ORACLE

Bring SQL to MongoDB API. Really.

AskTom JSON Office Hours

Hermann Bär, Josh Spiegel, Beda Hammerschmidt
Oracle Database Development

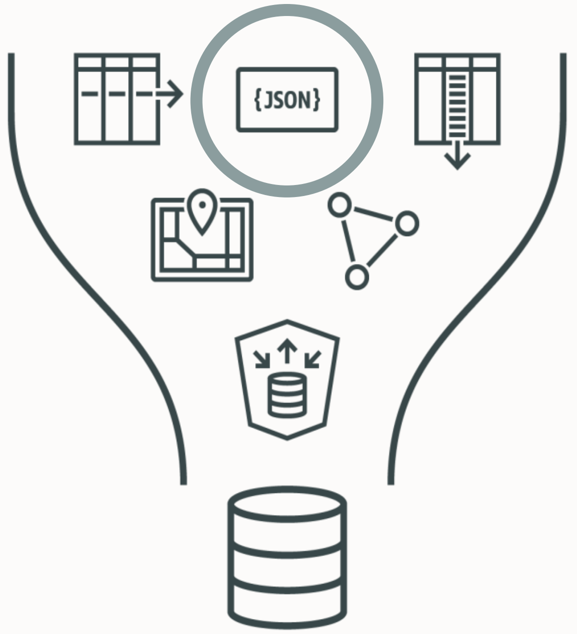
Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

The materials in this presentation pertain to Oracle Health, Oracle, Oracle Cerner, and Cerner Enviza which are all wholly owned subsidiaries of Oracle Corporation. Nothing in this presentation should be taken as indicating that any decisions regarding the integration of any EMEA Cerner and/or Enviza entities have been made where an integration has not already occurred.

