ORACLE

JSON関連新機能

Oracle Database 23c新機能セミナー

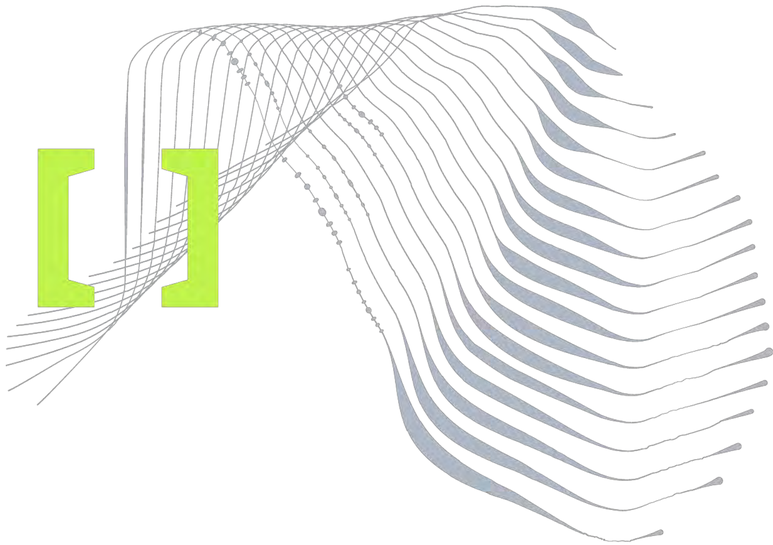
山川　薫

日本オラクル株式会社

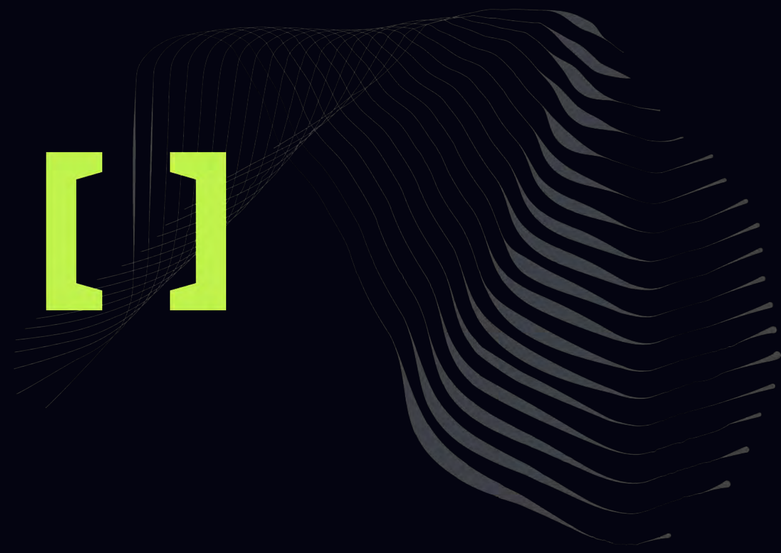
2023/10/20

Agenda

1. JSON Relational Duality（JSONリレーショナル二面性）
2. JSONスキーマ
3. JSON関連のその他の新機能



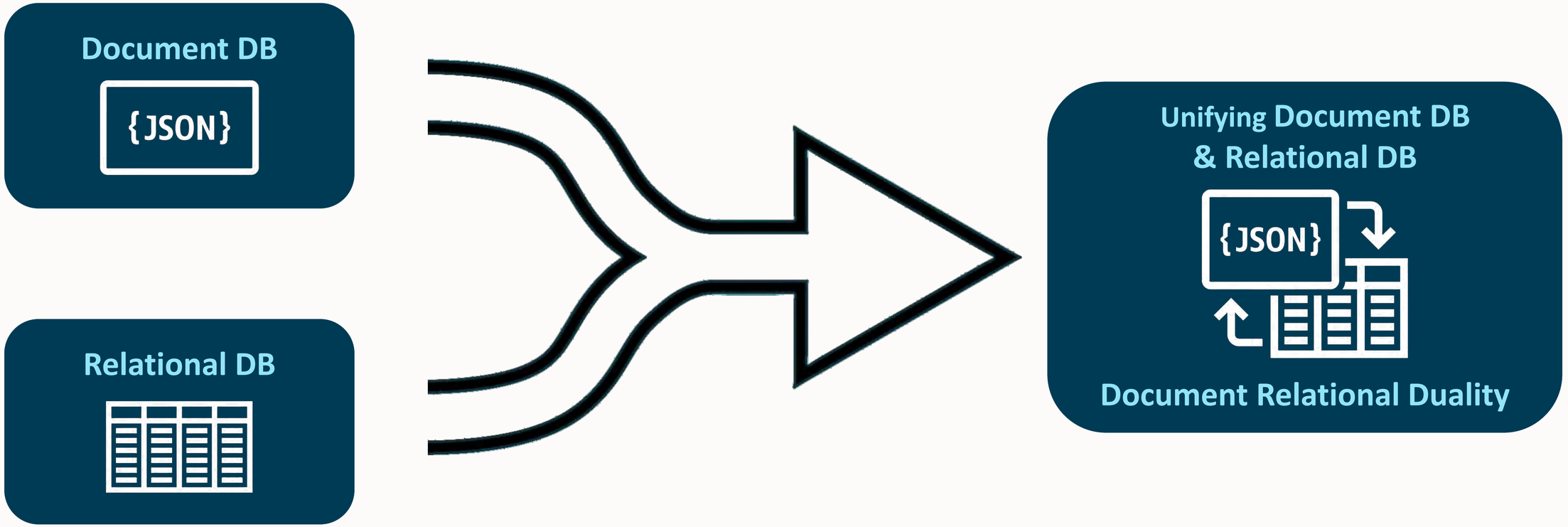
JSON Relational Duality
（JSONリレーショナル二面性）



JSON Relational Duality

アプリケーション開発におけるパラダイム・シフト

ドキュメントDBとリレーショナルDBを完全に統合
リレーショナルのすべての利点とJSONドキュメントのすべての利点の両方を享受可能



リレーショナルモデルとドキュメントモデルはそれぞれ利点がある

Relational

- データの重複なし（正規化）
- ユースケースの柔軟性
- 検索が容易
- データの一貫性

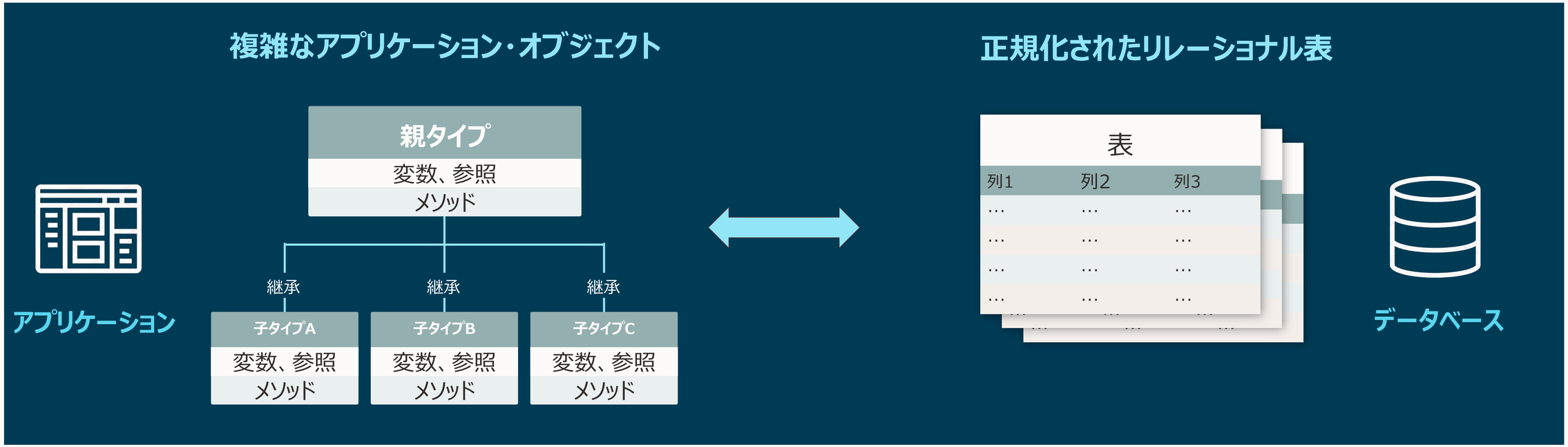
JSON

- アプリケーション操作に容易にマップ可能
- スキーマレスのアジャイル開発
- 階層型データフォーマット
- アプリケーションのデータ・クラスやオブジェクトへのマッピングが容易

→ 一方の利点はもう一方の欠点でもある

課題: アプリケーション・オブジェクトとリレーショナルのミスマッチ

アプリケーション・オブジェクトを正規化されたリレーショナル・データに変換するにはスキルと労力が必要



JSONドキュメント

JSONはそのシンプルさから、データ・アクセスおよびデータ交換形式として広く利用されている

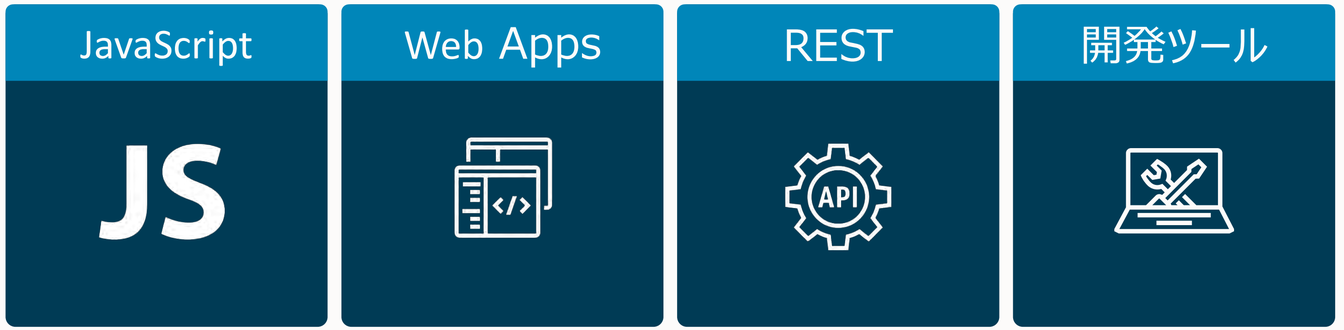
あらゆるデータに対応

JSONドキュメントは、複雑な言語オブジェクトを
名前と値のペアの単純な階層として
表現することが可能

```
{
 "name1" : "String Value1",
 "name2" :
    {

        "name3" : "14:00,
        "name4" :  1234
    }
}
```

送受信が簡単

自己記述型、自己完結型のフォーマットは、
変更に柔軟に適応し、システム間での
送受信が容易



Oracle DatabaseのJSONドキュメント対応はドキュメント・データベースよりも優れている

1

完全な**ドキュメントAPI**

-REST API
-Oracle Database API
 for MongoDB
-SODA API

2

標準ベースの**SQLによる**
ドキュメントへのアクセスを提供

JavaScript、JavaまたはPL/SQLで
記述された
ストアド・プロシージャ

ドキュメント・データベースより
優れている

3

ドキュメント間での完全な
ACID整合性

ドキュメントに対する分析、空間、
グラフ、ML、並列処理

ドキュメント・データベースより
優れている

SODA : Simple Oracle Document Access

ACID : Atomicity, Consistency, Isolation, Durability

Oracle Database API for MongoDB

アプリケーションにMongoDB互換のインターフェースを提供（MongoDB Shell、pyMongo等）

ORDSのインストールとOracle Database API for MongoDBの利用手順

1. ORDSのzipファイルのダウンロード（バージョン22.3以降）　ORDS : Oracle REST Data Services
   https://www.oracle.com/database/sqldeveloper/technologies/db-actions/download/
2. ORDSのインストールと構成
   `ords install`
3. Oracle Database API for MongoDBの有効化
   `ords config set mongo.enabled true`
4. ORDSの起動
   `ords serve`
5. スキーマに対してORDSを有効化
   `SQL > exec ORDS.ENABLE_SCHEMA;`
6. MongoDBクライアントからコマンドを実行
   `student_schedule.find({"name":"Jill"})`

サポートされるツール、ドライバ	サポートされる最低バージョン
C	1.19.0
C#	2.13.0
Compass	1.28.1
Database Tools	100.5.0
Go	1.6.0
Java	4.3.0
MongoSH	0.15.6
Node.js　driver	4.1.0
PyMongo	3.12.0
Ruby	2.16.0
Rust	2.1.0

マニュアル：Oracle REST Data Service Installation and Configuration Guide
Oracle Database API for MongoDB

Oracle Databaseは優れたドキュメント・データベースである

1998年からXMLドキュメントをサポート

2014年からJSONドキュメントをサポート

すべてのリリースで継続的に機能を強化

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

Oracle Databaseは優れたドキュメント・データベースである

Oracle DatabaseにおけるJSON関連機能の主な進化

- 12c Release 1
  JSONのサポートを開始（CLOB/BLOB/VARCHAR2列へのJSONの格納）
- 12c Release 2
  JSONデータ・ガイドのサポート（JSONデータからビューや仮想列を作成）
  JSONデータ生成のためのSQL/JSONファンクションを実装（JSON_OBJECT、JSON_ARRAY等）
- 18c
  JSONデータの格納および問合せ用の複数のSQL拡張機能を実装
- 19c（2019年）
  JSON_SERIALIZE（バイナリJSONデータのテキスト化、出力フォーマットの指定）をサポート
- 21c
  JSONデータ型（ネイティブ・バイナリ形式）をサポート
  Oracle Database API for MongoDBをサポート（ORDS 22.3以降）
- **23c**
  JSON Relational Dualityのサポート

Oracle Databaseは優れたドキュメント・データベースである

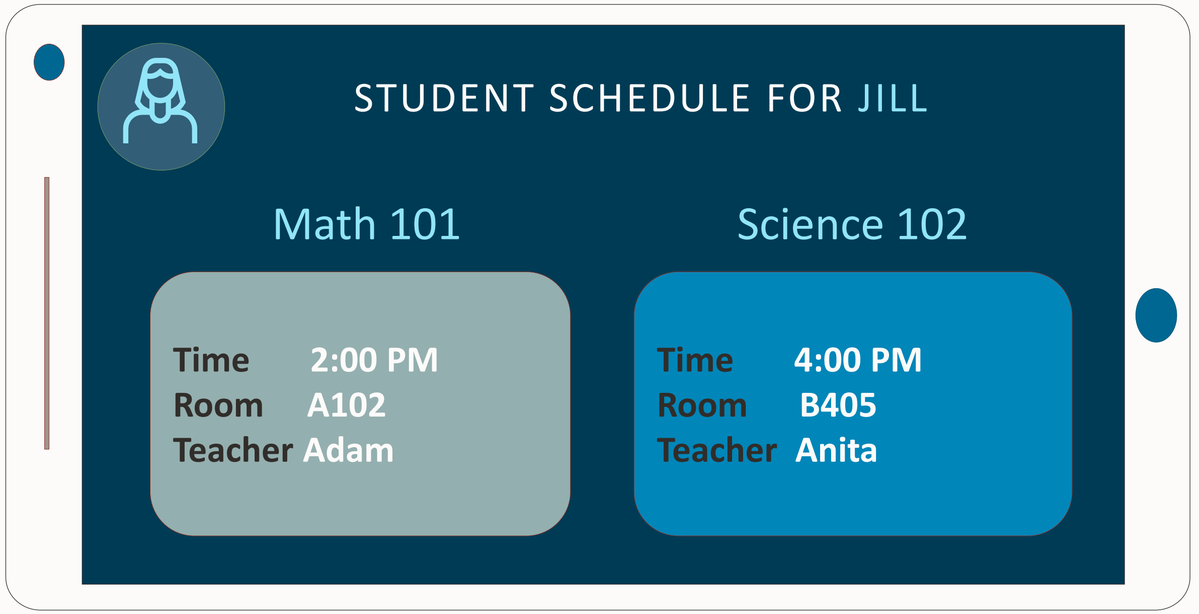
SQL APIまたはドキュメントAPIを使用して、
ドキュメントを簡単に格納、取得、分析および
操作が可能

- OracleはドキュメントのためのSQL拡張機能を
  オープンSQL標準に定期的に提供
- JSONドキュメントはJSON型（推奨）、
  CLOB型、BLOB型、VARCHAR2型の列に格納可能

```
CREATE TABLE student_schedule(
  student_id   number,
  schedule_doc JSON
);

SELECT schedule_doc
FROM   student_schedule s
WHERE  s.schedule_doc.student = 'Jill';
```

