MySQL Implementation Essentials Bootcamp

Database Design -bonus

Selena Sanchez` (Selena.sanchez@oracle.com)

Selena Sánchez, MySQL Solutions Engineer

JSON Data

- JSON (JavaScript Object Notation) defined by RFC 7159
    - Open standard file format that uses human-readable text
- It use Objects and Arrays
- Examples

```
"firstName":"John"

{"firstName":"John", "lastname":"Doe"}

"employess":[
 {"firstName":"John", "lastname":"Doe"}
 {"firstName":"Anna", "lastname":"Smith"}
 {"firstName":"Peter","lastname":"Jones"}
]
```

JSON: Why is it popular?

- Developer Friendly
  - Simple data format that allows programmers to store and communicate sets of values
  - JSON's lack of a predefined schema makes it easy for developers to store and update documents
- Its ease of use, speed of processing and easy integration made it extremely popular for Web-based applications
- As JSON adoption has grown, JSON-centric document databases have become more popular
- MySQL 5.7 integrated JSON features, resulting in a best-of-both worlds benefit to developers and database administrators

Core New JSON features in MySQL 8.0

- Native JSON data type
  - Native internal binary format for efficient processing & storage
  - Up to 10x faster than storing as text
  - 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'.
```

- Built-in JSON functions
  - Allowing you to efficiently store, search, update, and manipulate Documents
- Indexing of Documents using Generated Columns (Indirect)
  - Automatically uses the best "functional" index available for even faster results
  - Key/Array references enable quick read-access to look up document elements directly
- JSON Comparator & New inline syntax for easy SQL integration
  - Allows for easy integration of Document data within your SQL queries

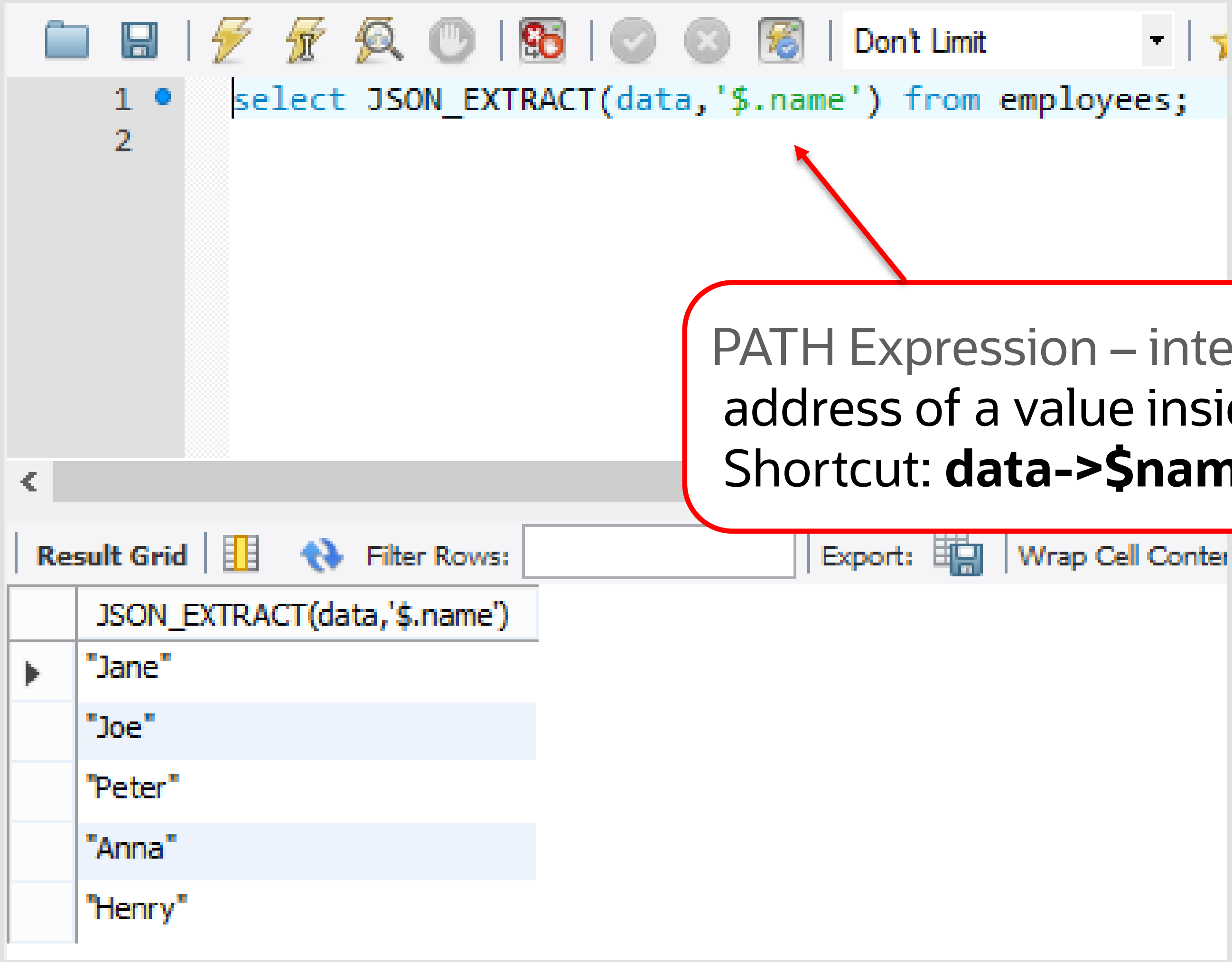
JSON Functions

- 5.7/8.0 supports functions to CREATE, SEARCH, MODIFY and RETURN JSON values

JSON_ARRAY_APPEND()	JSON_LENGTH()	JSON_UNQUOTE()
JSON_ARRAY_INSERT()	JSON_MERGE()	JSON_VALID()
JSON_ARRAY()	JSON_OBJECT()	JSON_PRETTY()
JSON_CONTAINS_PATH()	JSON_QUOTE()	JSON_STORAGE_SIZE()
JSON_CONTAINS()	JSON_REMOVE()	JSON_STORAGE_FREE()
JSON_DEPTH()	JSON_REPLACE()	JSON_ARRAYAGG()
JSON_EXTRACT()	JSON_SEARCH()	JSON_OBJECTAGG()
JSON_INSERT()	JSON_SET()	JSON_TABLE()
JSON_KEYS()	JSON_TYPE()	

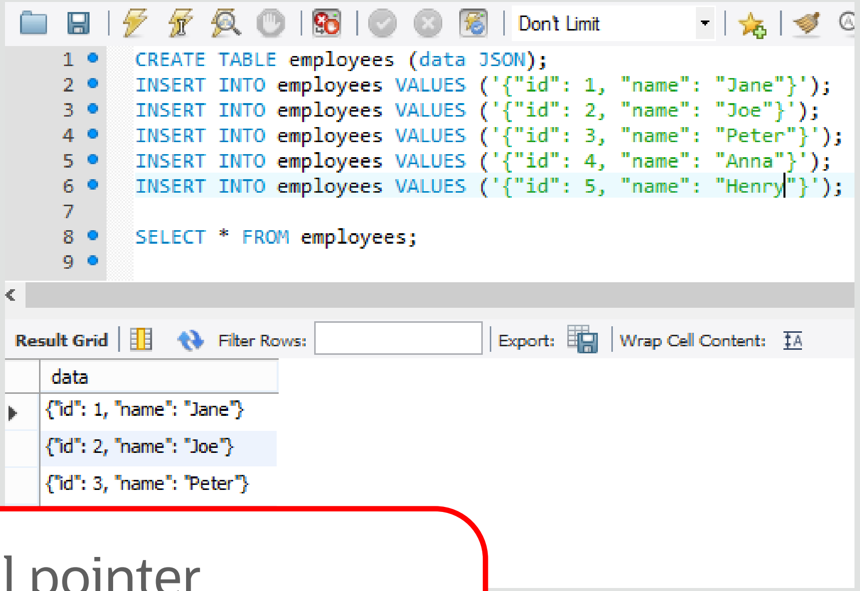
https://dev.mysql.com/doc/refman/8.0/en/json-functions.html

JSON reference



