NoSQL + SQL = MySQL

Nicolas De Rico – Principal Solutions Architect

nicolas.de.rico@oracle.com

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract.

It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE

What If I Told You…

?

?

NoSQL + SQL

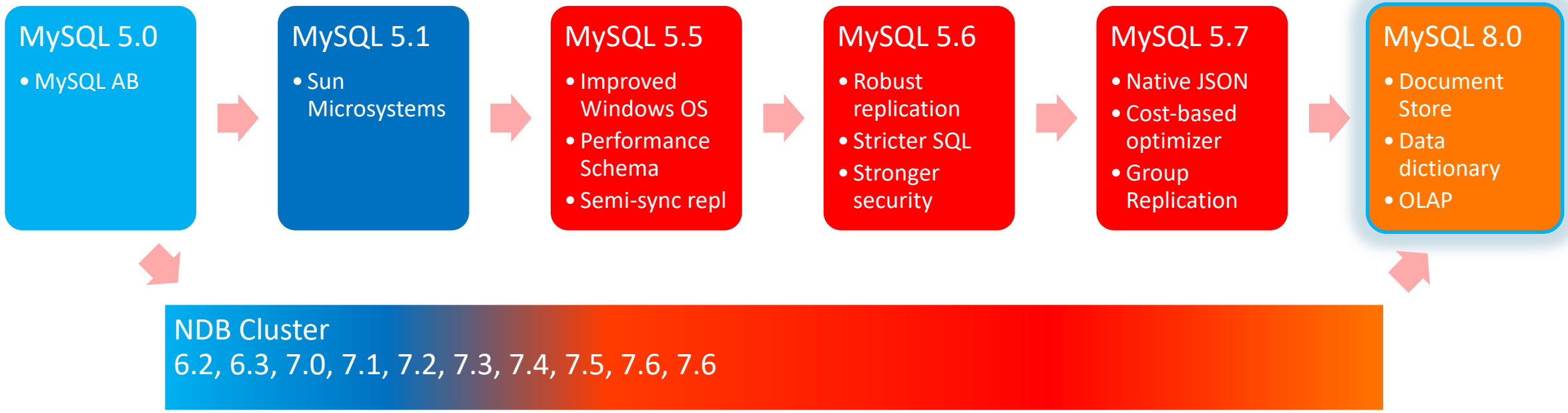
?

?

*…**is possible**?*


MySQL
ORACLE
MySQL 8.0
The MySQL Document Store
ORACLE

MySQL 8.0



MySQL Open Source (…Because It Makes Sense)

- GPLv2
  - Slightly modified for FOSS and OpenSSL
  - No extraneously restrictive licensing
- MySQL source code available on Github
  - MySQL Receives many contributions from community and partners
  - Development collaboration with some leading MySQL users
- Open Core business model
  - Additional tools and extensions available in Enterprise Edition
  - Server and client are GPL open source
    - This also helps to keep the ecosystem open source