アプリケーション開発の例- 学生の受講スケジュール

学生の受講スケジュールを作成する
アプリケーションを開発すると想定



アプリケーション開発の例- 学生の受講スケジュール

アプリケーションに必要なデータは、
リレーショナル・スキーマ内の正規化された表に格納

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

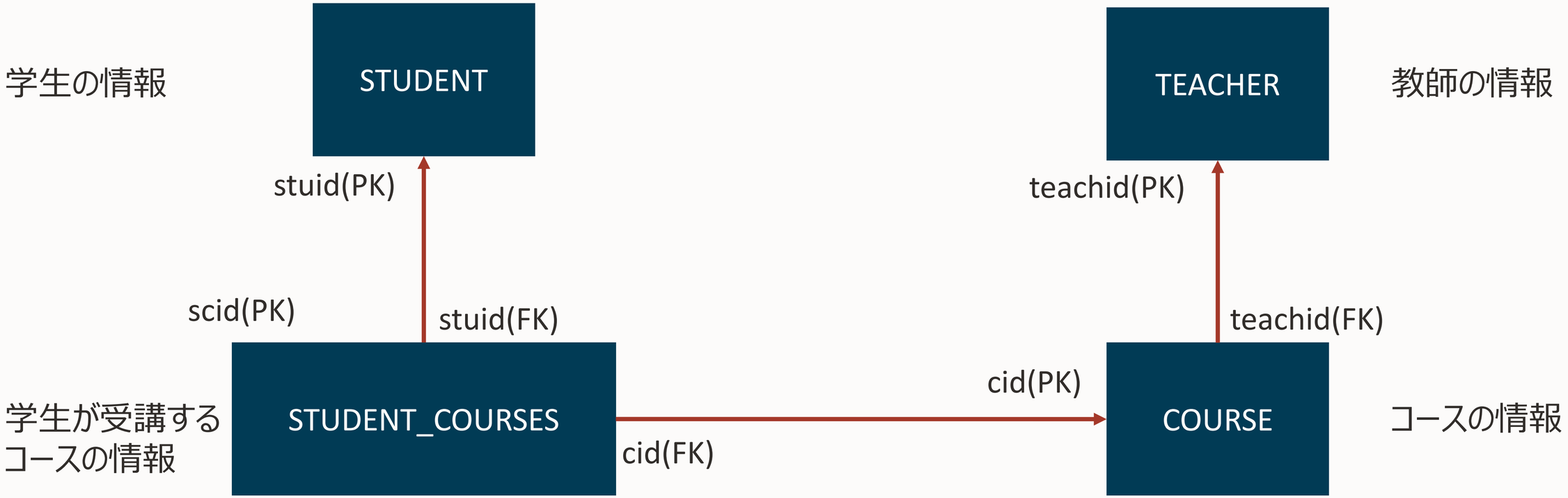
CID	CNAME	ROOM	TIME	TEACHID
C123	MA_01	A102	14:00	T543
C345	SCI_02	B405	16:00	T789
C567	HIS_02	A102	14:00	T612
C789	LA_01	A256	12:00	T543

STUDENT COURSES

SCID	STUID	CID
1	S3245	C123
2	S8524	C567
3	S3245	C345
4	S3409	C123

アプリケーション開発の例- 学生の受講スケジュール

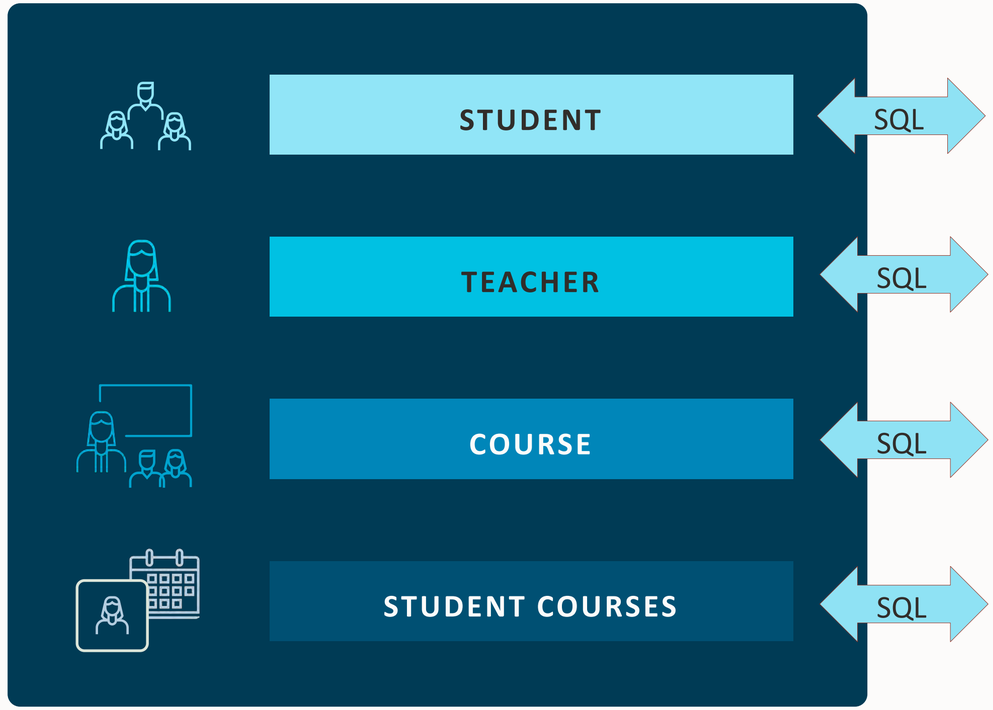
アプリケーションに必要なデータは、
リレーショナル・スキーマ内の正規化された表に格納



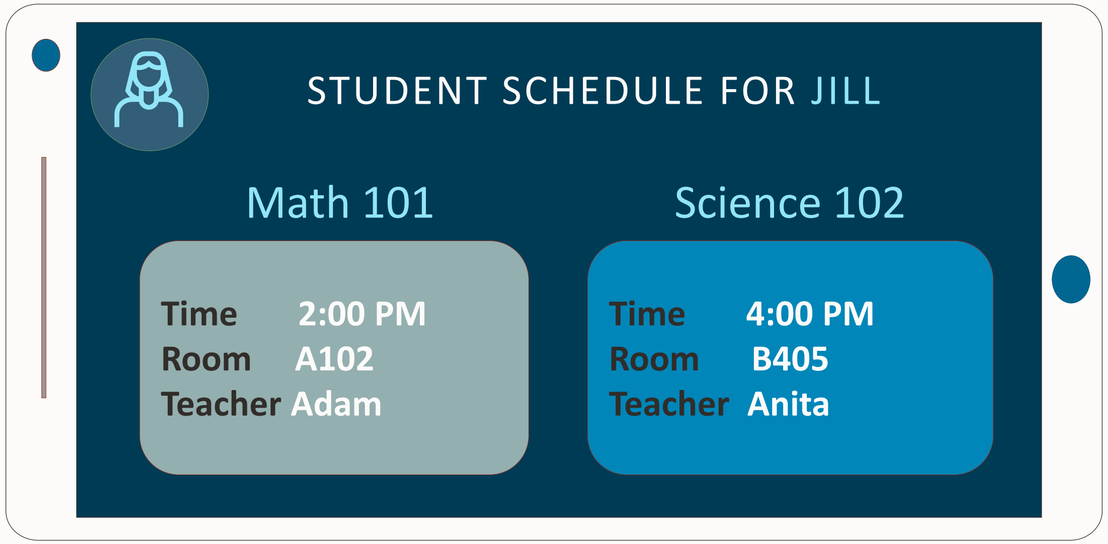
各PK：GENERATED BY DEFAULT ON NULL AS IDENTITY

アプリケーション開発の例- 学生の受講スケジュール

正規化された表を使用したアプリケーションの開発は
非常に柔軟だが、開発者にとっては必ずしも容易ではない

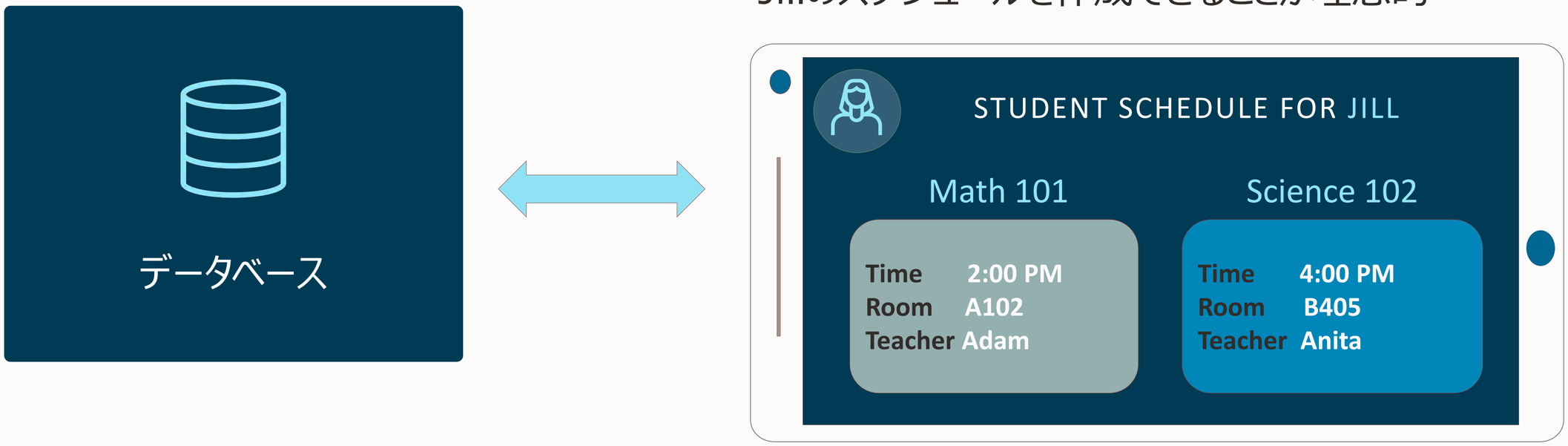


Jillのスケジュールを作成するには、開発者は4つの表それぞれに
対してデータベース操作を実行する必要がある

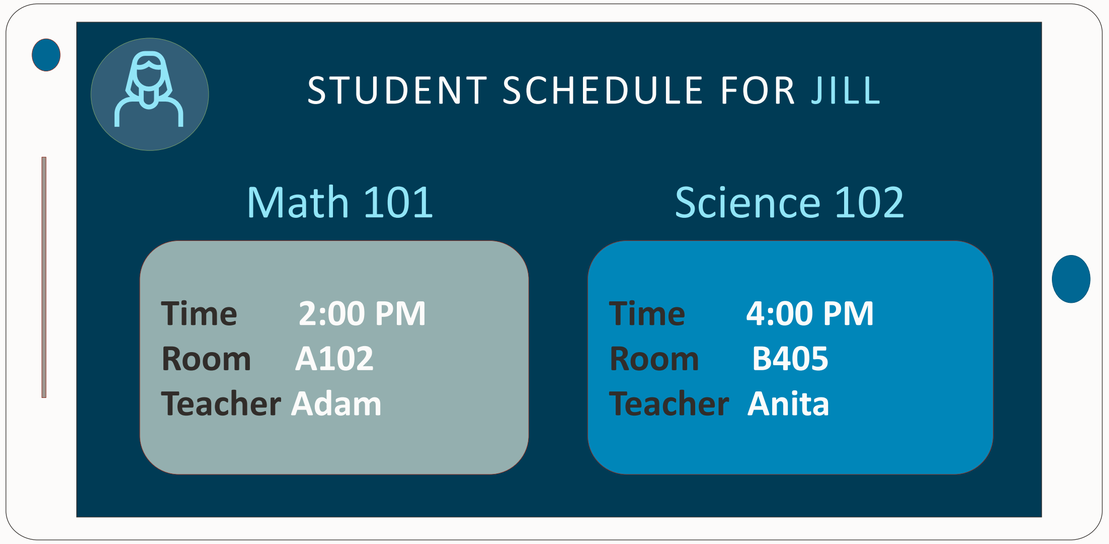


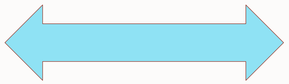
リレーショナル・データと開発者

開発者にとっては、単一のデータベース操作で
Jillのスケジュールを作成できることが理想的



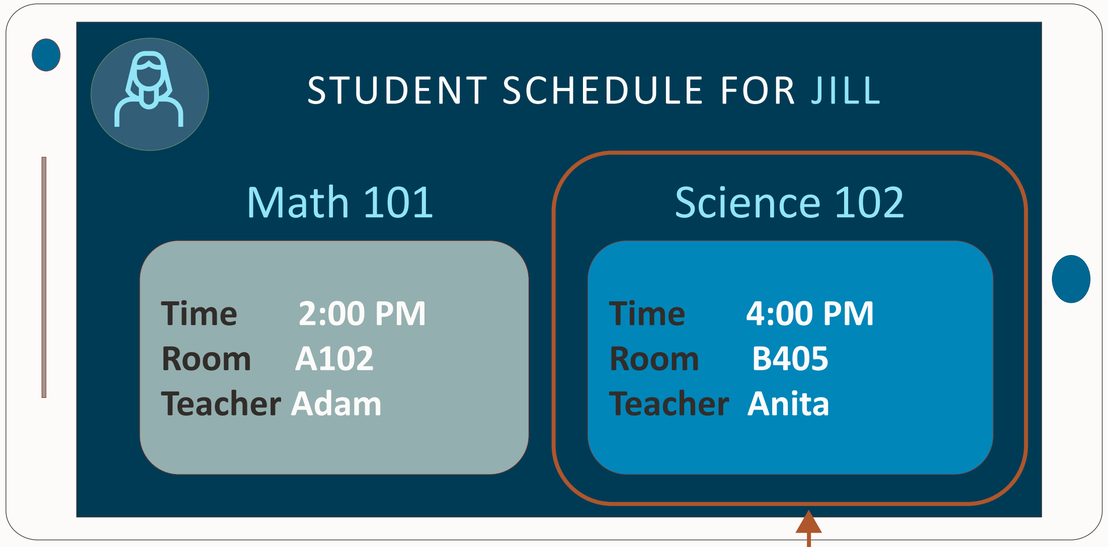
Jillの受講スケジュールはJSONでシンプルに表現可能





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

Jillの受講スケジュールはJSONでシンプルに表現可能



Jillが受講する各コースのデータは、
JillのJSONドキュメントに埋め込まれる

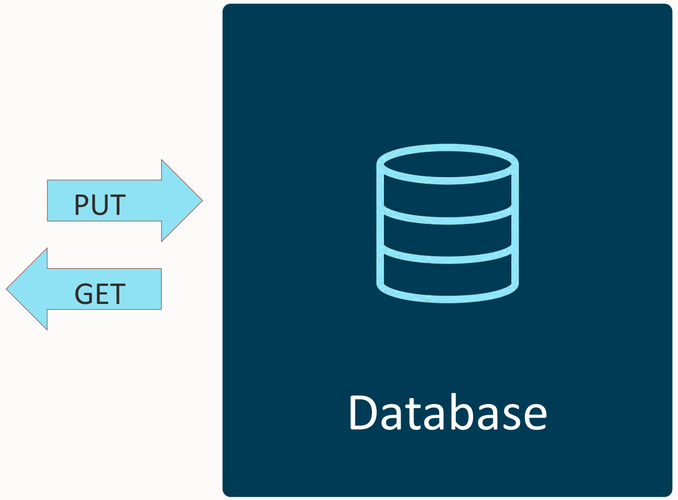
```
{
 "student"     : "Jill",
 "schedule "   :
   [
      {
       "time"    : "14:00",
       "course"  : "Math 101",
       "room"    : "A102",
       "teacher" : "Adam"
      },
      {
       "time"    : "16:00",
       "course"  : "Science 102",
       "room"    : "B405",
       "teacher" : "Anita"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

ドキュメント・データベースはJSONの操作を簡単にする

ドキュメント・データベースは、
JSONドキュメントへのアクセス、
JSONドキュメントの格納が容易

単純なGET/PUT APIの使用

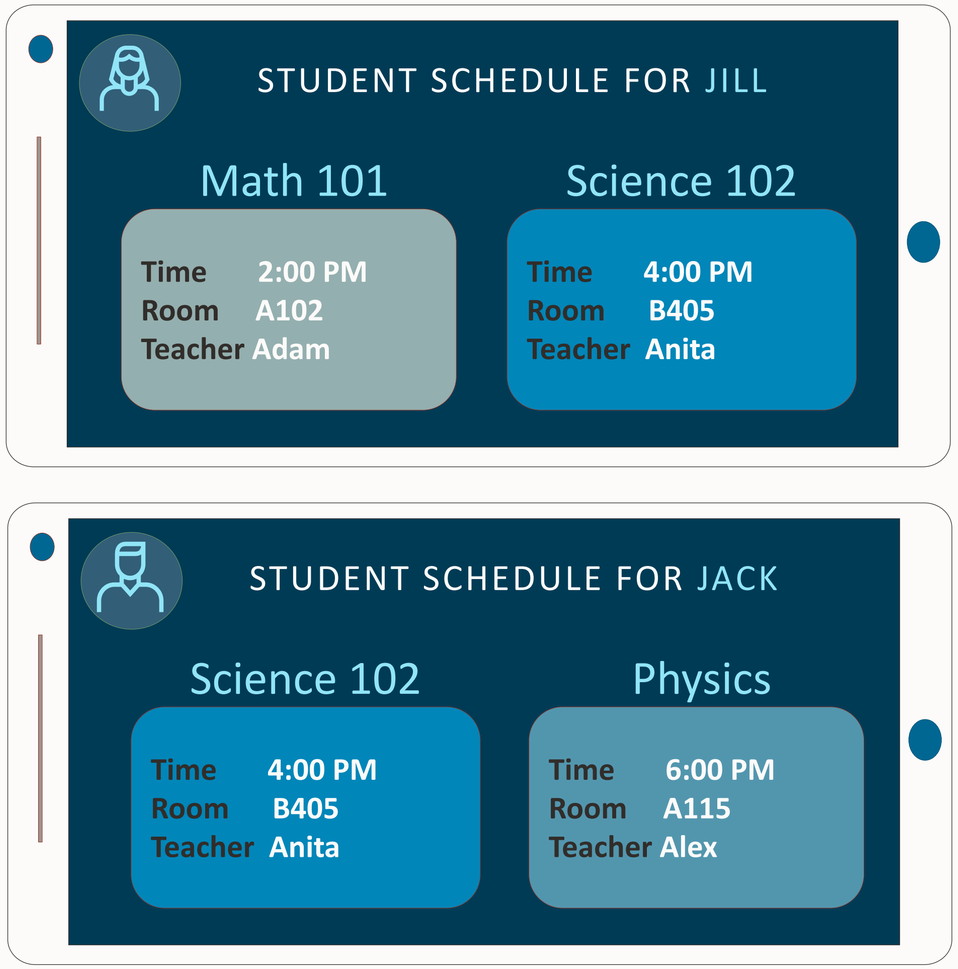
```
{
 "student"     : "Jill",
 "schedule "   :
   [
      {
       "time"    : "14:00",
       "course"  : "Math 101",
       "room"    : "A102",
       "teacher" : "Adam"
      },
      {
       "time"    : "16:00",
       "course"  : "Science 102",
       "room"    : "B405",
       "teacher" : "Anita"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



