Modern Development With MySQL

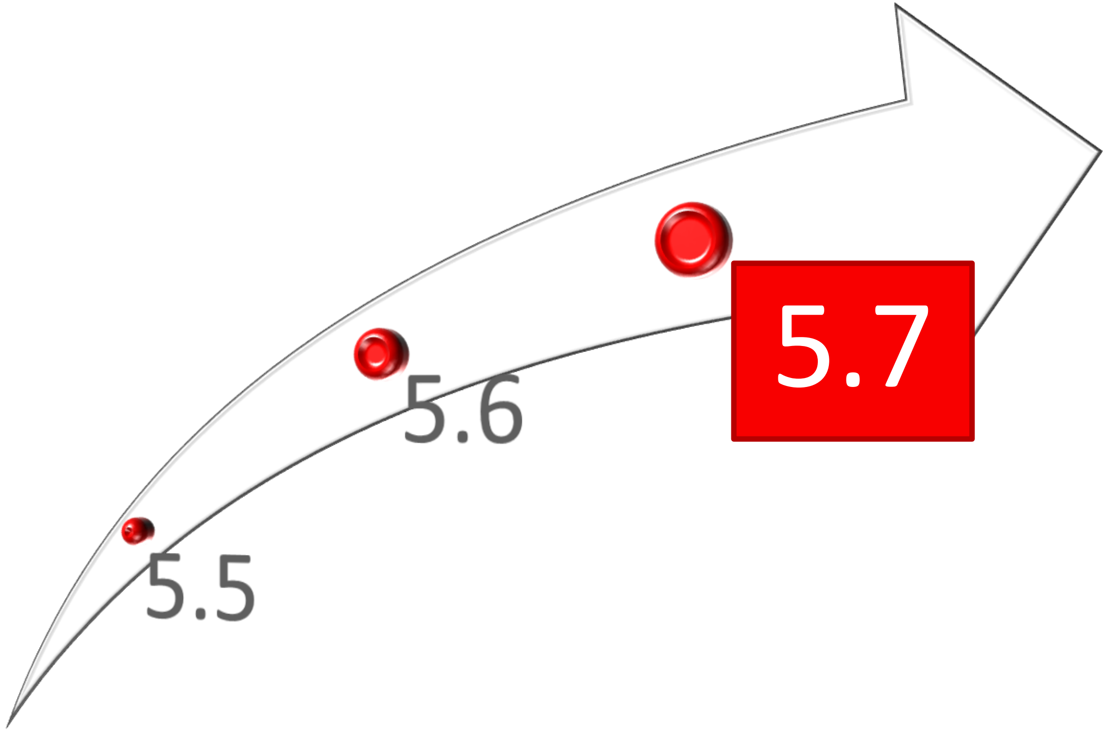
Nicolas De Rico

nicolas.de.rico@oracle.com

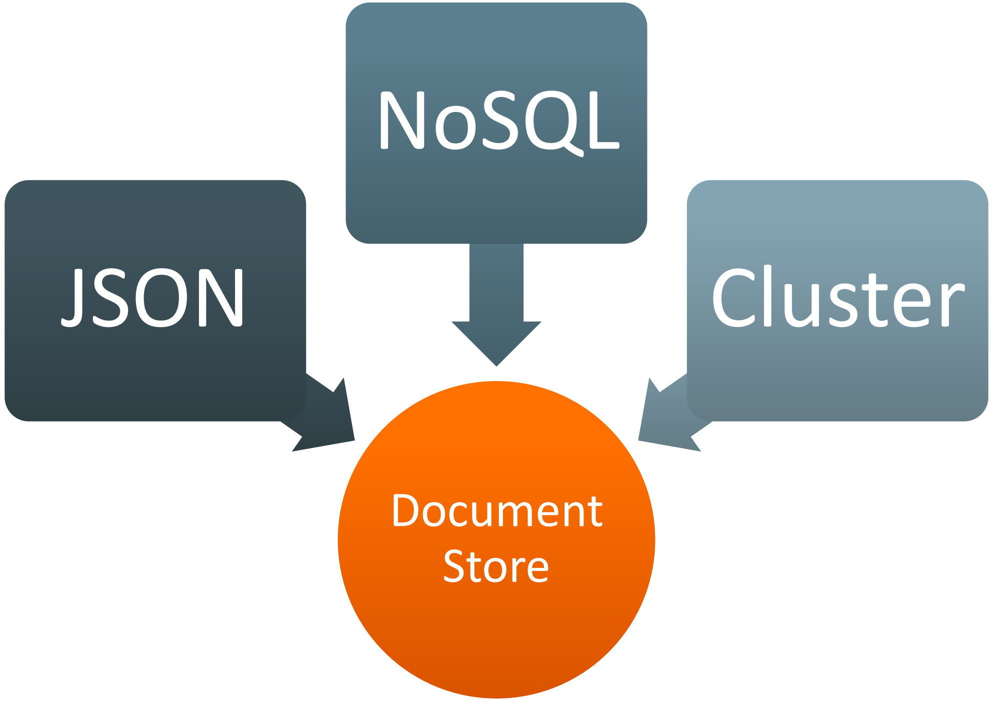
Safe Harbor Statement

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.


5.5
5.6
5.7

MySQL Document Store


NoSQL
JSON
Cluster
Document Store

New! 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)
```

JSON Data Type Specifications

- utf8mb4 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

SQL Example

```
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

Advantages of Native JSON

- Provides Document Validation:

```
> INSERT INTO employees VALUES ('some random text');

ERROR 3130 (22032): Invalid JSON text: "Expect a value here." at position 0 in value
(or column) 'some random text'.
```

- Efficient Binary Format.
  - Allows quicker access to object members and array elements
  - Well suited for InnoDB Barracuda file format

Naive Comparison JSON Vs. TEXT

Unindexed traversal of 206K documents

JSON type

```
SELECT DISTINCT
feature->"$.type" as json_extract
FROM features;
+---------------+
| json_extract |
+---------------+
| "Feature"     |
+---------------+
1 row in set (1.25 sec)
```

TEXT type

```
SELECT DISTINCT
feature->"$.type" as json_extract
FROM features;
+---------------+
| json_extract |
+---------------+
| "Feature"     |
+---------------+