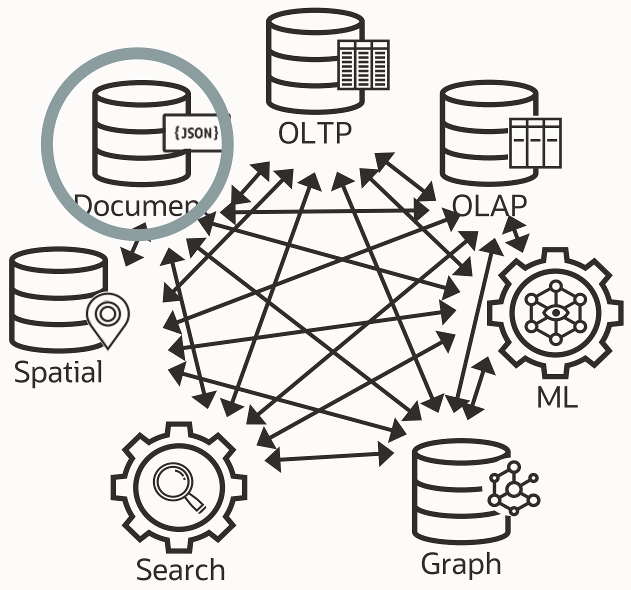
Oracle Converged Database

Converged Database Architecture



for **any** data type or workload

Single-purpose databases

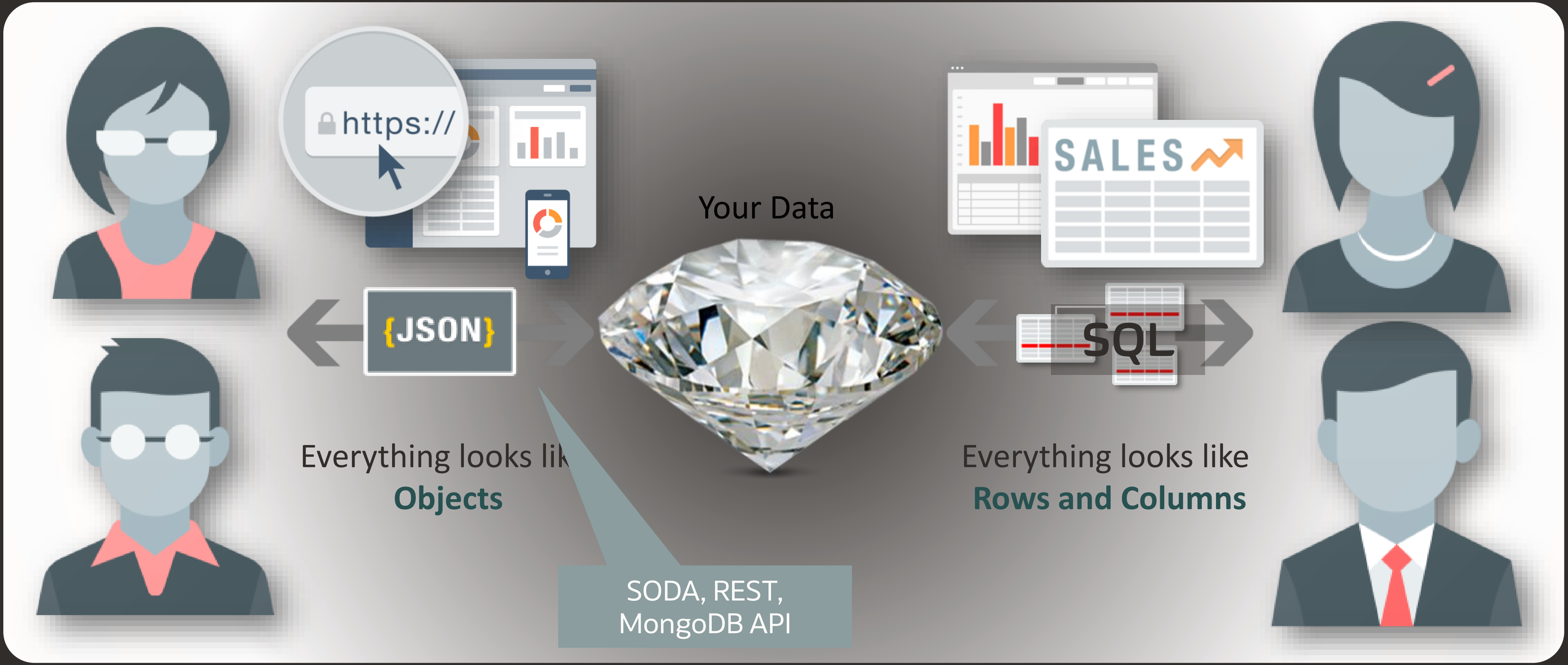


for each data type and workload

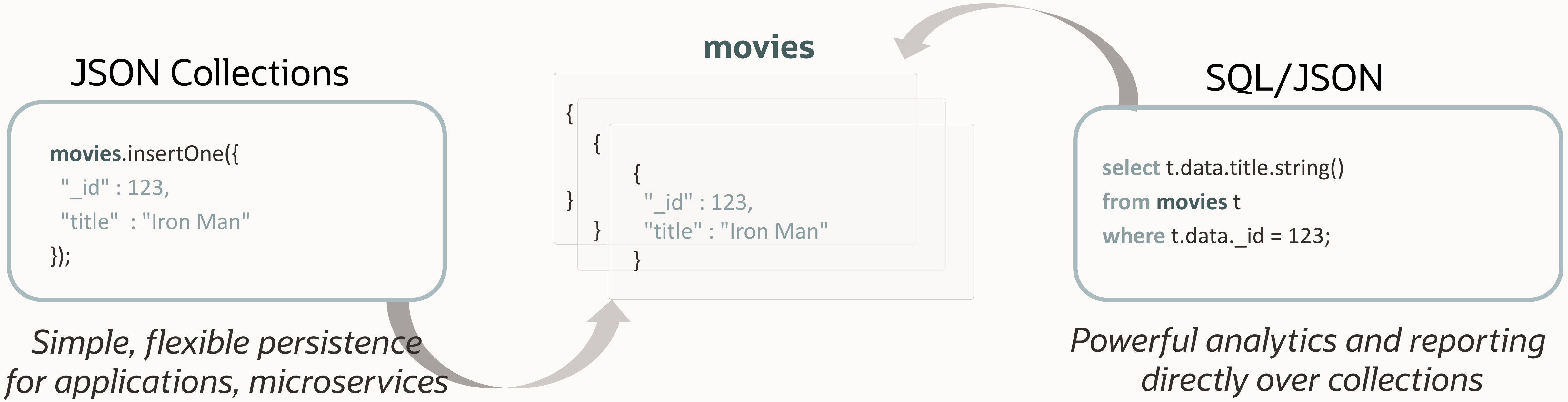
Multiple security models, languages, skills, licenses, etc

