MySQL 8
Tips and Tricks

Dave Stokes
@stoker
david.stokes@oracle.com
Elephantdolphin.blogger.com
OpensourceDBA.wordpress.com

MySQL

ORACLE

What This Talk Is About??

MySQL 8 Features

This is not a simple talk on performance tuning a database or a cookbook where you set X to Y and get Z percent better performance.

Instead this a talk about developments that have the potential to make big changes in the way you use MySQL Instances.

Simple Answer:

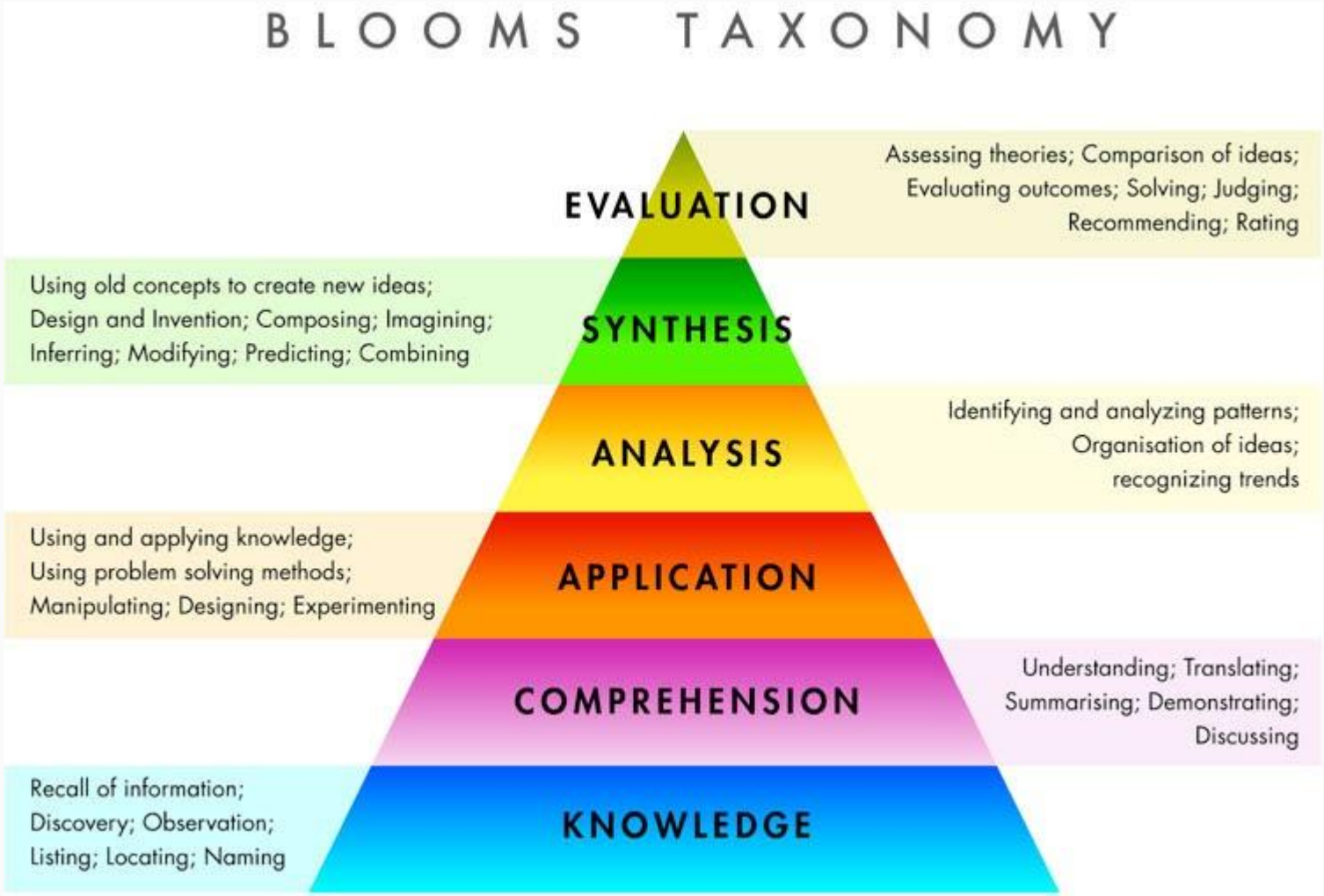
Set INNODB_BUFFER_POOL_SIZE To ~ 80% of RAM

Quick cl setting:

```
mysql> SELECT
@@innodb_buf /1024;
```

Simple answers are great if

… you only live in a simple world!


BLOOMS TAXONOMY
EVALUATION
Assessing theories; Comparison of ideas; Evaluating outcomes; Solving; Judging; Recommending; Rating
SYNTHESIS
Using old concepts to create new ideas; Design and Invention; Composing; Imagining; Inferring; Modifying; Predicting; Combining
ANALYSIS
Identifying and analyzing patterns; Organisation of ideas; recognizing trends
APPLICATION
Using and applying knowledge; Using problem solving methods; Manipulating; Designing; Experimenting
COMPREHENSION
Understanding; Translating; Summarising; Demonstrating; Discussing
KNOWLEDGE
Recall of information; Discovery; Observation; Listing; Locating; Naming

Quiet Database Revolution

Cloud

NoSQL

Security

Self-tuning

1. Upgrade

Minor Interruption

Please excuse this small rant about help forums!

Hi!

I know nothing about brain surgery but ….

I popped the top of the skull off my coworker in an attempt to adjust their attitudes.

How do I do make those adjustments?

And what is the red stuff leaking on the carpet?

I have an Ikea allen wrench, a screwdriver, and some duct tape!

Please advise ASAP as the coworker is vital to production.

And how do you clean a carpet??

Many questions on sites like Quora.com and Stackoverflow.com are … frustrating

End of Rant

Big Changes Behind the Scenes

https://stackoverflow.com/questions/50505236/mysql-8-0-group-by-performance

To compare MySQL 5.7 and 8.0 I created a table using sysbench. And I tried the test. The performance of the server is exactly the same As a result, oltp_point_select showed almost similar performance.

However, when doing the group by tests below, MySQL 8.0 showed *10 times better performance*.

But I do not know why it is fast.

I do not know if I can find the MySQL 8.0 Release Notes. In 8.0, who will tell me why group by are faster?

Oystein Answers

MySQL 8.0 uses a new storage engine, TempTable, for internal temporary tables. (See MySQL Manual for details.) This engine does not have a max memory limit per table, but a common memory pool for all internal tables. It also has its own overflow to disk mechanism, and does not overflow to InnoDB or MyISAM as earlier versions.

The profile for 5.7 contains "converting HEAP to ondisk". This means that the table reached the max table size for the MEMORY engine (default 16 MB) and the data is transferred to InnoDB. Most of the time after that is spent accessing the temporary table in InnoDB. In MySQL 8.0, the default size of the memory pool for temporary tables is 1 GB, so there will probably not be any overflow to disk in that case.

Please Upgrade

Besides the obvious security and bug updates there are some major improvements waiting for you in MySQL 8

2. Data Dictionary

Metadata before 8

MySQL Server incorporates a transactional data dictionary that stores information about database objects. In previous MySQL releases, dictionary data was stored in metadata files, non transactional tables, and storage engine-specific data dictionaries.

Metadata was kept in a series of files --- eatinging up inodes, getting damaged or deleted at the wrong time, and hard to fix

Data Dictionary

Benefits of the MySQL data dictionary include:

- Simplicity of a centralized data dictionary schema that uniformly stores dictionary data.
- Removal of file-based metadata storage.
- Transactional, crash-safe storage of dictionary data. Uniform and centralized caching for dictionary objects. A simpler and improved implementation for some INFORMATION_SCHEMA tables.
- Atomic DDL.

Big Change

Good news: You can now have millions of tables within a schema

Bad news: You can now have millions of tables within a schema

Instant Add Column

This INSTANT ADD COLUMN patch was contributed by the Tencent Games DBA Team. We would like to thank and acknowledge this important and timely contribution by Tencent Games.

Bye Bye Bug #199

No more Innodb auto_increment stats loss

3. CATS

Contention Aware Transaction Schedule

https://arxiv.org/pdf/1602.01871.pdf

Identifying the Major Sources of Variance in Transaction Latencies: Towards More Predictable Databases -- University of Michigan

The CATS algorithm is based on a simple intuition: **not all transactions are equal**, and not all objects are equal. **When a transaction already has a lock on many popular objects, it should get priority when it requests a new lock**. In other words, unblocking such a transaction will indirectly contribute to unblocking many more transactions in the system, which means higher throughput and lower latency overall.

Indexes Versus Histograms

Indexes are great but have a *cost* at *insert update, delete*, and at *statistic gathering time*.

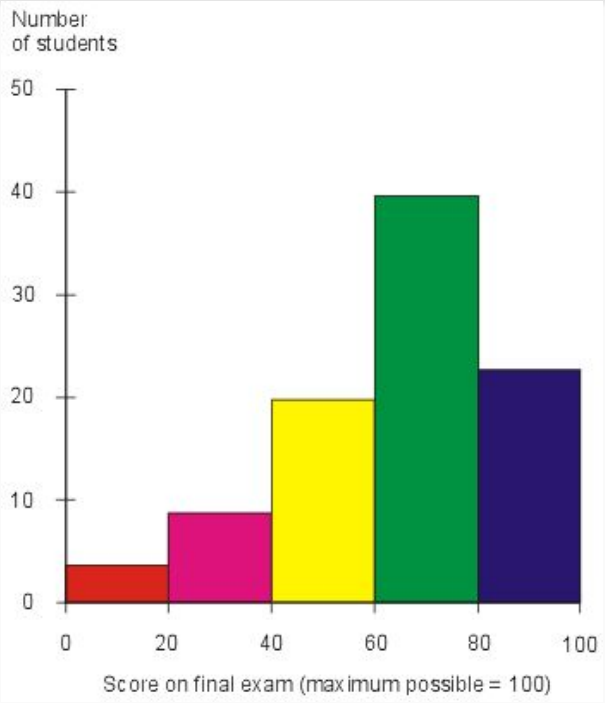
Histograms can be run after major changes to data or at slack times.

Histograms

The query optimizer needs statistics to create a query plan.

- How many rows are there in each table?
- How many distinct values are there in each column?
- How is the data distributed in each column?

What is a Histogram?



What is a Histogram?

A *histogram* is an approximation of the data distribution for a column. It can tell you with a reasonably accuray whether your data is skewed or not, which in turn will help the database server understand the nature of data it contains.

MySQL has chosen to support two different types: The “singleton” histogram and the “equi-height” histogram. Common for all histogram types is that they split the data set into a set of “buckets”, and MySQL automatically divides the values into buckets, and will also automatically decide what type of histogram to create.

Syntax

ANALYZE TABLE tbl_name
UPDATE HISTOGRAM ON col_name [, col_name]
WITH N BUCKETS;

ANALYZE TABLE tbl_name **DROP HISTOGRAM** ON col_name [, col_name];

4. Invisible Indexes

What is an Invisible Index?

