



ORACLE

# Develop Modern Apps with JSON Relational Duality

Oracle Database 23

---



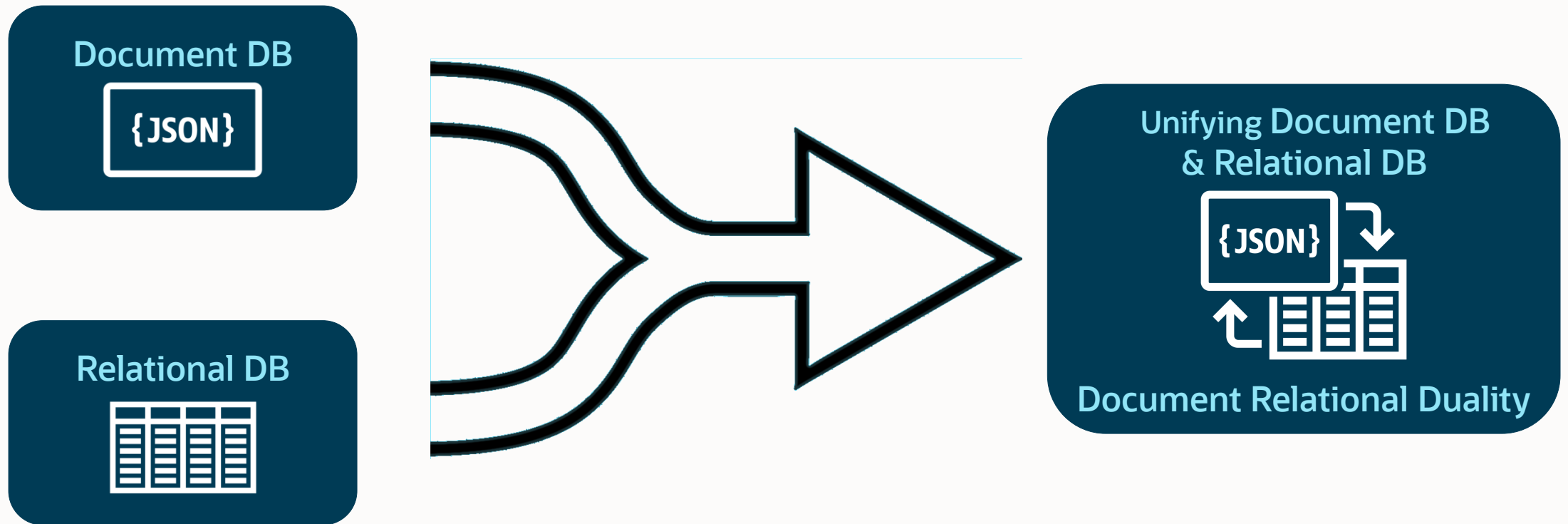
# Safe harbor statement

---

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.

# Document Relational Duality – Paradigm shift in App development

Documents and Relational are Now Fully Unified  
You Get all the Benefits of Relational **Plus** All the Benefits of JSON





Marc Staimer, Senior Analyst, Wikibon, says:

*“Oracle Database 23c definitively ends the long-running ‘relational vs. document’ debate with JSON Relational Duality delivering the best of both worlds”*



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

*”JSON Relational Duality brings simplicity and flexibility to modern app dev*

*...*

*No other databases offer such a revolutionary solution”*

## Both Relational and JSON Models have their own benefits

### Relational

- No Data Duplication
- Use Case Flexibility
- Queryability
- Consistency

### JSON

- Easy mapping to app operations
- Agile schema-less development
- Hierarchical data format
- Easy mapping to app data classes

# Relational approach is powerful but not always the easiest for app dev

Relational model is very powerful

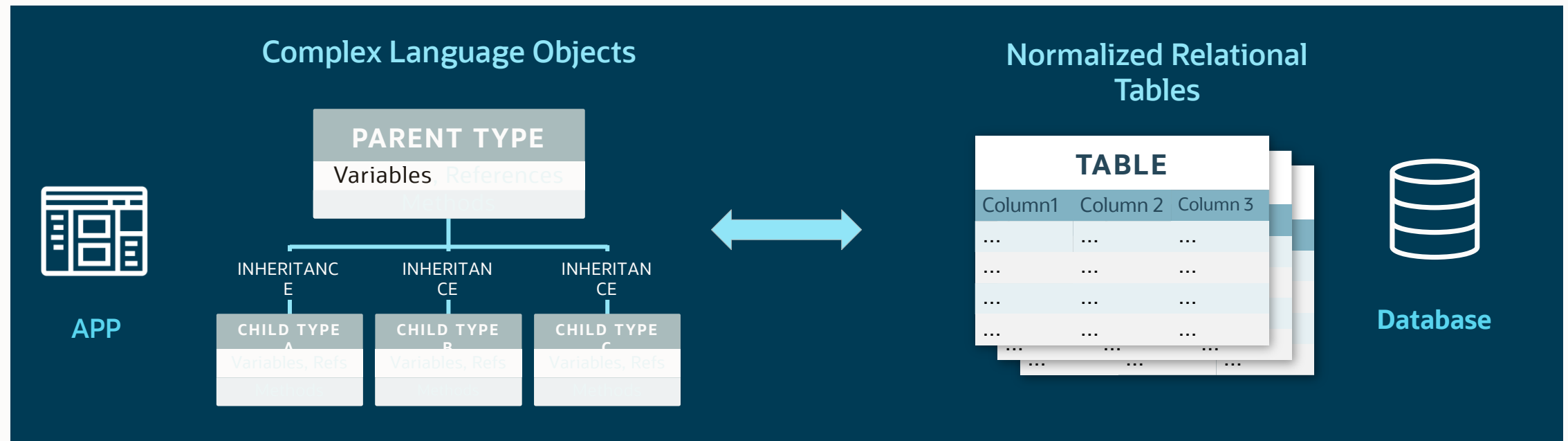
- Uses data normalization to ensure data integrity while avoiding data duplication.
- Relational operations make modeling and accessing data very flexible.

But Relational model is not always the easiest for developers:

- Developers typically build apps in terms of app-tier language objects. Constructing individual application-tier objects often requires accessing multiple tables
- While relational databases store data as tables, rows, and columns.

# Challenge: Language Object to Relational Mismatch

Converting language objects to normalized relational data requires some skill and effort



Object–Relational impedance mismatch





## Using ORM framework

To get around these difficulties, developers often use Object Relational Mapping (ORM) frameworks. While ORMs can simplify app-dev, they also introduce significant overheads:

- They usually require multiple database round-trips
- They do not take full advantage of the capabilities of the database engine
- They do not manage concurrency control very well
- Applications need to use different ORM frameworks for different languages.

# JSON Documents

JSON is popular as an access and interchange format because it is simple

## Simple for any data

A JSON document can represent complex language objects as a simple hierarchy of name value pairs

```
{  
  "name1" : "String Value1",  
  "name2" :  
    {  
      "name3" : "14:00",  
      "name4" : 1234  
    }  
}
```

## Simple to transmit

Self-describing, self-contained format adapts to changes and is easy to transmit between systems

# Oracle Documents is Better than Document Databases

1

Full **document** APIs

SODA: Simple Oracle  
Document Access API -  
MongoDB compatible

2

Provide standards-based  
**SQL** access to documents

Stored Procedures written  
in JavaScript, Java, or  
PL/SQL

3

Full ACID consistency  
across documents

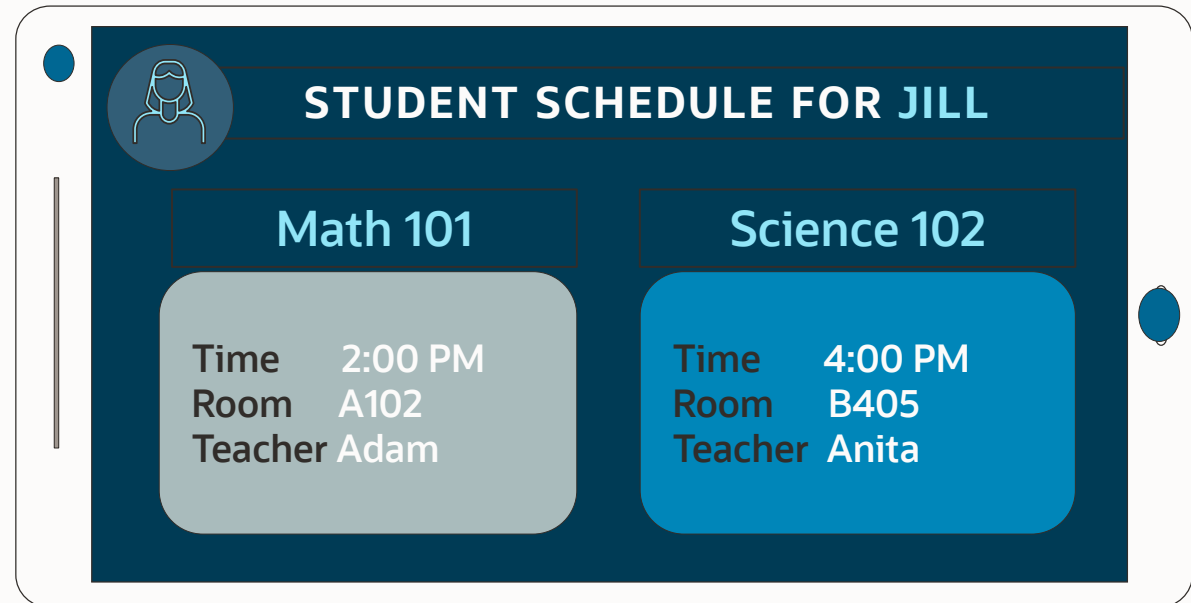
Analytics, Spatial, Graph,  
ML, Parallelization on  
documents

Plus, a serverless Autonomous JSON Database

# Why store data as Documents

An app developer example — A Student Schedule


Imagine we've been asked to build an app that creates a student course schedule




# Why store data as Documents

An app developer example — A Student Schedule


- The data needed for this app is stored in normalized tables inside a relational schema.




