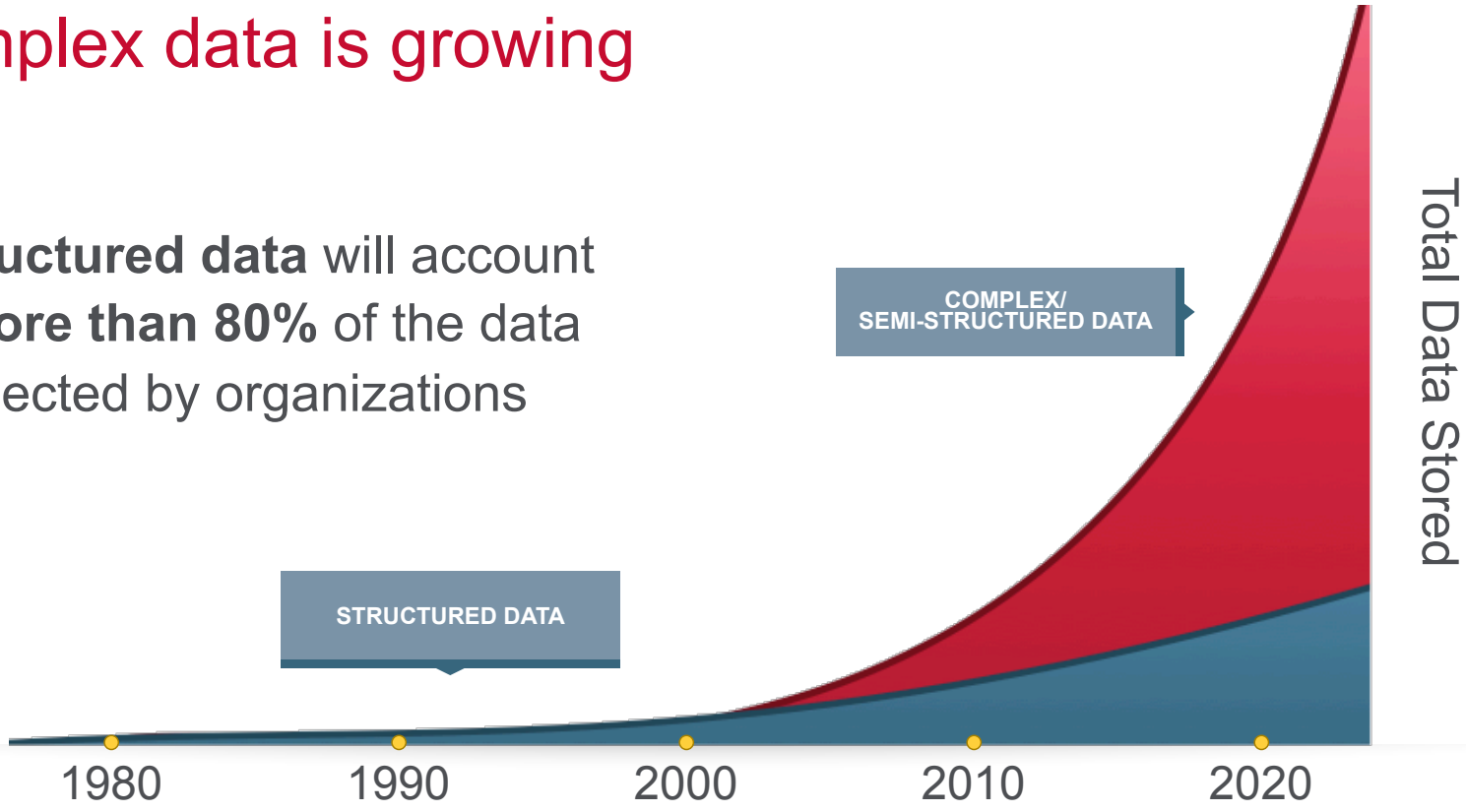
Adding complex data to Spark stack

Neeraja Rentachintala, Director of Product Management

6/16/2015

Complex data is growing

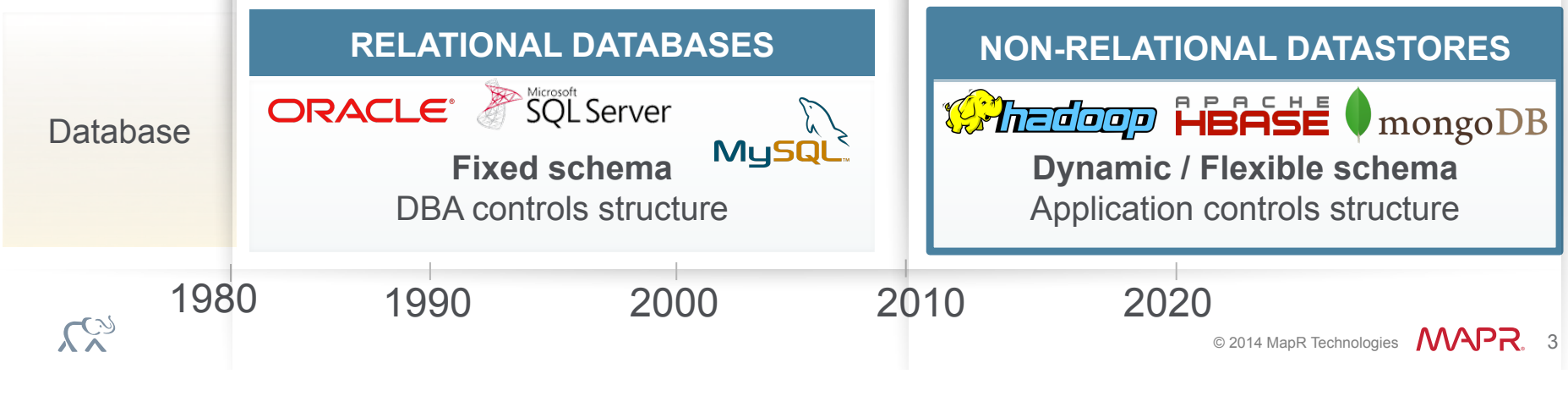
Unstructured data will account for **more than 80%** of the data collected by organizations



