## Contents

### Preface
- Audience vi
- Documentation Accessibility vi
- Related Resources vi
- Conventions vi

### 1 Overview of Oracle Database API for MongoDB
1.1 Purpose of Oracle Database API for MongoDB 1-1
1.2 Tools and Drivers for Oracle Database API for MongoDB 1-2
1.3 Terms and Concepts: MongoDB and Oracle Database 1-2

### 2 Develop Applications with Oracle Database API for MongoDB
2.1 Indexing and Performance Tuning 2-1
2.2 Users, Authentication, and Authorization 2-6
2.3 Migrate Application Data from MongoDB to Oracle Database 2-8
2.4 Replace MongoDB Aggregation Pipelines 2-11
2.5 MongoDB Documents and Oracle Database 2-14
2.6 Other Differences Between MongoDB and Oracle Database 2-17

### 3 Support for MongoDB Operations — Reference
3.1 Collection and Index Definition Commands (DDL) 3-1
3.2 Document (Collection Content) Commands (DML) 3-2
3.3 Data Information Commands 3-4
3.4 Administrative Commands 3-5
3.5 Authentication and Authorization Commands 3-5
3.6 Session Commands 3-6

### Index
<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Indexing a Singleton Scalar Field Using the JSON Page of Database Actions</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>Indexing a Singleton Scalar Field Using SODA</td>
<td>2-4</td>
</tr>
<tr>
<td>2-3</td>
<td>Indexing a Singleton Scalar Field Using SQL</td>
<td>2-4</td>
</tr>
<tr>
<td>2-4</td>
<td>Creating a Materialized View And an Index For Fields Within Elements of an Array</td>
<td>2-5</td>
</tr>
<tr>
<td>2-5</td>
<td>Migrate JSON Data to Oracle Database Using mongoexport and mongoimport</td>
<td>2-9</td>
</tr>
<tr>
<td>2-6</td>
<td>Loading JSON Data Into a Collection Using DBMS_CLOUD.COPY_COLLECTION</td>
<td>2-10</td>
</tr>
<tr>
<td>2-7</td>
<td>Replacing MongoDB Aggregation Pipeline Code with SQL Code</td>
<td>2-13</td>
</tr>
<tr>
<td>Table</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1-1</td>
<td>Application-User Terms</td>
<td>1-3</td>
</tr>
<tr>
<td>2-1</td>
<td>Conversion of BSON Field <code>_id</code> Value To Column ID VARCHAR2 Value</td>
<td>2-16</td>
</tr>
<tr>
<td>2-2</td>
<td>JSON Scalar Type Conversions: BSON to OSON Format</td>
<td>2-17</td>
</tr>
</tbody>
</table>
Preface

This document provides a conceptual overview of Oracle Database API for MongoDB.

- Audience
- Documentation Accessibility
- Related Resources
- Conventions

Audience

This document is intended for users of Oracle Database API for MongoDB.

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.

Related Resources

For more information, see these Oracle resources:

- Oracle Database API for MongoDB at Oracle Help Center for complete information about this product
- Autonomous JSON Database
- Oracle Database JSON Developer’s Guide
- Oracle as a Document Store for general information about using JSON data in Oracle Database, including with Simple Oracle Document Access (SODA) and Oracle Database API for MongoDB

Conventions

The following text conventions are used in this document:
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Overview of Oracle Database API for MongoDB

Oracle Database API for MongoDB lets applications interact with collections of JSON documents in Oracle Database using MongoDB commands.

Oracle Database API for MongoDB is available on these Autonomous Oracle Databases with shared infrastructure: Autonomous JSON Database and Autonomous Transaction Processing. It is not available on Autonomous Data Warehouse or Autonomous Database with dedicated infrastructure.

- **Purpose of Oracle Database API for MongoDB**
  Oracle Database understands *Mongo-speak*. That's the purpose of Oracle Database API for MongoDB.

- **Tools and Drivers for Oracle Database API for MongoDB**
  Oracle Database API for MongoDB supports a variety of MongoDB tools and drivers.

- **Terms and Concepts: MongoDB and Oracle Database**
  Some application-user terms and concepts used by MongoDB are presented, together with description of their relation to Oracle Database..

---

**See Also:**

Using the Oracle Database API for MongoDB in *Using Oracle Autonomous Database on Shared Exadata Infrastructure* for information about using an Autonomous Database (including an Autonomous JSON Database) with Oracle Database API for MongoDB. This covers configuring the database for use with the API, including for security and connection.

---

1.1 Purpose of Oracle Database API for MongoDB

Oracle Database understands *Mongo-speak*. That's the purpose of Oracle Database API for MongoDB.

You have one or more applications that interact with a MongoDB NoSQL database. You’re used to using MongoDB commands, particularly for the business logic of your applications (query by example — QBE) but also for data definition (creating collections and indexes), data manipulation (CRUD operations), and some database administration (status information). You expect and depend on the flexibility of a JSON document store: no fixed data schemas, easy to use document-centric APIs.

On the other hand, you’re looking to future-proof your applications and make them more robust. You want advanced security; fully ACID transactions (atomicity, consistency, isolation, durability); standardized, straightforward JOINs with all sorts of data; and state-of-the-art analytics, machine-learning, and reporting capabilities — all that and more, out of the box.
Oracle Database API for MongoDB, or **Mongo API** for short, provides all of that. It translates the MongoDB wire protocol into SQL statements that are executed by Oracle Database. This means you can continue to use the drivers, frameworks, and tools you’re used to, to develop your JSON document-store applications.

Oracle Database is a *converged* database, which opens a whole new world to your applications. It’s multi-model and polyglot — seemingly different kinds of databases rolled into one, providing synergy across very different features, supporting different workloads and data models.

Oracle Database is also *multitenant*, which means you can have both consolidation and isolation, for different teams and purposes. And it provides a single, common approach for security, upgrades, patching, and maintenance. But if you use an Autonomous Oracle Database, such as Autonomous JSON Database, then Oracle takes care of all such database administration responsibilities. And there’s Always Free access to an autonomous database.

The standard, declarative language SQL (Structured Query Language) underlies processing on Oracle Database. You might develop your application using MongoDB-speak or Simple Oracle Document Access (SODA) with a popular application development language, but SQL is behind it all, and it enables your app to play well with everything else on Oracle Database.

### 1.2 Tools and Drivers for Oracle Database API for MongoDB

Oracle Database API for MongoDB supports a variety of MongoDB tools and drivers. Oracle recommends that you use the following tool and driver versions, or higher, with support for load-balanced connections.

- C 1.19.0
- C# 2.13.0
- Compass 1.28.1
- Database Tools 100.5.0 (includes `mongoexport`, `mongorestore`, and `mongodump`)
- Go 1.6.0
- Java 4.3.0
- MongoSH 0.15.6
- Node.js driver 4.1.0
- PyMongo 3.12.0 (for Python language)
- Ruby 2.16.0
- Rust 2.1.0

You can download these drivers from [https://www.mongodb.com/docs/drivers/](https://www.mongodb.com/docs/drivers/).

### 1.3 Terms and Concepts: MongoDB and Oracle Database

Some application-user terms and concepts used by MongoDB are presented, together with description of their relation to Oracle Database.
Some of the same terms are also used in Oracle Database API for MongoDB. In general, application developers need not be concerned with the Oracle Database concepts and technologies that underlie such terms.