STUDENT		
STUID	SNAME	SINFO
S3245	Jill	...
S8524	John	...
S1735	Jane	...
S3409	Jim	...



TEACHER		
TEACHID	TNAME	TINFO
T123	Anika	...
T543	Adam	...
T789	Anita	...
T612	Alex	...



COURSE				
CID	CNAME	ROOM	TIME	TEACHID
C123	MA_01	A102	14:00	T543
C345	SCI_02	B405	16:00	T789
C567	HIS_02	A102	14:00	T612
C789	LA_01	A256	12:00	T543

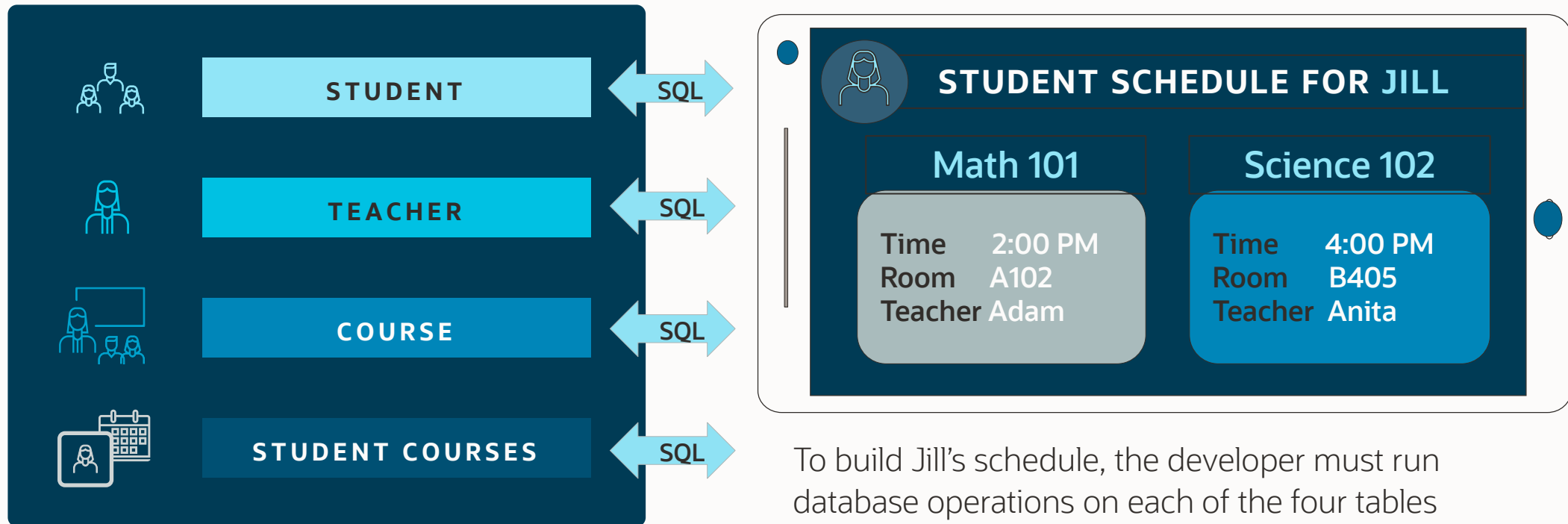


STUDENT COURSES	
STUID	CID
S3245	C123
S8524	C567
S3245	C345
S3409	C123

# Why store data as Documents

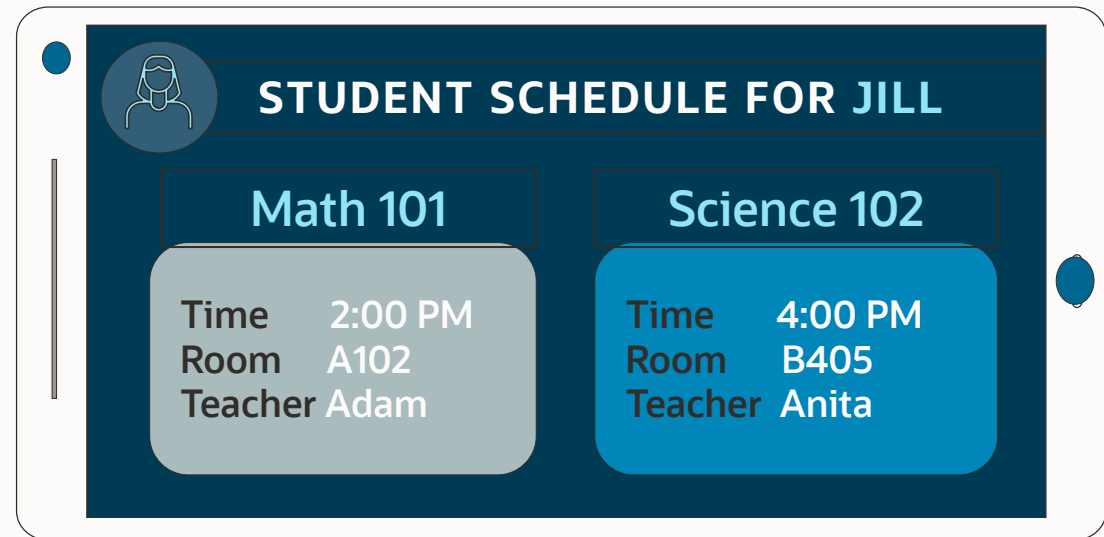
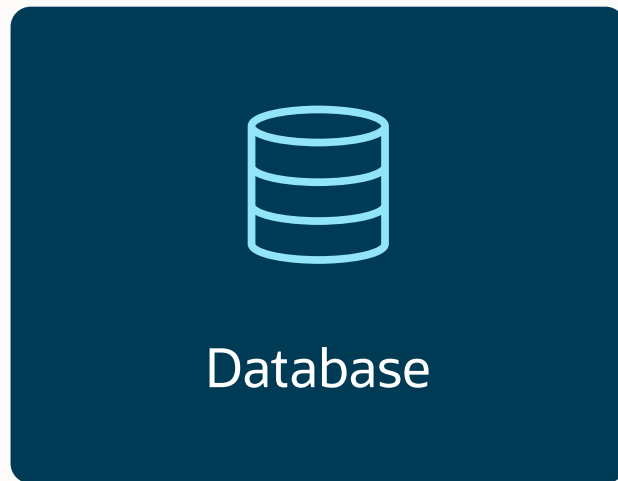
An app developer example — A Student Schedule

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

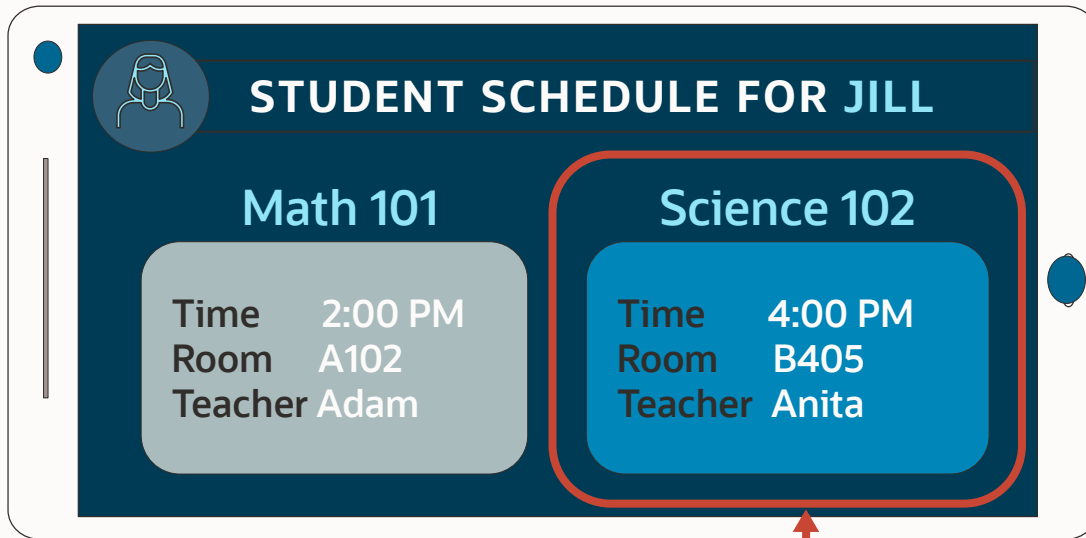


# Relational Data and Developers

- Ideally, the developer wants to build Jill's schedule using a single database operation
- Instead of writing SQL to read from and write to each of the four tables