Source: Human-Computer Interaction & Knowledge Discovery in Complex Unstructured, Big Data

Data Increasingly Stored in *Non-Relational* Datastores

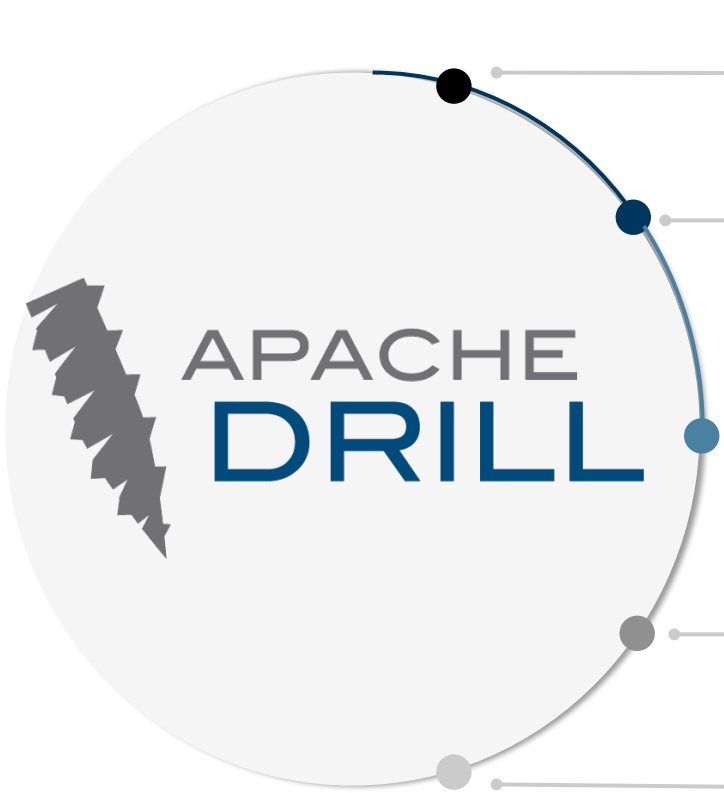
Volume	GBs-TBs	TBs-PBs
Structure	Structured	Structured, semi-structured and unstructured
Development	Planned (release cycle = months-years)	Iterative (release cycle = days-weeks)