```
select JSON_EXTRACT(data,'$.name') from employees;
```

JSON_EXTRACT(data,'$.name')
"Jane"
"Joe"
"Peter"
"Anna"
"Henry"



```
CREATE TABLE employees (data JSON);
INSERT INTO employees VALUES ('{"id": 1, "name": "Jane"}');
INSERT INTO employees VALUES ('{"id": 2, "name": "Joe"}');
INSERT INTO employees VALUES ('{"id": 3, "name": "Peter"}');
INSERT INTO employees VALUES ('{"id": 4, "name": "Anna"}');
INSERT INTO employees VALUES ('{"id": 5, "name": "Henry"}');

SELECT * FROM employees;
```

data
{"id": 1, "name": "Jane"}
{"id": 2, "name": "Joe"}
{"id": 3, "name": "Peter"}

PATH Expression – internal pointer
address of a value inside a JSON document
Shortcut: **data->$name**

Introducing Generated Columns

- Columns computed from an expression included in the column definition
  - VIRTUAL: computed when read, not stored, indexable
  - STORED: computed when inserted/updated, stored in SE, indexable

id	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!

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

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

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

Creates index only. Does not modify table rows

```
SELECT feature FROM features WHERE feature_type = “feature“;
. . .

SELECT feature FROM features WHERE feature->"$.type” = “feature”;
…
```

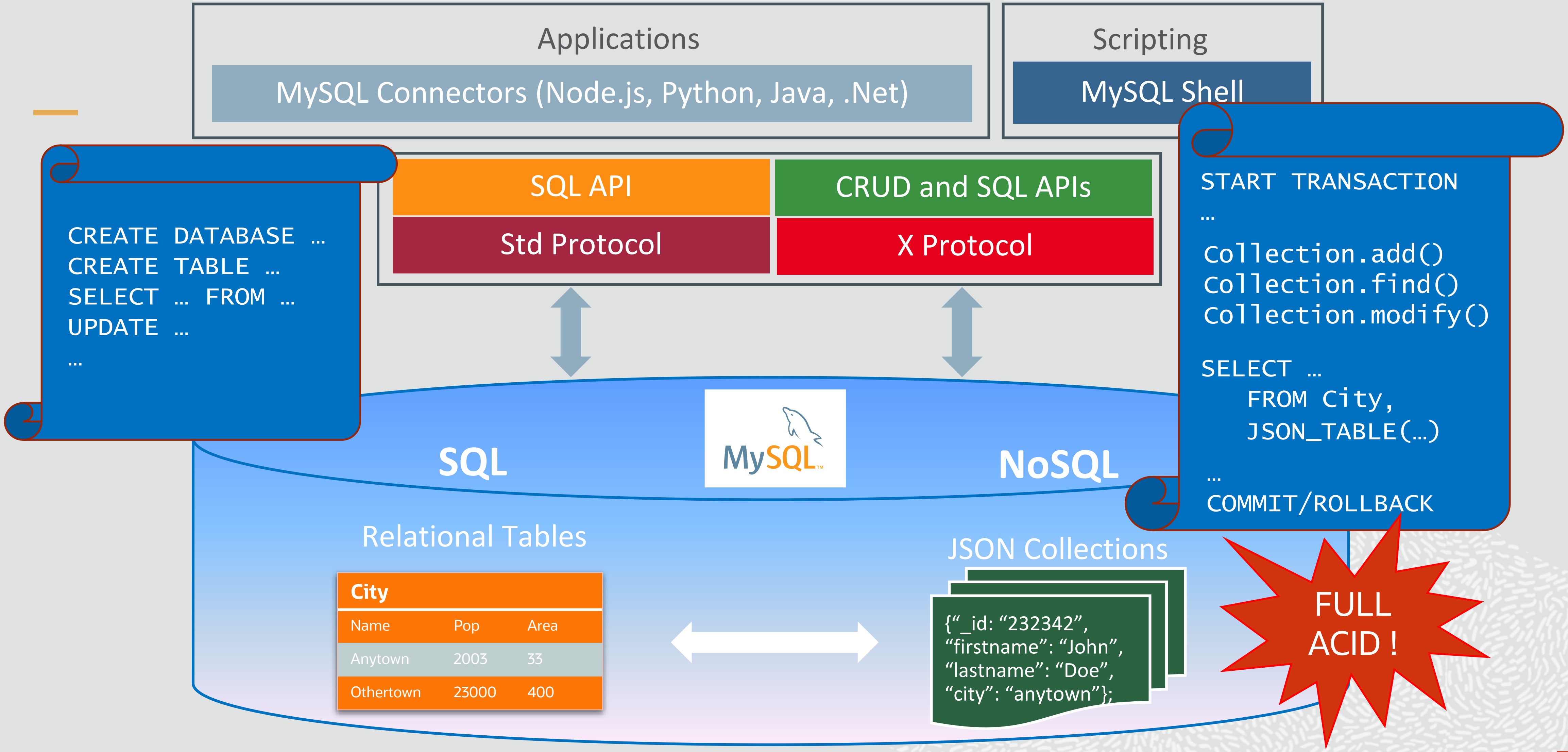
Optimizer automatically recognizes Index expressions

DBMS or NoSQL ?

Why not both ?

https://www.youtube.com/watch?v=1Dk517M-_7o

MySQL 8.0 Document Store: SQL+NoSQL database



