ORACLE
CloudWorld

Modern Application Development Using JSON and Java

Josh Spiegel
Software Engineer, Oracle Database

@joshjspiegel

https://linkedin.com/in/joshjspiegel/

Outline

Introduction

Why develop using JSON?

Why store JSON Oracle Database?

SQL/JSON

Binary JSON data

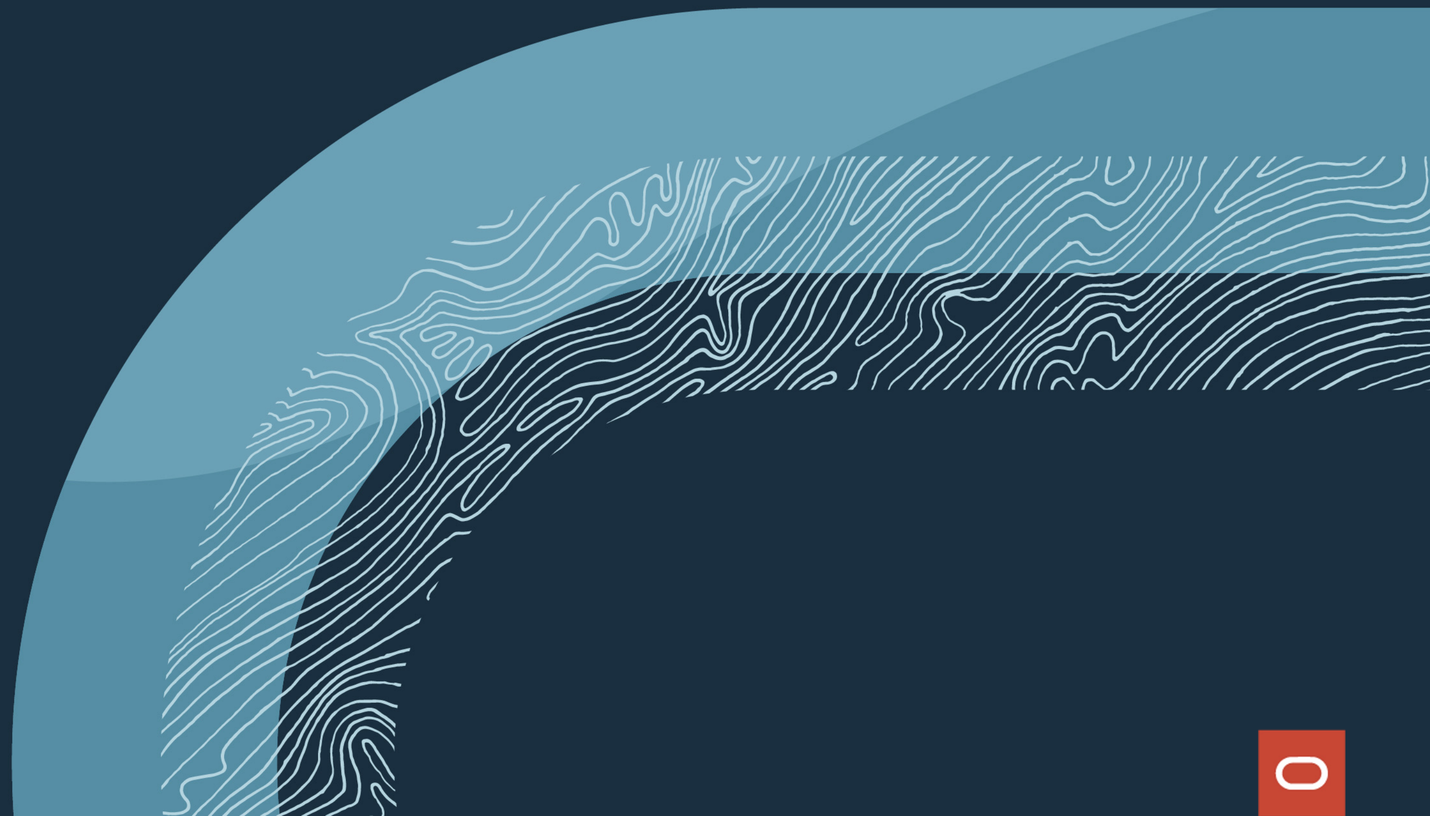
Querying JSON from SQL

JDBC/JSON

Modeling JSON in Java (`oracle.sql.json`)
Binding and reading JSON from a statement

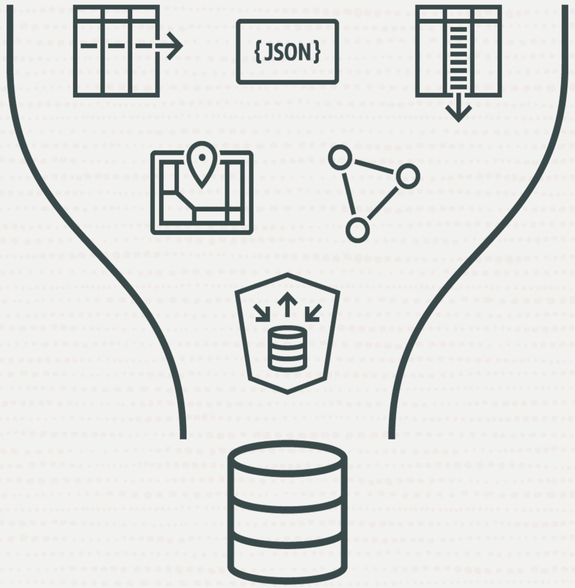
JSON Collections

Oracle API for MongoDB
SODA



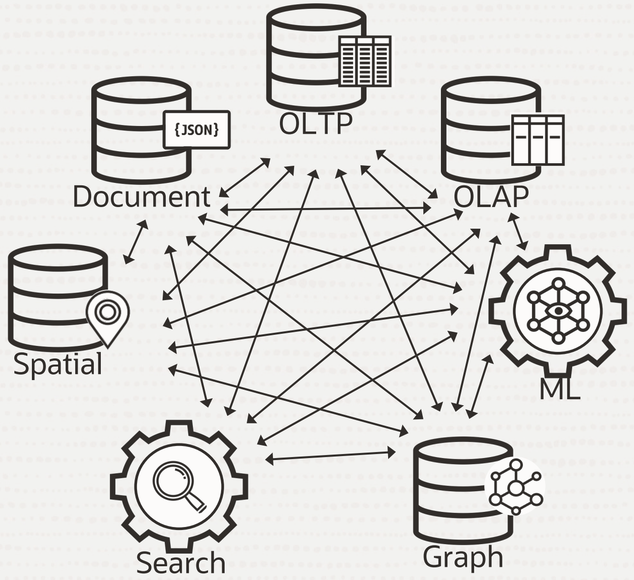
Oracle Converged Database

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

Why JSON?