Session topics

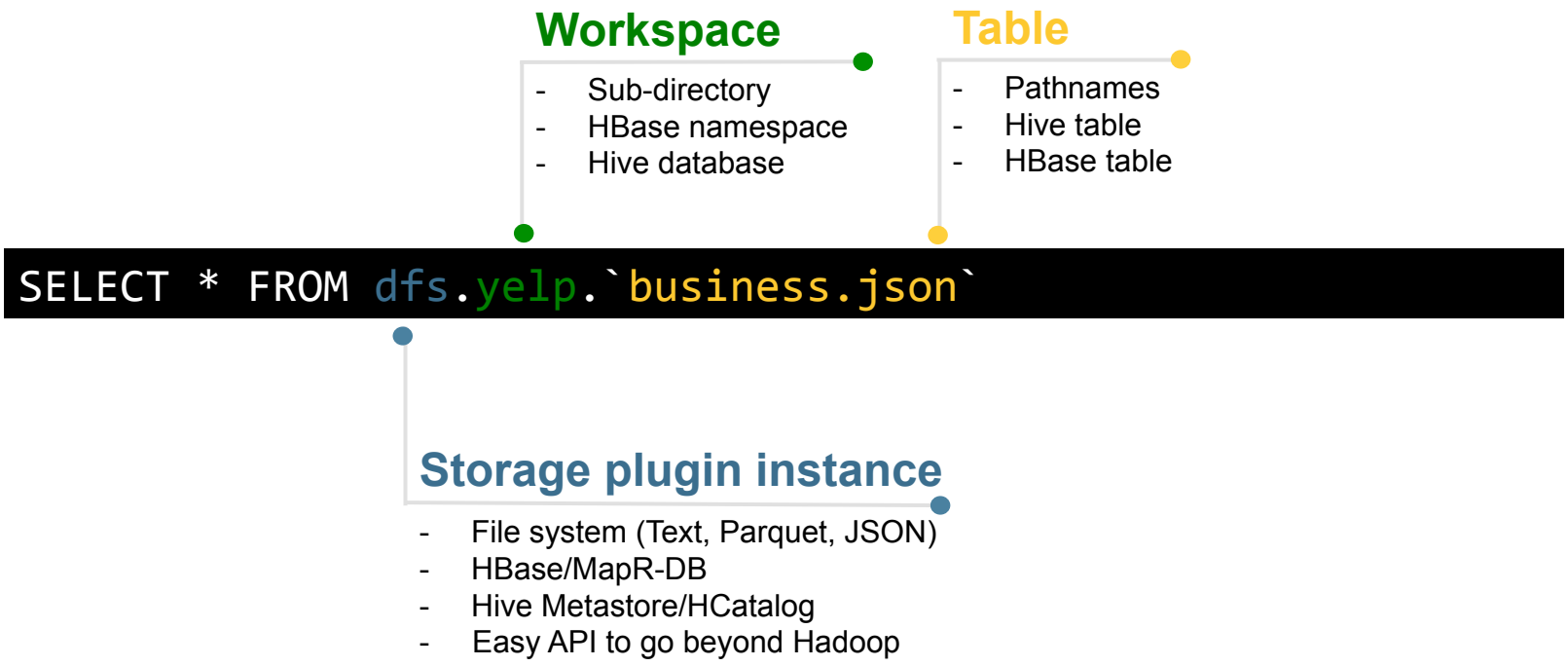
- Apache Drill overview
- Integrating Drill & Spark
- Status & Next steps

Drill is the Industry's first Schema-Free SQL engine



- Point-and-query vs. schema-first
- Access to any data source & types
- Industry standard APIs (ANSI SQL, JDBC/ODBC, REST)
- Performance at Scale
- Extreme Ease of Use