1 row in set (12.85 sec)
```

Explanation: Binary format of JSON type is very efficient at searching. Storing as TEXT performs over 10x worse at traversal.

New! JSON Functions

Functions to CREATE, SEARCH, MODIFY and RETURN JSON values:

JSON_ARRAY_APPEND()	JSON_INSERT()	JSON_REPLACE()
JSON_ARRAY_INSERT()	JSON_KEYS()	JSON_SEARCH()
JSON_ARRAY()	JSON_LENGTH()	JSON_SET()
JSON_CONTAINS_PATH()	JSON_MERGE()	JSON_TYPE()
JSON_CONTAINS()	JSON_OBJECT()	JSON_UNQUOTE()
JSON_DEPTH()	JSON_QUOTE()	JSON_VALID()
JSON_EXTRACT()	JSON_REMOVE()	

New! Generated Columns

id	my_integer	*my_integer_plus_one*
1	***10***	***11***
2	***20***	***21***
3	***30***	***31***
4	***40***	***41***

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 my_integer INT,
 my_integer_plus_one INT AS (my_integer+1)
);
```

Column automatically maintained based on your specification

JSON and Generated Columns

- Available as either VIRTUAL (default) or STORED:

```
ALTER TABLE features ADD feature_type varchar(30) AS (feature->>"$.type")
VIRTUAL;
Query OK, 206560 rows affected (4.70 sec)
Records: 206560  Duplicates: 0  Warnings: 0
```

- 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!

Have Both Schema + Schemaless!

- “Unstructured” is usually “semi-structured”
  - Some fixed-schema columns can complement flexible-schema JSON
  - Best of both worlds performance and flexibility