Bridging the Gap between JSON and Relational World

Two facets of the Same Data



SQL or Document Store APIs – whenever you need it…



JSON Collections

```
movies.insertOne({
  "_id" : 123,
  "title"  : "Iron Man"
});
```

Simple, flexible persistence for applications, microservices

movies

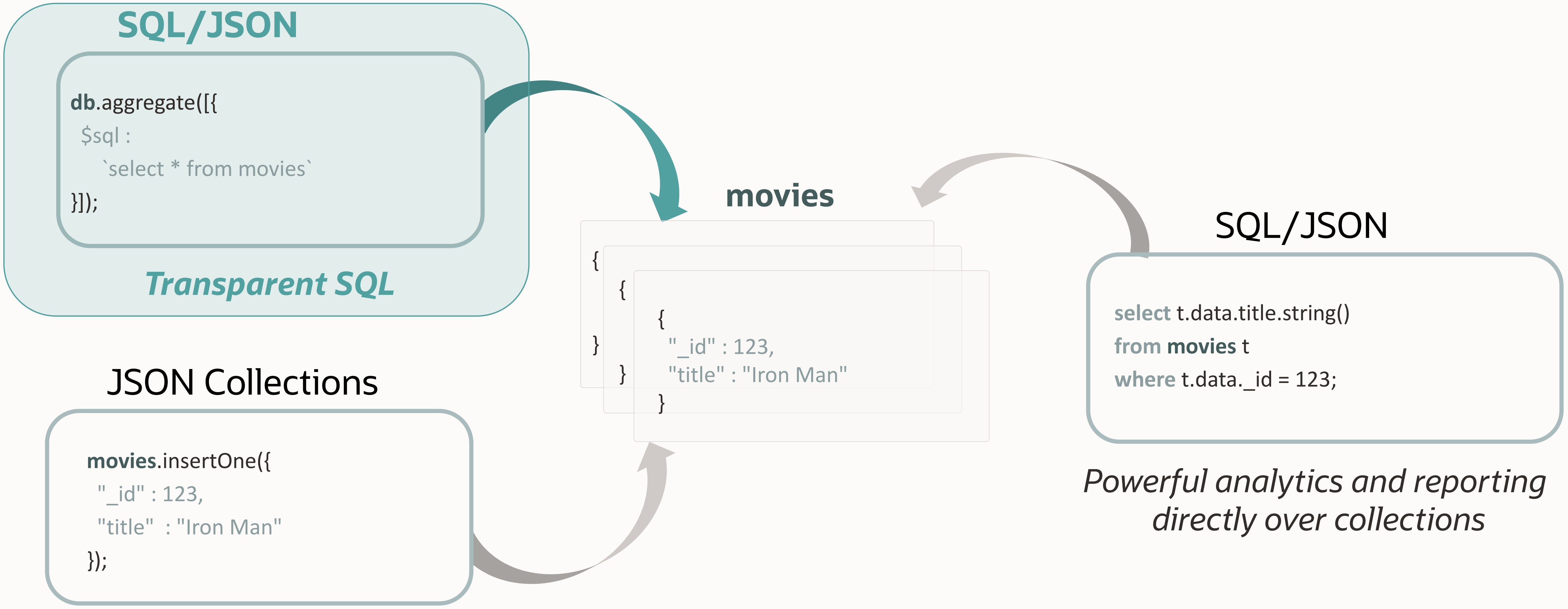
```
{
  {
    {
      "_id" : 123,
      "title" : "Iron Man"
    }
  }
}
```

SQL/JSON

```
select t.data.title.string()
from movies t
where t.data._id = 123;
```

Powerful analytics and reporting directly over collections

SQL or Document Store APIs – whenever and wherever you need it...



