Practical JSON in MySQL 5.7

ABOUT ME

Who am I?

1. Database Architect at **Flite** since 2007
2. **@iowalker** on twitter
3. Organizer of Boston MySQL Meetup
4. Blog at **mechanics.flite.com**

FLITE

WHAT THIS TALK IS ABOUT

- Using JSON in MySQL 5.7

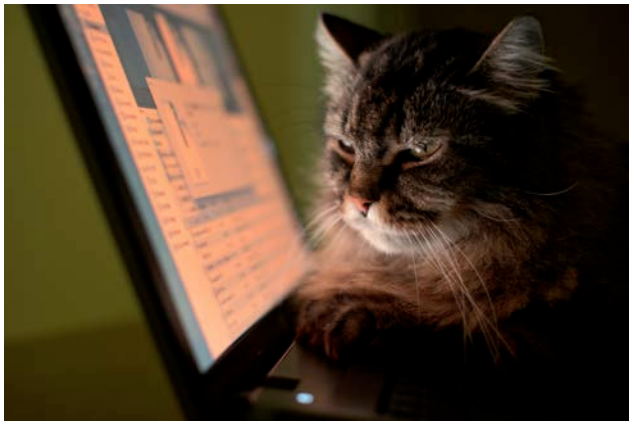
FLITE

FLITE

WHAT THIS TALK IS NOT ABOUT

- Whether you ***should*** use JSON in MySQL

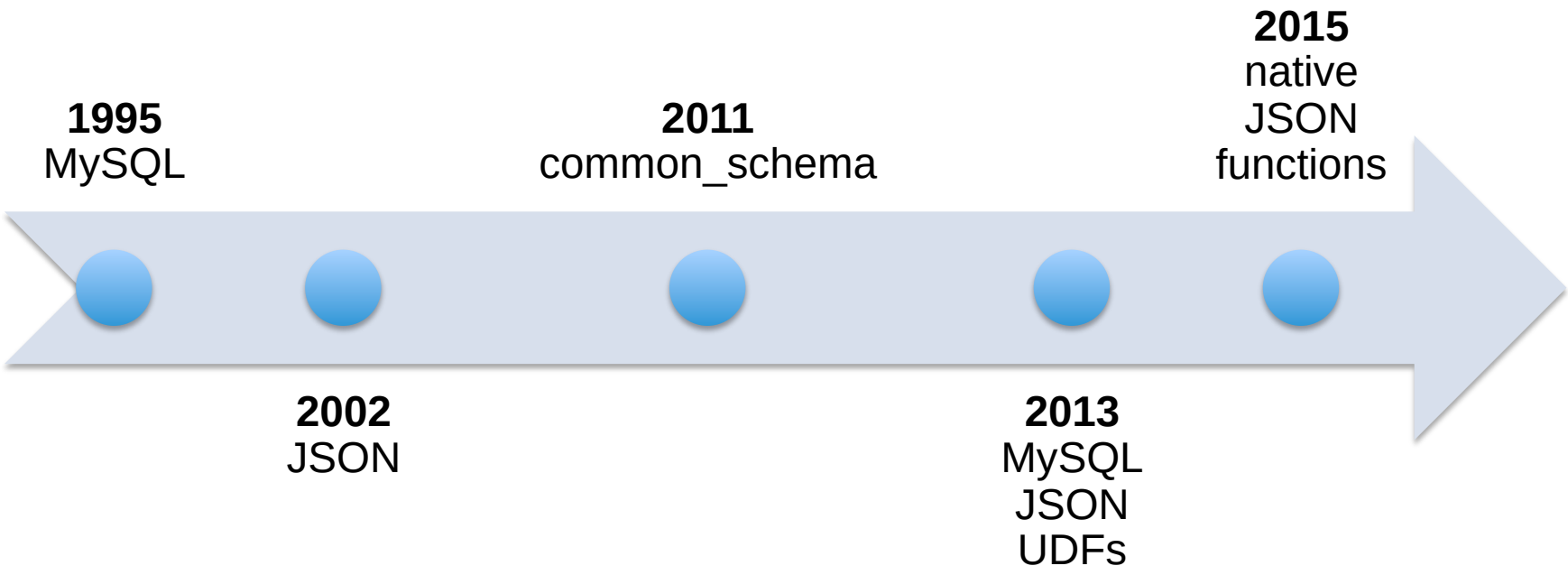
YES!



