IBM Technical Summit 2013

Agility in a Relational Database World - Dynamic Schema with JSON and DB2

Presented By : Mario Briggs
Slide courtesy : Bill Bireley
IBM Information Management
mabriggs@in.ibm.com

Stay ahead.

Please note the following

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

- New Era Applications
- NoSQL and JSON Data Store Overview
- Details for DB2 as a JSON Data Store
- Positioning JSON Data Stores in the Enterprise

A New Era of Engaging Applications…

From transactions to <u>interactions</u>
Complement systems of record with <u>systems of engagement</u>

…Needs New Capabilities…

Add a "call a taxi when you land" service provided by a partner cab company in days, not weeks!

1. **Add new offerings from business partners rapidly, and enhance applications quickly in response to user feedback**

Add a list of alternative flights, when displaying delays, in a day!

A hurricane causes flight cancellations, and a surge in usage of web site and apps

2. **Handle dramatically increased and widely varying load, driven by expanded stakeholder engagement and mobile access**

Book an alternative flight, only if hotel is available

Live chat with agent when alternative flight needs to be booked due to delay. Instant restaurant e-coupon delivery from business partner.

3. **Handle transactions across multiple interactions reliably, and provide secure querying and reporting capabilities to the business and partners**

Which day did coupons provide most revenue? (requires joins across JSON and Relational data) What discount did we offer on that day? (requires time travel query)

…Leading to New Technology Requirements

1. **Schema Flexibility & Developer Agility**

2. **Consistent Low Latency & Scalability**

3. **With Security, Transactions, Joins, and Operational Tooling**

Traditional relational model is not Agile.
Need schema flexibility to support rapid iterative changes.
Process involved to change relational schema and app code around it is slow

Comparing Relational VS NoSQL JSON for evolving applications

- Relational
  - Database Object definition changes
  - Develop Migration Scripts
  - Change Data objects in code
  - Change ORM Mapping
  - Update Data Access layer code
  - Update Service Interface code
  - Update UI code

- NoSQL JSON store
  - Update UI code

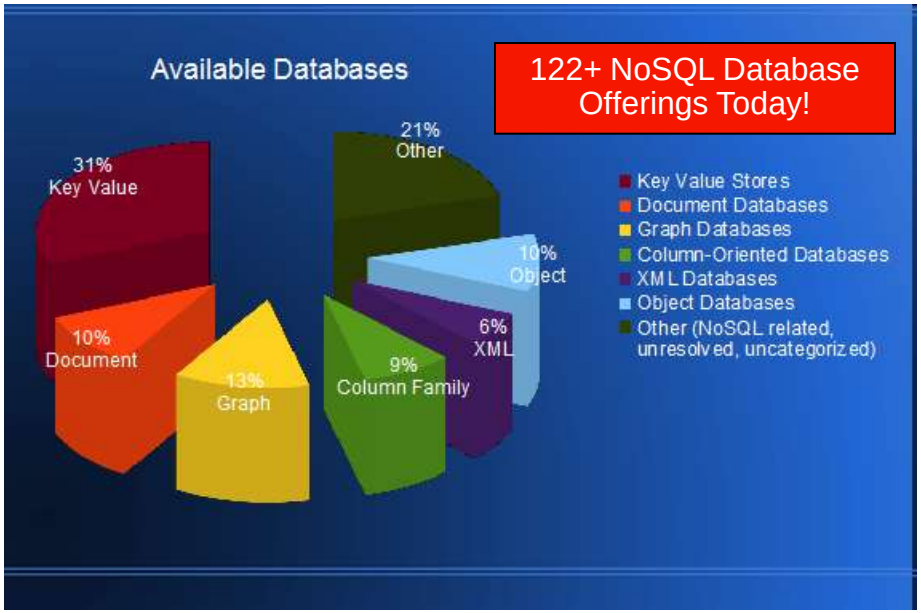
JSON : Javascript Object Notation.

Object representation format of javascript, the UI Dev langauge.

Lightweight, flexible.

Eliminates mapping and transformation code in application if database can store JSON natively

What is NoSQL?



