDE_IDX"}';

    -- Create the index
    status := collection.create_index(indexSpec);
    DBMS_OUTPUT.put_Line('Status: ' || status);
END;
/
```

The simple index specification it uses is equivalent to this one, which makes explicit the default values:

```
{"name" : "SEARCH_AND_DATA_GUIDE_IDX",
 "dataguide" : "on",
 "search_on" : "text_value"}
```

If you instead wanted *only ad hoc* (search) indexing then you would explicitly specify a value of `"off"` for field `dataguide`. If you instead wanted *only data-guide* support then you would explicitly specify a value of `"none"` for field `search_on`.

> **Note:**
>
> To create a data guide-enabled JSON search index, or to data guide-enable an existing JSON search index, you need database privilege `CTXAPP` and Oracle Database Release 12c (12.2.0.1) or later.

**Example 3-32 Dropping an Index with SODA for PL/SQL**

This example uses `SODA_COLLECTION_T` method `drop_index()` to drop index `myIndex` on collection `myCollectionName`.

```
DECLARE
    coll    SODA_COLLECTION_T;
    status  NUMBER;
BEGIN
    -- Open the collection
    coll := dbms_soda.open_Collection('myCollectionName');

    -- Drop the index using name
    status := coll.drop_index('myIndex');
    DBMS_OUTPUT.put_Line('Status: ' || status);
END;
/
```

**Example 3-33 Getting an Index Specification with SODA for PL/SQL**

This example uses method `get_index()` to get the specification used to define the index named `ZIPCODE_IDX` in database schema (user name) `MY_SCHEMA` for the documents in collection `myCollectionName`. Each of these names must be written just as it appears in the data dictionary.

```
DECLARE
  spec    VARCHAR2(1000);
  coll    SODA_Collection_T;
BEGIN
  coll := DBMS_SODA.open_collection('myCollectionName');
  spec := coll.get_index('ZIPCODE_IDX', 'MY_SCHEMA');
  DBMS_OUTPUT.put_line(json_query(spec, '$' pretty));
END;
/
```

**Example 3-34 Getting All Index Specifications For a Collection with SODA for PL/SQL**

This example uses method `list_indexes()` to retrieve, in variable `idx`, all index specifications defined for the documents in collection `myCollectionName`. It then prints them, along with their count (obtained using method `count` for data type `SODA_INDEX_LIST_T`).

```
DECLARE
  coll    SODA_COLLECTION_T;
  idx     SODA_INDEX_LIST_T;
BEGIN
  coll := DBMS_SODA.open_collection('myCollectionName');
  idx := coll.list_indexes;
  DBMS_OUTPUT.put_line('Number of indexes: ' || idx.COUNT);