Indexes can be marked as ‘invisible’ to the optimizer

Use EXPLAIN to see query plan and tell if index aids or hinders query

```
ALTER TABLE t1 ALTER INDEX i_idx INVISIBLE;
ALTER TABLE t1 ALTER INDEX i_idx VISIBLE;
```

5. Replacing Many-to-many joins with JSON

Relational Database + JSON Fields (hybrid)

Leverage power of RDMS but augmented with JSON fields

- Use JSON to eliminate one of the issues of traditional relational databases -- the many-to-many join
- Allows more freedom to store unstructured data (data with pieces missing)
- You can still use SQL to work with the data via a database connector but the JSON documents can be manipulated directly in code

JSON Document Tips

- Minimize joins - *reducing how many joins you need can speed up queries*. Faster access over data denormalization
- Plan for mutability - Schema-less design are based mutability. Build your applications with the ability to change the document as needed (and within reason)
- Use embedded arrays and lists to store relationship among documents
  - This can be as simple as embedding the data in document or embedding an array of document ids in the document. In the first case the data is available when you read the document. In the second, it takes only one more step to get the data.
  - In cases of seldom read (used) relationships the array of ids is more efficient as there is less data to read on the first pass

Quick Example

Customer table -- ID

Address -- Address1 .. n

Phone -- Phone1..n

Payment -- Bank1...n

4 or more reads to process an order

Customer table -- ID

JSON docs -- Address, Phone, Payment

1 read

6. Resource Groups

Resource Groups

MySQL supports creation and management of resource groups, and permits assigning threads running within the server to particular groups so that threads execute according to the resources available to the group.

Group attributes enable control over its resources, to enable or restrict resource consumption by threads in the group. DBAs can modify these attributes as appropriate for different workloads.

Resource Groups

Currently, CPU time is a manageable resource, represented by the concept of "virtual CPU" as a term that includes CPU cores, hyperthreads, hardware threads, and so forth.

The *server determines at startup how many virtual CPUs are available*, and database administrators with appropriate privileges *can associate these CPUs with resource groups and assign threads to groups*.

Create a Resource Group

```
CREATE RESOURCE GROUP Batch
  TYPE = USER
  VCPU = 2-3 -- assumes a system with at least 4 CPUs
  THREAD_PRIORITY = 10;
```

Using a Resource Group

```
INSERT /*+ RESOURCE_GROUP(Batch) */ INTO t2 VALUES(2);
```

9. Autonomy

Self Tuning Databases

Databases are getting better at realizing their environments (cores, disks, busses, virtual, container, buffers), loads, query patterns, and networks.

You will see much more of this much sooner than you would expect.

"I expect you all to be independent, innovative, critical thinkers who will do exactly as I say!"

The Payoff is less ...

Human Labor

Human Error

No Manual Labor

10. JSON Updates

JSON Data Type Extremely Popular

Introduced in MySQL 5.7, the JSON data type provides a 1GB document store in a column of a row in a table.

Over thirty functions to support JSON data types

The foundation on the MySQL Document Store, a NoSQL JSON document store

Inplace Update of JSON columns

In MySQL 8.0, the optimizer can perform a partial, in-place update of a JSON column instead of removing the old document and writing the new document in its entirety to the column.

New JSON Functions

JSON_PRETTY

JSON array and object aggregations

JSON_SIZE and JSON_FREE

Change in JSON_MERGE : JSON_MERGE_PRESERVE and JSON_MERGE_PATCH

The JSON Functions

Name	Description
JSON_ARRAY()	Create JSON array
JSON_ARRAY_APPEND()	Append data to JSON document
JSON_ARRAY_INSERT()	Insert into JSON array
->	Return value from JSON column after evaluating path; equivalent to JSON_EXTRACT().
JSON_CONTAINS()	Whether JSON document contains specific object at path
JSON_CONTAINS_PATH()	Whether JSON document contains any data at path
JSON_DEPTH()	Maximum depth of JSON document
JSON_EXTRACT()	Return data from JSON document
->>	Return value from JSON column after evaluating path and unquoting the result; equivalent to JSON_UNQUOTE(JSON_EXTRACT()).
JSON_INSERT()	Insert data into JSON document
JSON_KEYS()	Array of keys from JSON document
JSON_LENGTH()	Number of elements in JSON document
JSON_MERGE() (deprecated 8.0.3)	Merge JSON documents, preserving duplicate keys. Deprecated synonym for JSON_MERGE_PRESERVE()
JSON_MERGE_PATCH()	Merge JSON documents, replacing values of duplicate keys
JSON_MERGE_PRESERVE()	Merge JSON documents, preserving duplicate keys
JSON_OBJECT()	Create JSON object
JSON_PRETTY()	Prints a JSON document in human-readable format, with each array element or object member printed on a new line, indented two spaces with respect to its parent.
JSON_QUOTE()	Quote JSON document
JSON_REMOVE()	Remove data from JSON document
JSON_REPLACE()	Replace values in JSON document
JSON_SEARCH()	Path to value within JSON document
JSON_SET()	Insert data into JSON document
JSON_STORAGE_FREE()	Freed space within binary representation of a JSON column value following a partial update
JSON_STORAGE_SIZE()	Space used for storage of binary representation of a JSON document; for a JSON column, the space used when the document was inserted, prior to any partial updates
JSON_TABLE()	Returns data from a JSON expression as a relational table
JSON_TYPE()	Type of JSON value
JSON_UNQUOTE()	Unquote JSON value
JSON_VALID()	Whether JSON value is valid

JSON_TABLE

JSON_TABLE takes schema-less JSON documents and turn it into a temporary relational table that can be processed like any other relational table.

```
mysql> select country_name,
              IndyYear
              from countryinfo,
          json_table(doc,"$" columns
                    (country_name char(20) path "$.Name",
                     IndyYear int path "$.IndepYear"))
                as stuff
                where IndyYear > 1992;
+----------------+----------+
| country_name   | IndyYear |
+----------------+----------+
| Czech Republic |     1993 |
| Eritrea        |     1993 |
| Palau          |     1994 |
| Slovakia       |     1993 |
+----------------+----------+
```

JSON_TABLE Example

```
mysql> select country_name,
              IndyYear
              from countryinfo,
          json_table(doc,"$" columns
                     (country_name char(20) path "$.Name",
                      IndyYear int path "$.IndepYear"))
                as stuff
                where IndyYear > 1992;
+----------------+----------+
| country_name   | IndyYear |
+----------------+----------+
| Czech Republic |     1993 |
| Eritrea        |     1993 |
| Palau          |     1994 |
| Slovakia       |     1993 |
+----------------+----------+
```

JSON_TABLE Example

12. Sys Schema

What is in the SYS Schema

MySQL 8.0 includes the `sys` schema, a set of objects that helps DBAs and developers interpret data collected by the Performance Schema. `sys` schema objects can be used for typical tuning and diagnosis use cases. Objects in this schema include:

- Views that summarize Performance Schema data into more easily understandable form.
- Stored procedures that perform operations such as Performance Schema configuration and generating diagnostic reports.
- Stored functions that query Performance Schema configuration and provide formatting services.

Statements in Highest 5 Percent by Runtime

List all statements whose average runtime, in microseconds is in highest 5 percent

Query	Full T...	Executed (#)	Errors (#)	Warnings (#)	Total Time (	Maximum Ti...	Avg Time (us)
CREATE TABLE SYSTEM_USER (`username` VARCHA...		1	0	0	354508.33	354508.33	354508.33
CREATE TABLE `testx` (`id` INTEGER , `c1` JSON)		1	0	0	351993.09	351993.09	351993.09
CREATE TABLE `UserRecords` (`id` INTEGER (?) NO...		1	0	0	286686.84	286686.84	286686.84
CREATE TABLE `test` (`id` INTEGER , `c1` JSON)		2	1	0	425183.98	419627.39	212591.84
INSERT INTO `testx` VALUES (...) /* , ... */		1	0	0	111718.35	111718.35	111718.35

Tables with Full Table Scans

Find tables that are being accessed by full table scans ordering by the number of rows scanned descending

Schema	Object	Full Scanne...
world_x	city	4079
test	userrecords	55
test	testx	24
test	user	8

Top I/O Time by User/Thread

Show the top IO time consumers by User/thread

User	Thread Id	Process List Id	Total IOs (#)	Total Time (	Min Time (us)	Avg Time (us)	Max Time (us)
main	1	0	52116	27199168.48	0.21	4286.21	690767.80
log_flusher_thread	18	0	100	2300070.77	18.37	23000.71	131842.48
page_flush_coordinator_...	14	0	394	1031708.76	9.48	2618.55	129726.92
io_write_thread	10	0	20	620911.77	7232.35	31045.59	116960.38
io_write_thread	11	0	8	258066.16	8405.60	32258.27	93117.47
dict_stats_thread	27	0	20	221030.76	231.19	11051.54	62549.93
io_write_thread	13	0	5	166138.96	8025.01	33227.79	110270.01
worker	29	0	84	160630.76	8.66	1912.27	28306.08
log_writer_thread	17	0	362	140328.05	8.14	387.65	101744.19
root@localhost	52	13	35	115720.94	1.32	3110.60	21231.09
root@localhost	50	11	1	15043.91	15043.91	15043.91	15043.91
log_checkpointer_thread	15	0	9	11583.21	39.75	1287.02	10889.76

Statement Statistics

Shows statement execution statistics for each user

User	Statement	Total Event...	Total Time (	Max Time (us)	Lock Time (us)	Rows Sent (#)	Rows Exami...	Rows Affec...	Full Scans (#)
background	select	1	82664.71	82664.71	0.00	1	0	0	0
root	create_table	5	1418372.24	419627.39	1211446.00	0	0	0	0
root	select	117	1155600.71	269190.47	547608.00	8009	61155	0	74
root	insert	8	713038.15	131034.97	3523.00	0	0	13	0
root	show_status	241	621996.64	12009.79	60422.00	101440	206320	0	241
root	show_fields	109	173671.66	34202.20	125162.00	1013	5412	0	0
root	show_...iables	29	135444.66	34890.88	44192.00	1753	31726	0	29
root	show_tables	6	112752.87	73131.69	28860.00	411	2479	0	0
root	show_...bases	6	61244.50	36198.52	53072.00	38	212	0	6
root	set_option	44	51612.63	21146.78	0.00	0	0	0	0
root	show_...status	2	38312.63	36273.94	23829.00	0	4	0	0
root	update	2	34036.64	27277.57	11679.00	0	1225	939	0
root	show_keys	2	26894.47	25873.77	26075.00	0	20	0	0
root	Ping	184	26136.68	314.59	0.00	0	0	0	0
root	show_plugins	1	23417.73	23417.73	263.00	43	43	0	1
root	show_...gines	2	18428.13	18140.58	14618.00	18	18	0	2
root	stmt	1	14415.05	14415.05	0.00	0	1	0	0
root	set	2	14368.78	14367.87	0.00	0	1	0	0
root	Init DB	1	14247.69	14247.69	5972.00	0	0	0	0
root	freturn	152	14079.12	13412.08	0.00	0	0	0	0
root	show_grants	4	13623.91	13326.45	0.00	0	0	0	0
root	jump_if_not	777	9730.42	8495.19	0.00	0	0	0	0
root	drop_table	1	8491.28	8491.28	8171.00	0	0	0	0
root	show_...ations	2	4224.91	2215.06	740.00	540	1784	0	2
root	show_...arsets	2	3990.54	3219.54	2989.00	82	246	0	2
root	show_...status	2	1701.56	999.66	872.00	0	4	0	0
root	error	4	688.98	253.60	0.00	0	0	0	0
root	change_db	4	442.29	145.13	50.00	0	0	0	0
root	Quit	2	191.10	103.66	0.00	0	0	0	0