ドキュメント・データベースの課題

JSONドキュメントは、アプリケーションが
データのアクセス形式として使用するのが簡単

データのストレージ形式として使用すると、
データの重複とデータの一貫性に関する
問題が発生

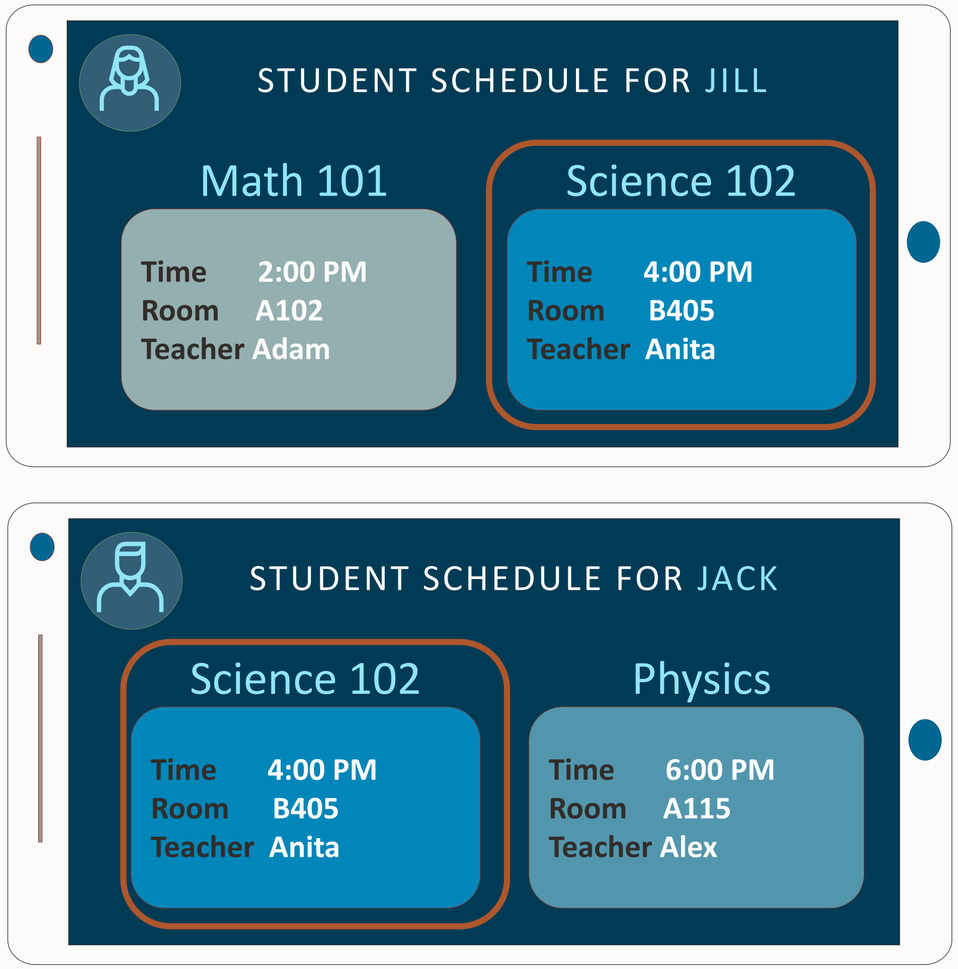


ドキュメント・データベースの制限

JSONを使用して学生の受講スケジュールを
保存すると、コースおよび教師の情報が
各学生のスケジュールに重複して格納される

データの重複によって

- データの保存が非効率
- データの更新コストが増大
- データの一貫性を保つことが困難



ドキュメント・データベースの制限

JSONを使用して学生の受講スケジュールを保存すると、コースおよび教師の情報が各学生のスケジュールに重複して格納される

データの重複によって

- データの保存が非効率
- データの更新コストが増大
- データの一貫性を保つことが困難

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
      }
   ]
}
```

STUDENT SCHEDULE FOR: JACK

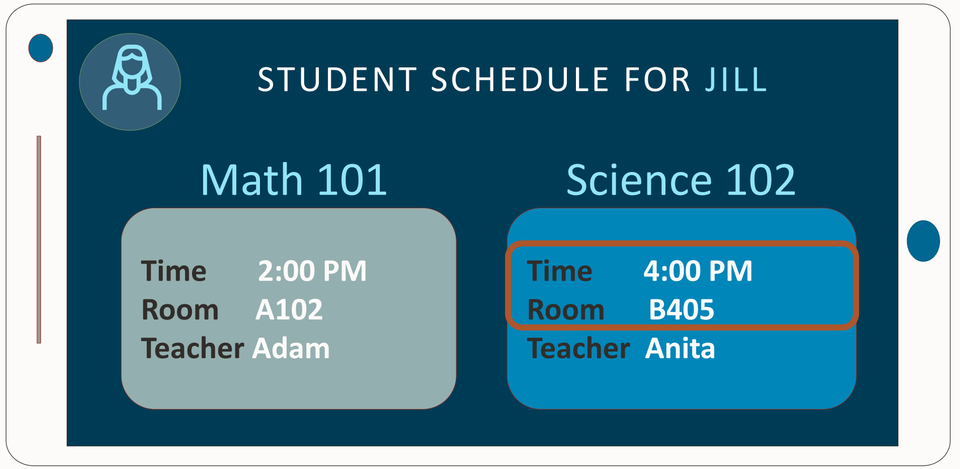
```
{
 "student"      : "S4356",
 "name"         : "Jack",
 "schedule "    :
   [
      {
        "time"     : "16:00",
        "course"   : "Science 102",
        "room"     : ”B105",
        "teacher"  : "Anita
      }
      {
        "time"     : "14:00",
        "course"   : ”Physics",
        "room"     : ”B405",
        "teacher"  : "Anita
      }
   ]
}
```

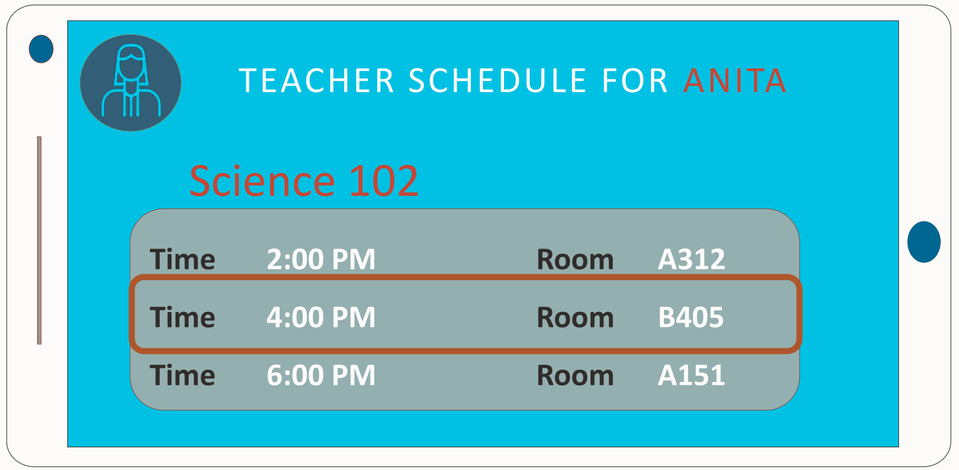
複数のユース・ケースで同じデータを利用する場合

新しいユース・ケースで同じデータが使用されると、
データの重複が増加する

例：教師の講義スケジュールのユース・ケースを追加

- 教師をルートとする新しいドキュメントが必要
- ただし、学生のドキュメントと同じコース・データを共有





複数のユースケースで同じデータを利用する場合

データが重複して格納されるため、
コースの教室を変更するには、
多数の学生の受講スケジュールのドキュメントを
一度に更新する必要がある

教師の講義スケジュールのドキュメントも
同時に更新する必要がある

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"     : "B405",
        "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"
      }
   ]
}
```

あらゆるユース・ケースで両方のメリットを享受することは可能か

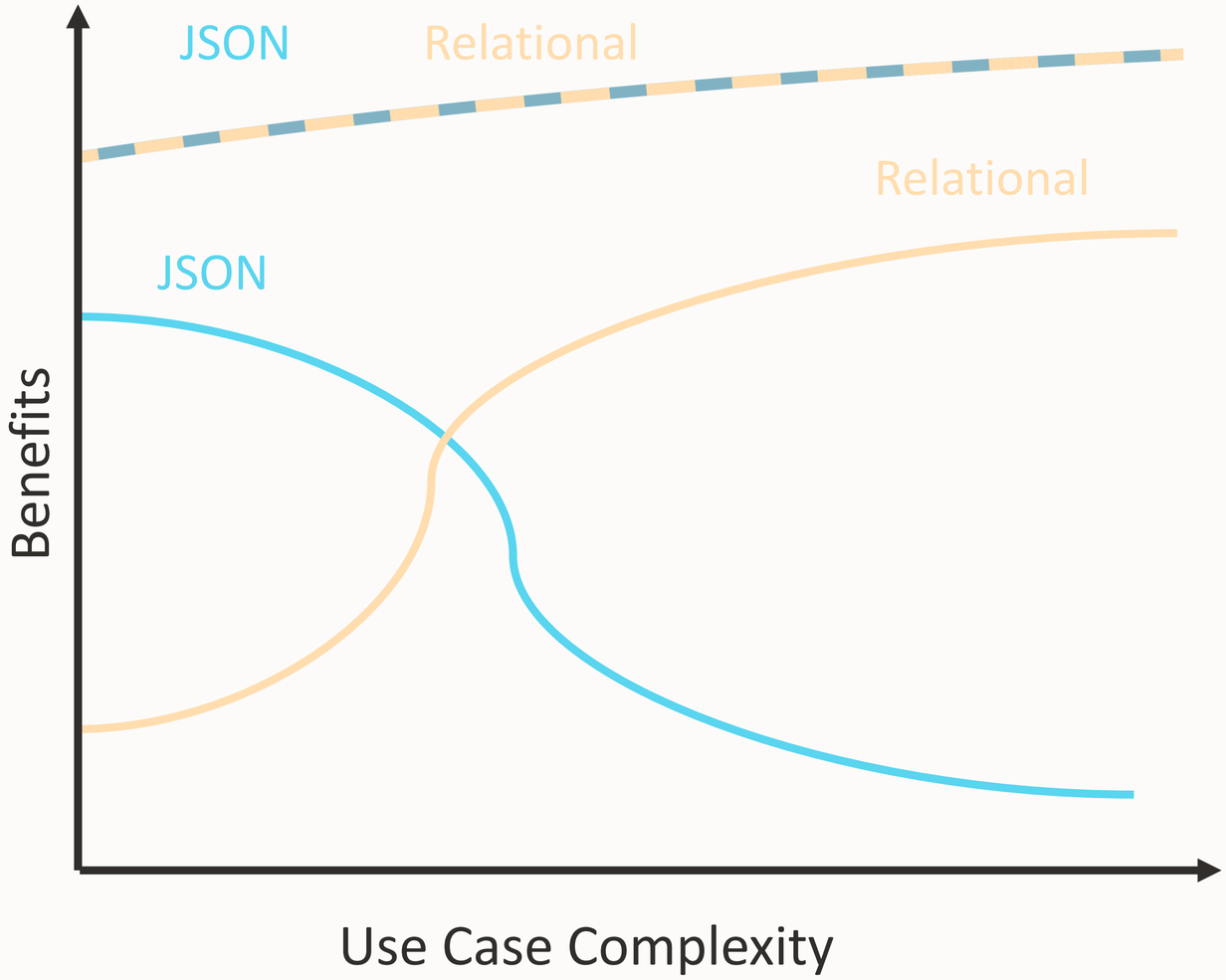
リレーショナル

- ユースケースの柔軟性
- 問合せ可能性
- 一貫性
- ストレージ効率

＋

ドキュメント

- 言語の型への簡単なマッピング
- 俊敏なスキーマレス開発
- 階層データ形式
- 標準データ交換フォーマット

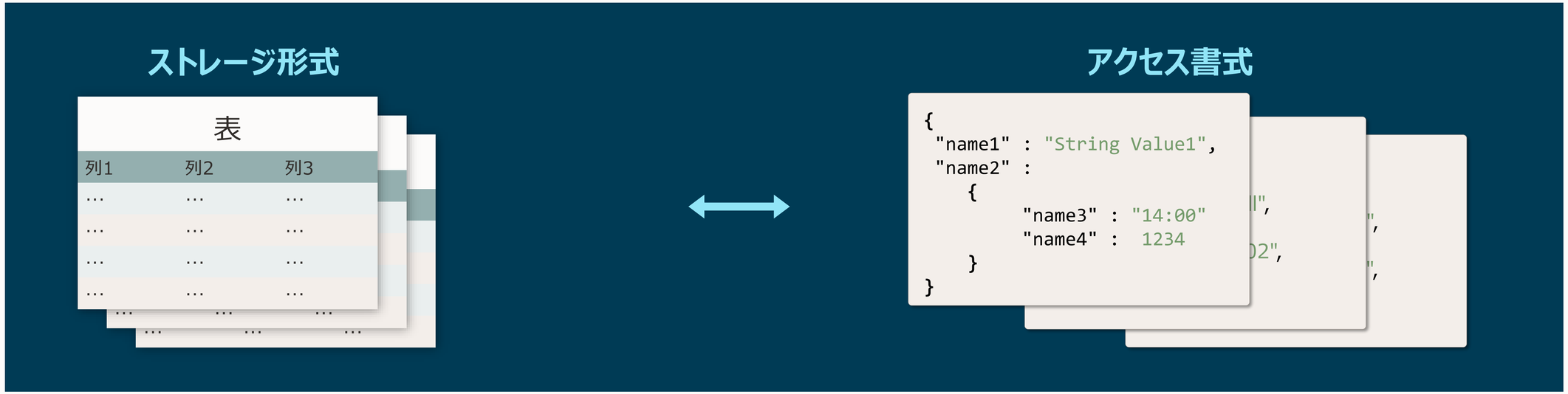


JSON Relational Duality

JSON Relational Dualityとは

リレーショナル・モデルおよびSQLアクセスの利点を提供するために、データを表に**行として**格納

データにJSONドキュメントとしてアクセスし、ドキュメント・アプリケーションのシンプルさを実現



JSON Relational Dualityビューを使用したアプリ開発の簡素化

- Dualityビューは、テーブルやビューなどと同じデータベース・オブジェクト
  - CREATE JSON DUALITY VIEW文
  - 個別のマッピング言語やツール、プログラミング、デプロイ、設定が不要
- ドキュメントレベルの一貫性とテーブルの行レベルの一貫性が同時に保証され、どちらか一方への変更は透過的かつ即座にもう一方に反映される
  - 複数のアプリケーションによってドキュメントやその基礎となるテーブルを同時に更新することも可能
  - 既存のSQLツールは、アプリケーションがそれらの行に基づくドキュメントを更新するのと同時に、テーブルの行を更新することが可能
- ドキュメントの一部を処理するルールを、アプリケーションのコードではなくデータベースで一元的に定義、アプリケーションや言語を問わず同じルールが適用される
- どのデータベース機能でも、どのアプリケーションでも利用可能

JSON Relational Dualityビューを使用したアプリ開発の簡素化

Duality ビューは..