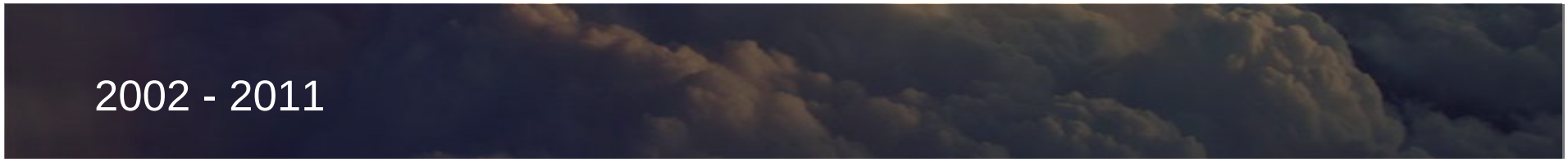
NO!

FLITE

Timeline: MySQL and JSON



FLITE

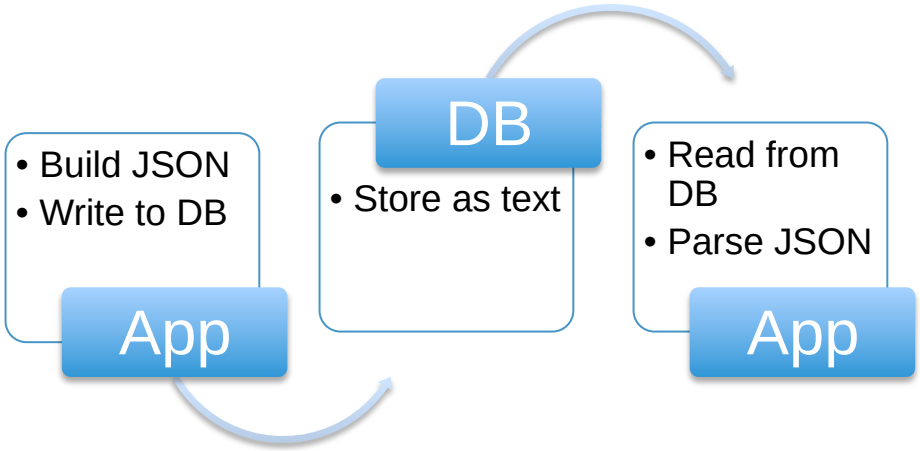


There’s an App for that

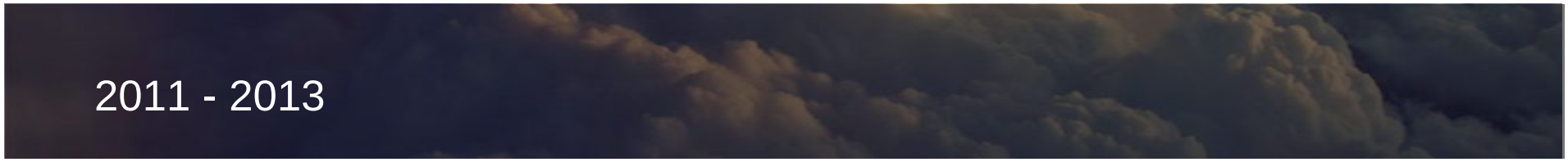
FLITE

2002 - 2011

- Store JSON as text
- Rewrite full string every time
- Parsing happens exclusively in the application layer
- *Or* write your own stored procedures/functions



FLITE



Standard Procedures

FLITE

2011 - 2013

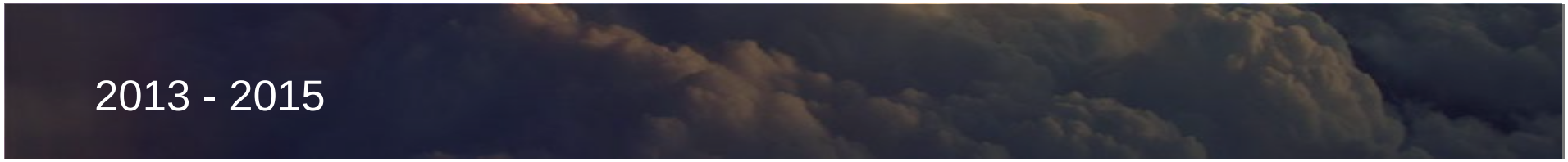
- Store JSON as text
- Rewrite full string every time
- Some simple parsing can be done with common_schema:
- get_option()
- extract_json_value()