  if (idx.COUNT <> 0) then
    for i in idx.FIRST..idx.LAST
    loop
      DBMS_OUTPUT.put_line('Index ' || i || ': ');
      DBMS_OUTPUT.put_line(json_query(idx(i), '$' pretty));
    end loop;
  else
    DBMS_OUTPUT.put_line('No indexes defined on this collection');
  end if;
END;
/
```

# 3.16 Getting a Data Guide for a Collection with SODA for PL/SQL

You can use `SODA_COLLECTION_T` method `get_data_guide()` or terminal `SODA_OPERATION_T` method `get_data_guide()` to obtain a data guide for a collection. A **data guide** is a JSON document that summarizes the structural and type information of the JSON documents in the collection. It records metadata about the fields used in those documents.

> **Note:**
>
> SODA for PL/SQL support for JSON data guide was added in Oracle Database 18.3. You need that database release or later to use this SODA feature.

There are two alternative ways to create a data guide for a collection, using two different methods named `get_data_guide()`:

- Use terminal **`SODA_OPERATION_T`** method `get_data_guide()` together with operation `sample()` or a query-by-example (QBE) `filter()` operation. This creates a data guide dynamically from scratch, for only the documents selected by

your query. You can thus *limit the set of documents* on which the data guide is based. Example 3-35 illustrates this.

(This method corresponds to using SQL function `json_dataguide`.)

- Use **`SODA_COLLECTION_T`** method `get_data_guide()`. This always creates a data guide based on *all* documents in the collection. Example 3-36 illustrates this.

  This method makes use of *persistent data-guide information* that is stored as part of a JSON search index, so before you can use this method you must first create a data guide-enabled JSON search index on the collection. Example 3-31 shows how to do that. The data-guide information in the index is persistent, and is updated automatically as new JSON content is added.

  (This method corresponds to using PL/SQL function `get_index_dataguide`.)

The index-based `SODA_COLLECTION_T` method incurs an ongoing cost of updating relevant data persistently: document writes (creation and updating) entail index updates. But because data-guide information is readily available in the index, it need not be gathered from scratch when generating the data-guide document.

Because the `SODA_OPERATION_T` method starts from scratch each time, a typical use of it involves applying the method to only a random sample of documents or to only the documents that satisfy some filter (QBE). You can use `SODA_OPERATION_T` method `sample()` to obtain a random sample, as shown in Example 3-35.

> **See Also:**
>
> - JSON Data Guide in *Oracle Database JSON Developer's Guide*
> - GET_DATA_GUIDE Function for type `SODA_OPERATION_T` in *Oracle Database PL/SQL Packages and Types Reference*
> - GET_DATA_GUIDE Function for type `SODA_COLLECTION_T` in *Oracle Database PL/SQL Packages and Types Reference*
> - SELECT statement, *sample_clause*, in *Oracle Database SQL Language Reference* for information about using SQL to select a sample of data

**Example 3-35 Creating a Data Guide Dynamically with SODA for PL/SQL**

This example uses `SODA_OPERATION_T` terminal method `get_data_guide()`, together with nonterminal operation `sample()`[1], to obtain a data guide for a random sample of documents in collection `MyCollectionName`. The percent chance for any given document to be included in the sample is 40% (argument value `40`).

The example pretty-prints the content of the data-guide document in the flat format. Finally, it frees the temporary LOB used for the data-guide document.

You use operation `sample()` only for *read* operations — it is ignored for write operations. Creating a dynamic data guide is a typical use case for `sample()`. You can also use it with `SODA_OPERATION_T` terminal method `get_cursor()`.

[1] Operation `sample()` corresponds to the *sample_clause* of a SQL SELECT statement.

Another common way to limit the documents represented by a dynamically created data guide, besides using a random sample, is to use a query-by-example (QBE) `filter()` operation in place of operation `sample()`.

```
DECLARE
    coll         SODA_COLLECTION_T;
    qbe          VARCHAR2(100);
    dataguide    CLOB;
    dgflag       PLS_INTEGER;
    dgformat     PLS_INTEGER;
BEGIN
    -- Open the collection.
    coll := DBMS_SODA.open_Collection('myCollectionName');

    dgflag   := DBMS_SODA.DATAGUIDE_PRETTY;
    dgformat := DBMS_SODA.DATAGUIDE_FORMAT_FLAT;

    -- Get dynamic data guide for the collection.
    dataguide := coll.find().sample(40).get_data_guide(flag   => dgflag,
                                                       format =>
dgformat);
    DBMS_OUTPUT.put_line(dataguide);

    -- Important: Free the temporary LOB.
    IF DBMS_LOB.isTemporary(dataguide) = 1
    THEN
        DBMS_LOB.freeTemporary(dataguide);
    end if;
END;
/
```

**See Also:**

- Data-Guide Formats and Ways of Creating a Data Guide in *Oracle Database JSON Developer's Guide* for information about flat and hierarchical data-guide formats
- A Flat Data Guide For Purchase-Order Documents in *Oracle Database JSON Developer's Guide* for an example of a pretty-printed flat-format data guide
- SELECT statement, *sample_clause*, in *Oracle Database SQL Language Reference*

**Example 3-36 Creating a Data Guide Using a JSON Search Index with SODA for PL/SQL**

This example uses `SODA_COLLECTION_T` method `get_data_guide()` to obtain a data guide for all documents in collection `MyCollectionName`. To use this method, a data guide-enabled JSON search index must be defined on the collection.

The example uses SQL/JSON function `json_query` to pretty-print the content of the data-guide document. Finally, it frees the temporary LOB used for the data-guide document.