Motivation

- Many apps need fewer database features (simplicity)
- Need rapid application evolution/deployment, with minimal interaction with DBA
- Some apps need extremely high scale (e.g. Twitter)
- Need for a low-latency, low-overhead API to access data
- Increasing use of distributed analytics

Dominant Flavors

- **Key Value Stores**
  - Hash table of keys, where the data part of key-value is in a binary object
  - Examples pure key-value stores: MemcacheD, REDIS, WebSphere eXtreme Scale
- **Document Stores**
  - Stores documents made up of tagged elements, which have keys and document-like objects
  - Examples. MongoDB, couchDB
- **Column Family**
  - Each storage block contains data from only one column/column set
  - Examples. Hbase, Cassandra
- **Graph Store**
  - Key-values are related through graph structure
  - Common Model : *RDF*
  - Examples : *Jena, Sesame*

Quick RDF/Linked Data/Semantic web Introduction

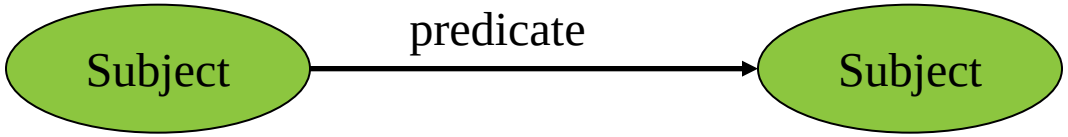
- **Problem Statement :**

  Lots of datasets exists on the web, but requires a human to manually link them to get integrated value.

  Is there a way machines can do it without human intervention ?

- **Semantic Web solution**

  – A single universal schema (RDF) which allows machines to link datasets



  – a common query language for the data model (SPARQL)

  – an knowledge representation language (OWL) using which machines can do deductive reasoning on the data.

- **Same architecture being adopted by Applications for data integration**

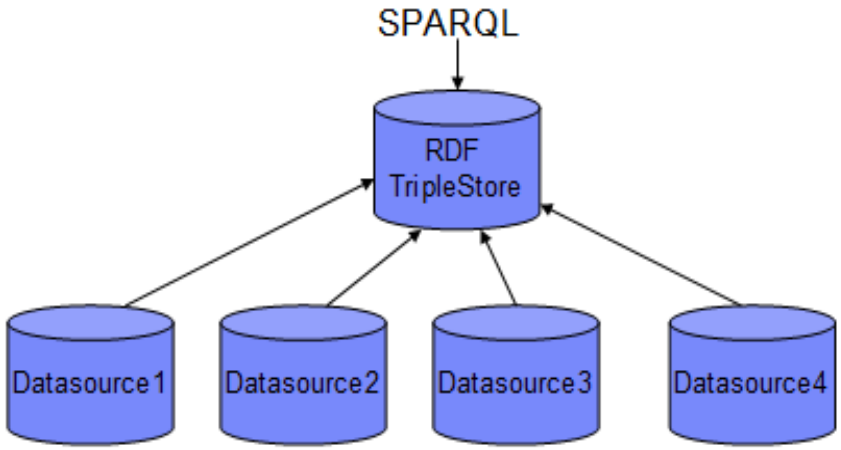
RDF Use-case

- Data Integration

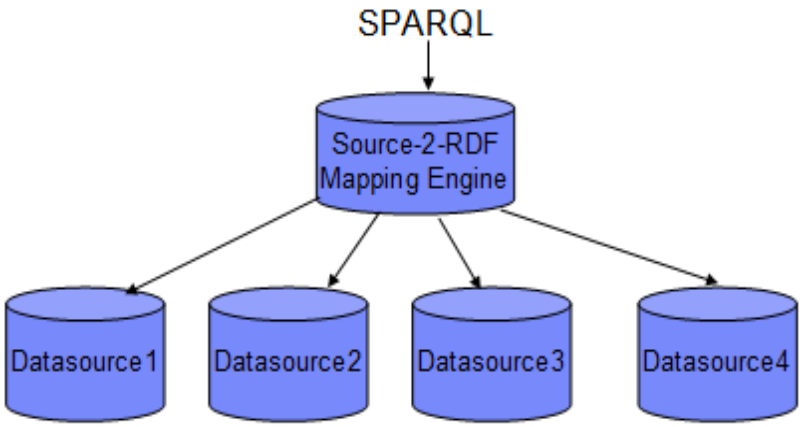
  Integrate multiple independently developed, evolving Applications and data schemas.