# Jill's Course Schedule is Simple to Represent in JSON



The data for each of Jill's classes  
is embedded in Jill's JSON  
Schedule document



```
{
  "student"    : "Jill",
  "schedule"   :
  [ {
    "time"      : "14:00",
    "course"    : "Math 101",
    "room"      : "A102",
    "teacher"   : "Adam"
  },
  {
    "time"      : "16:00",
    "course"    : "Science 102",
    "room"      : "B405",
    "teacher"   : "Anita"
  }
]
}
```

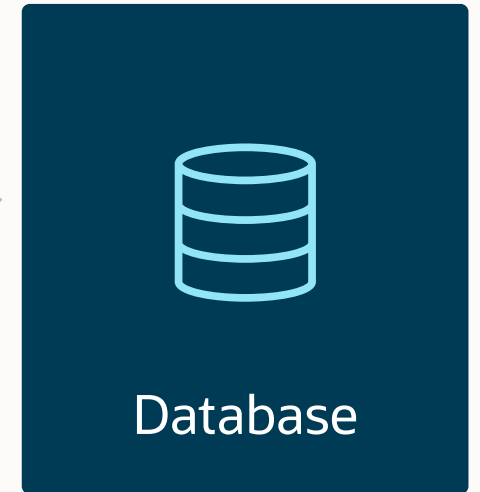


# Document Databases Make Operating on JSON Simple

Document Databases are popular because they allow JSON documents to be easily accessed and stored

Using simple GET/PUT APIs

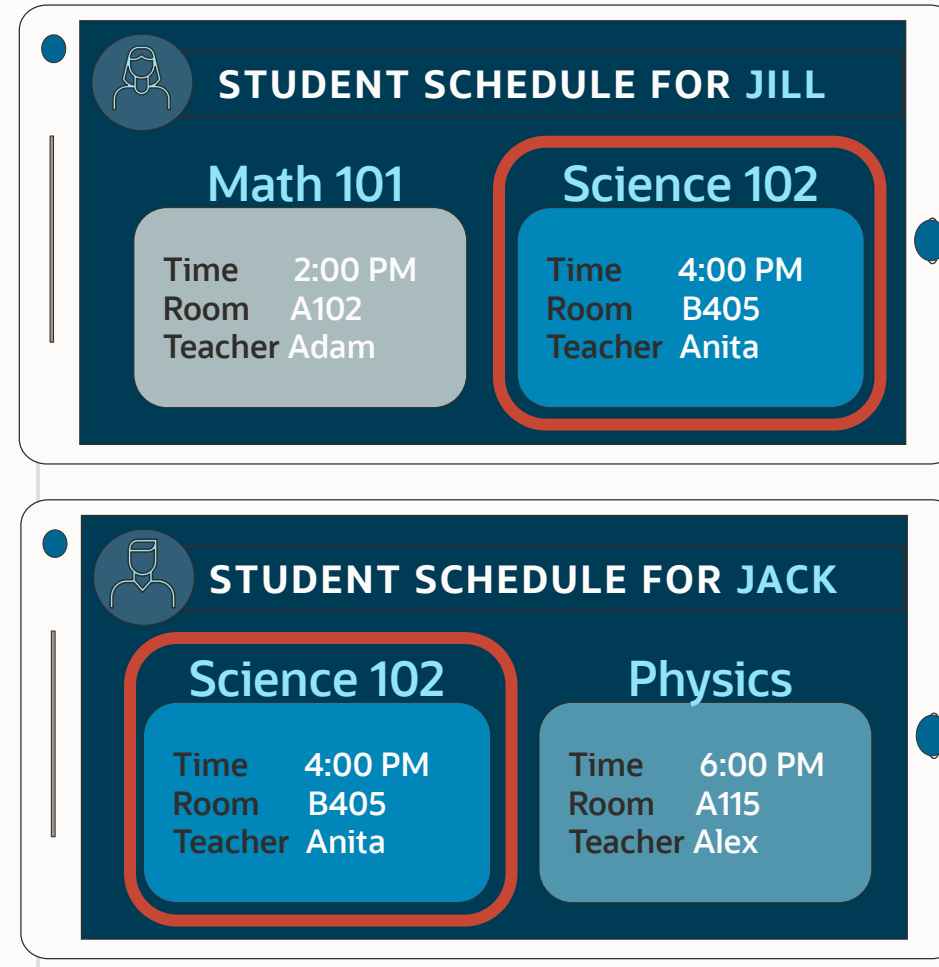
```
{
  "student"    : "Jill",
  "schedule"   :
  [ {
    "time"      : "14:00",
    "course"    : "Math 101",
    "room"      : "A102",
    "teacher"   : "Adam"
  },
    {
    "time"      : "16:00",
    "course"    : "Science 102",
    "room"      : "B405",
    "teacher"   : "Anita"
  }
  ]
}
```



# Limitations of Document Databases

JSON Documents are simple for apps to use as a data **access format**

When used as a **storage format**, they create issues with data duplication and data consistency

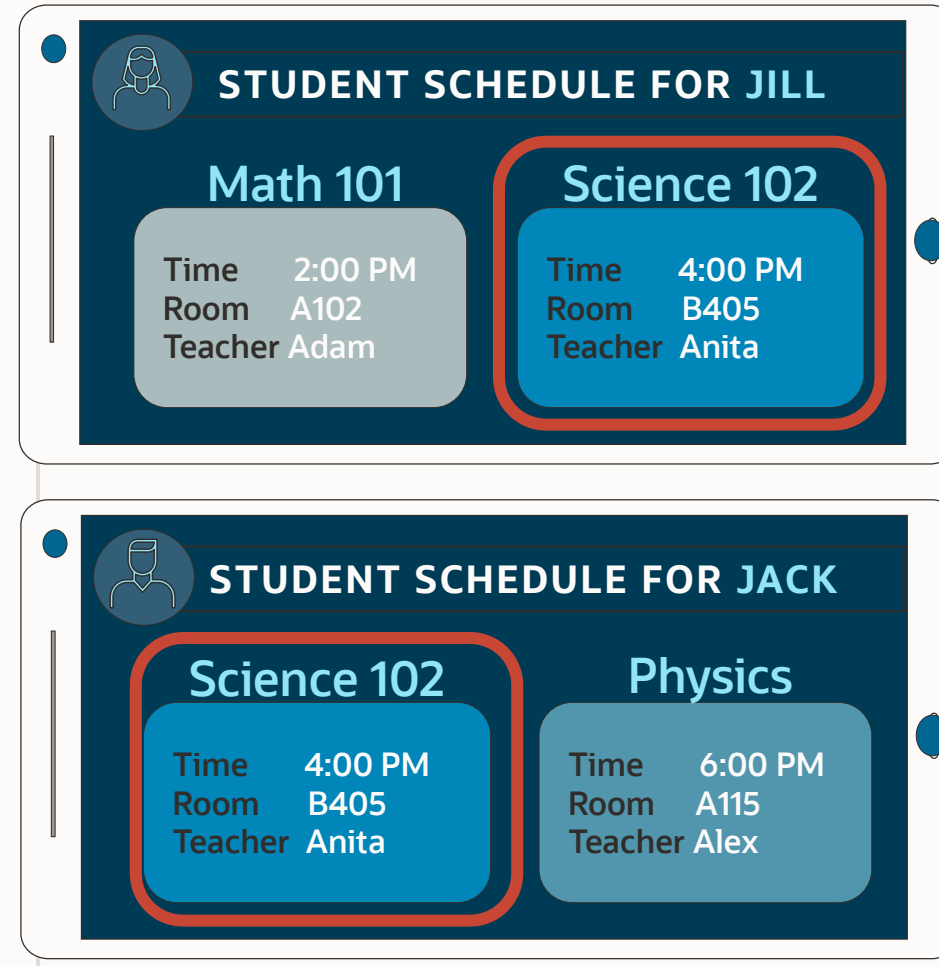


# Limitations of Document Databases

When JSON is used to store student schedules, the course and teacher information is redundantly stored in each student schedule