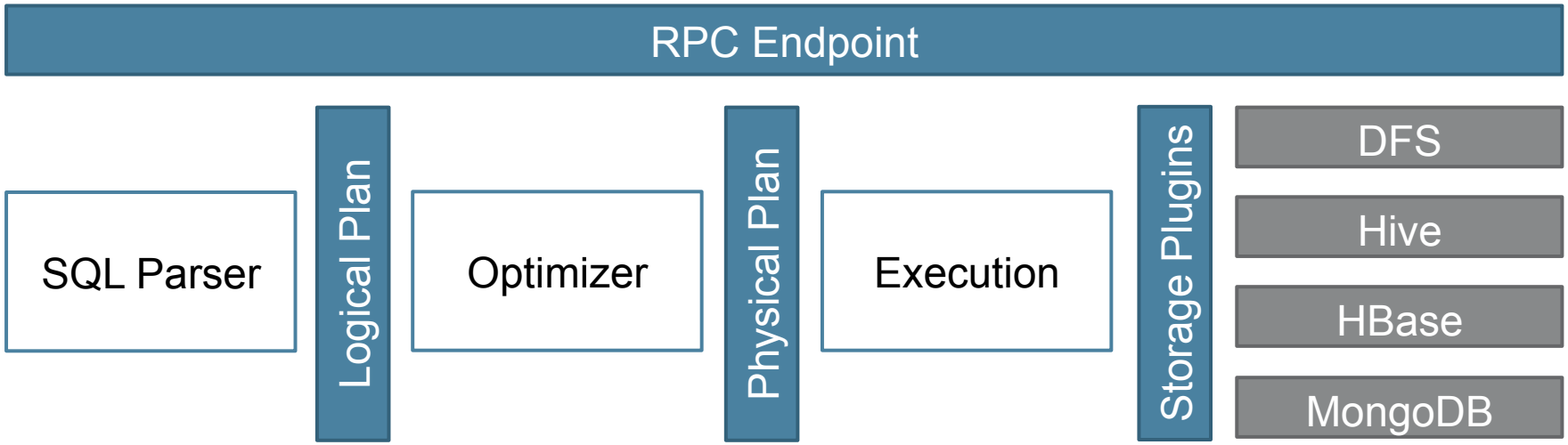
Drill enables ‘SQL on Everything’ (Omni-SQL)



