AOUG Frühstück
JSON in der Datenbank

Calvin Schmidt
Georg Schmidt
Technology Engineers
5.Nov 2024

Agenda: JSON in der Datenbank

- Frühstück und Begrüßung
- Überblick 23AI für Entwickler
- JSON in der Oracle Datenbank
- Mongo DB API
- JSON Relational Duality Views
- Praktische Beispiele
- Erfahrungsaustausch

Oracle Database Vision

With Generative AI (LLM)

Make modern apps and analytics
easy to **generate** and run
for all use cases at any scale

Oracle DB | History

1977
Software Development Laboratories (SDL)
Santa Clara, California

1979
SDL becomes **Relational Software Systems, Inc.**
- First commercial RDBMS

1981
C Programming Language chosen for v3

1982
Relational Software Systems, Inc. becomes **Oracle**

V3
Portability at its best
- Atomic execution of SQL statements
- Nonblocking queries
- Mapping of data in pre-join formats

V4
- Read Consistency
- Export/Import utility
- Early Reporting capabilities

V5
- Client/Server mode
- Clustered Configuration (VAX Clusters)
- Security and Audit Features

V6
- Row level locking
- Hot backup
- First version of PL/SQL
- High Speed OLTP

V6.2
- OPS for VAX Cluster
- Foundations for HA & Scalability
- Oracle Card
- 1-2-3 Data Lens

V7
- PL/SQL Stored procedures
- Triggers
- Distributed 2-phase commit
- Shared cursors
- Cost based optimizer

V5.1
- Distributed Queries (single query access to data stored in multiple locations)
- SQL * Forms is released
- SQL * Plus is released

Oracle becomes a publicly traded company

V8
- Recovery Manager
- Partitioning
- Expanded support for Java

V8.1
- Native Internet Protocols
- Virtual Private Database
- Standby Database foundation for Data Guard

V9.1
- Real Application Clusters (RAC)
- Integrated Data Mining with OLAP
- Data Guard Broker

10g
First Database designed for enterprise grid computing

Foundation for:
- Grid Infrastructure, ASM
- Flashback Database

10gR2
Foundation for:
- Real Application Testing
- Database Vault
- Advanced Compression & TDE
- Data Guard Fast-Start Failover

11g
Foundation for:
- Active Data Guard
- Secure Files

11gR2
Foundation for:
- EBR, HCC, Data Redaction
- ACFS
- Golden Gate Replication

12c
- Multitenant Architecture
- In-Memory Column Store
- Native JSON
- SQL Pattern Matching

12c R2
- Sharding
- Snapshot-based replication
- PDB Hot Clone
- Enhanced Partitioning

18g
- Autonomous Database announced
- User-defined sharding
- PDB Snapshot carousel
- DB In-memory support for External tables

19c
Generally Available since April 2019
- Automatic Indexing
- Hybrid Partition Tables
- Active Data Guard DML Redirect

21c
Innovation Release
- Per-PDB Data Guard
- Native JSON Type
- SQL Macros
- JavaScript Stored Logic

23ai
- AI Vector Search
- JSON Duality
- Graph
- TrueCache
- Global Distributed Database

1977 1979 1981 1982 1983 1984 1985 1986 1988 1991 1992 1997 1998 2001 2003 2005 2007 2009 2013 2016 2018 2019 2020 2023 2024

Oracle Database 23ai: AI Made Simple for Enterprise Data

Over 50 SQL simplifications

Data Use Case Domains

ORACLE Database 23ai

Read-Only Per-PDB Standby

Property Graphs

Real-time SQL Plan Management

Lock-Free Reservations

Microservice Support

JSON / Relational Duality

AI Vector Search

True Cache

SQL Firewall

Priority Transactions

Globally Distributed Database

Rolling Patching

JavaScript Stored Procedures

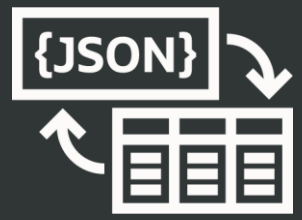
Developer Role

Lock-Free long-running transactions

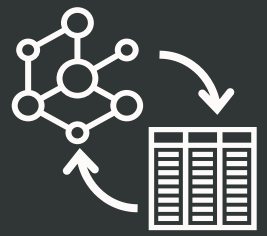
Next Generation Converged Database – Database 23ai

Unifies JSON Document, Graph, and AI with Relational Models

Unification of
JSON and Relational



Unification of
Graph and Relational



Unification of
AI and Databases

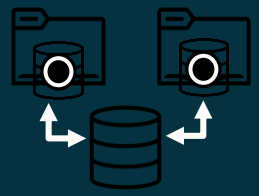
Copyright © 2024, Oracle and/or its affiliates

Oracle Database 23ai Mission Critical Apps Enhancements

Priority Transactions

Automatically prioritizes high-priority transactions over low-priority transactions

Low-priority transactions that block high-priority transactions will be automatically aborted



True Cache

A (nearly) disk-less Oracle database instance that is deployed as a cache

Unlike conventional mid-tier caches such as Redis, data in True Cache is automatically updated

ANY SQL Query can be transparently directed to the cache instead of the database

Active-Active Globally Distributed Database

Database sharding with Raft replication supports applications that require low latency and high availability plus helps meet data sovereignty requirements

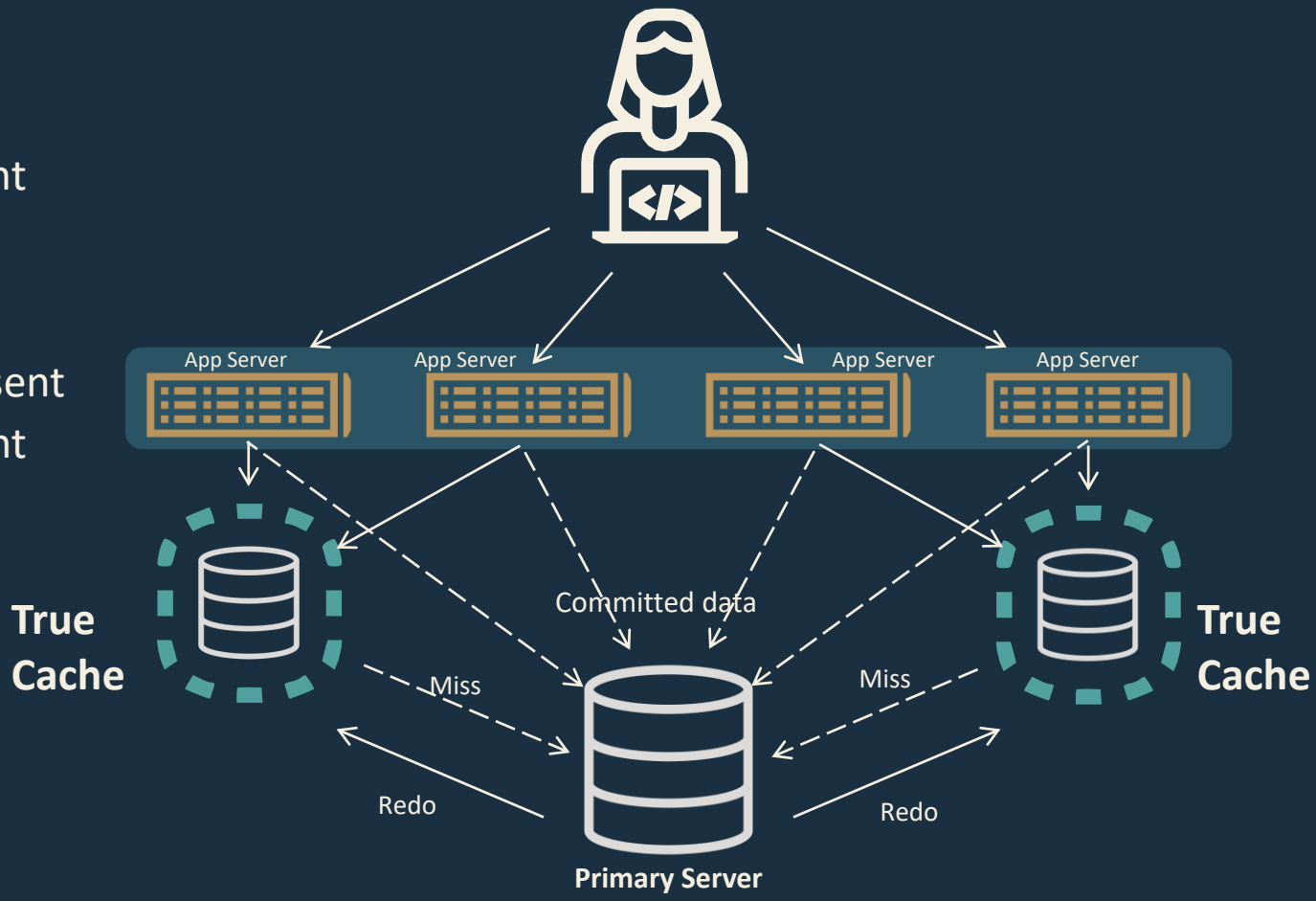
Readable Per-PDB Standby

Per-PDB standby databases can now be opened for read-only workloads

Improving production database performance by offloading resource-intensive backup and reporting operations to standby systems

Oracle Database 23ai True Cache

- Solid lines represent relatively frequent requests
- Dotted lines represent relatively infrequent requests



App connect to True Cache and perform SQL queries

True Cache is an in-memory, consistent, and automatically managed full SQL cache

Oracle Database 23ai Security Enhancements

In-Database Firewall

An easy-to-use firewall solution, with minimal perf and operational overhead

Built-in to ensure it cannot be bypassed

Protection against attacks by monitoring and blocking “unauthorized SQL” and SQL injection attacks

Read-Only Users

Users may be created as, or altered to, READ ONLY status (default READ WRITE)

```
ALTER USER joe
READ ONLY;
```

Read-only users can not insert or update data, nor can they create database objects

Developer Role

It’s complex to grant all the privileges developers need to create, debug, etc.

Now it’s simple using the new DB_DEVELOPER_ROLE :

```
GRANT DB_DEVELOPER_ROLE
TO    scott;
```

Schema Privileges

Managing the privileges on all the tables, views, and procedures used by an app can be tricky

Now this is simple using GRANT on a schema

```
GRANT SELECT ANY TABLE
   ON SCHEMA sales
   TO mary;
```

Oracle Database 23ai Manageability and Availability Enhancements

Shrink Tablespace

A simple way to reclaim unused or free space in a tablespace

Optimizes the storage of big file tablespaces by moving objects to the datafile head, and then resizing the datafile by removing the tail

Real-time SQL Plan Management

Automatically repairs SQL performance regressions

The optimizer detects a plan regression and tries to find a previous plan with better performance

If an alternative plan is found to perform better, a SQL plan baseline is automatically created and that plan will be used

Rolling Patching for Complex Changes

Enables non-rolling patches to be applied online in stages

Phase 1:
The patch is applied to all instances but not enabled

Phase 2:
The patch is enabled via a SQL command

Enhanced Error Messages & Logging

Improved error messages that provide useful problem diagnosis in context and suggest actionable solutions. Easily searchable error message portal.