ORACLE

New! Alter Table - Instant Add Column

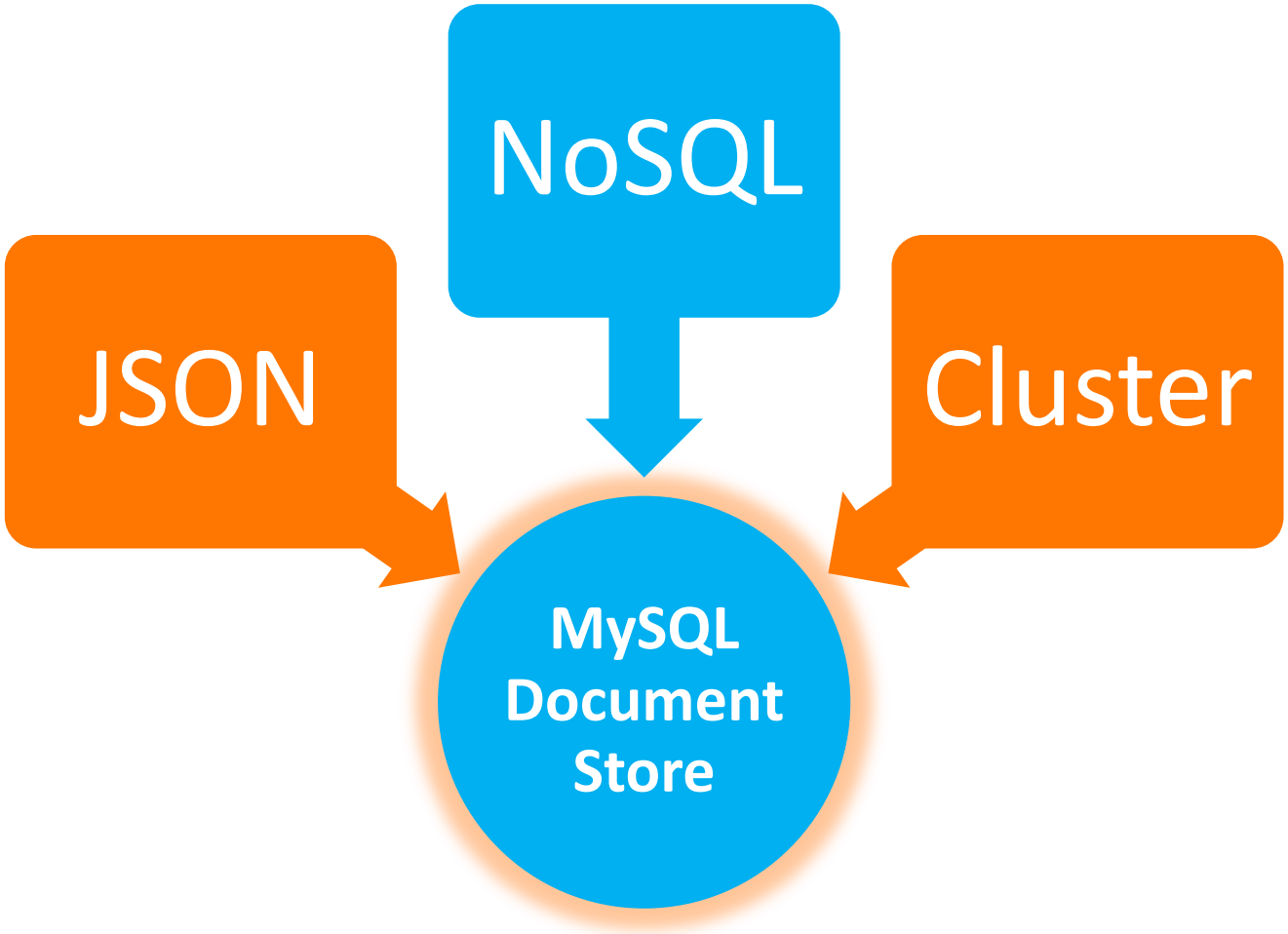
- Contribution from Tencent
  - Only a metadata change
  - No copying of data
  - Smaller final data size
  - Forward compatibility with old data file

Tencent Contribution

- `ALTER TABLE … ADD COLUMN col, ALGORITHM = INSTANT;`
- Supports DYNAMIC/COMPACT/REDUNDANT row formats

ORACLE

MySQL Document Store



Native JSON Data Type

```
CREATE TABLE employees (data JSON);
INSERT INTO employees VALUES ('{"id": 1, "name": "Jane"}');
INSERT INTO employees VALUES ('{"id": 2, "name": "Joe"}');

SELECT * FROM employees;
+----------------------------+
| data                       |
+----------------------------+
| {"id": 1, "name": "Jane"} |
| {"id": 2, "name": "Joe"}  |
+----------------------------+
2 rows in set (0,00 sec)
```

ORACLE

JSON Data Type Specifications

- utf8mb4 default character set
- Optimized for read intensive workload
  - Parse and validation on insert only
- Dictionary:
  - Sorted objects' keys
  - Fast access to array cells by index
- Full type range supported:
  - Standard: numbers, string, bool, objects, arrays
  - Extended: date, time, timestamp, datetime, others
  - JSON Objects and Arrays, including embedded within each other

JSON Functions

JSON_ARRAY_APPEND()

JSON_ARRAY_INSERT()

JSON_ARRAY()

JSON_CONTAINS_PATH()

JSON_CONTAINS()

JSON_DEPTH()

JSON_EXTRACT()