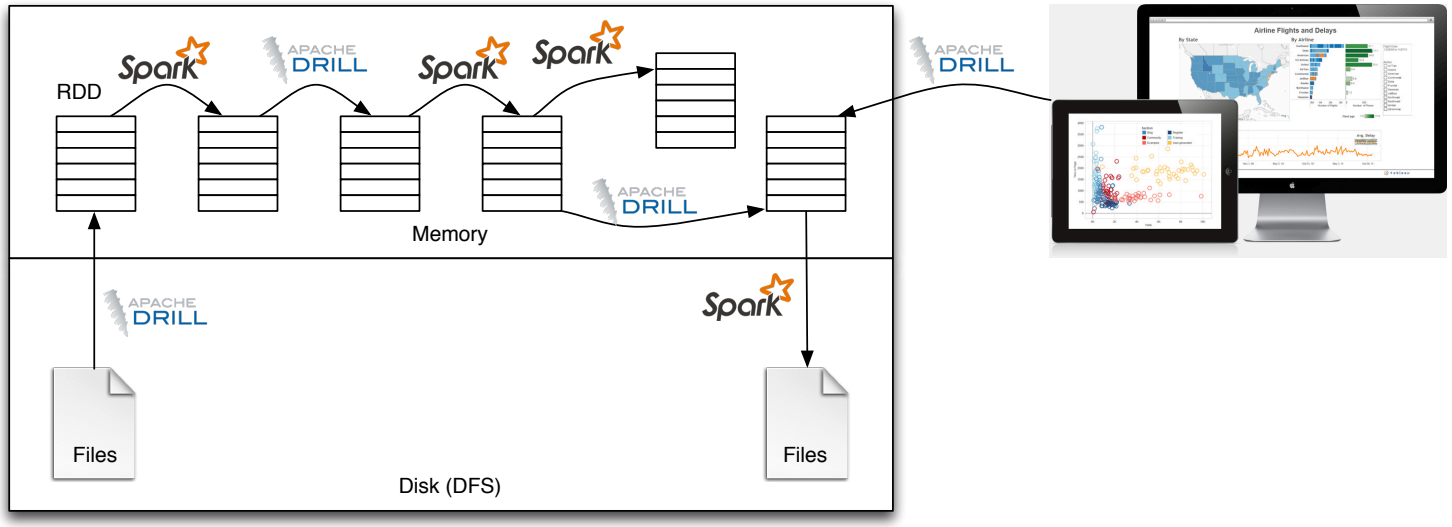
Drill is a Distributed SQL query engine

- Scale-out (single node to 1000's of nodes)
- Columnar and Vectorized execution
- Optimistic execution (no MR, Spark, Tez)
- Extensible

Core Modules within `drillbit`



Integrating Spark and Drill

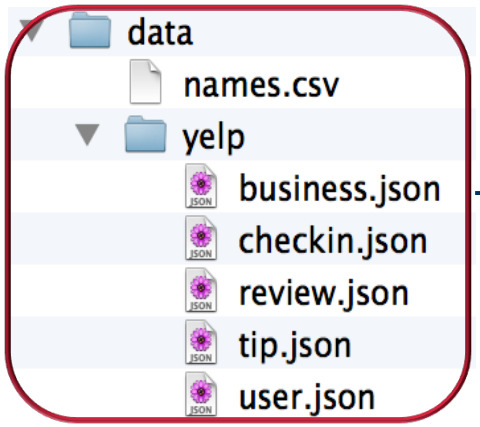


Features at a glance:

- Use Drill as an input to Spark
- Query Spark RDDs via Drill and create data pipelines

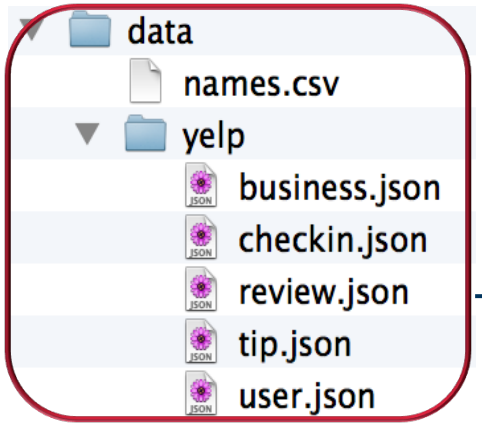
Drill examples walkthrough

business.json




```
{
        "business_id": "4bEjOyTaDG24SY5TxsaUNQ",
        "full_address": "3655 Las Vegas Blvd S\nThe Strip\nLas Vegas, NV 89109",
        "hours": {
                "Monday": {"close": "23:00", "open": "07:00"},
                "Tuesday": {"close": "23:00", "open": "07:00"},
                "Friday": {"close": "00:00", "open": "07:00"},
                "Wednesday": {"close": "23:00", "open": "07:00"},
                "Thursday": {"close": "23:00", "open": "07:00"},
                "Sunday": {"close": "23:00", "open": "07:00"},
                "Saturday": {"close": "00:00", "open": "07:00"}
        },
        "open": true,
        "categories": ["Breakfast & Brunch", "Steakhouses", "French", "Restaurants"],
        "city": "Las Vegas",
        "review_count": 4084,
        "name": "Mon Ami Gabi",
        "neighborhoods": ["The Strip"],
        "longitude": -115.172588519464,
        "state": "NV",
        "stars": 4.0,
        "attributes": {
                "Alcohol": "full_bar”,
                "Noise Level": "average",
                "Has TV": false,
                "Attire": "casual",
                "Ambience": {
                        "romantic": true,
                        "intimate": false,
                        "touristy": false,
                        "hipster": false,
                        "classy": true,
                        "trendy": false,
                        "casual": false
                },
                "Good For": {"dessert": false, "latenight": false, "lunch": false,
                      "dinner": true, "breakfast": false, "brunch": false},
        }
}
```

review.json




```
{
  "votes": {"funny": 0, "useful": 2, "cool": 1},
  "user_id": "Xqd0DzHaiyRqVH3WRG7hzg",
  "review_id": "15SdjuK7DmYqUAj6rjGowg",
  "stars": 5,
  "date": "2007-05-17",
  "text": "dr. goldberg offers everything ...",
  "type": "review",
  "business_id": "vcNAWiLM4dR7D2nwwJ7nCA"
}
```