```
DECLARE
    collection SODA_COLLECTION_T;
    dataguide  CLOB;
BEGIN
    -- Open the collection.
    collection := DBMS_SODA.open_Collection('myCollectionName');

    -- Get the data guide for the collection.
    dataguide := collection.get_data_guide;
    DBMS_OUTPUT.put_line(json_query(dataguide, '$' pretty));

    -- Important: Free the temporary LOB.
    IF DBMS_LOB.isTemporary(dataguide) = 1
    THEN
        DBMS_LOB.freeTemporary(dataguide);
    end if;
END;
/
```

**Related Topics**

- Creating a View from a Data Guide with SODA for PL/SQL
  You can use `SODA_COLLECTION_T` method `create_view_from_dg()` to create a database view with relational columns, whose names and values are taken from the scalar JSON fields specified in the data guide. A data guide-enabled JSON search index is *not* required for this; the data guide itself is passed to the method.

# 3.17 Creating a View from a Data Guide with SODA for PL/SQL

You can use `SODA_COLLECTION_T` method `create_view_from_dg()` to create a database view with relational columns, whose names and values are taken from the scalar JSON fields specified in the data guide. A data guide-enabled JSON search index is *not* required for this; the data guide itself is passed to the method.

**Example 3-37 Creating a Relational View from a JSON Data Guide with SODA for PL/SQL**

This example, like Example 3-36, gets and pretty-prints a JSON data guide for a collection. It then uses `create_view_from_dg()` to create a relational view with columns that are based on the scalar JSON fields in the data guide. Finally, it frees the temporary LOB used for the data-guide document.

```
DECLARE
    coll  SODA_COLLECTION_T;
    dg    CLOB;
    n     NUMBER;
BEGIN
    -- Open a collection
    coll := DBMS_SODA.open_collection('myCollectionName');

    -- Get and print the data guide for the collection
```

```
    dg := coll.get_data_guide;
    DBMS_OUTPUT.put_line(json_query(dg, '$' pretty));

    -- Create view from data guide
    n = coll.create_view_from_dg('MY_VIEW_FROM_DG', dg);

    -- Free the temporary LOB containing the data guide
    if DBMS_LOB.isTemporary(dg) = 1
    then
      DBMS_LOB.freeTemporary(dg);
    end if;
END;
```

**Related Topics**

- Getting a Data Guide for a Collection with SODA for PL/SQL
  You can use `SODA_COLLECTION_T` method `get_data_guide()` or terminal `SODA_OPERATION_T` method `get_data_guide()` to obtain a data guide for a collection. A **data guide** is a JSON document that summarizes the structural and type information of the JSON documents in the collection. It records metadata about the fields used in those documents.

# 3.18 Handling Transactions with SODA for PL/SQL

As usual in PL/SQL and SQL, you can treat individual SODA read and write operations, or groups of them, as a transaction. To commit a transaction, use a SQL `COMMIT` statement. If you want to roll back changes, use a SQL `ROLLBACK` statement.

SODA operations `DBMS_SODA.create_collection` and `DBMS_SODA.drop_collection` do *not* automatically commit before or after they perform their action. This differs from the behavior of SQL DDL statements, which commit both before and after performing their action.

One consequence of this is that, before a SODA collection can be dropped, any outstanding write operations to it must be explicitly committed or rolled back — you must explicitly use SQL `COMMIT` or `ROLLBACK`. This is because `DBMS_SODA.drop_collection` does not itself issue commit before it performs its action. In this, the behavior of `DBMS_SODA.drop_collection` differs from that of a SQL `DROP TABLE` statement.

**See Also:**

- COMMIT in *Oracle Database SQL Language Reference* for information about the SQL `COMMIT` statement
- ROLLBACK in *Oracle Database SQL Language Reference* for information about the SQL `ROLLBACK` statement
- SODA_COLLECTION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `insert_one()`

### Example 3-38 Transaction Involving SODA Document Insertion and Replacement

This example shows the use of SQL `COMMIT` and `ROLLBACK` statements in an anonymous PL/SQL block. It opens a SODA collection, inserts a document, and then replaces its content. The combination of the document insertion and document content replacement operations is *atomic*: a single transaction.

