

ORACLE®

# MySQL 5.7 + JSON



Morgan Tocker  
MySQL Product Manager

# 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.

# Today's Agenda

- 1 Introduction
- 2 Core New JSON Features
- 3 To JSON or !JSON?

# MySQL 5.7 - The Team

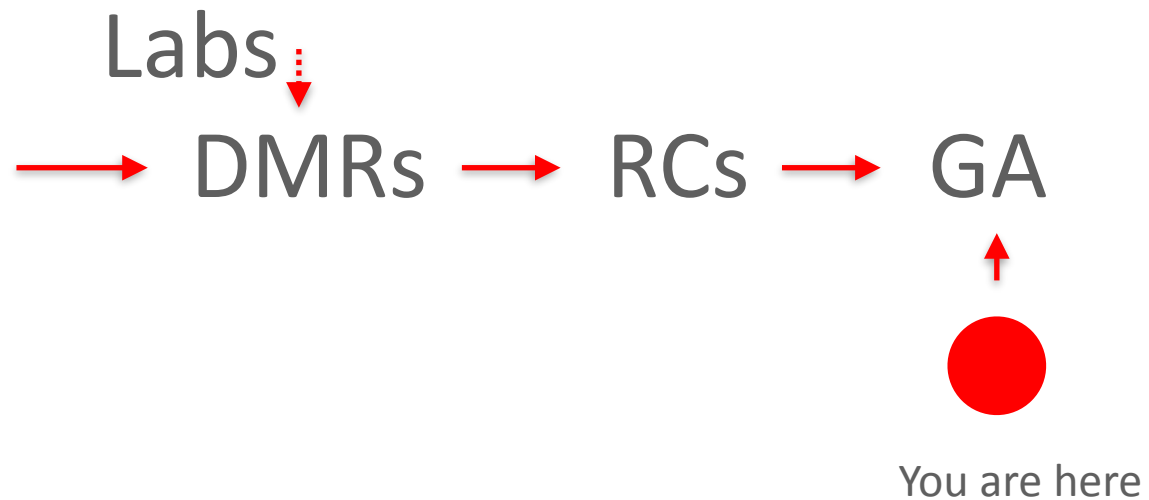
- **2x** Engineering Staff
- **3x** QA Staff
- **2x** Support Staff



Jan 2010

2015

# Release Process



# MySQL 5.7 is GA!

## Performance & Scalability

3 X Faster than MySQL 5.6

Enhanced InnoDB: faster online & bulk load operations

Replication Improvements (incl. multi-source, multi-threaded slaves...)

New Optimizer Cost Model: greater user control & better query performance

## Manageability

Native JSON Support

Improved Security: safer initialization, setup & management

Performance Schema Improvements

MySQL SYS Schema

Complete list of new features in MySQL 5.7

There are over 150 new features in MySQL 5.7.

The MySQL manual is very good, but verbose. This is a list of new features in short bullet form. I have tried very hard to make sure each feature is only mentioned once. So *InnoDB native partitioning* could be mentioned under either InnoDB or partitioning.

[Download MySQL 5.7.8 RC2](#)  
or install from [yum/apt](#) repo.

### Replication

1. Multi source replication [1]
2. Online GTID migration path [1 2 3]
3. Improved semi-sync performance [1 2]
4. Loss-less semi-sync replication [1 2]
5. Semi-sync can now wait for a configurable number of slaves [1]
6. Intra-schema parallel replication [1]
7. Ability to tune group commit via `binlog_group_commit_sync_delay` and `binlog_group_commit_sync_no_delay` options. [1 2]
8. Non-blocking `SHOW SLAVE STATUS` [1 2]
9. Online `CHANGE REPLICATION FILTER` [1]
10. Online `CHANGE MASTER TO` without stopping SQL thread [1]
11. Improved master-slave throughput performance [1 2]
12. Multi-threaded slave ordered commits (Sequential Consistency) [1]
13. Support `SLAVE_TRANSACTION_RETRIES` in multi-threaded slave mode [1]
14. A `WAIT_FOR_EXECUTED_GTID_SET` function has been introduced [1]
15. Optimize GTIDs for Passive Slaves [1 2]
16. Defaults change: `binlog_format=ROW`
17. Defaults change: `sync_binlog=1`
18. Defaults change: `binlog_gtid_simple_recovery=1`
19. Defaults change: `binlog_error_action=ABORT_SERVER`
20. Defaults change: `slave_net_timeout=60`