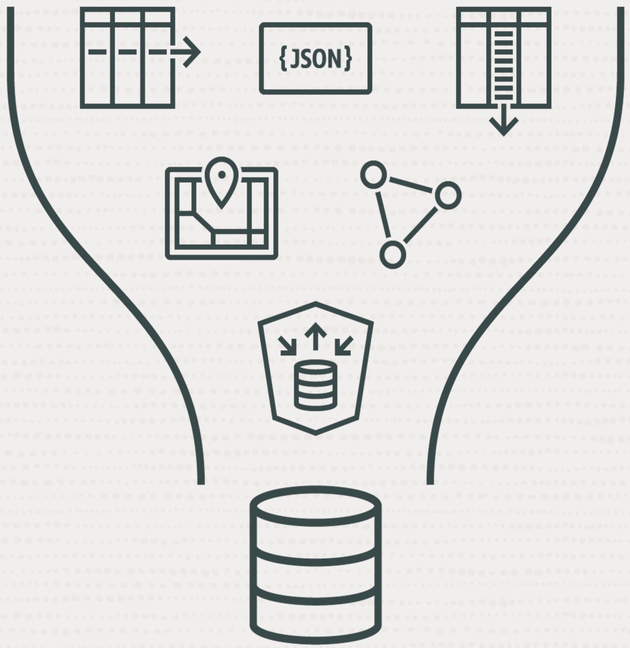
- Schema-flexible
  - Less upfront schema design
  - Applications controls evolution
- Easily consumed by applications
  - Nested structures
  - Maps to application objects
  - Read/write without joins
- Good common format
  - Supported by most programming languages
  - Human readable
  - Simplify data exchange across app, servers, and database tiers

```
{
   "name" : "Thomas Anderson",
   "job"  : "Programmer",
   "addresses" : [
      {
        "street" : "123 Main",
        "city" : "Santa Cruz",
        "zip" : 95041
      }
   ]
}
```

Autonomous JSON Database

- Low-latency, scalable, JSON storage
- **MongoDB** APIs or **SQL**
- No database management
- Always-**free** service
- All the features of the Autonomous Database

The converged database
as a **managed cloud service**



Autonomous JSON Database

Elastic compute and storage

Single-digit latency reads and writes

Highly available

Low price, always-free tier

Autonomous JSON Database

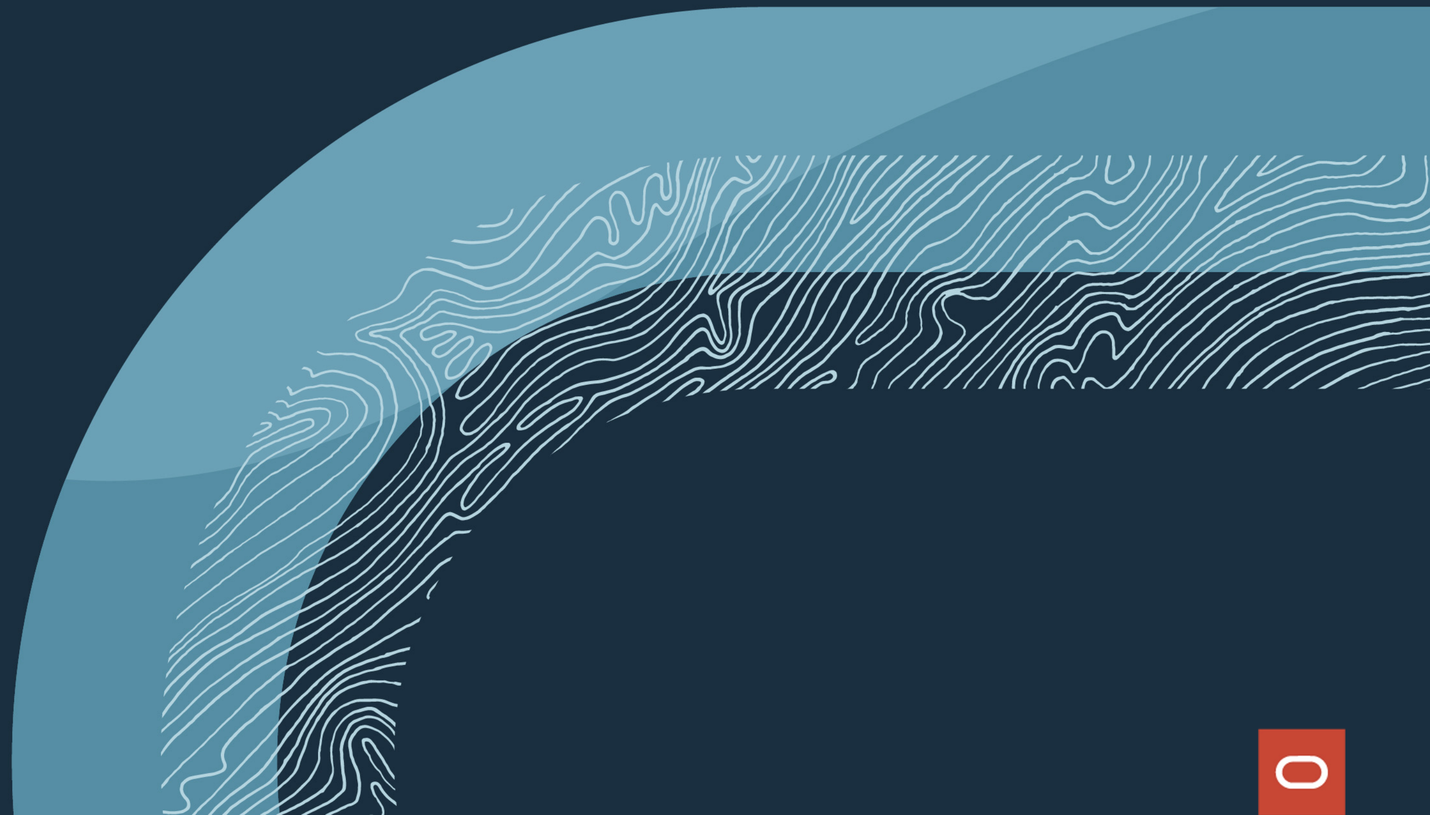
More than a NoSQL document store.