```
DECLARE
    collection SODA_COLLECTION_T;
    status NUMBER;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    status := collection.insert_one(
                SODA_Document_T(
                  b_content => utl_raw.cast_to_raw('{"a":"aval", "b":"bval",
"c":"cval"}')));
    status := collection.replace_one(
                'key1',
                SODA_DOCUMENT_T(
                  b_content => utl_raw.cast_to_raw('{"x":"xval",
"y":"yval"}')));
    -- Commit the transaction
    COMMIT;
    DBMS_OUTPUT.put_line('Transaction is committed');
-- Catch exceptions and roll back if an error is raised
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.put_line (SQLERRM);
    ROLLBACK;
    DBMS_OUTPUT.put_line('Transaction has been rolled back');
END;
/
```

#### Related Topics

- Dropping a Document Collection with SODA for PL/SQL
  You use PL/SQL function `DBMS_SODA.drop_collection` to drop a document collection.
- Inserting Documents into Collections with SODA for PL/SQL
  To insert a document into a collection, you invoke `SODA_COLLECTION_T` method (member function) **`insert_one()`** or **`insert_one_and_get()`**. These methods create document keys automatically, unless the collection is configured with client-assigned keys and the input document provides the key.
- Replacing Documents in a Collection with SODA for PL/SQL
  You can chain together `SODA_OPERATION_T` replace-operation method `replace_one()` or `replace_one_and_get()` with nonterminal method `key()` to uniquely identify a document to be replaced. You can optionally make use of additional nonterminal methods such as `version()` and `filter()`. You can use nonterminal method `acquire_lock()` to lock a document for updating.

4

# SODA Collection Configuration Using Custom Metadata

SODA collections are highly configurable. You can customize collection metadata, to obtain different behavior from that provided by default.

> **Note:**
>
> You can customize collection metadata to obtain different behavior from that provided by default. However, changing some components requires familiarity with Oracle Database concepts, such as SQL data types. Oracle recommends that you do *not* change such components unless you have a compelling reason. Because SODA collections are implemented on top of Oracle Database tables (or views), many collection configuration components are related to the underlying table configuration.
>
> For example, if you change the content column type from the default value to `VARCHAR2`, then you must understand the implications: content size for `VARCHAR2` is limited to 32K bytes, character-set conversion can take place, and so on.

- Getting the Metadata of an Existing Collection
  You use `SODA_COLLECTION_T` method `get_metadata()` to get all of the metadata for a collection, as a JSON document.
- Creating a Collection That Has Custom Metadata
  To create a document collection that has custom metadata, you pass its metadata, as JSON data, to PL/SQL function `DBMS_SODA.create_collection`.

> **See Also:**
>
> - Overview of SODA Document Collections in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for general information about SODA document collections and their metadata
> - SODA Collection Metadata Components (Reference) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)* for reference information about collection metadata components

# 4.1 Getting the Metadata of an Existing Collection

You use `SODA_COLLECTION_T` method `get_metadata()` to get all of the metadata for a collection, as a JSON document.

> **See Also:**
>
> - GET_METADATA Function in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `get_metadata()`
> - JSON_QUERY in *Oracle Database SQL Language Reference* for information about SQL/JSON function `json_query`

**Example 4-1 Getting the Metadata of a Collection**

This example shows the result of invoking `SODA_COLLECTION_T` method `get_metadata()` on the collection with the default configuration that was created using Example 3-3. (It also uses SQL/JSON function `json_query`, with keyword `PRETTY`, to pretty-print the JSON data obtained.)

```
DECLARE
    collection  SODA_COLLECTION_T;
BEGIN
    collection := DBMS_SODA.open_collection('myCollectionName');
    IF collection IS NULL THEN
        DBMS_OUTPUT.put_line('Collection does not exist');
    ELSE
        DBMS_OUTPUT.put_line('Metadata: '
                             || json_query(collection.get_metadata, '$'
PRETTY));
    END IF;
END;
/
```

The default metadata for a collection is presented in Default Collection Metadata in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*.

# 4.2 Creating a Collection That Has Custom Metadata

To create a document collection that has custom metadata, you pass its metadata, as JSON data, to PL/SQL function `DBMS_SODA.create_collection`.

The optional second argument to PL/SQL function `DBMS_SODA.create_collection` is a SODA **collection specification**. It is JSON data that specifies the metadata for the new collection.

If a collection with the same name already exists then it is simply opened and its handle is returned. If the custom metadata provided does not match the metadata of

the existing collection then the collection is not opened and an error is raised. (To match, all metadata fields must have the same values.)

**See Also:**

- CREATE_COLLECTION Function in *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL function `DBMS_SODA.create_collection`
- SODA_COLLECTION_T Type in *Oracle Database PL/SQL Packages and Types Reference* for information about `SODA_COLLECTION_T` method `get_metadata()`
- JSON_QUERY in *Oracle Database SQL Language Reference* for information about SQL/JSON function `json_query`

**Example 4-2 Creating a Collection That Has Custom Metadata**

This example creates a collection with the default metadata, except that the key assignment method is set to `CLIENT`.

The example uses `SODA_COLLECTION_T` method `get_metadata()` to get the complete metadata from the newly created collection, which it passes to SQL/JSON function `json_query` to pretty-print (using keyword `PRETTY`).