SQL thread [1]

- Improved master dump thread performance

InnoDB

- Online buffer pool resize [1]
- Improved crash recovery performance [1]
- Improved read-only transaction scalability [1 2 3 4]
- Improved read-write transaction scalability [1 2 3 4]
- Several optimizations for high performance temporary tables [1 2 3 4 5]
- ALTER TABLE RENAME INDEX** only requires meta-data change [1]
- Inplace **VARCHAR** only requires meta-data change [1]
- ALTER TABLE** performance improved [1]
- Multiple `page_cleaner` threads [1]
- Optimized buffer pool flushing [1]
- New `innodb_log_checksum_algorithm` option [1]
- Improved NUMA support [1]
- General Tablespace support [1]
- Transparent Page Compression [1]
- `innodb_log_write_ahead_size` introduced to address potential 'read-on-write' with redo logs [1]
- Fulltext indexes now support pluggable parsers [1]
- Support for ngram and MeCab full-text parser

- Defaults change: `slave_net_timeout=60`

`innodb_buffer_dump_pct` [1]

- The doublewrite buffer is now disabled on filesystems that supports atomic writes (aka Fusion-io support) [1]
- Page fill factor is now configurable [1]
- Support for 32K and 64K pages [1]
- Online undo log truncation [1]
- `Update_time` meta data is now updated [1]
- TRUNCATE TABLE** is now atomic [1]
- Memcached API performance improvements [1]
- InnoDB now implements `information_schema.files` [1]
- Defaults change:  
`innodb_file_format=Barracuda`
- Defaults change: `innodb_large_prefix=1`
- Defaults change:  
`innodb_page_cleaners=4`
- Defaults change:  
`innodb_purge_threads=4`
- Defaults change:  
`innodb_buffer_pool_dump_at_shutdown`
- Defaults change:  
`innodb_buffer_pool_load_at_startup`
- Defaults change:  
`innodb buffer pool dump pct=25`

Complete list of new featur... x

www.thecompletestlistoffeatures.com

Morgan

18. Fulltext search optimizations [1]

19. Buffer pool dump now supports

**Optimizer**

- Improved optimizer cost model, leading to more consistently better query plans [1 2 3 4]
- Optimizer cost constants are now configurable on a global or per engine basis [1 2]
- Query parser has been refactored and improved [1]
- EXPLAIN FOR CONNECTION** [1]
- UNION ALL** does not use a temporary table [1]
- Filesort is now optimized to pack values [1]
- Subqueries in **FROM** clause handled same as a view [1]
- Queries using row value constructors are now optimized [1 2]
- Optimizer now supports a new condition filtering optimization [1 2]
- EXPLAIN FORMAT=JSON** now shows cost information [1]
- Support for **STORED** and **VIRTUAL** generated columns (aka functional indexes) [1]
- Prepared statements refactored internally and

**Security**

- Username size increased to 32 characters [1]

36. Defaults change:  
innodb\_checksum\_algorithm=crc32

performance improved [1 2]