- SQL directly over JSON
- Analytics
- Reporting
- Caching
- Indexing, full-text search
- Secure by default

SQL/JSON

Binary JSON Data

Querying JSON with SQL



JSON Data Type (21c+)

- Backed by binary JSON
- Simplified SQL access
- Extended types
- Performance

```
CREATE TABLE emp (data JSON);

INSERT INTO emp VALUES (
  JSON {
    'name' : 'Smith',
    'job'  : 'Agent'
  }
);
```

SQL/JSON

- Use SQL to query JSON data
  - JSON to relational
  - Relational to JSON
- Joins, aggregation, projection
- Construct new JSON values
- Update JSON values
- Unnest nested arrays

JSON aggregation and construction

```
SELECT JSON {
  'count' : count(*),
  'dept' : e.jcol.dept.string()
}
FROM emps e
WHERE e.jcol.title.string() = 'Engineer'
GROUP BY e.jcol.dept.string()
```

JSON unnesting

```
SELECT name, title, street
FROM emps NESTED jcol COLUMNS (
  name, title,
  NESTED address[*] columns (
    street
  )
)
WHERE name = 'Anderson'
```

JSON Storage History

12g

JSON-text storage
and query processing
(clob, blob, varchar2)

18c

19c

Binary JSON storage (OSON)
and Mongo API support added
for **Autonomous** Databases
(in BLOB columns)

21c

Native JSON datatype and
collections backed by OSON (all
database types)

Why OSON? - Extended types

Standard

- OBJECT { }
- ARRAY [ ]
- STRING
- TRUE/FALSE
- NULL
- NUMBER

Extended

- BINARY_FLOAT
- BINARY_DOUBLE
- TIMESTAMP/DATE
- INTERVALDS/INTERVALYM
- RAW

Fidelity with relational data

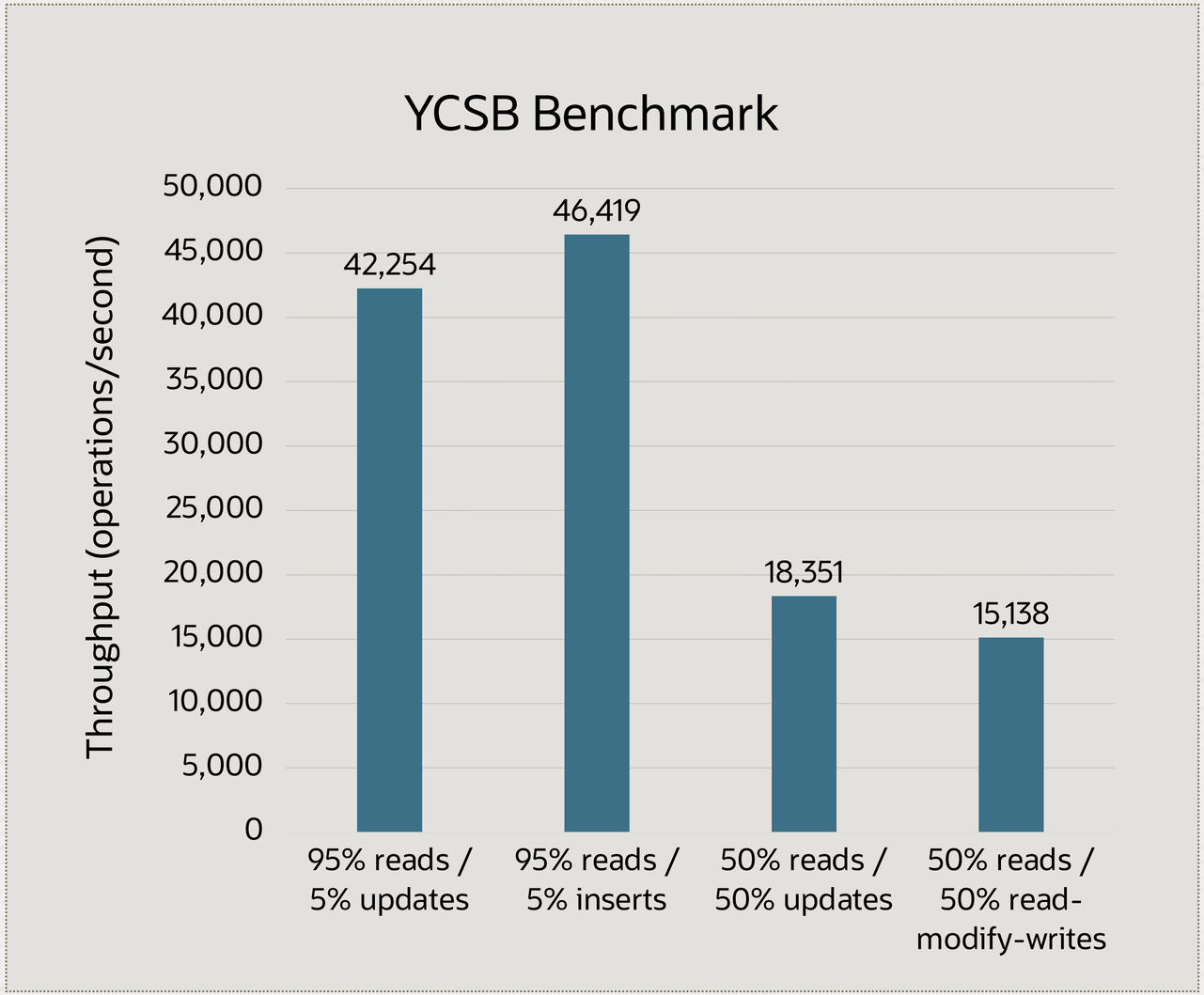
```
CREATE TABLE orders VALUES (
    oid          NUMBER,
    created      TIMESTAMP,
    status       VARCHAR2(10),
);


{
  "oid":123,
  "created":"2020-06-04T12:24:29Z",
  "status":"OPEN"
}
```

Why OSON? - Performance

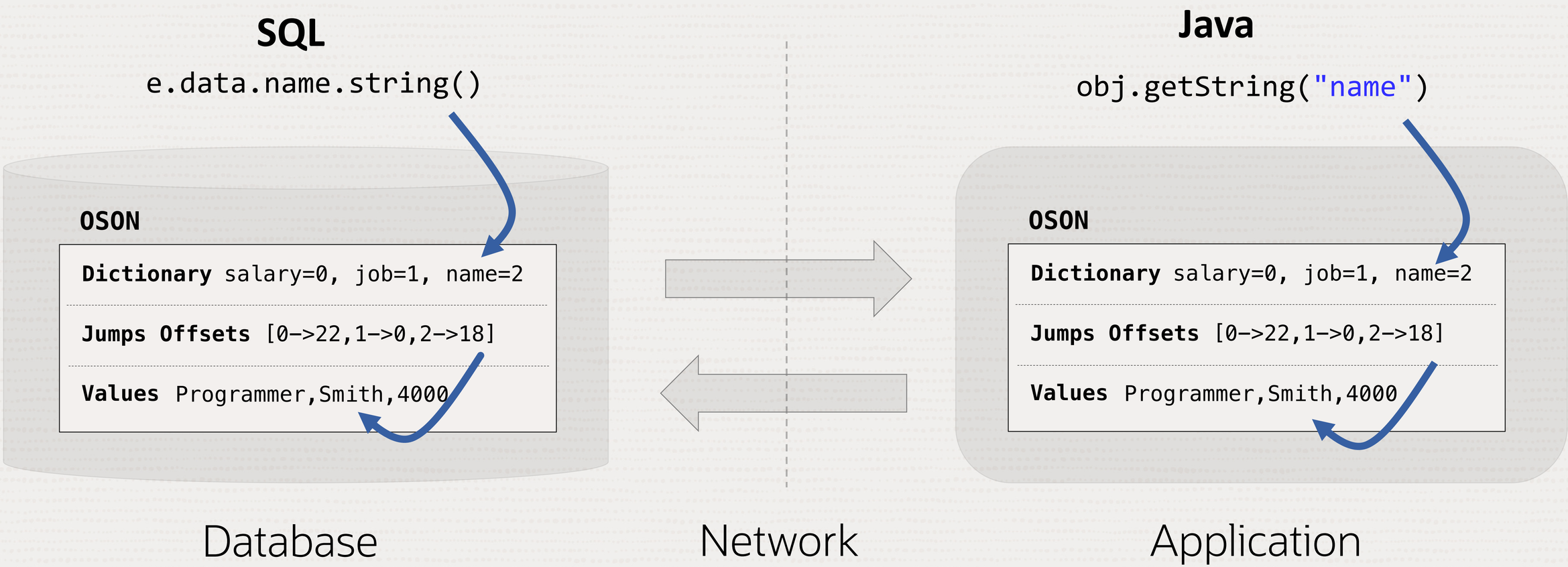
- Faster path evaluation
- Efficient random access
- Smaller than JSON text and BSON
- Less network and storage IO
- No text parsing or serialization