Duplicate data is

- Inefficient to store
- Expensive to update
- Difficult to keep consistent

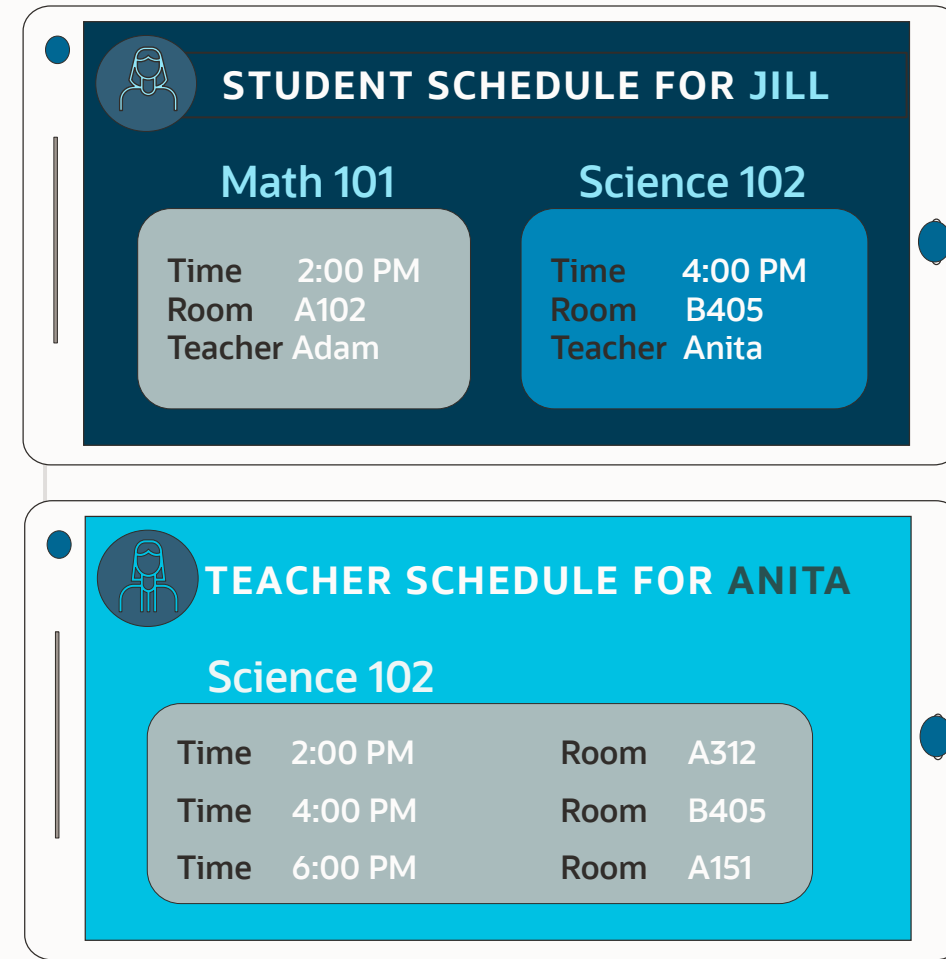


# Same Data for Multiple Use Cases

Successful apps inevitably add more use cases over time

Data duplication increases as the same data is used in **new use cases**

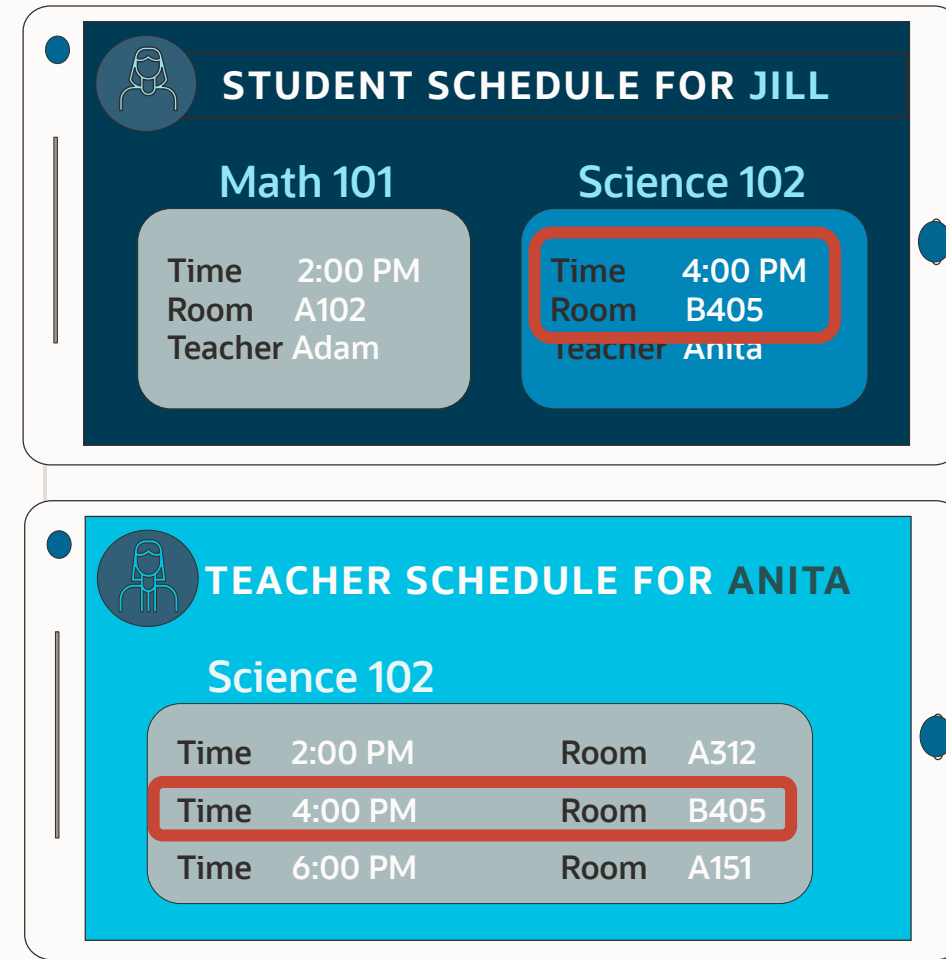
For example, when adding a teacher schedule use case



# Same Data for Multiple Use Cases

The teacher schedule use case requires a new document shape with the teacher as the root


- But shares some of the same course data as the student document




# Same Data for Multiple Use Cases

Since data is stored redundantly, changing the classroom for a course requires atomically updating **many** student schedule documents

**AND** many teacher schedule documents

STUDENT SCHEDULE FOR: JILL 

```
{
  "student"    : "S3245",
  "name"       : "Jill",
  "schedule"   :
    [ {
      "time"    : "14:00",
      "course"  : "Math 101",
      "room"    : "A102",
      "teacher" : "Adam"
    },
    {
      "time"    : "16:00",
      "course"  : "Science 102",
      "room"    : "B105",
      "teacher" : "Anita"
    }
  ]
}
```

TEACHER SCHEDULE FOR: ANITA 

```
{
  "teacher"    : "T9351",
  "name"       : "Anita",
  "schedule"   :
    [{
      "time"    : "14:00",
      "course"  : "Science 102",
      "room"    : "A312"
    },
    {
      "time"    : "16:00",
      "course"  : "Science 102",
      "room"    : "B405"
    },
    {
      "time"    : "18:00",
      "course"  : "Science 102",
      "room"    : "A115"
    }
  ]
}
```

# Document Databases approach to normalization leads to fragmentation

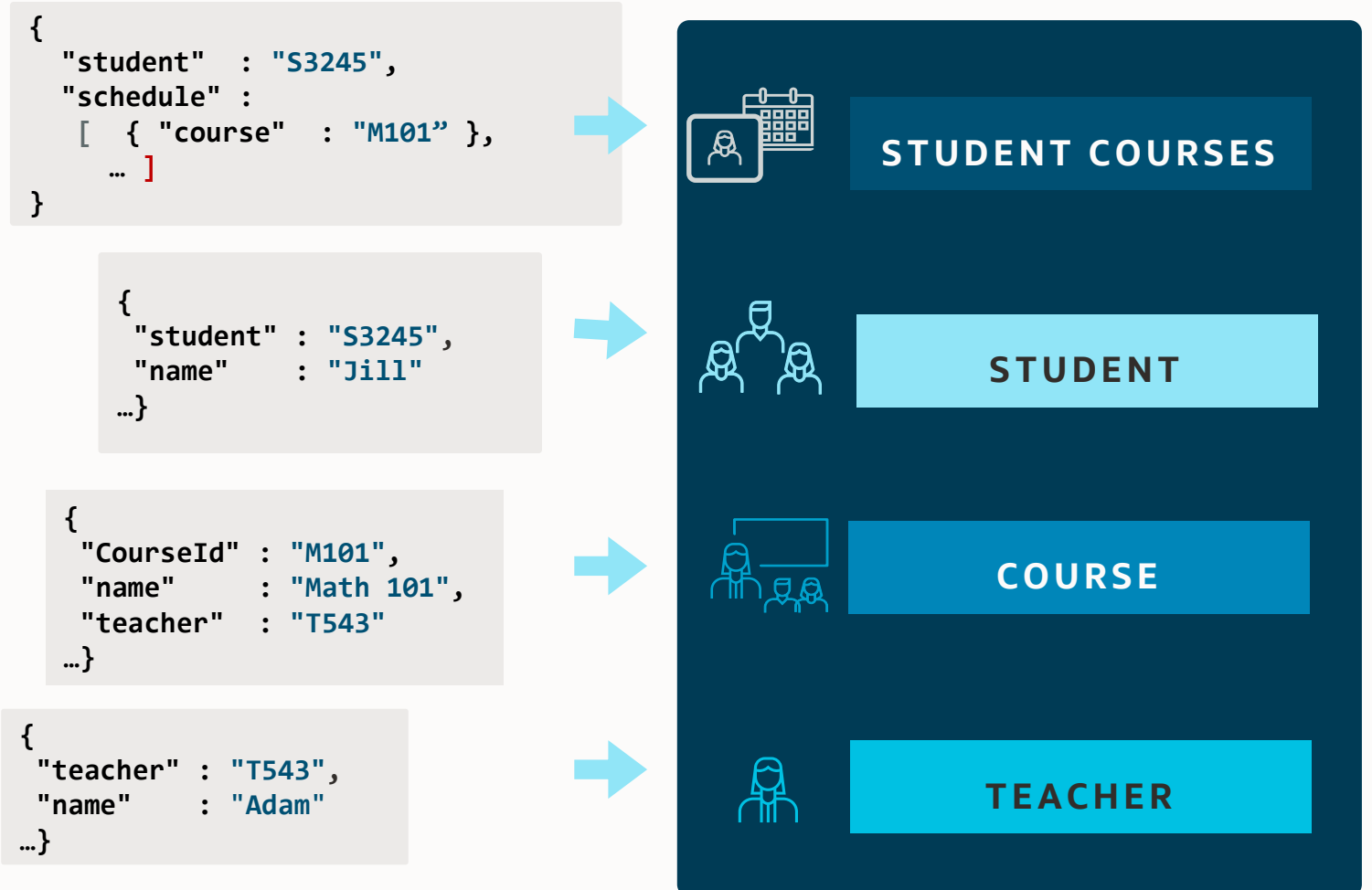
## Normalizing documents produces the worst of both worlds