```
DECLARE
    collection SODA_COLLECTION_T;
    metadata VARCHAR2(4000) :=
      '{"keyColumn" : {"name" : "ID", "assignmentMethod" : "CLIENT" },
        "contentColumn" : {"name" : "JSON_DOCUMENT"},
        "versionColumn" : {"name" : "VERSION"},
        "lastModifiedColumn" : {"name" : "LAST_MODIFIED"},
        "creationTimeColumn" : {"name" : "CREATED_ON"}}';
BEGIN
    collection := DBMS_SODA.create_collection('myCustomCollection',
                                              metadata);
    DBMS_OUTPUT.put_line('Collection specification: ' ||
                         json_query(collection.get_metadata, '$' PRETTY));
END;
/
```

This is the pretty-printed output. The values of any fields for `keyColumn` and `contentColumn` that are not specified in the collection specification, are defaulted. The values of fields other than those provided in the collection specification (that is, other than `keyColumn` and `contentColumn`) are also defaulted. The value of field `tableName` is defaulted from the collection name. The value of field `schemaName` is the database schema (user) that was current when the collection was created.

```
Collection specification: {
  "schemaName" : "mySchemaName",
  "tableName" : "myCustomCollection",
  "keyColumn" :
  {
```

```
    "name" : "ID",
    "sqlType" : "VARCHAR2",
    "maxLength" : 255,
    "assignmentMethod" : "CLIENT"
  },
  "contentColumn" :
  {
    "name" : "JSON_DOCUMENT",
    "sqlType" : "BLOB",
    "compress" : "NONE",
    "cache" : true,
    "encrypt" : "NONE",
    "validation" : "STANDARD"
  },
  "lastModifiedColumn" :
  {
    "name" : "LAST_MODIFIED"
  },
  "versionColumn" :
  {
    "name" : "VERSION",
    "method" : "UUID"
  },
  "creationTimeColumn" :
  {
    "name" : "CREATED_ON"
  },
  "readOnly" : false
}
```

**Related Topics**

- Creating a Document Collection with SODA for PL/SQL
  You can use PL/SQL function `DBMS_SODA.create_collection` to create a document collection with the default metadata.

# A
# Redefining a SODA Collection

You can use *online redefinition* to *change* the metadata or other properties of an existing collection. In particular, after upgrading so that database initialization parameter `compatible` is at least `20`, you can migrate a collection to reflect the new default metadata.

The default collection metadata for a database with `compatible` initialization parameter at least `20` has `"JSON"` as the value of metadata field `contentColumn.sqlType`. And it has `"UUID"` as the value of metadata field `versionColumn.method`. If your `compatible` setting is `20` or greater then Oracle recommends that you use online redefinition to change the metadata of an existing collection so that it uses these values.

Online redefinition for a SODA collection is similar to online redefinition for a database table. The PL/SQL procedures used (in package `DBMS_SODA`) for a collection are analogous to their counterparts for a table in package `DMBS_REDEFINITION`.

Starting with the collection to be redefined, you apply SODA online-redefinition procedures, one by one. At each step, you can use subprogram `DBMS_SODA.abort_redef_collection` to abort the migration process if an error is raised.

As an example, the steps presented here migrate collection `MyCollection` so that its metadata reflects that of the default metadata for a database with initialization parameter `compatible` `20` or greater.

The following code creates the initial collection to be migrated, which uses textual JSON data stored as `BLOB` content. The default metadata for a database with parameter `compatible` less than `20` is specified here explicitly, for illustration purposes. In particular, `contentColumn.sqlType` is `"BLOB"`, and `versionColumn.method` is `"SHA256"`.