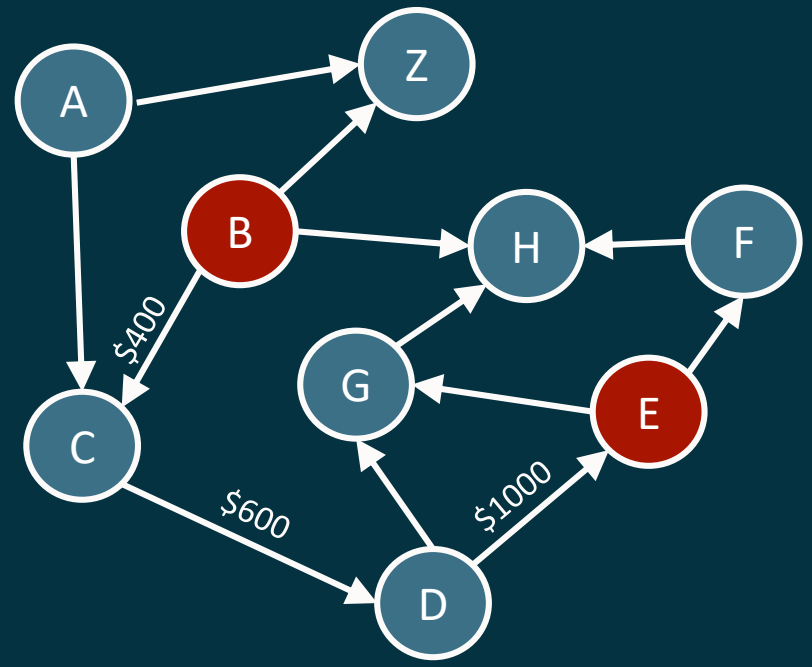
New attention log that highlights issues requiring prompt remediation

Property Graph Views in Oracle Database 23ai

Allow queries between connections and relationships in the data

For example, to discover indirect money movements from bank account 'B' to bank account 'E'

```
SELECT graph.path
FROM   GRAPH_TABLE (
       bank_graph
        MATCH (v1)-[e is BANK_TRANSFERS]->{1,3} (v2)
         WHERE  v1.id = 'B'
         AND    v2.id = 'E'
        COLUMNS LISTAGG(e.to_acc, ',') AS path)
       ) graph
;
```



Oracle Database 23ai – Additional Features For App Dev

Boolean Datatype

A more intuitive way of storing and manipulating logical values within the database

```
CREATE TABLE customers(
        cust_id number,
        Active boolean);

SELECT cust_id
FROM customers
WHERE active;
```

JavaScript Stored Procedures

JavaScript joins PL/SQL & Java as first-class server-side dev languages

Executed by our fast Multilingual Engine (MLE), powered by GraalVM

Reduces the number of roundtrips to the database

Wider Tables

Support for up to 4096 columns per table

Simplifies development of applications that need large numbers of attributes such as for ML and IoT workloads

```
ALTER SYSTEM SET
max_columns = EXTENDED;
```

Lock-free Column Value Reservations

Allows applications to reserve part of a value in a column without locking the row

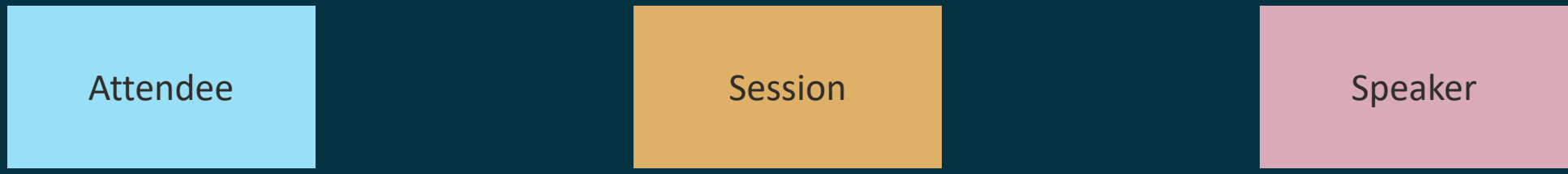
For example, reserve part of a bank account balance or reserve an item in inventory without locking out all other operations on the bank account or item

Carl Olofson, Research VP,
Data Management Software, IDC, says:

”Oracle’s JSON Relational Duality, a truly revolutionary solution, is perhaps one of the most important innovations in information science in 20 years.”

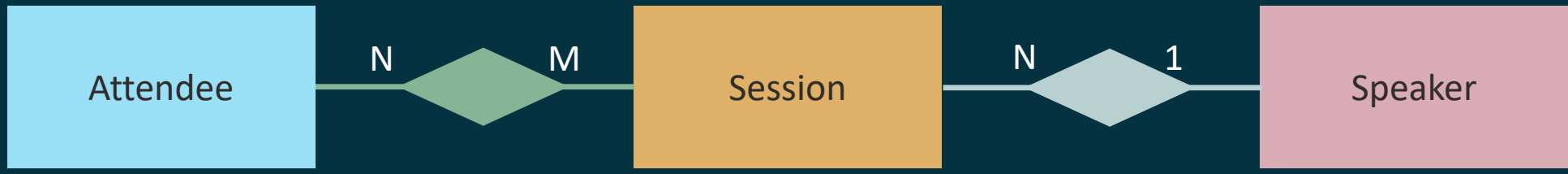
Simple Example: Conference

Entities



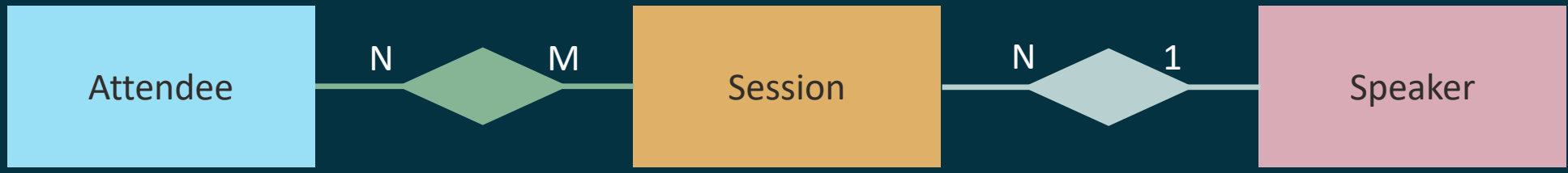
Simple Example: Conference

Relationships, Cardinalities



Simple Example: Conference, RELATIONAL

Relationships, Cardinalities



Tables

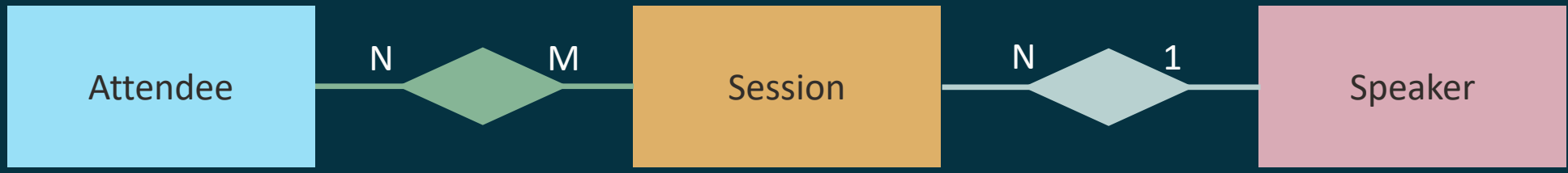
ATTENDEE	
AID	**NAME**
A1	Jill
A2	Sanjay

ATT_SES_MAP	
AID	**SID**
A1	S1
A2	S2

SESSION			
SID	**NAME**	**ROOM**	**SPID**
S1	JSON	OSLO	SP1
S2	SQL	TOKYO	SP2

SPEAKER		
SPID	**NAME**	**PHONE**
SP1	Carla	650..
SP2	Pascal	408...

Simple Example: Conference, RELATIONAL



References, Links → used for Joins

ATTENDEE

AID	NAME
A1	Jill
A2	Sanjay

ATT_SES_MAP

AID	SID
A1	S1
A2	S2

SESSION

SID	NAME	ROOM	SPID
S1	JSON	OSLO	SP1
S2	SQL	TOKYO	SP2

SPEAKER

SPID	NAME	PHONE
SP1	Carla	650..
SP2	Pascal	408...

Relational: the GOOD

- ⊕ No data duplication, consistency is guaranteed
- ⊕ **Use case flexibility**
  - Examples: session catalog, speaker/attendee/room schedules,...
  - access any number of tables (with joins)
  - select only those column values that are needed
- ⊕ declarative language to express operations: SQL
  - many ways to improve access performance (indexes, in-memory, ...)
  - optimizer picks the best execution plan

ATTENDEE	
AID	**NAME**
A1	Jill
A2	Sanjay

ATT_SES_MAP	
AID	**SID**
A1	S1
A2	S2

SESSION			
SID	**NAME**	**ROOM**	**SPID**
S1	JSON	OSLO	SP1
S2	SQL	TOKYO	SP2

SPEAKER		
SPID	**NAME**	**PHONE**
SP1	Carla	650..
SP2	Pascal	408...

Relational: the BAD

- ⊖ Requires upfront schema definition
  - tables, columns, data types
  - "schema first, data later"
  - harder to evolve: **not schema-flexible**
- ⊖ 'Normalization' breaks business objects into many tables
- ⊖ SQL is usually not integrated into the programming language:
  - SQL is a string (sometimes generated by an ORM tool)