See VLDB 2020:
Native JSON Datatype Support: Maturing SQL and NoSQL convergence in Oracle Database



OSON Performance

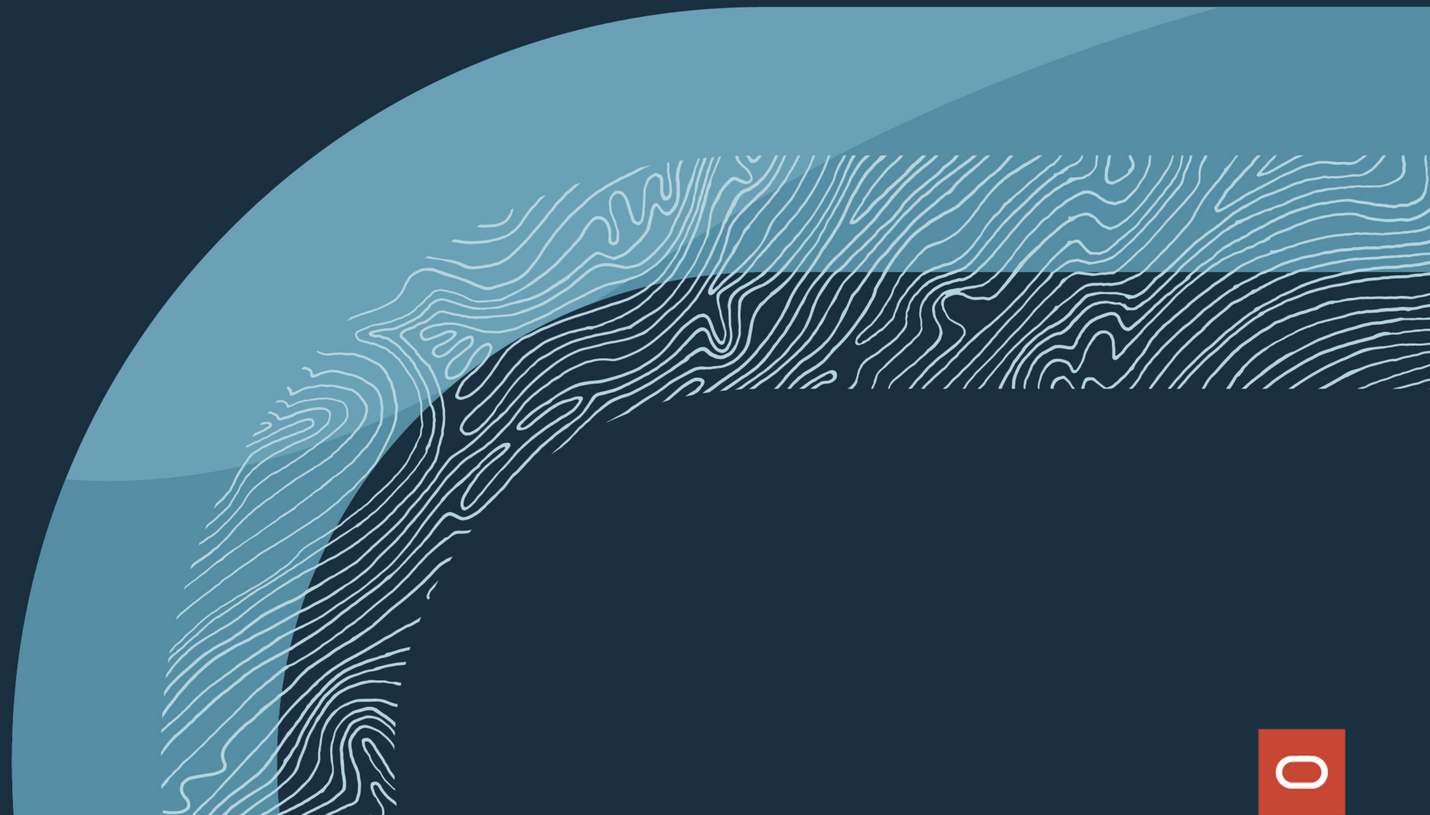
```
{"name":"Smith","job":"Programmer","salary":40000}
```



JDBC/JSON

Modeling JSON in Java (`oracle.sql.json`)

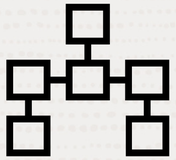
Binding and reading JSON from a statement



The Java API for JSON in JDBC (21c+)

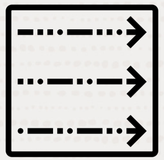
- Facilities to read, write, and modify binary JSON values from Java
- Features
  - Mutable tree/object model
  - Event-based parser and generator
  - Access to extended SQL/JSON types (TIMESTAMP, DATE, etc.)
  - Supports both JSON text and OSON
  - Optional integration with JSON-P / JSR-374
- Where is it?
  - **JAR:** ojdbc8.jar or ojdbc11.jar
    - OTN
    - Maven Central Repository
      (`groupId = com.oracle.database.jdbc, artifactId = ojdbc8`)
  - **Package:** `oracle.sql.json`

Package `oracle.sql.json`



Tree Model

```
OracleJsonObject
OracleJsonArray
OracleJsonString
OracleJsonDecimal
OracleJsonDouble
OracleJsonTimestamp
OracleJsonBinary
...
```



Event Model

```
OracleJsonParser
OracleJsonGenerator
```

Factory

```
OracleJsonFactory
```

OracleJsonFactory methods

Read/write **OSON**

```
createJsonBinaryGenerator(OutputStream)
createJsonBinaryValue(ByteBuffer)
createJsonBinaryValue(InputStream)
createJsonBinaryParser(ByteBuffer)
createJsonBinaryParser(InputStream)
```