Zero to Results in 2 minutes

```
$ tar -xvzf apache-drill-1.0.0.tar.gz

$ bin/sqlline -u jdbc:drill:zk=local

> SELECT state, city, count(*) AS businesses
  FROM dfs.yelp.`business.json`
  GROUP BY state, city
  ORDER BY businesses DESC LIMIT 10;
```

state	city	businesses
NV	Las Vegas	12021
AZ	Phoenix	7499
AZ	Scottsdale	3605
EDH	Edinburgh	2804
AZ	Mesa	2041
AZ	Tempe	2025
NV	Henderson	1914
AZ	Chandler	1637
WI	Madison	1630
AZ	Glendale	1196

Install

Launch shell (embedded mode)

Query files and directories

Results

Directories are implicit partitions

```
sales
├── 2014
│   ├── q1
│   ├── q2
│   ├── q3
│   └── q4
└── 2015
    └── q1
```

```
SELECT dir0, SUM(amount)
FROM sales
GROUP BY dir1 IN (q1, q2)
```

Intuitive SQL access to complex data

```
// It's Friday 10pm in Vegas and looking for Hummus

> SELECT name, stars, b.hours.Friday friday, categories
  FROM dfs.yelp.`business.json` b
  WHERE b.hours.Friday.`open` < '22:00' AND
        b.hours.Friday.`close` > '22:00' AND
        REPEATED_CONTAINS(categories, 'Mediterranean') AND
        city = 'Las Vegas'
  ORDER BY stars DESC
  LIMIT 2;

+------------+------------+------------+------------+
|    name    |   stars    |   friday   | categories |
+------------+------------+------------+------------+
| Olives     | 4.0        | {"close":"22:30","open":"11:00"} |
["Mediterranean","Restaurants"] |
| Marrakech Moroccan Restaurant | 4.0        | {"close":"23:00","open":"17:30"} |
["Mediterranean","Middle Eastern","Moroccan","Restaurants"] |
+------------+------------+------------+------------+
```

Query data with any levels of nesting

ANSI SQL compatibility

```
//Get top cool rated businesses

➢ SELECT b.name from dfs.yelp.`business.json` b
   WHERE b.business_id IN
  (SELECT r.business_id FROM dfs.yelp.`review.json` r
   GROUP BY r.business_id HAVING SUM(r.votes.cool) > 2000 ORDER BY
   SUM(r.votes.cool) DESC);

+------------+
|    name    |
+------------+
| Earl of Sandwich |
| XS Nightclub |
| The Cosmopolitan of Las Vegas |
| Wicked Spoon |
+------------+
```

Use familiar SQL functionality (Joins, Aggregations, Sorting, Sub-queries, SQL data types)

Logical views

```
//Create a view combining business and reviews datasets

> CREATE OR REPLACE VIEW dfs.tmp.BusinessReviews AS
    SELECT b.name, b.stars, r.votes.funny,
           r.votes.useful, r.votes.cool, r.`date`
      FROM dfs.yelp.`business.json` b, dfs.yelp.`review.json` r
      WHERE r.business_id = b.business_id;

+------------+------------+
|     ok     |  summary   |
+------------+------------+
| true       | View 'BusinessReviews' created successfully in 'dfs.tmp' schema |
+------------+------------+

> SELECT COUNT(*) AS Total FROM dfs.tmp.BusinessReviews;

+------------+
|    Total   |
+------------+
|  1125458   |
+------------+
```

Lightweight file system based views for granular and de-centralized data management