FLITE

2011 - 2013

```
mysql> select common_schema.extract_json_value(f.event_data,'/age') as age,
    ->   common_schema.extract_json_value(f.event_data,'/gender') as gender,
    ->   sum(f.event_count) as event_count
    -> from json_event_fact f
    -> group by age, gender;

+----------+---------+-------------+
| age      | gender  | event_count |
+----------+---------+-------------+
| Over 30  | female  |     3710983 |
| Over 30  | male    |     2869302 |
| Under 30 | female  |     5027591 |
| Under 30 | male    |     4918382 |
| unknown  | female  |       42039 |
| unknown  | male    |       50173 |
| unknown  | unknown |        8372 |
+----------+---------+-------------+
```



Lab Experiments

FLITE

2013 - 2015

- Store JSON as text
- Some updates can be done with UDFs
- Much faster parsing supported by UDFs

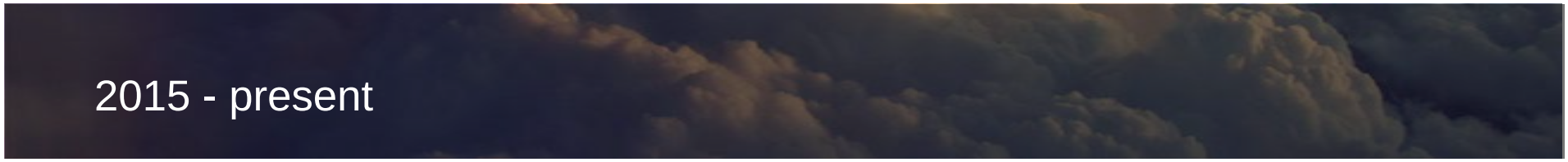
FLITE

2013 - 2015

```
mysql> select json_extract(f.event_data,'state') as state,
    ->   json_extract(f.event_data,'age') as age,
    ->   json_extract(f.event_data,'gender') as gender,
    ->   sum(f.event_count) as event_count
    -> from json_event_fact f
    -> group by state, age, gender;

+-------+----------+--------+-------------+
| state | age      | gender | event_count |
+-------+----------+--------+-------------+
| CA    | 18-24    | female |        5384 |
| CA    | 18-24    | male   |         791 |
| CA    | 18-24    | null   |          25 |
| CA    | 25-34    | female |        6731 |
| CA    | 25-34    | male   |        1292 |
| CA    | 25-34    | null   |          38 |
| CA    | 35-44    | female |       10638 |
| CA    | 35-44    | male   |        1822 |
| CA    | 35-44    | null   |          20 |
+-------+----------+--------+-------------+
```

FLITE



Going Native

FLITE

2015 - present

- Store JSON as text or JSON datatype
- JSON datatype is binary, with keys sorted
- Fast access to embedded data
- Updates executed via native functions
- Extensive parsing supported by native functions

FLITE

JSON_EXTRACT

```
mysql> select json_unquote(json_extract(event_data,'$.country')) as country,
    ->   sum(event_count) as events
    -> from json_event_fact
    -> where d = current_date() - interval 1 day
    -> and ad_id = 2
    -> group by country;
+---------+--------+
| country | events |
+---------+--------+
| nl      | 107954 |
| us      |  27373 |
+---------+--------+
2 rows in set (0.00 sec)
```

FLITE

JSON_SEARCH

```
mysql> select json_search(event_data,'one','android') as json_path
    -> from json_event_fact
    -> having json_path is not null
    -> limit 1;
+-----------+
| json_path |
+-----------+
| "$.os"    |
+-----------+
1 row in set (0.01 sec)

mysql> select json_search('{"fa":"la","la":["la","la"]}','all','la')\G
*************************** 1. row ***************************
json_search('{"fa":"la","la":["la","la"]}','all','la'): ["$.fa", "$.la[0]",
"$.la[1]"]
1 row in set (0.00 sec)
```