SQL/JSON

```
db.aggregate([{
  $sql :
    `select * from movies`
}]);
```

Transparent SQL

movies

```
{
  {
    {
     "_id" : 123,
     "title" : "Iron Man"
    }
  }
}
```

SQL/JSON

```
select t.data.title.string()
from movies t
where t.data._id = 123;
```

Powerful analytics and reporting directly over collections

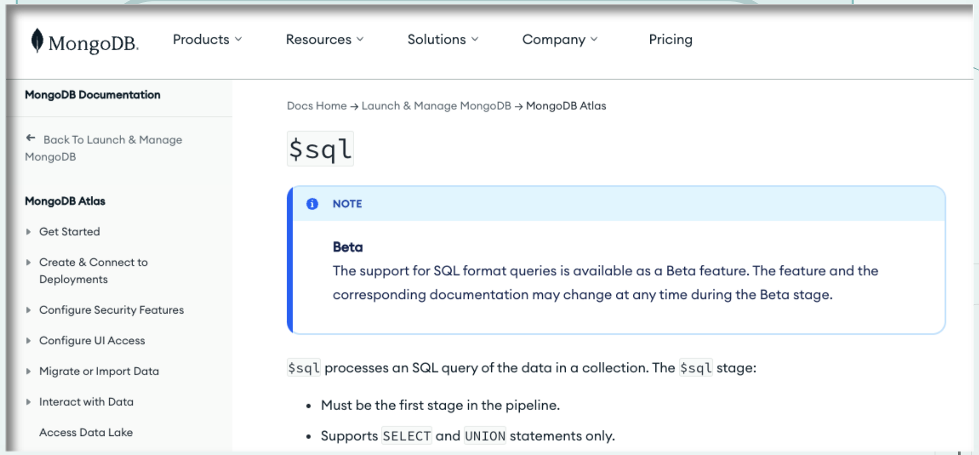
JSON Collections

```
movies.insertOne({
  "_id" : 123,
  "title"  : "Iron Man"
});
```

Simple, flexible persistence for applications, microservices

SQL cannot be that bad ...

SQL/JSON



MongoDB. Products Resources Solutions Company Pricing

MongoDB Documentation

← Back To Launch & Manage MongoDB

MongoDB Atlas

- Get Started
- Create & Connect to Deployments
- Configure Security Features
- Configure UI Access
- Migrate or Import Data
- Interact with Data
- Access Data Lake

Docs Home → Launch & Manage MongoDB → MongoDB Atlas

`$sql`

NOTE

Beta
The support for SQL format queries is available as a Beta feature. The feature and the corresponding documentation may change at any time during the Beta stage.

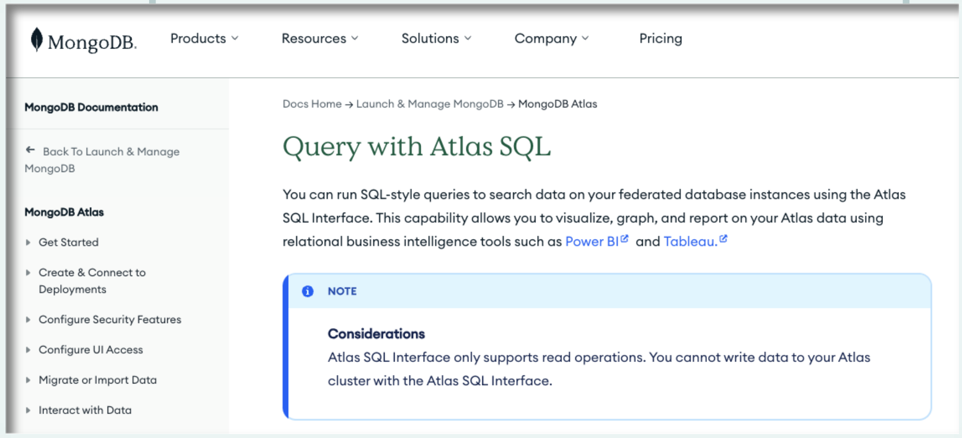
`$sql` processes an SQL query of the data in a collection. The `$sql` stage:

- Must be the first stage in the pipeline.
- Supports `SELECT` and `UNION` statements only.

movies

```
{
  "_id" : 123,
  "title" : "Iron Man"
}
```

SQL/JSON



MongoDB. Products Resources Solutions Company Pricing

MongoDB Documentation

← Back To Launch & Manage MongoDB

MongoDB Atlas

- Get Started
- Create & Connect to Deployments
- Configure Security Features
- Configure UI Access
- Migrate or Import Data
- Interact with Data

Docs Home → Launch & Manage MongoDB → MongoDB Atlas

Query with Atlas SQL

You can run SQL-style queries to search data on your federated database instances using the Atlas SQL Interface. This capability allows you to visualize, graph, and report on your Atlas data using relational business intelligence tools such as Power BI and Tableau.