JSON_INSERT()

JSON_KEYS()

JSON_LENGTH()

JSON_MERGE()

JSON_OBJECT()

JSON_QUOTE()

JSON_REMOVE()

JSON_REPLACE()

JSON_SEARCH()

JSON_SET()

JSON_TYPE()

JSON_UNQUOTE()

JSON_VALID()

MySQL 8.0:

JSON_TABLE()

JSON_PRETTY()

JSON_STORAGE_SIZE()

JSON_STORAGE_FREE()

JSON_ARRAYAGG()

JSON_OBJECTAGG()

ORACLE

Shortcut Syntax

```
mysql> SELECT DISTINCT
       data->'$.zoning' AS Zoning
       FROM lots;
+---------------+
| Zoning        |
+---------------+
| "Commercial"  |
+---------------+
1 row in set (1.22 sec)
```

Special new syntax to access data inside JSON documents

Shortcut Syntax + Unquote

```
mysql> SELECT DISTINCT
       data->>'$.zoning' AS Zoning
       FROM lots;
+-------------+
| Zoning      |
+-------------+
| Commercial  |
+-------------+
1 row in set (1.22 sec)
```

Special new syntax to access data inside JSON documents + UNQUOTE

Indexing JSON Documents With Generated Columns

- Available as either VIRTUAL (default) or STORED:

```
ALTER TABLE features
ADD feature_type varchar(30)
AS (feature->>"$.type") VIRTUAL;
```

- Both types of computed columns permit for indexes to be added as "functional indexes"
  - Use **`ALTER TABLE… ADD INDEX(generated_column)`**
  - Use virtual generated columns to index JSON fields!