### Table 1-1 Application-User Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
</table>
| Database    | A set of collections.  
On Oracle Database this corresponds to a database schema.  
Because of this possible confusion over use of the word database, in this documentation that word is used for Oracle Database, and the term **schema**, or **database schema**, is used for what MongoDB calls a "database". |
| User        | For log-in purposes, a **user** of Oracle Database API for MongoDB is an Oracle Database user, which is also called a database schema (see previous).  
To use the collections in a given schema ("database") , you log in with the Oracle Database API for MongoDB using the MongoDB PLAIN $external mechanism and providing the credentials for that schema.  
A **root user**, that is, a user who has MongoDB role root, can create additional database schemas. And a root user can use the collections of any schema without needing to log in separately for that schema. See command `saslStart` in Authentication and Authorization Commands. |
| Collection  | A collection contains a set of documents.  
A collection name is unique for a given database schema: Different collections can have the same name if they are in different schemas.  
On Oracle Database, a table or a view underlies a collection. The table name is derived from the collection name and is typically the same. (Exceptions include collection names that use words reserved by Oracle Database.) Typically all documents in a collection are JSON documents. |
| Document    | The basic unit of storage for data in a collection.  
On Oracle Database a document corresponds roughly to a row in the table or view that underlies the collection.  
A document is typically a JSON document, that is, it contains only JSON data. On Oracle Autonomous Database a document is always a JSON document.  
On Oracle Autonomous Database the table **column** used to store documents is named **data**. |
| Primary Key | On Oracle Database a primary key is used to uniquely identify a table or view row.  
MongoDB uses a unique `_id` field in a document to identify the document. On Oracle Database the primary key for a JSON document is stored in a column named `id`. Its value is automatically set to the value of the document's `_id` field. See Document Key: Differences and Conversion. |
| QBE         | A **Query By Example** JSON object, which is sent by an application client to the server (Oracle Database), to act on documents of a collection.  
The QBE object can contain **QBE operator** fields, whose names start with `. The operators are interpreted, and their operations are invoked to act on the collection. The server returns the action results to the client.  
QBEs are typically used to query a collection, but they can also be used to project or update data in documents.  
Oracle Database API for MongoDB translates QBEs into SQL (Structured Query Language) queries. |
Table 1-1  (Cont.) Application-User Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Indexes enhance performance when acting on collections (querying, inserting, updating, and deleting documents). An index name is unique for a given database schema: Different indexes can have the same name if they are in different schemas. <strong>Note</strong>: MongoDB commands to create or drop indexes are ignored by Oracle Database API for MongoDB. You must instead create Oracle Database indexes that are relevant for your JSON data.</td>
</tr>
<tr>
<td>Pipeline</td>
<td>MongoDB aggregation operations chain multiple operations together, invoking them sequentially as a pipeline. Aggregation operations are carried out differently using Oracle Database API for MongoDB — pipelines are not used. See Replace MongoDB Aggregation Pipelines.</td>
</tr>
</tbody>
</table>

Related Topics

- **MongoDB Documents and Oracle Database**
  Presented here is the relationship between a JSON document used by MongoDB and the same content as a JSON document used by Oracle Database.

- **Migrate Application Data from MongoDB to Oracle Database**
  Some ways to export your JSON data from MongoDB and then import it into Oracle Database are described. Migration considerations are presented.

See Also:

- **Overview of SODA Document Collections in Oracle Database Introduction to Simple Oracle Document Access (SODA)** for information about collections
- **Overview of SODA Documents in Oracle Database Introduction to Simple Oracle Document Access (SODA)** for information about documents
- **Overview of SODA Filter Specifications (QBEs) in Oracle Database Introduction to Simple Oracle Document Access (SODA)** for information about QBEs
- **Query JSON Data in Oracle Database JSON Developer’s Guide** for information about querying JSON data using SQL
2

Develop Applications with Oracle Database API for MongoDB

Considerations when developing or migrating applications — a combination of (1) how-to information and (2) descriptions of differences and possible adjustments.

**Indexing and Performance Tuning**
Oracle Database offers multiple technologies to accelerate queries over JSON data, including indexes, materialized views, in-memory column storage, and Exadata storage-cell pushdown. Which performance-tuning approaches you take depend on the needs of your application.

**Users, Authentication, and Authorization**
Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

**Migrate Application Data from MongoDB to Oracle Database**
Some ways to export your JSON data from MongoDB and then import it into Oracle Database are described. Migration considerations are presented.

**Replace MongoDB Aggregation Pipelines**
Oracle Database does not use MongoDB aggregation pipelines. Replace any such pipelines for your application with direct use of SQL. The Oracle Database optimizer executes queries in an optimal order.

**MongoDB Documents and Oracle Database**
Presented here is the relationship between a JSON document used by MongoDB and the same content as a JSON document used by Oracle Database.

**Other Differences Between MongoDB and Oracle Database**
Various differences between MongoDB and Oracle Database are described. These differences are generally not covered in other topics. Consider these differences when you migrate an application to Oracle Database or you develop a new application for Oracle Database that uses MongoDB commands.

### 2.1 Indexing and Performance Tuning

Oracle Database offers multiple technologies to accelerate queries over JSON data, including indexes, materialized views, in-memory column storage, and Exadata storage-cell pushdown. Which performance-tuning approaches you take depend on the needs of your application.

MongoDB index operations, such as `createIndex` and `dropIndex`, are not usable with Oracle Database; they are ignored. And you typically do not need to create indexes equivalent to those you might use for MongoDB. If needed, you can instead create relevant indexes for Oracle Database, using the JSON Page of *Using Oracle Database Actions* (see Creating Indexes for JSON Collections), Simple Oracle Document Access (SODA), or SQL (`CREATE INDEX`). Using the JSON page is perhaps the easiest approach to indexing JSON data.
MongoDB allows different collections in the same "database" to have indexes of the same name. This is not allowed in Oracle Database — the name of an index must be unique across all collections of a given database schema ("database").

Consider, for example, indexing a collection of purchase-order documents such as this one:

```json
{
  "PONumber" : 1600,
  "User" : "ABULL",
  "LineItems" : [{
    "Part" : {
      "Description" : "One Magic Christmas",
      "UnitPrice" : 19.95,
      "UPCCode" : 13131092899
    },
    "Quantity" : 9.0
  },
  { "Part" : {
    "Description" : "Lethal Weapon",
    "UnitPrice" : 19.95,
    "UPCCode" : 85391628927
  },
  "Quantity" : 5.0
}]
```

Two important use cases are (1) indexing a singleton scalar field, that is, a field that occurs only once in a document (2) indexing a scalar field in objects within the elements of an array. Indexing the value of field \texttt{PONumber} is an example of the first case. Indexing the value of field \texttt{UPCCode} is an example of the second case.

Example 2-1, Example 2-2, and Example 2-3 illustrate the first case. Example 2-4 illustrates the second case.

You can also index GeoJSON (spatial) data, using a function-based SQL index that returns \texttt{SDO\_GEOMETRY} data. And for all JSON data you can create a JSON search index, and then perform full-text queries using SQL/JSON condition \texttt{json\_textcontains}.

**Example 2-1 Indexing a Singleton Scalar Field Using the JSON Page of Database Actions**

To create an index for field \texttt{PONumber} using the JSON Page, do the following.

1. Right-click the collection name (here, \texttt{collection\_orders}) and select \texttt{Indexes} from the popup menu.
2. In the **New Index** page:

   - Type `*` in the **Properties** search box.

     This populates the **Properties** list with paths to all scalar fields in your collection. These paths are provided by sampling the collection data using a JSON data guide — see JSON_DATAGUIDE in *Oracle Database SQL Language Reference*.

     If you turn on option **Advanced**, by pushing its slider to the right, then the types of the listed scalar fields are also shown. The types shown are those picked up by sampling the collection. But you can change the type of a field for indexing purposes.

   - Select the paths of the fields to be indexed. In this case we want only a single scalar field indexed, **PONumber**, so select that.

     **Note:** This dialog box lets you select multiple paths. If you select more than one path then a composite index is created for the data at those paths. But if you want to index two different fields separately then create two indexes, not one composite index (which indexes both fields together).

     The index data type is determined automatically by the types of the data at the selected paths, but you can control this by turning on **Automatic** and changing the data types. For example, JSON numbers in the collection data for a given field cause a type of **number** to be listed, but you can edit this to **VARCHAR2** to force indexing as a string value.

     The values of field **PONumber** are unique — the same numeric value is not used for the field more than once in the collection, so select **Unique** index.

     Select **Index Nulls** also. This is needed for queries that use **ORDER BY** to sort the results. It causes every document to have an entry in the index.

     The values in field **PONumber** are JSON numbers, which means the index can be used for numerical comparison.

---

1. MongoDB calls a composite index a compound index. A composite index is also sometimes called a concatenated index.
Example 2-2   Indexing a Singleton Scalar Field Using SODA

Each SODA implementation (programming language or framework) that supports indexing provides a way to create an index. They all use a SODA index specification to define the index to be created. For example, with SODA for REST you use an HTTP POST request, passing URI argument action=index, and providing the index specification in the POST body.