```
CREATE TABLE pc_components
(
  id INT NOT NULL PRIMARY KEY,
  description VARCHAR(60) NOT NULL,
  vendor VARCHAR(30) NOT NULL,
  serial_number VARCHAR(30) NOT NULL,
  attributes JSON NOT NULL
);
```

New! NoSQL

- Fluent API, method chaining, stateless sessions
- CRUD for Collections of Documents and Tables
  - Documents as simple basic domain objects
  - Search expressions match SQL SELECT expressions
- Implemented in MySQL Shell & MySQL X DevAPI Connectors
  - Javascript
  - Python
  - C#
  - Java
  - C++

Node.js Example

```
// Create a new collection
db.createCollection('myCollection').then(function(myColl)) {

  // Insert a document
  myColl.add( { name: 'Sakila', age: 21 } ).execute();

  // Insert several documents at once
  myColl.add( [
                 { name: 'Sastry', age: 45 }
                 { name: 'Nicolas', age: 25 }
               ] ).execute();
});

var myDocs = myColl.find('name like :name').bind('name', 'S%').execute();
```

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 automatically generated 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

SHOW CREATE TABLE `myCollection`\G

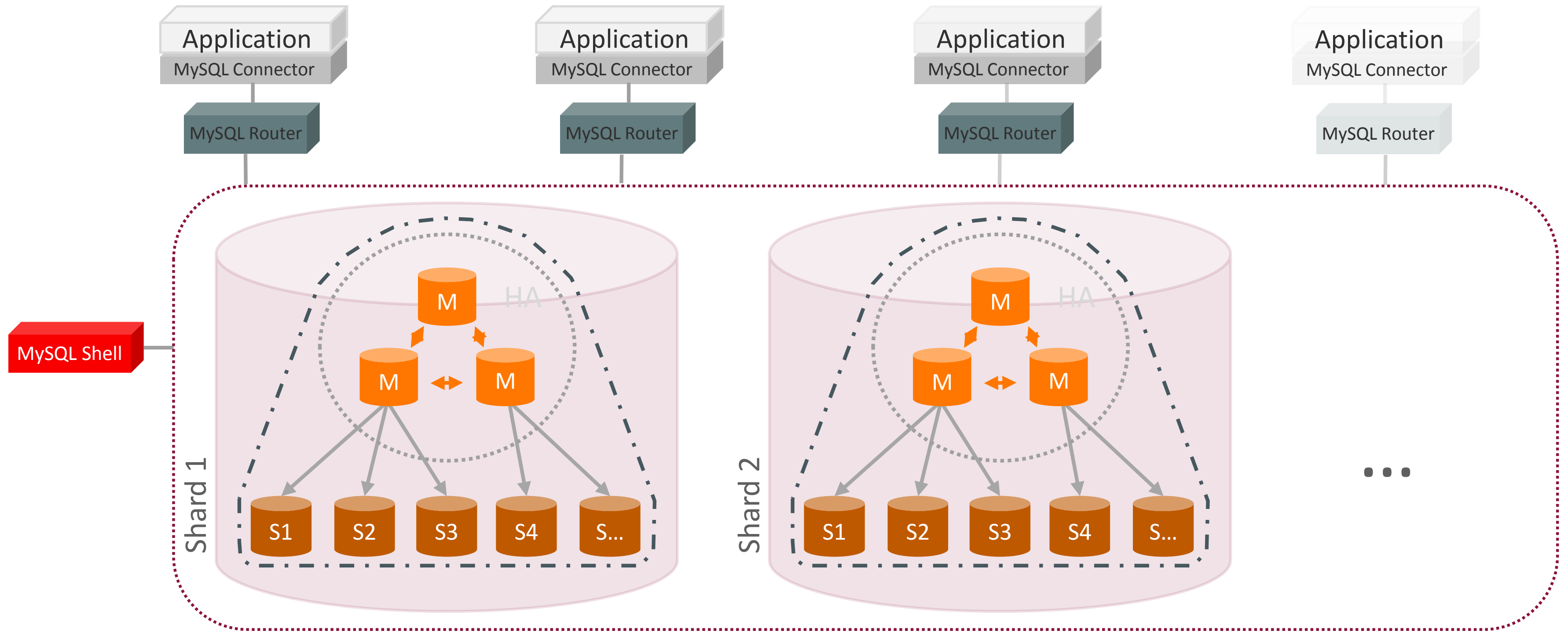
```
       Table: myCollection
Create Table: CREATE TABLE `myCollection`
(
  `doc` json DEFAULT NULL,
  `_id` varchar(32) GENERATED ALWAYS AS
        (json_unquote(json_extract(`doc`,'$._id')))
        STORED NOT NULL,
  PRIMARY KEY (`_id`),
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