NOTE

Considerations
Atlas SQL Interface only supports read operations. You cannot write data to your Atlas cluster with the Atlas SQL Interface.

JSON Collections

```
movies.insertOne({
  "_id" : 123,
  "title"  : "Iron Man"
});
```

Simple, flexible persistence for applications, microservices

Oracle Database API for MongoDB

Connect MongoDB client drivers and tools to Oracle Database

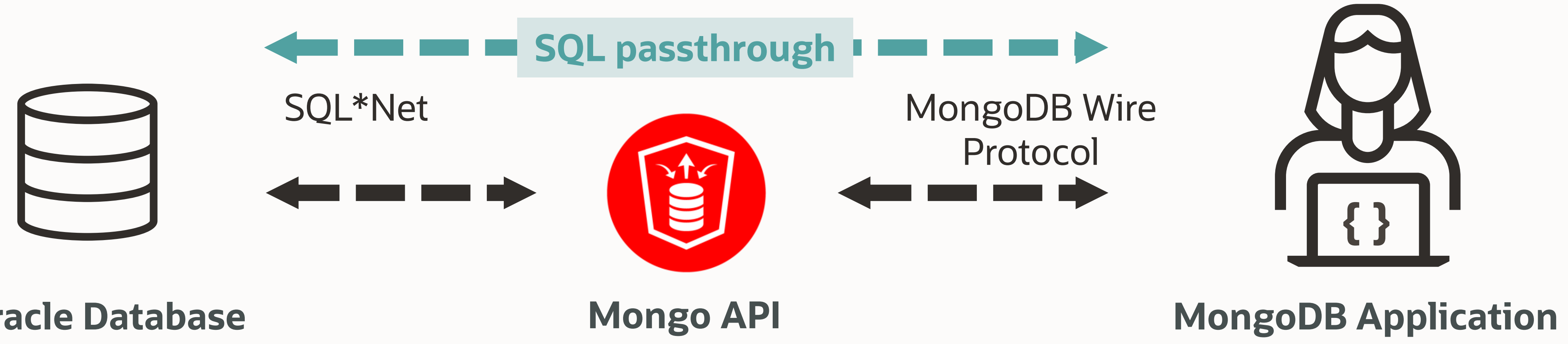
MongoDB does not have tables – it stores collections of JSON documents

- Transparency simplifies migrations from MongoDB to Oracle

MongoDB developers keep using the same skills, tools, and frameworks

Enhance applications with SQL passthrough

- **Statements and data**



$sql in MongoDB API

The Basics

SQL integration with MongoDB API
$sql stage

Syntax

```
{
  $sql: {
    statement:      <SQL-statement>,
    binds:          <variables>,
    format:         <format>,
    dialect:        <dialect>
  }
}
```

Simplified form

```
{
  $sql: <SQL-statement>
}
```

Statement

- Specifies the SQL or PL/SQL to execute

Binds

- Optional
- Specifies the binds parameters
- Positional or named

Format

- Optional
- Only supported value 'oracle'

Dialect

- Optional
- Only supported value 'oracle'

SQL integration with MongoDB API

binds

Parameters

Index [number]

- Positional bind value.
- Inferred if not specified

Name [string]

- Name of bind value

Value [any]

- Bind value

datatype [string]

- Optional
- SQL bind type
- Mapped based on BSON type when not specified

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:my_empno,:my_name)`,
     binds : [
       {name:"my_empno", value:"E123"},
       {name:"my_name", value:"JOHN DOE"}
     ]
  }
}]);

db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:1,:2)`,
     binds : [
       {index:1, value:"E123"},
       {index:2, value:"JOHN DOE"}
     ]
  }
}]);
```

SQL integration with MongoDB API
binds

Single execution case 1: Array containing objects

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:my_empno,:my_name)`,
     binds : [
       {name:"my_empno", value:"E123"},
       {name:"my_name", value:"JOHN DOE"}
     ]
  }}]);
```

Single execution case 2: Array of primitive values

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:1,:2)`,
     binds : [ "E123", "JOHN DOE" ]
  }
}]);
```

Single execution case 3: Value is an object

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:empno,:ename)
     `,
     binds : {"empno":"E123", "ename":"JOHN DOE"}
  }
}]);
```