This is a SODA index specification for the unique index on field PONumber:

```
{ "name" : "poNumIdx",
 "unique" : true,
 "fields" : [ { "path" : "PONumber",
               "dataType" : "NUMBER",
               "order"    : "ASC" } ] }
```

Example 2-3   Indexing a Singleton Scalar Field Using SQL

You can use Database Actions to create an index for field PONumber in column data of table orders with this SQL code. This uses SQL/JSON function json_value to extract values of field PONumber.

Item method number() is used in the path expression that identifies the field to index, to convert the field value to a JSON number value. For example, a PONumber string value of "42" is converted to the number 42.

The code uses ERROR ON ERROR handling, to raise an error if a document no PONumber field or it has more than one.

```
CREATE UNIQUE INDEX idx1 ON orders
  (json_value(data, '%$PONumber.number()') ERROR ON ERROR)
```
Example 2-4  Creating a Materialized View And an Index For Fields Within Elements of an Array

This example indexes field `UPCCode`, which can occur multiple times in a document because it is contained in objects within an array (objects as elements or at lower levels within elements).

Starting with Oracle Database 21c you can create a multivalue index for such fields. Prior to Oracle Database 21c you can instead create a materialized view that extracts the data you want to index, and create a function-based index on that view data.

This SQL code does that. It creates materialized view `mv_UPCCode` with column `upccode`, which is a projection of field `UPCCode` from within array `LineItems` of column `data` of table `orders`. It then creates index `mv_UPCCode_idx` on column `upccode` of the materialized view (`mv_UPCCode`).

```sql
CREATE MATERIALIZED VIEW mv_UPCCode
    BUILD IMMEDIATE
    REFRESH FAST ON STATEMENT WITH PRIMARY KEY
AS SELECT o.id, jt.upccode
    FROM orders o,
         json_table(data, '$.LineItems[*]' ERROR ON ERROR NULL ON EMPTY
            COLUMNS (upccode NUMBER PATH '$.UPCCode')) jt;

CREATE INDEX mv_UPCCode_idx ON mv_UPCCode(upccode);
```

The query optimizer is responsible for finding the most efficient method for a SQL statement to access requested data. In particular, it determines whether to use an index that applies to the queried data, and which index to use if more than one is relevant. In most cases the best guideline is to rely on the optimizer.

In some cases, however, you might prefer to specify that a particular index be picked up for a given query. You can do this with a MongoDB `hint` that names the index. (Oracle does not support the use of MongoDB index specifications — just provide the index name.) For example, this query uses index `myindex`:

```javascript
db.fruit.find({"name":"apple"}).hint("myindex")
```

Alternatively, you can specify an index to use by passing an Oracle SQL hint, using query-by-example (QBE) operator `$native`, which is an Oracle extension to the MongoDB hint syntax.

The argument for `$native` has the same syntax as a SQL hint string (that is, the actual hint text, without the enclosing SQL comment syntax `/*+...*/`). You can pass any SQL `hint` using `$native`. In particular, you can turn on `monitoring` for the current SQL statement using `hint MONITOR`. This code does that for a `find()` query:

```javascript
db.col.find().hint({"$native":"MONITOR"})
```
2.2 Users, Authentication, and Authorization

Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

By default, MongoDB does not enable user authentication and authorization checks. Oracle Database always requires authentication, and it always verifies that a connected user is authorized to perform a requested operation. A valid username and password must be provided for authentication.

Oracle Database API for MongoDB supports only the following connection-option values for authentication:

- **PLAIN** value (plain-text authentication) for option authMechanism. In particular, the SCRAM-SHA-* authentication methods are not supported.
- **$external** value for option authSource. (This is anyway required for MongoDB whenever the authentication method is PLAIN.)
Oracle Database API for MongoDB relies on Oracle Database users, privileges, and roles. You cannot add or modify these users and roles using MongoDB clients or drivers. You can instead do this using SQL or the Oracle Autonomous Database console. The minimum Oracle Database roles required to use the API are CONNECT, RESOURCE, and SODA_APP.

For MongoDB, a "database" is a set of collections. For Oracle Database API for MongoDB, this corresponds to an Oracle Database schema.

**Note:**

Using Oracle API for MongoDB to drop a "database" does not drop the underlying database schema. Instead, it drops all collections within the schema. An administrative user can drop a schema using SQL (for example, using Database Actions with an Autonomous Oracle Database).

For the API, a username must be a database schema name. The name is case-insensitive, it cannot start with a nonalphabetic character (including a numeral), and it must be provided with a secure password.

Normally, a user of the API can only perform operations within its schema (the username is the schema name). Examples of such operations include creating new collections, reading and writing documents, and creating indexes.

When an administrative user tries to insert data into a database schema (user) that does not exist, that schema is created automatically as a schema-only account, which means that it does not have a password and it cannot be logged into. The new schema is granted these privileges: SODA_APP, CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, CREATE PROCEDURE, and CREATE JOB. The schema is also given an unlimited tablespace quota, and is enabled for using Oracle REST Data Services (ORDS).

For an ordinary user of the API, a MongoDB shell command (such as `use otherDB`) that switches the current MongoDB database to another one is typically not supported — switching to another database schema raises an error.

However, an administrative user, which is one that has all of the following privileges, can create new users (database schemas), and can access any schema as any user: CREATE USER, ALTER USER, DROP USER. User admin is a predefined administrative user.

An administrative user can do the following:

- Use the schemas of other users.
  
  Access to other schemas than that of the current user makes use of a proxied connection. For example, someone connected as an administrative user can perform operations in schema other_user using the same roles and privileges as if connected directly as other_user.

- Create new users (schemas).
  
  For example, if an administrative user tries to create a collection in a schema toto that does not already exist, that schema (user) is automatically created.

Oracle recommends that you do not allow production applications to make use of an administrative user. Applications should instead connect as ordinary users, with a minimum of
privileges. In particular, connect an application to the database using a MongoClient that is specific to a particular schema (user).

Related Topics

- **Terms and Concepts: MongoDB and Oracle Database**
  Some application-user terms and concepts used by MongoDB are presented, together with description of their relation to Oracle Database.

- **Migrate Application Data from MongoDB to Oracle Database**
  Some ways to export your JSON data from MongoDB and then import it into Oracle Database are described. Migration considerations are presented.

Related Topics

- **MongoDB Documents and Oracle Database**
  Presented here is the relationship between a JSON document used by MongoDB and the same content as a JSON document used by Oracle Database.

- **Users, Authentication, and Authorization**
  Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

See Also:

- Create Users on Autonomous Database in *Using Oracle Autonomous Database on Shared Exadata Infrastructure*
- Manage User Roles and Privileges on Autonomous Database in *Using Oracle Autonomous Database on Shared Exadata Infrastructure*
- CREATE USER in *Oracle Database SQL Language Reference* for information about using SQL to create database schemas (also called database users)
- GRANT in *Oracle Database SQL Language Reference* for information about using SQL to grant roles to database schemas
- Using the Oracle Database API for MongoDB in *Using Oracle Autonomous Database on Shared Exadata Infrastructure* for information about using an Autonomous Database (including an Autonomous JSON Database) with Oracle Database API for MongoDB. This covers configuring the database for use with the API, including for security and connection.

### 2.3 Migrate Application Data from MongoDB to Oracle Database

Some ways to export your JSON data from MongoDB and then import it into Oracle Database are described. Migration considerations are presented.

You can migrate your application data in any of these ways:

- Use the MongoDB command-line tools `mongoexport` and `mongoimport`.
mongoexport exports data from a MongoDB instance to your file system, and mongoimport imports the exported data from your file system to Oracle Database. Provide your database connection information when using mongoimport. Example 2-5 illustrates this.

- Use a MongoDB tool such as Compass to import data into Oracle Database after connecting that tool to the database. Select the name of your JSON collection, then select ADD DATA.

This displays a popup dialog box where you browse to and import the JSON file containing your collection data. See MongoDB Compass.

- After exporting JSON data to your file system, import it to the Oracle Cloud Object Store, then load it from there into a collection using PL/SQL procedure DBMS_CLOUD.copy_collection. Example 2-6 illustrates this.