JSON_REPLACE

```
mysql> set @json = cast('{"foo":"bar"}' as json);
Query OK, 0 rows affected (0.00 sec)

mysql> set @json = json_replace(@json, '$.foo', 'UPDATED');
Query OK, 0 rows affected (0.00 sec)

mysql> select @json;
+--------------------+
| @json              |
+--------------------+
| {"foo": "UPDATED"} |
+--------------------+
1 row in set (0.00 sec)
```

FLITE

JSON_ARRAY

```
mysql> set @json = cast('{"foo":"bar"}' as json);
Query OK, 0 rows affected (0.00 sec)

mysql> set @json = json_replace(@json, '$.foo', json_array('bar', 'car', 'far'));
Query OK, 0 rows affected (0.01 sec)

mysql> select @json;
+--------------------------------+
| @json                          |
+--------------------------------+
| {"foo": ["bar", "car", "far"]} |
+--------------------------------+
1 row in set (0.00 sec)
```

JSON_OBJECT

```
mysql> set @json = json_object(
    -> 'id','007',
    -> 'name','James Bond',
    -> 'cars',json_array('Alfa Romeo','Aston Martin','BMW')
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> select @json\G
*************************** 1. row ***************************
@json: {"id": "007", "cars": ["Alfa Romeo", "Aston Martin", "BMW"], "name":
"James Bond"}
1 row in set (0.00 sec)
```

JSON COMPARATOR

```
mysql> select cast('"hello"' as json) = 'hello';
1
mysql> select cast(42 as json) = 42;
1
mysql> select cast(false as json) = false;
1
mysql> select cast('{"foo":"bar"}' as json) = cast('{"foo":"bar"}' as
json);
1
```

FLITE

BUT WAIT, THERE'S MORE!

https://dev.mysql.com/doc/refman/5.7/en/json-function-reference.html

Table 13.20 JSON Functions

Name	Description
JSON_APPEND()	Append data to JSON document
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()	Merge JSON documents
JSON_OBJECT()	Create JSON object
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_TYPE()	Type of JSON value
JSON_UNQUOTE()	Unquote JSON value
JSON_VALID()	Whether JSON value is valid

FLITE

FLEXIBLE ROLLUPS

GOAL: Support different fields for multiple customers within a single dimension

FLITE

FLEXIBLE ROLLUPS

For example, some ads track age and gender and others track country and os:

```
{"age":"Over 30","gender":"female"}
{"country":"us","os":"android"}
```

FLEXIBLE ROLLUPS

```
-- try to create rollup with JSON datatype
mysql> create table json_event_fact (
    ->   d date not null,
    ->   ad_id int not null,
    ->   event_data json not null,
    ->   event_count int not null,
    ->   primary key (d,ad_id,event_data)
    -> );
ERROR 3152 (42000): JSON column 'event_data' cannot be
used in key specification.
```

FLITE

FLEXIBLE ROLLUPS

```
-- use text instead
mysql> create table json_event_fact (
    ->   d date not null,
    ->   ad_id int not null,
    ->   event_data varchar(750) not null,
    ->   event_count int not null,
    ->   primary key (d,ad_id,event_data)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

FLITE

FLEXIBLE ROLLUPS

```
-- generated column hack
mysql> create table json_event_fact (
    ->   d date not null,
    ->   ad_id int not null,
    ->   event_data json not null,
    ->   event_data_text varchar(750) as (cast(event_data
as char)) stored,
    ->   event_count int not null,
    ->   primary key (d,ad_id,event_data_text)
    -> );
Query OK, 0 rows affected (0.16 sec)
```

FLITE

FLEXIBLE ROLLUPS

```
mysql> select event_data->'$.age' as age,
    ->   sum(event_count) as events
    -> from json_event_fact
    -> where d = current_date() - interval 1 day
    -> and ad_id = 1
    -> group by age;
+------------+---------+
| age        | events  |
+------------+---------+
| "Over 30"  |  810424 |
| "Under 30" | 1205544 |
+------------+---------+
2 rows in set (0.03 sec)
```

FLITE

FLEXIBLE ROLLUPS

