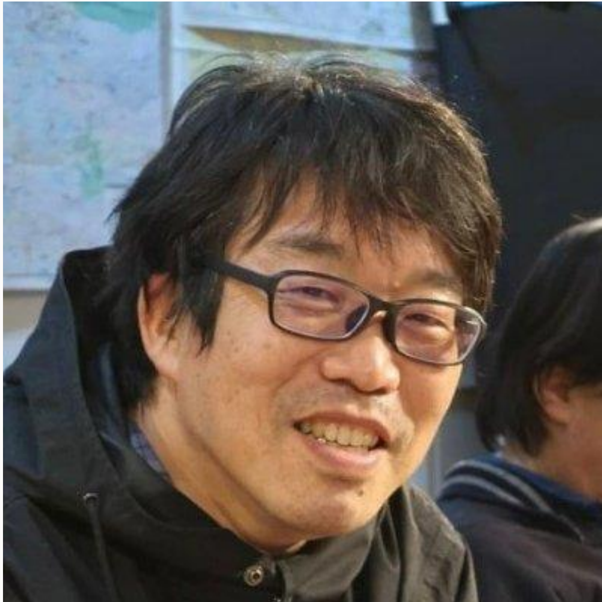
ORACLE

JSONにJavaScript - Webフロントエンド技術とMySQL HeatWaveの2024最新事情

Open Source Conference Online/Spring
2024年3月1日

日本オラクル
MySQL Global Business Unit
MySQL Principal Solution Engineer
大塚 恒平 / Kohei Otsuka

自己紹介



- 名前：大塚 恒平 （おおつか こうへい）
- 所属：日本オラクル株式会社
  MySQL Community Team /
  MySQL Global Business Unit
- 役割：MySQLのプリセールス、MySQL 及び
  MySQL HeatWave Database Service の
  普及促進活動、など
- 専門分野：GIS、地図、地理などの業界で20年
- Github：kochizufan
- 出身地：姫路市
- 趣味：オープンソース開発、
  地方史研究（群馬、奈良など）、
  石造文化財研究

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

Oracle CloudWorld 2023 での MySQL 関連発表

MySQL HeatWave での JSON データ型対応を発表

Sep. 19-21 2023

ORACLE CloudWorld

The Future of Scale-out Data Processing with HeatWave Lakehouse

Edward Screven
Chief Corporate Architect
September 20, 2023

JSON query acceleration in MySQL HeatWave

ORDER OF MAGNITUDE FASTER WITHOUT INDEXES

- JSON queries get order of magnitude acceleration without indexes
- No change in applications
- Real-time analytics on JSON documents
- JSON documents are compressed, partitioned

Application / MySQL Connectors
Scripting / MySQL Shell
SQL API / Standard Protocol
CRUD & SQL APIs / X Protocol
MySQL HeatWave
OLTP / OLAP queries / Real-time change propagation / Analytics / In-database ML / Autopilot

Announcing JSON acceleration in HeatWave

QUERY PROCESSING AND REAL-TIME ANALYTICS ON JSON DOCUMENTS

MySQL

DMLs propagated in **real-time**

- Data compressed up to **3X**
- Scales across nodes

JSON Queries (512 GB)	MySQL (sec)	HeatWave (sec)	Speedup
Simple Filter Queries	5200	240	**20X**
Aggregation Queries	5500	250	**22X**
Large Join Queries	>10 hrs	300	**144X**

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

Oracle CloudWorld 2023 での MySQL 関連発表

MySQL HW の JavaScript ストアドプログラムを発表

Sep. 19-21 2023

JavaScript Stored Programs (LA)

```
CREATE FUNCTION construct_url (path VARCHAR(50),
search VARCHAR(20)) RETURNS VARCHAR(100)
LANGUAGE JAVASCRIPT AS $$
  let url = `${path}${search &&
    !search.startsWith('?') ? '?' : ''}${search ?? ''}`;
  return encodeURI(url);
$$
```

```
SELECT construct_url('/page', 'query=шел  лы');
/page?query=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
```

```
CREATE PROCEDURE update_item_urls(OUT url_count INT)
LANGUAGE JAVASCRIPT AS $$
  let result = mysql.getSession().runSql(
    `UPDATE my_table
     SET url = construct_url(path, CONCAT('item=',product))
     WHERE product IS NOT NULL`
  );
  url_count = result.getAffectedItemsCount();
$$
```

- Seamless MySQL ⇔ JavaScript type conversion for input / output arguments
- Can be used anywhere a SQL stored function can be used – e.g. SELECT, WHERE, ORDER BY
- Support for DML, DDL, Views
- Existing XDevAPI used to execute SQL inside JavaScript

JavaScript stored programs are first-class objects in MySQL HeatWave – simplify the execution of complex operations

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

アジェンダ

1. MySQL HeatWave での JSON データ型対応
   1-1. なぜ MySQL で JSON？
   1-2. JSON データ型のふりかえり
   1-3. MySQL ドキュメントストア
   1-4. MySQL HeatWave の JSON データ型対応
2. MySQL HeatWave での JavaScript ストアドプログラム対応
   2-1. なぜ JavaScript でストアドプログラム？
   2-2. GraalVM
   2-3. MySQL HeatWave の JavaScript ストアドプログラム対応
3. 本セッションのまとめ

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

1. MySQL HeatWave での JSON データ型対応

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

1-1. なぜ MySQL で JSON？

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

リレーショナルとスキーマレスとの差とは？

- リレーショナル

```
SQL > SELECT * FROM pizza;
+------+------------------+-------+
| code | name             | price |
+------+------------------+-------+
| CLA  | Classic Pizza    | 400   |
| MAR  | Margherita Pizza | 500   |
+------+------------------+-------+
SQL > SELECT * FROM toppings;
+--------+------------+
| p_code | name       |
+--------+------------+
| CLA    | Pepperoni  |
| CLA    | Parmesan   |
| MAR    | Basil      |
| MAR    | Mozzarella |
+--------+------------+
```

テーブル（表形式）

表、カラム、行

- スキーマの適用が容易、長期的なアプリの変更管理に有用
- データに型や外部キーなどの制約を設定できる
- 正規化でデータ重複の削減
- トランザクション: ACID特性
- パワフルで標準化された SQL 言語

- スキーマレス（ドキュメント）

```
{
    "name": "Classic Pizza",
    "price": 400,
    "toppings": [ "Pepperoni", "Parmesan" ]
}
{
    "name": "Margherita Pizza",
    "price": 500,
    "toppings": [ "Basil", "Mozzarella" ],
    "options": [ { "name": "Olive", "price": 100 } ]
}
```

JSON

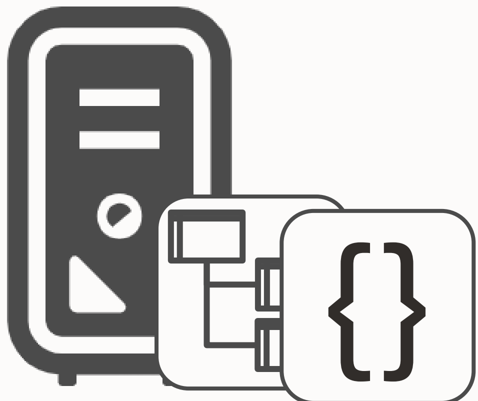
属性値のペアのコレクション

- ネストされる配列やオブジェクトのツリー構造 ([], {})
- スキーマレス: スキーマ設計、正規化、各種制約等の設計不要
- リレーショナルモデルではモデル化しにくいデータを柔軟に表現
- 迅速で容易な設計・プロトタイピング
- ORMを利用せずに、オブジェクトの永続化が可能

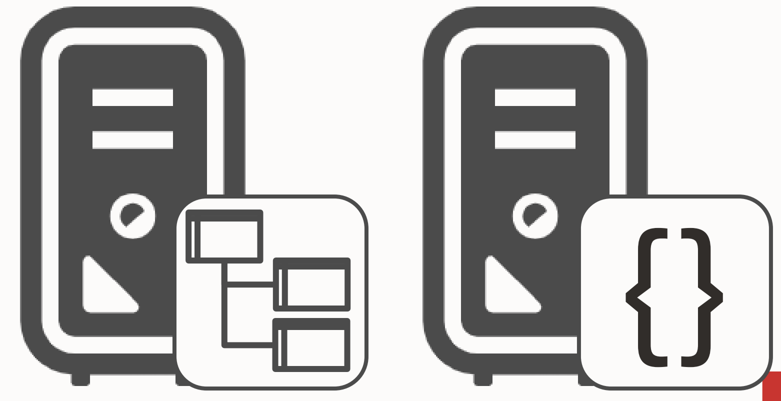
Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

単一データベースで管理するか、複数データベースで管理するか？

- 単一データベース
  - より多くのメンバーが共通スキルで管理可能
  - 管理コストが抑制される
  - 少ないライブラリで開発可能
  - 少ないツールで管理可能
  - 容易なデータ連携
  - 運用及び分析を一緒に
  - SQL処理、CRUD処理共に可能