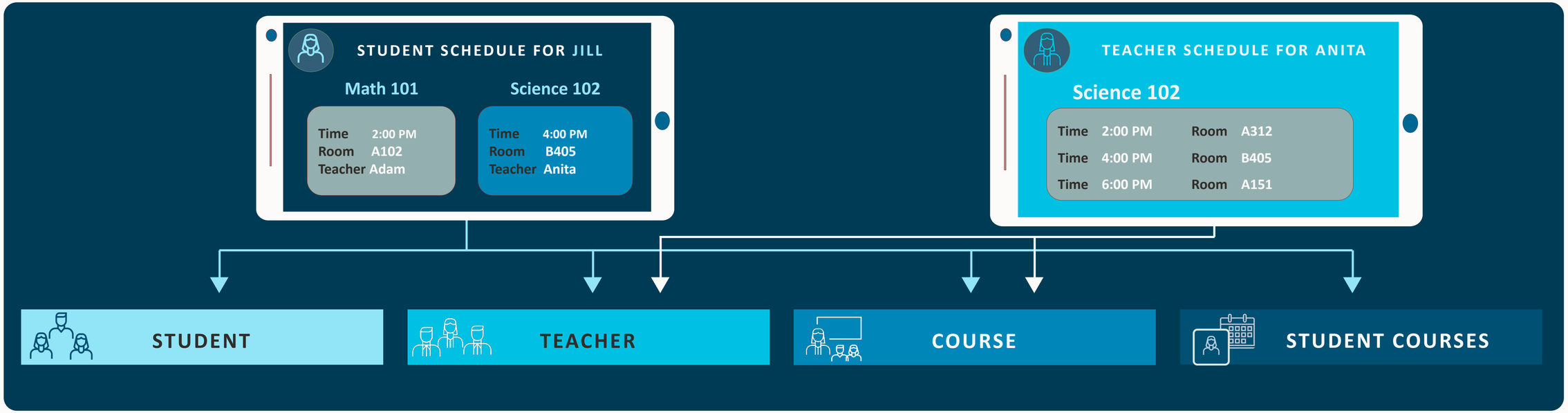
- ✓ 言語に依存しない
- ✓ データベースによって最適化される
- ✓ ユースケースで必要なすべてのデータを1往復で取得
- ✓ オブジェクト・マッピングをDBに集約
- ✓ より良い同時実行制御を提供

ユース・ケースに特化したドキュメントの生成

JSON Relational Dualityはユース・ケースに特化したドキュメントを生成する画期的な方法

- アプリケーションのユース・ケースごとに、そのケースに特化した JSON ドキュメントを生成する Dualityビューを作成可能
- データの重複や一貫性の問題を発生させることなく、同じデータに対して新しいユース・ケースを簡単に追加可能

例：同じ行データを使用して、生徒のスケジュールと教師のスケジュールのドキュメントを作成することが可能



JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

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

GraphQL構文を使用してstudent_schedule Dualityビューを宣言する方法の例

ビューの構造が必要なJSONオブジェクトの構造を反映

STUDENT SCHEDULE : JILL

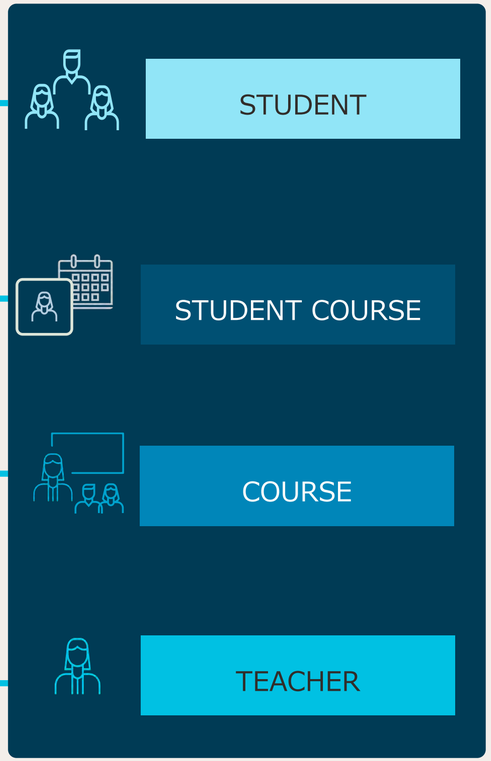
```
{
 ….
   [  {
                ….
      },
      {
                ..
      }
   ]
}
```

JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

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

JSONドキュメントを構成する
データを含む表を指定

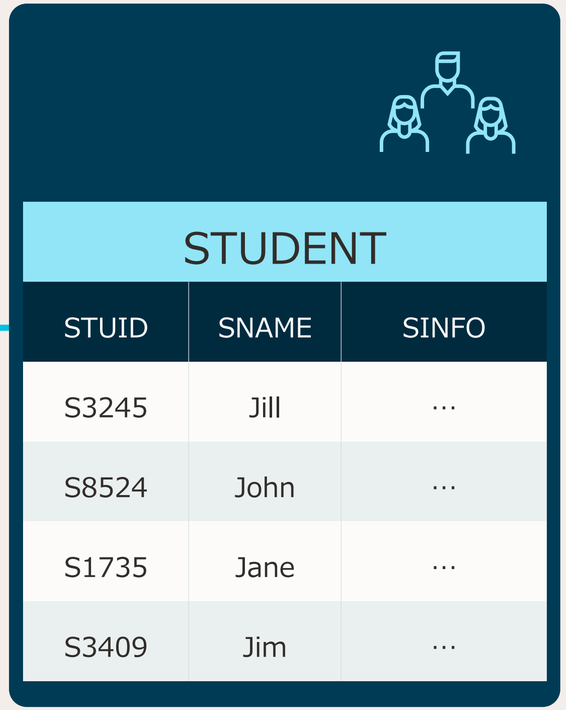


JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

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

各JSONプロパティの名前および
その値を取得するための表の列名を指定



JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

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

更新可能性ルールの指定
(Dualityビュー student_schedule経由で
student_coursesを更新可能、
student、course、teacherは更新不可)

JSON Relational Dualityの定義

JSON Relational Dualityビューの更新可能性制御

- デフォルトではDuality ViewはRead Only

- アノテーション（@：GraphQL、WITH：SQL）で更新（変更）可否を制御
  - INSERT/NOINSERT（表レベルで指定可能）
  - DELETE/NODELETE（表レベルで指定可能）
  - UPDATE/NOUPDATE（表レベルおよび列レベルで指定可能、列レベルの指定が優先）

- Duality Viewのルート表のINSERT/DELETEが指定されている場合、ドキュメントのINSERT/DELETEが可能
- Duality Viewの定義の中（表または列）でUPDATEが指定されている場合は、ドキュメントのUPDATEが可能

JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

更新不可のプロパティを更新しようとした場合：

```
ERROR at line 1:
ORA-40937: Cannot insert into table 'TEACHER' in JSON Relational Duality View
'STUDENT_SCHEDULE': Missing INSERT annotation or NOINSERT annotation specified.
Help: https://docs.oracle.com/error-help/db/ora-40937/
```

```
ERROR at line 1:
ORA-40940: Cannot update field 'teacher' corresponding to column 'TNAME' of
table 'TEACHER' in JSON Relational Duality View 'STUDENT_SCHEDULE': Missing
UPDATE annotation or NOUPDATE annotation specified.
Help: https://docs.oracle.com/error-help/db/ora-40940/
```

JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

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

ネストされたオブジェクトのプロパティを
親プロパティ内でネストしない場合に指定

JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

@unnest 指定なし

```
"schedule" :
[
  {
    "scid" : 1,
    "course" :
      {
        "time" : "14:00",
        "course" : "MA_01",
        "course_id" : "C123",
        "room" : "A102",
        "teacher" :
          {
            "teacher" : "Adam",
            "teacher_id" : "T543"
          }
      }
  }
]
```

@unnest 指定あり

```
"schedule" :
[
  {
     "scid" : 1,
     "time" : "14:00",
     "course" : "MA_01",
     "course_id" : "C123",
     "room" : "A102",
     "teacher" : "Adam",
     "teacher_id" : "T543"
    }
]
```

JSON Relational Dualityの定義

JSON Relational Dualityビューは、正規化された行をJSONドキュメントに組み込むためのレシピを宣言

```
CREATE OR REPLACE JSON DUALITY VIEW student_schedule AS
SELECT JSON { 'name': s.sname,
              'student_id': s.stuid,
              'schedule':
                [ (SELECT JSON { 'scid': sc.scid,
                  UNNEST
                    (SELECT JSON { 'time': c.time,
                                   'course': c.cname,
                                   'course_id':c.cid,
                                   'room':c.room,
                                    UNNEST
                                      (SELECT JSON { 'teacher':t.tname,
                                                     'teacher_id':t.teachid }
                                       FROM teacher t
                                       WHERE c.teachid = t.teachid )}
                      FROM course c
                      WHERE c.cid = sc.cid )}
                   FROM student_courses sc WITH INSERT UPDATE DELETE
                   WHERE s.stuid = sc.stuid )]
            }
FROM student s;
```

SQL構文を使用してstudent_schedule Dualityビューを宣言する方法の例

Dualityビューの使用

学生のスケジュールのDualityビューを検索すると、ベースとなる表にアクセスし、Jillのスケジュールを JSONドキュメントとして返す

- ドキュメントにはユースケースに必要な すべてのデータが含まれている
- データの更新に必要なID

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",
      }
    ]
}
```

Dualityビューの使用

Tips

Dualityビューのベースとなる表

- 適切なPRIMARY KEY（PK）、UNIQUE KEY、FOREIGN KEY（FK）が設定されている必要がある
- 検索パフォーマンス向上のためFOREIGN KEYが設定されている列にはインデックスを作成する

Dualityビューで使用可能な列の型

- VARCHAR2
- NVARCHAR2
- CHAR
- NCHAR
- NUMBER
- BINARY_DOUBLE
- BINARY_FLOAT
- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- BOOLEAN
- JSON
- BLOB
- CLOB
- NCLOB
- RAW

Dualityビューの使用例

SQLまたはドキュメントAPIを使用して
Dualityビューにアクセス可能

＜SQL＞：JSON型のdata列にアクセス

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

＜MongoDB API＞：ビューと同名のコレクションとしてアクセス

```
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",
      }
    ]
}
```

Dualityビューの使用例

Dualityビューの構造

- Dualityビューは単一のJSON型の列（data）のみを持つ

```
SQL> desc student_schedule
 Name                          Null?    Type
 ----------------------------- -------- ---------------
 DATA                                   JSON
```

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

Dualityビューの使用例

Dualityビューに対する検索

```
SQL> SELECT JSON_SERIALIZE(data PRETTY) json_data
  2  FROM   student_schedule s
  3  WHERE  s.data.name = 'Jill';

JSON_DATA
------------------------------------------------------------
{
  "_metadata" :
  {
    "etag" : "D341258911E06C5956538288BAC4CD91",
    "asof" : "00000000002A5096"
  },
  "name" : "Jill",
  "student_id" : "S3245",
  "schedule" :
  [
    {
      "scid" : 1,
      "time" : "14:00",
      "course" : "MA_01",
      "course_id" : "C123",
      "room" : "A102",
      "teacher" : "Adam",
      "teacher_id" : "T543"
    },
                                  ⋮
```

メタデータ

Dualityビュー定義に基づく
JSONデータ

Dualityビューの使用例

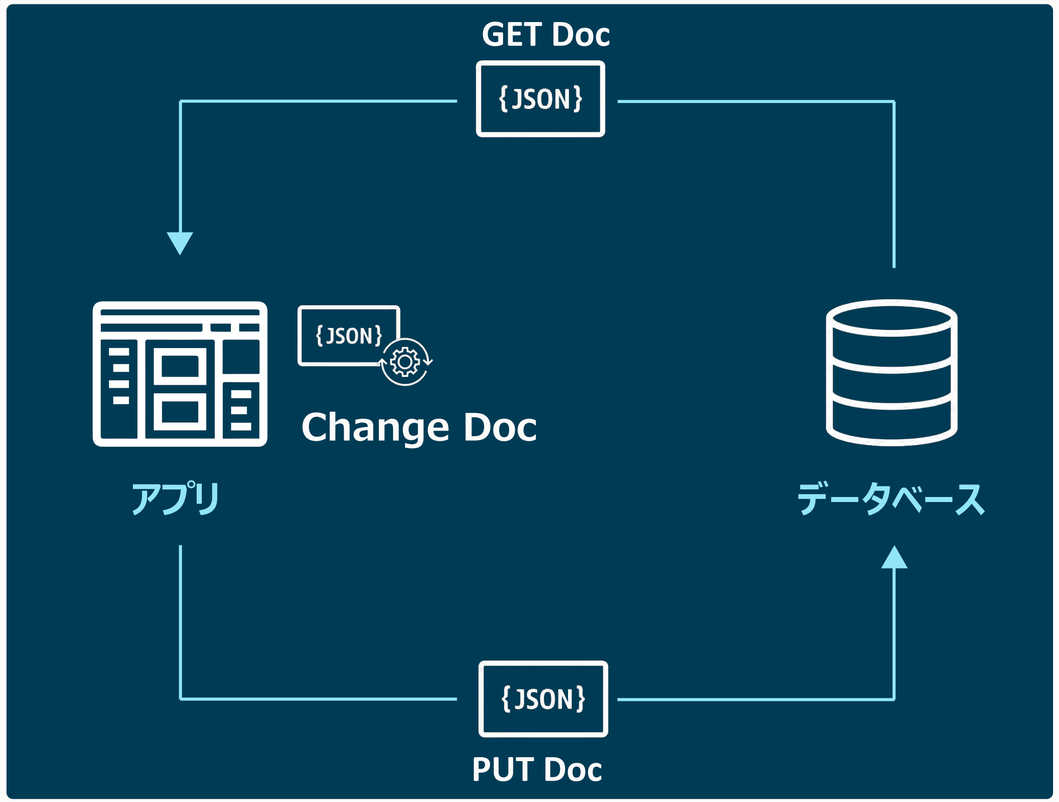
AUTO REST

Dualityビューは、RESTを使用して非常に簡単にアクセス可能

- Dualityビューからのドキュメントの取得（GET）
- GETしたドキュメントに必要な変更を加える
- Dualityビューへのドキュメントの書き戻し（PUT）

データベースは、新しいドキュメントの変更を自動的に検出し、ビューのベースとなる表の行を変更

- 同じデータを共有するすべてのDualityビューに変更が即時に反映される
- 開発者は不整合を心配する必要がない



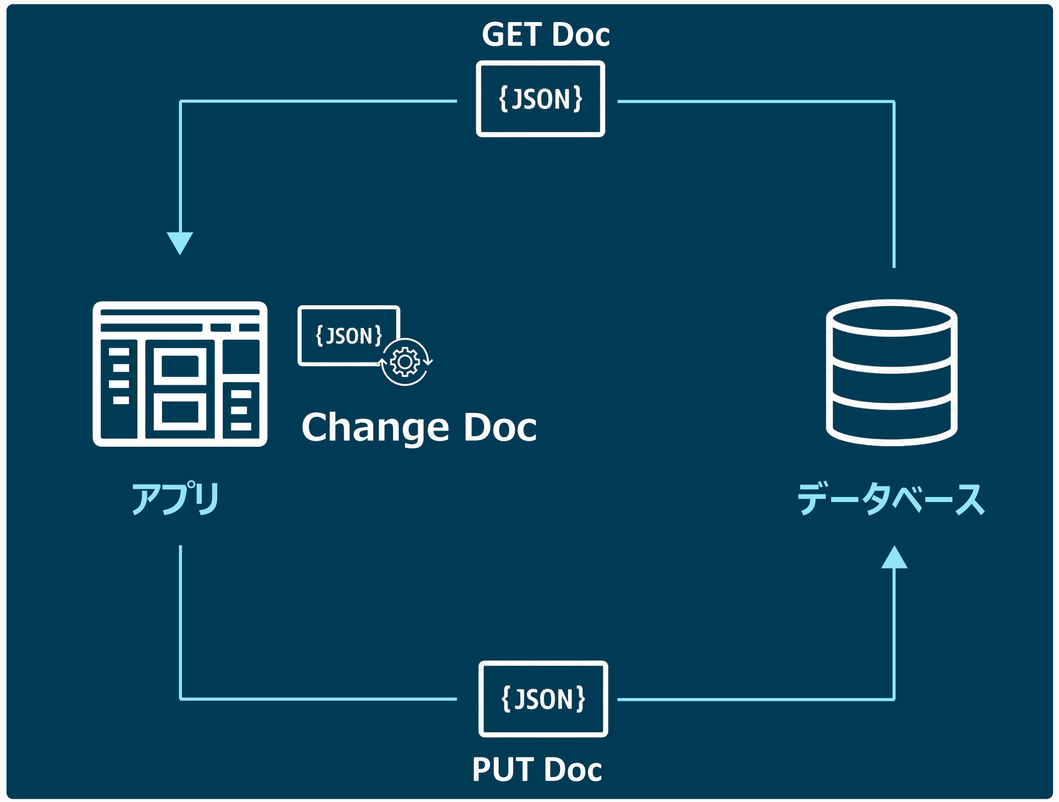