MySQL™ InnoDB Cluster

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

ORACLE®

Long Term Goal: Automatically Sharded Document Store

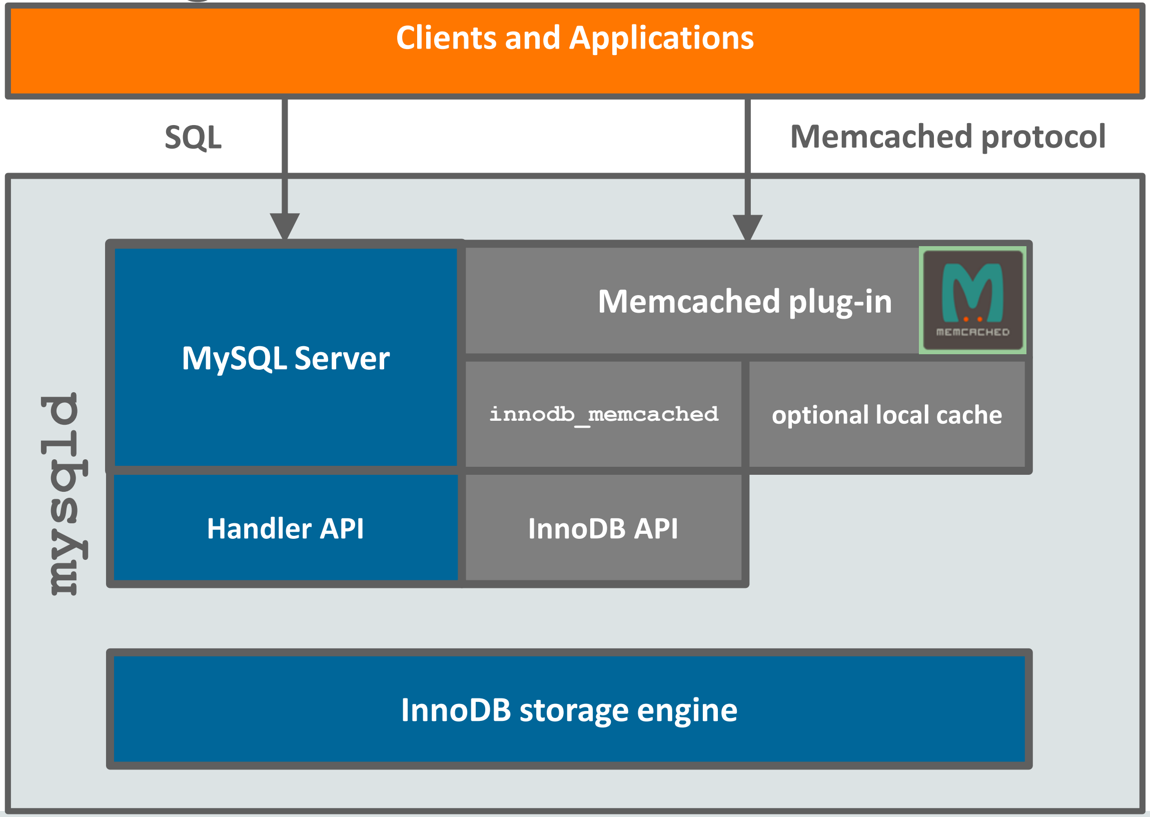


MySQL Document Store

- ✓ Built on Proven SQL/InnoDB/Replication
- ✓ Schema-less/Relational/Hybrid
- ✓ ACID/Transactions
- ✓ CRUD/JSON/Documents
- ✓ NoSQL and SQL
- ✓ Modern/Efficient Protocol
- ✓ SQL Queries/Analytics over JSON Documents
- ✓ Transparent and Easy HA/Scaling/Sharding