- 複数のデータベース
  - 追加のスキルが必要となり、管理・開発の複雑化
  - 管理コストが増加
  - 開発ライブラリの複数使い分けが必要
  - 管理ツールの複数使い分けが必要
  - データ連携に工数・コストがかかる
  - 運用と分析を別システムで処理



Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

1-2. JSON データ型のふりかえり

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

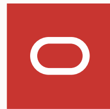
JSON データ型

MySQL 5.7.9、2015年10月 ～

- ネイティブJSONデータ型 (バイナリ形式)
- Insert時のJSON構文バリデーション機能
- 組み込みJSON関数(保存、検索、更新、操作)
- ドキュメントにインデックス設定し高速アクセス
- SQLとの統合を容易にする、新しいインライン構文
- utf8mb4の文字セットとutf8mb4_binの照合「🐬」
- サイズはmax_allowed_packetの値で制限 (Default:4MB)

```
SQL > select feature from mapdata.features where json_extract(feature,'$.properties.STREET') = 'MARKET'
limit 1¥G
*************** 1. row ***************
feature: {"type": "Feature", "geometry": {"type": "Polygon", "coordinates": [[[-122.39836263491878,
37.79189388899312, 0], [-122.39845248797837, 37.79233030084018, 0], [-122.39768507706792,
37.7924280850133, 0], [-122.39836263491878, 37.79189388899312, 0]]]}, "properties": {"TO_ST": "388", "BLKLOT":
"0265003", "STREET": "MARKET", "FROM_ST": "388", "LOT_NUM": "003", "ST_TYPE": "ST", "ODD_EVEN": "E",
"BLOCK_NUM": "0265", "MAPBLKLOT": "0265003"}}
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.



JSON データ型: サポートする値

- JSONで表現する全てのデータ型をサポート
  - 数値、文字列、Boolean (true / false)
  - オブジェクト {"キー" : "値"}、配列 [123456, "文字列", …]
  - null
- 拡張
  - 日付 (date)、時刻、日付 (datetime)、タイムスタンプ、その他

```
SQL > show create table T_JSON_SUPPORT ¥G
*************************** 1. row ***************************
       Table: T_JSON_SUPPORT
Create Table: CREATE TABLE `T_JSON_SUPPORT` (
  `id` int NOT NULL AUTO_INCREMENT,
  `body` json DEFAULT NULL,
  `type` varchar(20) GENERATED ALWAYS AS (json_type(`body`))
VIRTUAL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.0011 sec)
```

```
SQL > SELECT * FROM T_JSON_SUPPORT;
+----+------------------------------+----------+
| id | body                         | type     |
+----+------------------------------+----------+
|  8 | 1234567890                   | INTEGER  |
|  9 | NULL                         | NULL     |
| 10 | true                         | BOOLEAN  |
| 11 | "abcde"                      | STRING   |
| 12 | {"id": 5, "name": "Object"}  | OBJECT   |
| 13 | [-122.4220035, 37.80848009]  | ARRAY    |
| 14 | "2023-12-13"                 | DATE     |
| 15 | "2023-12-13 04:08:01.000000" | DATETIME |
+----+------------------------------+----------+
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型の作成

- 2通りの作り方

```
 SQL > SET @a = JSON_OBJECT('p1', 1, 'p2', JSON_OBJECT('p3', 'val'));
Query OK, 0 rows affected (0.0009 sec)
 SQL > SET @b = CAST('{"p2": {"p3": "val"}, "p1": 1}' AS JSON);
Query OK, 0 rows affected (0.0008 sec)
 SQL > SELECT @a, @b, @a = @b;
+--------------------------------+--------------------------------+---------+
| @a                             | @b                             | @a = @b |
+--------------------------------+--------------------------------+---------+
| {"p1": 1, "p2": {"p3": "val"}} | {"p1": 1, "p2": {"p3": "val"}} |       1 |
+--------------------------------+--------------------------------+---------+
1 row in set (0.0009 sec)
```

p1を先に定義

p2を先に定義

一意になる

  - JSON_OBJECT関数で、属性と値を列挙して作成
  - JSONとしてValidな文字列を、CAST関数でJSONデータ型にキャストして作成
- 属性の並び順は、定義時の順序に関係なく正規化される

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型カラムの CREATE / INSERT

- JSON カラムを含むテーブルの作成、データの挿入

```
SQL > CREATE TABLE `jsontable` (`ID` int PRIMARY KEY, `jdoc` JSON);
Query OK, 0 rows affected (0.0538 sec)
SQL > INSERT INTO `jsontable` (`id`, `jdoc`) VALUES (1, JSON_OBJECT('p1', 1, 'p2', JSON_OBJECT('p3', 'val1')));
Query OK, 1 row affected (0.0094 sec)
SQL > INSERT INTO `jsontable` (`id`, `jdoc`) VALUES (2, '{"p2": {"p3": "val2"}, "p1": 2}');
Query OK, 1 row affected (0.0102 sec)
SQL > SELECT * FROM `jsontable`;
+----+---------------------------------+
| ID | jdoc                            |
+----+---------------------------------+
|  1 | {"p1": 1, "p2": {"p3": "val1"}} |
|  2 | {"p1": 2, "p2": {"p3": "val2"}} |
+----+---------------------------------+
2 rows in set (0.0010 sec)
```

p1を先に定義

p2を先に定義

一意になる

- データ挿入方法は2通り
  - JSON_OBJECT関数で、属性と値を列挙して指定
  - JSONとしてValidな文字列を指定（キャストは暗黙で行われる）

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型: 属性の抽出

- JSON カラムからのデータの抽出

```
 SQL > SELECT JSON_EXTRACT(jdoc, '$.p1'), jdoc->'$.p2.p3' FROM jsontable;
+---------------------------+------------------+
| JSON_EXTRACT(jdoc, '$.p1') | jdoc->'$.p2.p3' |
+---------------------------+------------------+
| 1                          | "val1"           |
| 2                          | "val2"           |
+---------------------------+------------------+
2 rows in set (0.0007 sec)
 SQL > SELECT @a->'$.p1';
ERROR: 1064: You have an error in your SQL syntax; check the manual that corresponds to your My
SQL server version for the right syntax to use near '->'$.p1'' at line 1
```

変数に->演算子を使うとエラー

- JSON_EXTRACT関数でJSON内の属性値を抽出
  - 属性の指定はルートを'$'として、'.'区切りで属性を指定（配列の場合は[数字]）
  - 簡易な書き方として、'->'演算子でも同じ効果（ただし、変数に対しては使えない）
  - 文字列値に対しては、ダブルクオート"で囲まれた値として出力される

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型: 属性の抽出（文字列値の展開）

- JSON カラムからのデータの抽出（文字列値からダブルクオートを外して展開）

```
SQL > SELECT JSON_UNQUOTE(JSON_EXTRACT(jdoc, '$.p2.p3')), jdoc->>'$.p2.p3' FROM jsontable;
+----------------------------------------------+-------------------+
| JSON_UNQUOTE(JSON_EXTRACT(jdoc, '$.p2.p3')) | jdoc->>'$.p2.p3' |
+----------------------------------------------+-------------------+
| val1                                         | val1              |
| val2                                         | val2              |
+----------------------------------------------+-------------------+
2 rows in set (0.0009 sec)
```

- JSON_EXTRACT + JSON_UNQUOTE関数の組み合わせで文字列値からダブルクオートを外す
  - 簡易な書き方として、'->>'演算子でも同じ効果（ただし、変数に対しては使えない）
  - 文字列値以外の場合も、JSONオブジェクトは文字列表現となるなど影響があるデータ型がある

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型: 属性による検索

- WHERE 文に JSON_EXTRACT 関数、’->’演算子、’->>’演算子などを指定して条件設定

```
SQL > SELECT * FROM jsontable WHERE jdoc->>'$.p2.p3' = 'val1';
+----+-----------------------------------+
| ID | jdoc                              |
+----+-----------------------------------+
|  1 | {"p1": 1, "p2": {"p3": "val1"}} |
+----+-----------------------------------+
1 row in set (0.0011 sec)
SQL > SELECT * FROM jsontable WHERE jdoc->>'$.p2' = '{"p3": "val1"}';
+----+-----------------------------------+
| ID | jdoc                              |
+----+-----------------------------------+
|  1 | {"p1": 1, "p2": {"p3": "val1"}} |
+----+-----------------------------------+
1 row in set (0.0011 sec)
SQL > SELECT * FROM jsontable WHERE jdoc->'$.p2' = '{"p3": "val1"}';
Empty set (0.0011 sec)
SQL > SELECT * FROM jsontable WHERE jdoc->'$.p2' = JSON_OBJECT('p3', 'val1');
+----+-----------------------------------+
| ID | jdoc                              |
+----+-----------------------------------+
|  1 | {"p1": 1, "p2": {"p3": "val1"}} |
+----+-----------------------------------+
1 row in set (0.0034 sec)
```