This processes the data in parallel, so it is typically faster than mongoimport.

- Write a program that reads JSON documents from a connection to MongoDB and writes them to a connection to Oracle Database.

**Example 2-5  Migrate JSON Data to Oracle Database Using mongoexport and mongoimport**

This example exports collection sales from MongoDB to file-system file sales.json. It then imports the data from that file to Oracle Database as collection sales. The user is connected to host <host> as database schema <user> with password <password>.

```bash
mongoexport --collection=sales --out sales.json

mongoimport 'mongodb://<user>:<password>@<host>:27017/<user>?
authMechanism=PLAIN&authSource=$external&ssl=true' --collection=sales --file=sales.json
```
Note:

Use URI percent-encoding to replace any reserved characters in your connection-string URI — in particular, characters in your username and password. These are the reserved characters and their percent encodings:

<table>
<thead>
<tr>
<th>Character</th>
<th>% Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>%21</td>
</tr>
<tr>
<td>#</td>
<td>%23</td>
</tr>
<tr>
<td>$</td>
<td>%24</td>
</tr>
<tr>
<td>%</td>
<td>%25</td>
</tr>
<tr>
<td>&amp;</td>
<td>%26</td>
</tr>
<tr>
<td>'</td>
<td>%27</td>
</tr>
<tr>
<td>(</td>
<td>%28</td>
</tr>
<tr>
<td>)</td>
<td>%29</td>
</tr>
<tr>
<td>*</td>
<td>%2A</td>
</tr>
<tr>
<td>+</td>
<td>%2B</td>
</tr>
<tr>
<td>,</td>
<td>%2C</td>
</tr>
<tr>
<td>/</td>
<td>%2F</td>
</tr>
<tr>
<td>:</td>
<td>%3A</td>
</tr>
<tr>
<td>;</td>
<td>%3B</td>
</tr>
<tr>
<td>=</td>
<td>%3D</td>
</tr>
<tr>
<td>?</td>
<td>%3F</td>
</tr>
<tr>
<td>@</td>
<td>%40</td>
</tr>
<tr>
<td>[</td>
<td>%5B</td>
</tr>
<tr>
<td>]</td>
<td>%5D</td>
</tr>
</tbody>
</table>

For example, if your username is `RUTH` and your password is `@least1/2#?` then your MongoDB connection string to server `<server>` might look like this:

'mongodb://RUTH:%40@least1%2F%23%3F@<server>:27017/ruth/ ...'

Depending on the tools or drivers you use, you might be able to provide a username and password as separate parameters, instead of as part of a URI connection string. In that case you likely won’t need to encode any reserved characters they contain.

See also:

- Percent Encoding - Reserved Characters
- Uniform Resource Identifier (URI): Generic Syntax

See Also:

Using the Oracle Database API for MongoDB in Using Oracle Autonomous Database on Shared Exadata Infrastructure for information about using an Autonomous Database (including an Autonomous JSON Database) with Oracle Database API for MongoDB. This covers configuring the database for use with the API, including for security and connection.

Example 2-6  Loading JSON Data Into a Collection Using DBMS_CLOUD.COPY_COLLECTION

This example loads data from the Oracle Cloud Object Store into a new collection, `newCollection`, using PL/SQL procedure `DBMS_CLOUD.copy_collection`. It assumes that the data was exported from MongoDB to your file system and then imported from there to the object-store location that’s passed as the value of parameter `file_uri_list`.

The value passed as `copy_collection` parameter `FORMAT` is a JSON object with fields `recorddelimiter` and `type`:
Field $recorddelimiter$ specifies that records in the input data are separated by newline characters. A JSON document is created for each record, that is, for each line in the newline-delimited input data.

Field $type$ specifies that the input JSON data can contain EJSON extended objects, and that these should be interpreted.

See DBMS_CLOUD Package Format Options in Using Oracle Autonomous Database on Shared Exadata Infrastructure for information about parameter $FORMAT$.

```sql
BEGIN
  DBMS_CLOUD.copy_collection(
    collection_name => 'newCollection',
    file_uri_list => 'https://objectstorage.../data.json',
    format => json_object(
      'recorddelimiter' : '***\n***',
      'type'            : 'ejson')
  );
END;
/```

Related Topics

- **Users, Authentication, and Authorization**
  Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

- **Terms and Concepts: MongoDB and Oracle Database**
  Some application-user terms and concepts used by MongoDB are presented, together with description of their relation to Oracle Database.

See Also:

- `mongoexport` and `mongoimport`
- Load an Array of JSON Documents into a Collection in Using Oracle Autonomous JSON Database for information about using PL/SQL procedure `DBMS_CLOUD.COPY_COLLECTION`

### 2.4 Replace MongoDB Aggregation Pipelines

Oracle Database does not use MongoDB aggregation pipelines. Replace any such pipelines for your application with direct use of SQL. The Oracle Database optimizer executes queries in an optimal order.

MongoDB's aggregation pipeline is essentially a weak emulation of SQL capabilities. With MongoDB you express operations such as sort and group as separate steps in a procedural pipeline — you specify them as executing in sequence.

Oracle Database does not, in general, execute aggregation operators in a sequential manner, and you need not specify the execution order. Instead, the database relies on declarative SQL code and an optimizer. The optimizer picks an optimal execution plan based on available indexes, data statistics, cost estimates, and so on.
Unlike MongoDB, Oracle Database does not limit the size of the data to be sorted, joined or grouped. You can use it for reporting or analytical work that spans millions of documents across any number of collections.

As a user of Oracle Database API for MongoDB, you can apply SQL directly to JSON data in the database without worrying about manually sequencing any operations you perform on it.

You can use Oracle Database simplified dot notation for JSON data, or standard SQL/JSON functions `json_value`, `json_query`, and `json_table`, to extract values from your JSON data for reporting or analytics. You can easily join JSON data from different collections, or join JSON data with non-JSON data stored in the database. And you can encapsulate queries as database views or materialized views.

You can generally replace a MongoDB aggregation-pipeline expression with equivalent SQL code. (In fact, most MongoDB commands use the same or similar names as an equivalent SQL operator.) Example 2-7 illustrates this.

These are correspondences between MongoDB aggregation operators and Oracle SQL constructs:

- **$avg** — Use aggregate function `avg`. See AVG in Oracle Database SQL Language Reference.in Oracle Database SQL Language Reference.
- **$count** — Use aggregate function `count`. See COUNT in Oracle Database SQL Language Reference in Oracle Database SQL Language Reference.
- **$first** — Use SQL `SELECT` with `FETCH FIRST`. See SELECT - row_limiting_clause in Oracle Database SQL Language Reference in Oracle Database SQL Language Reference.
- **$geoNear** — Use function `sdo_nn` or other Oracle Spatial and Graph functions, to return `SDO_GEOMETRY` object-type instances. See Using GeoJSON Geographic Data in Oracle Database JSON Developer’s Guide.
- **$group** — Use GROUP BY. See SELECT in Oracle Database SQL Language Reference.in Oracle Database SQL Language Reference.
- **$limit** — Use SQL `SELECT` with `FETCH NEXT`. See SELECT - row_limiting_clause in Oracle Database SQL Language Reference in Oracle Database SQL Language Reference.
- **$lookup** — Use a SQL join. See Joins in Oracle Database SQL Language Reference.
- **$match** — Use function `json_value` or condition `json_exists`, in a WHERE clause.
- **$max** — Use aggregate function `max`. See MAX in Oracle Database SQL Language Reference in Oracle Database SQL Language Reference.
- **$merge** — Use MERGE. See MERGE in Oracle Database SQL Language Reference.
- **$min** — Use aggregate function `min`. See MIN in Oracle Database SQL Language Reference in Oracle Database SQL Language Reference.
- **$out** — Use INSERT AS SELECT. See INSERT in Oracle Database SQL Language Reference.
- **$project** — Use the JSON generation functions or updating function `json_transform`. See Generation of JSON Data Using SQL and Oracle SQL Function JSON_TRANSFORM in Oracle Database JSON Developer’s Guide.
$redact — Use JSON updating function `json_transform`. See Oracle SQL Function `JSON_TRANSFORM` in Oracle Database JSON Developer’s Guide.

$search — Use Oracle SQL condition `json_textcontains`. See Oracle SQL Condition `JSON_TEXTCONTAINS` in Oracle Database JSON Developer’s Guide.

$set — Use a JSON updating function: `json_transform` or `json_mergepatch`. See Oracle SQL Function `JSON_TRANSFORM` and Oracle SQL Function `JSON_MERGE_PATCH` in Oracle Database JSON Developer’s Guide.

$skip — Use SQL `SELECT` with `OFFSET`. See SELECT - row_limited_clause in Oracle Database SQL Language Reference.

$sort — Use `ORDER BY`. See Sorting Query Results in Oracle Database SQL Language Reference.

$sum — Use aggregate function `sum`. See SUM in Oracle Database SQL Language Reference.

$unionWith — Use `UNION`. See Set Operators in Oracle Database SQL Language Reference.

$unset — Use a JSON updating function: `json_transform` or `json_mergepatch`. See Oracle SQL Function `JSON_TRANSFORM` and Oracle SQL Function `JSON_MERGE_PATCH` in Oracle Database JSON Developer’s Guide.

$unwind — Use function `json_table`. See SQL/JSON Function `JSON_TABLE` in Oracle Database JSON Developer’s Guide.

You can convert relational and other kinds of data (including spatial and graph data) to JSON data using the SQL/JSON generation functions. You can join JSON data from multiple tables and collections with a single SQL `FROM` clause.

Example 2-7  Replacing MongoDB Aggregation Pipeline Code with SQL Code

This example calculates average revenues by zip code. It first shows a MongoDB aggregation pipeline expression to do this; then it shows equivalent SQL code.

**MongoDB aggregation pipeline:**

This code tells MongoDB how to calculate the result; it specifies the order of execution.