Read/write **JSON text**

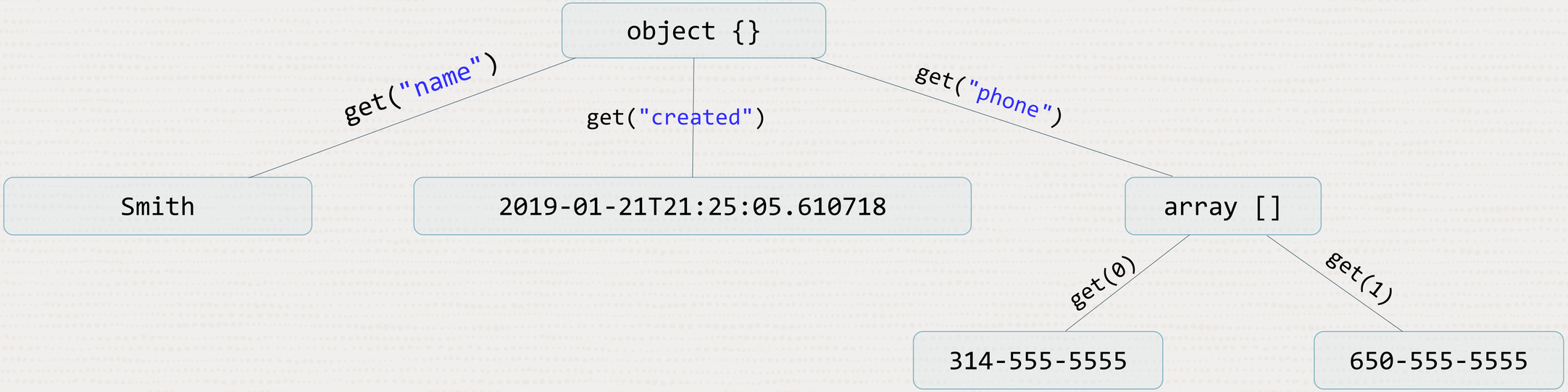
```
createJsonTextGenerator(OutputStream)
createJsonTextGenerator(Writer)
createJsonTextValue(InputStream)
createJsonTextValue(Reader)
createJsonTextParser(InputStream)
createJsonTextParser(Reader)
```

In-memory **tree model** creation

```
createObject()
createObject(OracleJsonObject)
createArray()
createArray(OracleJsonArray)
createString(String)
createDecimal(BigDecimal)
createDecimal(int)
createDecimal(long)
createDouble(double)
createTimestamp(Instant)
....
```

Tree Model (DOM)

```
{
  "name" : "Smith",
  "created" : "2019-01-21T21:25:05.610718",
  "phone" : ["314-555-5555","650-555-5555"]
}
```



Example – Creating a value

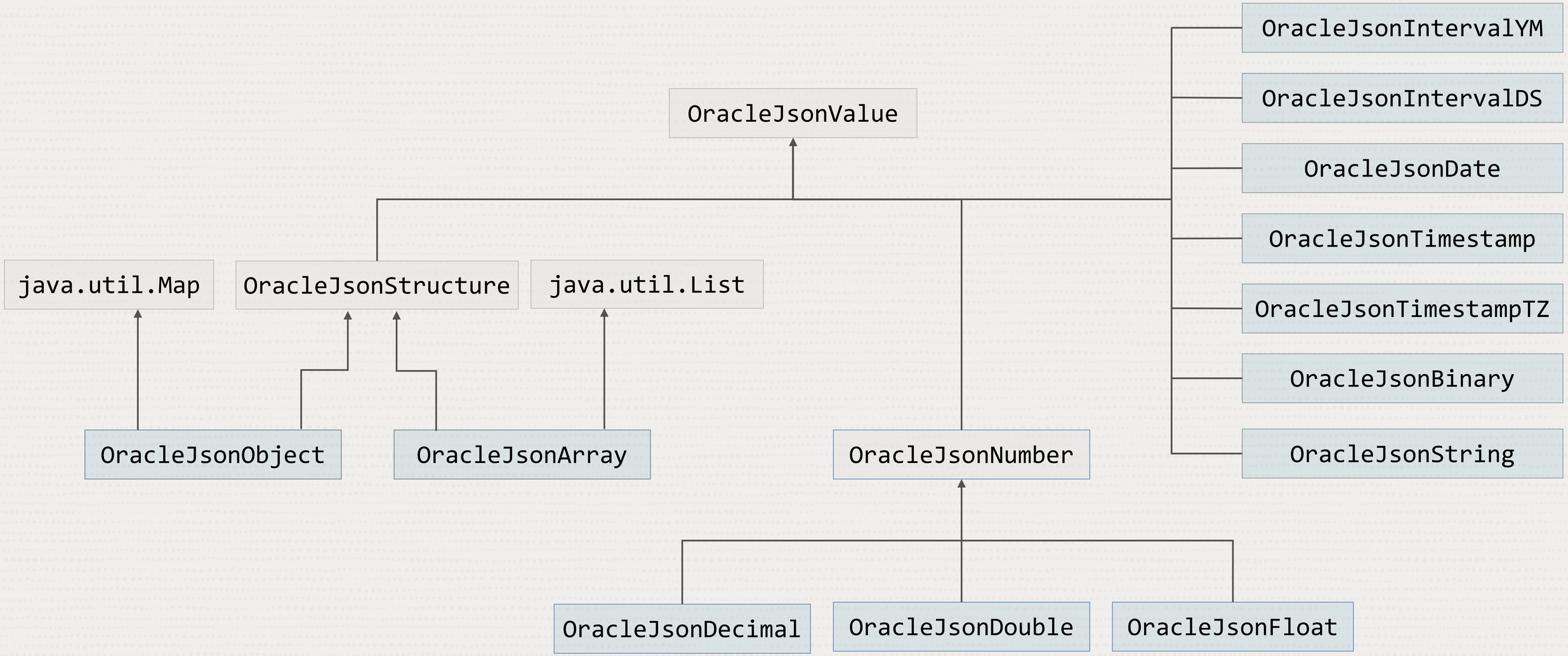
```
OracleJsonFactory factory = new OracleJsonFactory();

OracleJsonObject emp = factory.createObject();
emp.put("name", "Smith");
emp.put("salary", 40000);
emp.put("created", Instant.now());

System.out.println(emp.get("name"));
System.out.println(emp.toString());
```

```
Smith
{"name":"Smith","salary":40000,"created":"2019-01-21T21:25:05.610718"}
```

Tree Model – Type Hierarchy



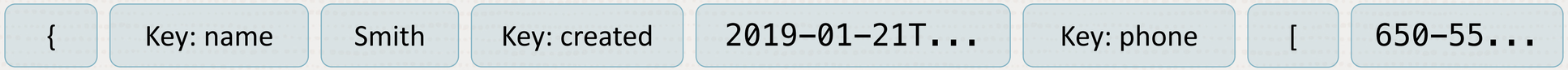
Tree Model Sources

`OracleJsonObject` and **`OracleJsonArray`** support updates when not directly created from OSON.