- New query hints using comment-like `/** */` syntax [1]
- Server-side query rewrite framework [1]
- ONLY\_FULL\_GROUP\_BY** now more standards compliant [1]
- Support for gb18030 character set [1]
- Defaults change:  
internal\_tmp\_disk\_storage\_engine=InnoDB [1]
- Defaults change:  
eq\_range\_index\_dive\_limit=200
- Defaults change:  
sql\_mode=ONLY\_FULL\_GROUP\_BY,  
STRICT\_TRANS\_TABLES,  
NO\_ZERO\_IN\_DATE, NO\_ZERO\_DATE,  
ERROR\_FOR\_DIVISION\_BY\_ZERO,  
NO\_AUTO\_CREATE\_USER,  
NO\_ENGINE\_SUBSTITUTION
- Defaults change: **EXTENDED** and **PARTITIONS** keywords for **EXPLAIN** enabled by default

installation

The screenshot shows a web browser window with the address bar displaying 'www.thecompleteoffsetoffeatures.com'. The page content is organized into three main sections: Security, Performance Schema, and GIS. Each section contains a numbered list of features. The Security section lists 16 items, including changes to username size, support for IF [NOT] EXISTS clause, secure transport options, AES encryption modes, user account locking/unlocking, password expiration policy, password strength enforcement, and removal of test database and anonymous users. The Performance Schema section lists 15 items, focusing on reducing overhead and memory footprint, improving pfs\_lock, batching IO statistics, and various instrumentation features. The GIS section lists 5 items, including spatial datatypes and expanded spatial analysis functions.

**Security**

1. Username size increased to 32 characters [1]
2. Support for `IF [NOT] EXISTS` clause in `CREATE/DROP USER` [1]
3. Server option to require secure transport [1]
4. Support for multiple AES Encryption modes [1 2]
5. Support to `LOCK/UNLOCK` user accounts [1 2]
6. Support for password expiration policy [1 2]
7. Password strength enforcement
8. test database no longer created on installation
9. Anonymous users no longer created on installation
10. MySQL Firewall (commercial)
11. Random password generated by default on installation
12. New `ALTER USER` command
13. `SET password= ''` now accepts a password instead of hash
14. Server now generates SSL keys by default
15. Insecure `old_password` hash removed
16. Ability to create utility users for stored programs that can not login [1]

**Performance Schema**

1. Overhead has been reduced in client connect/disconnect phases
2. Memory footprint has been reduced
3. `pfs_lock` implementation has been improved
4. Table IO statistics are now batched for improved performance
5. Memory usage instrumentation
6. Stored programs instrumentation
7. Replication instrumentation
8. Metadata Locking (MDL) instrumentation
9. Transactions
10. Prepared Statements
11. Stage Progress
12. `SX-lock` and `rw_lock`
13. Thread status and variables
14. Defaults change: `performance-schema-consumer-events_statements_history=ON`
15. Defaults change: `performance-schema-consumer-events_transactions_history=ON`

**GIS**

1. InnoDB supports indexing of spatial datatypes [1]
2. MySQL Geometry Data Type [1]
3. MySQL Geometry Functions [1]
4. MySQL Geometry Operators [1]
5. Expanded spatial analysis functions [1]

Complete list of new features

www.thecompletestlistoffeatures.com

## GIS

1. InnoDB supports indexing of spatial datatypes [1]
2. GIS now based on Boost Geometry [1]
3. Expanded spatial relation check functions [1]
4. Expanded geometry set operations [1]
5. Expanded spatial analysis functions [1]
6. WKB Geometry Container [1]
7. Geohash functions [1]
8. GeoJSON functions [1]

## Triggers

1. Multiple triggers per event per table [1]
2. **BEFORE** Triggers are not processed for **NOT**

**NULL** columns [1]

## Partitioning

1. Index condition pushdown optimization now supported
2. **HANDLER** command is now supported
3. **WITHOUT VALIDATION** option now supported for **ALTER TABLE ... EXCHANGE**
4. Support for Transportable Tablespaces
5. Partitioning is now storage-engine native for InnoDB

**PARTITION**

## SYS (new)

1. **SYS** schema bundled by default
2. 100 new views, 21 new stored functions and 26 new stored procedures to help understand

and interact with Performance Schema and Information Schema [1]