```
db.sales.aggregate(
    [{
        "$group": {
            "_id": "$address.zip",
            "avgRev": {
                "$avg": "$revenue"}
        },
        "$sort": {
            "avgRev": -1
        }
    ]
)
```

**SQL:**

This code specifies the grouping and order of the output presentation *declaratively*. It does not specify *how* the computation is to be carried out, including the order of execution. It simply says that the results are to be grouped by zipcode and presented in descending order of the average revenue figures. The query returns rows of two columns with scalar values for zipcode (a string) and average revenue (a number).

```
SELECT s.data.address.zip.string(),
       avg(s.data.revenue.number())
FROM sales s
```
GROUP BY s.data.address.zip.string()
ORDER BY 2 DESC;

The following query is similar, but it provides the result as rows of JSON objects, each with a string field `zip`, for the zipcode, and a numeric field `avgRev`, for the average revenue. SQL/JSON generation function `json_object` constructs JSON objects from the results of evaluating its argument SQL expressions.

```sql
SELECT json_object('zip'    : s.data.address.zip.string(),
                   'avgRev' : avg(s.data.revenue.number()))
FROM sales s
GROUP BY s.data.address.zip.string()
ORDER BY avg(s.data.revenue.number()) DESC;
```

**Related Topics**

- **Indexing and Performance Tuning**
  Oracle Database offers multiple technologies to accelerate queries over JSON data, including indexes, materialized views, in-memory column storage, and Exadata storage-cell pushdown. Which performance-tuning approaches you take depend on the needs of your application.

**See Also:**
- Simple Dot-Notation Access to JSON Data in *Oracle Database JSON Developer's Guide*
- SQL/JSON Function `JSON_VALUE` in *Oracle Database JSON Developer's Guide*
- SQL/JSON Function `JSON_TABLE` in *Oracle Database JSON Developer's Guide*
- Generation of JSON Data Using SQL in *Oracle Database JSON Developer's Guide*
- Query JSON Data in *Oracle Database JSON Developer's Guide* for information about extracting JSON data
- Creating Relational Views of JSON Documents in the JSON Page of *Using Oracle Database Actions*

## 2.5 MongoDB Documents and Oracle Database

Presented here is the relationship between a JSON document used by MongoDB and the same content as a JSON document used by Oracle Database.

You can migrate an existing application and its data from MongoDB to Oracle Database, or you can develop new applications on Oracle Database, which use the same or similar data as applications on MongoDB. JSON data in both cases is stored in **documents**.
It's helpful to have a general understanding of the differences between the documents used by MongoDB and those used by Oracle Database. In particular, it helps to understand what happens to a MongoDB document that you import, to make it usable with Oracle Database.

Some of the information here presents details that you can ignore if you read this topic just to get a high-level view. But it's good to be aware of what's involved; you may want to revisit this at some point.

When you import a collection of MongoDB documents, the key and the content of each document are converted to forms appropriate for Oracle Database.

A MongoDB document has a native binary JSON format called BSON. An Oracle Database document has a native binary JSON format called OSON. So one change that's made to your MongoDB document is to translate its binary format from BSON to OSON. This translation applies to both the key and the content of a document.

**Document Key: Differences and Conversion**

For MongoDB, the unique key of a document, which identifies it, is the value of mandatory field `_id`, in the document itself. For Oracle Database, the unique key that identifies a document is separate from the document; the key is stored in a separate database column from the column that stores the document. The key column has is named `id`, and it is the primary key column for the table that stores your collection data.

When you import a collection into Oracle Database, Oracle Database API for MongoDB creates `id` column values from the values of field `_id` in your MongoDB documents. MongoDB field `_id` can have values of several different data types. The Oracle Database `id` column that corresponds to that field is always of SQL data type `VARCHAR2` (character data; in other words, a string).

The `_id` field in your imported documents is untouched during import or thereafter. Oracle Database doesn't use it — it uses column `id` instead. But it also doesn't change it, so any use your application might make of that field is still valid. Field `_id` in your documents is never changed; even applications cannot change (delete or update) it.

If you need to work with your documents using SQL or Simplified Oracle Document Access (SODA) then you can directly use column `id`. You can easily use that primary-key column to join JSON data with other database data, for instance. The documents that result from importing from MongoDB are SODA documents (with native binary OSON data).

Be aware of these considerations that result from the separation of document key from document:

- Though all documents imported from MongoDB will continue to have their `_id` fields, for Oracle Database the documents in a JSON collection need not have an `_id` field. And because, for Oracle Database, a document and its key are separate, a document other than one imported from MongoDB could have an `_id` field that has no relation whatsoever with the document key.
- Because MongoDB allows `_id` values of different types, and these are all converted to string values (`VARCHAR2`), if for some reason your collection has documents with `_id` values "123" (JSON string) and 123 (JSON number) then importing the collection will raise a duplicate-key error, because those values would each be translated as the same string value for column `id`.

BSON values of field `_id` are converted to `VARCHAR2` column `id` values according to Table 2-2. If an `_id` field value is any type not listed in the table then it is replaced by a generated ObjectId value, which is then converted to the `id` column value.
Table 2-1  Conversion of BSON Field _id Value To Column ID VARCHAR2 Value

<table>
<thead>
<tr>
<th>_id Field Type</th>
<th>ID Column VARCHAR2 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Canonical numeric format string</td>
</tr>
<tr>
<td>32-bit integer</td>
<td>Canonical numeric format string</td>
</tr>
<tr>
<td>64-bit integer</td>
<td>Canonical numeric format string</td>
</tr>
<tr>
<td>Decimal128</td>
<td>Canonical numeric format string</td>
</tr>
<tr>
<td>String</td>
<td>No conversion, including no character escaping</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Lowercase hexadecimal string</td>
</tr>
<tr>
<td>Binary data (UUID)</td>
<td>Lowercase hexadecimal string</td>
</tr>
<tr>
<td>Binary data (non-UUID)</td>
<td>Uppercase hexadecimal string</td>
</tr>
</tbody>
</table>

The canonical numeric format for a VARCHAR2 value is as follows:

- If the input number has no fractional part (it is integral), and if it can be rendered in 40 digits or less, then it is rendered as an integer. If necessary, trailing zeros are used, to avoid notation with an exponent. For example, 1000000000 is used instead of 1E+9.

- If the input number has a fractional part, the number is rendered in 40 digits or less with a decimal point separator. If necessary, zeros are used to avoid notation with an exponent. For example, 0.00001 is used instead of 1E-5.

- If conversion of the input number would result in a loss of digit precision in the 40-digit format, the number is instead rendered with an exponent. This can happen for a number whose absolute value is extremely small or extremely large, even if the number is integral. For example, 1E100 is used, to avoid a 1 followed by 100 zeros.

In practice, this canonical numeric format means that in most cases the numeric _id field value results in an obvious, or "pretty" VARCHAR2 value for column _id. A format that uses an exponent is used only when necessary, which generally means infrequently.

Document Content Conversion

Two general considerations:

- BSON format allows duplicate field values in the same object. OSON format does not. When converting to OSON, detection of duplicate fields in BSON data raises an error.

- OSON format has no notion of the order of fields in an object; applications cannot depend on or expect any particular order (in keeping with the JSON standard). BSON format maintains the order of object fields; applications can depend on the order not changing.

Table 2-2 specifies the type mappings that are applied when converting scalar BSON data to scalar OSON data. The OSON scalar types used are SQL data types, except as noted. Any BSON types not listed are not converted; instead, an error is raised when they are encountered. This includes BSON types regex, and JavaScript.
### Table 2-2   JSON Scalar Type Conversions: BSON to OSON Format

<table>
<thead>
<tr>
<th>BSON Type</th>
<th>OSON Type†</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>BINARY_DOUBLE</td>
<td>NA</td>
</tr>
<tr>
<td>32-bit integer</td>
<td>NUMBER (Oracle number)</td>
<td>Flagged as int.</td>
</tr>
<tr>
<td>64-bit integer</td>
<td>NUMBER (Oracle number)</td>
<td>Flagged as long.</td>
</tr>
<tr>
<td>Decimal128</td>
<td>NUMBER (Oracle number)</td>
<td>Flagged as decimal.  <strong>Note:</strong> This conversion can be lossy.</td>
</tr>
<tr>
<td>Date</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Always UTC time zone.</td>
</tr>
<tr>
<td>String</td>
<td>VARCHAR2</td>