DESCRIPTION	SOURCE	READ-ONLY?
From the database	`ResultSet.getObject()`	Yes
Created from OSON	`createJsonBinaryValue(ByteBuffer)` `createJsonBinaryValue(InputStream)`	Yes
Created from JSON text	`createJsonTextValue(Reader)` `createJsonTextValue(InputStream)`	No
Created programmatically	`createObject()` `createArray()`	No
Copied	`createObject(OracleJsonObject)` `createArray(OracleJsonArray)`	No

Event Model

```
{
  "name" : "Smith",
  "created" : "2019-01-21T21:25:05.610718",
  "phone" : ["314-555-5555","650-555-5555"]
}
```



`OracleJsonParser`

- Produces a sequence of events
- `parser.next()`
- Events can come from text or OSON

`OracleJsonGenerator`

- Consumes a sequence of events
- `generator.write(...)`
- Generates text or OSON

SQL Integration

- **SQL** statements can **consume** and **produce** JSON values
- Relies on standard `getObject()` and `setObject()` methods throughout JDBC

Getting JSON from the Database - getObject()

SQL Methods that return JSON

`java.sql.ResultSet`
`  .getObject(*, Class<T>)`
 	
`java.sql.CallableStatement`
`  .getObject(*, Class<T>)`

Class	Content
`java.lang.String, java.io.Reader` `java.io.InputStream`	JSON text
`oracle.sql.json.OracleJsonValue` `jakarta.json.JsonValue`	Tree Model
`oracle.sql.json.OracleJsonParser` `jakarta.json.stream.JsonParser`	Event Model
`oracle.sql.json.OracleJsonDatum`	Raw OSON

```
ResultSet rs = stmt.executeQuery("SELECT data FROM emp");
rs.next();
OracleJsonObject smith = rs.getObject(1, OracleJsonObject.class);
```

Sending JSON to the Database

SQL methods accept JSON
java.sql.PreparedStatement **.setObject**(*, Object, SQLType) java.sql.ResultSet **.updateObject**(*, Object, SQLType) java.sql.CallableStatement **.setObject**(*, Object, SQLType) java.sql.RowSet **.setObject**(*, Object, SQLType)

Class	Content
java.lang.CharSequence java.lang.String java.io.Reader	JSON text
java.io.InputStream, byte[]	UTF8 or OSON
oracle.sql.json.OracleJsonValue jakarta.json.JsonValue	Tree Model
oracle.sql.json.OracleJsonParser jakarta.json.stream.JsonParser	Event Model
oracle.sql.json.OracleJsonDatum	Raw OSON

```
OracleJsonObject emp = ...;
PreparedStatement pstmt = con.prepareStatement("INSERT INTO emp VALUES (:1)");
pstmt.setObject(1, emp, OracleType.JSON);
```

JSR-374 JSON-P

- JSON-P (JSON Processing) is a Java API to parse and generate JSON
- **`oracle.sql.json`** can loosely be thought of as an extension of JSON-P
- JSON-P 1.0 -> **`javax.json`**
- JSON-P 2.0 -> **`jakarta.json`**
- Has tree model and event model
- Differences from **`oracle.sql.json:`**
  - supports extended types
  - supports a mutable tree model
  - supports conversion to and from OSON
  - does not support builders/reader or the services API

JSON-P: Wrap and Unwrap

- Interfaces are different views of the same data
- Wrap and unwrap do not make a deep copy or incur a performance penalty (e.g. in-place reads supported for jakarta.json)
- jakarta.json instances produced by JDCB implement java.sql.Wrapper

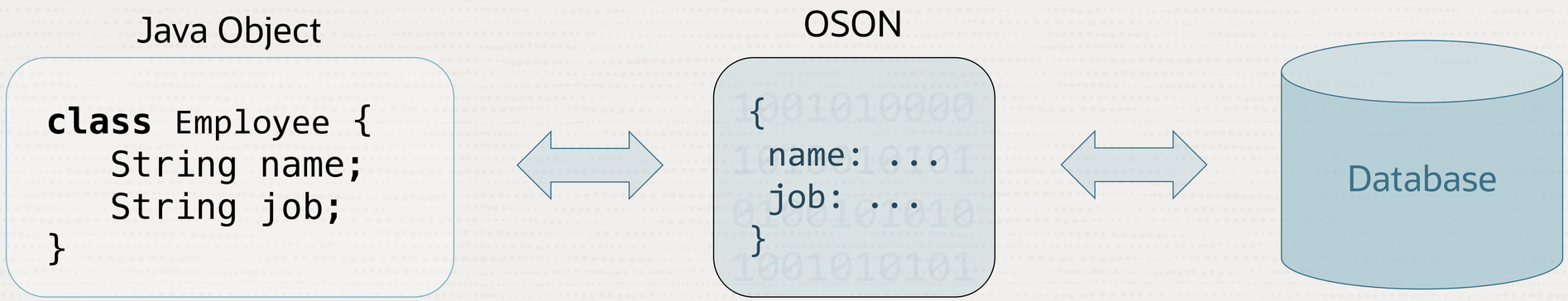
When to use JSON-P?

- Your application already implements **`jakarta.json`** interfaces and you want the database to consume them
- Your application uses a third-party library that implements **`jakarta.json`** and you need the database to consume them
- You want to keep your application independent of Oracle Database specific APIs and extended types
- You want to use JSON-B

```
ResultSet rs = stmt.executeQuery("SELECT data FROM emp ");
rs.next();
JsonObject smith = rs.getObject(1, jakarta.json.JsonObject.class);
```

JSR-367 JSON-B

- **JSON-B** (JSON Binding) convert Java objects to/from JSON
  - Default mapping between any Java object and JSON
  - Java Annotations to customize default mapping if necessary
- **Yasson** is the default implementation of JSON-B
- Use **Yasson** and **JDBC** to persist Java objects as binary JSON ***without parsing/serializing JSON text***
- Provides efficient ingest, retrieve, and query of application domain objects
- Complex objects with nested relationships can be retrieved without joins