Stats by user

13. Set Persist

Saving Configuration Changes

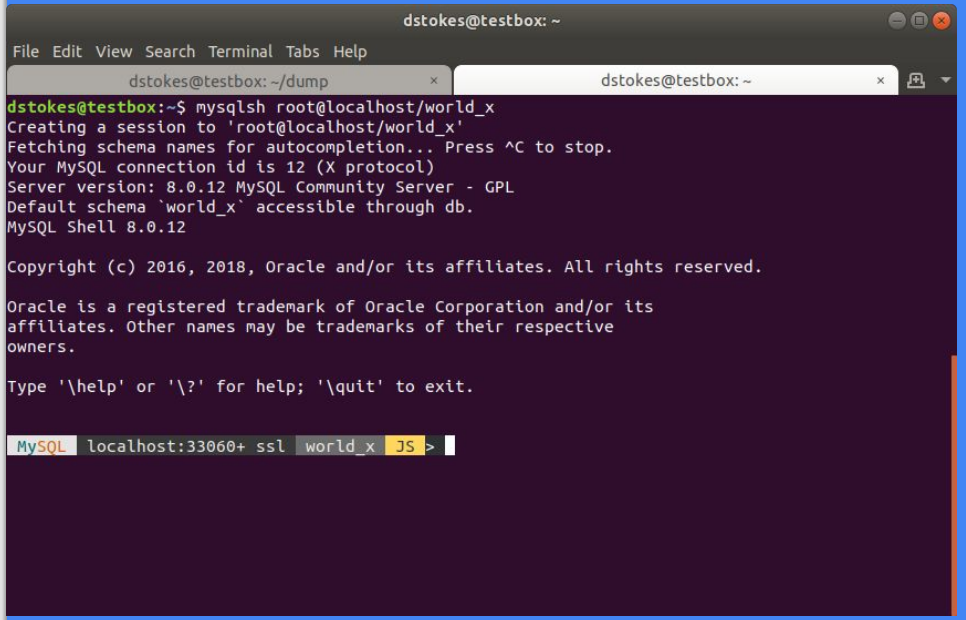
SET **PERSIST** innodb_buffer_pool_size = 512 * 1024 * 1024;

The file mysqld-auto.cnf is created the first time a SET PERSIST statement is executed. Further SET PERSIST statement executions will append the contents to this file.

14. New Shell

MySQL Shell

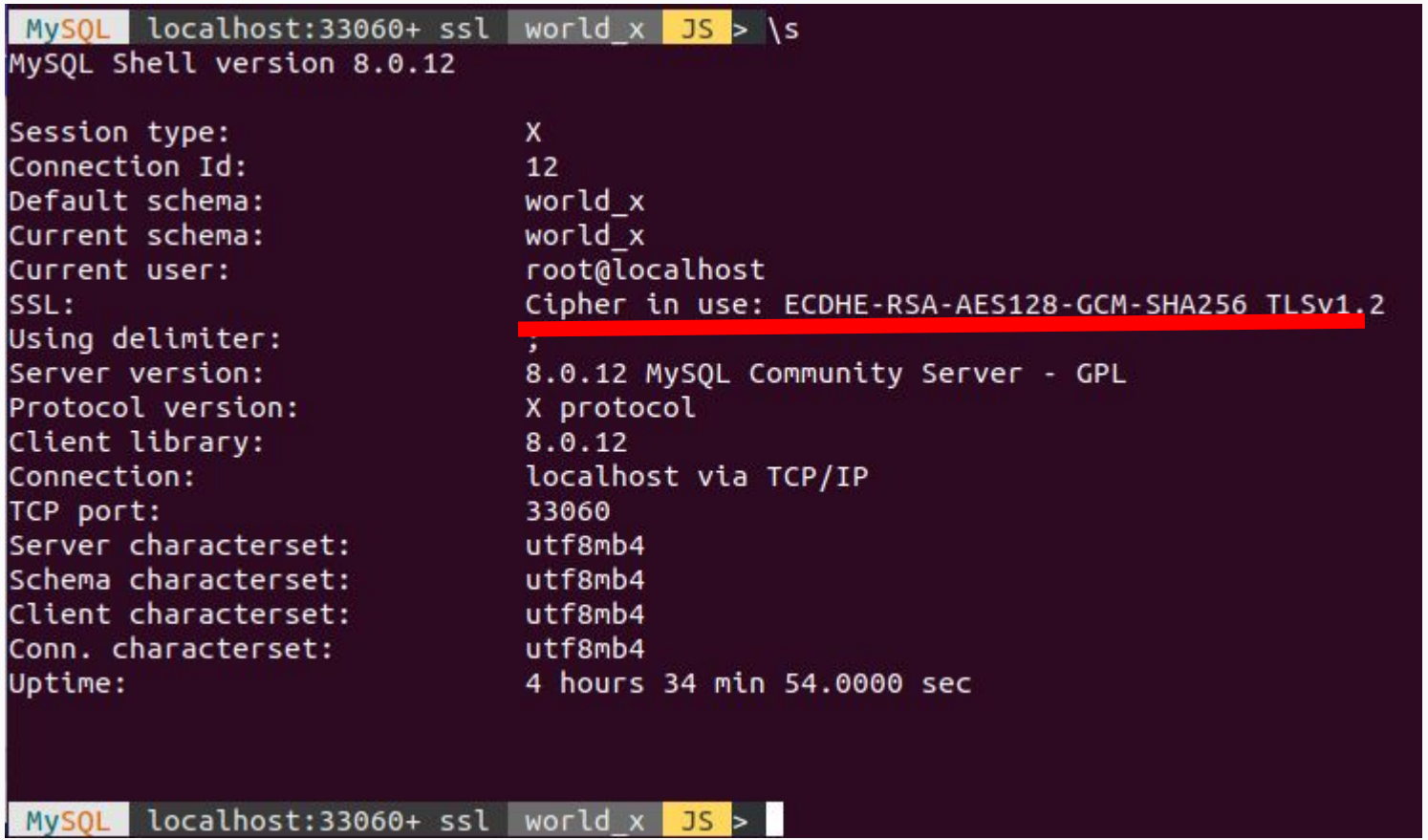
Query tool, administration tool, cluster manager, and supports Python, JavaScript & SQL



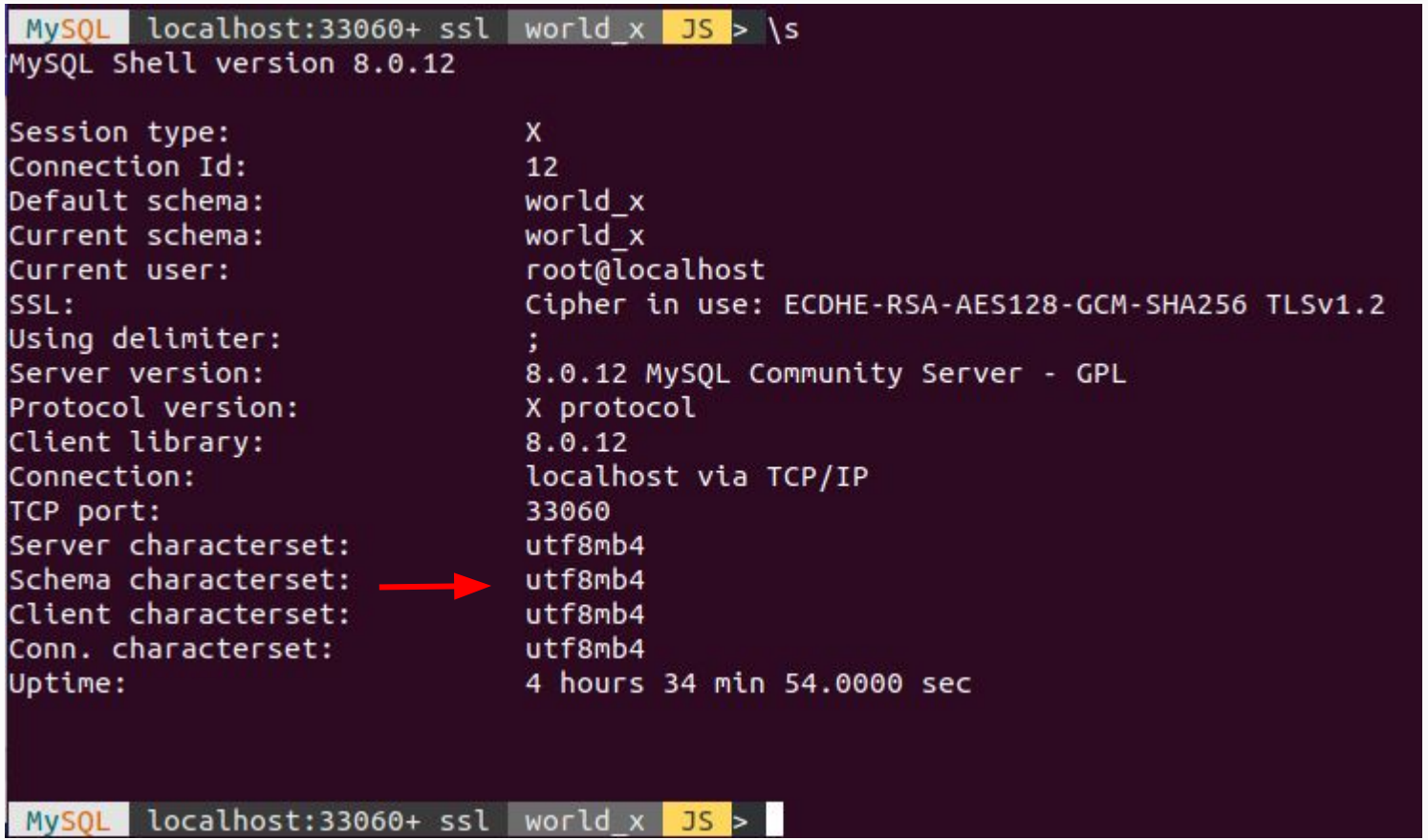
MySQL Shell

```
MySQL  localhost:33060+ ssl  world_x  JS > \s
MySQL Shell version 8.0.12

Session type:                   X
Connection Id:                  12
Default schema:                 world_x
Current schema:                 world_x
Current user:                   root@localhost
SSL:                            Cipher in use: ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2
Using delimiter:                ;
Server version:                 8.0.12 MySQL Community Server - GPL
Protocol version:               X protocol
Client library:                 8.0.12
Connection:                     localhost via TCP/IP
TCP port:                       33060
Server characterset:            utf8mb4
Schema characterset:            utf8mb4
Client characterset:            utf8mb4
Conn. characterset:             utf8mb4
Uptime:                         4 hours 34 min 54.0000 sec


MySQL  localhost:33060+ ssl  world_x  JS >
```