The simplicity of documents at the application level is lost

- Without gaining the power of SQL and relational at the database level

Performance suffers due to reference chasing and loss of shard locality

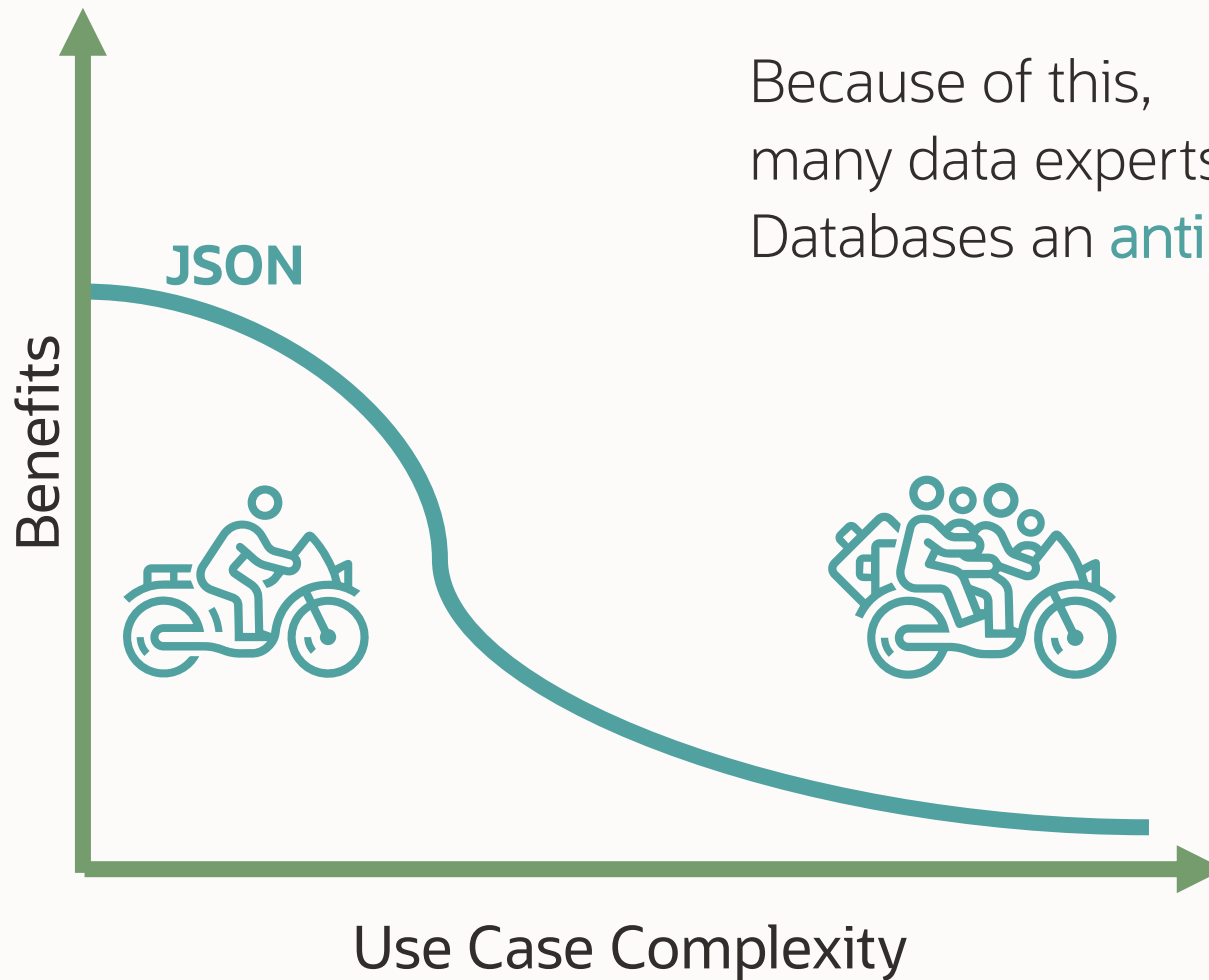
- Referential integrity must be enforced by every application