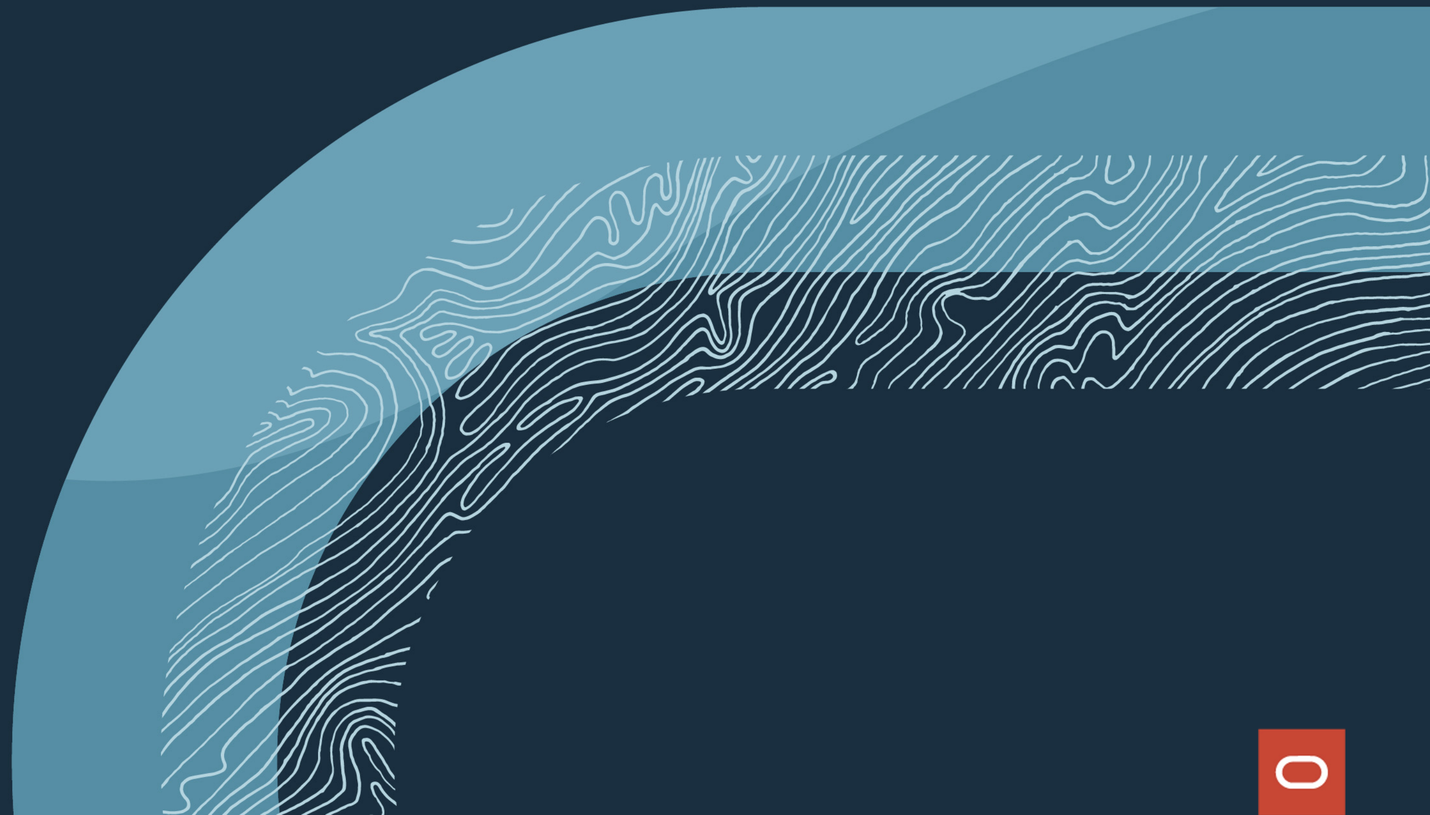


Demo

JSON and JDBC

https://bit.ly/3ViRzR8

JSON Collections



Oracle Database API for MongoDB

- **Data model:** JSON collections, not tables
- **Developers keep their skills** and continue to use MongoDB's tool, drivers, etc.
- **Easy migrations** of MongoDB workloads to Oracle
- Enable SQL
  - More and faster analytical capabilities, machine learning
  - Query JSON alongside other data models: relational, XML, spatial, etc.
  - Expose relational data, reports, query results as MongoDB collections

Relational Model

- Multiple "flat" tables that are *related*
- A table contains rows
- A schema contains tables
- Rows are structured (the 'S' in SQL)
- Data is accessed using SQL
- Related entities are joined

Normalized Tables

id	name	job
123	Anderson	Programmer
345	Smith	Agent

Accessed with SQL

```
select e.id, e.name
from employee e
where e.job = 'Programmer'
```

JSON Collections

A *document* is a JSON value

- Structure is flexible
- Have a unique *key* (_id)

A *collection* contains documents

- Supports insert, get, update, filter

A database contains collections

Access data programmatically –
No SQL required

MongoDB Java Example

```
MongoClient mongoClient = MongoClients.create(connString);
MongoDatabase database = mongoClient.getDatabase("admin");

MongoCollection<Document> coll =
  database.createCollection("employees");

Document po = Document.parse(json);
coll.insertOne(po);

Bson filter = eq("job", "Programmer");
MongoCursor<Document> cursor = coll.find(filter).cursor();
Document doc = cursor.next();
```

JSON Collections

Database => Schema

Collections created in database "admin" will be in the "ADMIN" schema

MongoDB Java Example

```
MongoClient mongoClient = MongoClients.create(connString);
MongoDatabase database = mongoClient.getDatabase("admin");

MongoCollection<Document> coll =
  database.createCollection("employees");

Document po = Document.parse(json);
coll.insertOne(po);

Bson filter = eq("job", "Programmer");
MongoCursor<Document> cursor = coll.find(filter).cursor();
Document doc = cursor.next();
```

JSON Collections

Collection => Table

Collections are an abstraction or view of a table with a single JSON column.

```
CREATE TABLE employee
(
    ID VARCHAR2,
    DATA JSON
)
```

MongoDB Java Example

```
MongoClient mongoClient = MongoClients.create(connString);
MongoDatabase database = mongoClient.getDatabase("admin");

MongoCollection<Document> coll =
  database.createCollection("employees");

Document po = Document.parse(json);
coll.insertOne(po);

Bson filter = eq("job", "Programmer");
MongoCursor<Document> cursor = coll.find(filter).cursor();
Document doc = cursor.next();
```

JSON Collections

Document => Row

Inserting a document into a collection inserts a row into the backing table.

```
INSERT INTO
  employees (data)
  VALUES (:1);
```

MongoDB Java Example

```
MongoClient mongoClient = MongoClients.create(connString);
MongoDatabase database = mongoClient.getDatabase("admin");

MongoCollection<Document> coll =
  database.createCollection("employees");

Document po = Document.parse(json);
coll.insertOne(po);

Bson filter = eq("job", "Programmer");
MongoCursor<Document> cursor = coll.find(filter).cursor();
Document doc = cursor.next();
```

JSON Collections

Filter => Query

Filter expressions are executed as SQL over the backing table. Fully utilizes core Oracle Database features such as indexing, cost-based optimization, etc.

```
SELECT data
FROM employee e
WHERE e.data.job = 'Programmer'
```

MongoDB Java Example

```
MongoClient mongoClient = MongoClients.create(connString);
MongoDatabase database = mongoClient.getDatabase("admin");

MongoCollection<Document> coll =
  database.createCollection("employees");

Document po = Document.parse(json);
coll.insertOne(po);

Bson filter = eq("job", "Programmer");
MongoCursor<Document> cursor = coll.find(filter).cursor();
Document doc = cursor.next();
```

SQL – but only when you need it…

JSON Collections

```
employee.insertOne({
   "name" : "Bond",
   "job"  : "Agent"
});
```

Simple, flexible persistence for applications, microservices

employee

```
{
  {
    {
      "name" : "John"
      "job" : "Developer"
    }
  }
}
```