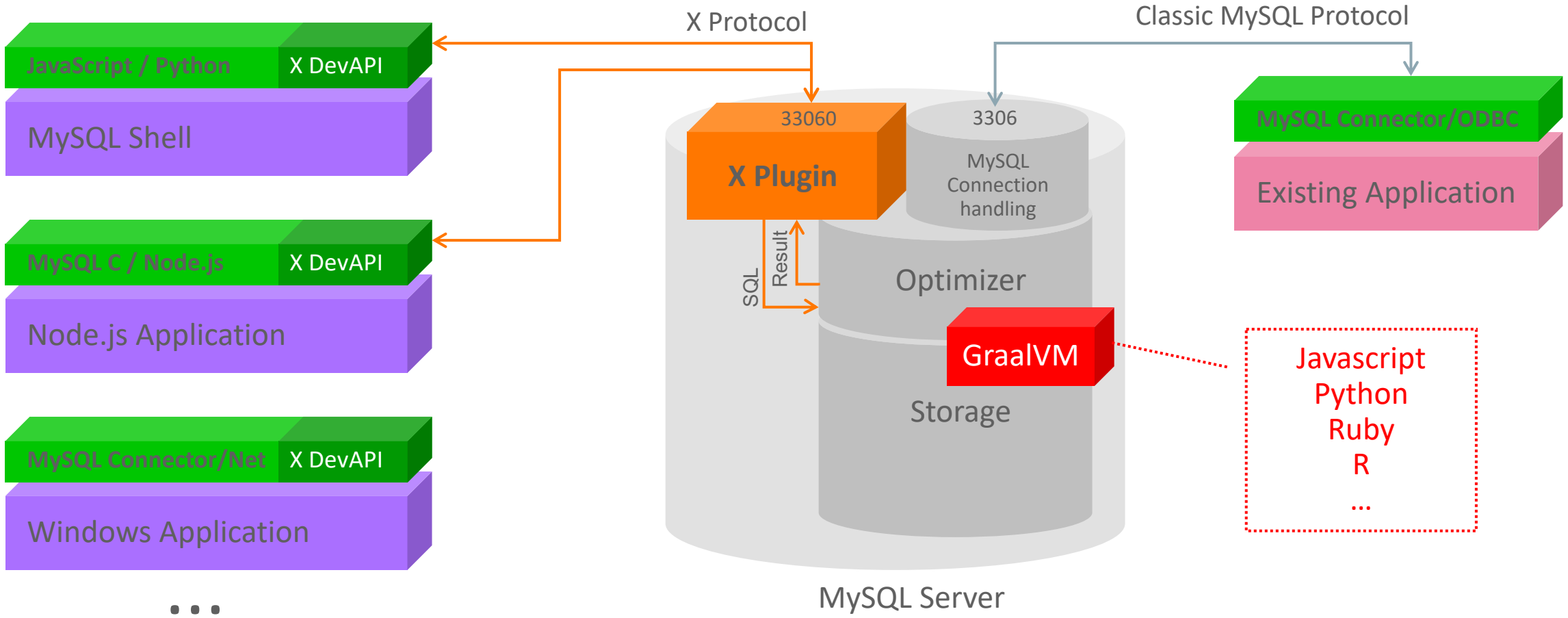
MySQL InnoDB Cluster

- Group-based replication
  - Group awareness
  - Conflict detection
  - Consensus 50% + 1
- Multi-primary mode
- Single primary mode
  - MySQL Router
- Automated failover
- Tooling for controlling cluster

"High Availability becomes a core first class feature of MySQL!"

ORACLE

MySQL Has Native NoSQL



Node.JS Example

Native MySQL Programming

```
var schema = session.getSchema('mySchema');
var collection = schema.getCollection('myColl');
var query = "$.name == :name";
collection.find(query).bind('name','Alfredo').execute(function (doc) {
   console.log(doc);
 }).catch(function (err) {
   console.log(err.message);
   console.log(err.stack);
 });
```

ORACLE

Tables or Collections?

- A collection is a table with 2+ columns:
  - Primary key: `_id`
  - JSON document: `doc`
    - The document's `_id` field can be supplied or be automatically generated by server as UUID
    - This field is also used to populate the primary key
- Can add extra columns and indexes to a collection
- SQL, NoSQL, tables, collections, all can be used simultaneously
- Operations compatible with replication

ORACLE


MySQL
ORACLE
NoSQL + SQL Demo
ORACLE

NoSQL

- The demo uses MySQL Shell, which is part of the MySQL 8 distribution
- Create a collection in Javascript
  - View what is a collection in MySQL?
- Insert JSON documents
  - With and without *_id*
- Find JSON documents
- Update JSON documents
  - Confirm the changes
- Use SQL commands on JSON documents

Creating A Collection

Show the current database

JS> db

Change the current database

JS> \use demo

JS> db

Create a new collection

JS> var Collection=db.createCollection("architects")

JS> Collection

JS> \sql

SQL> SHOW CREATE TABLE `architects`;

Accessing An Existing Collection

Opening a collection

```
JS> var Collection=db.getCollection("architects")
JS> Collection
```

The help() Function

General help

JS> Collection.help()

In-depth help

JS> Collection.help("add")

Inserting Documents Into The Collection

The missing _*id* field is automatically generated.

```
JS> Collection.add({"name": "nicolas"})
JS> Collection.add({"name": "sastry"})
JS> Collection.add({"name": "dale"})
JS> Collection.add({"name": "michael"})
JS> Collection.add({"name": "kathy"})
JS> Collection.add({"name": "lee", "title": "manager"})
JS> Collection.add({"name": "benjamin"},{"name": "priscila"})
JS> \sql
SQL> SELECT * FROM `demo`.` architects`;
```

Inserting Documents Into The Collection (cont'd)

The existing _*id* field is used.

```
JS> Collection.add({"_id": "xoxoxoxo", "name": "tony"})
JS> \sql
SQL> SELECT * FROM `demo`.` architects`;
```

Retrieving Documents

All documents

```
JS> Collection.find()
```

With a condition

```
JS> Collection.find("name='Nicolas'")
```

With a bound variable

```
JS> var Value = "nicolas"
JS> Collection.find("name=:nm").bind("nm",Value).execute()
```

A single document

```
JS> var Document=Collection.getOne("xoxoxoxo")
JS> Document
```

Modifying A Document

Add a field

```
JS> Collection.modify("_id='xoxoxoxo'").set("weight",150).execute()
```

Transform the field into an object

```
JS> Collection.modify("_id='xoxoxoxo'").set("dimmensions",{weight:150}).execute()
JS> Collection.modify("_id='xoxoxoxo'").unset("weight").execute()
```