MySQL Shell


MySQL localhost:33060+ ssl world_x JS > \s
MySQL Shell version 8.0.12
Session type: X
Connection Id: 12
Default schema: world_x
Current schema: world_x
Current user: root@localhost
SSL: Cipher in use: ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2
Using delimiter: ;
Server version: 8.0.12 MySQL Community Server - GPL
Protocol version: X protocol
Client library: 8.0.12
Connection: localhost via TCP/IP
TCP port: 33060
Server characterset: utf8mb4
Schema characterset: utf8mb4
Client characterset: utf8mb4
Conn. characterset: utf8mb4
Uptime: 4 hours 34 min 54.0000 sec
MySQL localhost:33060+ ssl world_x JS >

MySQL Shell


MySQL localhost:33060+ ssl world_x JS > \s
MySQL Shell version 8.0.12
Session type: X
Connection Id: 12
Default schema: world_x
Current schema: world_x
Current user: root@localhost
SSL: Cipher in use: ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2
Using delimiter: ;
Server version: 8.0.12 MySQL Community Server - GPL
Protocol version: X protocol
Client library: 8.0.12
Connection: localhost via TCP/IP
TCP port: 33060
Server characterset: utf8mb4
Schema characterset: utf8mb4
Client characterset: utf8mb4
Conn. characterset: utf8mb4
Uptime: 4 hours 34 min 54.0000 sec
MySQL localhost:33060+ ssl world_x JS >

MySQL Shell

Python, JavaScript & SQL modes

Management

- **util.checkForServerUpgrade('user@host.com:3306')**
- **dba. configureLocalInstance**
- **dba.createCluster**

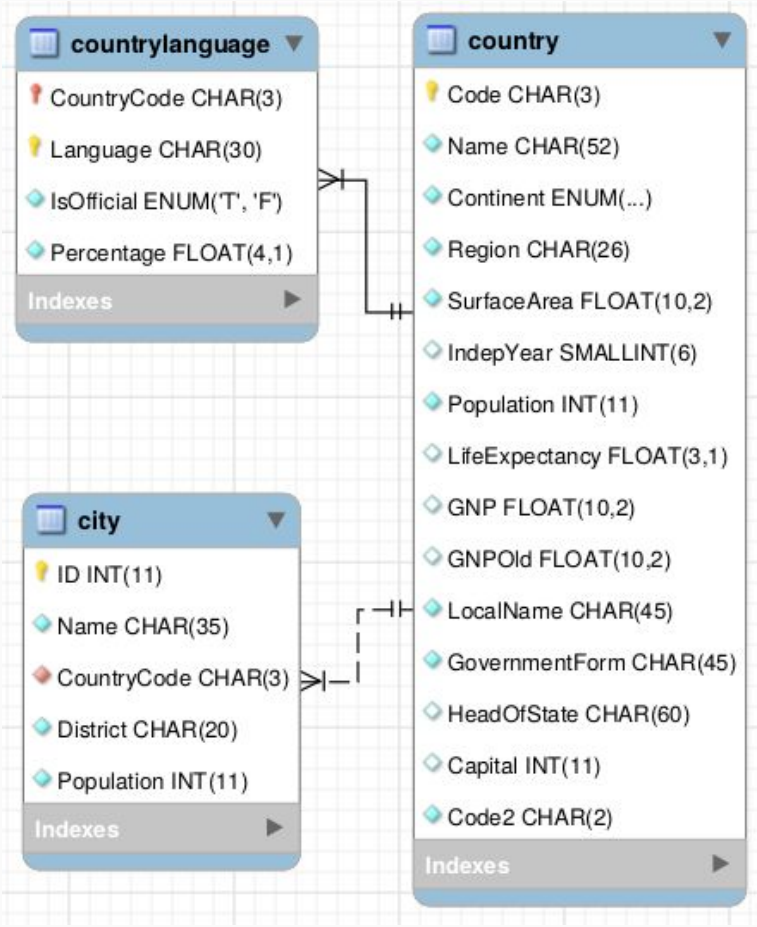
```
Mysqlx.Crud.Find {
  collection { name: "collection_name", schema: "test" }
  data_model: DOCUMENT
  criteria {
    type: OPERATOR
    operator {
      name: "=="
      param {
        type: IDENT,
        identifier { name: "_id" }
      }
      param {
        type: LITERAL,
        literal {
          type: V_STRING,
          v_string: { value: "some_string" }
        }
      }
    }
  }
}
```

New Protocol based on Google ProtoBuf

15. MySQL Document Store

NoSQL or Document Store

- Schemaless
  - No schema design, no normalization, no foreign keys, no data types, …
  - Very quick initial development
- Flexible data structure
  - Embedded arrays or objects
  - Valid solution when natural data can not be modelized optimally into a relational model
  - Objects persistence without the use of any ORM - *mapping object-oriented*
- JSON
- close to frontend
- native in JS
- easy to learn




```
{
  "GNP" : 249704,
  "Name" : "Belgium",
  "government" : {
     "GovernmentForm" :
        "Constitutional Monarchy, Federation",
     "HeadOfState" : "Philippe I"
  },
  "_id" : "BEL",
  "IndepYear" : 1830,
  "demographics" : {
     "Population" : 10239000,
     "LifeExpectancy" : 77.8000030517578
  },
  "geography" : {
     "Region" : "Western Europe",
     "SurfaceArea" : 30518,
     "Continent" : "Europe"
  }
}
```

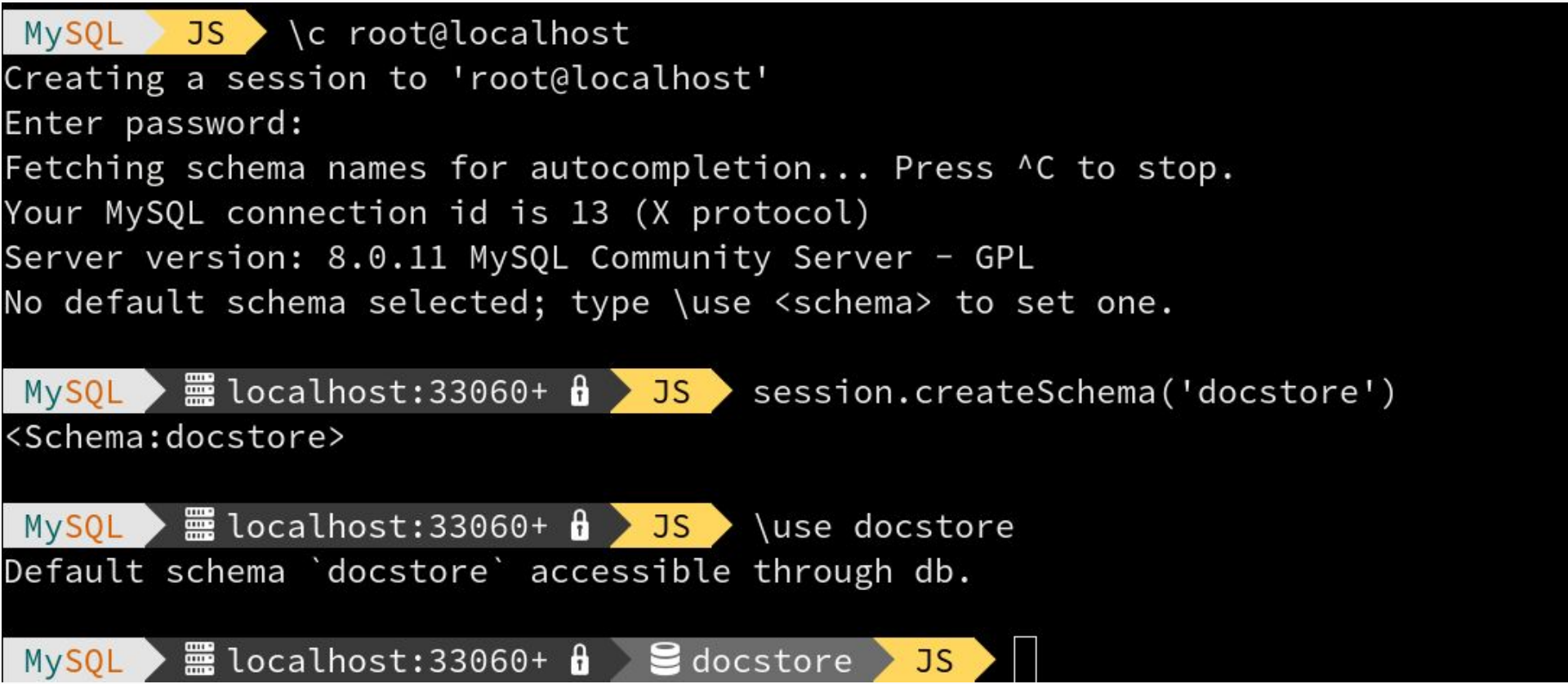

How DBAs see data as opposed to how Developers see data

What if there was a way to provide both SQL and NoSQL on one stable platform that has proven stability on well know technology with a large Community and a diverse ecosystem ?

With the MySQL Document Store, SQL is now optional!

- ★ **Provides a schema flexible JSON Document Store**
- ★ ***No*** **SQL required**
- ★ **No need to define all possible attributes, tables, etc.**
- ★ **Uses new X DevAPI**
- ★ **Can leverage generated column to extract JSON values into materialized columns that can be indexed for fast SQL searches.**
- ★ **Document can be ~1GB**
  - **It's a column in a row of a table**
- ★ **Allows use of modern programming styles**
  - **No more embedded strings of SQL in your code**
  - **Easy to read**
- ★ ***Also*** **works with relational Tables**
- ★ **Proven MySQL Technology**

- ★ **Connectors for**
  - **C++, Java, .Net, Node.js, Python, PHP**
  - **working with Communities to help them supporting it too**
- ★ **New MySQL Shell**
  - **Command Completion**
  - **Python, JavaScripts & SQL modes**
  - **Admin functions**
  - **New Util object**
  - **A new high-level session concept that can scale from single MySQL Server to a multiple server environment**
- ★ **Non-blocking, asynchronous calls follow common language patterns**
- ★ **Supports CRUD operations**



Starting using MySQL in few seconds

Migration from MongoDB to MySQL Document Store

For this example, I will use the well known restaurants collection:

We need to dump the data to a file and

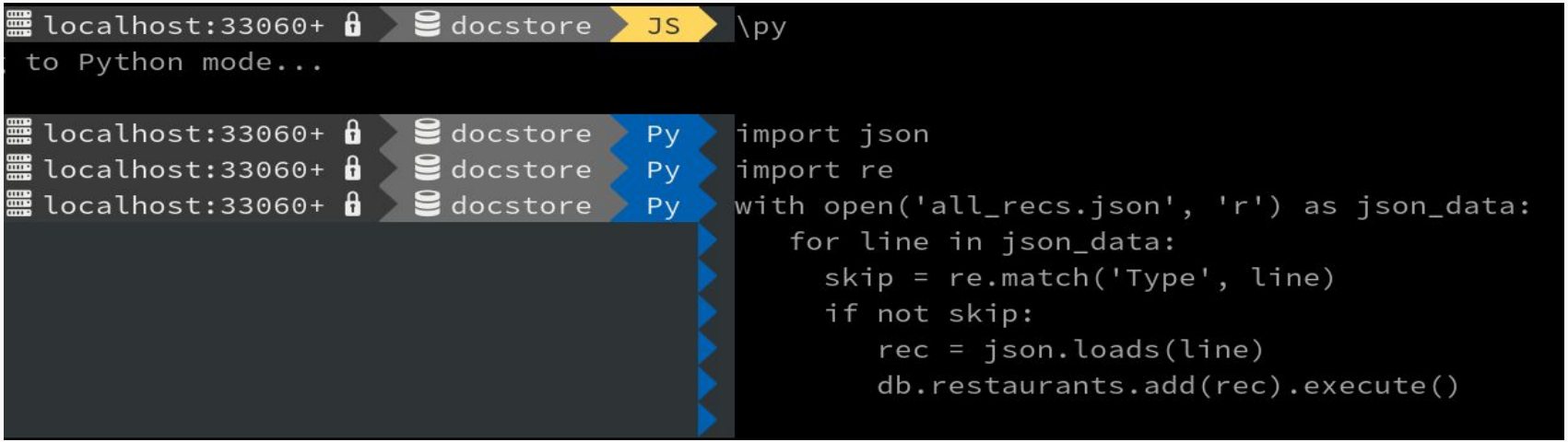
we will use the MySQL Shell

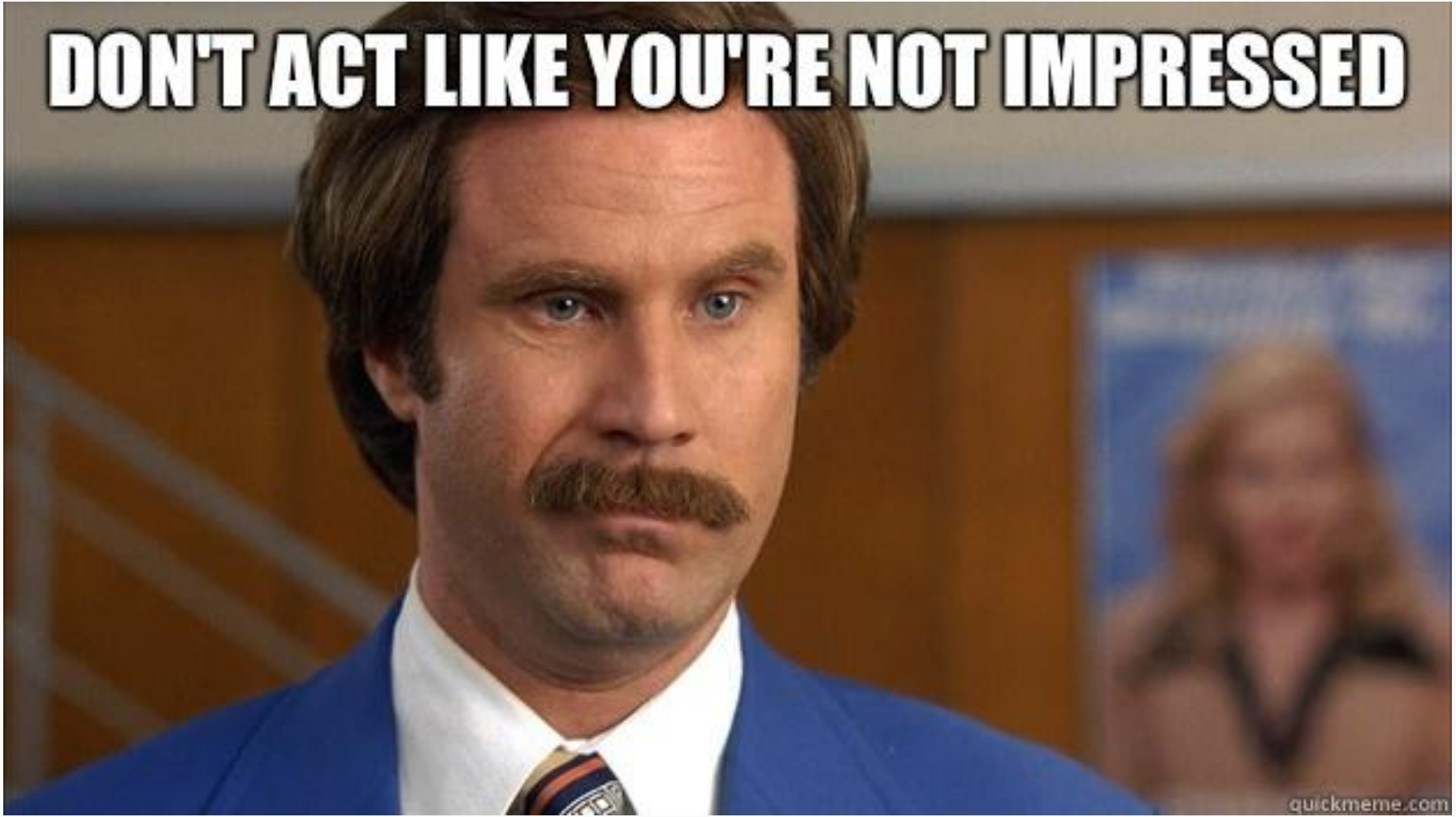
with the Python interpreter to load the data.

Dump and load using MySQL Shell & Python

This example is inspired by @datacharmer's work: https://www.slideshare.net/datacharmer/mysql-documentstore

```
$ mongo quiet eval 'DBQuery.shellBatchSize=30000;
                    db.restaurants.find().shellPrint()' \
 | perl -pe 's/(?:ObjectId|ISODate)\(("[^"]+")\)/ $1/g' > all_recs.json
```




DON'T ACT LIKE YOU'RE NOT IMPRESSED
quickmeme.com

Let’s query

```
MySQL 127.0.0.1:33060 JS restaurants.find()
```

Too many records to show here … let’s limit it!

```
MySQL 127.0.0.1:33060 JS restaurants.find().limit(1)
[
    {
        "_id": "5943c83d1adc26055941640c",
        "address": {
            "building": "351",
            "coord": [
                -73.9851,
                40.7677
            ],
            "street": "West   57 Street",
            "zipcode": "10019"
        },
        "borough": "Manhattan",
        "cuisine": "Irish",
        "grades": [
            {
                "date": "2014-09-06T00:00:00Z",
                "grade": "A",
                "score": 2
            },
            {
                "date": "2013-07-22T00:00:00Z",
                "grade": "A",
                "score": 11
            },
            {
                "date": "2012-07-31T00:00:00Z",
                "grade": "A",
                "score": 12
            },
            {
                "date": "2011-12-29T00:00:00Z",
                "grade": "A",
                "score": 12
            }
        ],
        "name": "Dj Reynolds Pub And Restaurant",
        "restaurant_id": "30191841"
    }
]
1 document in set (0.08 sec)
```

```
MySQL 127.0.0.1:33060 JS restaurants.find().fields(["name","cuisine"]).limit(2)
[
    {
        "cuisine": "Irish",
        "name": "Dj Reynolds Pub And Restaurant"
    },
    {
        "cuisine": "American",
        "name": "Riviera Caterer"
    }
]
2 documents in set (0.00 sec)
```

More Examples!

```
MySQL 127.0.0.1:33060 JS restaurants.find("cuisine='Italian'").fields(["name","cuisine"]).limit(2)
[
    {
        "cuisine": "Italian",
        "name": "Philadelhia Grille Express"
    },
    {
        "cuisine": "Italian",
        "name": "Isle Of Capri Resturant"
    }
]
2 documents in set (0.00 sec)
```

Let’s add a selection criteria

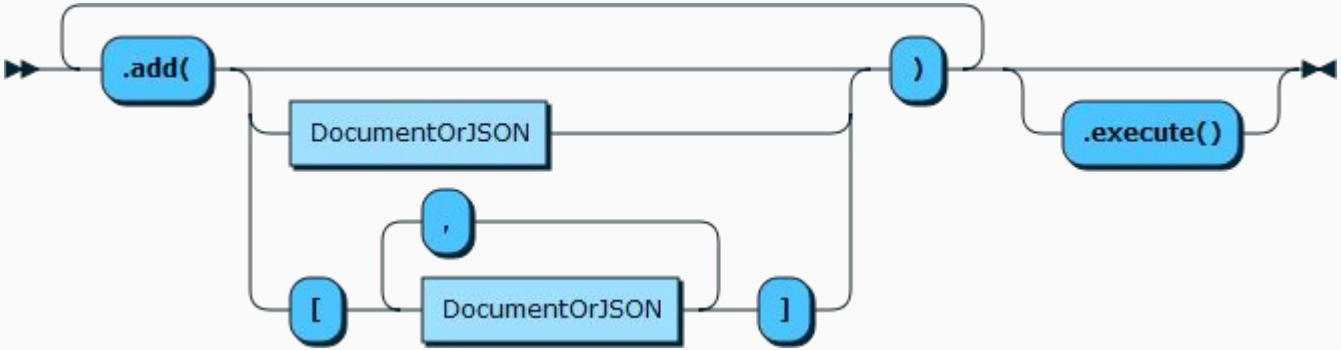
```
MySQL 127.0.0.1:33060 JS restaurants.find("cuisine='French' AND borough!='Manhattan'").
fields(["name","cuisine","borough"]).limit(2)
[
    {
        "borough": "Queens",
        "cuisine": "French",
        "name": "La Baraka Restaurant"
    },
    {
        "borough": "Queens",
        "cuisine": "French",
        "name": "Air France Lounge"
    }
]
2 documents in set (0.00 sec)
```

```
> db.restaurants.find({"cuisine": "French",
    "borough": { $not: /^Manhattan/} },
    {"_id":0, "name": 1,"cuisine": 1, "borough": 1}).limit(2)
{ "borough" : "Queens", "cuisine" : "French",
  "name" : "La Baraka Restaurant" }
{ "borough" : "Queens", "cuisine" : "French",
   "name" : "Air France Lounge" }
```