## JSON (new)

1. Native JSON Data Type [1]
2. JSON Comparator
3. 3 new functions that create JSON values [1]
4. 5 new functions that create JSON values [1]
5. 8 new functions that modify JSON values [1]
6. 4 new functions that return JSON value attributes [1]

Complete list of new features

www.thecompletestlistoffeatures.com

- 3 new functions that create JSON values [1]
- attributes [1]
- 5 new functions that search JSON values [1]

### Client Programs

1. New `mysqlpump` utility
2. The `mysql` client now supports `Ctrl+C` to clear statement buffer
3. `rewrite-db` option added to `mysqlbinlog`
4. New utility to help setup SSL
5. SSL support added to `mysqlbinlog`
6. Idempotent mode added to `mysqlbinlog`
7. Client `--ssl` changed to now force SSL
8. Enhancements to the `innochecksum` utility
9. Removal of several outdated/unsafe command line utilities [1]
10. Many Perl command line clients converted to C++
11. Client Side Protocol Tracing
12. Client API method to reset connection

### Misc

1. Server new connection throughput improved considerably [1]
2. `mysql_install_db` replaced by `mysqld -initialize`
3. Native support for `syslog`
4. Native support for `systemd`
5. `disabled_storage_engines` option allows a block list of engines
6. `SET GLOBAL offline_mode=1`
7. `super_read_only` option
8. Detect transaction boundaries
9. Server version token and check
10. `SELECT GET_LOCK()` can now acquire multiple locks
11. Configurable maximum statement execution time on a global and per query basis [1 2]
13. DTrace support [1]
14. More consistent `IGNORE` clause and `STRICT` mode
15. A number of tables in the `mysql` schema have moved from `MyISAM` to `InnoDB`
16. Server error log format improved to be consistent
17. Compiler switched to `GCC` on `Solaris`
18. Extract query digest moved from `performance_schema` into the server directly
19. Improved scalability of meta data locking
20. Increased control over error log verbosity
21. Stacked Diagnostic Areas
22. Defaults change: `log_warnings=2`
23. Defaults change: `table_open_cache_instances=1`

# Today's Agenda

- 1 Introduction
- 2 Core New JSON Features
- 3 To JSON or !JSON?

# Core New JSON features in MySQL 5.7

- Native JSON datatype
- JSON Functions
- Generated Columns

# The JSON 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 Type Tech Specs

- 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

# JSON Type Tech Specs (cont.)

- Supports all native JSON types
  - Numbers, strings, bool
  - Objects, arrays
- Extended
  - Date, time, datetime, timestamp
  - Other

# Advantages over TEXT/VARCHAR

## 1. 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'.
```

## 2. Efficient Binary Format

Allows quicker access to object members and array elements

# JSON Functions

```
SET @document = '[10, 20, [30, 40]]';
```

```
SELECT JSON_EXTRACT(@document, '$[1]');
```

```
+-----+
| JSON_EXTRACT(@document, '$[1]') |
+-----+
| 20                                |
+-----+
```

```
1 row in set (0.01 sec)
```

# JSON\_EXTRACT

- Accepts a *JSON Path*, which is similar to a selector:



`$("#type")`



`JSON_EXTRACT  
(column_name, "$.type")`

- JSON\_EXTRACT also supports a short hand:  
`column_name->"$.type"`

# Using Real Life Data

- Via **SF OpenData**
- 206K JSON objects representing subdivision parcels.