SQL/JSON

```
SELECT t.data.name.string(),
FROM employee t
WHERE t.data.job.string()
      = 'Agent';
```

Powerful analytics and reporting directly over collections

JSON Collections APIs from Java

Oracle API for MongoDB

- Use MongoDB tools and drivers (`mongo-java-driver`)
- Available automatically in ADB-S 19c or later
- Runs in ORDS standalone for other databases 21c or later

SODA Java

- A Java library that wraps JDBC, no SQL required
- Similar to the Mongo Java API
- Supports earlier versions of the database (12g+) and JSON text storage
- Supports `oracle.sql.json`

SODA REST

- Runs inside ORDS
- Exposes documents as JSON text

Mongo API Main Differences and Limitations

- MongoAPI uses Oracle database authentication and authorization
  - use Oracle database username/password only

- Authenticate as the user corresponding to the database being accessed.
  - e.g. authenticate as ADMIN, use ADMIN schema/database

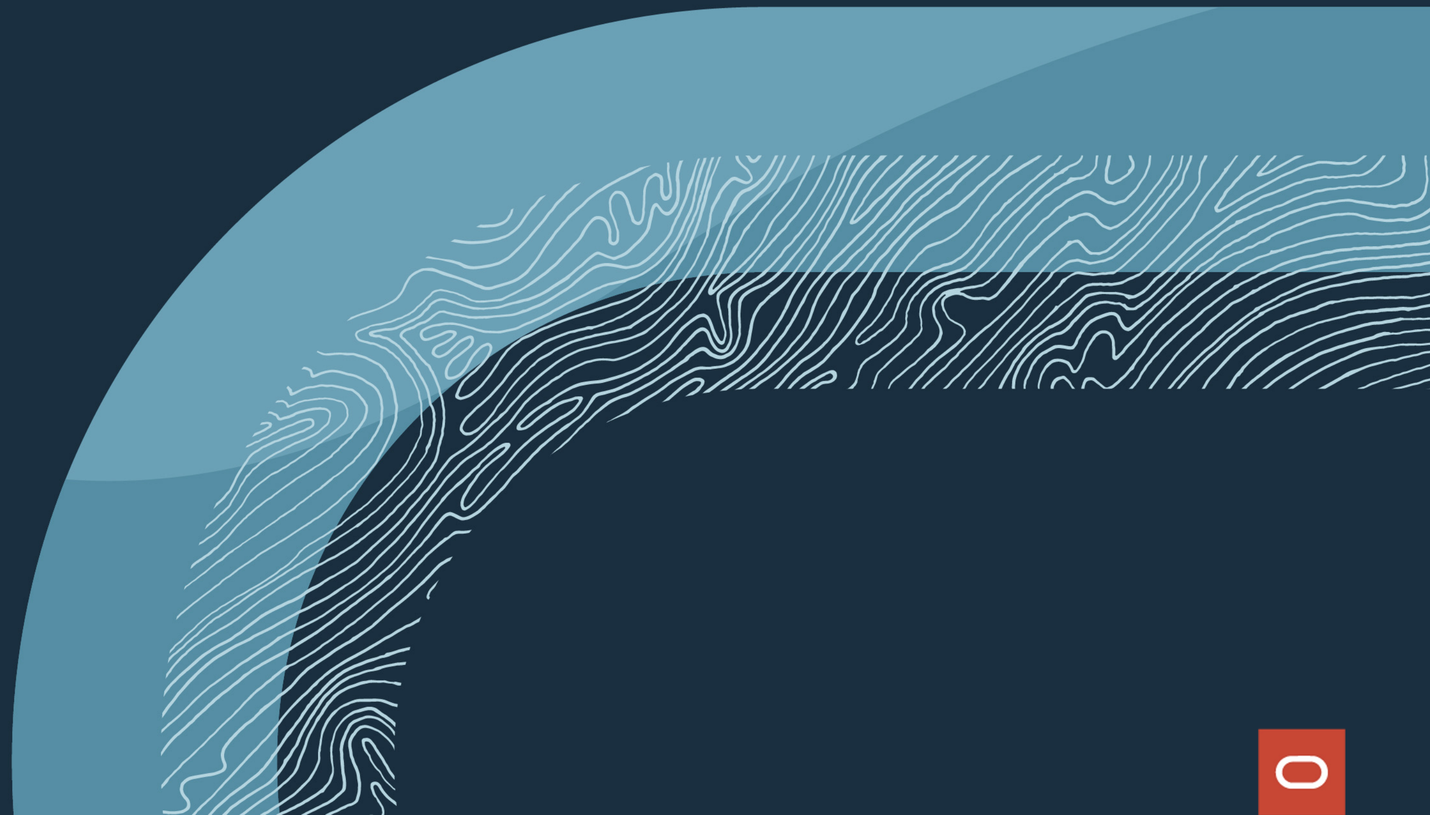
- Indexes must be created from SQL
  - db.createIndex() from MongoAPI does not create an index

```
SQL> create index lastidx on
        employee (json_value(data, '$.job'  error on error));
```

- Aggregation pipeline is not supported

Demo

JSON collections



https://bit.ly/3CRFStL

Suggested Sessions

- A Stateful Spring Boot Microservice Using Spring Data, Hibernate, and JSON
  LRN2797
  Wednesday, Oct 19 5:00 PM - 5:45 PM PDT, Summit 227, Caesars Forum
- NYSE Shares Best Practices for Developing with JSON in Oracle Database
  LRN3785
  Thursday, Oct 20 11:30 AM - 12:15 PM PDT, Forum 127, Caesars Forum
- Using the Database API for MongoDB
  HOL4079
  Thursday, Oct 20 9:00 AM - 10:30 AM PDT, Bellini 2003, The Venetian, Level 2
- Developing with JSON and SODA in Oracle Database
  HOL3994
  Thursday, Oct 20 1:00 PM - 2:30 PM PDT, Forum 128, Caesars Forum

References

- Examples using JSON type and JDBC (oracle.sql.json)
  https://github.com/oracle/json-in-db/tree/master/JdbcExamples
- oracle.sql.json Javadocs
  https://javadoc.io/static/com.oracle.database.jdbc/ojdbc8/21.4.0.0/oracle/sql/json/package-summary.html
- Example using the Oracle API for MongoDB from Java
  https://github.com/oracle/json-in-db/tree/master/MongoExamples
- Examples using Spring Boot and the MongoDB API for Java
  https://github.com/oracle/json-in-db/tree/master/MongoExamples/SpringBoot
- Examples using SODA Java
  https://github.com/oracle/json-in-db/tree/master/SodaExamples
- Autonomous JSON Database
  https://docs.oracle.com/en/cloud/paas/autonomous-json-database/index.html

ORACLE
CloudWorld

Thank you

Join us on Slack!

After joining **`oracledevrel`**,
find us in the channel
`#oracle-db-json`

https://join.slack.com/t/oracledevrel/shared_invite/zt-1h0fhz7f8-gHM298rFasYzlTn0Nii2sQ