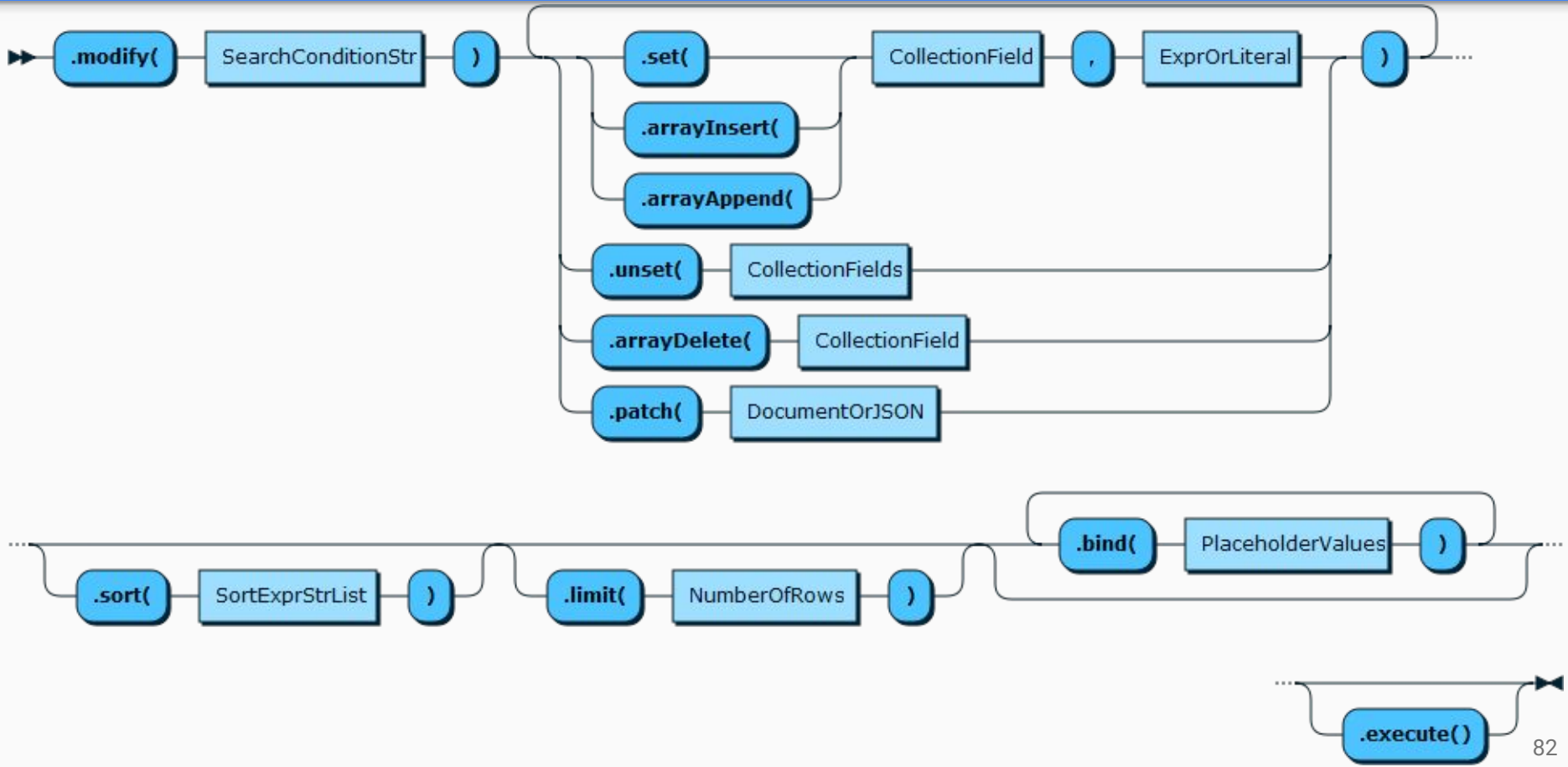
Syntax is slightly different than MongoDB

```
MySQL 127.0.0.1:33060 JS restaurants.remove("cuisine='French' AND borough!='Manhattan'"
).limit(2)
Query OK, 2 items affected (0.16 sec)

MySQL 127.0.0.1:33060 JS restaurants.find("cuisine='French' AND borough!='Manhattan'").
fields(["name","cuisine","_id"]).limit(2)
[
    {
        "_id": "5943c83e1adc2605594170aa",
        "cuisine": "French",
        "name": "Bar Tabac"
    },
    {
        "_id": "5943c83e1adc260559417255",
        "cuisine": "French",
        "name": "Tournesol"
    }
]
2 documents in set (0.01 sec)
```

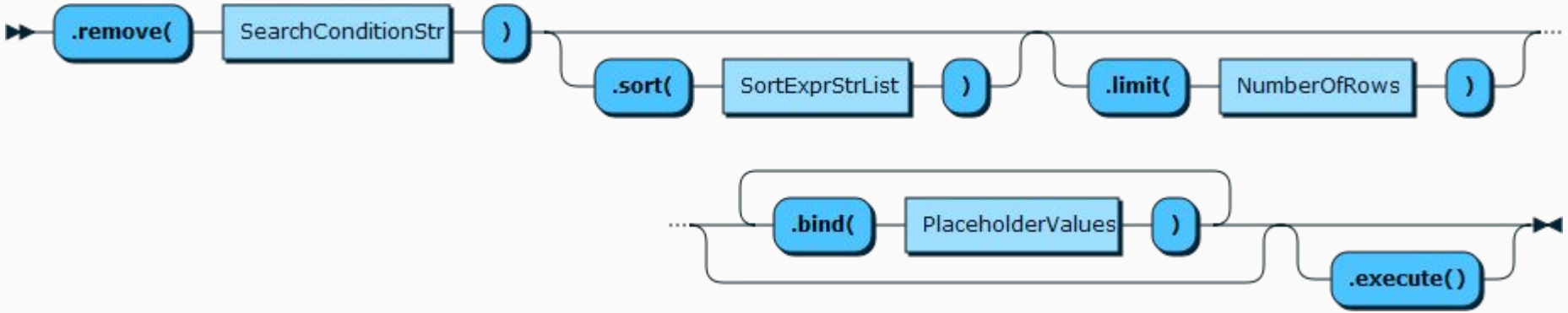
Add a Document



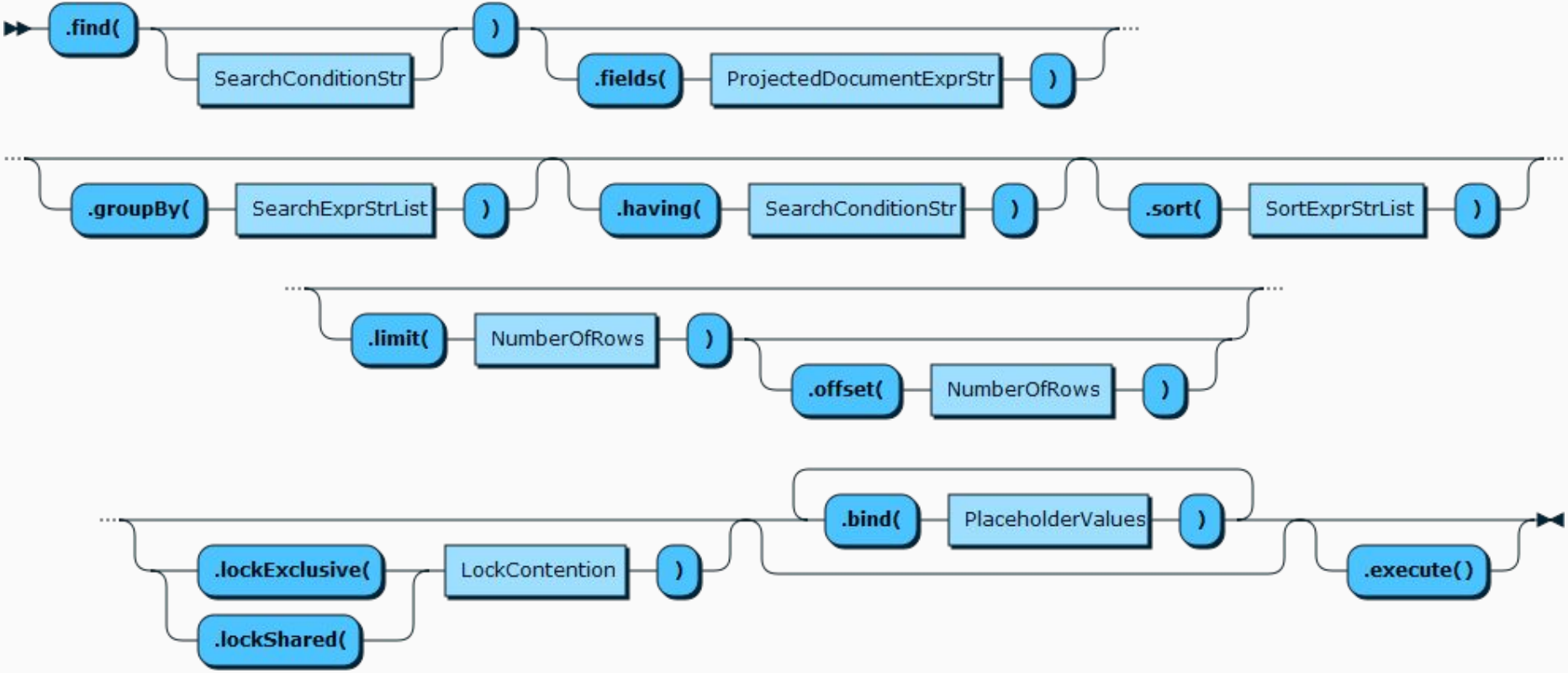
Modify a Document



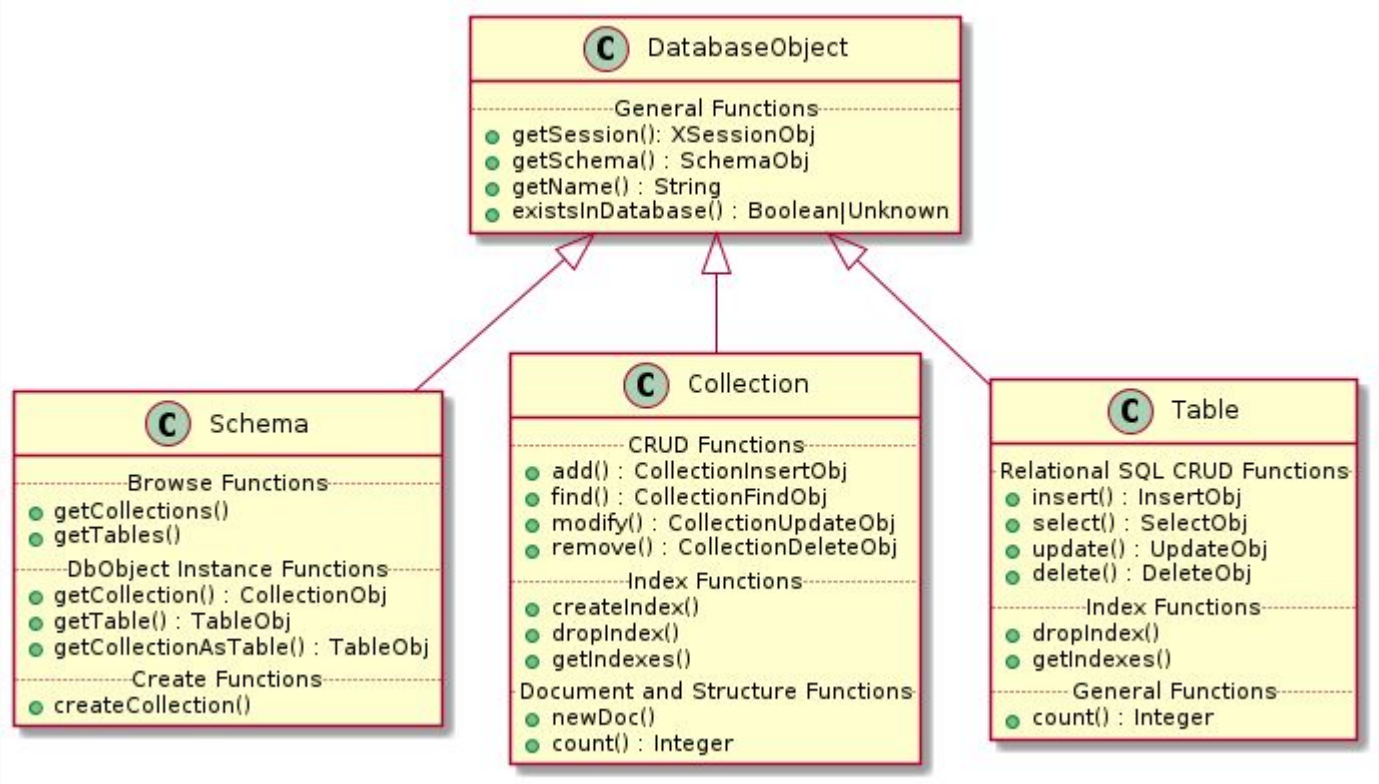
Remove a Document



Find a Document



MySQL Document Store Objects Summary



```
MySQL localhost:33060+ fred JS session.startTransaction()
Query OK, 0 rows affected (0.0006 sec)

MySQL localhost:33060+ fred JS test.add({name: 'the René'})
Query OK, 1 item affected (0.1661 sec)

MySQL localhost:33060+ fred JS test.find()
[
    {
        "_id": "00005ade551100000000000001",
        "name": "fred"
    },
    {
        "_id": "00005ade551100000000000002",
        "name": "the René"
    }
]
2 documents in set (0.0019 sec)
```