文字列同士の比較

オブジェクトを->>演算子で
文字列にしての比較

->演算子ではオブジェクトはオブジェクトに
文字列との比較は失敗する

オブジェクト同士の比較

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型: 属性値の更新

- JSON_SET、JSON_REPLACE、JSON_INSERT、JSON_REMOVEなどの関数を用いて一部を変更した JSON オブジェクトを作成

```
SQL > SET @c = CAST('{"p2": [10, 20, 30], "p1": "val1"}' AS JSON);
SQL > SELECT JSON_SET(@c, '$.p2[0]', 1, '$.p1', true);
+------------------------------------------+
| JSON_SET(@c, '$.p2[0]', 1, '$.p1', true) |
+------------------------------------------+
| {"p1": true, "p2": [1, 20, 30]}          |
+------------------------------------------+
1 row in set (0.0124 sec)
```

- JSON_SET: 存在するパスの値を置き換え、存在しないパスの値を追加
- JSON_INSERT: 存在しないパスの値は追加されるが、存在するパスの値は置き換えない
- JSON_REPLACE: 存在するパスの値を置き換えるが、存在しないパスの値は追加しない
- JSON_REMOVE: 指定されたパスの値を削除する

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型カラムの UPDATE

- 元のカラムの JSON を JSON_SET などの関数を用いて一部を変更し、UPDATE 構文で更新

```
 SQL > UPDATE jsontable SET jdoc = JSON_REPLACE(jdoc, '$.p2.p3', 'replace') WHERE jdoc->'$.p1' < 5;
Query OK, 4 rows affected (0.0274 sec)

Rows matched: 4  Changed: 4  Warnings: 0
 SQL > SELECT * FROM jsontable WHERE jdoc->'$.p1' < 6;
+----+-------------------------------------+
| ID | jdoc                                |
+----+-------------------------------------+
|  1 | {"p1": 1, "p2": {"p3": "replace"}} |
|  2 | {"p1": 2, "p2": {"p3": "replace"}} |
|  3 | {"p1": 3, "p2": {"p3": "replace"}} |
|  4 | {"p1": 4, "p2": {"p3": "replace"}} |
|  5 | {"p1": 5, "p2": {"p3": "val5"}}    |
+----+-------------------------------------+
5 rows in set (0.0035 sec)
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型カラム内の属性によるインデックス付与

- JSON の属性値で GENERATED COLUMN を生成し、そのカラムにインデックスを付与

```
SQL > ALTER TABLE jsontable ADD COLUMN virt_p1 integer GENERATED ALWAYS AS (jdoc->>'$.p1') VIRTUAL;
Query OK, 700 rows affected (0.3825 sec)

Records: 700  Duplicates: 0  Warnings: 0
SQL > ALTER TABLE jsontable ADD INDEX virt_index_p1 ( virt_p1 );
Query OK, 0 rows affected (0.0572 sec)

Records: 0  Duplicates: 0  Warnings: 0
SQL > EXPLAIN SELECT * FROM jsontable WHERE jdoc->>'$.p1' > 10 AND jdoc->>'$.p1' < 30;
+----+-------------+-----------+------------+-------+---------------+---------------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type  | possible_keys | key           | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+-------+---------------+---------------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | jsontable | NULL       | range | virt_index_p1 | virt_index_p1 | 5       | NULL |   19 |      100 | Using where |
+----+-------------+-----------+------------+-------+---------------+---------------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.0052 sec)
Note (code 1003): /* select#1 */ select `jsontable`.`jsontable`.`ID` AS `ID`,`jsontable`.`jsontable`.`jdoc` AS `jdoc`,`jsontable`.`jsontable`.`virt_p1` AS `virt_p1` from `jsontable`.`jsontable` where ((`jsontable`.`jsontable`.`virt_p1` > 10) and (`jsontable`.`jsontable`.`virt_p1` < 30))
```

ASの後の演算結果で仮想的なカラム作成
- STORED: 挿入時に値計算、領域消費
- VIRTUAL: 評価時に値計算、領域消費しない
- 基本的にVIRTUALでよい

GENERATED カラムに
インデックス生成

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型関数

- JSON データ作成
  - JSON_ARRAY
  - JSON_OBJECT
  - JSON_QUOTE
- JSON データ変更
  - JSON_ARRAY_APPEND
  - JSON_ARRAY_INSERT
  - JSON_INSERT
  - JSON_MERGE
  - JSON_MERGE_PATCH
  - JSON_MERGE_PRESERVE
  - JSON_REMOVE
  - JSON_REPLACE
  - JSON_UNQUOTE
- JSON データ検索
  - JSON_CONTAINS
  - JSON_CONTAINS_PATH
  - JSON_EXTRACT
  - -> 演算子
  - ->> 演算子
  - JSON_KEYS
  - JSON_OVERLAPS
  - JSON_SEARCH
  - JSON_VALUE
  - value MEMBER OF
- JSON テーブル関数
  - JSON_TABLE
- JSON 属性データ取得
  - JSON_DEPTH
  - JSON_LENGTH
  - JSON_TYPE
  - JSON_VALID
- JSON スキーマ検証
  - JSON_SCHEMA_VALID
  - JSON_SCHEMA_VALIDATION_REPORT
- JSON ユーティリティ関数
  - JSON_PRETTY
  - JSON_STORAGE_FREE
  - JSON_STORAGE_SIZE

参照: https://dev.mysql.com/doc/refman/8.2/en/json-functions.html

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

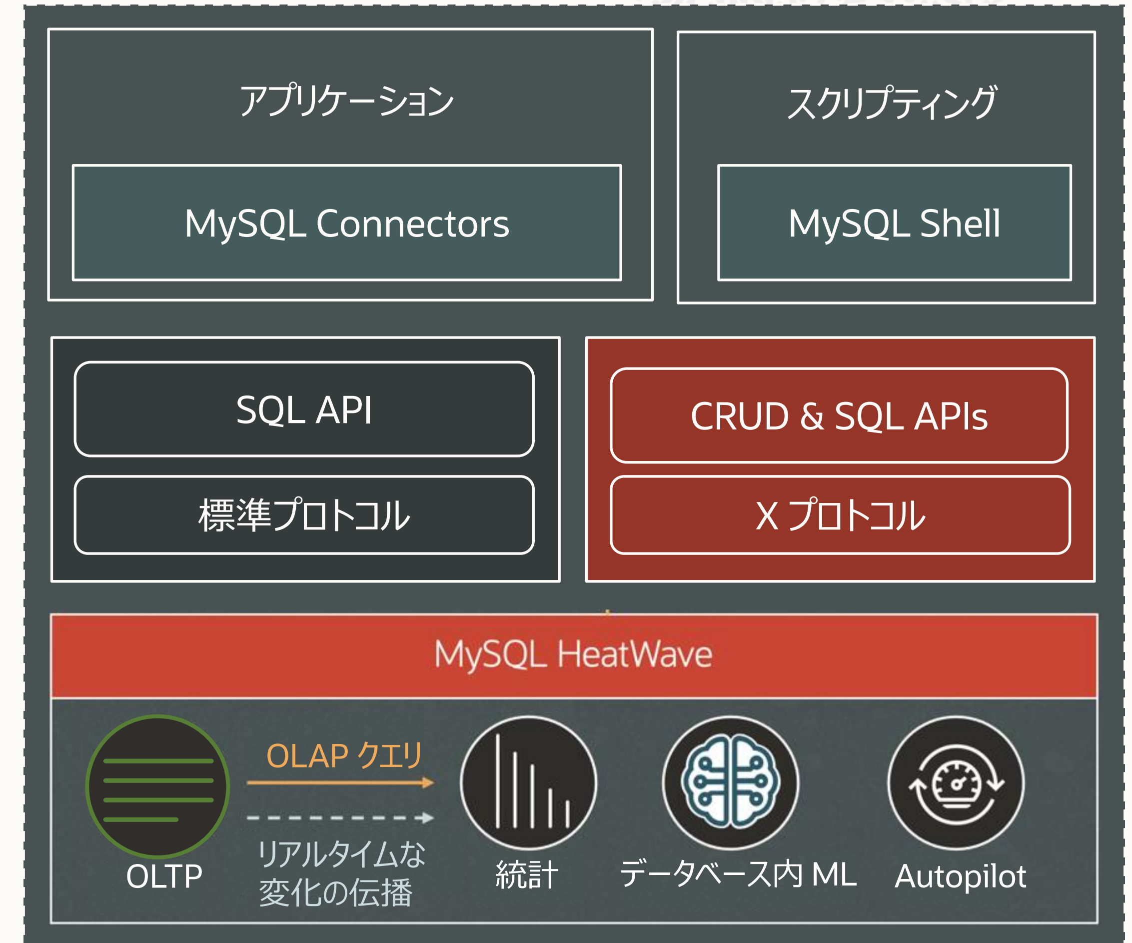
1-3. MySQL ドキュメントストア

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

ドキュメントストア機能

MySQL 5.7.12、2016年4月 ～

- X プロトコル
  - MySQL をドキュメントストアとして用いるために、X プラグイン (mysqlx) により実装、デフォルト有効
- X DevAPI
  - SQL 処理とドキュメントに対する CRUD 処理
  - 各言語の MySQL Connectors により実装
- MySQL Shell (mysqlsh) (2017年4月GA)
  - コマンドラインクライアント (Javascript, Python, SQL)
- MySQL Shell for VSCode (2022年3月以降LA)
  - Visual Studio Code のプラグインとして動作する MySQL GUI (MySQL Shell、ノートブック I/F)



Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

X プロトコルと X DevAPI

- X プラグインにより実装（デフォルトで有効化）
- X プロトコル
  - クライアントとサーバー間の柔軟な接続性
  - コマンドのパイプライン化に対応した非同期 API
  - TCP ポートとして 3306 ポートではなく 33060 ポートを利用（デフォルト）
  - 通信はラップされた protobuf 標準に則ったメッセージを使用
  - SQL とドキュメントに対する新しい CRUD API の両方をサポート
- X DevAPI
  - X プラグイン (MySQL) ⇔ X プロトコル ⇔ X DevAPI (ドライバー) の組み合わせで動作
  - ドキュメントとテーブルのコレクションに対しての CRUD 処理
  - NoSQL ライクな API 構文でドキュメントに対し CRUD 処理可能
  - 各言語の MySQL Connectors ドライバー、MySQL Shell、MySQL Shell for VSCode などで動作

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL Connectors での X DevAPI

- SQL、CRUD APIsの利用
- スキーマレスドキュメントおよびリレーショナルテーブルに対応

操作	ドキュメント	リレーショナル
Create	Collection.add()	Table.insert()
Read	Collection.find()	Table.select()
Update	Collection.modify()	Table.update()
Delete	Collection.remove()	Table.delete()

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL Connectors での X DevAPI

動作サンプル