The document already retrieved doesn’t change

```
JS> Document
```

Deleting A Document

The remove() function

JS> Collection.help("remove")

JS> Collection.remove("_id='xoxoxoxo'")

Creating An Index

NoSQL

```
JS> Collection.createIndex("nosql_index",{"fields":[{"field":"$.name","type":"TEXT(20)"}],"type":"INDEX"})
JS> \sql
SQL> SHOW CREATE TABLE `architects`;
```

SQL

```
SQL> ALTER TABLE `architects` ADD COLUMN `name` TEXT GENERATED ALWAYS AS
      (JSON_UNQUOTE(JSON_EXTRACT(`doc`,_utf8mb4'$.name’))) VIRTUAL;
SQL> ALTER TABLE `architects` ADD INDEX `sql_index`(`name`(20));
```

Verifying Index Is Used

The explain command

```
SQL> EXPLAIN SELECT * FROM `architects`;
```

```
SQL> EXPLAIN SELECT * FROM `architects` WHERE `doc`->>'$.name' = 'nicolas’;
```

Bulk Importing Documents

Create collection (optional)

```
JS> var Restaurants = db.createCollection("restaurants")
```

Import documents

```
JS> utils.help("importJson")
JS> util.importJson("restaurants",{collection : "restaurants", convertBsonOid : true})
```

Search for documents

```
JS> Restaurants.find()
JS> Restaurants.find("name='Europa Cafe'")
JS> Restaurants.find("name='Europa Cafe' and address.street = 'Lexington Avenue'")
```

ORACLE


MySQL
ORACLE
Mixing SQL And JSON
ORACLE

Exercise

- We are going to use the "restaurants" collection in SQL to find great restaurants near Times Square in New York City.
- The collection has geographical locations that we can index to limit the search of the scope to 0.5 miles around Times Square.
- We can also index the cuisine types for each restaurant so that we can choose based on our mood.
- The restaurants collection is a standard example collection from MongoDB (thanks Mongo!).

ORACLE

Adding Indexes To The Collection

Create a Spatial index (can also be done in NoSQL)

```
SQL> ALTER TABLE `demo`.`restaurants`
        ADD COLUMN `location` GEOMETRY
        GENERATED ALWAYS AS (POINT(`doc`->>'$.address.coord[0]',`doc`->>'$.address.coord[1]'))
        STORED NOT NULL SRID 0;
SQL> ALTER TABLE `demo`.`restaurants` ADD SPATIAL INDEX(`location`);
```

Create a full-text search index

```
SQL> ALTER TABLE `demo`.`restaurants` ADD COLUMN `Cuisine` TEXT
        GENERATED ALWAYS AS (`doc`->>'$.cuisine') STORED;
SQL> ALTER TABLE `demo`.`restaurants` ADD FULLTEXT INDEX(`Cuisine`);
```

Best Italian And Chinese Restaurants Near Times Square

```
WITH `CTE` AS
(SELECT `doc`->>'$.name' AS `Restaurant`,
        `Cuisine`,
        (SELECT AVG(`Grades`.`Score`) AS `Score`
         FROM JSON_TABLE(`doc`,'$.grades[*]' COLUMNS (`Score` INT PATH '$.score')) AS
           `Grades`) AS `Average`,
        ST_Distance_Sphere(`location`,@TimesSq) AS `Distance`
  FROM `demo`.`restaurants`
  WHERE ST_Contains(ST_MakeEnvelope(POINT(ST_X(@TimesSq) + @Dist,ST_Y(@TimesSq) + @Dist),
                                    POINT(ST_X(@TimesSq) - @Dist,ST_Y(@TimesSq) - @Dist)),
                                    `location`)
  AND MATCH(`Cuisine`) AGAINST('Italian Chinese' IN BOOLEAN MODE)
  ORDER BY `Distance`) ...
```

Best Italian And Chinese Restaurants Near Times Square

```
…
SELECT
  `Restaurant`,
  `Cuisine`,
  RANK() OVER (PARTITION BY `Cuisine` ORDER BY `Average`) AS `Rank`,
  `Distance`
FROM
  `CTE`
ORDER BY
  `Rank`, `Average` DESC
LIMIT 10;
```


MySQL
ORACLE
Thank You!
ORACLE