MySQL K/V Store

Memcached Plug-in for InnoDB



InnoDB Memcached Plug-in Tables

```
mysql> USE innodb_memcache;
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_innodb_memcache |
+---------------------------+
| cache_policies            |
| config_options            |
| containers                |
+---------------------------+

mysql> USE test;
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| demo_test      |
+----------------+
```

```
mysql> SELECT * FROM
innodb_memcache.containers\G
************* 1. row *************
                 name: aaa
            db_schema: test
             db_table: demo_test
          key_columns: c1
        value_columns: c2
                flags: c3
           cas_column: c4
   expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM
test.demo_test;
+----+------------------+------+------+------+
| c1 | c2               | c3   | c4   | c5   |
+----+------------------+------+------+------+
| AA | HELLO, HELLO     |    8 |    0 |    0 |
+----+------------------+------+------+------+
```

Basic Memcached Operations

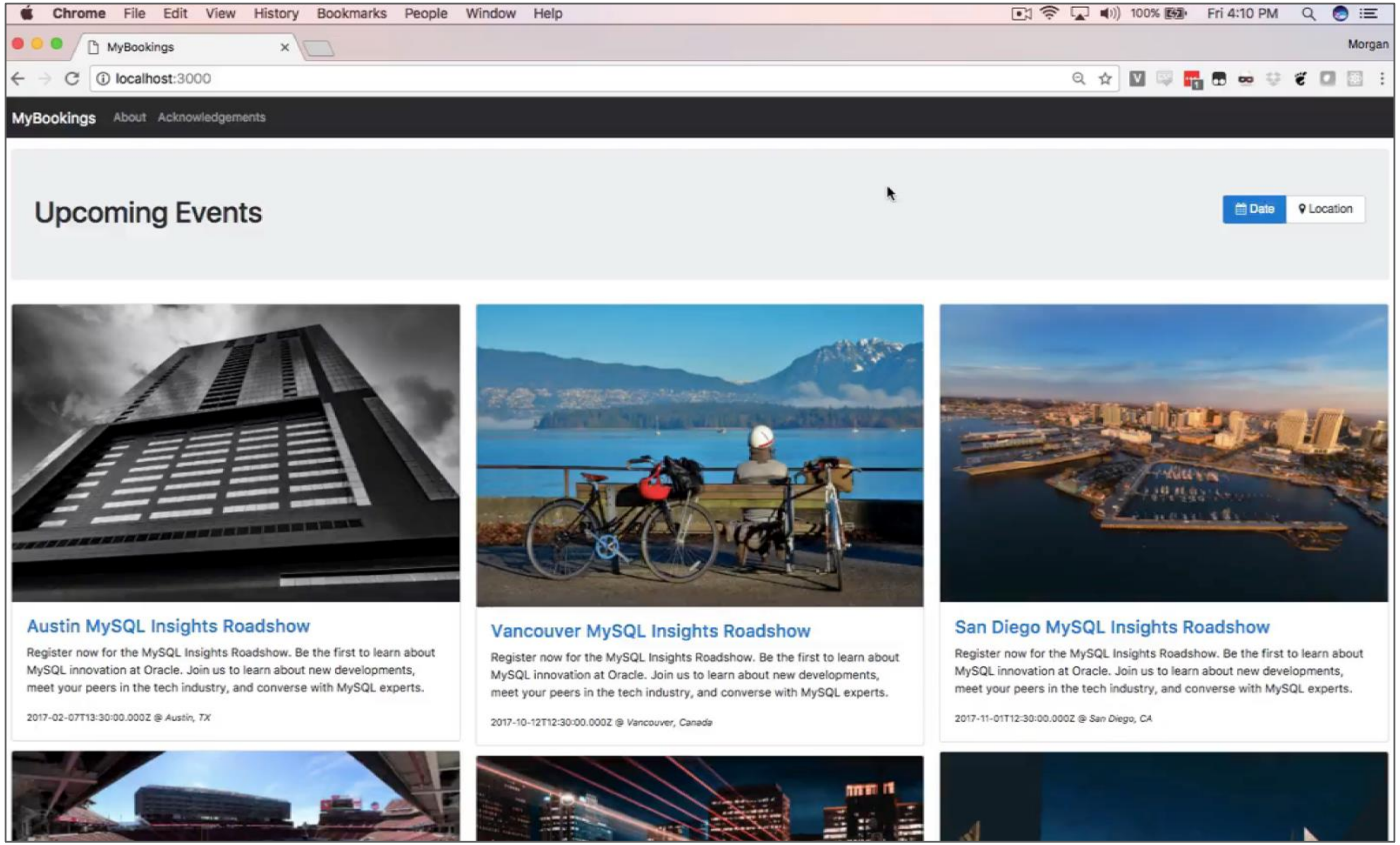
- All language interfaces support the following methods for storing and retrieving cache information:

Method	Purpose
`get(key)`	Retrieves the value for `key` if the key exists
`set(key, value, [expiry])`	Sets existing key to provided value, or adds a new item if the key does not exist
`add(key, value, [expiry])`	Adds a new key-value pair to cache
`replace(key, value, [expiry])`	Replaces the value associated with the key with the specified `value`
`delete(key, [time])`	Deletes the key-value pair
`incr(key, [value])`	Adds `1` or `value` to the value for specified key
`decr(key, [value])`	Subtracts `1` or `value` from the value for specified key
`flush_all()`	Expires all items in the cache

MySQL K/V Store

- ✓ Built on Proven SQL/InnoDB/Replication
- ✓ Schema-less/Relational/Hybrid
- ✓ ACID/Transactions
- ✓ CRUD/JSON/Documents
- ✓ Memcached and SQL

Mobile MySQL



Upcoming Events

Austin MySQL Insights Roadshow

Register now for the MySQL Insights Roadshow. Be the first to learn about MySQL innovation at Oracle. Join us to learn about new developments, meet your peers in the tech industry, and converse with MySQL experts.

2017-02-07T13:30:00.000Z @ *Austin, TX*

Vancouver MySQL Insights Roadshow

Register now for the MySQL Insights Roadshow. Be the first to learn about MySQL innovation at Oracle. Join us to learn about new developments, meet your peers in the tech industry, and converse with MySQL experts.

2017-10-12T12:30:00.000Z @ *Vancouver, Canada*

San Diego MySQL Insights Roadshow

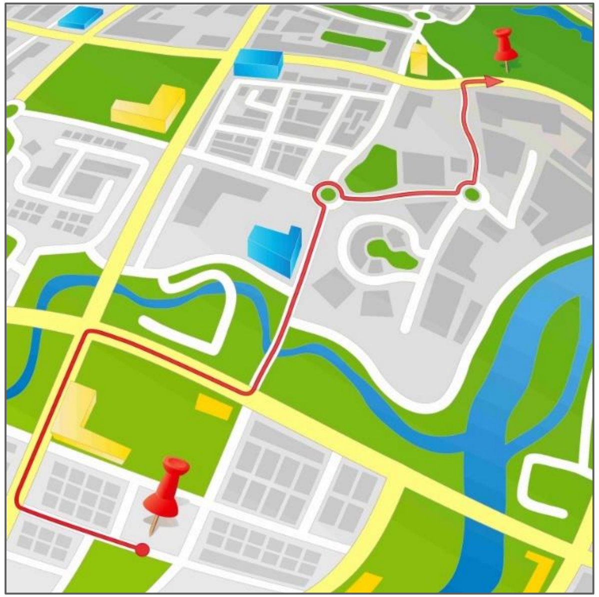
Register now for the MySQL Insights Roadshow. Be the first to learn about MySQL innovation at Oracle. Join us to learn about new developments, meet your peers in the tech industry, and converse with MySQL experts.