```
[opc@test01 node]$ cat sample_node_X_API.js
const mysqlx = require('@mysql/xdevapi');
const config = { collection: 'hwCollection', schema: 'hwSchema', user: 'root', server: '10.0.X.X', password: 'password'};

const main = async () => {
  const session = await mysqlx.getSession({ host: config.server, port: 33060, user: config.user, password: config.password});
  let schema = await session.getSchema(config.schema);
  if (!(await schema.existsInDatabase())) schema = await session.createSchema(config.schema);
  const collection = await schema.createCollection(config.collection, { reuseExisting: true });
  await collection.add([{baz: { foo: "bar"}},{foo: { bar: "baz"}}]).execute();
  const res = await collection.find('$.baz.foo = :foo').bind('foo', 'bar').execute();
  const row = res.fetchOne();
  console.log("Row: %j", row);
  console.log("Collection find done!");
  await collection.remove('$.foo.bar = :bar').bind('bar', 'baz').execute();
  console.log("Document deleted");
  await schema.dropCollection(config.collection);
  return;
};

main();
[opc@test01 node]$ node sample_node_X_API.js
Row: {"_id":"000065768493000000000000011","baz":{"foo":"bar"}}
Collection find done!
Document deleted
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL Shell

- 開発および管理用のシェルの統合
- 一般的なスクリプト・インターフェースを介して利用可能な MySQLコマンドラインクライアント
- PythonやJavaScriptなどのスクリプト言語でさまざまな製品と対話するための完全な開発用API
- スキーマレスドキュメントおよびリレーショナルテーブルに対応

```
[opc@test01 ~]$ mysqlsh --help | egrep -i "Start in"
  --sql                     Start in SQL mode, auto-detecting the
  --sqlc                    Start in SQL mode using a classic session.
  --sqlx                    Start in SQL mode using an X protocol
  --js, --javascript        Start in JavaScript mode.
  --py, --python            Start in Python mode.
```

- バッチ処理に利用可能

```
[opc@test01 ~]$ mysqlsh --file code.js
[opc@test01 ~]$ mysqlsh < code.js
[opc@test01 ~]$ echo "show databases" | mysqlsh –sql
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL Shell

動作サンプル

```
[opc@test01 ~]$ mysqlsh root@10.0.X.X
MySQL Shell 8.1.1

Copyright (c) 2016, 2023, Oracle and/or its affiliates.
 ......
Server version: 8.2.0-u1-cloud MySQL Enterprise - Cloud
No default schema selected; type ¥use <schema> to set one.
 MySQL  10.0.X.X:33060+ ssl  JS > const db = session.getSchema('tpch_10g');
 MySQL  10.0.X.X:33060+ ssl  JS > db.createCollection('CUSTOMER_DOC');
<Collection:myShCollection>
 MySQL  10.0.X.X:33060+ ssl  JS > db.CUSTOMER_DOC.add({"title":"Hello World", "text":"This is the fir
st post via mysqlx"});
Query OK, 1 item affected (0.0042 sec)
 MySQL  10.0.X.X:33060+ ssl  JS > db.CUSTOMER_DOC.find('$.title = "Hello World"').sort(["title"]);
{
    "_id": "000065768493000000000000013",
    "text": "This is the first post via mysqlx",
    "title": "Hello World"
}
1 document in set (0.0007 sec)
```

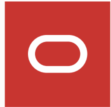

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

ドキュメントストアの仕組み

MySQL Shell、JavaScript インターフェースでの DB オブジェクトの一覧

```
MySQL  10.0.X.X:33060+ ssl  tpch_10g  JS > db.getTables();
[
    <Table:CUSTOMER>,
    <Table:LINEITEM>,
    <Table:NATION>,
    <Table:ORDERS>,
    <Table:PART>,
    <Table:PARTSUPP>,
    <Table:REGION>,
    <Table:SUPPLIER>,
]
MySQL  10.0.X.X:33060+ ssl  tpch_10g  JS > db.getCollections();
[
    <Collection:CUSTOMER_DOC>,
    <Collection:LINEITEM_DOC>,
    <Collection:NATION_DOC>,
    <Collection:ORDERS_DOC>,
    <Collection:PARTSUPP_DOC>,
    <Collection:PART_DOC>,
    <Collection:REGION_DOC>,
    <Collection:SUPPLIER_DOC>
]
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

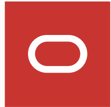


ドキュメントストアの仕組み

MySQL Shell、SQL インターフェースでの DB オブジェクトの一覧

```
MySQL  10.0.X.X:33060+ ssl  tpch_10g  SQL > show tables;
+---------------------+
| Tables_in_tpch_10g  |
+---------------------+
| CUSTOMER            |
| CUSTOMER_DOC        |
| LINEITEM            |
| LINEITEM_DOC        |
| NATION              |
| NATION_DOC          |
| ORDERS              |
| ORDERS_DOC          |
| PART                |
| PARTSUPP            |
| PARTSUPP_DOC        |
| PART_DOC            |
| REGION              |
| REGION_DOC          |
| SUPPLIER            |
| SUPPLIER_DOC        |
+---------------------+
16 rows in set (0.0012 sec)
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.



ドキュメントストアの仕組み

ドキュメントストア Collection の正体

```
MySQL  10.0.X.X:33060+ ssl  tpch_10g  SQL > SHOW CREATE TABLE LINEITEM_DOC;

 ......