# Big Picture - Documents

Become hazardous as apps get more complex

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

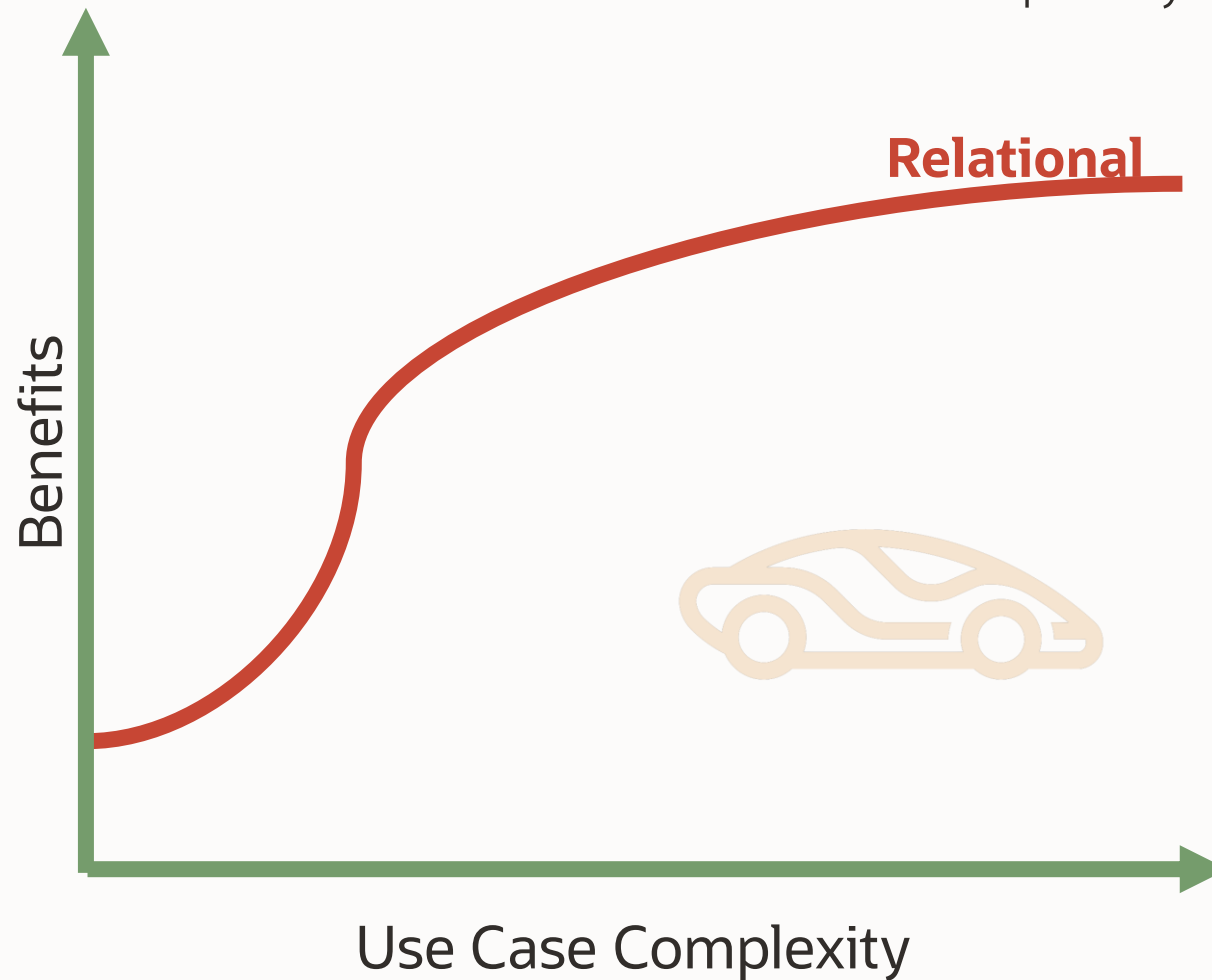




# Big Picture - Relational

Relational is not as easy for simple apps

Its power becomes vital as app complexity increases



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

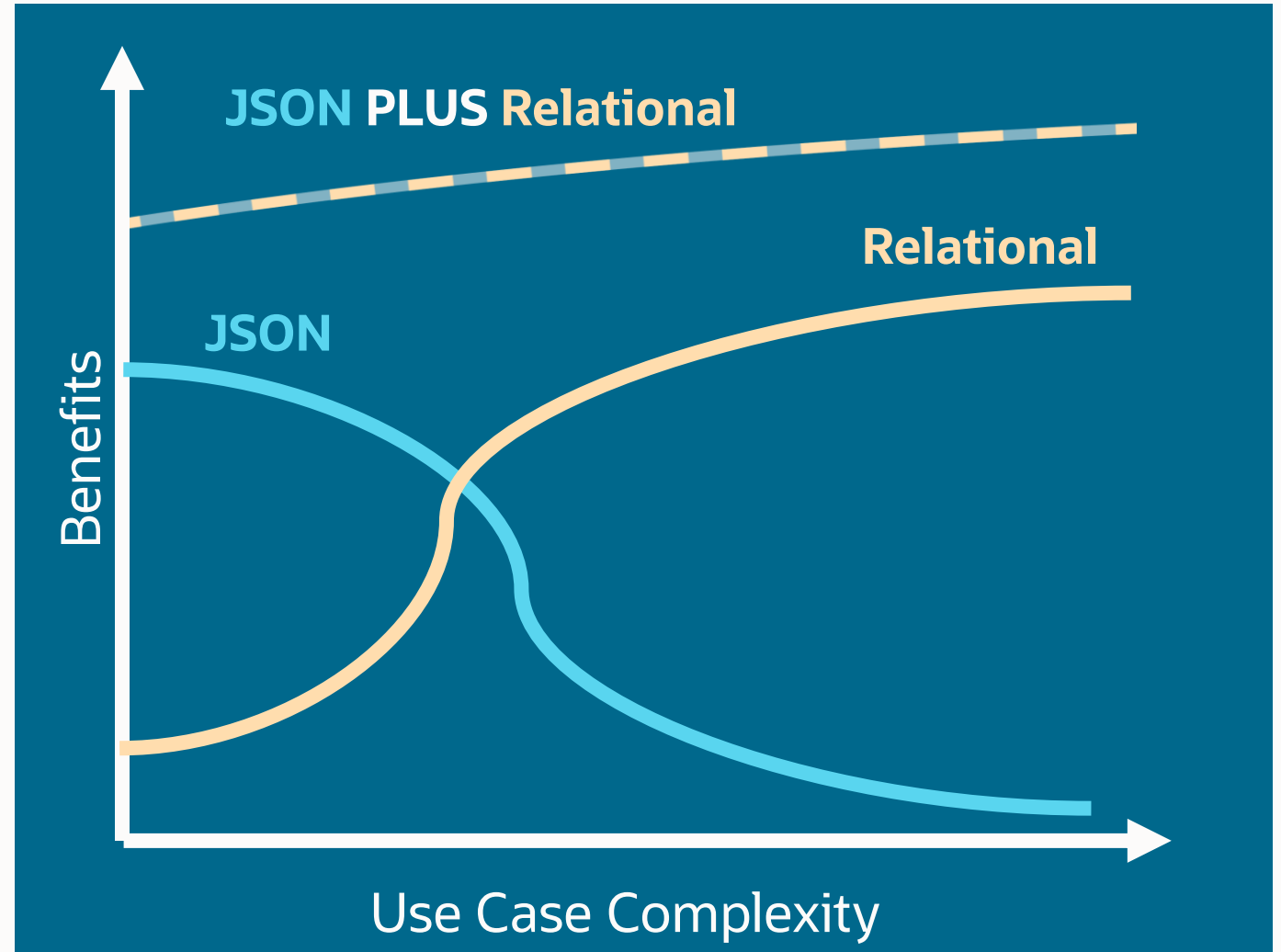
## Relational

- Use Case Flexibility
- Queryability
- Consistency
- Space Efficiency

**+ PLUS**

## Document

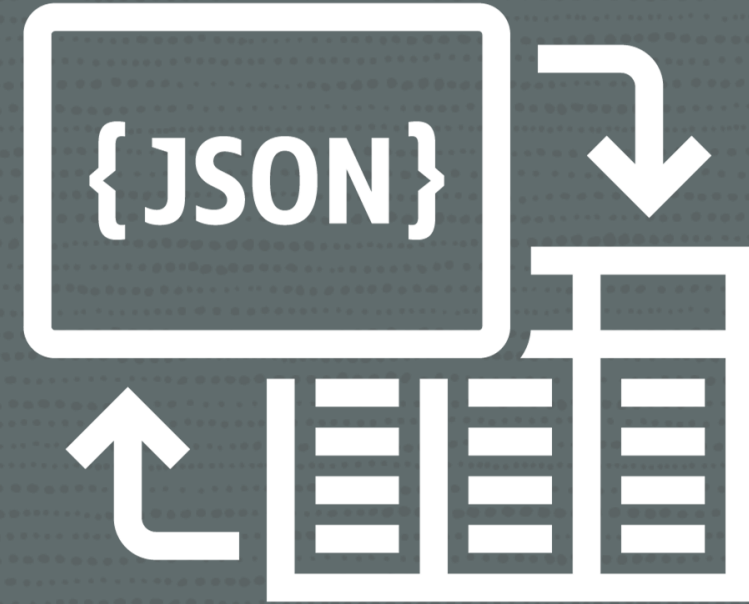
- Easy mapping to language types
- Agile schema-less development
- Hierarchical data format
- Standard interchange format



INTRODUCING

# JSON Document Relational Duality

Oracle Database 23c



Instead of choosing Relational **OR** Documents - we CAN get the benefits of Relational **PLUS** Documents...

# JSON Document Relational Duality – Best of both worlds

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



# Simplify App development using Duality views

- Duality view is a database object like table, view etc - No separate mapping language or tools, no programming, no deploying, no configuring
- Document-level consistency, and table row-level consistency, are guaranteed together. Changes to either are transparently and immediately reflected in the other.
  - Multiple applications can also update documents or their underlying tables concurrently.
  - Existing SQL tools can update table rows at the same time applications update documents based on those rows.
- Define rules for handling parts of documents centrally in the database, not in application code. Same rule applies across applications or languages.
- Available to any database feature and any application.

# Simplify App development using Duality views

## Duality Views are ..

- ✓ Language independent
- ✓ Optimized by the database
- ✓ Get all use-case data in one round trip
- ✓ Consolidate object mapping in the DB
- ✓ Provide better concurrency control

# Declarative document definitions

# Declarative Document Definition

A key benefit of SQL and the relational model is declarative programming

- Allows developers to tell the database what they want
- The database automatically determines how best to do it

Say What, Not How

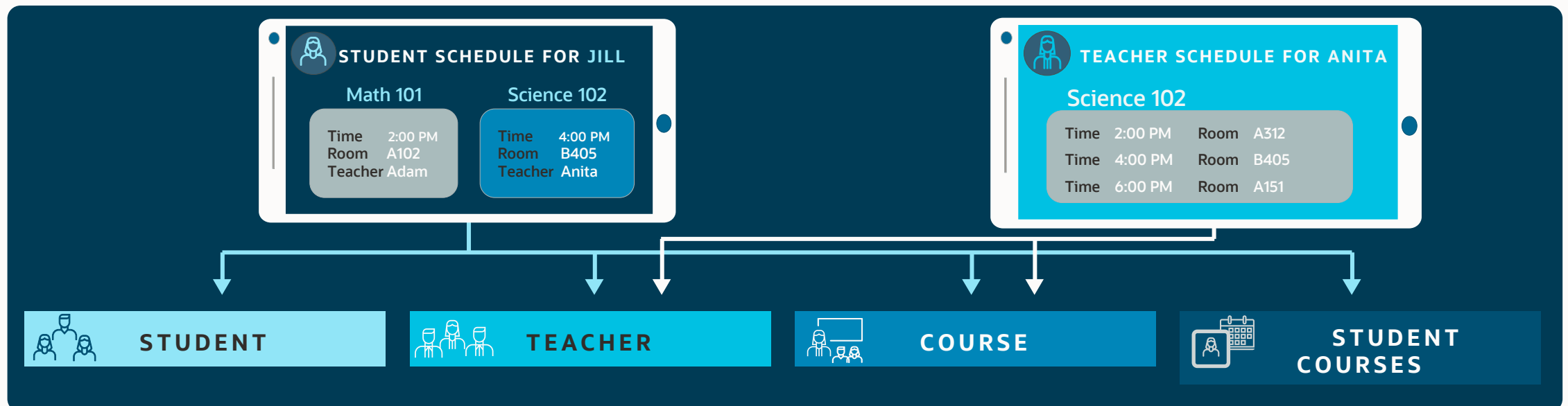


# Game Changing Document Specialization by Use Case

Document Relational Duality enables game-changing specialization of Documents

- A Duality View can be created for each app use-case that produces a specialized JSON document for that case
- Makes it easy to add new use-cases for the same data, without creating duplication or consistency issues

E.g., the **same underlying row data** can be used to create Student and Teacher Schedule documents