<td>Always in character set AL32UTF8 (Unicode UTF-8).</td>
</tr>
<tr>
<td>Boolean</td>
<td>BOOLEAN</td>
<td>There is no Oracle SQL type for this.</td>
</tr>
<tr>
<td>ObjectId</td>
<td>ID(RAW(12))</td>
<td>NA</td>
</tr>
<tr>
<td>Binary data (UUID)</td>
<td>ID(RAW(16))</td>
<td>NA</td>
</tr>
<tr>
<td>Binary data (non-UUID)</td>
<td>RAW</td>
<td>NA</td>
</tr>
<tr>
<td>Null</td>
<td>NULL</td>
<td>Used for JSON null.</td>
</tr>
</tbody>
</table>

† These are SQL data types, except as noted.

### Related Topics

- **Other Differences Between MongoDB and Oracle Database**
  Various differences between MongoDB and Oracle Database are described. These differences are generally not covered in other topics. Consider these differences when you migrate an application to Oracle Database or you develop a new application for Oracle Database that uses MongoDB commands.

- **Users, Authentication, and Authorization**
  Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

---

### See Also:

Overview of SODA Documents in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*

---

### 2.6 Other Differences Between MongoDB and Oracle Database

Various differences between MongoDB and Oracle Database are described. These differences are generally not covered in other topics. Consider these differences when you migrate an application to Oracle Database or you develop a new application for Oracle Database that uses MongoDB commands.

- **With MongoDB, fields in a JSON object are ordered. With Oracle Database, they are not ordered.** For example, field `_id` is not necessarily the first field in an object. Applications must not expect or rely on any particular field order. According to the JSON language...
standard, object fields are not ordered; only array elements are ordered. See JSON Syntax and the Data It Represents in Oracle Database JSON Developer’s Guide.

• With MongoDB, the value of field \_id can be a JSON object. Oracle Database API for MongoDB supports only BSON types ObjectId, String, Double, 32-bit integer, 64-bit integer, Decimal128, and Binary data (subtype for UUID) for field \_id; an error is raised for any other type. See BSON Types.

If you are migrating an existing application that expects object values for \_id then consider copying the values of field \_id in your data to some new field and using a string value for \_id.

• Read and write concerns regarding MongoDB transactions do not apply to Oracle Database. Oracle Database transactions are fully ACID-compliant, and thus reliable — atomicity, consistency, isolation, and durability. ACID compliance ensures that your data remains accurate and consistent despite any failure that might occur while processing a transaction.

• Oracle API for MongoDB does not support the following MongoDB transaction capabilities:
  – Inclusion of DDL operations, such as createCollection, within a transaction. Attempts to create a collection or an index within a transaction raise an error.
  – Inclusion of operations across multiple databases. All operations within a transaction must be confined to a single database (schema). Otherwise, an error is raised.

• Retryable writes or commits when an error is raised.
  MongoDB retryWrite operations raise an error. If you use a driver that has retryWrite turned on by default, then set retryWrites=false in your connection string to turn this off.

• Oracle Database and MongoDB have different read isolation and consistency levels. Oracle Database API for MongoDB uses read-committed consistency as described in Data Concurrency and Consistency of Oracle Database Concepts.

• Oracle Database API for MongoDB supports only the PLAIN (LDAP SASL) authentication mechanism, and it relies on Oracle Database authentication and authorization.

• Oracle Database does not support the MongoDB collation field for any command (such as find). An error is raised if you use field collation. Oracle collates values using the Unicode binary collation order.

• MongoDB allows different collections in the same “database” to have indexes of the same name. This is not allowed in Oracle Database — the name of an index must be unique across all collections of a given database schema (“database”).

• The maximum size of a document for MongoDB is 16 MB. The maximum size for Oracle Database (and thus for the MongoDB API) is 32 MB.

Related Topics

• MongoDB Documents and Oracle Database
  Presented here is the relationship between a JSON document used by MongoDB and the same content as a JSON document used by Oracle Database.
• Users, Authentication, and Authorization
Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

See Also:
Unicode Collation Algorithm, Unicode® Technical Standard #10
Support for MongoDB Operations — Reference

MongoDB commands supported by Oracle Database are listed, together with information about their support.

Only MongoDB commands that are supported are listed. Unsupported commands raise an error. A command that is ignored is considered supported, but it is listed as a no-op.

Oracle Database API for MongoDB supports requests from MongoDB client versions 2.4 through 4.0.

- **Collection and Index Definition Commands (DDL)**
  Supported MongoDB commands that create and drop collections are listed. These correspond to Oracle SQL Data Definition Language (DDL) operations.

- **Document (Collection Content) Commands (DML)**
  Supported MongoDB commands that create and act on the content of collections (their documents) are listed. These correspond to Oracle SQL Data Manipulation Language (DML) operations.

- **Data Information Commands**
  MongoDB commands that provide information about collections and the "databases" (Oracle Database schemas) they belong to are described. These correspond to Oracle SQL Data Dictionary operations.

- **Administrative Commands**
  Administrative commands, such as obtaining database server and connection information, are described.

- **Authentication and Authorization Commands**
  Oracle Database commands, such as logout, that authenticate or authorize users (Oracle Database schemas) are described.

- **Session Commands**
  Supported MongoDB commands that provide information about the server-side sessions that are used by the current Oracle Database API for MongoDB instance are described.

### 3.1 Collection and Index Definition Commands (DDL)

Supported MongoDB commands that create and drop collections are listed. These correspond to Oracle SQL Data Definition Language (DDL) operations.

MongoDB commands to create or drop indexes are ignored by Oracle Database API for MongoDB. You must instead create Oracle Database indexes that are relevant for your JSON data.

To index data in your JSON collections you can use Oracle SQL or Simple Oracle Document Access (SODA). You can use the JSON page (JSON workshop) in Database Actions to index data using SQL interactively.

- create
If the specified collection already exists then the operation simply succeeds.

**Note:**

Besides creating a collection with explicit use of command `create`, a collection is automatically created upon its first insertion of a document. That is, to create a collection it is sufficient to refer to it by name when inserting a document into it.

- **drop**
- **dropDatabase**

This command *deletes all collections* in the database schema (“database”). It does not delete (drop) the schema itself. The command is available only to a user who is logged in with role `root` — see command `saslStart` in Authentication and Authorization Commands for information about `root` users.

- **repairDatabase**

This command is ignored (a no-op — no error is raised).

**See Also:**

The JSON Page in Using Oracle Database Actions for information about JSON workshop

### 3.2 Document (Collection Content) Commands (DML)

Supported MongoDB commands that create and act on the content of collections (their documents) are listed. These correspond to Oracle SQL Data Manipulation Language (DML) operations.

Such commands include creating, reading, updating, and deleting documents (CRUD commands — create, read, update, or delete documents), as well as aggregate operations such as counting documents.

- **aggregate**
- **count**
  
  **Supported fields:** query.
- **delete**
  
  **Supported fields:** deletes, ordered.

  These fields in an element of the deletes array are supported: q, limit.

  The same query-by-example (QBE) operators are supported as for command find.
- **distinct**
  
  **Supported fields:** key, query.
This command lists the distinct scalar values that match the path specified by input field key. ("Distinct" here means there are no duplicate values in the result.) Unlike MongoDB, non-scalar values addressed by the path are discarded (not listed).

- **find**
  
  **Supported fields:** batchSize, filter, limit, projection, returnKey, singleBatch, skip, sort.

  `returnKey` can only return the primary key (e.g. the ObjectID) associated with the documents found. You cannot use it to return only the index key if an index is used to support the query.

  `$` cannot be used in a projection specification. Only simple field selections or omissions can be performed.

  When a `sort` is performed, by default it uses a *string* (lexicographical) ordering. To request a numeric ordering, date ordering, or timestamp ordering, you use a `hint`, providing the relevant JSON scalar type with `$type`.

  For example, the following code requests an ascending lexicographical sort on field `name`, then an ascending numeric sort on field `age`, and then a descending date-time (that is, reverse chronological) sort on field `birthday`. (A positive number, such as 1, means ascending; a negative number, such as -1, means descending.)