MySQL Document Store is Fully ACID Compliant

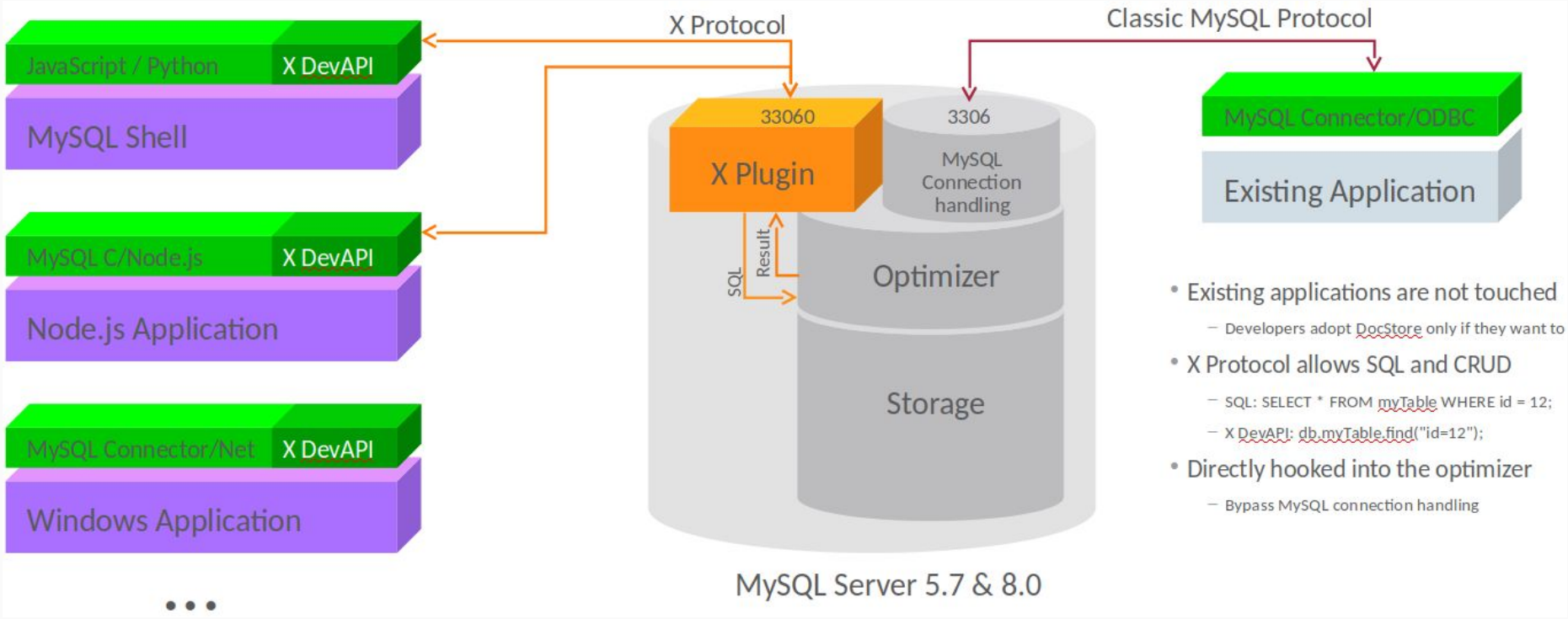
```
MySQL localhost:33060+ fred JS session.rollback()
Query OK, 0 rows affected (0.0992 sec)

MySQL localhost:33060+ fred JS test.find()
[
    {
        "_id": "00005ade551100000000000000001",
        "name": "fred"
    }
]
1 document in set (0.0130 sec)
```

MySQL Document Store is Fully ACID Compliant

What about old SQL? The Hidden Part of the Iceberg

JSON datatype is behind the scene

- ★ Native datatype (since 5.7.8)
- ★ JSON values are stored in MySQL tables using UTF8MB4
- ★ Conversion from "native" SQL types to JSON values
- ★ JSON manipulation functions (JSON_EXTRACT, JSON_KEYS, JSON_SEARCH, JSON_TABLES, ...)
- ★ Generated/virtual columns
  - ○ Indexing JSON data
  - ○ Foreign Keys to JSON data
  - ○ SQL Views to JSON data



- Existing applications are not touched
  - Developers adopt DocStore only if they want to
- X Protocol allows SQL and CRUD
  - SQL: SELECT * FROM myTable WHERE id = 12;
  - X DevAPI: db.myTable.find("id=12");
- Directly hooked into the optimizer
  - Bypass MySQL connection handling

How Does It Work??

```
MySQL localhost:33060+ docstore SQL DESC restaurants;
+-------+---------------+------+-----+---------+-----------------+
| Field | Type          | Null | Key | Default | Extra           |
+-------+---------------+------+-----+---------+-----------------+
| doc   | json          | YES  |     | NULL    |                 |
| _id   | varbinary(32) | NO   | PRI | NULL    | STORED GENERATED |
+-------+---------------+------+-----+---------+-----------------+
2 rows in set (0.0668 sec)

MySQL localhost:33060+ docstore SQL show create table restaurants\G
*************************** 1. row ***************************
       Table: restaurants
Create Table: CREATE TABLE `restaurants` (
  `doc` json DEFAULT NULL,
  `_id` varbinary(32) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,_utf8mb4'$._id'
))) STORED NOT NULL,
  PRIMARY KEY (`_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

What does a collection look like on the server ?

_id

Every document has a unique identifier called the *document ID*, which can be thought of as the equivalent
of a table´s primary key. The document ID value can be manually assigned when adding a document.

If novalue is assigned, a document ID is generated and assigned to the document automatically !

Use *getDocumentId()* or *getDocumentIds()* to get _ids(s)

Mapping to SQL Examples

```
createCollection('mycollection')

CREATE TABLE `test`.`mycoll` (
        doc JSON,
        _id VARCHAR(32)
                GENERATED ALWAYS AS (doc->"$._id") STORED
                PRIMARY KEY
) CHARSET utf8mb4;

INSERT INTO `test`.`mycoll` (doc) VALUES (
        CONVERT('{"_id":"6380fe367e611ec845bdac28bf",
'test',1234));
```

More Mapping to SQL Examples

```
mycollection.find("test > 100")
```

```
SELECT doc
    FROM `test`.`mycoll`
    WHERE (JSON_EXTRACT(doc,'$.test') >100);
```

```
MySQL  localhost:33060+  docstore  SQL  ALTER TABLE restaurants ADD COLUMN borough
VARCHAR( 20) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`, '$.borough'))) VIRTUAL;
```

```
MySQL  localhost:33060+  docstore  SQL  SELECT _id, borough FROM restaurants LIMIT 5;

+--------------------------+-----------+
| _id                      | borough   |
+--------------------------+-----------+
| 59ca58986b977d7c822812b7 | Bronx     |
| 59ca58986b977d7c822812b8 | Brooklyn  |
| 59ca58986b977d7c822812b9 | Manhattan |
| 59ca58986b977d7c822812ba | Brooklyn  |
| 59ca58986b977d7c822812bb | Queens    |
+--------------------------+-----------+
5 rows in set (0.00 sec)
```

SQL and JSON Example

```
MySQL  localhost:33060+  docstore  JS  db.restaurants.createIndex('cuisine_idx',
 {fields:{field: "$.cuisine",required: false, type: "text(20)"}})
```

```
MySQL  localhost:33060+  docstore  SQL  show create table restaurants\G
*************************** 1. row ***************************
       Table: restaurants
Create Table: CREATE TABLE `restaurants` (
  `doc` json DEFAULT NULL,
  `_id` varbinary(32) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,_utf8mb4'$._id')
)) STORED NOT NULL,
  `$ix_t20_BC26D4DF1273E3F7412529AEE9E95A0CC8475CEB` text GENERATED ALWAYS AS (json_unquote
(json_extract(`doc`,_utf8mb4'$.cuisine'))) VIRTUAL,
  PRIMARY KEY (`_id`),
  KEY `cuisine_idx` (`$ix_t20_BC26D4DF1273E3F7412529AEE9E95A0CC8475CEB`(20))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

It's also possible to create indexes without using SQL syntax

```
MySQL  localhost:33060+  docstore  SQL  ALTER TABLE restaurants ADD COLUMN cuisine
VARCHAR(20) GENERATED ALWAYS AS (doc->>"$.cuisine") VIRTUAL, WITH VALIDATION;
ERROR: 1406: Data too long for column 'cuisine' at row 10
```

```
MySQL  localhost:33060+  docstore  SQL  ALTER TABLE restaurants ADD COLUMN cuisine
VARCHAR(20) GENERATED ALWAYS AS (LEFT(doc->>"$.cuisine",20)) VIRTUAL, WITH VALIDATION;
Query OK, 25359 rows affected (0.00 sec)
```

```
MySQL  localhost:33060+  docstore  SQL  SELECT _id, borough, cuisine FROM restaurants
 LIMIT 5;
+--------------------------+-----------+---------------+
| _id                      | borough   | cuisine       |
+--------------------------+-----------+---------------+
| 59ca58986b977d7c822812b7 | Bronx     | Bakery        |
| 59ca58986b977d7c822812b8 | Brooklyn  | Hamburgers    |
| 59ca58986b977d7c822812b9 | Manhattan | Irish         |
| 59ca58986b977d7c822812ba | Brooklyn  | American      |
| 59ca58986b977d7c822812bb | Queens    | Jewish/Kosher |
+--------------------------+-----------+---------------+
```

SQL and JSON Example (2): validation

```
MySQL  localhost:33060+  docstore  SQL  EXPLAIN SELECT doc->>"$.name" AS name, cuisine,
 borough FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: restaurants
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 24890
     filtered: 1.0000001192092896
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
Note (code 1003): /* select#1 */ select json_unquote(json_extract(`docstore`.`restaurants`.`
doc`,'$.name')) AS `name`,`docstore`.`restaurants`.`cuisine` AS `cuisine`,`docstore`.`restau
rants`.`borough` AS `borough` from `docstore`.`restaurants` where ((`docstore`.`restaurants`
.`borough` = 'Brooklyn') and (`docstore`.`restaurants`.`cuisine` = 'Italian')) limit 2
```