```
mysql> select json_unquote(event_data->'$.country') as country,
    ->   sum(event_count) as events
    -> from json_event_fact
    -> where d = current_date() - interval 1 day
    -> and ad_id = 2
    -> group by country;
+---------+--------+
| country | events |
+---------+--------+
| nl      | 107954 |
| us      |  27373 |
+---------+--------+
2 rows in set (0.00 sec)
```

FLITE

CONFIGURATION DATA

GOAL: Store extensible configuration data of arbitrary depth for each ad

FLITE

CONFIGURATION DATA

```
create table ad_config_data (
  ad_config_data_id int not null primary key,
  ad_id int not null,
  name varchar(50) not null,
  config_data json not null
);
```

FLITE

CONFIGURATION DATA

```
{
  "a56a81eb": {
    "type": "AD",
    "uuid": "d9e9d8ae-8a33-11e6-97e0-22000b93579c",
    "subConfig": {},
    "dataSupport": true
  },
  "a6529578": {
    "type": "VIDEO",
    "uuid": "e09b40af-8a33-11e6-97e0-22000b93579c",
    "subConfig": {
      "video_url": "https://www.youtube.com/watch?v=dQw4w9WgXcQ",
      "hd": "true”
    }
  },
  "a6caab6e": {
    "type": "AD",
    "uuid": "e89a3877-8a33-11e6-97e0-22000b93579c",
    "subConfig": {}
  }
}
```

CONFIGURATION DATA

```
{
  "paths": {
    "path_3007ea93": ["action_312c40f4"],
    "path_30972a80": ["action_3158f2a7", "action_3185a6da", "action_31aedd9b"],
    …
  },
  "actions": {
    "action_312c40f4": {
      …
    }
  }
}
```

CONFIGURATION DATA

```
mysql> select json_extract(config_data,'$.paths.path_30c06190') as sub_paths from
ad_config_data where ad_id = 1 and name = 'actions'\G
*************************** 1. row ***************************
sub_paths: ["action_5b5343af", "action_5b8b6c39", "action_5bbb05d6"]
1 row in set (0.00 sec)

mysql> update ad_config_data
    -> set config_data =
json_replace(config_data,'$.paths.path_30c06190',json_array('action_5b5343af',
'action_5bbb05d6'))
    -> where ad_id = 1 and name = 'actions';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select json_extract(config_data,'$.paths.path_30c06190') as sub_paths from
ad_config_data where ad_id = 1 and name = 'actions'\G
*************************** 1. row ***************************
sub_paths: ["action_5b5343af", "action_5bbb05d6"]
1 row in set (0.00 sec)
```

CONFIGURATION DATA

```
mysql> select json_extract(config_data,'$.*.type')
    -> from ad_config_data
    -> where ad_id = 1
    -> and name = 'layers';
+-------------------------------------+
| json_extract(config_data,'$.*.type') |
+-------------------------------------+
| ["AD", "VIDEO", "AD"]                |
+-------------------------------------+
1 row in set (0.00 sec)
```

FLITE

CONFIGURATION DATA

```
mysql> select json_search(config_data,'one','action_312c40f4')
    -> from ad_config_data
    -> where ad_id = 1
    -> and name = 'actions';
+-----------------------------------------------+
| json_search(config_data,'one','action_312c40f4') |
+-----------------------------------------------+
| "$.paths.path_3007ea93[0]"                    |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

FLITE

FLITE

EAV ANTIDOTE

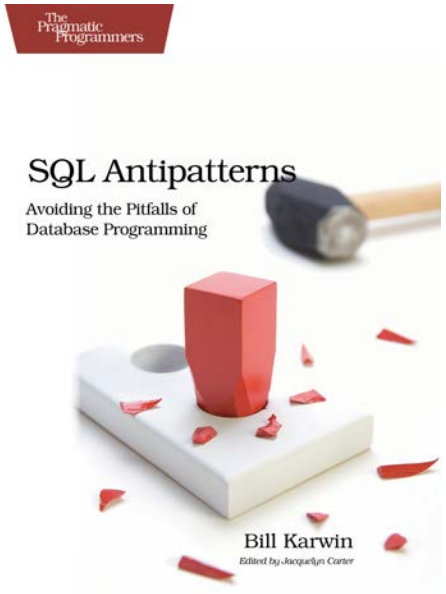
GOAL: Store Entity-Attribute-Value style data while avoiding the EAV antipattern