```
CREATE DATABASE …
CREATE TABLE …
SELECT … FROM …
UPDATE …
…
```

```
START TRANSACTION
…
Collection.add()
Collection.find()
Collection.modify()

SELECT …
   FROM City,
   JSON_TABLE(…)
…
COMMIT/ROLLBACK
```

City		
Name	Pop	Area
Anytown	2003	33
Othertown	23000	400

```
{"_id: "232342",
"firstname": "John",
"lastname": "Doe",
"city": "anytown"};
```

Tables or Collections?

- A collection is a table with 2+ columns:
  - Primary key: `_id`
  - JSON document: `doc`
- The document's `_id` field could be supplied or automatically generated as UUID
  - This field could be 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

X DevAPI

- Use SQL, CRUD APIs – Document (NoSQL) and Relational (SQL), or “All of the Above”
  - All of this is in addition to the Classic APIs
- Implemented in connectors for
  - C++, Java, .Net, Node.js, Python, PHP
  - working with Communities
- Non-blocking, asynchronous calls follow common language pat erns
- Supports CRUD operations

Operation	Document	Relational
Create	Collection.add()	Table.insert()
Read	Collection.find()	Table.select()
Update	Collection.modify()	Table.update()
Delete	Collection.remove()	Table.delete()

https://dev.mysql.com/doc/refman/8.0/en/mysql-shell-tutorial-javascript.html

MySQL Document Store cheat sheet

```
js> session.createSchema('name')
js> \use name
js> db.getCollections()
js> db.createCollection(‘myCollection')
js> db.getCollections()
js> db.myCollection.add({“param1":“value1", “param2":“value2”})
```

Create

```
js> db.myCollection.find()
js> db.myCollection.find().limit(1)
js> db.myCollection.find("_id = '00005af018430000000000000002'")
```

Read