ATTENDEE	
AID	**NAME**
A1	Jill
A2	Sanjay

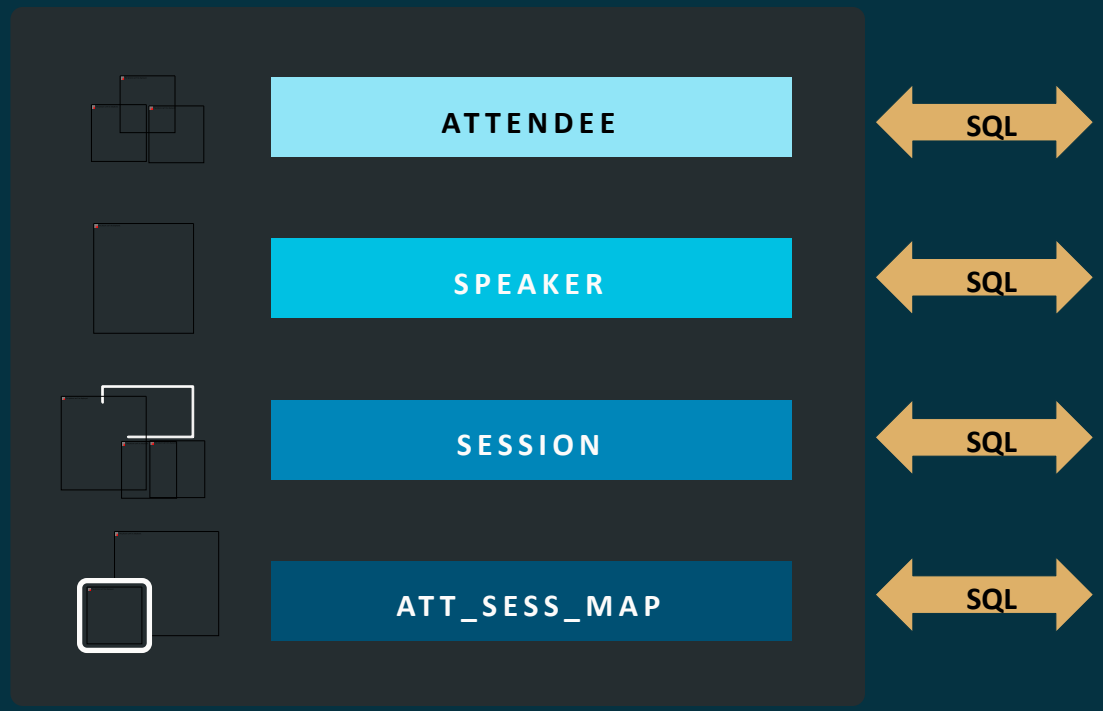
ATT_SES_MAP	
AID	**SID**
A1	S1
A2	S2

SESSION			
SID	**NAME**	**ROOM**	**SPID**
S1	JSON	OSLO	SP1
S2	SQL	TOKYO	SP2

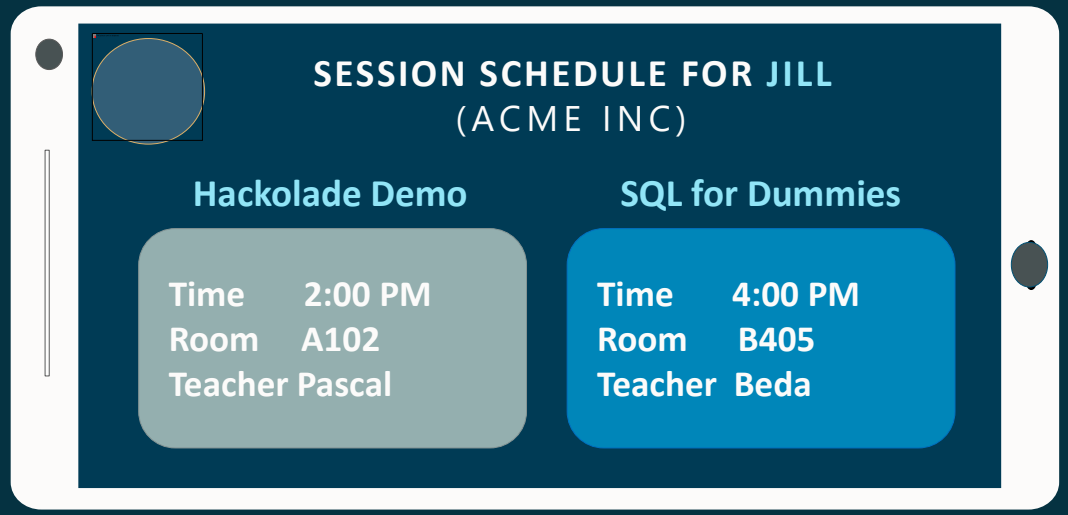
SPEAKER		
SPID	**NAME**	**PHONE**
SP1	Carla	650..
SP2	Pascal	408...

App Dev Example — Conference Schedule

Developing apps using normalized tables is very flexible, but it is not always easy for developers.

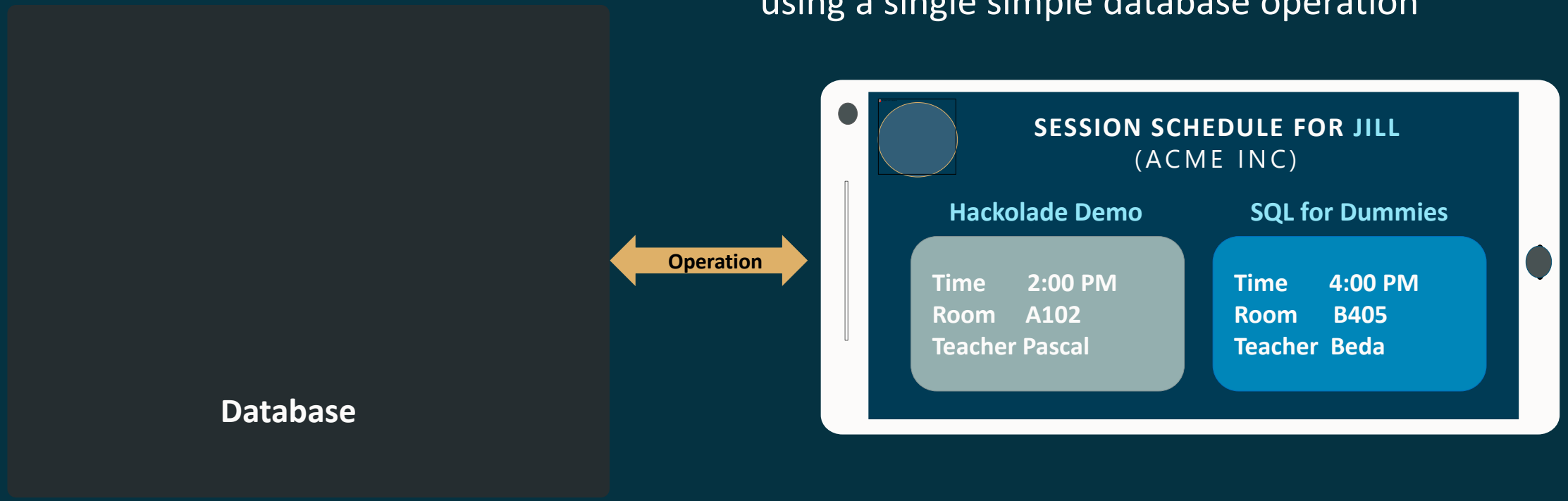


To build Jill's schedule, the developer must run database operations on each of the four tables.

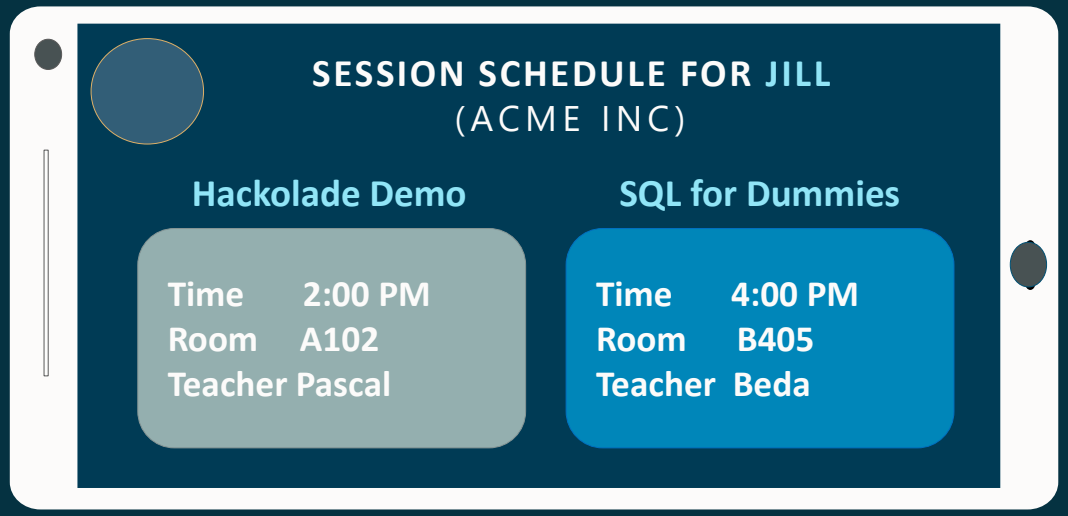


Relational Data and Developers

Ideally, the developer wants to build Jill's schedule using a single simple database operation



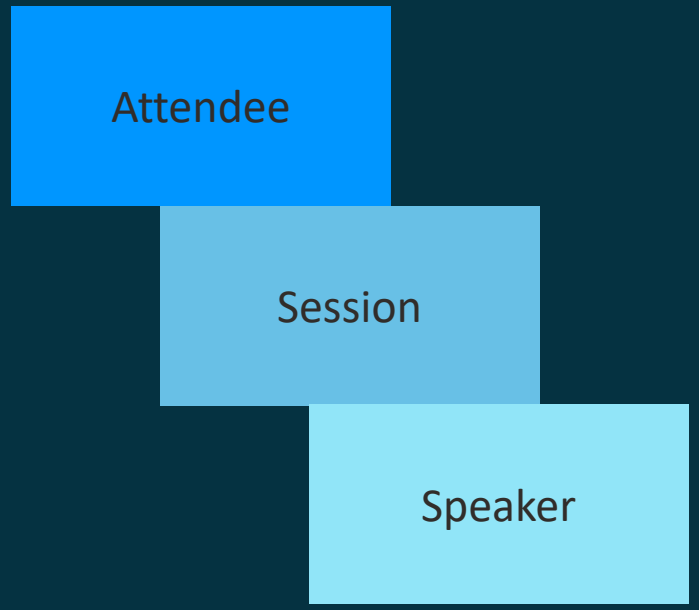
JSON: Attendee Schedule




```
{
 "_id"          : "3245",
 "name"         : "Jill",
 "company"      : "ACME Inc",
 "schedule"     : [
     {
        "code"    : "DB12",
        "session" : "SQL",
        "time"    : "14:00",
        "room"    : "A102",
        "speaker" : "Adam"
     },
     {
        "code"    : "CODE3",
        "session" : "NodeJs",
        "time"    : "16:00",
        "room"    : "R12",
        "speaker" : "Claudia"
     }
   ]
}
```

JSON: Attendee Schedule

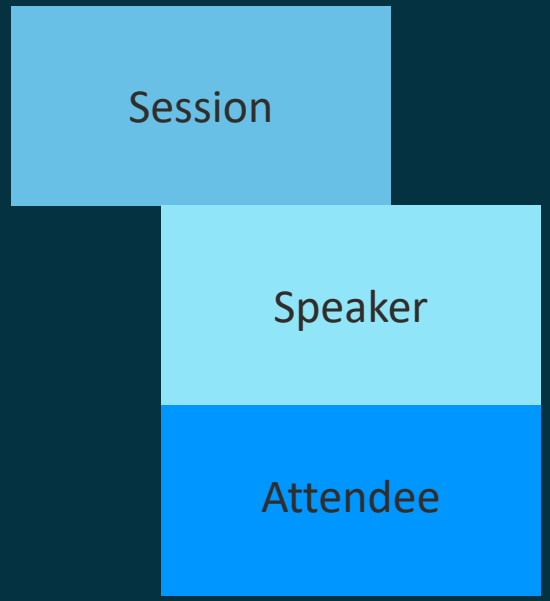
Attendee hierarchy




```
{
 "_id"          : "3245",
 "name"         : "Jill",
 "company"      : "ACME Inc",
 "schedule"     : [
     {
        "code"    : "DB12",
        "session" : "SQL",
        "time"    : "14:00",
        "room"    : "A102",
        "speaker" : "Adam"
      },
      {
        "code"    : "CODE3",
        "session" : "NodeJs",
        "time"    : "16:00",
        "room"    : "R12",
        "speaker" : "Claudia"
      }
   ]
}
```

JSON: Session Catalog

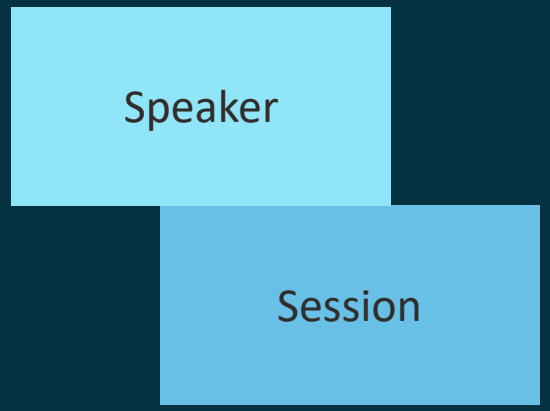
Session hierarchy




```
{
   "code"    : "DB12",
   "name".   : "SQL",
   "time"    : "14:00",
   "room"    : "A102",
   "speaker" : "Adam",
   "numAtt"  : 12,
   "roomCap" : 60
}

{
   "code"    : "CODE2",
   "name".   : "NodeJS",
   "time"    : "16:00",
   "room"    : "R12",
   "speaker" : "Claudia",
   "numAtt"  : 75,
   "roomCap" : 75
}
```