FLITE

EAV ANTIDOTE

The EAV antipattern is described well by Bill Karwin in his book: “SQL Antipatterns”

In Bill’s example the EAV anti-pattern is used to store two types of issues (bugs and features) in a single shared table.

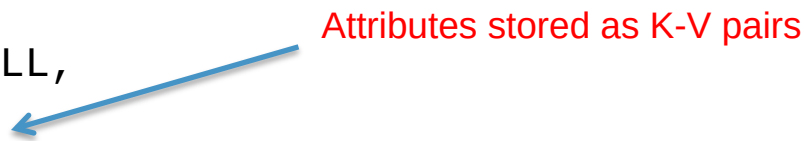


FLITE

EAV ANTIDOTE

```
CREATE TABLE Issues (
  issue_id    SERIAL PRIMARY KEY
);

CREATE TABLE IssueAttributes (
  issue_id    BIGINT UNSIGNED NOT NULL,
  attr_name   VARCHAR(100) NOT NULL,
  attr_value  VARCHAR(100),
  PRIMARY KEY (issue_id, attr_name),
  FOREIGN KEY (issue_id) REFERENCES Issues(issue_id)
);
```



FLITE

EAV ANTIDOTE

```
CREATE TABLE Issues (
  issue_id          SERIAL PRIMARY KEY,
  reported_by       BIGINT UNSIGNED NOT NULL,
  product_id        BIGINT UNSIGNED,
  priority          VARCHAR(20),
  version_resolved  VARCHAR(20),
  status            VARCHAR(20),
  issue_type        VARCHAR(10),   -- BUG or FEATURE
  attributes        TEXT NOT NULL, -- all dynamic attributes for the
row
  FOREIGN KEY (reported_by) REFERENCES Accounts(account_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

FLITE

EAV ANTIDOTE

```
CREATE TABLE Issues (
  issue_id         SERIAL PRIMARY KEY,
  reported_by      BIGINT UNSIGNED NOT NULL,
  product_id       BIGINT UNSIGNED,
  priority         VARCHAR(20),
  version_resolved VARCHAR(20),
  status           VARCHAR(20),
  issue_type       VARCHAR(10),   -- BUG or FEATURE
  attributes       JSON NOT NULL, -- all dynamic attributes for the row
  severity         VARCHAR(20) AS (attributes->"$.severity"),  -- only for bugs
  version_affected VARCHAR(20) AS (attributes->"$.version_affected"),  -- only for bugs
  sponsor          VARCHAR(50) AS (attributes->"$.sponsor"),  -- only for feature
requests
  FOREIGN KEY (reported_by) REFERENCES Accounts(account_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

FLITE

JSON + GENERATED COLUMNS

- Allows you to expose one or more JSON fields as table columns
- Supports indexes
- Choose virtual (not stored) unless the index is Primary Key, FULLTEXT, or GIS

FLITE

EXAMPLE #1: ADD AGE COLUMN TO ROLLUP TABLE

```
mysql> alter table json_event_fact
    -> add column age varchar(20) as (event_data->'$.age');
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select age,
    ->   sum(event_count) as events
    -> from json_event_fact
    -> where d = current_date() - interval 1 day
    -> and ad_id = 1
    -> group by age;
+------------+---------+
| age        | events  |
+------------+---------+
| "Over 30"  |  810424 |
| "Under 30" | 1205544 |
+------------+---------+
2 rows in set (0.00 sec)
```

FLITE

EXAMPLE #2: ADD GENDER COLUMN/INDEX TO ROLLUP TABLE

```
mysql> alter table json_event_fact
    -> add column gender varchar(20) as (event_data->'$.gender'),
    -> add index (gender);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select sum(event_count) as events
    -> from json_event_fact
    -> where gender = '"female"';
+---------+
| events  |
+---------+
| 1081286 |
+---------+
1 row in set (0.00 sec)
```