```
v_original_collection := 'MyCollection';
v_original_metadata :=
  '{"keyColumn": {"name": "ID",
                  "sqlType": "VARCHAR2",
                  "maxLength": 255,
                  "assignmentMethod": "UUID"},
    "contentColumn": {"name": "JSON_DOCUMENT",
                      "sqlType": "BLOB",
                      "compress": "NONE",
                      "cache": true,
                      "encrypt": "NONE",
                      "validation": "STANDARD"},
    "versionColumn": {"name": "VERSION",
                      "method": "SHA256"},
    "lastModifiedColumn": {"name": "LAST_MODIFIED"},
    "creationTimeColumn": {"name": "CREATED_ON"},
    "readOnly": false}';

DBMS_SODA.create_collection(v_original_collection,
                            v_original_metadata);
```

The steps below change fields `contentColumn.sqlType` and `versionColumn.method`. The other metadata fields are left unchanged, except that fields that no longer apply have been removed: `compress`, `cache`, `encrypt`, and `validation`. (Those fields do not apply to document content stored as `JSON` data type.)

To perform online redefinition for a collection, you need the following database privileges:

- Privilege `EXECUTE` for PL/SQL package `DBMS_REDEFINITION`
- System privilege `CREATE MATERIALIZED VIEW`
- System privilege `CREATE TABLE` or, if the collection is backed by a table in a database schema different from the current one, `CREATE ANY TABLE`

> **See Also:**
>
> Summary of SODA Online Redefinition Subprograms in *Oracle Database PL/SQL Packages and Types Reference*

1. Use subprogram `can_redef_collection`, to check whether the collection is eligible for online redefinition. An error is raised if it is not eligible.

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
BEGIN
  v_original_collection := 'MyCollection';
  DBMS_SODA.can_redef_collection(v_original_collection);
END;
```

2. Use subprogram `create_interim_collection`, to create an interim collection to which data is copied while the original collection continues to handle production workload of SODA operations.

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
  v_interim_collection_name  NVARCHAR2(2000);
BEGIN
  v_original_collection := 'MyCollection';
  v_interim_collection := 'MyCollection_int';
  v_metadata := '{"contentColumn": {"sqlType": "JSON"},
                  "versionColumn": {"method": "UUID"}}}';

  DBMS_SODA.create_interim_collection(v_original_collection,
                                      v_interim_collection,
                                      v_metadata);
END;
```

Argument `v_metadata` specifies the metadata to change. You need not specify any metadata that remains unchanged.

The metadata for the interim collection (argument `v_metadata`) can include a `tableName` value that differs from that of the original collection, to specify the name of the table to which the interim collection is mapped.

If this table already exists then a mapped interim collection will be created on top of it. In this case, the table must not have any dependents (indexes, constraints, or triggers), or else an error is raised. Such dependents are instead taken (copied) from the original collection, in Step 4

3. Use subprogram `start_redef_collection`, to start the process of collection redefinition.

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
  v_interim_collection_name  NVARCHAR2(2000);
BEGIN
  v_original_collection := 'MyCollection';
  v_interim_collection := 'MyCollection_int';

  DBMS_SODA.start_redef_collection(v_original_collection,
                                   v_interim_collection);
END;
```

If your original collection has Virtual Private Database (VPD) policies then copy them to the interim collection *before* using `start_redef_collection`. And in that case use `start_redef_collection(v_original_collection, v_interim_collection, DBMS_REDEFINITION.cons_vpd_manual)`, to indicate that the VPD policies have been copied manually.

4. Use subprogram `copy_collection_dependents`, to copy everything that depends on the original collection to the interim collection. This includes all constraints and indexes (including indexes defined automatically by SODA).

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
  v_interim_collection_name  NVARCHAR2(2000);
  v_metadata                 VARCHAR2(2000);
  v_num_errors               NUMBER;
BEGIN
  v_original_collection := 'MyCollection';
  v_interim_collection := 'MyCollection_int';

  DBMS_SODA.copy_collection_dependents(v_original_collection,
                                       v_interim_collection,
                                       num_errors => v_num_errors);
END;
```

The value, `v_num_errors`, of output parameter `num_errors` indicates how many errors were raised.

Even if the collection to be modified has no user-defined dependents, such as indexes on JSON content, it necessarily has some internal SODA-defined dependents, which must be copied to the interim collection.

5. Use subprogram `sync_interim_collection`, to synchronize the data in the interim collection to that of the original collection, to minimize downtime during the last step (Step

7). Do this if a large number of DML operations are performed on the original collection while you are performing online redefinition with the interim collection.

Subprogram `sync_interim_collection` checks for all dependents required for a SODA collection. An error is raised if they are not all present.

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
  v_interim_collection_name  NVARCHAR2(2000);
BEGIN
  v_original_collection := 'MyCollection';
  v_interim_collection := 'MyCollection_int';

  DBMS_SODA.sync_interim_collection(v_original_collection,
                                    v_interim_collection);
END;
```

6. **Caution:**

   This step is important. The effect of Step 7 *cannot* be undone.