Materialized Views AKA Tables

```
> ALTER SESSION SET `store.format` = 'parquet';

> CREATE TABLE dfs.yelp.BusinessReviewsTbl AS
    SELECT b.name, b.stars, r.votes.funny funny,
           r.votes.useful useful, r.votes.cool cool, r.`date`
      FROM dfs.yelp.`business.json` b, dfs.yelp.`review.json` r
      WHERE r.business_id = b.business_id;

+------------+--------------------------+
|  Fragment  | Number of records written |
+------------+--------------------------+
| 1_0        | 176448                   |
| 1_1        | 192439                   |
| 1_2        | 198625                   |
| 1_3        | 200863                   |
| 1_4        | 181420                   |
| 1_5        | 175663                   |
+------------+--------------------------+
```

Save analysis results as tables using familiar CTAS syntax

Extensions to ANSI SQL to work with repeated values

```
// Flatten repeated categories

> SELECT name, categories
  FROM dfs.yelp.`business.json` LIMIT 3;

+------------+------------+
|    name    | categories |
+------------+------------+
| Eric Goldberg, MD | ["Doctors","Health & Medical"] |
| Pine Cone Restaurant | ["Restaurants"] |
| Deforest Family Restaurant | ["American (Traditional)","Restaurants"] |
+------------+------------+

> SELECT name, FLATTEN(categories) AS categories
  FROM dfs.yelp.`business.json` LIMIT 5;
+------------+------------+
|    name    | categories |
+------------+------------+
| Eric Goldberg, MD | Doctors    |
| Eric Goldberg, MD | Health & Medical |
| Pine Cone Restaurant | Restaurants |
| Deforest Family Restaurant | American (Traditional) |
| Deforest Family Restaurant | Restaurants |
+------------+------------+
```

Dynamically flatten repeated and nested data elements as part of SQL queries. No ETL necessary

Extensions to ANSI SQL to work with repeated values

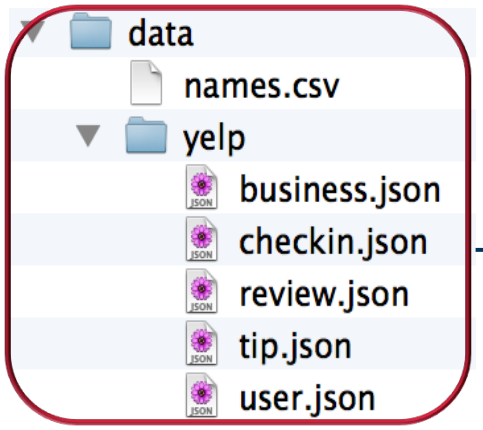
```
// Get most common business categories

>SELECT category, count(*) AS categorycount
  FROM (SELECT name, FLATTEN(categories) AS category
        FROM dfs.yelp.`business.json`) c
  GROUP BY category ORDER BY categorycount DESC;

+------------+------------+
|  category  | categorycount|
+------------+------------+
| Restaurants | 14303      |
…
| Australian | 1          |
| Boat Dealers | 1          |
| Firewood   | 1          |
+------------+------------+
```

Extensions to ANSI SQL to work with embedded JSON

```
-- embedded JSON value inside column donutjson inside column-
family cf1  of an hbase table donuts
SELECT
   d.name, COUNT(d.fillings)
FROM (
   SELECT convert_from(cf1.donutjson, JSON) as d
   FROM hbase.donuts);
```

Working with Dynamic Columns

checkin.json




```
check-in
{
     'type': 'checkin',
    'business_id': (encrypted business id),
    'checkin_info': {
        '0-0': (number of checkins from 00:00 to 01:00 on all Sundays),
        '1-0': (number of checkins from 01:00 to 02:00 on all Sundays),
        ...
        '14-4': (number of checkins from 14:00 to 15:00 on all Thursdays),
        …
        '23-6': (number of checkins from 23:00 to 00:00 on all Saturdays)
    }, # if there was no checkin for a hour-day block it will not be in the
dataset
}
```