| LINEITEM_DOC | CREATE TABLE `LINEITEM_DOC` (
  `doc` json DEFAULT NULL,
  `_id` varbinary(32) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,_utf8mb4'$._id'))) STORED NOT NUL
L,
  `_json_schema` json GENERATED ALWAYS AS (_utf8mb4'{"type":"object"}') VIRTUAL,
  PRIMARY KEY (`_id`),
  CONSTRAINT `$val_strict_37756BDA39FEEC09BB17E7305CECE8F5EC904E4D` CHECK (json_schema_valid(`_json_schema`,
`doc`)) /*!80016 NOT ENFORCED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |c
```

- Collectionの正体は特別な構造を持ったテーブル
  - JSON オブジェクトを保持する JSON 型カラム
  - JSON から ID の値を抜き出した STORED Generated Column
  - JSON 構文チェック用の JSON Schema 制約を含む VIRTUAL Cenerated Column

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

1-4. MySQL HeatWave の JSON データ型対応

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWaveとは：
Oracle Cloud InfrastructureのMySQLマネージドサービス

開発者
DevOps

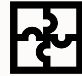
Low Code
Visual Builder
Digital Assistant
APEX

開発者
Developer
API/SDKs

Infrastructure
as Code
Resource Manager
Terraform

アプリケーション開発

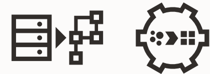
サーバーレス
Functions, Events
API Gateway,
Streaming

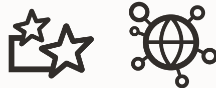

インテグレーション
Integration, Apiary

アナリティクス

Analytics
Analytics, Cloud SQL
Data Science

データ管理


データ管理
GoldenGate
Database Migration
Data Integration,
Catalog


データ処理
Data Flow
Big Data

Autonomous Database
Transaction,
Data Warehouse

データベース
Bare Metal, VM
Exadata, NoSQL,
MySQL HeatWave,
SQL Server

インフラストラクチャ

コンピュート
Bare Metal / VM

コンテナ
K8s, Registry

ストレージ
Block, File,
Object, Archive

ネットワーク
VCN, LB, VPN
FastConnect

セキュリティ
ガバナンス

ガバナンス
IAM, Policy,
Tagging
Compartment
Cost Analysis

セキュリティ
IAM, Encryption
Vault, DDoS, WAF

監視
Monitoring, Logging
Notification, Events,
Alarm

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

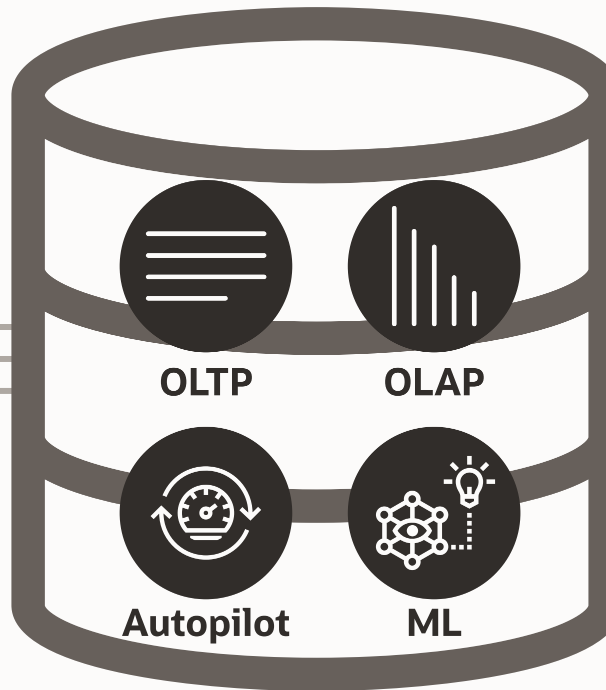
MySQL HeatWave

OLTP、OLAP、機械学習、データレイクを一つのデータベースで

トランザクション処理	分析処理	Autopilot	機械学習	レイクハウス

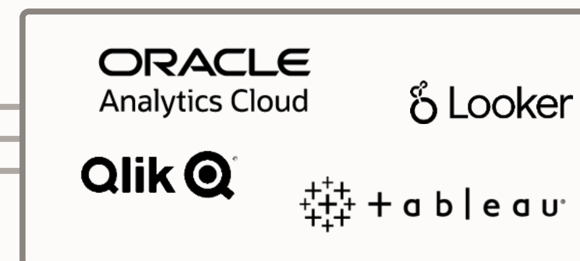
OLTP アプリケーション

MySQL HeatWave



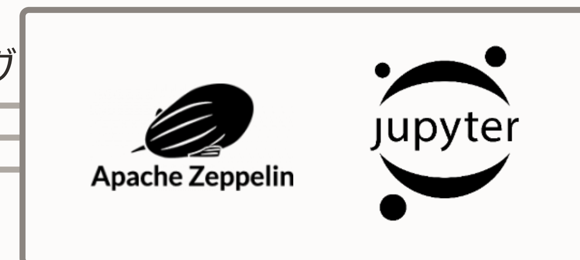
リアルタイム分析

分析ツール



最新データに
機械学習トレーニング

機械学習ツール



MySQL HeatWaveなら一つで2役以上！

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JSON データ型: クエリの最適化

JSON 処理のパフォーマンス調整に多くの時間と労力を費やす

- JSON ドキュメントで頻繁にアクセスされるフィールドに注目
- Virtual generated columns を用いて、フィールドの値を抽出
- これらのフィールドにアクセスするクエリーのパフォーマンスを向上させるために、セカンダリインデックスを作成
- インデックスの維持のために多くの作業と労力がかかる

1. 最適化すべきクエリーの特定

```
SQL > SELECT * FROM features
    -> WHERE feature->"$.properties.STREET" = 'MARKET';
```

2. `feature->properties.STREET` と対応する Virtual generated columns を作成

```
SQL > ALTER TABLE features ADD COLUMN feature_street
    -> VARCHAR(30) GENERATED ALWAYS AS
    -> (JSON_UNQUOTE(feature->"$.properties.STREET") VIRTUAL);
```

3. Virtual column にインデックスを作成

```
SQL > ALTER TABLE features ADD INDEX (feature_street);
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

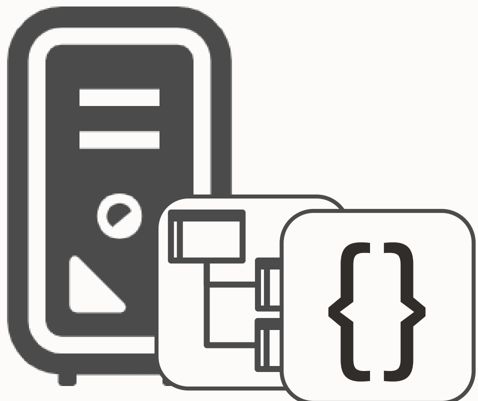
単一データベースで管理するか、複数データベースで管理するか？（再掲）

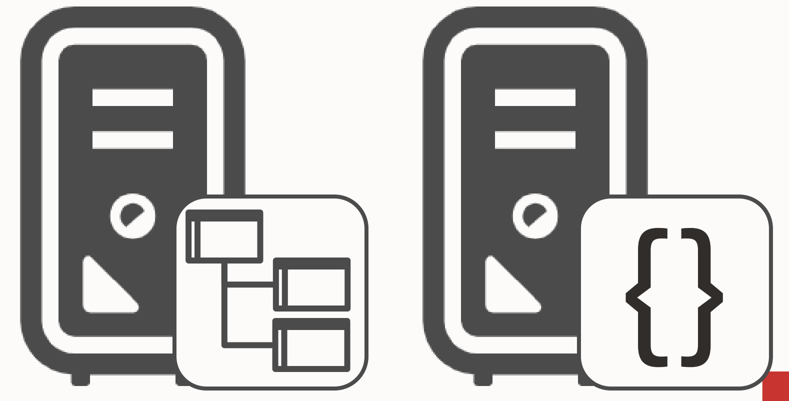
- 単一データベース
  - より多くのメンバーが共通スキルで管理可能
  - 管理コストが抑制される
  - 少ないライブラリで開発可能
  - 少ないツールで管理可能
  - 容易なデータ連携
  - 運用及び分析を一緒に
  - SQL処理、CRUD処理共に可能

ETL不要
単独DBでOLAP分析

- 複数のデータベース
  - 追加のスキルが必要となり、管理・開発の複雑化
  - 管理コストが増加
  - 開発ライブラリの複数使い分けが必要
  - 管理ツールの複数使い分けが必要
  - データ連携に工数・コストがかかる
  - 運用と分析を別システムで処理

ETLの手間
別途の OLAP DB



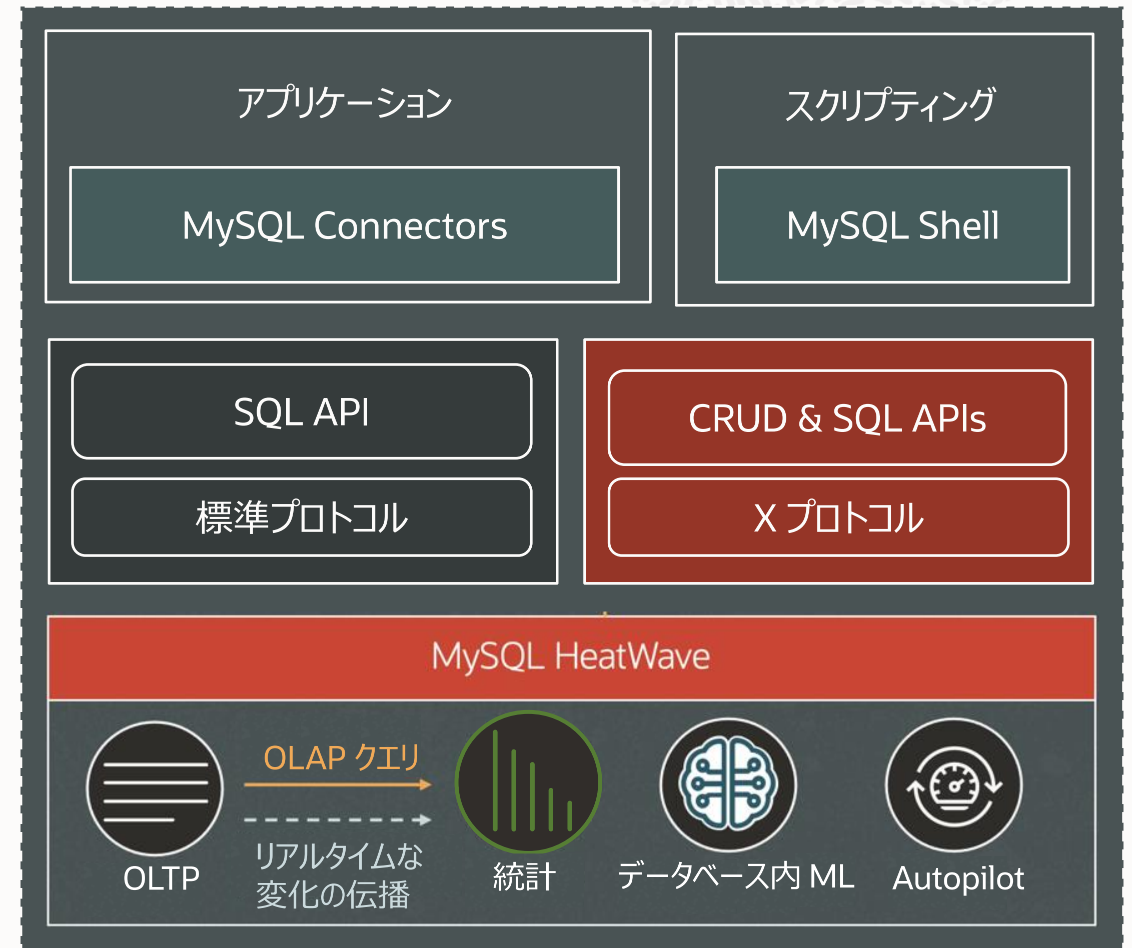


 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーの高速化

インデックスなしで桁違いに速い

- インデックスなしで JSON データへのクエリが桁違いに高速化
- アプリケーションへの変更は必要なし
- JSON ドキュメントへのリアルタイム分析
- JSON ドキュメントは圧縮、パーティショニングされて、費用対効果の高いバイナリ形式で HeatWave に保存



Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーの高速化

JSON ドキュメントに対するクエリ実行とリアルタイム分析

HeatWave ノード

InnoDB

MySQL

DML (データ操作言語) は**リアルタイム**に伝播

- **最大3倍**のデータ圧縮
- ノード間でスケールアウト

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーの高速化

TPCH 512GB のデータにおいて、20倍-144倍の高速化

- サポートする JSON 関数 (フェーズ1)
  - JSON_EXTRACT
  - JSON_OBJECT
  - JSON_LENGTH
  - JSON_DEPTH
  - JSON_ARRAY
  - JSON_UNQOUTE
  - -> / JSON_EXTRACT
  - ->> / JSON_UNQUOTE(JSON_EXTRACT())
- リアルタイムで HeatWave に伝播する JSON ドキュメントの DML (データ操作言語)
  - JSON_INSERT
  - JSON_SET
  - JSON_UPDATE

TPCH_JSON_512	MySQL (秒)	HeatWave (秒)	高速化率
単純な検索クエリー	5200	240	**20倍**
集計クエリー	5500	250	**22倍**
大きな JOIN クエリー	10 時間以上	300	**144倍**

Dec. 12th, 2023
フェーズ 1 GA
8.2 Innovation Release – u2 から

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーのテスト結果

8.2 PREVIEWでの実行結果、8.3で状況再確認

- TPC-H のダミーデータ **10GB** を利用、各テーブルをJSONデータに変換（下記はLINEITEM テーブルでの説明）
- LINEITEM の各行は 1 つの JSON ドキュメントに変換
- LINEITEM テーブルのカラムは JSON ドキュメントのフィールドに変換
- 各テーブルと各JSONテーブルで、等価の SQL を実行、HeatWave ノードオフ、オン（1、2、4台）で実行時間を比較

ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	...	...	COMMENT

tpch_10g.**LINEITEM** テーブル

id	jdoc

tpch_10g.**LINEITEM_JSON** テーブル

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーのテスト結果: 単独テーブル SQL

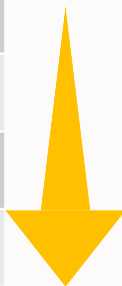
リレーショナルテーブル向け SQL

```
SQL > select sum(L_EXTENDEDPRICE * L_DISCOUNT) as revenue from LINEITEM
   -> where L_SHIPDATE >= date '1994-01-01' AND L_SHIPDATE < date '1994-01-01' + interval '1' year AND
   -> L_DISCOUNT between 0.06 - 0.01  AND  0.06 + 0.01 AND L_QUANTITY < 24;
```

JSON データ向け SQL

```
SQL > select sum(jdoc->'$.L_EXTENDEDPRICE' * jdoc->'$.L_DISCOUNT') as revenue from LINEITEM_JSON
   -> where jdoc->'$.L_SHIPDATE' >= date '1994-01-01' AND jdoc->'$.L_SHIPDATE' < date '1994-01-01' +
   -> interval '1' year AND jdoc->'$.L_DISCOUNT' between 0.06 - 0.01  AND  0.06 + 0.01 AND
   -> jdoc->'$.L_QUANTITY' < 24;
```

HeatWaveノード	リレーショナルテーブル向け SQL		JSON データ向け SQL	
オフ	13.0455秒	－	42.6677秒	⇒ 約3分の1低速
1 ノード	0.0428秒	304倍高速	11.0844秒	3.85倍高速
2 ノード	0.0347秒	376倍高速	5.7144秒	7.46倍高速
4 ノード	0.0282秒	463倍高速	2.9387秒	14.52倍高速



Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーのテスト結果: 複数テーブル JOIN SQL

リレーショナルテーブル向け SQL

```
SQL > SELECT
    ->   l_shipmode,
    ->   SUM(CASE
    ->     WHEN
    ->       o_orderpriority = '1-URGENT'
    ->         OR o_orderpriority = '2-HIGH'
    ->     THEN
    ->       1
    ->     ELSE 0
    ->   END) AS high_line_count,
    ->   SUM(CASE
    ->     WHEN
    ->       o_orderpriority <> '1-URGENT'
    ->         AND o_orderpriority <> '2-HIGH'
    ->     THEN
    ->       1
    ->     ELSE 0
    ->   END) AS low_line_count
    -> FROM
    ->   ORDERS,
    ->   LINEITEM
    -> WHERE
    ->   o_orderkey = l_orderkey
    ->   AND (l_shipmode = 'MAIL' OR l_shipmode = 'SHIP')
    ->   AND l_commitdate < l_receiptdate
    ->   AND l_shipdate < l_commitdate
    ->   AND l_receiptdate >= DATE '1994-01-01'
    ->   AND l_receiptdate < DATE '1994-01-01' + INTERVAL '1' YEAR
    -> GROUP BY l_shipmode
    -> ORDER BY l_shipmode;
```

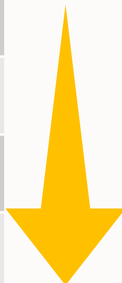
JSON データ向け SQL

```
SQL > SELECT
    ->   l.jdoc->>'$.L_SHIPMODE',
    ->   SUM(CASE
    ->     WHEN
    ->       o.jdoc->>'$.O_ORDERPRIORITY' = '1-URGENT'
    ->         OR o.jdoc->>'$.O_ORDERPRIORITY' = '2-HIGH'
    ->     THEN
    ->       1
    ->     ELSE 0
    ->   END) AS high_line_count,
    ->   SUM(CASE
    ->     WHEN
    ->       o.jdoc->>'$.O_ORDERPRIORITY' <> '1-URGENT'
    ->         AND o.jdoc->>'$.O_ORDERPRIORITY' <> '2-HIGH'
    ->     THEN
    ->       1
    ->     ELSE 0
    ->   END) AS low_line_count
    -> FROM
    ->   ORDERS_JSON o,
    ->   LINEITEM_JSON l
    -> WHERE
    ->   o.jdoc->'$.O_ORDERKEY' = l.jdoc->'$.L_ORDERKEY'
    ->   AND (l.jdoc->>'$.L_SHIPMODE' = 'MAIL' OR l.jdoc->>'$.L_SHIPMODE' = 'SHIP')
    ->   AND l.jdoc->'$.L_COMMITDATE' < l.jdoc->'$.L_RECEIPTDATE'
    ->   AND l.jdoc->'$.L_SHIPDATE' < l.jdoc->'$.L_COMMITDATE'
    ->   AND l.jdoc->'$.L_RECEIPTDATE' >= DATE '1994-01-01'
    ->   AND l.jdoc->'$.L_RECEIPTDATE' < DATE '1994-01-01' + INTERVAL '1' YEAR
    -> GROUP BY l.jdoc->>'$.L_SHIPMODE’
    -> ORDER BY l.jdoc->>'$.L_SHIPMODE';
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave による JSON クエリーのテスト結果: 複数テーブル JOIN SQL

HeatWaveノード	リレーショナルテーブル向け SQL		JSON データ向け SQL	
オフ	20.5435秒	–	82.26981秒	⇒ 約4分の1低速
1 ノード	0.2254秒	91.14倍高速	17.9719秒	4.58倍高速
2 ノード	0.2014秒	102.00倍高速	12.2419秒	6.72倍高速
4 ノード	0.1331秒	154.34倍高速	5.8289秒	14.11倍高速



- リレーショナルテーブルと同じノード数見積もりでは、リレーショナルテーブルほどの高速化は得られない
- が、ノード数を増やすほど、並列処理の効果で高速化が実現
- JSON データ型では、データサイズよりも得たい高速化レベルでノード数を検討した方がよさそう
- 今後、JSON データに適切なノード数見積もり方法が示されるかも

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL HeatWave によるJSON ドキュメントストアの高速化は未対応？

```
CREATE TABLE `LINEITEM_DOC` (
  `doc` json DEFAULT NULL,
  `_id` varbinary(32) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,_utf8mb4'$._id'))) STORED NOT NUL
L,
  `_json_schema` json GENERATED ALWAYS AS (_utf8mb4'{"type":"object"}') VIRTUAL,
  PRIMARY KEY (`_id`),
  CONSTRAINT `$val_strict_37756BDA39FEEC09BB17E7305CECE8F5EC904E4D` CHECK (json_schema_valid(`_json_schema`,
`doc`)) /*!80016 NOT ENFORCED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |c
```

- Collection に対応するテーブルに HeatWave ノードを適用、ロードを試みるも…