```
GET school.edu/student_schedule?q={"student":"Jill"}
```

Dualityビューの使用例

API

RESTではなくAPIを使用するアプリケーションは、
以下のAPIを利用可能

- Simple Oracle Document Access API(SODA)
- MongoDB互換API
  （Oracle Database API for MongoDB）



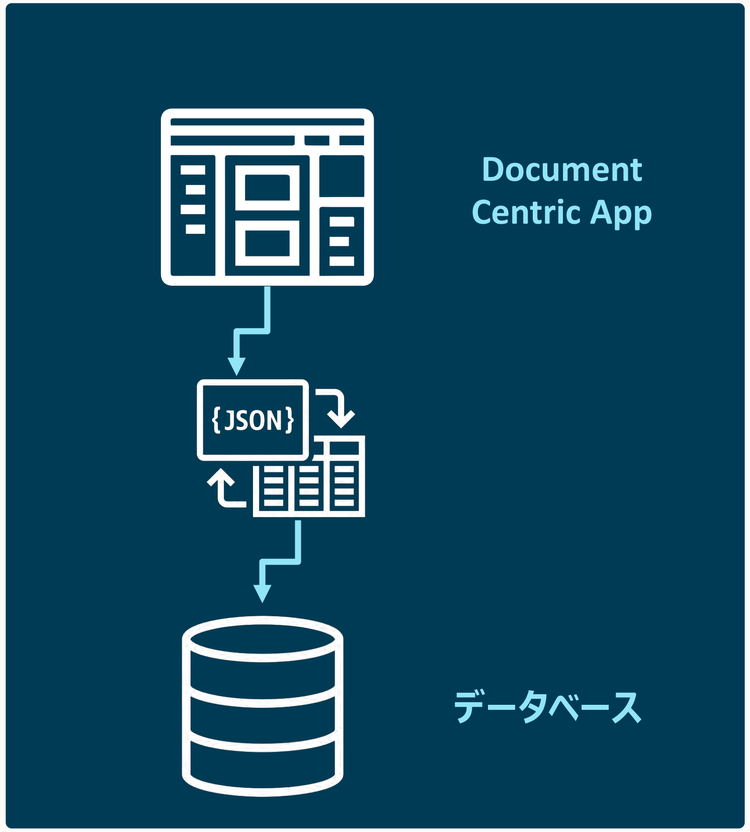
きわめて高い柔軟性

開発者はドキュメント中心のアプリを簡単に構築可能

- 新しいドキュメントをリレーショナル・データとして格納
- 既存のリレーショナル・データをドキュメントとして操作
- リレーショナル・データベースの上にJSONベースのマイクロサービスを構築

さらに、

Oracleのコンバージド・データベースのすべてのメリットを享受可能

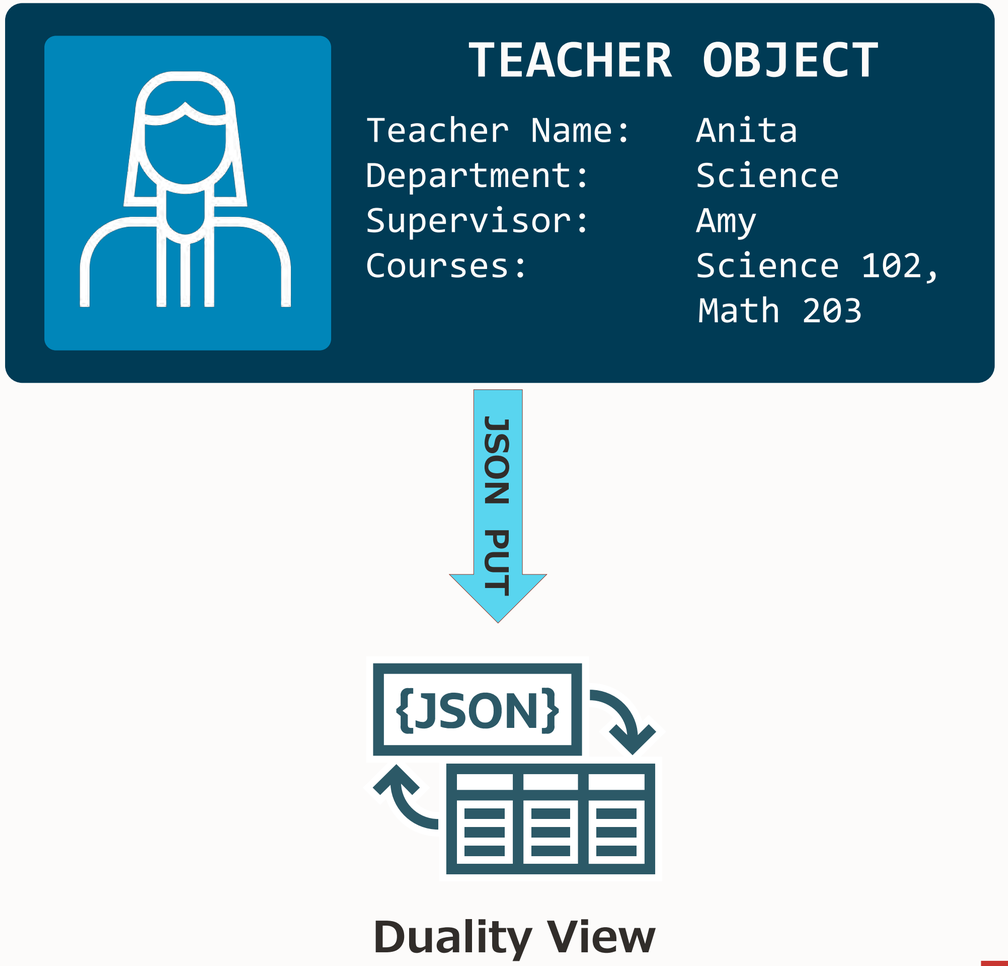


シンプルなアプリケーション・オブジェクトの永続性

アプリケーション・オブジェクトを永続化するためには

- 開発者は言語固有のオブジェクト表現をJSONに変換するだけ
- あとはデータベースがJSON Dualityビューを使用して、自動的にJSONを表に永続化

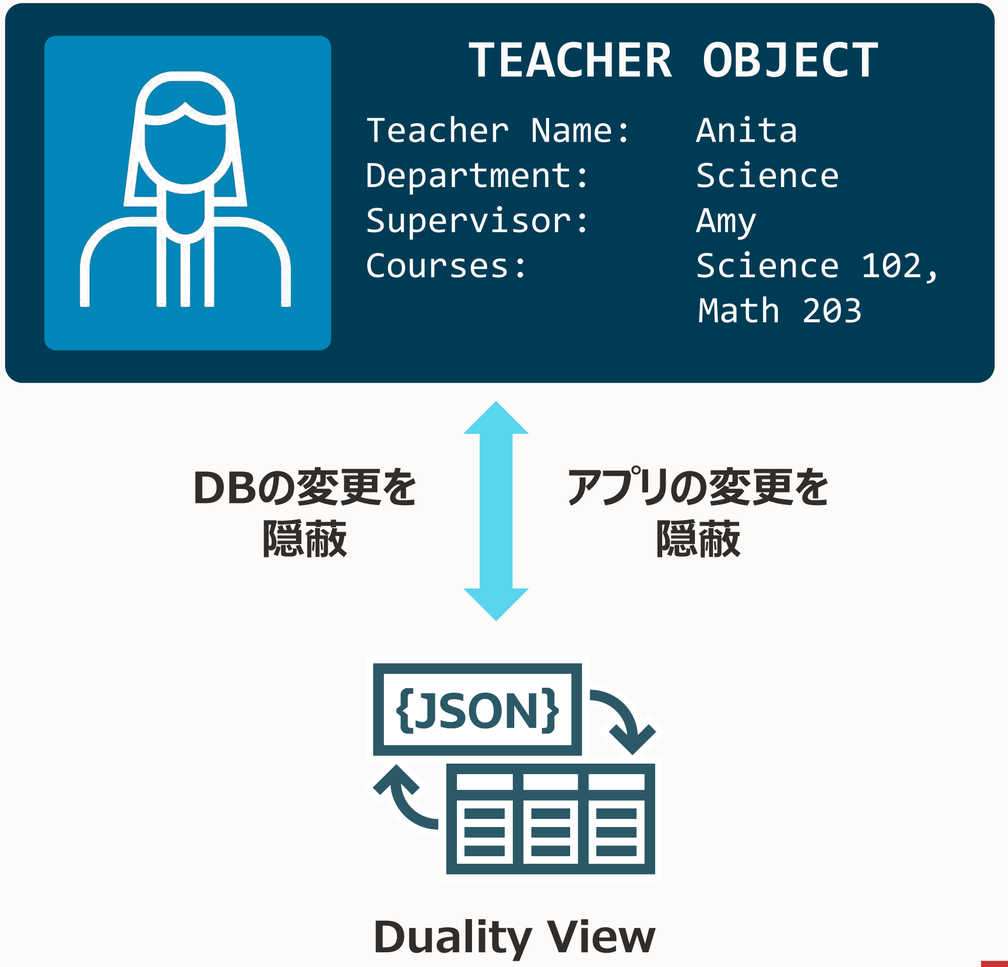
JSONへの変換は、REST、JavaScriptなどのコーディング時に使用されることが多いため、開発者にとっては使いやすく簡単



透過的なスキーマの変更

Dualityビューによって提供される抽象化により、
透過的なアプリケーション・オブジェクトの展開が可能

- アプリケーションで使用している表スキーマに対する変更を、ビューを更新することでアプリケーションから隠蔽
- 新しいアプリケーションに必要なJSONフォーマットへの変更は新しいビューを作成することで既存のアプリケーションから隠蔽



画期的なロックフリー同時実行性制御

REST GETおよびPUT APIを使用している場合、
従来のロックは機能しない

- GETおよびPUTはステートレスAPI
- トランザクションとロックは、ステートレス・コール間で保持できない

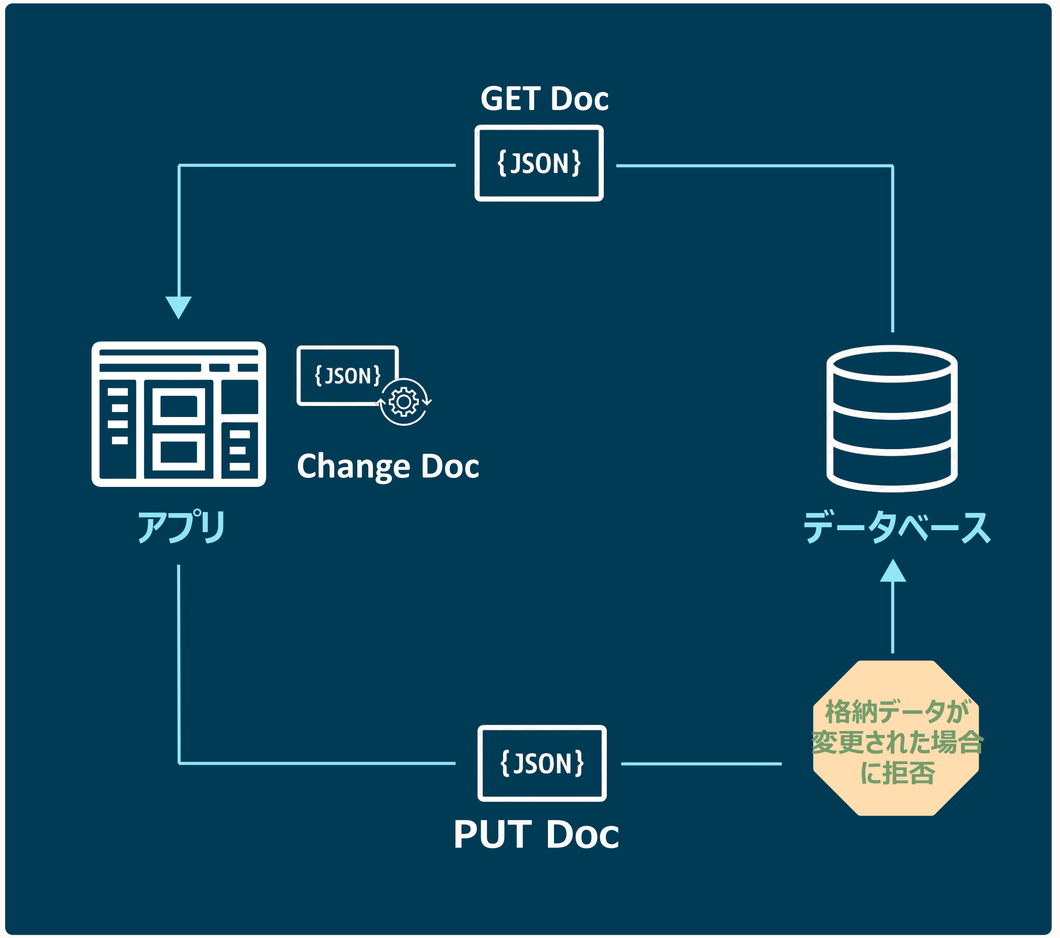
JSON Relational Dualityは、ドキュメント操作のために
新しい革新的なロックフリーの同時実行性制御プロトコルを実装

画期的なロックフリー同時実行性制御

ドキュメントのベースとなるデータが、最初のドキュメントの読取り（GET）と後続の書込み（PUT）の間に変更された場合、変更をデータベースが自動的に検出

- 変更が発生していた場合、書込み操作は自動的に拒否され、エラーが返される
- その後、アプリケーションは変更されたデータに基づいて、書込みを再実行可能

楽観的同時実効性制御と呼ばれる



ロックフリー同時実行性制御の利点

1. 人間が考えている間はデータがロックされないため、
   対話型アプリケーションに最適

2. 古いドキュメントによる書込みが拒否されるため、
   切断が発生する可能性のあるモバイルアプリケーションに最適

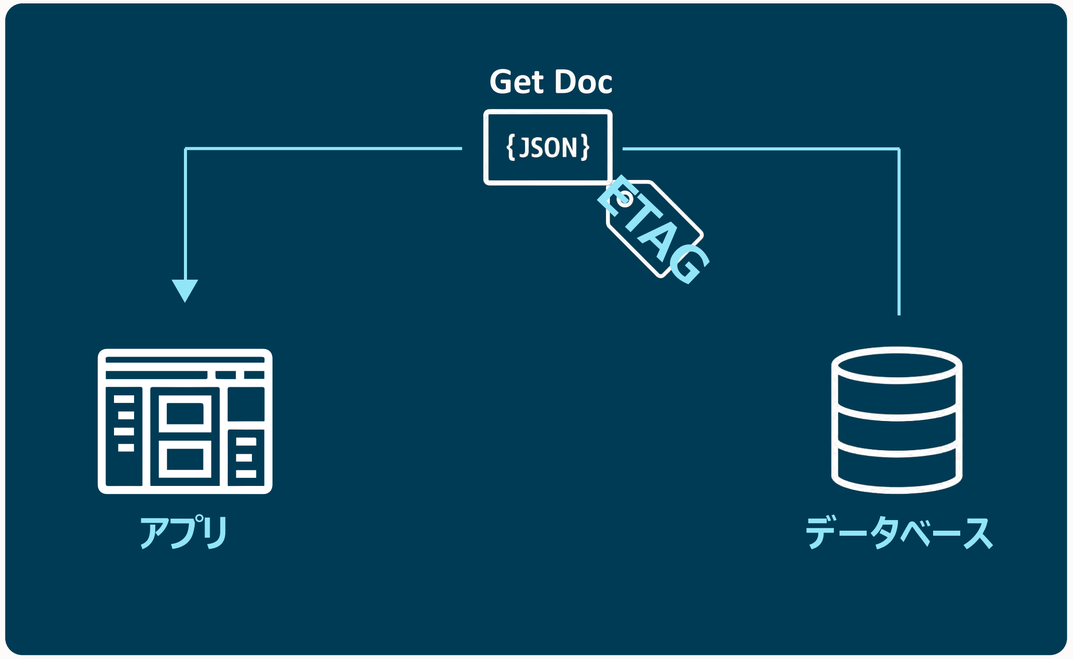
3. 失効しているキャッシュされたドキュメントの書込みが
   拒否されるため、ドキュメント・キャッシュに最適

ロックフリー同時実行性制御- 仕組み
ETAGを使用

HTTPプロトコルでは、ETAGはWebページの内容の署名またはフィンガープリント

Oracleは、ロックフリーの同時実行性制御を実装するために、HTTP ETAGをコア・データベースに独自に拡張

- データベースはETAGを自動的に計算し、戻されるドキュメント自体に挿入
- ETAGはドキュメントのベースとなる列の値から算出されるハッシュ値



ロックフリー同時実行性制御- 仕組み

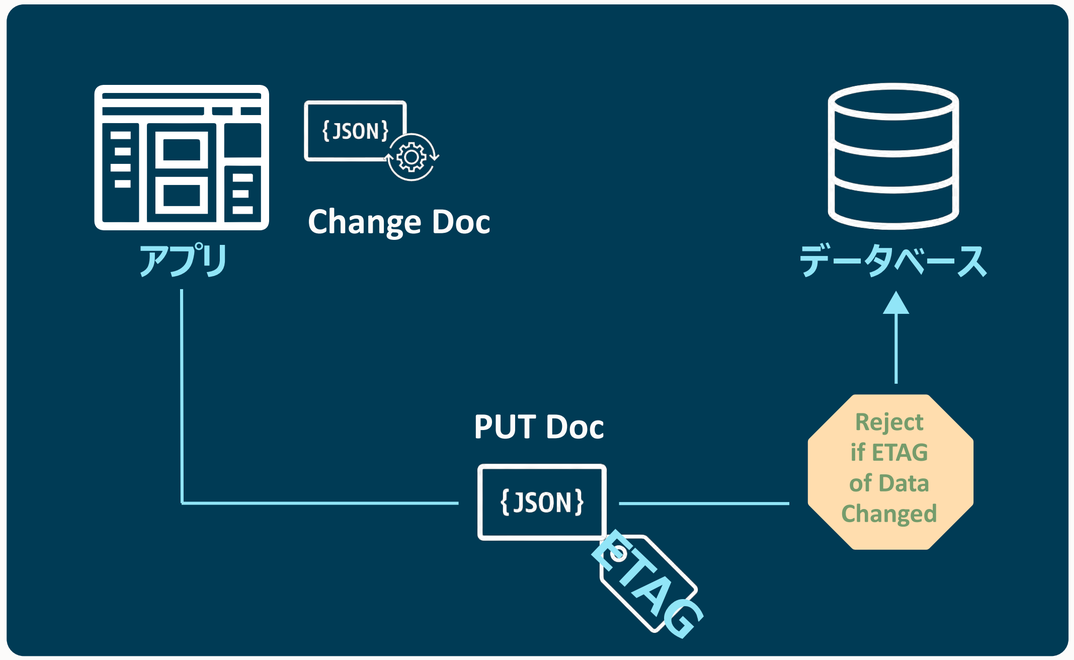
```
SQL> SELECT json_serialize(data PRETTY)
  2  FROM   student_schedule s
  3  WHERE  s.data.name = 'Jill';