Makes it easy to work with dynamic/unknown columns

```
> jdbc:drill:zk=local> SELECT KVGEN(checkin_info) checkins
   FROM dfs.yelp.`checkin.json` LIMIT 1;
+------------+
|  checkins  |
+------------+
| [{"key":"3-4","value":1},{"key":"13-5","value":1},{"key":"6-6","value":1},{"key":"14-5","value":1},
{"key":"14-6","value":1},{"key":"14-2","value":1},{"key":"14-3","value":1},{"key":"19-0","value":1},
{"key":"11-5","value":1},{"key":"13-2","value":1},{"key":"11-6","value":2},{"key":"11-3","value":1},
{"key":"12-6","value":1},{"key":"6-5","value":1},{"key":"5-5","value":1},{"key":"9-2","value":1},
{"key":"9-5","value":1},{"key":"9-6","value":1},{"key":"5-2","value":1},{"key":"7-6","value":1},
{"key":"7-5","value":1},{"key":"7-4","value":1},{"key":"17-5","value":1},{"key":"8-5","value":1},
{"key":"10-2","value":1},{"key":"10-5","value":1},{"key":"10-6","value":1}] |
+------------+
```

Convert Map with a wide set of dynamic columns into an array of key-value pairs

```
> jdbc:drill:zk=local> SELECT FLATTEN(KVGEN(checkin_info))
checkins FROM dfs.yelp.`checkin.json` limit 6;

+------------+
|  checkins  |
+------------+
| {"key":"3-4","value":1} |
| {"key":"13-5","value":1} |
| {"key":"6-6","value":1} |
| {"key":"14-5","value":1} |
| {"key":"14-6","value":1} |
| {"key":"14-2","value":1} |
+------------+
```

Makes it easy to work with dynamic/unknown columns

```
// Count total number of checkins on Sunday midnight

➢  jdbc:drill:zk=local> SELECT SUM(checkintbl.checkins.`value`) as
   SundayMidnightCheckins FROM
   (SELECT FLATTEN(KVGEN(checkin_info)) checkins
    FROM dfs.yelp.checkin.json`) checkintbl
    WHERE checkintbl.checkins.key='23-0';

+-----------------------+
| SundayMidnightCheckins |
+-----------------------+
| 8575                  |
+-----------------------+
```

Combining Drill & Spark together

Using Drill as an input to Spark Jobs

```
object DosDemo extends App
{

      val conf = new SparkConf().setAppName("DrillSql").setMaster("local")

      val sc = new SparkContext(conf)


      def queryOne(): Unit =
      {
      val sql = "SELECT name, full_address, hours, categories FROM `business.json` where
      repeated_contains(categories,'Restaurants') "

      val result = sc.drillRDD(sql)

      val formatted = result.map { r =>

      val (name, address, open, close) = (r.name, r.full_address, r.hours.friday.open, r.hours.friday,close)

      s"$name$address$open$close"      }

      formatted.foreach(println)

      }
```

Create RDDs from Drill queries

Access nested data elements intuitively

Query Spark RDDs via Drill

```
def queryTwo(): Unit = {

      var easy = sc.parallelize(0 until 10)

      val squared = easy.map { i =>

      CObject(("index", i), ("value", i*i))
      }

      sc.registerAsTable("squared", squared)

      val sql = "select * from spark.squared where index>3 and index<10 limit 5"

      val result = sc.drillRDD(sql)

      result.foreach(println)

      }

}
```

Construct complex objects as part of the Spark pipelines

Operate on the complex data directly using Drill queries

Combine benefits of Drill & Spark

- Flexibility (schema-discovery, complex data, UDFs)
- Rich storage plugin support
- Columnar execution
- ANSI SQL

- Rapid application development
- Advanced analytic workflows
- Efficient batch processing

Status and next steps

- Functional integration complete
- Beta release coming soon
- Next steps
  - Improve memory efficiencies
  - Improve serialization efficiencies
  - Improve task and data locality
- Ask questions and/or contribute
  - user@drill.apache.org
  - dev@drill.apache.org

Recommendations On Trying and Using Drill

New to Drill?

- Get started with Free MapR On Demand training
- Test Drive Drill on cloud with AWS
- Learn how to use Drill with Hadoop using MapR sandbox

Ready to play with your data?

- Try out Apache Drill in 10 mins guide on your desktop
- Download Drill for your cluster and start exploration
- Comprehensive tutorials and documentation available