```
 SQL > ALTER TABLE LINEITEM_DOC SECONDARY_ENGINE = RAPID;
Query OK, 0 rows affected (0.0037 sec)

Records: 0  Duplicates: 0  Warnings: 0
 SQL > ALTER TABLE LINEITEM_DOC SECONDARY_LOAD;
ERROR: 3877: RAPID does not support pure virtual columns yet
```

- HeatWave ノードは Virtual Generated カラムに現時点で未対応の模様
- ドキュメントストアへの対応は将来の期待

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

2. MySQL HeatWave での JavaScript ストアドプログラム対応

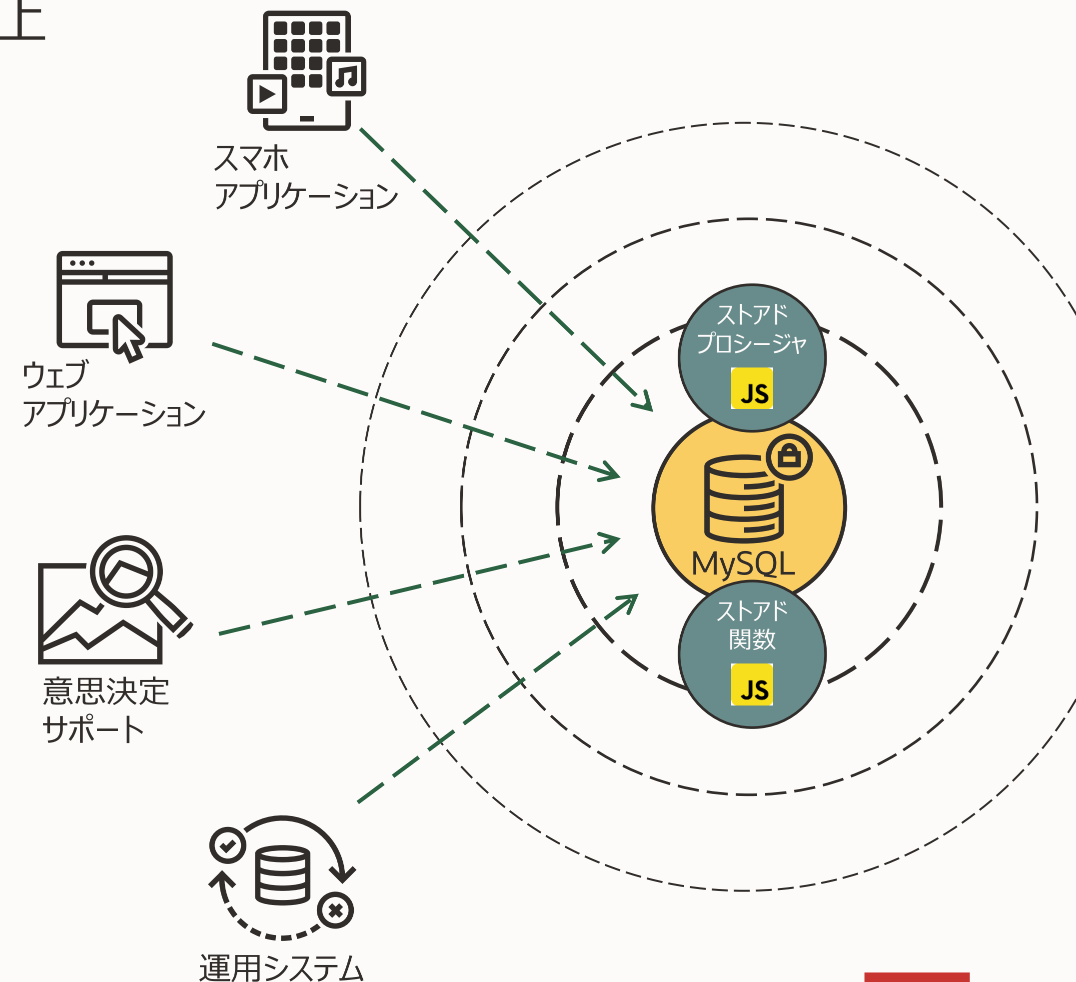
Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

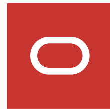
2-1. なぜ JavaScript でストアドプログラム？

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL での JavaScript ストアドプログラム

- ストアドプログラムでデータ指向アプリケーションの機能性を向上
  - コスト削減
  - データ移動の低減
  - セキュリティ改善
  - 簡潔なアーキテクチャ

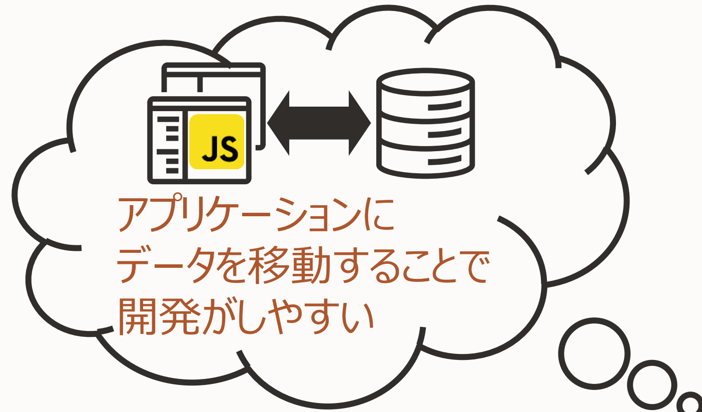




Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

アプリ開発時のデザイントレードオフ

アプリケーションでのデータ処理

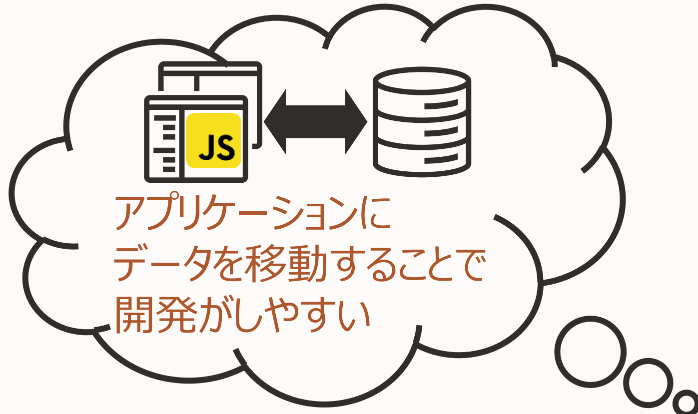


データベース内でのデータ処理

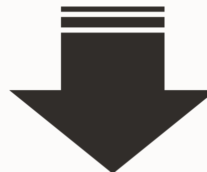
Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

アプリ開発時のデザイントレードオフ

アプリケーションでのデータ処理



データベース内でのデータ処理



MySQLなら
開発はJavaScriptで行う
データベース内でのデータ処理が可能に

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

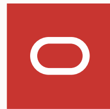
2-2. GraalVM

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

GraalVM

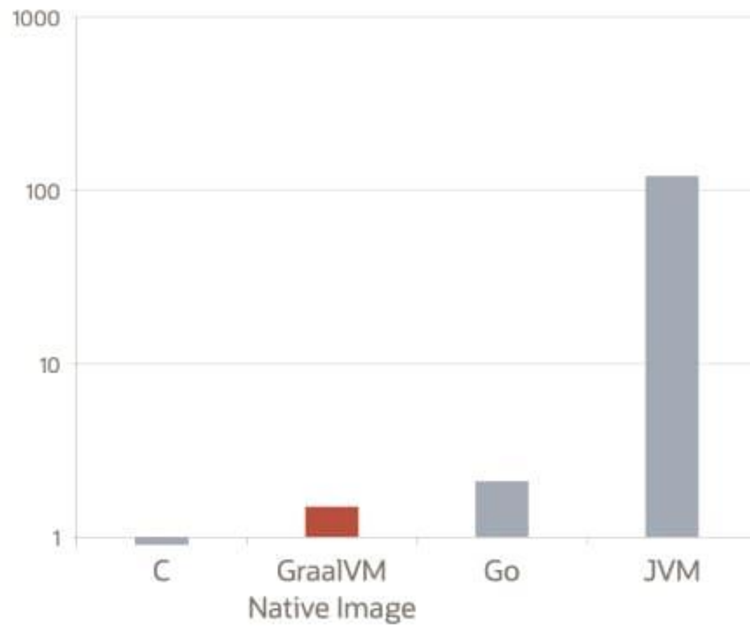
- Oracle のコンパイラ・エコシステム
  - 高パフォーマンスのJDK
  - 言語実装（JavaScript、R、Python、Ruby、Javaなど）を含む
  - 実行時（Just-In-Time、JIT）コンパイラ、事前（Ahead-Of-Time、AOT）コンパイラ
  - サンドボックス機能
  - ツールサポートを含むフルマネージドVM
- GraalVM Enterprise Edition は MySQL の JavaScript 機能と統合
  - MySQL HeatWave だけでなく、MySQL Enterprise Edition でも JavaScript ストアドプログラム機能は利用可能

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.



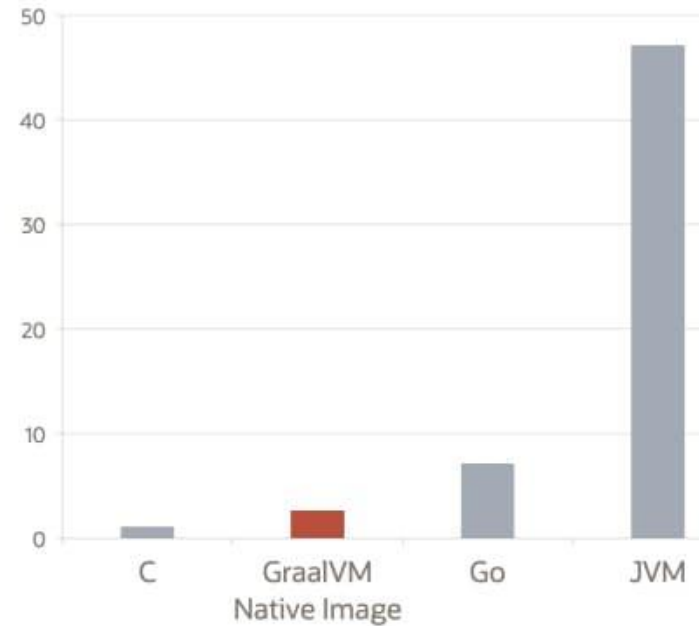
GraalVM の性能

実行時間[ms]



HelloWorldの実行時間
数千のマイクロサービスを
ほぼ瞬時に起動

最大メモリ消費 [MB]

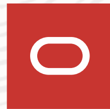


HelloWorldの実行に必要なメモリ
クラウド運用コストの低減
効率的なアプリ配布

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

2-3. MySQL HeatWave の JavaScript ストアドプログラム対応

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.



MySQL の JavaScript ストアドプログラム対応が提供するもの

- 8.3 現在のリリースで提供されるものは以下の通り
  - ECMAScript 2021標準規格に基づいたJavaScript言語
  - ストアドプロシージャ、およびストアド関数
  - MySQLのデータ型
    (例: 整数、浮動小数点、CHAR/VARCHARなど全てのバリエーション)
- npmなど既存のモジュール管理システムには、現状未対応

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JavaScriptストアドプログラムの定義

- 従来のストアド関数/プロシージャ定義の SQL の派生形で定義

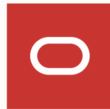
```
SQL > CREATE FUNCTION gcd_js (a INT, b INT) RETURNS INT
   -> LANGUAGE JAVASCRIPT AS $$
   ->
   ->   let [x, y] = [Math.abs(a), Math.abs(b)];
   ->   while(y) [x, y] = [y, x % y];
   ->   return x;
   ->
   -> $$;
```

関数名、引数、戻り値の型を指定

JavaScript 定義のデリミタを指定

yが0になるまで、xにy、yにxをyで割った余りを代入（最大公約数算出）

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.



JavaScriptストアドプログラムの実行

- 以下のテーブルで、生成した JavaScript ストアド関数を実行