JSON: Speaker Schedule

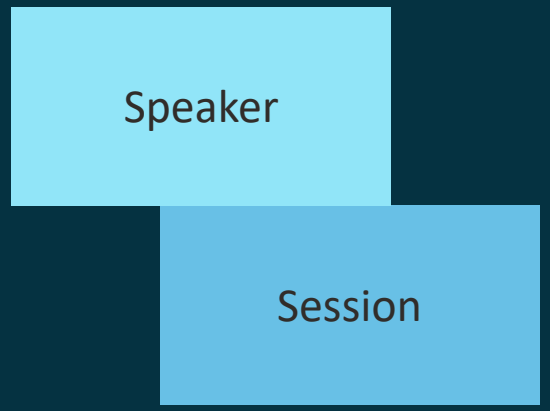
Speaker hierarchy



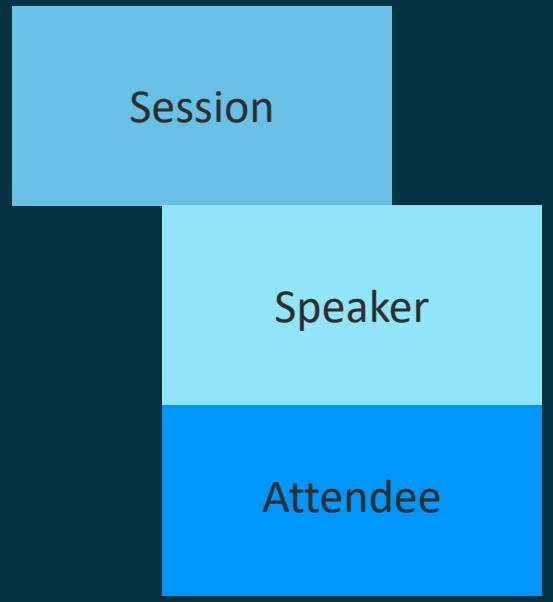

```
{
   "speakerId" : "S1",
   "name".     : "Adam",
   "phone"     : "650-392-000",
}

{
   "speakerId" : "S2",
   "name".     : "Claudia",
   "phone"     : "+49 871 393",
}
```

JSON: too many hierarchies

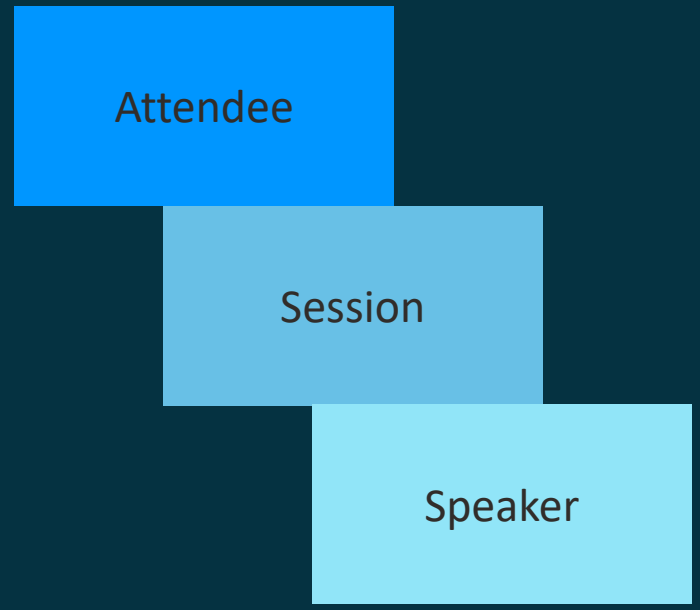
Speaker hierarchy



Session hierarchy



Attendee hierarchy



JSON: the GOOD, the BAD

⊕ JSON Object contains all information for one use case

- No joins needed

⊕ Object usually retrieved by a simple 'get' operation (e.g. REST, document API)

- No SQL strings in the programming language

⊕ JSON is schema-flexible: "data first, schema later"

⊖ Single hierarchy is only possible for few simple use cases!

⊖ Embedding of the same values causes duplication!

⊖ Much harder to keep consistent and optimize

Initial simplicity for the developer causes long term complexities

Can't we normalize JSON the same way as tables?

Document Database Normalization

SCHEDULE FOR: JILL

```
{
 "attendee"     : "S3245",
 "name"         : "Jill",
 "schedule"     :
   [  {
        "time"    : "14:00",
        "session" : "SQL++",
        "room"    : "A102",
        "speaker" : "Adam"
      },
     …
   ]
}
```

```
{
 "attendee"     : "S3245",
 [  {"session" : "M201 }
     … ]
}
```

```
{
 "attendeeId" : "S3245",
 "name"       : "Jill",
 "company"    : "ACME inc"
…}
```

```
{
 "sessionId" : "M201",
 "name"      : "SQL++",
 "teacher"   : "T543"
…}
```

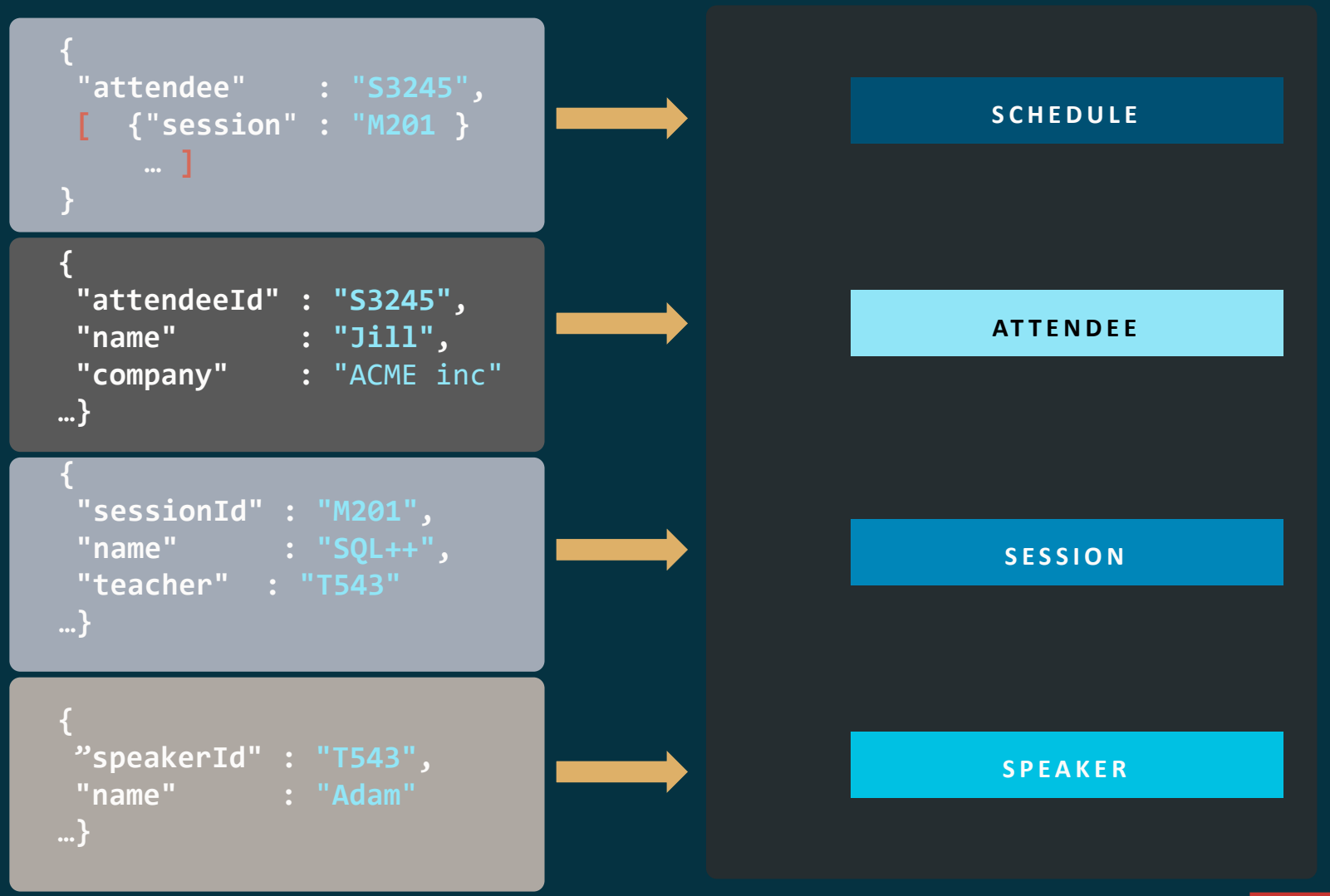
```
{
 ”speakerId" : "T543",
 "name"      : "Adam"
…}
```

When documents
are normalized
their simplicity is lost

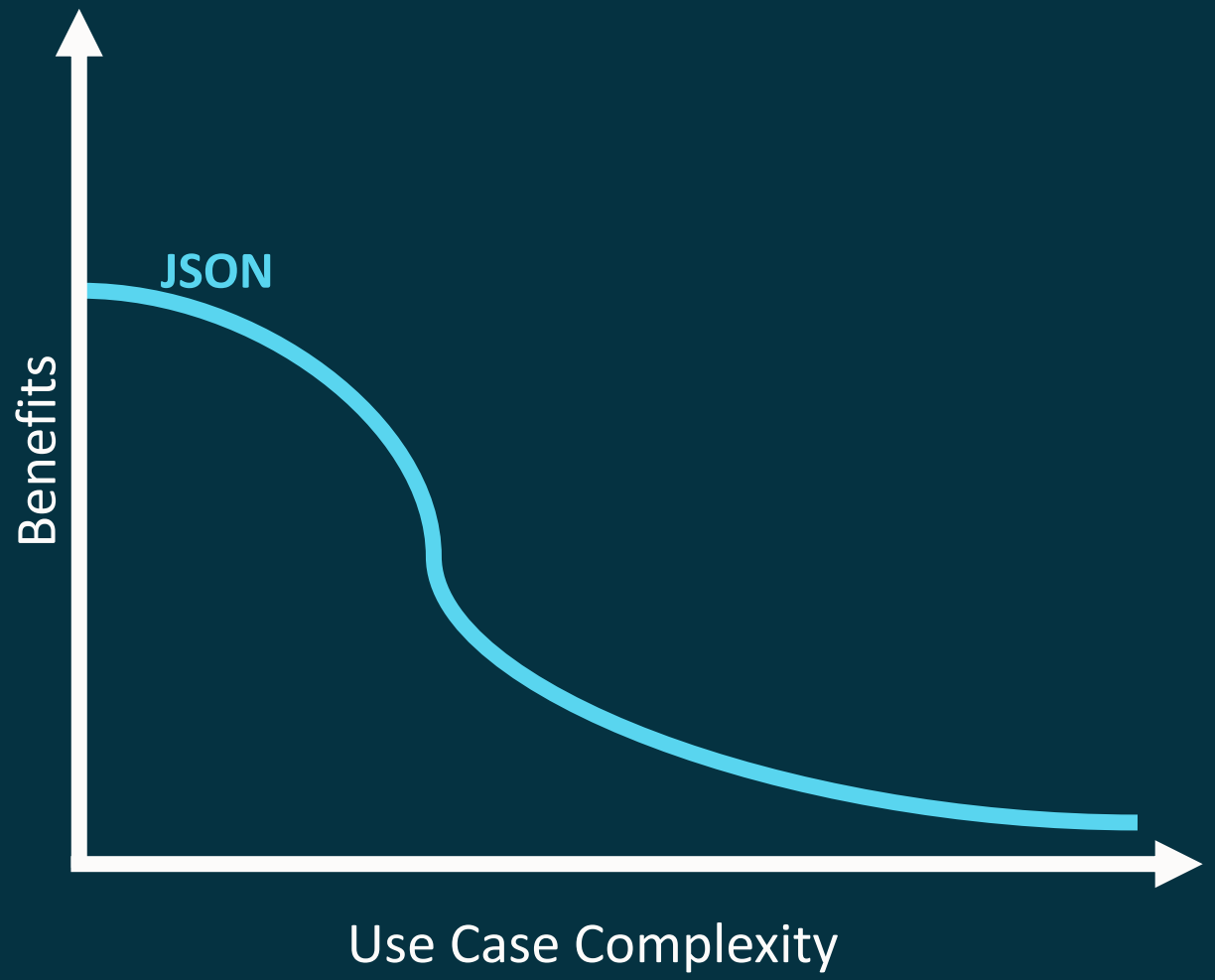
Document Database Fragmentation

Normalizing produces the worst of both worlds

- The structure now mirrors the normalized tables
- Without gaining the power of SQL and relational at the database level
- Referential integrity must be enforced by every app
- Performance suffers due to reference chasing and loss of shard locality