JSON_SERIALIZE(DATAPRETTY)
----------------------------------------
{
  "_metadata" :
  {
    "etag" : "D341258911E06C5956538288BAC4CD91",
    "asof" : "000000000026E1FC"
  },
  "name" : "Jill",
  "student_id" : "S3245",
  "schedule" :
  [
                        ⋮
```

← ETAG：
ドキュメントのベースとなるデータベース内の列の値から
算出されるハッシュ値

ロックフリー同時実行性制御- 仕組み

変更されたドキュメントがPUTでデータベースに
書き戻される時

- DBは、ドキュメントのベースとなる列データのETAGの値がGETによって取得したETAGの値と一致していることを検証
- 列データのETAGの値がGETによって取得したETAGの値と一致する場合、列は自動的に更新される
- 一致しない場合は別のユーザーによってデータが変更されているため、PUT操作は拒否される
- 新しいデータを使用してPUTを再試行する



ETAGが一致しない（データが変更されている）場合：

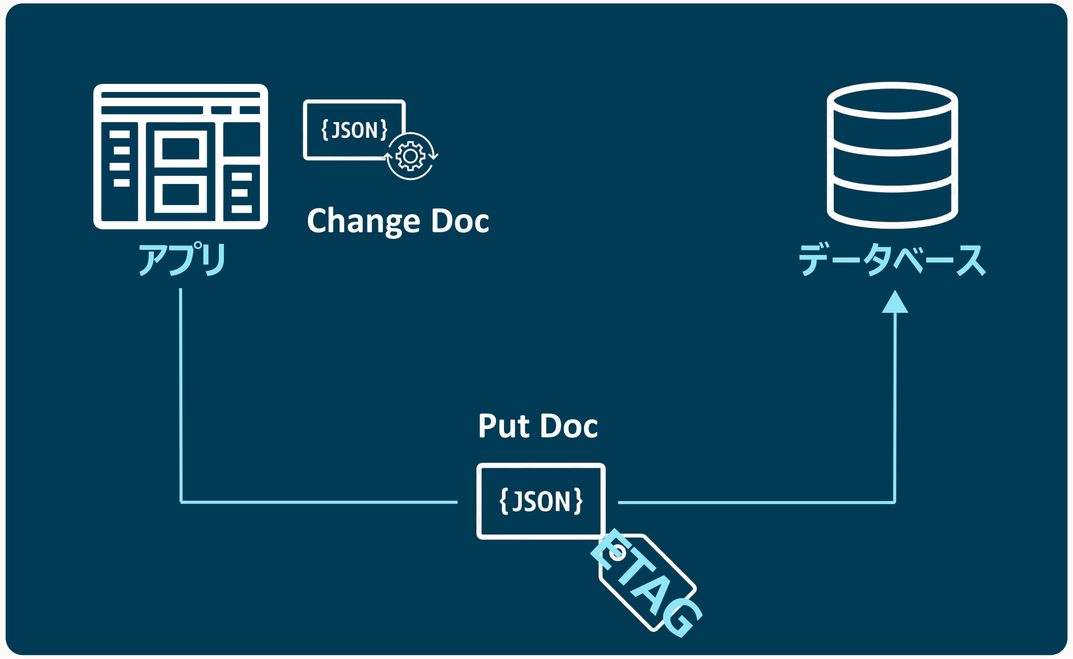
```
ERROR at line 1:
ORA-42699: Cannot update JSON Relational Duality View 'STUDENT_SCHEDULE':
The ETAG of document with ID 'FB04533332343500' in the database did not match the ETAG passed in.
Help: https://docs.oracle.com/error-help/db/ora-42699/
```

値ベースの同時実行性制御

ETAGによる同時実行性制御は値ベース

更新の競合はETAGを使用して
データの値自体を検証することで検出

ロックやバージョンをデータに追加する形ではない

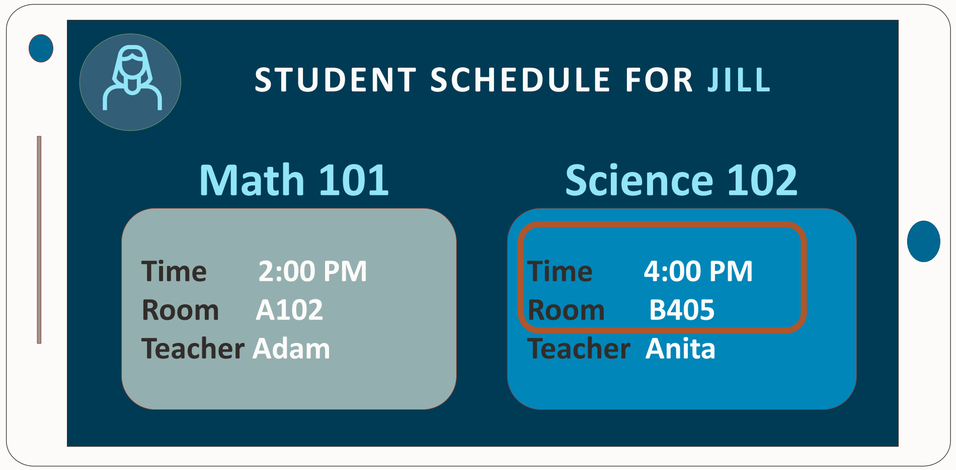


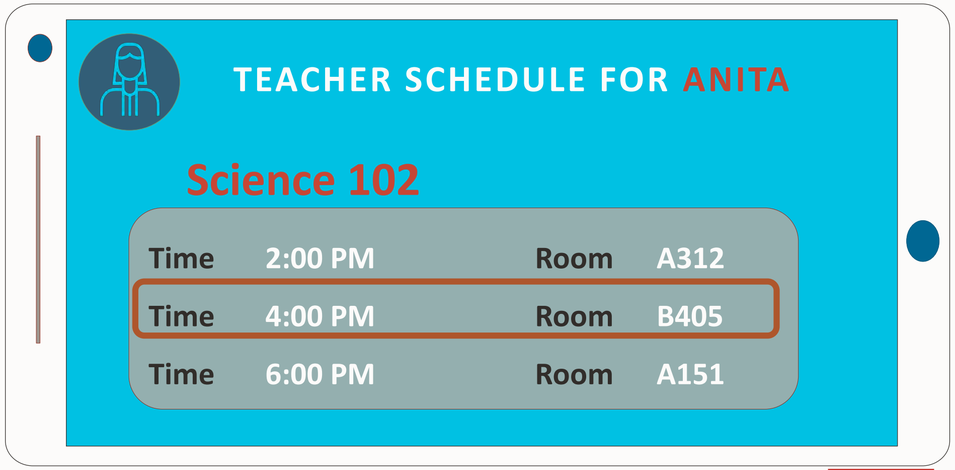
値ベースは同じデータを使用した異なるビューでも機能

値ベースであるため、ETAGは多くの異なるドキュメントで共有されるデータへの更新を自動的に同期

- 例：コース・データが多くの学生のドキュメントで共有されている
- 例：教師や学生のドキュメントのように、異なるドキュメント・ルートを持つドキュメントによって共有される場合

このような場合、従来のロックまたはバージョンベースの同時実行性制御はうまく機能しない





値ベースにより重要でない変更を無視することが可能

@nocheckにより特定のプロパティをETAGの計算対象から除外

ETAGの計算対象から、更新によってPUTが拒否されるべきでないデータを除外することが可能

- デフォルトではドキュメント内の全てのフィールドがETAGの計算対象
- @nocheckを指定したプロパティはETAGの計算から除外
- 表レベル、列レベル（プロパティレベル）で指定可能
- 列レベルの指定が優先される
- 以下のような場合には、すべてのプロパティを除外可能
  - アプリケーションで同時実行性制御を実施している
  - アプリケーションがシングルスレッドで同時更新が起きない

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

ETAGの計算対象からroomを除外すると、コースの場所が変更されていた場合に発生するDualityビューに対するPUTの失敗を回避可能

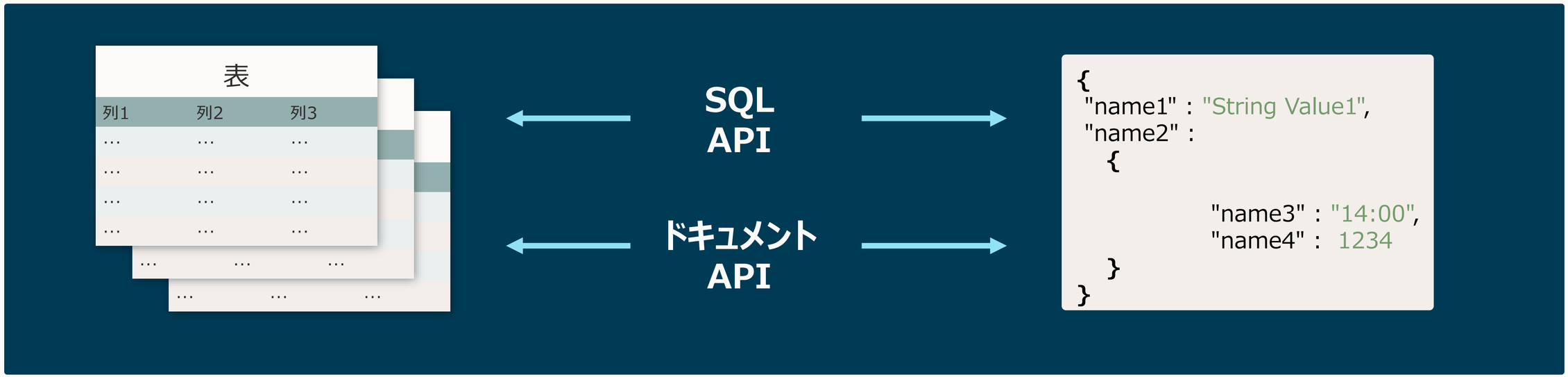
データベース内ドキュメントのすべてのユースケース

開発者はデータをリレーショナルとして格納可能

- 標準のSQLを使用して、リレーショナルとしてアクセス
- Dualityビューを使用して、ドキュメントとしてアクセス

開発者はデータをドキュメントとして格納可能

- JSON_TABLE等を使用してJSONデータに対して
  リレーショナル・ビューを作成して、リレーショナルとしてアクセス
- ANSI JSON拡張を使用してSQLから、またはRESTや
  SODA、MongoDB APIを使用して、ドキュメントとしてアクセス



データベース内ドキュメントのすべてのユースケース

開発者はデータをリレーショナルとして格納可能

- 標準のSQLを使用したリレーショナルとしてのアクセス
- Dualityビューを使用したドキュメントとしてのアクセス

開発者はデータをドキュメントとして格納可能

- JSONデータ・ガイドを使用してJSONデータに対してリレーショナル・ビューを作成し、リレーショナルとしてアクセス
- ANSI JSON拡張を使用してSQLから、またはSODAやMongoDB APIからドキュメントとしてアクセス

各ユースケースに応じて、最適なストレージ形式とアクセス形式を選択可能

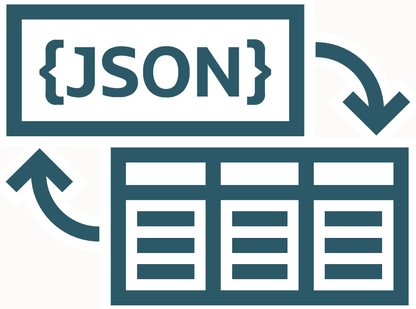
- データベースの制限による不適切なストレージ形式またはアクセス形式を強制されない