# Defining JSON Relational Duality

JSON Duality Views **declare** the **recipe** for assembling normalized rows into a JSON document



```
CREATE OR REPLACE JSON DUALITY VIEW FROM Student AS
student_schedule
{
  name:      sname
  student_id: stuid
  schedule:  student_courses
  {
    course: course
    {
      time
      course:  cname
      course_id: cid
      room
      teacher: teacher
      {
        teacher:  tname
        teacher_id: tid
      }
    }
  }
};
```

Example of how to declare a **Student Schedule Duality View** using in-database GraphQL syntax

The structure of the view mirrors the structure of the desired JSON object

**STUDENT SCHEDULE FOR:JILL**

```
{
  .....
  [ {
    .....
  },
  {
    .....
  }
  ]
}
```

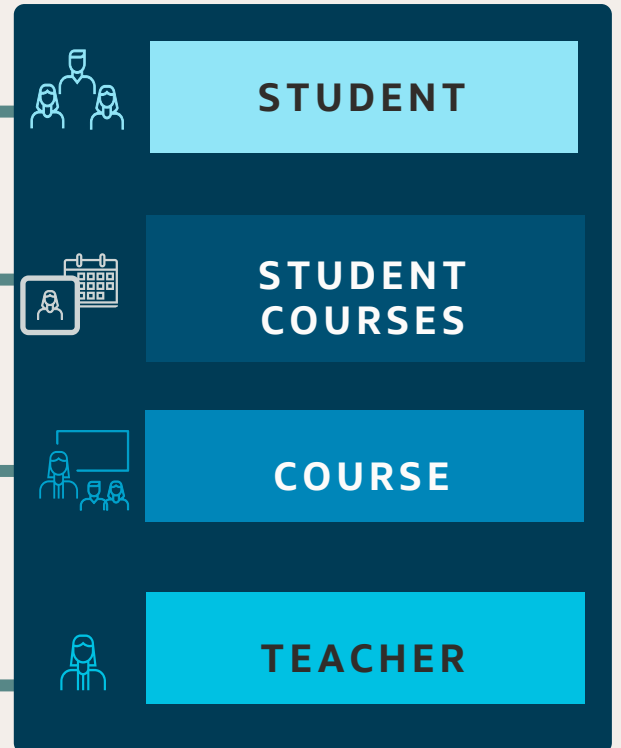
# Defining JSON Relational Duality

JSON Duality Views **declare** the **recipe** for assembling normalized rows into a JSON document



```
CREATE OR REPLACE JSON DUALITY VIEW FROM Student AS
student_schedule
{
  name:      sname
  student_id: stuid
  schedule:  student_courses
  {
    course: course
    {
      time
      course:  cname
      course_id: cid
      room
      teacher: teacher
      {
        teacher:  tname
        teacher_id: tid
      }
    }
  }
};
```

Specify the **table**  
containing the data for  
the JSON document

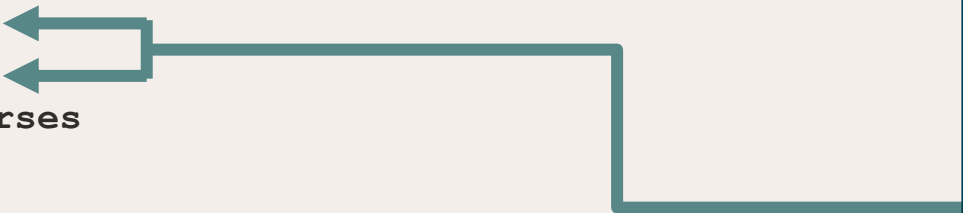


# Defining JSON Relational Duality

JSON Duality Views **declare** the **recipe** for assembling normalized rows into a JSON document



```
CREATE OR REPLACE JSON DUALITY VIEW FROM Student AS
student_schedule
{
  name:      sname
  student_id: stuid
  schedule:  student_courses
  {
    course: course
    {
      time
      course:  cname
      course_id: cid
      room
      teacher: teacher
      {
        teacher:  tname
        teacher_id: tid
      }
    }
  }
};
```



Specify each **JSON property** name and **table column** to get its value from

STUDENT		
STUID	SNAME	SINFO
S3245	Jill	...
S8524	John	...
S1735	Jane	...
S3409	Jim	...



# Defining JSON Relational Duality

JSON Duality Views **declare** the **recipe** for assembling normalized rows into a JSON document



```
CREATE OR REPLACE JSON DUALITY VIEW FROM Student AS
```

```
student_schedule
```

```
{
```

```
  name:      sname
```

```
  student_id: stuid
```

```
  schedule:  student_courses @delete @insert @update
```

```
  {
```

```
    course: course
```

```
    {
```

```
      time
```

```
      course:  cname
```

```
      course_id: cid
```

```
      room
```

```
      teacher: teacher
```

```
      {
```

```
        teacher:  tname
```

```
        teacher_id: tid
```

```
      }
```

```
    } }
```

```
};
```

Specify **updatability rules**  
(students can update student schedules, but not classes or teachers)

# Defining JSON Relational Duality

JSON Duality Views **declare** the **recipe** for assembling normalized rows into a JSON document



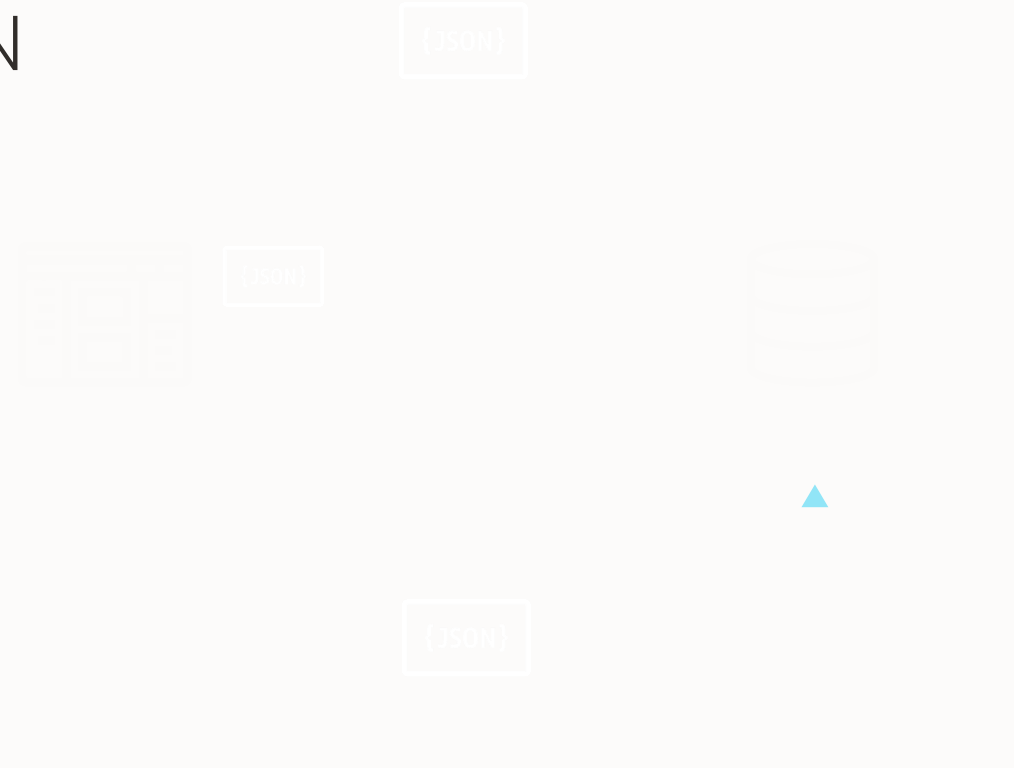
```
CREATE OR REPLACE JSON DUALITY VIEW FROM Student AS
student_schedule
{
  name:      sname
  student_id: stuid
  schedule:  student_courses @delete @insert @update
  {
    course: course @unnest
    {
      time
      course:  cname
      course_id: cid
      room
      teacher: teacher @unnest
      {
        teacher:  tname
        teacher_id: tid
      }
    }
  }
};
```

Specify when properties in a nested object should be unnested into the parent

# Extreme Simplicity for Developers

Flexibility to use variety of tools with JSON  
Relational Duality (some examples)

- SQL
- AutoREST
- Oracle MongoDB compatible API



# Using Duality Views

Selecting from the student 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

## STUDENT SCHEDULE FOR: JILL



```
{
  "student_id" : "S3245",
  "name"       : "Jill",
  "schedule"   :
    [ {
      "time"    : "14:00",
      "course"  : "Math 101", "course_id" : "C123",
      "room"    : "A102",
      "teacher" : "Adam",      "teacher_id": "T543"
    },
    {
      "time"    : "16:00",
      "course"  : "Science 102", "course_id" : "C345",
      "room"    : "B405",
      "teacher" : "Anita",      "teacher_id" : "T789"
    }
  ]
}
```