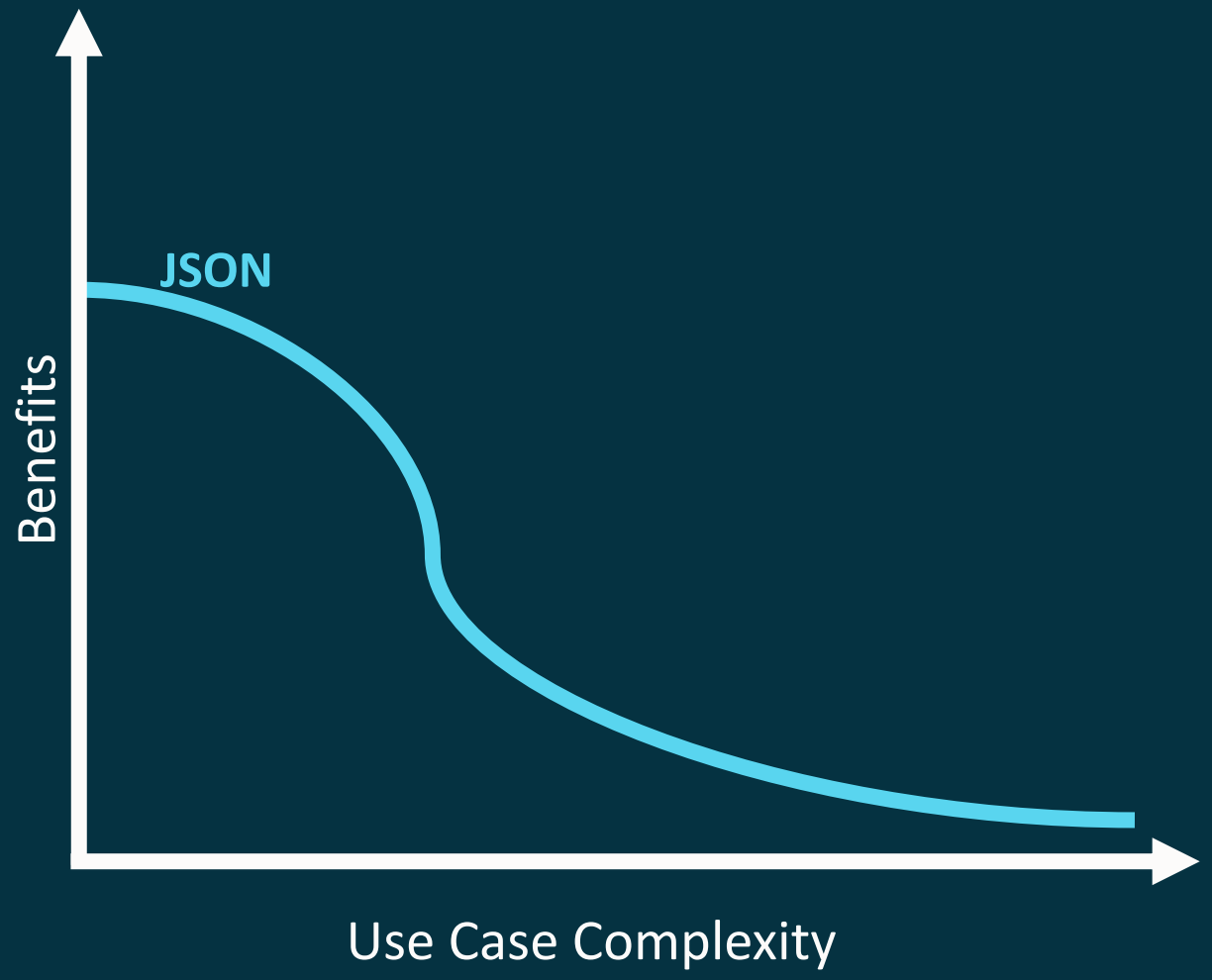


Big Picture — Documents



Documents are great for simple apps

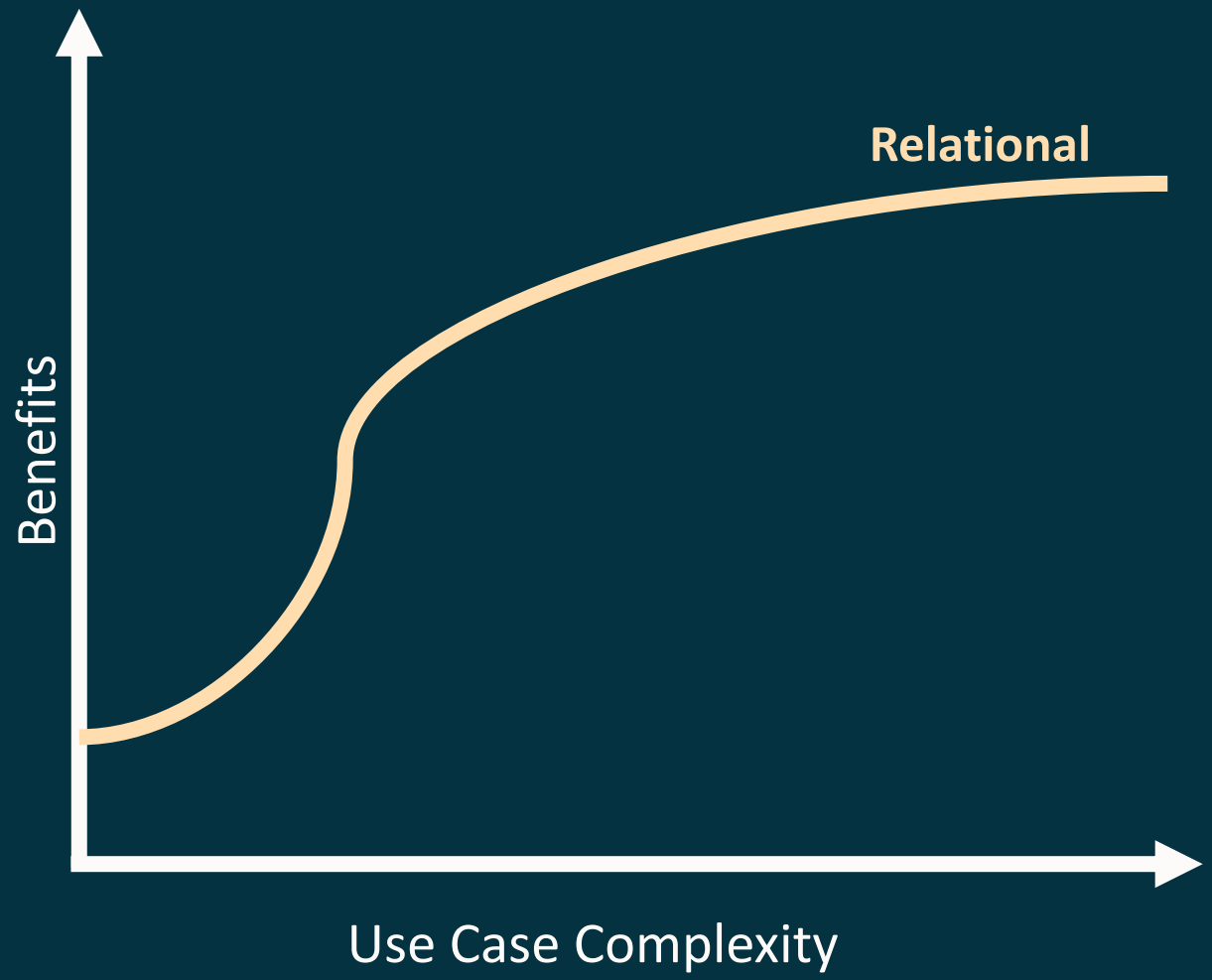
Big Picture — Documents



Become hazardous as apps get more complex

Because of this, many data experts consider pure Document Databases an anti-pattern

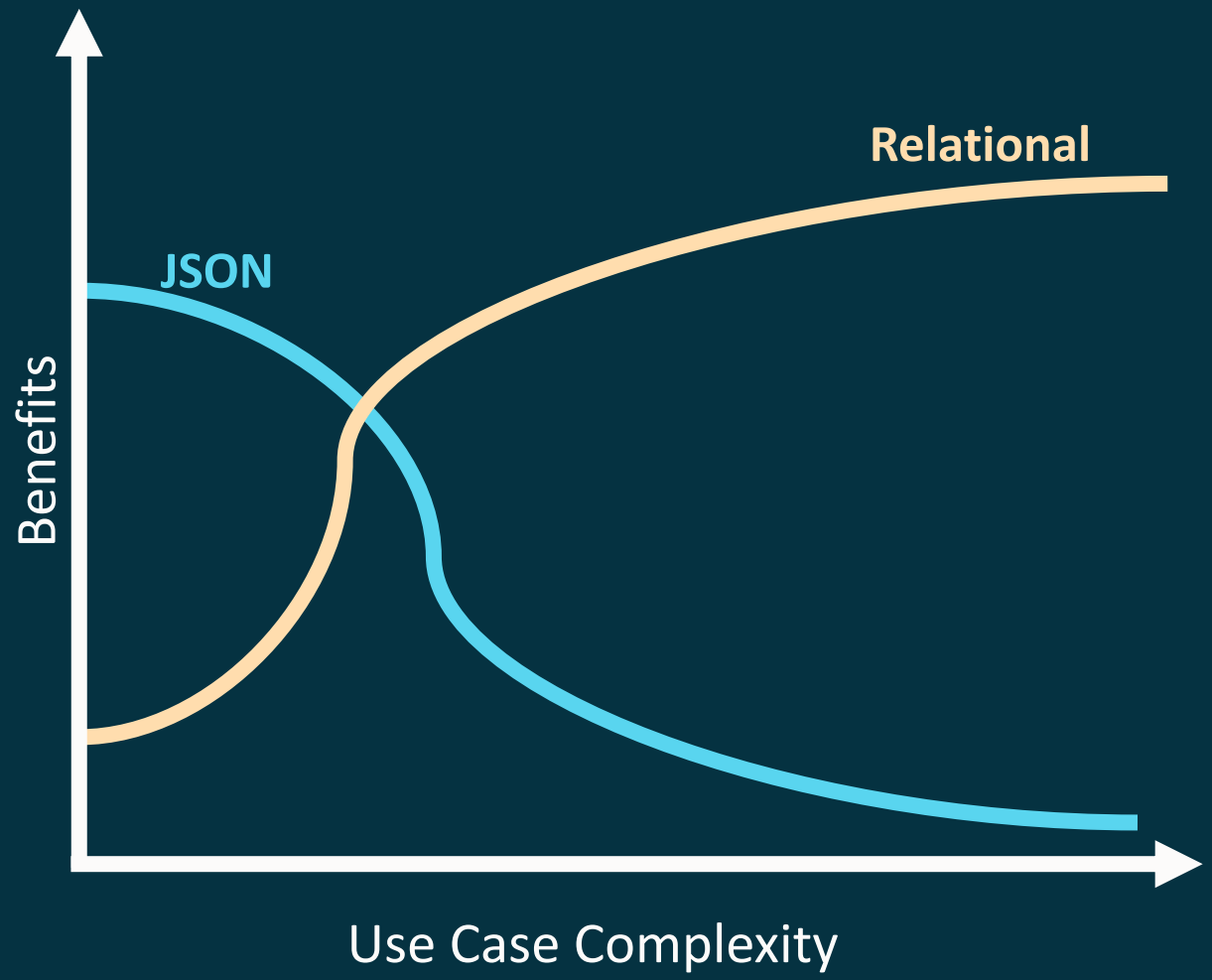
Big Picture — Relational



Relational is not as easy for simple apps

Its power becomes vital as app complexity increases

Oracle Enables Developers to Deliver the Best of Both



With Oracle, developers can already choose the data format that maximizes the benefits for each use case

This is great

Can we do even better?