Multiple executions: Array of values of Case 1, Case 2, or Case 3

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:1,:2)`,
     binds : [
       ["E123", "JOHN DOE"],
       ["E456", "JANE DOE"]
     ]
  }}]);
```

SQL integration with MongoDB API

binds

Single execution case 1: Array containing objects

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:my_empno,:my_name)`,
     binds : [
       {name:"my_empno", value:"E123"},
       {name:"my_name", value:"JOHN DOE"}
     ]
  }}]);
```

Single execution case 2: Array of primitive values

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:1,:2)`,
     binds : [ "E123", "JOHN DOE" ]
  }
}]);
```

Single execution case 3: Value is an object

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:empno,:ename)
     `,
     binds : {"empno":"E123", "ename":"JOHN DOE"}
  }
}]);
```

Multiple executions:
Array of values of Case 1, Case 2, or Case 3

```
db.aggregate([{
  $sql:{
     statement : `
       insert into emp(empno, ename)
       values(:1,:2)`,
     binds : [
       ["E123", "JOHN DOE"],
       ["E456", "JANE DOE"]
     ]
  }}]);
```

SQL integration with MongoDB API
$sql stage

Stage on database and collection level

```
db.aggregate([{ $sql:{ … }}]);
db.movies.aggregate([{ $sql:{ … }}]);
```

Supports all types of SQL operations

- SELECT, DML, and DDL
- PL/SQL

```
db.aggregate([{ $sql: "create index …" }]);
db.aggregate([{ $sql: "insert into …" }]);
db.aggregate([{ $sql: "begin … end" }]);
```

Supports bind variables

- Positional or named

SQL integration with MongoDB API
$sql stage

Stage on database and collection level

```
db.aggregate([{ $sql:{ … }}]);
db.movies.aggregate([{ $sql:{ … }}]);
```

Supports all types of SQL operations

- SELECT, DML, and DDL
- PL/SQL

```
db.aggregate([{ $sql: "create index …" }]);
db.aggregate([{ $sql: "insert into …" }]);
db.aggregate([{ $sql: "begin … end" }]);
```

Supports bind variables

- Positional or named

Single stage or embedded into multiple stages

- If embedded into multiple stages or on a collection
  - Only SELECTs are allowed
  - The SELECT statement must return a JSON object

```
db.movies.aggregate([{ $sql:{ … }}]);
db.movies.aggregate([{ $match: … },
                     { $sql:{ … }},
                     { $limit:   }]);
```

Transactional consistency

- Auto-commit (default)
- Multi-statement (mongo-driven) transactions

```
session = db.getMongo().startSession( { … } );
session.startTransaction( { … } );
```

SQL integration with MongoDB API

SQL SELECT data flow

Each row in a SELECT statement is mapped to a JSON object

```
jason> db.aggregate([{$sql:`select systimestamp my_time, banner where_am_i from v$version`}])
[
  {
    MY_TIME: ISODate('2024-04-05T21:05:28.777Z'),
    WHERE_AM_I: 'Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production'
  }
]
```

SQL integration with MongoDB API

SQL SELECT data flow

Each row in a SELECT statement is mapped to a JSON object

```
jason> db.aggregate([{$sql:`select systimestamp my_time, banner where_am_i from v$version`}])
[
  {
    MY_TIME: ISODate('2024-04-05T21:05:28.777Z'),
    WHERE_AM_I: 'Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production'
  }
]
```

JSON documents of previous stages are available as relational view **INPUT (DATA JSON)***

```
jason> db.movies.aggregate([{ $match: { genre: { $eq: 'Drama' } } },
...     {$sql: `select json_mergepatch(i.DATA, json{'leap_year': leap_year(i.DATA.YEAR.number())}) from INPUT i`},
...     {$project: {"_id": 0, year:1, title: 1, gross: 1, leap_year: 1}},
...     {$sort: {year: 1}}
... ])
[
  {
    title: 'Samson and Delilah',
    year: 1950,
    gross: 25600000,
    leap_year: false
  },
  …
]
```

*Oracle Database 23 only

SQL integration with MongoDB API

SQL SELECT data flow

Each row in a SELECT statement is mapped to a JSON object

```
jason> db.aggregate([{$sql:`select systimestamp my_time, banner where_am_i from v$version`}])
[
  {
    MY_TIME: ISODate('2024-04-05T21:05:28.777Z'),
    WHERE_AM_I: 'Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production'
  }
]
```