SQL and JSON Example (3): explain

```
MySQL  localhost:33060+  docstore  SQL  EXPLAIN SELECT doc->>"$.name" AS name, cuisine,
 borough FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: restaurants
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 24890
     filtered: 1.0000001192092896
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
Note (code 1003): /* select#1 */ select json_unquote(json_extract(`docstore`.`restaurants`.`
doc`,'$.name')) AS `name`,`docstore`.`restaurants`.`cuisine` AS `cuisine`,`docstore`.`restau
rants`.`borough` AS `borough` from `docstore`.`restaurants` where ((`docstore`.`restaurants`
.`borough` = 'Brooklyn') and (`docstore`.`restaurants`.`cuisine` = 'Italian')) limit 2
```

SQL and JSON Example (3): explain

```
MySQL > localhost:33060+ > docstore > SQL > ALTER TABLE restaurants ADD INDEX cuisine_borought_idx(cuisine,borough);
Query OK, 0 rows affected (0.00 sec)
MySQL > localhost:33060+ > docstore > SQL > EXPLAIN SELECT doc->>"$.name" AS name, cuisine, borough FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: restaurants
   partitions: NULL
         type: ref
possible_keys: cuisine_borought_idx
          key: cuisine_borought_idx
      key_len: 166
          ref: const,const
         rows: 192
     filtered: 100
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
Note (code 1003): /* select#1 */ select json_unquote(json_extract(`docstore`.`restaurants`.`doc`,'$.name')) AS `name`,`docstore`.`restaurants`.`cuisine` AS `cuisine`,`docstore`.`restaurants`.`borough` AS `borough` from `docstore`.`restaurants` where ((`docstore`.`restaurants`.`borough` = 'Brooklyn') and (`docstore`.`restaurants`.`cuisine` = 'Italian')) limit 2
```

SQL and JSON Example (4): add index

```
MySQL  localhost:33060+  docstore  SQL  ALTER TABLE restaurants ADD INDEX cuisine_borought_idx(c
uisine,borough);
Query OK, 0 rows affected (0.00 sec)
MySQL  localhost:33060+  docstore  SQL  EXPLAIN SELECT doc->>"$.name" AS name, cuisine, borough
FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: restaurants
   partitions: NULL
         type: ref
possible_keys: cuisine_borought_idx
          key: cuisine_borought_idx
      key_len: 166
          ref: const,const
         rows: 192
     filtered: 100
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
Note (code 1003): /* select#1 */ select json_unquote(json_extract(`docstore`.`restaurants`.`doc`,'$.n
ame')) AS `name`,`docstore`.`restaurants`.`cuisine` AS `cuisine`,`docstore`.`restaurants`.`borough` A
S `borough` from `docstore`.`restaurants` where ((`docstore`.`restaurants`.`borough` = 'Brooklyn') an
d (`docstore`.`restaurants`.`cuisine` = 'Italian')) limit 2
```

SQL and JSON Example (4): add index

```
MySQL  localhost:33060+  docstore  SQL  select doc->>'$.grades' from restaurants limit 1\G
*************************** 1. row ***************************
doc->>'$.grades': [{"date": "2014-03-03T00:00:00Z", "grade": "A", "score": 2}, {"date": "2013-09-11T0
0:00:00Z", "grade": "A", "score": 6}, {"date": "2013-01-24T00:00:00Z", "grade": "A", "score": 10}, {"
date": "2011-11-23T00:00:00Z", "grade": "A", "score": 9}, {"date": "2011-03-10T00:00:00Z", "grade": "
B", "score": 14}]
```

```
MySQL  localhost:33060+  docstore  SQL  select doc->>'$.grades[0]' from restaurants limit 1\G
*************************** 1. row ***************************
doc->>'$.grades[0]': {"date": "2014-03-03T00:00:00Z", "grade": "A", "score": 2}
```

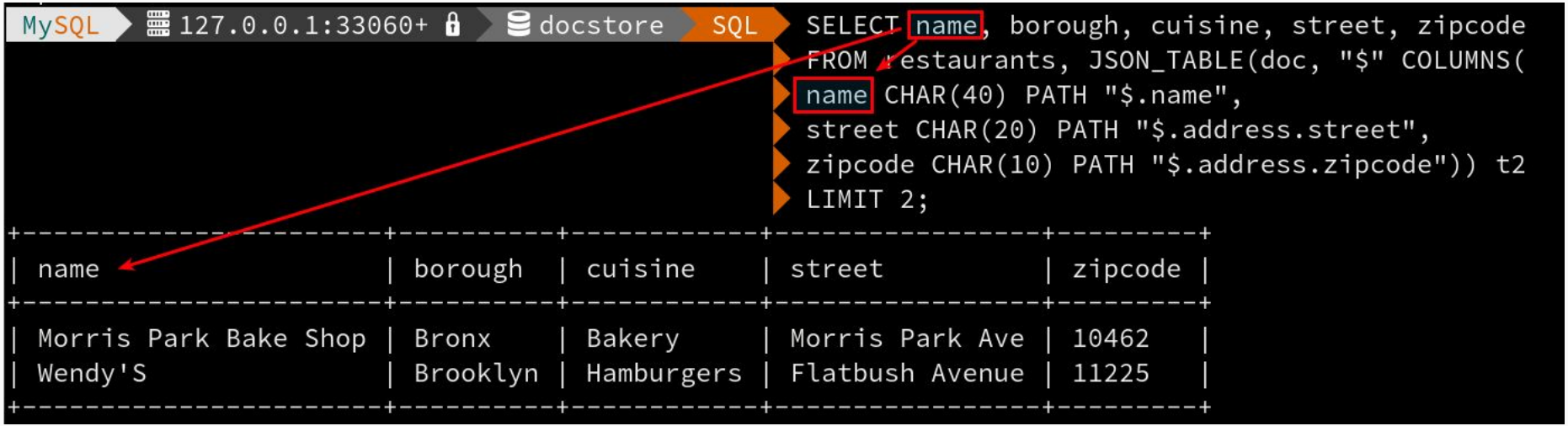
```
MySQL  localhost:33060+  docstore  SQL  select doc->>'$.grades[last]' from restaurants limit 1\G

*************************** 1. row ***************************
doc->>'$.grades[last]': {"date": "2011-03-10T00:00:00Z", "grade": "B", "score": 14}
```

```
MySQL  localhost:33060+  docstore  SQL  select doc->>'$.grades[1 to 2]' from restaurants limit 1
\G
*************************** 1. row ***************************
doc->>'$.grades[1 to 2]': [{"date": "2013-09-11T00:00:00Z", "grade": "A", "score": 6}, {"date": "2013
-01-24T00:00:00Z", "grade": "A", "score": 10}]
```

SQL and JSON Example (5): arrays

```
MySQL 127.0.0.1:33060+ docstore SQL SELECT _id, borough, cuisine, name
                                    FROM restaurants, JSON_TABLE(doc, "$"
                                    COLUMNS(name CHAR(40) PATH "$.name"))
                                    t2 LIMIT 2;
+--------------------------+----------+-----------+----------------------+
| _id                      | borough  | cuisine   | name                 |
+--------------------------+----------+-----------+----------------------+
| 59c3a6946273b7d975cf3b39 | Bronx    | Bakery    | Morris Park Bake Shop |
| 59c3a6946273b7d975cf3b3a | Brooklyn | Hamburgers | Wendy'S              |
+--------------------------+----------+-----------+----------------------+
```

JSON_TABLE turns your un-structured JSON data into a temporary structured table!



This temporary structured table can be treated like any other table -- LIMIT, WHERE, GROUP BY ...

More Sophisticated Analysis

Dig deeper into your data for results

```
WITH cte1 AS (SELECT doc->>"$.name" AS 'name',
 doc->>"$.cuisine" AS 'cuisine',
        (SELECT AVG(score) FROM
        JSON_TABLE(doc, "$.grades[*]"
        COLUMNS (score INT PATH "$.score")) as r ) AS avg_score
 FROM restaurants)
 SELECT *, rank() OVER
  (PARTITION BY cuisine ORDER BY avg_score) AS `rank`
  FROM cte1
  ORDER by `rank`, avg_score DESC limit 10;
```

This query uses a **Common Table Expression (CTE)** and a **Windowing Function** to rank the average scores of each restaurant, by each cuisine with unstructured JSON data

Find the top 10 restaurants by grade for each cuisine

This is the best of the two worlds in one product !

- **Data integrity**
- **ACID Compliant**
- **Transactions**
- **SQL**

- **Schemaless**
- **flexible data structure**
- **easy to start (CRUD)**

16. Locking Changes

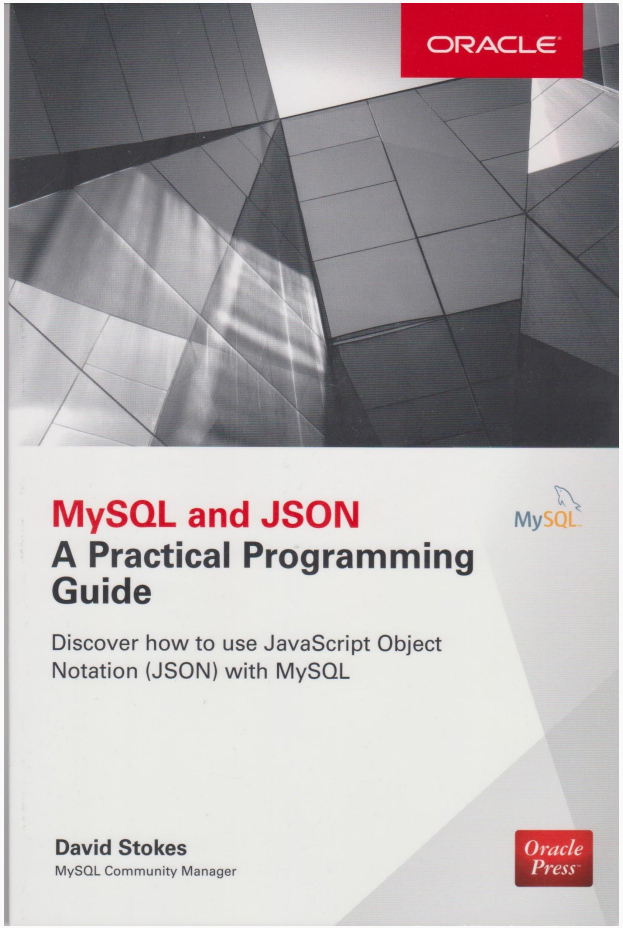
SKIP LOCKED and NOWAIT

```
START TRANSACTION;
SELECT * FROM seats WHERE seat_no BETWEEN 2 AND 3
AND booked = 'NO'
FOR UPDATE SKIP LOCKED;
----------
SELECT seat no
FROM seats JOIN seat rows USING ( row no )
WHERE seat no IN (3,4) AND seat_rows.row_no IN (12)
AND booked = 'NO'
FOR UPDATE OF seats SKIP LOCKED
FOR SHARE OF seat_rows NOWAIT;
```

Conclusion

Big Changes

1. Constant Integration
2. Smarter about environment
3. More powerful SQL
4. Data Dictionary
5. NoSQL and SQL -- Best of both worlds
6. Better Command and Control



Please Buy My Book!!!

Thanks!

Contact info:

Dave Stokes

David.Stokes@Oracle.com

@Stoker

slideshare.net/davidmstokes

speakerdeck.com/davidmstokes

Elepantdolphin.blogger.com

opensourcedba.Wordpress.com