Instead of choosing
Relational OR Documents

Can we get the benefits of
Relational PLUS Documents?

Can We Get All the Benefits of Both, for Every Use Case?

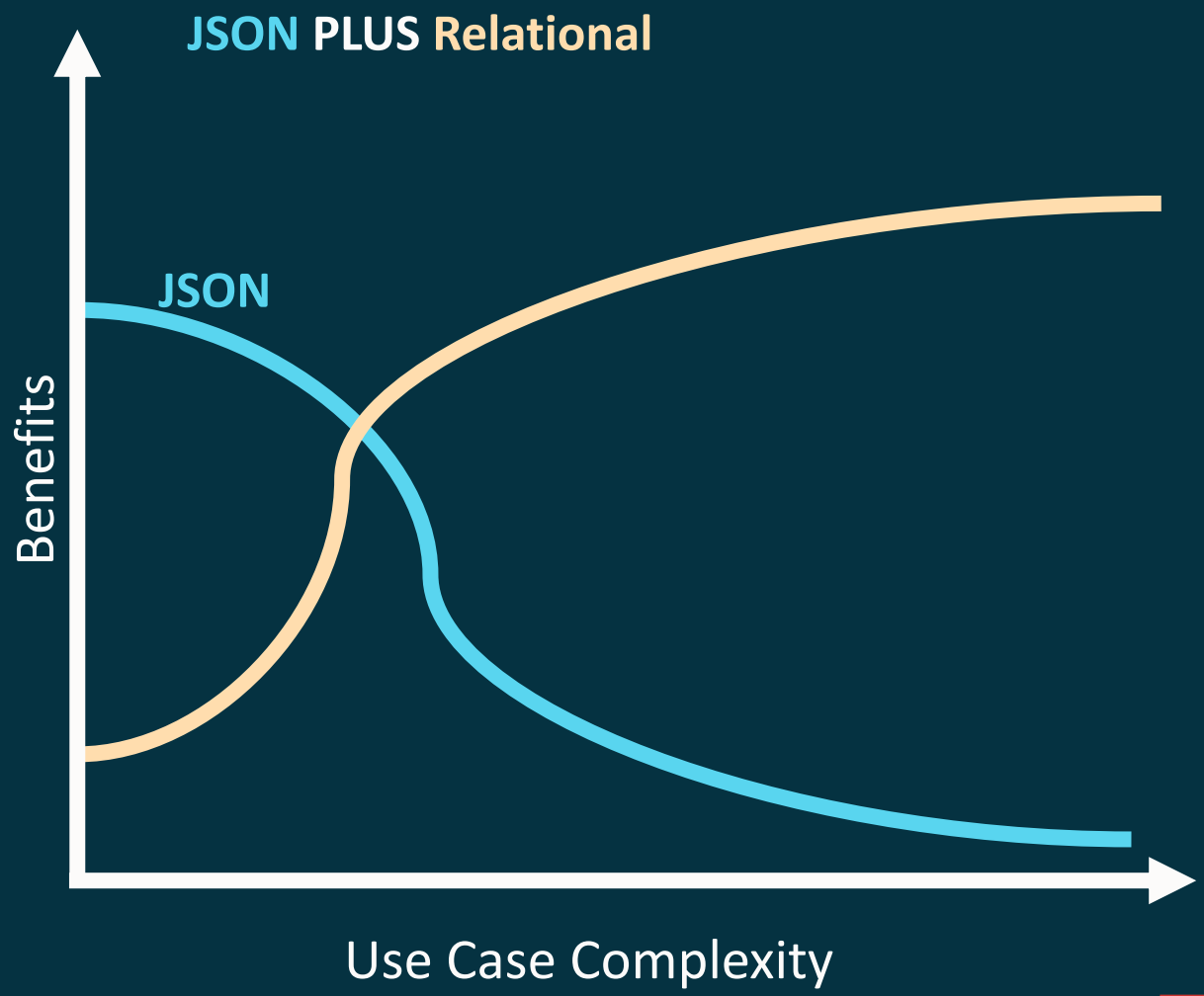
Relational

- Use Case Flexibility
- Queryability
- Consistency
- Space Efficiency

PLUS

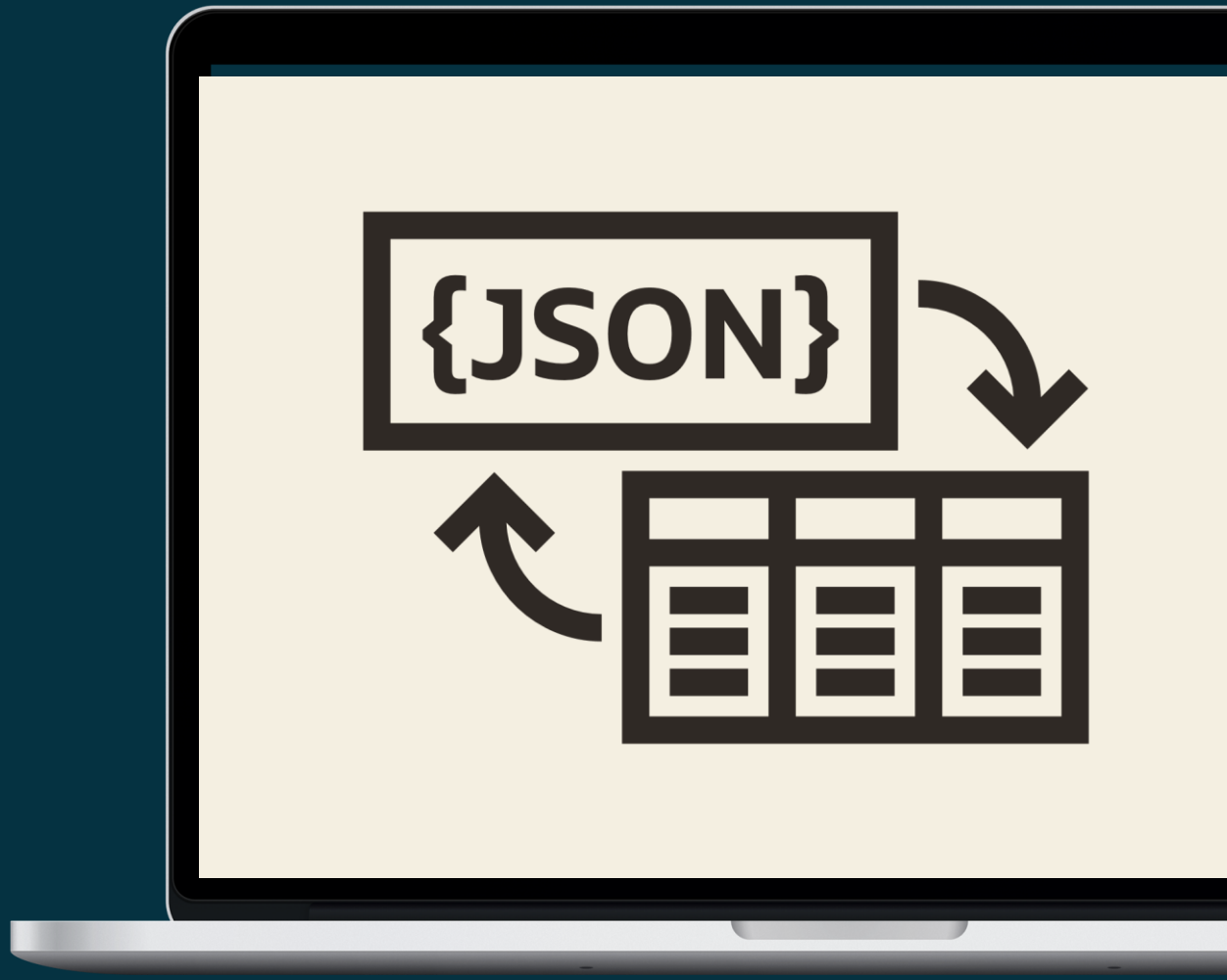
Document

- Easy mapping to language types
- Agile schema-less development
- Hierarchical data format
- Standard interchange format



It’s here, we call it

JSON Document Relational Duality

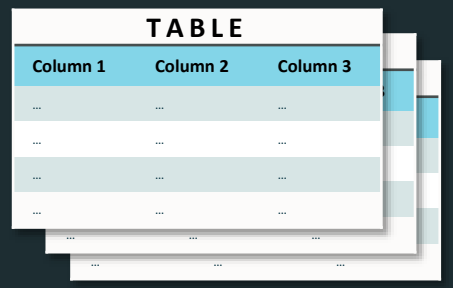


JSON Document Relational Duality

Data is **stored as rows**
in tables to provide the benefits of the relational model and SQL access

Rows can include JSON columns to store data whose schema is dynamic or evolving

Storage Format



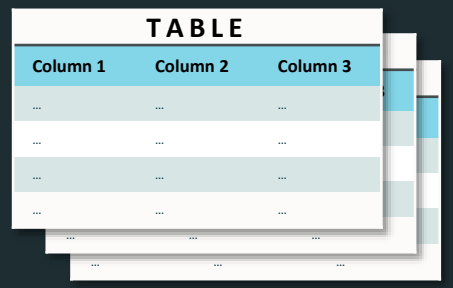
JSON Document Relational Duality

Data is **stored as rows**
in tables to provide the benefits of the relational model and SQL access

Rows can include JSON columns to store data whose schema is dynamic or evolving

Rows can include VECTOR columns to support AI similarity search operations

Storage Format

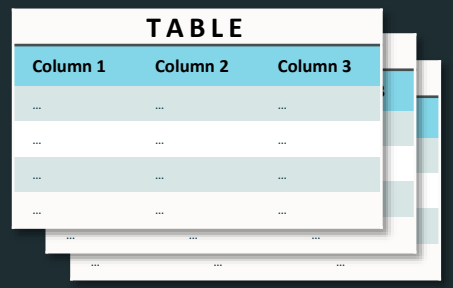


JSON Document Relational Duality

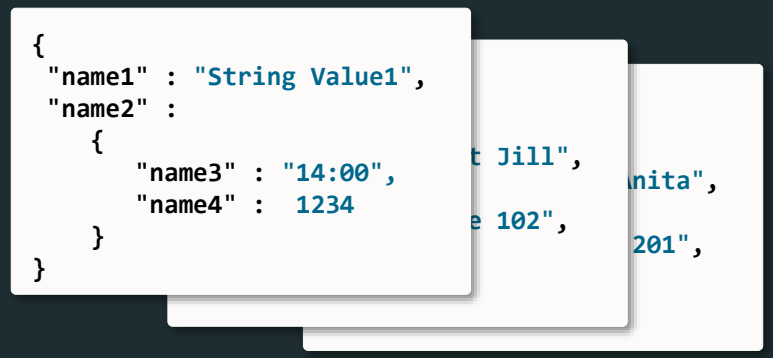
Data is **stored as rows**
in tables to provide the benefits of the relational model and SQL access