JSON documents of previous stages are available as relational view **INPUT (DATA JSON)***

```
jason> db.movies.aggregate([{ $match: { genre: { $eq: 'Drama' } } },
...     {$sql: `select json_mergepatch(i.DATA, json{'leap_year': leap_year(i.DATA.YEAR.number())}) from INPUT i`},
...     {$project: {"_id": 0, year:1, title: 1, gross: 1, leap_year: 1}},
...     {$sort: {year: 1}}
... ])
[
  {
    title: 'Samson and Delilah',
    year: 1950,
    gross: 25600000,
    leap_year: false
  },
  …
]
```

PL/SQL

Data enrichment/transformation

*Oracle Database 23 only

SQL integration with MongoDB API

Data flow for DML, DDL, and PL/SQL

Return set is a single object or array of objects (PL/SQL returns NULL)

- Key ‘result’, value equals the number of rows changed (if known)

DML

```
db.aggregate([{$sql:{
  statement: `insert into emps
              values (:name, :job)`,
  binds: [
{"name": "John", "job": "Programmer"},
{"name": "Jane", "job": "Manager"}
      ]
}}]);
```
```
[ { result: [ 1, 1 ] } ]
```

```
db.aggregate([{$sql:`
      delete from emps`
}]);
```
```
[ { result: 4 } ]
```

SQL integration with MongoDB API

Data flow for DML, DDL, and PL/SQL

Return set is a single object or array of objects (PL/SQL returns NULL)

- Key ‘result’, value equals the number of rows changed (if known)

DML

```
db.aggregate([{$sql:{
  statement: `insert into emps
              values (:name, :job)`,
  binds: [
{"name": "John", "job": "Programmer"},
{"name": "Jane", "job": "Manager"}
      ]
}}]);
[ { result: [ 1, 1 ] } ]
```

```
db.aggregate([{$sql:`
      delete from emps`
}]);
[ { result: 4 } ]
```

DDL

```
db.aggregate([{$sql: `
     create table toto as
     select * from all_users`}])
[ { result: 77 } ]
```

```
db.aggregate([{$sql: `
     create index i_toto
     on toto(username`}])
[ { result: 0 } ]
```

```
db.aggregate([{$sql: `
     drop table toto`}])
[ { result: 0 } ]
```

SQL integration with MongoDB API

Data flow for DML, DDL, and PL/SQL

Return set is a single object or array of objects (PL/SQL returns NULL)

- Key ‘result’, value equals the number of rows changed (if known)

DML

```
db.aggregate([{$sql:{
  statement: `insert into emps
              values (:name, :job)`,
  binds: [
{"name": "John", "job": "Programmer"},
{"name": "Jane", "job": "Manager"}
      ]
}}]);
[ { result: [ 1, 1 ] } ]
```

```
db.aggregate([{$sql:`
      delete from emps`
}]);
[ { result: 4 } ]
```

DDL

```
db.aggregate([{$sql: `
     create table toto as
     select * from all_users`}])
[ { result: 77 } ]
```

```
db.aggregate([{$sql: `
     create index i_toto
     on toto(username`}])
[ { result: 0 } ]
```

```
db.aggregate([{$sql: `
     drop table toto`}])
[ { result: 0 } ]
```

PL/SQL

```
db.aggregate([{$sql:`
   begin
      dbms_lock.sleep(10);
   end;`
}]);
[]
```

Use Cases

The Basics

Common Use Cases

Access to relational data

Relational processing

Bridge the gaps

SQL integration with MongoDB API

New aggregation pipeline stage $sql

- Allows transparent execution of user-defined SQL statements from within Mongo clients
- Passes data from and to MongoDB application

Database support*

Oracle Database 19c and Autonomous Database 19c

- $sql as single stage
- Limited support of aggregation pipeline operators ($match, $skip, $limit, $project, $count, ..)