FLITE

FLITE

TEXT VS. JSON DATA TYPE

- Use the JSON datatype in general
- There are a few exceptions. Use text types if you need to:
- Include the column in a primary key
- Store heterogeneous data (some rows are strings, some are JSON)
- Store JSON with depth greater than 100

FLITE

TEXT VS. JSON DATA TYPE

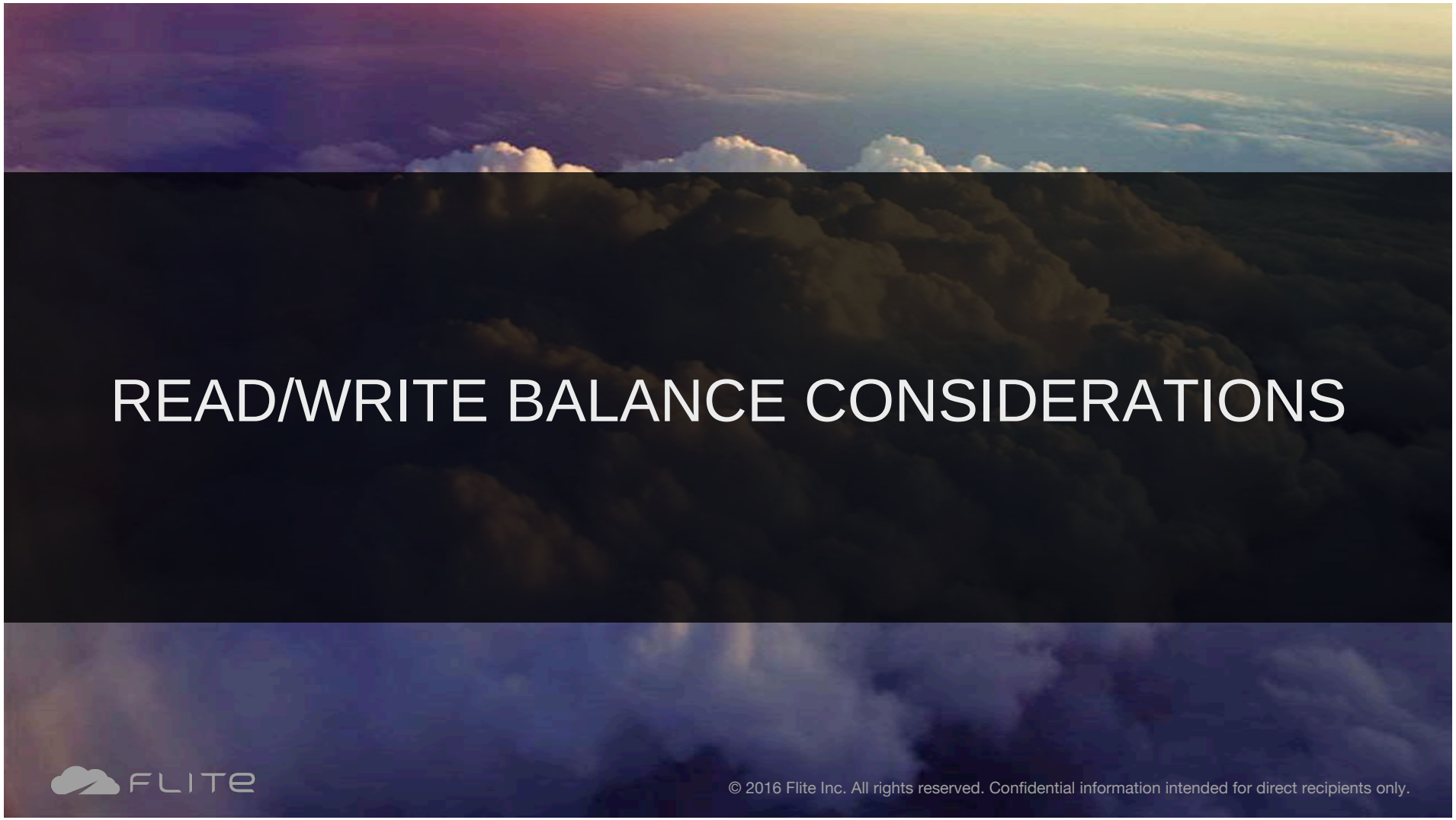
JSON data type sorts by keys:

```
mysql> select cast('{"c":"3","b":{"2":"2","1":"1"},"a":"1"}' as json) as key_sort_test;
+-------------------------------------------------+
| key_sort_test                                   |
+-------------------------------------------------+
| {"a": "1", "b": {"1": "1", "2": "2"}, "c": "3"} |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

FLITE

TEXT VS. JSON DATA TYPE

- Use the JSON_VALID() function to test validity of existing column values before you run ALTER TABLE
- Be aware that the JSON type automatically uses the utf8mb4 character set
- Morgan Tocker has a good blog post on this topic: http://mysqlserverteam.com/upgrading-json-data-stored-in-text-columns/



FLITE

READ/WRITE BALANCE CONSIDERATIONS

- New JSON type is read-optimized
- Every update requires rewriting the entire object
- Performance improvements are planned for writes:

http://dev.mysql.com/worklog/task/?id=9141
http://dev.mysql.com/worklog/task/?id=8985

FLITE

DISK STORAGE IMPLICATIONS

DISK STORAGE

- The JSON data type's binary format uses about the same amount of space as text types.
- Given the nature of JSON data (keys are repeated in every row), JSON data tends to compress well.

FLITE

GOTCHAS

- Namespace collisions between JSON UDFs and native functions:

```
JSON_APPEND()
JSON_DEPTH()
JSON_EXTRACT()
JSON_MERGE()
JSON_REMOVE()
JSON_REPLACE()
JSON_SEARCH()
JSON_SET()
JSON_VALID()
```

FLITE

GOTCHAS

- Stricter behavior of native functions

```
-- MySQL 5.6 UDF
mysql> select json_extract('','foo');
+------------------------+
| json_extract('','foo') |
+------------------------+
| NULL                   |
+------------------------+
1 row in set (0.00 sec)

-- MySQL 5.7 native function
mysql> select json_extract('','$.foo');
ERROR 3141 (22032): Invalid JSON text in argument 1 to function json_extract: "The
document is empty." at position 0.
```

GOTCHAS

- Native functions output JSON, not text

```
-- MySQL 5.6 UDF
mysql> select json_extract('{"foo":"bar"}','foo');
+-------------------------------------+
| json_extract('{"foo":"bar"}','foo') |
+-------------------------------------+
| bar                                 |
+-------------------------------------+
1 row in set (0.00 sec)

-- MySQL 5.7 native function
mysql> select json_extract('{"foo":"bar"}','$.foo');
+--------------------------------------+
| json_extract('{"foo":"bar"}','$.foo') |
+--------------------------------------+
| "bar"                                |
+--------------------------------------+
1 row in set (0.12 sec)
```

GOTCHAS

- Path differences between JSON UDFs and native functions

```
-- MySQL 5.6 UDF
mysql> select json_extract('{"parent":{"child":"hello"}}','parent','child');
+-----------------------------------------------------------------+
| json_extract('{"parent":{"child":"hello"}}','parent','child') |
+-----------------------------------------------------------------+
| hello                                                           |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

-- MySQL 5.7 native function
mysql> select json_extract('{"parent":{"child":"hello"}}','$.parent.child');
+-----------------------------------------------------------------+
| json_extract('{"parent":{"child":"hello"}}','$.parent.child') |
+-----------------------------------------------------------------+
| "hello"                                                         |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

FLITE

GOTCHAS

- Cannot index JSON columns directly: use generated columns for indexing

FLITE

GOTCHAS

- Validating existing JSON values can be difficult if some rows are not valid JSON and other rows have a depth higher than 100:

```
mysql> select json_text from old_table where
json_valid(json_text) = 0;
ERROR 3157 (22032): The JSON document exceeds the maximum
depth.

mysql> select id, json_depth(json_text) from old_table;
ERROR 3141 (22032): Invalid JSON text in argument 1 to
function json_depth: "Invalid value." at position 0.
```

Rate My Session!