```
CREATE TABLE features (  
  id INT NOT NULL auto_increment primary key,  
  feature JSON NOT NULL  
);
```

- Imported from <https://github.com/zemirco/sf-city-lots-json> + small tweaks

```
{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [-122.42200352825247, 37.80848009696725, 0],
        [-122.42207601332528, 37.808835019815085, 0],
        [-122.42110217434865, 37.808803534992904, 0],
        [-122.42106256906727, 37.80860105681814, 0],
        [-122.42200352825247, 37.80848009696725, 0]
      ]
    ]
  },
  "properties": {
    "TO_ST": "0",
    "BLKLOT": "0001001",
    "STREET": "UNKNOWN",
    "FROM_ST": "0",
    "LOT_NUM": "001",
    "ST_TYPE": null,
    "ODD_EVEN": "E",
    "BLOCK_NUM": "0001",
    "MAPBLKLOT": "0001001"
  }
}
```



# JSON Search

```
# Basic Find
```

```
SELECT * FROM features
```

```
WHERE feature->"$.properties.STREET" = 'MARKET'
```

```
LIMIT 1\G
```

```
***** 1. row *****
```

```
id: 12250
```

```
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"}}
```

```
1 row in set (0.02 sec)
```



# JSON Search

```
# Find where not exists
SELECT * FROM features
WHERE feature->"$.properties.STREET" IS NULL
LIMIT 1\G
Empty set (0.39 sec)
```

# Naive Performance Comparison

Unindexed traversal of 206K documents

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

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

Using short cut for  
JSON\_EXTRACT.

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

# Introducing Generated Columns

<u>id</u>	my_integer	my_integer_plus_one
1	10	11
2	20	21
3	30	31
4	40	41

Column automatically maintained based on your specification.

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

```
UPDATE t1 SET my_integer_plus_one = 10 WHERE id = 1;  
ERROR 3105 (HY000): The value specified for generated  
column 'my_integer_plus_one' in table 't1' is not  
allowed.
```

Read-only of course

# Generated Columns Support Indexes!

From table scan on 206K documents to index scan on 206K materialized values

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

```
Query OK, 0 rows affected (0.01 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

Meta data change only (FAST). Does not need to touch table.

```
ALTER TABLE features ADD INDEX (feature_type);
```

```
Query OK, 0 rows affected (0.73 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

Creates index only. Does not modify table rows.

```
SELECT DISTINCT feature_type FROM features;
```

```
+-----+  
| feature_type |  
+-----+  
| "Feature"    |  
+-----+
```

```
1 row in set (0.06 sec)
```

Down from 1.25 sec to 0.06 sec

# Generated Columns (cont.)

- Used for “functional index”
- Available as either VIRTUAL (*default*) or STORED:

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

```
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.

# Indexing Options Available

## STORED

Primary and Secondary

BTREE, Fulltext, GIS

Mixed with fields

Requires table rebuild

Not Online

## VIRTUAL

Secondary Only

BTREE Only

Mixed with fields

No table rebuild

INSTANT Alter

Faster Insert

**Bottom Line:** Unless you need a PRIMARY KEY, FULLTEXT or GIS index VIRTUAL is probably better.

# Unquote JSON String

```
SELECT
  DISTINCT JSON_UNQUOTE(feature->"$.type")
  as feature_type
FROM features;
```

```
+-----+
| feature_type |
+-----+
| Feature     |
+-----+
```

```
1 row in set (1.22 sec)
```

# JSON Path Search

- Provides a novice way to know the path. To retrieve via:  
*[[database.]table.]column->"\$<path spec>"*

```
SELECT JSON_SEARCH(feature,  
  'one', 'MARKET') AS  
  extract_path  
FROM features  
WHERE id = 121254;
```

```
+-----+  
| extract_path |  
+-----+  
| "$.properties.STREET" |  
+-----+  
1 row in set (0.00 sec)
```



```
SELECT  
feature->"$.properties.STREET"  
  AS property_street  
FROM features  
WHERE id = 121254;
```

```
+-----+  
| property_street |  
+-----+  
| "MARKET" |  
+-----+  
1 row in set (0.00 sec)
```