```
js> db.myCollection.modify("_id = '1234'").set(“param",“value")
```

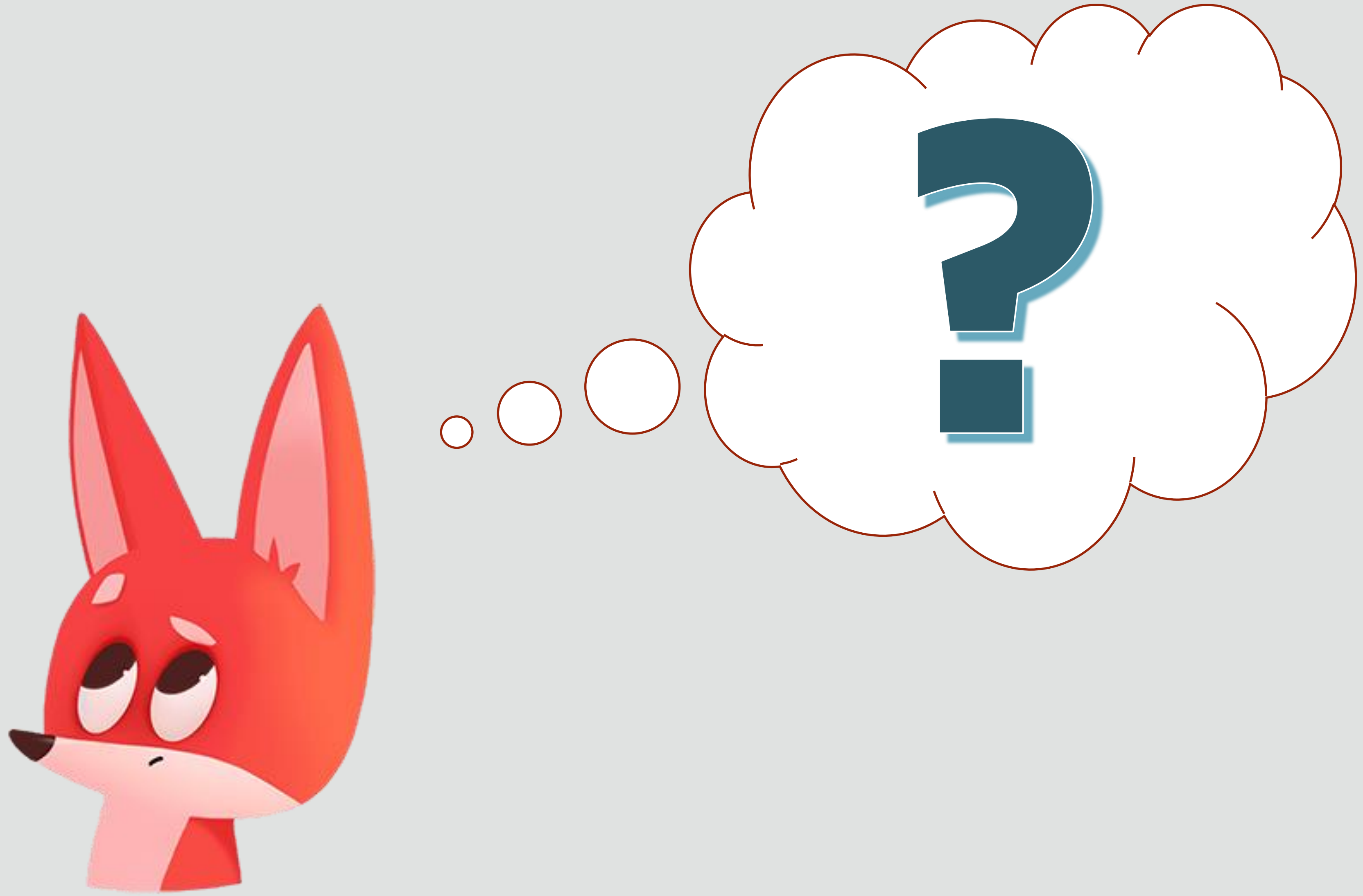
Update

```
js> db.myCollection.remove("_id = '1234'")
```

Delete

```
js> session.startTransaction()
js> …
js> session.rollback()
```

Transactions



Hands-On Labs

3b. MySQL JSON datatype

3c. MySQL Document Store

ORACLE