2017-11-01T12:30:00.000Z @ *San Diego, CA*

Geospatial Support (GIS)

- Spatial indexing in InnoDB
  - R-Tree bounding box implementation
  - Fully transactional support
  - Currently Eucledian plan, more later

```
CREATE TABLE events
(
 name VARCHAR(100),
 date TIMESTAMP,
 location GEOMETRY,
 SPATIAL KEY i_location(location)
);
```



Choosing The Nearest Event

```
SELECT
   name,
   venue,
   description,
   date,
   thumbnail,
   ST_distance_sphere(POINT(@X,@Y), location) AS distance
FROM
   events
ORDER BY distance;
```

SPATIAL KEY column

Choosing The Nearest Event Within 1 Mile

```
SELECT
  …
  ST_distance_sphere(POINT(@X,@Y), location) AS distance
FROM
  events
WHERE
  ST_Contains(ST_MakeEnvelope(POINT(@X+(1/69),@Y+(1/69)),
                              POINT(@X-(1/69),@Y-(1/69))),
              location)
ORDER BY distance;
```

Only locations within square bounding box

69 miles per degree

Agile Deployment

Containers

"A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure."

Making Your Own Containers

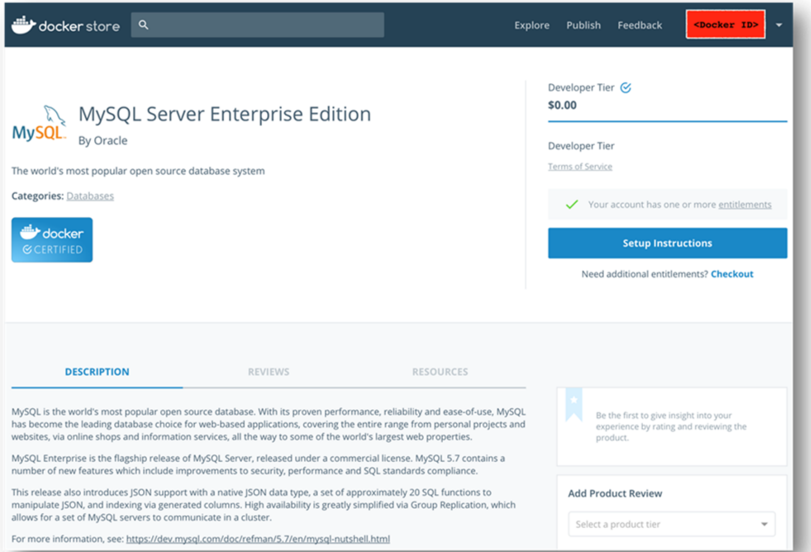
Dockerfile – where you define the container (also corresponding CLI arguments)

- ARG : arguments for use within the Dockerfile
- ENV : set environment variables for the container
- RUN : run command inside the container
- VOLUME : define volumes/mount points in the container
- ADD / COPY: add/copy files from the host to the container
- ENTRYFILE : where you define what’s run in the container when it starts
- CMD : the process to run inside the container
- EXPOSE : expose ports from the container
- HEALTHCHECK : exec something periodically to check the health of the container

Official MySQL Containers

- Official Server Release product
  - Part of each release, e.g. 5.7.20
  - Community and Enterprise
  - Fully supported
- Containers for all products
  - MySQL (NDB) Cluster
  - InnoDB Cluster
  - Router, Shell, Workbench, Utilities, …

MySQL Enterprise Edition



MySQL Server Enterprise Edition

By Oracle

The world's most popular open source database system

Categories: Databases

Developer Tier

$0.00

Developer Tier

Terms of Service

✓ Your account has one or more entitlements

Need additional entitlements? **Checkout**

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, covering the entire range from personal projects and websites, via online shops and information services, all the way to some of the world's largest web properties.

MySQL Enterprise is the flagship release of MySQL Server, released under a commercial license. MySQL 5.7 contains a number of new features which include improvements to security, performance and SQL standards compliance.

This release also introduces JSON support with a native JSON data type, a set of approximately 20 SQL functions to manipulate JSON, and indexing via generated columns. High availability is greatly simplified via Group Replication, which allows for a set of MySQL servers to communicate in a cluster.

For more information, see: https://dev.mysql.com/doc/refman/5.7/en/mysql-nutshell.html

Be the first to give insight into your experience by rating and reviewing the product.

Add Product Review

Thank you!

Nicolas De Rico

nicolas.de.rico@oracle.com