```
SQL > SELECT col1, col2, gcd_js(col1, col2) FROM my_table;
+------+------+--------------------+
| col1 | col2 | gcd_js(col1, col2) |
+------+------+--------------------+
|    1 |    2 |                  1 |
|    2 |    3 |                  1 |
|   10 |    9 |                  1 |
|   35 |   42 |                  7 |
+------+------+--------------------+
4 rows in set (0.0023 sec)
```

- 検索条件などでも利用可能

```
SQL > SELECT col1, col2, gcd_js(col1, col2) FROM my_table WHERE gcd_js(col1, col2) > 1;
+------+------+--------------------+
| col1 | col2 | gcd_js(col1, col2) |
+------+------+--------------------+
|   35 |   42 |                  7 |
+------+------+--------------------+
1 rows in set (0.0008 sec)
```


Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JavaScriptストアドプログラムのデバッグ

- 以下の定義で JavaScript ストアドプロシージャを定義

```
SQL > CREATE PROCEDURE division (IN a INT, IN b INT, OUT result DOUBLE)
    -> LANGUAGE JAVASCRIPT AS $$
    ->
    ->   function validate(num) {
    ->     console.log("validating input value: ", num);
    ->     if (num === 0) throw ("Division by Zero!");
    ->   }
    ->   validate(b);
    ->   result = a / b;
    ->
    -> $$;
```

OUT: 呼び出し元に値を返す引数

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

JavaScriptストアドプログラムのデバッグ

- 生成した JavaScript ストアドプロシージャを実行 => エラー、標準出力、スタックトレースを確認

```
SQL > CALL division( 5, 0, @res);
ERROR: 6000: MLE-JS> Division by Zero!
SQL > SELECT mle_session_state("stdout");
+------------------------------+
| mle_session_state("stdout") |
+------------------------------+
| validating input value:  0
 |
+------------------------------+
1 row in set (0.0005 sec)
SQL > SELECT mle_session_state("stack_trace");
+-------------------------------------------------------------------------------------------+
| mle_session_state("stack_trace")                                                          |
+-------------------------------------------------------------------------------------------+
| <js> validate(division:9:194-221)
<js> division(division:11:232-242)
<js> :anonymous(division:15:269-278)
 |
+-------------------------------------------------------------------------------------------+
1 row in set (0.0008 sec)
```

throwでの出力

console.logでの出力（標準出力）

スタックトレースへの出力

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

MySQL Enterprise Edition での注意事項

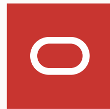
- MySQL Enterprise Edition での JavaScript サポートについて、The Oracle MySQL Japan Blog の中で記事として採り上げています
  - MySQLのJavaScriptサポートについて
- 同記事の中では、MySQL EE で JavaScript サポートを有効にする際の注意点が示されていますが、これらは MySQL HeatWave では考慮する必要はありません
  - コンポーネントインストールのため、SELinux では一時的に Enforcing モードをオフにする必要がありますが、MySQL HeatWave はマネージドサービスのため考慮の必要はありません
  - JavaScript コンパイラ導入のために、`INSTALL COMPONENT "file://component_mle";` を実行する必要がありますが、MySQL HeatWave ではデフォルトで有効になっています
  - グローバル関数 `log_bin_trust_function_creators` をオンにする必要がありますが、MySQL HeatWaveではデフォルトで `ON` になっています

 Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

セキュリティ、互換性

- GraalVM のセキュリティ保証の元に、最高レベルのセキュリティ、分離、データ保護を提供
  - VMのサンドボックスにより、悪意あるコードは MySQL サーバの他のモジュールに不正アクセスできない
  - すべてのストアドプログラムは、個別のコンテキストで解析、実行される
    - 他のストアドプログラムのデータやコードは変更できない
    - ユーザーコードからスレッド生成や操作は不可
    - ユーザーコードからネットワークやファイルシステムにはアクセスできない
- ストアドプログラムを作成できるのは権限を持つユーザーのみ
  - ストアドプログラムへのアクセス、他のユーザーによる実行なども権限設定で制御
- 従来の SQL ストアドプログラムともシームレスな互換性
  - ストレージエンジンへの依存なし、InnoDB、HeatWave、Lakehouseいずれでも利用可能

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.



3. 本セッションのまとめ

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

本セッションのまとめ 1

MySQL HeatWave での JSON データ型対応

- MySQL の JSON データ対応
  - リレーショナルとスキーマレス双方のデータを柔軟に扱える
  - JSON構文バリデーション、組み込みJSON関数
  - Generated カラムによるインデックス

- MySQL ドキュメントストア
  - 新しい X プロトコルの上に確立
  - 新しい Connectors ドライバーや MySQL Shell クライアント上で動作
  - SQL とドキュメントに対する新しい CRUD API (X DevAPI) の両方をサポート

- MySQL HeatWave の JSON データ型対応
  - Dec. 2023、MySQL HeatWave の 8.2 Innovation Releaseにて GA 化
  - ノード数を増やすほど、並列処理の効果で高速化が実現
  - ドキュメントストアには現状未対応？

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

本セッションのまとめ２

MySQL HeatWave での JavaScript ストアドプログラム

- JavaScript ストアドプログラム対応
  - アプリ作成のトレードオフ：
    データ移動を伴ったアプリでのデータ加工か、開発の困難なデータベース内でのデータ加工か
  - JavaScript ストアドプログラム：使い慣れた開発手法で、データベース内でデータ加工

- GraalVM
  - Oracle のコンパイラ・エコシステム
  - 複数言語対応、JIT コンパイラ、AOT コンパイラ
  - 速い実行速度、軽いメモリ消費

- MySQL HeatWave の JavaScript ストアドプログラム対応
  - 8.3 にて、MySQL Enterprise Edition でも対応
  - GraalVM のセキュリティ保証の元に最高レベルのセキュリティ、分離、データ保護
  - ストレージエンジンを問わずシームレスに利用可能

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

ORACLE