# Example of Using Duality View

You can access the view using SQL or Mongo API

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

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



## STUDENT SCHEDULE FOR: JILL



```
{
  "student_id" : "S3245",
  "name"       : "Jill",
  "schedule"   :
    [ {
      "time"    : "14:00",
      "course"  : "Math 101", "course_id" : "C123",
      "room"    : "A102",
      "teacher" : "Adam",    "teacher_id": "T543"
    },
    {
      "time"    : "16:00",
      "course"  : "Science 102", "course_id" : "C345",
      "room"    : "B405",
      "teacher" : "Anita",    "teacher_id" : "T789"
    }
  ]
}
```

# Example of Using Duality View - AutoREST

JSON Duality Views are extremely simple to access using REST:

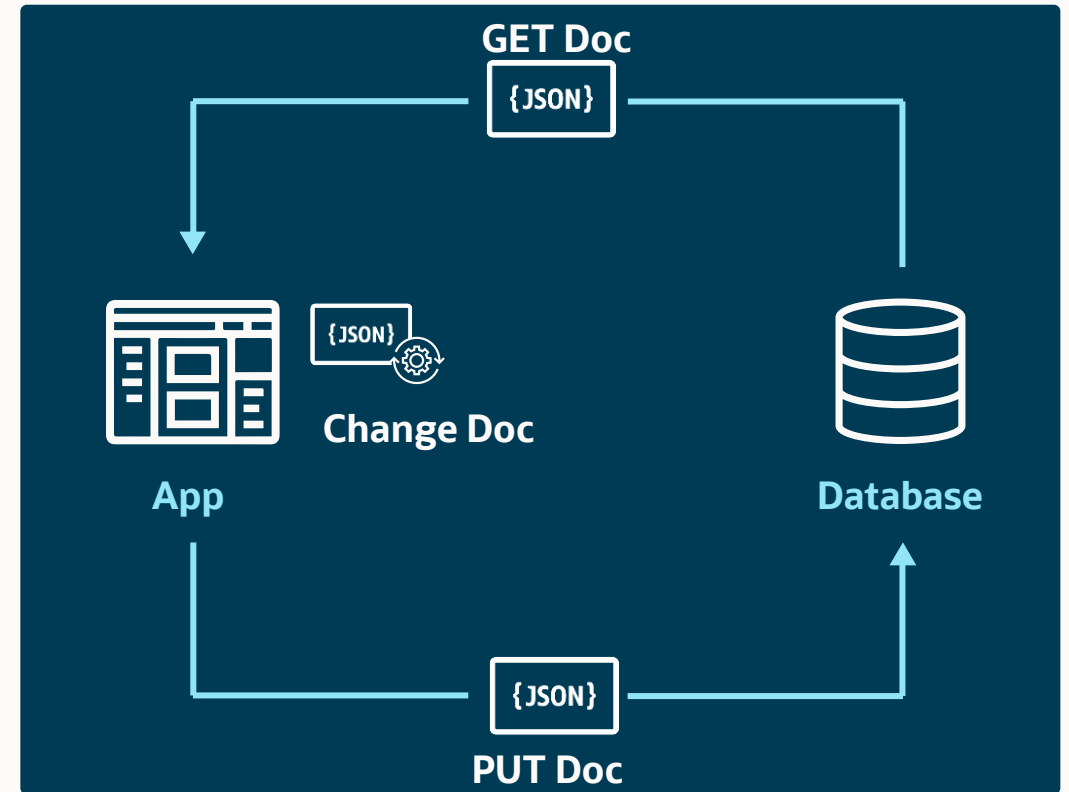
- GET a document from the View
- Make any changes needed to the document
- PUT the document back into the View



```
GET school.edu/student_schedule?q={"student":"Jill"}
```

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 have to worry about inconsistencies

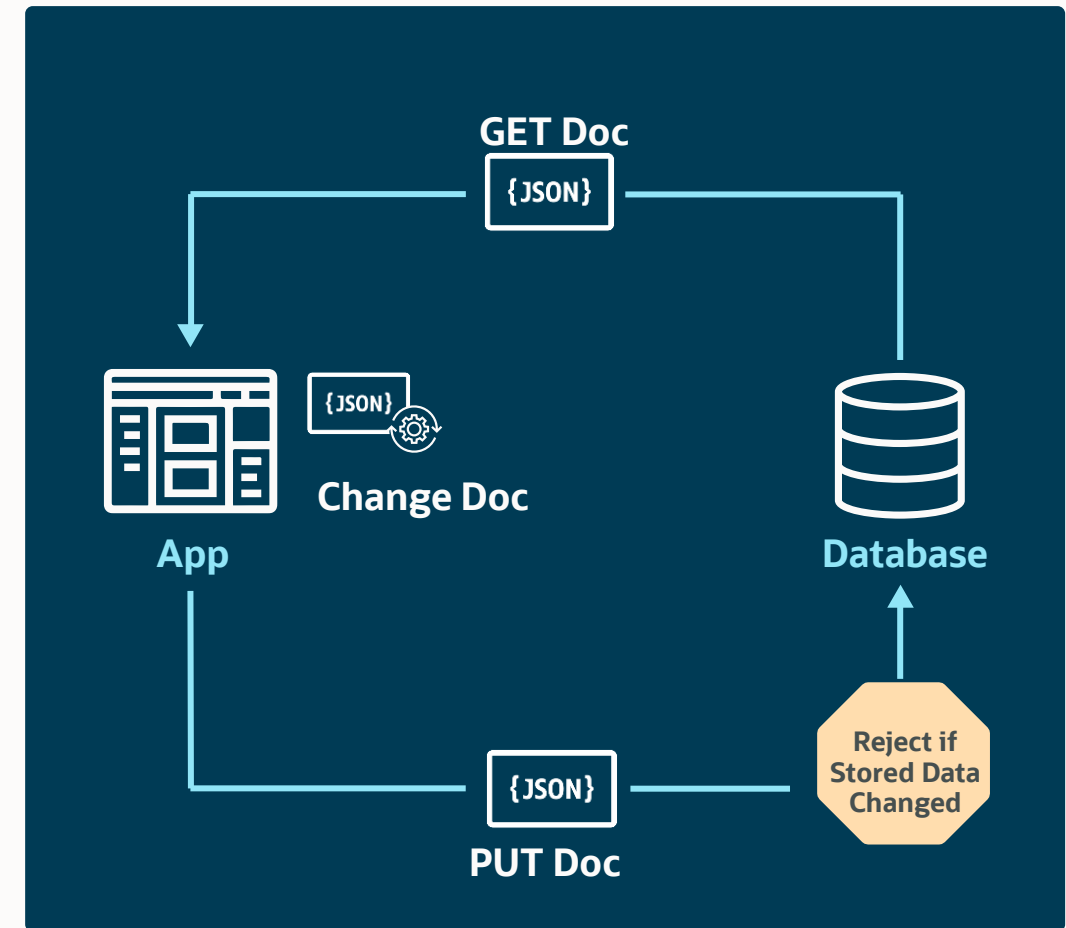


# Game Changing Lock-Free Concurrency Control

Conventional **locking** does not work when REST GET and PUT APIs are used

**Value-based concurrency control** - The database automatically detects when the database **data underlying a document** has changed between the initial document read and the subsequent write

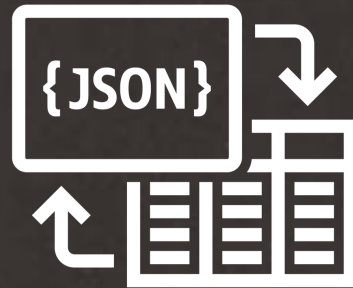
- If a change occurred, the write operation is automatically rejected and returns an error
  - The app can then reissue the write based on the changed data
1. Great for interactive applications since the data is not locked during human think time
  2. Great for mobile disconnected apps since writes of stale documents are rejected



# Key Take Aways

Paradigm shift in app development –  
Enabling extreme flexibility and simplicity for app dev

JSON Duality Views enable data to be transparently read and written as documents or as tables

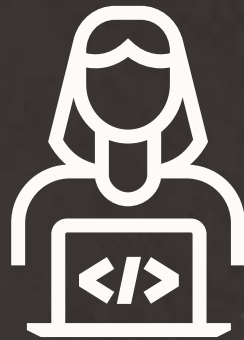


JSON Duality makes it easy to persist and operate on app objects as JSON, bridging the object relational mismatch

# Key Take Aways for Document Developers

JSON Duality delivers the development simplicity of JSON **plus** the use-case scalability of relational

Document schemas can be **specialized for each use case** without creating data duplication or consistency issues



Developers can easily build document-centric apps on existing relational data and use all the advanced features of Oracle's Converged Database

# Key Take Aways for Relational Developers

JSON Duality is more efficient, centralized, and consistent than an ORM

Enables simple access to all the rows needed for an app use case in one call and one round trip



Developers can easily create SQL-centric apps or analytics on data created by document-oriented apps

# Use LiveLabs to try out JSON Relational Duality

## **LiveLabs: JSON Relational Duality Views in Oracle Database 23c Free**

JSON Relational Duality converges the benefits of the relational and document worlds within a single database. Apps can access the same data as a set of JSON documents or as relational tables. We're offering several LiveLabs on this topic:

- **Exploring JSON Relational Duality Views in 23c Free with Java**
- **Exploring JSON Relational Duality Views in 23c Free using SQL**
- **AutoREST with JSON Relational Duality Views in 23c Free**





# Thank You

---