Data can be **accessed as JSON documents** to deliver the application simplicity of documents for each use case

Storage Format



Access Formats



The structure of the view mirrors the structure of the desired JSON, making it simple to define

```
CREATE JSON DUALITY VIEW attendeeSchedule
AS attendee
{
   _id         : aid
   name        : name
   company     : company
   schedule    : attendee_sessions
   [ {
      session @unnest
      {
          code    : sid
          session : sname
          time    : time
          room    : room
         speaker @unnest
         {
           speaker : sname
         }
       }
     } ]
};
```

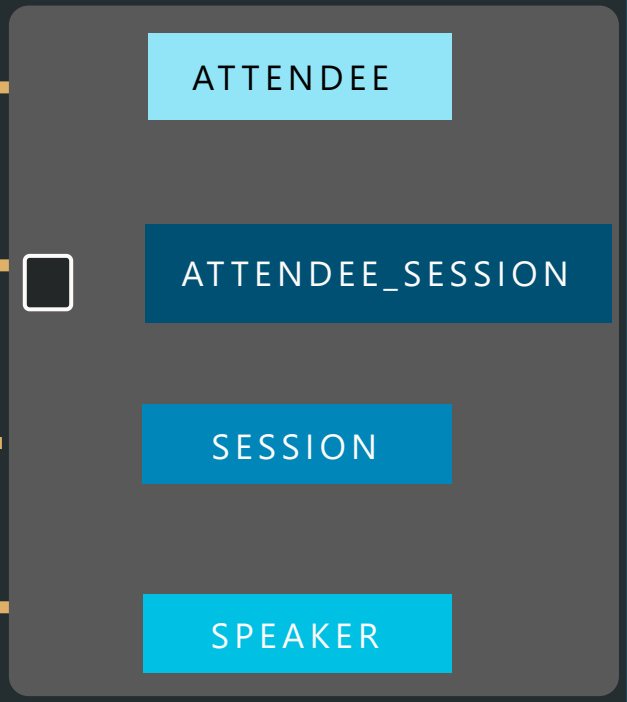
Uses familiar GraphQL syntax

SCHEDULE FOR: JILL

```
{
 "_id"          : "3245",
 "name"         : "Jill",
 "company"      : "ACME Inc",
 "schedule"     : [
     {
         "code"    : "DB12",
         "session" : "SQL",
         "time"    : "14:00",
         "room"    : "A102",
         "speaker" : "Adam"
      },
     …
   ]
}
```

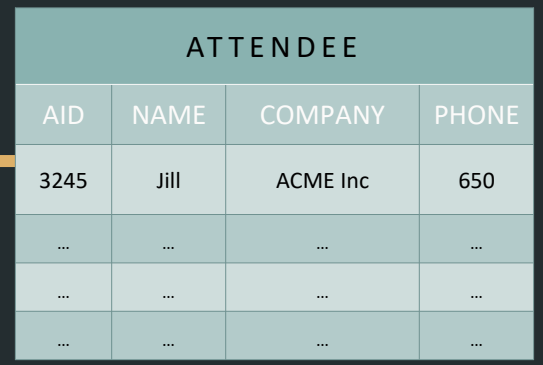
The view simply specifies the tables that contain the data to include in the JSON document

```
CREATE JSON DUALITY VIEW attendeeSchedule
AS attendee
{
   _id         : aid
   name        : name
   company     : company
   schedule    : attendee_sessions
   [ {
      session
      {
          code    : sid
          session : sname
          time    : time
          room    : room
         speaker
         {
           speaker : sname
         }
       }
     } ]
};
```



The view simply specifies the tables that contain the data to include in the JSON document

```
CREATE JSON DUALITY VIEW attendeeSchedule
AS attendee
{
   _id         : aid
   name        : name
   company     : company
   schedule    : attendee_sessions
   [ {
      session
      {
          code    : sid
          session : sname
          time    : time
          room    : room
         speaker
         {
           speaker : sname
         }
       }
     } ]
};
```



The view simply specifies the tables that contain the data to include in the JSON document

```
CREATE JSON DUALITY VIEW attendeeSchedule
AS attendee
{
   _id         : aid
   name        : name
   company     : company
   schedule    : attendee_sessions @delete @insert @Update
   [ {
      session
      {
          code    : sid
          session : sname
          time    : time
          room    : room
         speaker
         {
           speaker : sname
         }
       }
     } ]
};
```

update rules

The view simply specifies the tables that contain the data to include in the JSON document

```
CREATE JSON DUALITY VIEW attendeeSchedule
AS attendee
{
   _id         : aid
   name        : name
   company     : company
   schedule    : attendee_sessions @delete @insert @Update
   [ {
      session
      {
          code    : sid
          session : sname
          time    : time
          room    : room
         speaker @unnest
         {
           speaker : sname
         }
       }
     } ]
};
```

display rules

Example of Using Duality Views

Selecting from the schedule, Duality View accesses the underlying tables and returns Jill's schedule as a JSON document

- This document has all the data needed by the use case
- And the IDs needed to update the data

SCHEDULE FOR: JILL

```
{
 "_id"          : "3245",
 "name"         : "Jill",
 "company"      : "ACME Inc",
 "schedule"     : [
     {
        "code"    : "DB12",
        "session" : "SQL",
        "time"    : "14:00",
        "room"    : "A102",
        "speaker" : "Adam"
      },
      {
        "code"    : "CODE3",
        "session" : "NodeJs",
        "time"    : "16:00",
        "room"    : "R12",
        "speaker" : "Claudia"
      }
   ]
}
```

Example of Using Duality Views

You can access the view using SQL or document APIs

```
SELECT data
FROM   student_schedule s
WHERE  s.data.name = 'Jill';
```

```
student_schedule.find({"name":"Jill"})
```

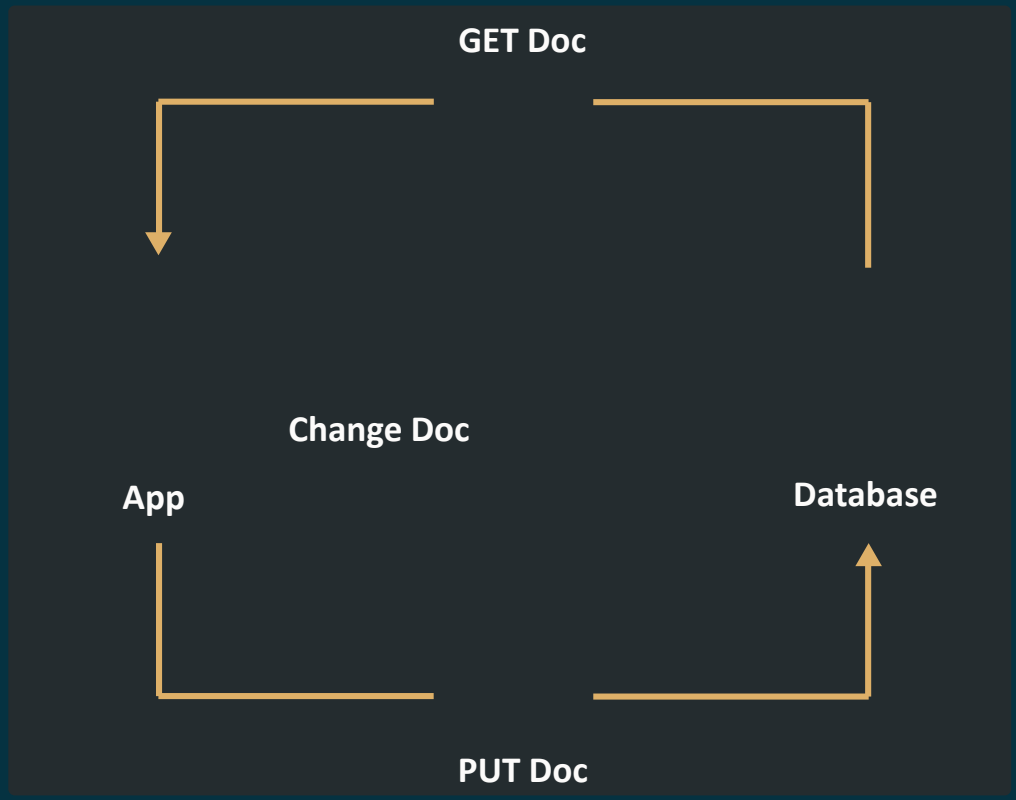
SCHEDULE FOR: JILL

```
{
 "_id"          : "3245",
 "name"         : "Jill",
 "company"      : "ACME Inc",
 "schedule"     : [
     {
        "code"    : "DB12",
        "session" : "SQL",
        "time"    : "14:00",
        "room"    : "A102",
        "speaker" : "Adam"
      },
      {
        "code"    : "CODE3",
        "session" : "NodeJs",
        "time"    : "16:00",
        "room"    : "R12",
        "speaker" : "Claudia"
      }
   ]
}
```

Extreme Simplicity for Developers

JSON Duality Views are extremely simple to access:

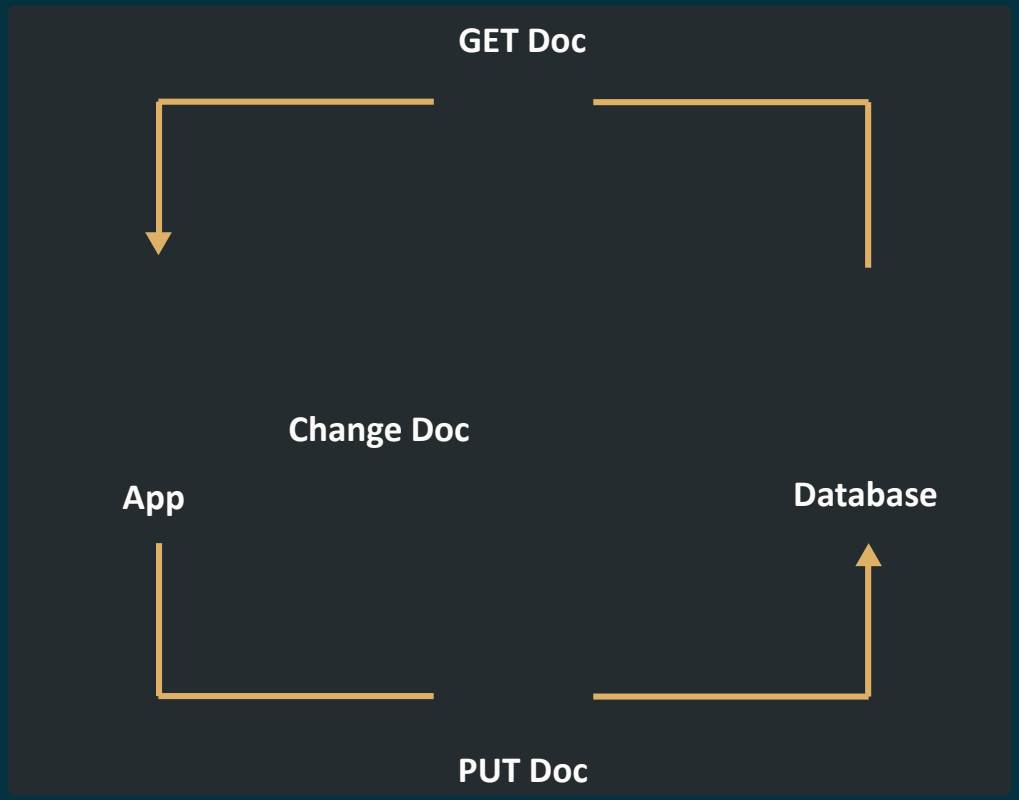
- GET a document from the View
- Make any changes needed to the document
- PUT the document back into the View