Oracle Database 23

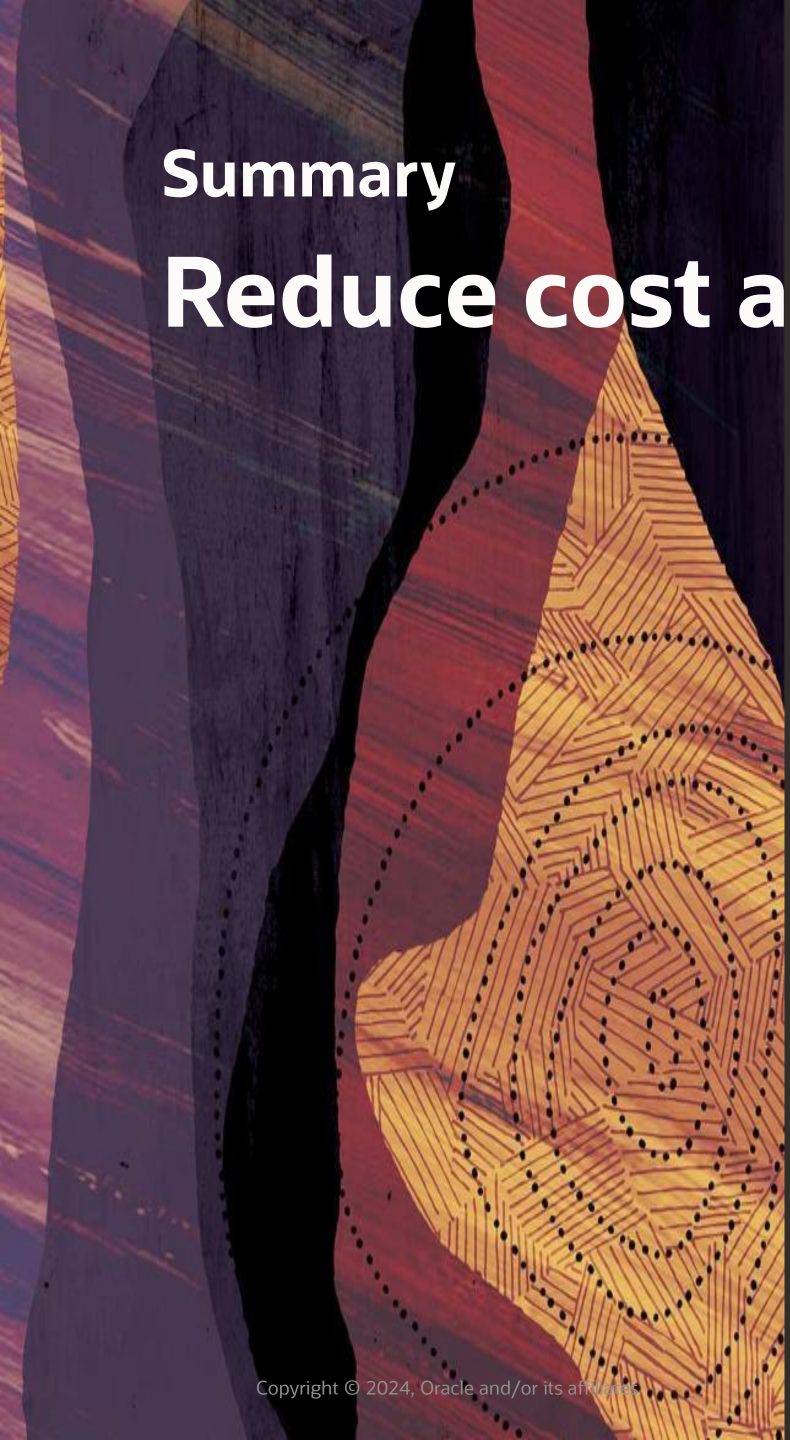
- Integrated operator in aggregation pipeline framework

Available in Autonomous Database Serverless and ORDS 24.1

* Details in the documentation

Summary

Reduce cost and risk. Simplify your work.



1 Store, use, and manage relational data and JSON documents in a single converged database. Unified management, security, consistency model

2 Flexible access with relational and document-store APIs and languages, like SQL, JDBC, MongoDB API, Python, and Oracle SODA

Where to get more information

O.com: Autonomous JSON Database

LiveLabs: Developing with JSON and SODA

LiveLabs: Using the Database API for MongoDB

LiveSQL: SQL/JSON features

Blog: Oracle Database API for MongoDB

Documentation: Overview of Oracle Database API for MongoDB

Documentation: Configure the Oracle Database API for MongoDB

Our mission is to help people see data in new ways, discover insights, unlock endless possibilities.