```
{
 "name1" : "String Value1",
 "name2" :
   {

              "name3" : "14:00",
              "name4" :  1234
   }
}
```

JSON Relational Dualityのメリット

アプリケーション・オブジェクトを簡単に
JSONとして永続化および操作し、
オブジェクトとリレーショナルの
ミスマッチをブリッジ可能

Dualityビューを使用すると
データをドキュメントまたは表として
透過的に読み書き可能



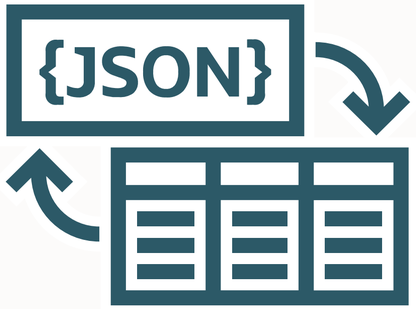
値ベースの同時実行性制御により
ドキュメントと表の間で一貫性を確保

ステートレスAPI、対話型ユース・ケース、モバイル切断クライアントを実現

JSON Relational Dualityのメリット
ドキュメント開発者

JSON Relational Dualityは
JSONの開発のシンプルさと
リレーショナルのユースケースの柔軟性を提供

データ重複や一貫性の問題を発生させることなく
ドキュメント・スキーマを各ユースケースに最適化



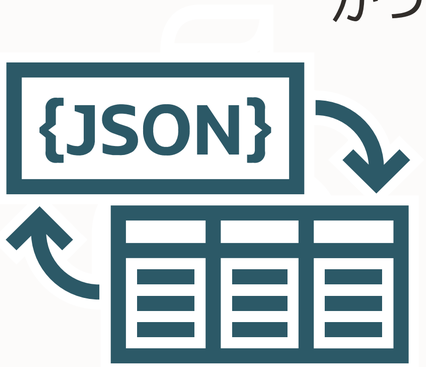
既存のリレーショナル・データを使用して
ドキュメントベースのアプリケーションを簡単に構築し、
Oracleのコンバージド・データベースの
すべての高度な機能を利用可能

JSON Relational Dualityのメリット

リレーショナル開発者

1回の呼出しおよび1回のラウンド・トリップで
アプリケーションのユースケースに必要な
すべての行に簡単にアクセス可能

JSON Relational Dualityは
ORMよりも効率的で、より一元化され、
かつ一貫性が高い



ドキュメント指向のアプリケーションで作成されたデータを使用した
SQLベースのアプリケーション作成やデータ分析が可能

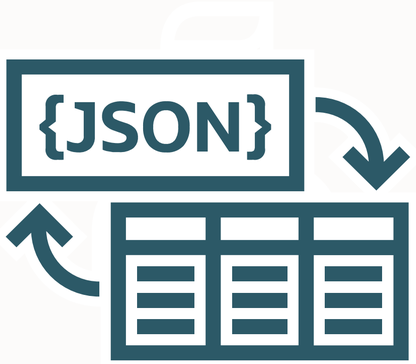
ORM: Object Relational Mapping

JSON Relational Dualityのメリット

データベース管理者、インフラ管理者

データベースが一元化されるため
管理・運用工数が削減

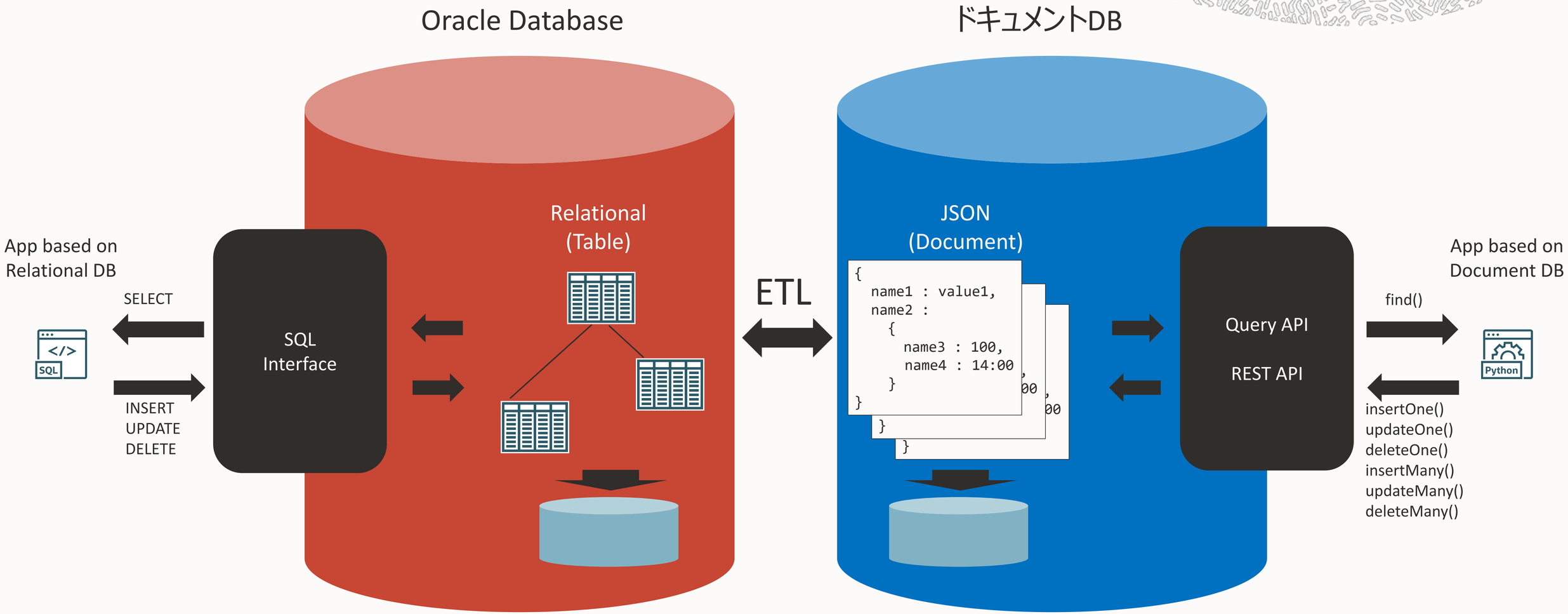
データが一元化されるため、
データの移動やデータ連携が不要



データが一元化されるため、
必要なストレージコストが削減

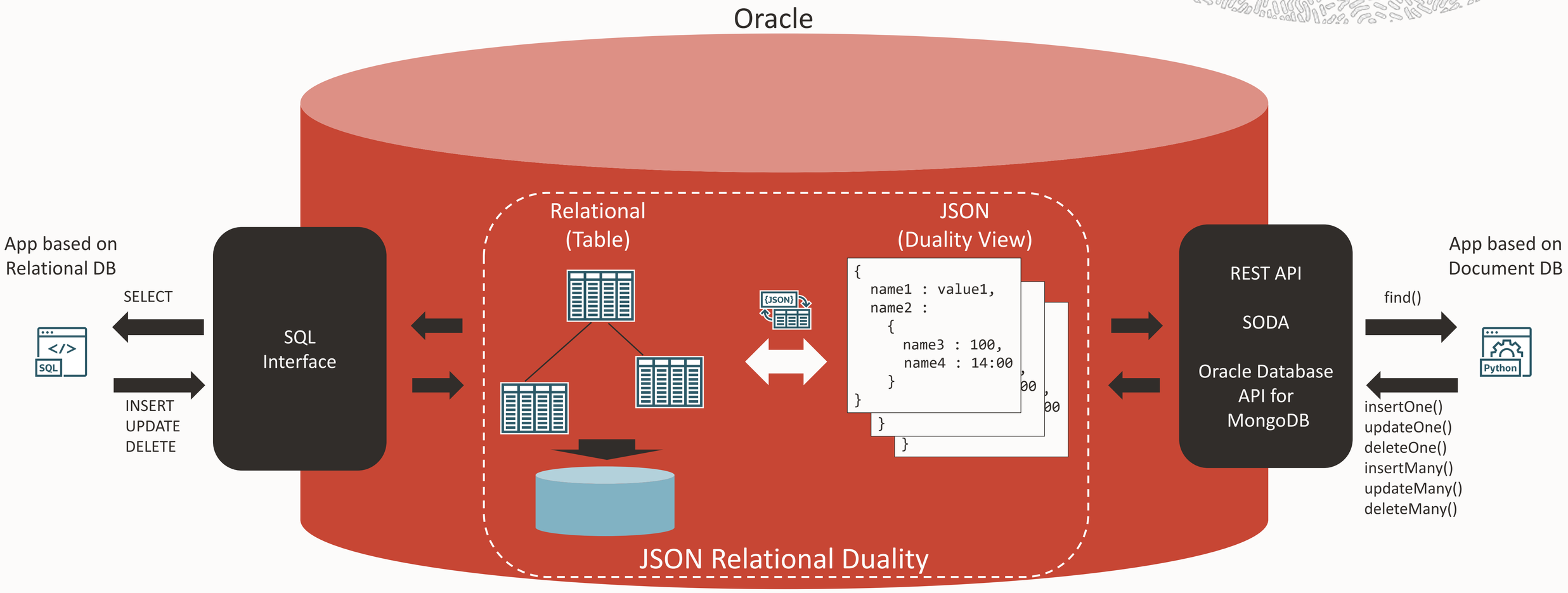
Before JSON Ralational Duality

DBの運用、DBストレージ、DBデータ、アプリケーション、ツールが全て別々、データ連携にはETLが必要



After JSON Ralational Duality

DBの運用、DBストレージ、DBデータは一つに集約、ETL不要、アプリケーション、ツールは用途によって使い分け



JSON Relational Duality 参考情報

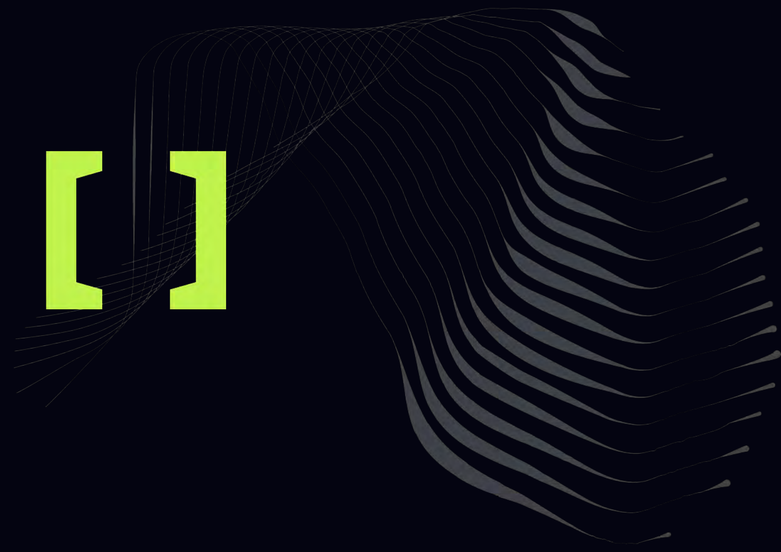
JSON Relational Dualityビュー関連のディクショナリ・ビュー　　＊：DBA or USER or ALL

- **＊_JSON_DUALITY_VIEWS**
  - Dualityビューの情報
- **＊_JSON_DUALITY_VIEW_TABS、**
  - Dualityビューから参照されている表の情報
- **＊_JSON_DUALITY_VIEW_TAB_COLS**
  - Dualityビューから参照されている表の列の情報
- **＊_JSON_DUALITY_VIEW_LINKS**
  - Dualityビューにおけるテーブル間のリンクに関する情報

JSON Relational Duality 参考情報

- マニュアル
  JSON-Relational Duality Developer's Guide
  JSONリレーショナル二面性開発者ガイド
- チュートリアル
  JSON Relational Duality tutorials
- LiveLabs
  AutoREST with JSON Relational Duality Views in 23c Free
  Exploring JSON Relational Duality Views in 23c Free with Java
  Exploring JSON Relational Duality Views in 23c Free using SQL
- Blog
  JSONとリレーショナルの二面性: ドキュメント、オブジェクトおよびリレーショナル・モデルの革新的な統合
- JSON Relational Dualityビューの制限事項
  Oracle Database 23c リリース・ノート： JSON Relational Dualityビューの制限

JSONスキーマ



JSONスキーマ

- 概要
  - 許可されるプロパティと対応する許可されるデータ型、およびそれらがオプションか必須かを指定するJSONドキュメント
    - RFCドラフトのJSONスキーマ標準に準拠（https://json-schema.org）
  - JSONデータはスキーマレスで柔軟性があるが、JSONデータが特定の構造と型を持っていることを確定させたい場合に、JSONスキーマによって検証することが可能
- メリット
  - JSONデータを事前に検証したり、チェック制約を使用して、スキーマ検証済みのJSONデータのみがJSON型の列に挿入されるようにすることができる

JSONスキーマの例:

```
{"type"       : "object",
 "properties" : {"firstName" : {"type"      : "string",
                                "minLength" : 1},
                 "salary"    : {"type"      : "number",
                                "minimum"   : 10000}},
 "required"   : ["firstName"]
}
```

有効なドキュメント：
・必須のプロパティfirstNameとオプションのプロパティsalaryを含むJSONオブジェクトである
・firstNameの値は1文字以上の文字列データである
・プロパティsalaryは必須ではないが、もし存在する場合、値は10,000以上の数値データである

JSONスキーマ

- JSONスキーマを用いたCHECK制約を使用してテーブルを作成する例（JSONスキーマを直接記述）

```
CREATE TABLE tab (
    jcol JSON
    CONSTRAINT jchk
    CHECK (jcol IS JSON VALIDATE '{"type"       : "object",
                                   "properties" : {"firstName" : {"type"      : "string",
                                                                  "minLength" : 1},
                                                    "salary"    : {"type"      : "number",
                                                                   "minimum"   : 10000}
                                                   },
                                   "required"   : ["firstName"]}'
          )
);
```

JSONスキーマ

- JSONスキーマを用いたCHECK制約を使用してテーブルを作成する例（SQLドメインを使用）

```
CREATE DOMAIN test_schema AS JSON VALIDATE USING
'{"type"       : "object",
 "properties"  : {"firstName" : {"type"      : "string",
                                 "minLength" : 1},
                  "salary"    : {"type"      : "number",
                                 "minimum"   : 10000}
                 },
 "required"    : ["firstName"]
}';


CREATE TABLE tab (
    jcol JSON DOMAIN test_schema -- 列に対してドメインを指定することで、自動的にCHECK制約が設定される
);
```

JSONスキーマを使用したCHECK制約の事前検証

- 概要
  - DBMS_JSON_SCHEMAパッケージ
    - DBMS_JSON_SCHEMA.DESCRIBEファンクション：データベース・オブジェクトからJSONスキーマを取得
    - DBMS_JSON_SCHEMA.IS_VALIDファンクション/プロシージャ：JSONデータをJSONスキーマを使用して検証
  - JSONスキーマを使用して、CHECK制約をアプリケーション側で事前検証することが可能
  - データベースの外部でJSONスキーマ・ベースの検証を実施し、データベースにデータを送る前にJSONデータの妥当性をチェックすることが可能

- メリット
  - 無効なデータを早期に検出することで、アプリケーション・システムのダウンタイムを低減可能

JSONスキーマを使用したCHECK制約の事前検証

DBMS_JSON_SCHEMA.DESCRIBEファンクション

- 表、ビュー、Dualityビュー、オブジェクト型のインスタンス、コレクション型のインスタンス、ドメインからJSONスキーマを取得

```
SQL> SELECT DBMS_JSON_SCHEMA.DESCRIBE(
  2                 object_name => 'TAB',       -- JSONスキーマのベースとなるオブジェクト名
  3                 owner_name  => 'TESTUSER', -- オブジェクトの所有者
  4                 column_name => 'JCOL'       -- JSONスキーマのベースとなる列名（オプション）
  5           ) AS json_schema;

JSON_SCHEMA
--------------------------------------------------------------------------------
{"dbColumn":"JCOL","allOf":[{"type":"object","properties":{"firstName":{"type":"
string","minLength":1},"salary":{"type":"number","minimum":10000}},"required":["
firstName"]}]}


SQL>
```

JSONスキーマを使用したCHECK制約の事前検証

DBMS_JSON_SCHEMA.IS_VALIDファンクション

- JSONスキーマを使用してJSONドキュメントの構造の検証を行う

```
SELECT DBMS_JSON_SCHEMA.IS_VALID(
            json_data    => 'JSON_doument', -- 検証対象となるJSONドキュメント
            json_schema  => 'JSON_Schema'   -- 検証に使用するJSONスキーマ
        );
```

JSONスキーマを使用したCHECK制約の事前検証

DBMS_JSON_SCHEMA.IS_VALIDファンクションの使用例

```
SQL> SELECT DBMS_JSON_SCHEMA.IS_VALID(
  2  '{"firstName":""}' , -- firstNameの値が1文字以上というJSONスキーマの要件を満たさないJSONドキュメント
'{"dbColumn":"JCOL","allOf":[{"type":"object","properties":{"firstName":{"type":"string","minLength":1},"
salary":{"type":"number","minimum":10000}},"required":["firstName"]}]}'
  3  ) AS result;

    RESULT
----------
         0

SQL> SELECT DBMS_JSON_SCHEMA.IS_VALID(
  2  '{"firstName":"Yamada"}', -- firstNameの値が1文字以上といJSONスキーマの要件を満たしているJSONドキュメント
'{"dbColumn":"JCOL","allOf":[{"type":"object","properties":{"firstName":{"type":"string","minLength":1},"
salary":{"type":"number","minimum":10000}},"required":["firstName"]}]}'
  3  ) AS result;

    RESULT
----------
         1
```

JSONスキーマを使用したCHECK制約の事前検証

JSONスキーマを使用してCHECK制約の事前検証を行う例

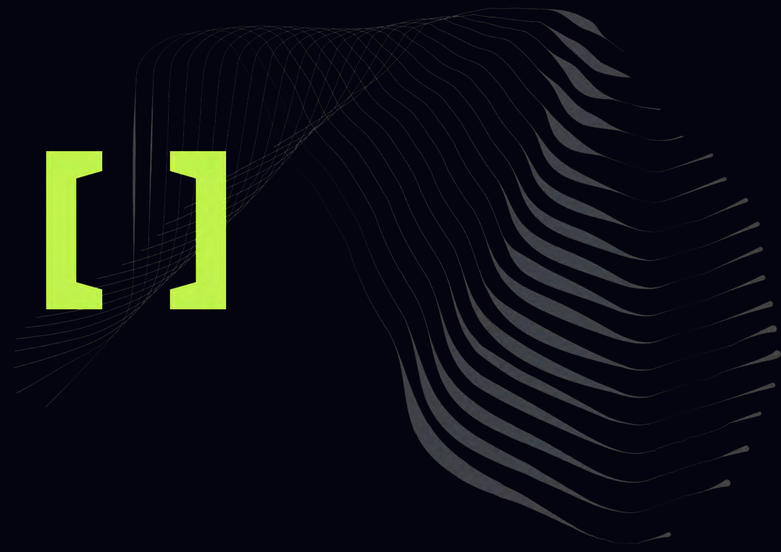
```
CREATE OR REPLACE FUNCTION validate_with_json_schema (
    l_json_text   IN VARCHAR2, -- チェック対象のJSONデータ
    l_table_name  IN VARCHAR2, -- JSONスキーマを取得する表の名前
    l_column_name IN VARCHAR2  -- JSONスキーマを取得する列の名前
)
RETURN BOOLEAN
IS
    l_json_data   JSON := JSON(l_json_text);
    l_json_schema JSON;
    l_result      BOOLEAN;
BEGIN
    -- DBMS_JSON_SCHEMA.DESCRIBEファンクションで、指定された列のJSONスキーマを取得
    SELECT DBMS_JSON_SCHEMA.DESCRIBE(object_name => UPPER(l_table_name),column_name => UPPER(l_column_name))
      INTO l_json_schema;
    -- DBMS_JSON_SCHEMA.IS_VALIDファンクションで、入力されたJSONデータに対してJSONスキーマを使用した検証を実行
    SELECT DBMS_JSON_SCHEMA.IS_VALID(l_json_data, l_json_schema) INTO l_result;
    -- 検証結果をファンクションの戻り値として返す
    RETURN l_result;
END;
/
```

JSONスキーマを使用したJSONデータの事前検証

JSONスキーマを使用してDualityビューに登録するデータの事前検証を行う例