   Optional: Check that the interim collection works as expected. If it does *not*, use subprogram `DBMS_SODA.abort_redef_collection` to revert the changes, as follows:

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
  v_interim_collection_name  NVARCHAR2(2000);
BEGIN
  v_original_collection := 'MyCollectionName';
  v_interim_collection := 'MyCollectionName_int';

  DBMS_SODA.abort_redef_collection(v_original_collection,
                                   v_interim_collection);
END;
```

7. Use subprogram `finish_redef_table` to finish the redefinition process, swapping the names of the original collection and the interim collection.

   The effect of this step *cannot* be undone (unless it raises an error instead of committing).

   Both collections are *locked* for part of the duration of `finish_redef_table`. The interim collection is synchronized to the original collection during this step. This includes performing any DML that has taken place on the original collection since the last use of `sync_interim_collection` (or since `start_redef_collection`, if you have not used `sync_interim_collection`).

   The subprogram checks for all dependents required for a SODA collection. An error is raised if they are not all present.

```
DECLARE
  v_original_collection_name NVARCHAR2(2000);
  v_interim_collection_name  NVARCHAR2(2000);
```

```
BEGIN
  v_original_collection := 'MyCollection';
  v_interim_collection := 'MyCollection_int';

  DBMS_SODA.finish_redef_collection(v_original_collection,
                                    v_interim_collection;
END;
```

# Index


## A

acquire_lock() SODA_OPERATION_T method, *3-32*
as_of_scn() SODA_OPERATION_T method, *3-23*
as_of_timestamp() SODA_OPERATION_T method, *3-23*

## C

chaining together SODA_OPERATION_T methods, *3-21*
collection
  redefining, *A-1*
collection configuration, *4-1*
collection metadata
  custom, *4-1*, *4-2*
  getting, *4-2*
collections
  checking existence, *3-8*
  creating, *3-6*
    with custom metadata, *4-2*
  discovering, *3-9*
  dropping, *3-10*
  heterogeneous, full-text searching, *3-23*
  opening, *3-8*
    during creation, *3-6*
  truncating, *3-38*
committing a transaction, *3-46*
components of SODA documents, *3-11*
count() SODA_OPERATION_T method, *3-23*
create_collection function
  transaction handling, *3-46*
create_index() SODA_COLLECTION_T method, *3-38*
creating collections, *3-6*
  with custom metadata, *4-2*
creating documents, *3-11*

## D

data guide
  creating relational view from, *3-45*
  getting for a collection, *3-42*
database role SODA_APP, *3-3*
DBMS_SODA package subprograms
  create_collection
    example, *3-6*
    transaction handling, *3-46*
  drop_collection
    example, *3-10*
    transaction handling, *3-46*
  list_collection_names
    example, *3-9*
  open_collection
    example, *3-8*
DBMS_SODA.DOC_BLOB constant, *3-11*
DBMS_SODA.DOC_CLOB constant, *3-11*
DBMS_SODA.DOC_VARCHAR2 constant, *3-11*
deleting collections
  *See* dropping collections
deleting documents from collections
  *See* removing documents from collections
discovering collections
  checking existence, *3-8*
  listing, *3-9*
documents
  components, *3-11*
  creating, *3-11*
  finding in collections, *3-23*
  inserting into collections, *3-17*
  metadata, *3-11*
  removing from collections, *3-35*
  replacing in collections, *3-32*
drop_collection function
  example, *3-10*
  transaction handling, *3-46*
drop_index() SODA_COLLECTION_T method, *3-38*
dropping collections, *3-10*

## E

emptying a collection, *3-38*
existing collection, checking for, *3-8*