Extreme Simplicity for Developers

The database automatically detects the changes in the new document and modifies the underlying rows

- All duality views that share the same data immediately reflect this change
- Developers no longer worry about inconsistencies

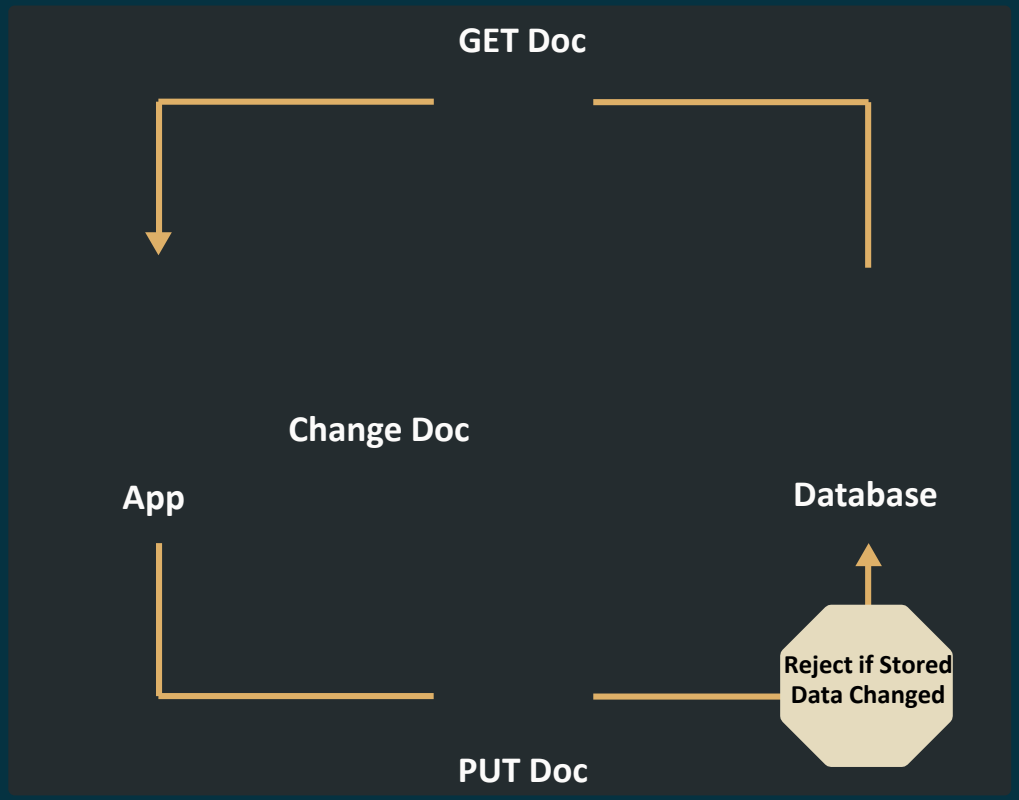


Game Changing Lock-Free Concurrency Control

The database automatically detects when the database data underlying a document has changed between the initial document read and the subsequent write

- If a change occurred, the write operation is automatically rejected and returns an error
- The app can then reissue the write based on the changed data

Called Optimistic Concurrency Control



JSON Relational Duality Views Benefits I

- Regardless of the role anyone can work on the same data set whether JSON or relational and can also build blended applications. Developers can use JSON Relational Duality to join relational and JSON (semi-unstructured) data efficiently.
- Data consistency and integrity across data models and use cases as data is always current and with no lagging and staleness.
- Data stored in Duality Views can be accessed via SQL, REST, document APIs (MongoDB compatible) and many languages and drivers/tool, giving developers broad choices for all use cases.

JSON Relational Duality Views Benefits II

- Better database performance and scalability for mixed workloads (relational + semi-structured).
- Duality View eliminates the need for complex ORMs outside the database because the app developers can directly map programming objects to Duality Views.
- Duality Views are programming language independent where as ORMs support only one programming language. All that makes app development simple and agile.

JSON Collections and the MongoDB API

A native MongoDB API compatible document store

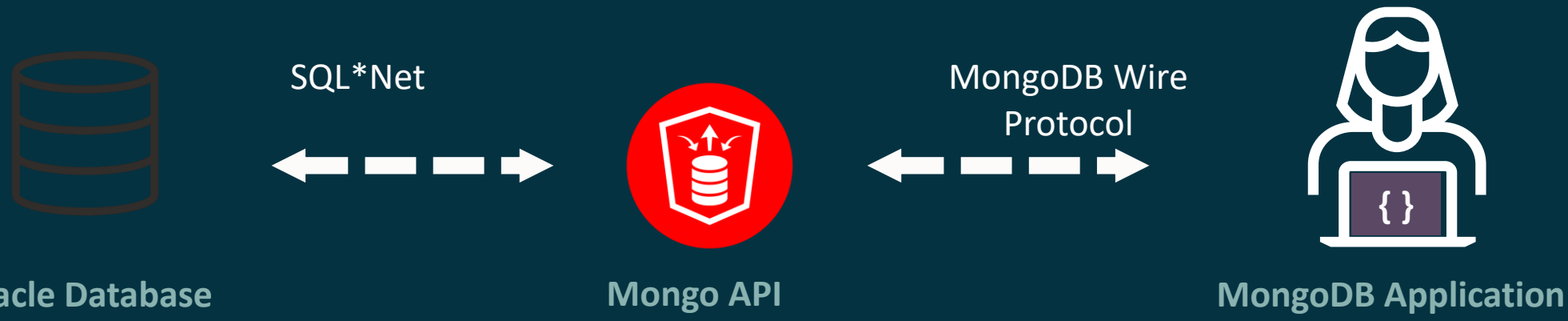
Oracle Database API for MongoDB

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



Oracle Database API for MongoDB

Connect MongoDB client drivers and tools to Oracle Database

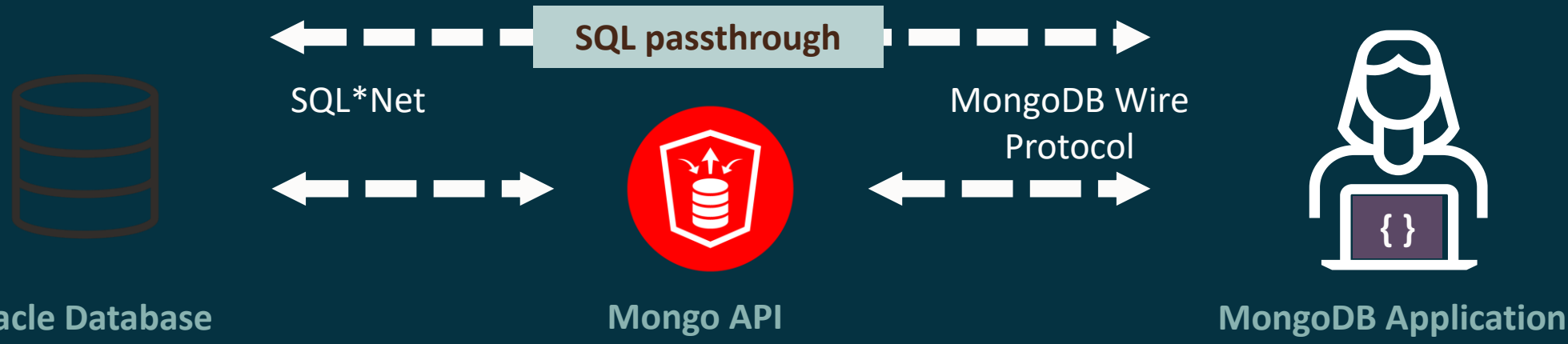
MongoDB does not have tables – it stores collections of JSON documents

- 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**



JSON Collections

A document is a JSON value

- Structure is flexible

A collection contains documents

- Supports insert, get, update, filter

A database contains collections

Access data programmatically – "No SQL"

MongoDB Example

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

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

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

Bsonfilter  =eq("title",  "Iron Man");
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 Example

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

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

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


Bsonfilter   =eq("title",  "Iron Man");
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
movies (
    ID VARCHAR2,
    DATA JSON
);
```

MongoDB Example

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

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

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

Bsonfilter   =eq("title",  "Iron Man");
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 movies
    (data)
  values
    (:1)
;
```

MongoDB Example

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

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

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

Bsonfilter  =eq("title",  "Iron Man");
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 movies e
where e.data.title =
'Iron Man'
```

MongoDB Example

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

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

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

Bsonfilter   = eq("title", "Iron Man");
MongoCursor<Document> cursor = coll.find(filter).cursor();

Document doc=cursor.next();
```

Installing Database API for MongoDB for any Oracle Database

Steps

Step 1: Download

Step 2: Create Directories

Step 3: Unzip the ORDS download

Step 4: Set Environment Variables

Step 5: Run the ORDS installer

Step 6: Configure ORDS to enable MongoDB API

Step 7: Restart ORDS

Step 8: Configure a database user

Step 9: (Optional) Run Database Actions by opening http://localhost:8080/ords/sql-developer

Step 10: Configure Firewall

3 Key Takeaways

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

2. Comprehensive document-store APIs and language support for Java, Python, node.js, and others, supporting MongoDB and Oracle SODA. No knowledge of Oracle or SQL required

3. Leverage existing MongoDB skills and easily move your applications and data to a single converged database and work with your data in whole new ways

Copyright © 2024, Oracle and/or its affiliates

Get Hands On with JSON

livelabs.oracle.com

Store query, and process JSON documents in collections using MongoDB API and SQL/JSON

Use SQL to query, generate and process JSON data

Configure the Mongo API to query or manipulate data in the Oracle Database

Learn the newest SQL Enhancements to work with JSON data

LiveLabs | Search Workshops and Sprints... | Event Code | Sign In

SQL, JSON, and MongoDB API: Unify worlds with Oracle Database 23ai Free

Share | Start

1 hour, 30 minutes

Outline

- Store, query, and process JSON documents in collections using MongoDB API and SQL/JSON
- Use SQL to query, generate, and process JSON data
- Configure the Mongo API to query or manipulate data in the Oracle Database
- Learn the newest SQL Enhancements to work with JSON data

About This Workshop

In this workshop, you will experience Oracle's JSON capabilities using both relational and document-store APIs, namely the Oracle Database API for MongDB. The workshop loosely follows the Moviestreams theme, a series of workshops that demonstrate Oracle converged database capabilities. You will work on our movies library throughout the workshop.

This lab is organized into different topics, each topic consists of multiple steps. After completing this workshop a user has a very good understanding of what JSON features are available in Oracle Database and when to use them. You will work against the same data using both SQL and using the Mongo DB API and will experience why Oracle database is better suited for JSON Development than MongoDB, etc.

You can complete this entire workshop using your web browser. There is no need to install any extra software on your local machine. When writing a real

Prerequisites

- An Oracle Database 23ai Free Developer Release or one running in a LiveLabs environment
- Familiarity with Oracle Database is desirable, but not required
- Familiarity with Mongo API is desirable, but not required
- Some understanding of database terms is helpful

Where To Get More Information

- Live Lab: Developing with JSON and SODA
- Live Lab: Using the Database API for MongoDB
- LiveSQL: SQL/JSON features
- O.com: JSON-based Development in Oracle Database
- O.com: Autonomous JSON Database
- Documentation: JSON Developer's Guide
- Documentation: Overview of Oracle Database API for MongoDB
- @bch_t @juliandontcheff @OracleDatabase
- DW-PM_us@oracle.com

67 Copyright © 2024, Oracle and/or its affiliates