```
CREATE OR REPLACE FUNCTION validate_with_json_schema_dv (
    l_json_text   IN VARCHAR2, -- チェック対象のJSONデータ
    l_dv_name     IN VARCHAR2  -- Dualityビューの名前
)
RETURN BOOLEAN
IS
    l_json_data   JSON := JSON(l_json_text);
    l_json_schema JSON;
    l_result      BOOLEAN;
BEGIN
    -- DBMS_JSON_SCHEMA.DESCRIBEファンクションでDualityビューからJSONスキーマを生成
    SELECT DBMS_JSON_SCHEMA.DESCRIBE(UPPER(l_dv_name)) INTO l_json_schema;
    -- DBMS_JSON_SCHEMA.IS_VALIDファンクションで、JSONスキーマを使用して入力されたJSONデータを検証
    SELECT DBMS_JSON_SCHEMA.IS_VALID(l_json_data, l_json_schema) INTO l_result;
    -- 検証結果をファンクションの戻り値として返す
    RETURN l_result;
END;
/
```

その他のJSON関連の新機能



その他のJSON関連の新機能

PL/SQLでのJSONコンストラクタとJSON_VALUEがデータ変換をサポート

- JSONコンストラクタとJSON_VALUEでJSON型とPL/SQLのデータ型との間のデータ変換が可能になりました

JSON_VALUE、JSON_QUERYのJSONパス式で述語をサポート

- JSON_VALUE、JSON_QUERYで述語を含むJSONパス式が使用可能になりました

JSON_SERIALIZEでORDERED句をサポート

- 要素名をアルファベット順に並べ替えるオプションのキーワードORDEREDが使用可能になりました

問合せによるJSON_ARRAYコンストラクタ

- JSON_ARRAYの引数として副問合せを使用して配列要素を定義可能になりました

JSON_TRANSFORMの機能拡張

- 右辺でのパス式、ネストされたパスおよび算術演算の利用、配列内の要素をソートできるSORT演算子をサポート

その他のJSON関連の新機能

JSONテキストとして格納されたデータをJSON型に移行するためのツール

- DBMS_JSON.JSON_TYPE_CONVERTIBLE_CHECKプロシージャ
- JSONテキストとして格納された既存のJSONデータをJSON型に移行できるかどうかをチェック可能

SQL*LoaderがSODA（Simple Object Document Access）をサポート

- SQL*Loaderを使用して、スキーマレス・ドキュメント（JSONやXMLなどの固定データ構造を持たないドキュメント）をSODA（Simple Oracle Document Access）コレクションとしてOracle Databaseにロードできるようになりました

その他のJSON関連の新機能

JSON Search Indexとデータガイドの変更

- JSON Search Index作成時に、デフォルトではデータガイドが作成されなくなりました
- DBMS_JSONパッケージの以下のプロシージャにおいて、デフォルトでフィールド名に番号を付与することで、フィールド名の競合を回避するようになりました
  - DBMS_JSON.CREATE_VIEW
  - DBMS_JSON.GET_VIEW_SQL
  - DBMS_JSON.ADD_VIRTUAL_COLUMNS
- フラグ・オプション DBMS_JSON.DETECT_DATETIMEを使用して、JSON_DATAGUIDE 関数がISO 8601形式の日付時刻文字列値を検出できるようになりました

外部表がJSON型をサポート

- 外部表を定義する際の列の型としてJSON型が指定可能になりました

その他のJSON関連の新機能

DBMS_AQがJSON配列をサポート

- JSON型の配列をAdvanced Queuing (AQ) のペイロードとして使用可能になりました

JSON_EXPRESSION_CHECKパラメータ

- Dualityビューに対するJSONパス式中のJSONフィールドがベース表の列とマッチしているかをコンパイル時にチェック
- デフォルトはOFF
- ALTER SYSTEM/ALTER SESSIONで変更可能

JSON関連のディクショナリ・ビューの追加

- *_JSON_INDEXES：JSONデータに対して作成されているインデックスの情報
- *_TABLE_VIRTUAL_COLUMNS：JSONデータに対するインデックス作成のための仮想列の情報

PL/SQLでのJSONコンストラクタとJSON_VALUEがデータ変換をサポート

- 概要
  - JSONコンストラクタとJSON_VALUEがJSON型とPL/SQLのデータ型との変換をサポートするようになった。
  - 例えば、PL/SQLでVARRAY型をJSON型に変換したり、連想配列をJSON型に変換したりできるようになった。
- メリット
  - アプリケーションとPL/SQLでデータ更新・抽出する際のデータ交換が効率的にできるようになった。

実行例(VARRAY型をJSON型に変換)

```
SQL>  DECLARE
          TYPE theVarray IS VARRAY(4) OF NUMBER;
          myVarray theVarray := theVarray(1, 2, 3, null);
          myJSON JSON;
      BEGIN
          myJSON := JSON(myVarray);
      END;
      /
```

実行例(JSON型をVARRAY型に変換)

```
SQL>  DECLARE
          TYPE theVarray IS VARRAY(5) OF NUMBER;
          myVarray theVarray;
      BEGIN
          myVarray := JSON_VALUE(JSON('[1, 2, 3, 4, 5]'), '$'
RETURNING theVarray);
      END;
      /
```

JSON_VALUE、JSON_QUERYのJSONパス式で述語をサポート

- 概要
  - JSON_VALUE、JSON_QUERYで指定するJSONパス式で述語がサポートされるようになった。
  - 例えば、SQLを使って特定のJSONテーブルに入っている特定のデータをフィルタリングして取得することができるようになった。
- メリット
  - アプリケーション側でJSONデータを解析しなくても、SQL側で必要なデータだけを抽出できるようになった。

実行例(JSON_VALUEでUserの値が”ABULL”のPONumberを抽出)

```
SQL>  SELECT json_value(po_document, '$?(@.User=="ABULL").PONumber') AS PONumber FROM
j_purchaseorder;
```

実行例(JSON_QUERYでnameの値が”Alexis Bull”であるShippingInstructionsを抽出)

```
SQL>  SELECT json_query(po_document, '$.ShippingInstructions?(@.name == "Alexis Bull")') AS
ShippingInstructions FROM j_purchaseorder;;
```

JSON_SERIALIZEでORDERED句をサポート

- 概要
  - SQL関数JSON_SERIALIZEに、要素名をアルファベット順に並べ替えるオプションのキーワードORDEREDが使えるようになりました。オプションのキーワードPRETTYおよびASCIIと組み合せることができます。
- メリット
  - シリアライズの結果の順序付けにより、ツールと人間の両方が値を比較しやすくなります。

JSON_SERIALIZEでORDERED句をサポート

実行例：

```
SQL> SELECT JSON_SERIALIZE(empattr ORDERED) FROM emptbl;

JSON_SERIALIZE(EMPATTRORDERED)
--------------------------------------------------------------------------------
{“employee”:{“age”:25,”commission”:100,"empID":"00001","family":[{"age":23,"name":"Hanbako","rel
ation":"partner"},{"age":1,"name":"Azusa","relation":"child"}],"firstname":"taro","lastname":"ya
mada","phone":[999],"saraly":1000}}
{"employee":{"age":45,"commision":300,"empID":"00002","family":[{"age":38,"name":"Yoshiko","rela
tion":"partner"},{"age":18,"name":"Nozomi","relation":"child"},{"age":7,"name":"Hikari","relatio
n":"child"},{"age":6,"name":"Kodama","relation":"child"}],"firstname":"Jiro","lastname":"Suzuki"
,"phone":[888],"saraly":900}}

SQL>
```

表emptblのJSON型の列empattrが、要素名順にソートされたテキストデータのJSONとして出力される

問合せによるJSON_ARRAYコンストラクタ

- 概要
  - SQL/JSON関数であるJSON_ARRAY関数の引数に副問合せを利用して、配列要素を作成できるようになりました。 この機能は、SQL/JSON標準の一部です。
- メリット
  - この機能により、開発者の生産性が向上し、他のSQL/JSON標準準拠ソリューションとの相互運用性が向上します。

実行例：

```
SELECT JSON_ARRAY(SELECT JSON_OBJECT('id'    : employee_id,
                                     'name'  : last_name,
                                     'sal'   : salary
                                      RETURNING JSON)
                  FROM employees
                  WHERE salary > 12000
                  ORDER BY salary) by_salary;
```

JSON_TRANSFORMの機能拡張

- 概要
  - JSON_TRANSFORMは、右側のパス式、ネストされたパスおよび算術演算をサポートするように拡張されました。
  - 配列内の要素をソートできるSORT演算子がサポートされました。
- メリット
  - この機能拡張により、算術計算やネストされた配列に対する操作などの更新機能が向上し、開発者の生産性が向上します。

JSON_TRANSFORMの機能拡張
RHS(右辺)でのパス式の利用

- 概要
  - 23cより前では、RHSにはSQL式のみ指定可能でしたが、23cではSQL式またはSQL/JSONパス式の指定が可能となります。RHSで指定するパス式は　キーワードPATHに続いてシングルクォーテーションで括ります。

実行例：

```
select json_transform(empattr,
        SET '$new' = JSON('["090-111-1111", "090-222-2222", "090-333-33333",]'),
        append '$.employee.phone' = PATH '$new[*] ',
        keep '$.employee.lastname','$.employee.phone')  from emptbl;
```

JSON_TRANSFORMの機能拡張

RHS(右辺)での算術演算の利用

- 概要
  - JSON_TRANSFORMのRHSのパス式に、基本的な算術演算の+ (加算)、- (減算)、* (乗算)および/ (除算)を使用して複数のパス式を結合できるようになりました。また、こうした演算はネストやグループ化が可能です。

実行例：

```
Select json_transform(empAttr,
         SET '$bonus' = 1000,
         SET '$factor' = 0.02,
         SET '$.employee.compensation' =
              PATH '($.employee.salary * $factor) + $.employee.commission + $bonus',
          keep '$.employee.lastname','$.employee.compensation')
       from emptbl;
```

JSON_TRANSFORMの機能拡張
ネストされたパス

- 概要
  - 23cでは、JSON_TRANSFORM関数内で、ネストしたパス操作が可能になりました。ネストしたパス操作では、一連の操作を適用するデータを特定部分に定義します。特定部分はキーワードNESTED PATHの直後ターゲット・パスを記載することで定義されます。そのパスの後に、かっこ((、))で括り一連の操作の操作を記載します。かっこで括られた一連の操作内で、ネストされたパスは@で表します。
    ネストしたパス操作の主なユースケースは、配列要素の反復です。

実行例：

```
select json_transform(empattr,
     nested path '$.employee'(
        SET '$new' = JSON('["090-111-1111", "090-222-2222", "090-333-33333",]'),
        append '@.phone' = PATH '$new[*] ',
        keep '@.lastname','@.phone'))  from emptbl;
```

JSON_TRANSFORMの機能拡張
SORT演算子

- 概要
  - 23cでは、JSON_TRANSFORM関数内で、指定したパスで対象となる配列の要素をソートできるようになりました。

実行例：

```
Select to_char(json_transform(empAttr,
      SORT '$.employee.family' ORDER BY '$.employee.family.age' ASC,
         KEEP '$.employee.family.name','$.employee.family.age'))
      from emptbl;
```

JSONテキストとして格納されたデータをJSON型に移行するためのツール

- 概要
  - DBMS_JSON.JSON_TYPE_CONVERTIBLE_CHECKプロシージャ
  - テキストベースでVARCHAR2型、CLOB型、BLOB型に保存されているJSONデータがJSON型に移行可能かをチェックするプロシージャ
- メリット
  - 既存のテキストベースで保存されているJSONデータをJSON型に移行可能を事前にチェックできる。

実行例(ユーザtestuserが所有する表json_test_tableのjson_text列がJSON型に移行可能かをチェックする)

```
BEGIN
  DBMS_JSON.JSON_TYPE_CONVERTIBLE_CHECK(
    owner       => 'testuser',  -- チェック対象の表の所有者
    tableName   => 'json_test_table', -- チェック対象の列を含む表の名前
    columnName => 'json_text', -- チェック対象のJSONを含む列名
    statusTableName => 'my_precheck_table' --チェック結果を格納する表の名前
  );
END;
/
```

SQL*LoaderがSODA（Simple Object Document Access）をサポート

- 概要
  - SQL*Loaderを使用してSODAコレクションにデータをロードできるようになりました。

- メリット
  - JSONやXMLといったスキーマレスなデータを簡単かつ高速に Oracle Databaseにロードできるようになりました。

SQL*LoaderでSODAコレクションにデータをロードする場合のコントロールファイルの例
(ファイルjson_data.txtの内容をSODAコレクションTestCollection1にロードする)

```
LOAD DATA
INFILE 'json_data.txt'
APPEND
INTO COLLECTION TestCollection1
FIELDS TERMINATED BY '0x02'
($CONTENT)
```

外部表がJSON型をサポート

- 概要
  - 外部表の作成時に表定義における列の型としてJSON型が指定可能になりました。
- メリット
  - 外部ファイルにあるJSONデータの活用およびJSON型の列へのデータロードが容易になりました。

実行例(JSONデータを含むテキストファイルを元に外部表を作成)

```
CREATE TABLE json_data_ext
    (json_data      JSON)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY json_data_dir
    ACCESS PARAMETERS (
        RECORDS DELIMITED BY 0x'0A'
        FIELDS (json_data CHAR(1000))
    )
    LOCATION ('json_data.txt')
);
```

JSON_EXPRESSION_CHECKパラメータ

- 概要
  - DualityビューにSQL/JSONファンクションJSON_VALUE、JSON_QUERYおよびJSON_EXISTを使用する問合せにおいて、パラメータ**JSON_EXPRESSION_CHECK**をONに設定すると、SQLのコンパイル時にSQL/JSONパス式またはドット表記法構文のJSONフィールド名の不一致が検出され、レポートされるようになりました。
  - 設定方法
    - init.oraファイル
    - ALTER SYSTEM/ALTER SESSION
    - ヒント(/*+ opt_param('json_expression_check', 'on') */)
- メリット
  - 不正なJSONパス式を実行時ではなくコンパイル時に検出しデバッグできるので、Dualityビューでの作業効率が向上します。

JSON関連のディクショナリ・ビューの追加

- 概要
  - *_JSON_INDEXESビュー：JSONデータに対して作成されているインデックスの情報
  - *_TABLE_VIRTUAL_COLUMNSビュー：インデックスのために自動的に作成された仮想列の情報

- メリット
  - JSONデータを扱うために作成されたデータベース・オブジェクトに関する情報を確認可能

JSON関連のディクショナリ・ビューの追加

*_JSON_INDEXESビュー

Column	Data Type	NULL	Description
INDEX_OWNER	VARCHAR2(128)	NOT NULL	インデックスの所有者
INDEX_NAME	VARCHAR2(128)	NOT NULL	インデックスの名前
TABLE_OWNER	VARCHAR2(128)	NOT NULL	インデックスが作成されている表の所有者
TABLE_NAME	VARCHAR2(128)	NOT NULL	インデックスが作成されている表の名前
COLUMN_NAME	VARCHAR2(128)	NOT NULL	インデックスが作成されているJSON列の名前
COLUMN_DATATYPE	VARCHAR2(13)		インデックスが作成されているJSON列のデータ型
SEARCH_INDEX	BOOLEAN		インデックスがサーチ・インデックスかどうか （TRUE/FALSE）
FUNCTIONAL_INDEX	BOOLEAN		インデックスがファンクション・インデックスかどうか （TRUE/FALSE）
COMPOSITE_INDEX	BOOLEAN		インデックスがコンポジット・インデックスかどうか （TRUE/FALSE）
BITMAP_INDEX	BOOLEAN		インデックスがビットマップ・インデックスかどうか （TRUE/FALSE）
MULTIVALUE_INDEX	BOOLEAN		インデックスが複数値インデックスかどうか （TRUE/FALSE）
INDEX_EXPRESSION	VARCHAR2(4000)		インデックス式のSQLテキスト

JSON関連のディクショナリ・ビューの追加

*_TABLE_VIRTUAL_COLUMNSビュー

Column	Data Type	NULL	Description
TABLE_OWNER	VARCHAR2(128)	NOT NULL	表の所有者
TABLE_NAME	VARCHAR2(128)	NOT NULL	表の名前
VIRTUAL_COLUMN_NAME	VARCHAR2(128)	NOT NULL	仮想列の名前
VIRTUAL_COLUMN_EXPRESSION	VARCHAR2(4000)		仮想列の式
COLUMN_ID	NUMBER	NOT NULL	作成された列のシーケンス番号
SEGMENT_COLUMN_ID	NUMBER	NOT NULL	列のセグメント内でのシーケンス番号
INTERNAL_COLUMN_ID	NUMBER	NOT NULL	列の内部シーケンス番号

ありがとうございました


ORACLE