## F

filter() SODA_OPERATION_T method, *3-23*

find() SODA_COLLECTION_T method, 3-23
finding documents in collections, 3-23
flashback querying of collection data, 3-23
full-text search of non-JSON documents, 3-23

## G

get_blob() SODA_DOCUMENT_T method, 3-11
get_clob() SODA_DOCUMENT_T method, 3-11
get_created_on() SODA_DOCUMENT_T method, 3-11
get_cursor() SODA_OPERATION_T method, 3-23
get_data_guide() SODA_COLLECTION_T method, 3-42
get_data_guide() SODA_OPERATION_T method, 3-42
get_data_type() SODA_DOCUMENT_T method, 3-11
get_index() SODA_COLLECTION_T method, 3-38
get_key() SODA_DOCUMENT_T method, 3-11
get_last_modified() SODA_DOCUMENT_T method, 3-11
get_media_type() SODA_DOCUMENT_T method, 3-11
get_metadata() SODA_COLLECTION_T method, 4-2
get_one() SODA_OPERATION_T method, 3-23
get_varchar2() SODA_DOCUMENT_T method, 3-11
get_version() SODA_DOCUMENT_T method, 3-11
getter methods, document, 3-11
getting collection metadata, 4-2
getting document components, 3-11

## H

handling transactions, 3-46
heterogeneous collection, full-text search, 3-23
hint
  SQL monitoring, 3-17, 3-19, 3-23
hint() SODA_OPERATION_T method, 3-23

## I

indexes, getting, 3-38
indexing a collection, 3-38
insert_one_and_get() SODA_COLLECTION_T method, 3-17
insert_one() SODA_COLLECTION_T method, 3-17
inserting documents into collections, 3-17

## J

JSON data guide
  creating relational view from, 3-45
  getting for a collection, 3-42

## L

list_collection_names function
  example, 3-9
list_indexes() SODA_COLLECTION_T method, 3-38
listing collections, 3-9
locking documents
  pessimistic, 3-32

## M

metadata of collections
  getting, 4-2
metadata of documents
  getting, 3-11
metadata, custom, 4-1, 4-2
MONITOR SQL hint, 3-17, 3-19, 3-23

## N

nonterminal SODA methods, definition, 3-21

## O

open_collection function
  example, 3-8
opening collections, 3-8
  during creation, 3-6

## P

prerequisites for using SODA for PL/SQL, 1-1

## R

read and write operations, 3-21
redefining a collection, A-1
redefining collection metadata, A-1
relational view, created from collection data guide, 3-45
remove() SODA_OPERATION_T method, 3-35
removing all documents from a collection, 3-38
removing documents from collections, 3-35
replace_one_and_get() SODA_OPERATION_T method, 3-32
replace_one() SODA_OPERATION_T method, 3-32

replacing documents in collections, *3-32*
role SODA_APP, *3-3*
rolling back a transaction, *3-46*

## S

sample() SODA_OPERATION_T method, *3-42*
SODA_APP database role, *3-3*
SODA_COLLECTION_T methods
  create_index(), *3-38*
  drop_index(), *3-38*
  find(), *3-23*
  get_data_guide(), *3-42*
  get_index(), *3-38*
  get_metadata(), *4-2*
  insert_one_and_get(), *3-17*
  insert_one(), *3-17*
  list_indexes(), *3-38*
SODA_DOCUMENT_T methods
  get_blob(), *3-11*
  get_clob(), *3-11*
  get_created_on(), *3-11*
  get_data_type(), *3-11*
  get_key(), *3-11*
  get_last_modified(), *3-11*
  get_media_type(), *3-11*
  get_varchar2(), *3-11*
  get_version(), *3-11*
SODA_DOCUMENT_T object type and
    constructors, *3-11*
SODA_OPERATION_T methods, *3-21*
  acquire_lock(), *3-32*
  as_of_scn(), *3-23*
  as_of_timestamp(), *3-23*
  count(), *3-23*
  filter(), *3-23*
  get_cursor(), *3-23*
  get_data_guide(), *3-42*
  get_one(), *3-23*
  hint(), *3-23*
  remove(), *3-35*
  replace_one_and_get(), *3-32*
  replace_one(), *3-32*
  sample(), *3-42*

## T

terminal SODA methods, definition, *3-21*
transaction handling, *3-46*
truncating a collection, *3-38*

## V

view, created from collection data guide, *3-45*

## W

write and read operations, *3-21*