  ```javascript
  find().sort({"name":1, "age":1, "birthday":-1}).hint({"$type":
  {"age":"number", "birthday":"dateTime"}})
  ```

  The JSON scalar types you can specify with `$type` are as follows:
  
  - string (default)
  - number
  - date — A date with no time component.
  - dateTime — A timestamp: a date with a time component.

  The following query-by-example (QBE) operators are supported:
  
  - Comparison and logical: $eq, $gt, $gte, $in, $lt, $lte, $ne, $nin, $and, $not, $nor, $or.
  
  - Element and evaluation: $type, $regex, $text.
  
  - Geospatial: $geoIntersects, $geoWithin, $near, $nearSphere.
  
  - Array: $all, $elemMatch.

- **findAndModify**
  
  **Supported fields:** arrayFilters, fields, new, query, remove, sort, update, upsert.

  These *field update* operators are supported: $bit, $currentDate, $inc, $min, $max, $mul, $rename, $set, $setOnInsert, $unset.

  These *array update* operators are supported: $, $[], $<identifier>, $addToOffset, $pop, $pull, $pullAll, $push.

  These *array update* operator *modifiers* are supported: $each, $position, $slice, $sort.

  The same query-by-example (QBE) operators are supported as for command `find`. 

---

*Chapter 3*

*Document (Collection Content) Commands (DML)*
• **getMore**  
  **Supported fields:** batchSize, collection.

• **insert**  
  **Supported fields:** documents.

• **killCursors**  
  **Supported field:** cursors.

• **update**  
  **Supported fields:** ordered, updates.

  These fields in an element of the updates array are supported: arrayFilters, multi, q, u, upsert.

  The returned response contains these fields: n, nModified, upserted, writeErrors. Array upserted contains only the document _id values — no index.

  The same query-by-example (QBE) operators are supported as for command find.

### 3.3 Data Information Commands

MongoDB commands that provide information about collections and the "databases" (Oracle Database schemas) they belong to are described. These correspond to Oracle SQL Data Dictionary operations.

• **collStats**  
  **Supported fields:** scale.

  This command lists statistics about the specified collection and the Oracle Database indexes that are relevant for it.

• **dataSize**  
  **Supported fields:** estimate, keyPattern, min, max.

  This command lists statistics about the documents in the specified collection.

• **dbStats**  
  **Supported field:** scale.

  This command lists statistics about a given "database" (Oracle Database schema). The information is aggregated from the collections of that schema and the Oracle indexes that are relevant to them.

• **listCollections**  
  This command lists the collections available for a given "database", that is, for a given Oracle Database schema.

• **listDatabases**  
  This command lists the Oracle Database schemas that are enabled for access by Oracle Database API for MongoDB and for Simple Oracle Document Access (SODA).

• **listIndexes**  
  This command lists Oracle Database indexes that are relevant for the specified collection.
3.4 Administrative Commands

Administrative commands, such as obtaining database server and connection information, are described.

- **buildInfo**
  Returns information about the particular Oracle Database API for MongoDB build being used.

- **compact**
  This command is ignored (a no-op — no error is raised).

- **connectionStatus**

- **getFreeMonitoringStatus**
  This command is ignored (a no-op — no error is raised), except that a message is sent saying that this feature is disabled.

- **getLastError**

- **getLog**
  This command is ignored (a no-op — no error is raised).

- **getParameter**
  Only getting the value for `authenticationMechanisms` is supported.

- **hello**

- **hostInfo**

- **isMaster**

- **listCommands**

- **ping**

- **replSetGetStatus**
  This command is ignored (a no-op — no error is raised).

- **resetError**

- **serverStatus**

- **setParameter**
  This command is ignored (a no-op — no error is raised). It returns a response saying that the command is not implemented.

- **whatsmyuri**

3.5 Authentication and Authorization Commands

Oracle Database commands, such as logout, that authenticate or authorize users (Oracle Database schemas) are described.

- **logout**
  With MongoDB, this command logs out all users of a particular "database". With Oracle Database API for MongoDB, the command logs out the current user of a database schema ("database") on a specific port.

- **saslStart**
Only PLAIN (LDAP SASL) authentication is supported.

For example, to log into the admin database schema with its <password> on a given Oracle Database <server>, provide a connection string of this form to the Oracle Database API for MongoDB instance:

```
mongodb://admin:<password>@<server>/admin?
authMechanism=PLAIN&authSource=$external
```

Successful logins are bound to the port on which command saslStart was run. Users of Oracle Database API for MongoDB are Oracle Database schemas. If a logged-in user has Oracle Database privileges CREATE USER, ALTER USER, and DROP USER then the MongoDB database role root is assigned to the connection; otherwise role dbOwner is assigned.

Role root lets a user do all of the following, none of which applies to role dbOwner:

- Proxy to any enabled database schema ("database"). For deployments of Oracle REST Data Services (ORDS), and for an Autonomous Oracle Database, this means that the user can proxy to any ORDS-enabled schema.
- Automatically create schemas ("databases"). When a root user tries to create a collection in a nonexistent schema, that schema is created automatically.
- Drop schemas ("databases"), using command dropDatabase.

Role dbOwner provides a user access only to the database schemalogged in to. A user with this role cannot create schemas ("databases"), drop schemas, or proxy to other schemas.

Related Topics

- Users, Authentication, and Authorization

Oracle Database security differs significantly from that of MongoDB. The security model of Oracle Database API for MongoDB is described: the creation of users, their authentication, and their authorization to perform different operations.

### 3.6 Session Commands

Supported MongoDB commands that provide information about the server-side sessions that are used by the current Oracle Database API for MongoDB instance are described.

- abortTransaction
- commitTransaction
- endSessions
- killAllSessions
- KillAllSessionsByPattern
- killSessions
- refreshSessions
- startSession

This starts a server-side session. It returns a secure, random UUID to the client. A client can create its own UUID. In that case, the server uses that UUID.
Index

Symbols
_id field
   and primary key, 1-2, 2-14
   supported types, 2-17
$\text{type}$ sorting-key type hint, 3-2

A
aggregate command, 3-2
aggregation operators, SQL correspondence, 2-11
aggregation pipeline, 1-2, 2-11
application migration from MongoDB, 2-8
authentication and authorization, 2-6
autonomous database, 1-1

B
BSON
   conversion of document, 2-14
   conversion of field _id, 2-14, 2-17
buildInfo command, 3-5

C
C driver version, 1-2
C# driver version, 1-2
collation field, 2-17
collection, definition, 1-2
collStats command, 3-4
commands
   about collections and schemas, 3-4
   administrative, 3-5
   aggregate, 3-2
   authentication and authorization, 3-5
   buildInfo, 3-5
   collection and index definition (DDL), 3-1
collStats, 3-4
   compact, 3-5
   connectionStatus, 3-5
count, 3-2
create, 3-1
data information, 3-4
commands (continued)
dataSize, 3-4
dbStats, 3-4
delete, 3-2
distinct, 3-2
document (DML), 3-2
drop, 3-1
dropDatabase, 3-1
dropCollection, 3-1
error information, 3-5
findAndModify, 3-2
getFreeMonitoringStatus, 3-5
getLastError, 3-5
getLog, 3-5
getMore, 3-2
getParameter, 3-5
hello, 3-5
hostInfo, 3-5
insert, 3-2
isMaster, 3-5
killAllSessions, 3-6
KillAllSessionsByPattern, 3-6
killCursors, 3-2
killSessions, 3-6
listCollections, 3-4
listCommands, 3-5
listDatabases, 3-4
listIndexes, 3-4
log information, 3-5
logout, 3-5
ping, 3-5
refreshSessions, 3-6
repairDatabase, 3-1
replSetGetStatus, 3-5
resetError, 3-5
saslStart, 3-5
serverStatus, 3-5
session, 3-6
setParameter, 3-5
startSession, 3-6
update, 3-2
whatsmyuri, 3-5
compact command, 3-5
Compass version, 1-2
<table>
<thead>
<tr>
<th>Page Dimensions: 612.0x792.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection URI, encoding reserved characters, 2-8</td>
</tr>
<tr>
<td>connectionStatus command, 3-5</td>
</tr>
<tr>
<td>converged database, 1-1</td>
</tr>
<tr>
<td>conversion</td>
</tr>
<tr>
<td>BSON field _id, 2-14</td>
</tr>
<tr>
<td>BSON scalar types, 2-14</td>
</tr>
<tr>
<td>count command, 3-2</td>
</tr>
<tr>
<td>create command, 3-1</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>data migration from MongoDB, 2-8</td>
</tr>
<tr>
<td>database schema, 1-2, 2-6</td>
</tr>
<tr>
<td>Database Tools version, 1-2</td>
</tr>
<tr>
<td>database, definition, 1-2, 2-6</td>
</tr>
<tr>
<td>dataSize command, 3-4</td>
</tr>
<tr>
<td>dbStats command, 3-4</td>
</tr>
<tr>
<td>DDL commands, 3-1</td>
</tr>
<tr>
<td>delete command, 3-2</td>
</tr>
<tr>
<td>DML commands, 3-2</td>
</tr>
<tr>
<td>document</td>
</tr>
<tr>
<td>conversion from BSON, 2-14</td>
</tr>
<tr>
<td>definition, 1-2</td>
</tr>
<tr>
<td>id, 1-2, 2-14</td>
</tr>
<tr>
<td>key, 2-14</td>
</tr>
<tr>
<td>maximum size, 2-17</td>
</tr>
<tr>
<td>drivers, supported, 1-2</td>
</tr>
<tr>
<td>drop command, 3-1</td>
</tr>
<tr>
<td>dropDatabase command, 3-1</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>encoding characters in a URI, 2-8</td>
</tr>
<tr>
<td>endSessions command, 3-6</td>
</tr>
<tr>
<td>escaping characters in a URI, 2-8</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>field order in an object, 2-17</td>
</tr>
<tr>
<td>find command, 3-2</td>
</tr>
<tr>
<td>findAndModify command, 3-2</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>getFreeMonitoringStatus command, 3-5</td>
</tr>
<tr>
<td>getLastError command, 3-5</td>
</tr>
<tr>
<td>getLog command, 3-5</td>
</tr>
<tr>
<td>getMore command, 3-2</td>
</tr>
<tr>
<td>getParameter command, 3-5</td>
</tr>
<tr>
<td>Go driver version, 1-2</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>hello command, 3-5</td>
</tr>
<tr>
<td>hint</td>
</tr>
<tr>
<td>$type (sorting-key type), 3-2</td>
</tr>
<tr>
<td>index, 2-1</td>
</tr>
<tr>
<td>SQL monitoring, 2-1</td>
</tr>
<tr>
<td>hostInfo command, 3-5</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>id column, 1-2, 2-14</td>
</tr>
<tr>
<td>in-memory column storage, 2-1</td>
</tr>
<tr>
<td>index names, unique, 2-17</td>
</tr>
<tr>
<td>INDEX SQL hint, 2-1</td>
</tr>
<tr>
<td>indexes, 1-2, 2-1</td>
</tr>
<tr>
<td>indexing scalar fields, 2-1</td>
</tr>
<tr>
<td>insert command, 3-2</td>
</tr>
<tr>
<td>isMaster command, 3-5</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>Java driver version, 1-2</td>
</tr>
<tr>
<td>JSON database, autonomous, 1-1</td>
</tr>
<tr>
<td>JSON Page, Database Actions, 2-1</td>
</tr>
<tr>
<td>JSON scalar type conversion from BSON, 2-14</td>
</tr>
<tr>
<td>K</td>
</tr>
<tr>
<td>key</td>
</tr>
<tr>
<td>document, 2-14</td>
</tr>
<tr>
<td>primary, 1-2</td>
</tr>
<tr>
<td>killAllSessions command, 3-6</td>
</tr>
<tr>
<td>KillAllSessionsByPattern command, 3-6</td>
</tr>
<tr>
<td>killCursors command, 3-2</td>
</tr>
<tr>
<td>killSessions command, 3-6</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>listCollections command, 3-4</td>
</tr>
<tr>
<td>listCommands command, 3-5</td>
</tr>
<tr>
<td>listDatabases command, 3-5</td>
</tr>
<tr>
<td>listIndexes command, 3-4</td>
</tr>
<tr>
<td>load JSON data, 2-8</td>
</tr>
<tr>
<td>logout command, 3-5</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>materialized views, 2-1</td>
</tr>
<tr>
<td>maximum document size, 2-17</td>
</tr>
<tr>
<td>migration from MongoDB, 2-8</td>
</tr>
<tr>
<td>MongoDB wire protocol, 1-1</td>
</tr>
<tr>
<td>MongoDB, differences from Oracle Database, 2-17</td>
</tr>
</tbody>
</table>
Index

mongodump, 1-2
mongoexport, 1-2
mongoimport, 1-2
mongorestore, 1-2
MongoSH version, 1-2
MONITOR SQL hint, 2-1
monitoring performance, 2-1
multitenant database, 1-1

refreshSessions command, 3-6
repairDatabase command, 3-1
replSetGetStatus command, 3-5
reserved characters in connection URI, 2-8
resetError command, 3-5
roles, 2-6
Ruby driver version, 1-2
Rust driver version, 1-2

Node.js driver version, 1-2

optimizer, 2-11
Oracle Database, differences from MongoDB, 2-17
order of fields in an object, 2-17
OSON format, 2-14

password, in connection URI, 2-8
performance improvement, 2-1
ping command, 3-5
pipeline, aggregation, 1-2, 2-11
primary key, 1-2, 2-14
protocol, MongoDB, 1-1
purpose of Oracle Database API for MongoDB, 1-1
PyMongo (Python) driver version, 1-2

saslStart command, 3-5
scalar type conversion from BSON, 2-14
schema, database, 1-2, 2-6
security, 2-6
serverStatus command, 3-5
setParameter command, 3-5
sorting key type ($type hint), 3-2
SQL (Structured Query Language), 1-1
SQL/JSON, 2-11
startSession command, 3-6
Structured Query Language (SQL), 1-1

tools, supported, 1-2
transactions, 2-17
type conversion from BSON, 2-14

update command, 3-2
URI reserved characters, encoding, 2-8
username, in connection URI, 2-8
users, 1-2, 2-6

whatmyuri command, 3-5
wire protocol, MongoDB, 1-1