# JSON Array Creation

```
SELECT JSON_ARRAY(id,  
  feature->"$.properties.STREET",  
  feature->'$.type') AS json_array  
FROM features ORDER BY RAND() LIMIT 3;
```

```
+-----+  
| json_array |  
+-----+  
| [65298, "10TH", "Feature"] |  
| [122985, "08TH", "Feature"] |  
| [172884, "CURTIS", "Feature"] |  
+-----+
```

```
3 rows in set (2.66 sec)
```

# JSON Object Creation

```
SELECT JSON_OBJECT('id', id,  
  'street', feature->"$.properties.STREET",  
  'type', feature->"$.type"  
) AS json_object  
FROM features ORDER BY RAND() LIMIT 3;
```

```
+-----+  
| json_object |  
+-----+  
| {"id": 122976, "type": "Feature", "street": "RAUSCH"} |  
| {"id": 148698, "type": "Feature", "street": "WALLACE"} |  
| {"id": 45214, "type": "Feature", "street": "HAIGHT"} |  
+-----+
```

```
3 rows in set (3.11 sec)
```

# JSON\_REPLACE

```
SELECT JSON_REPLACE(feature, '$.type',  
JSON_ARRAY('feature', 'bug')) as json_object FROM features  
LIMIT 1;
```

```
+-----+  
| json_object |  
+-----+  
| {"type": ["feature", "bug"], "geometry": {"type": ..}} |  
+-----+
```

# JSON Functions

- 5.7 supports 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()

<https://dev.mysql.com/doc/refman/5.7/en/json-functions.html>

# JSON Comparator

```
SELECT CAST(1 AS JSON) = 1;
```

```
+-----+
| CAST(1 AS JSON) = 1 |
+-----+
|                      1 |
+-----+
1 row in set (0.01 sec)
```

JSON value of 1 equals 1

JSON Objects Compare

```
SELECT CAST('{ "num": 1.1 }' AS JSON) = CAST('{ "num": 1.1 }' AS JSON);
```

```
+-----+
| CAST('{ "num": 1.1 }' AS JSON) = CAST('{ "num": 1.1 }' AS JSON) |
+-----+
|                                                                    1 |
+-----+
1 row in set (0.00 sec)
```

# Today's Agenda

- 1 Introduction
- 2 Core New JSON Features
- 3 To JSON or !JSON?**

# JSON or Column?

- Up to you!
- Advantages to both approaches

# Storing as a *Column*

- Easier to apply a *schema* to your application
- *Schema* may make applications easier to maintain over time, as change is **controlled**;
  - Do not have to expect as many permutations
  - Allows some constraints over data



# Storing as *JSON*

- More flexible way to represent data that is hard to model in *schema*;
  - Imagine you are a SaaS application serving many customers
  - Strong use-case to support **custom-fields**
  - Historically this may have used Entity–attribute–value model (EAV). Does not always perform well

## JSON (cont.)

- Easier *denormalization*; an optimization that is important in some specific situations
- No painful schema changes\*
- Easier prototyping
  - Fewer types to consider
  - No enforced schema, start storing values immediately

\* MySQL 5.6 has Online DDL. This is not as large of an issue as it was historically.

# Schema + Schemaless

```
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  
);
```

SSDs have *capacity\_in\_gb*, CPUs have a *core\_count*. These attributes are not consistent across products.

# Road Map

- In-place partial update of JSON/BLOB (*performance*)
- Partial streaming of JSON/BLOB (*replication*)
- Full text and GIS index on virtual columns
  - Currently works for "STORED"
- Multi-value Index

# Resources

- <http://mysqlserverteam.com/>
- <http://mysqlserverteam.com/tag/json/>
- <https://dev.mysql.com/doc/refman/5.7/en/mysql-nutshell.html>
- <http://dev.mysql.com/doc/relnotes/mysql/5.7/en/>
- <https://dev.mysql.com/doc/refman/5.7/en/json.html>
- <https://dev.mysql.com/doc/refman/5.7/en/json-functions.html>
- <http://www.thecompletelistoffeatures.com>

# **Hardware and Software Engineered to Work Together**

ORACLE®