  Benefit : More flexible than brittle point-2-point API integrations.

- Two possible architectures



Move the Data



Keep the data where it is

DB2RDF Features

- Released in DB2 10.1
  - Supported SPARQL 1.0 and Subquery / Aggregates from SPARQL 1.1
  - Supported FGAC with RDF/SPARQL

- In DB2 10.1 FP2
  - SPARQL 1.1 (minus Property Paths, Negation, BIND)
  - SPARQL 1.1 UPDATE
  - SPARQL 1.1 GRAPH STORE HTTP PROTOCOL
  - Support for querying versioned RDF Graphs
  - Number of performance enhancements

    SPARQL-2-SQL Cache

    Single recursive SQL for Describe Queries

    Streaming bulk loaders

- In DB2 10.5
  - Support for SPARQL 1.1 Property Paths.

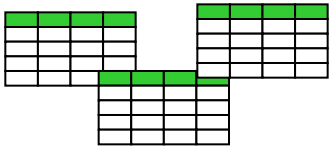
Relational Approaches to Schema Flexibility

Two Significant Trends in Data Representation and Storage

– Both driven by the Web

- Both enabling new applications of data

Relational Tables



3rd Normal Form

Variant on row based stores is column based stores

(1) De-normalized or Not-normalized

Intact Data :

LOBs, XML, JSON, Documents etc

(2) Highly Normalized

RDF (Resource Definition Framework) Triples and Ontologies

See "Data Normalization Reconsidered" –

http://www.ibm.com/developerworks/data/library/techarticle/dm-1112normalization/

http://www.ibm.com/developerworks/data/library/techarticle/dm-1201normalizationpart2/

Why is JSON Important?

- Helps enable a new era of applications
  - Mobile, Cloud, Social
- JavaScript everywhere
  - Data interchange format for JavaScript
- JSON becoming the language of the web
- **JSON support in the database tier**
  - **Schema Flexibility -> Development Agility**
  - **Eliminate mapping and data transformation through the tiers**
  - **Becoming predominant technology leveraged by NoSQL document stores**

"Less is better: less we need to agree upon to interoperate, the more easily we interoperate"

JavaScript: The Good Parts, O'Reilly

New Era Application Characteristics

- Applications evolve rapidly as the needs for mobile and Web presence try to keep pace with internet user needs

- Application developers are increasingly looking for solutions that allow nearly continuous integration of application changes
  - Amazon.com allows 1000's of their developers to check in product code changes daily…
  - Developers resist solutions that require delays to sync up with DBA change windows

- NoSQL JSON stores are appealing to these developers:
  - JSON schema can be evolved rapidly without intervention by DBAs or data modelers.
  - Objects like "shopping cart" in these applications really aren't used outside the Web application, so there is no need to interlock closely with the rest of the enterprise data model.
  - JSON offers a very simple and elegant model for persisting Java or JavaScript objects, without needing a heavy-weight persistence solution like OpenJPA or Hibernate.

- Performance and scalability is very good for JSON

  Store a single JSON document representing the object

  versus

  Store "n" rows in relational as a "normalized" object.

Data access example using Javascript and JSON

- Relational representation

Lastname	Firstname	Street
Jones	Billy	123 Maple Drive

- JSON representation

JSON_string = ‘{“Lastname”:”Jones”,”Firstname”:”Billy”,”Street”:”123 Maple Drive”}’;

- Javascript data access

```
var JSONelems = JSON.parse(JSON_string);
l_name = JSONelems.Lastname;
f_name = JSONelems.Firstname;
l_street = JSONelems.Street;
```

Simple Database API for JSON

Insert a record, a blog post by Joe:
db.posts.insert({author:"Joe", date:"2012-04-20", post:"..."})

Find all posts by Joe:
db.posts.find({author:"Joe"})

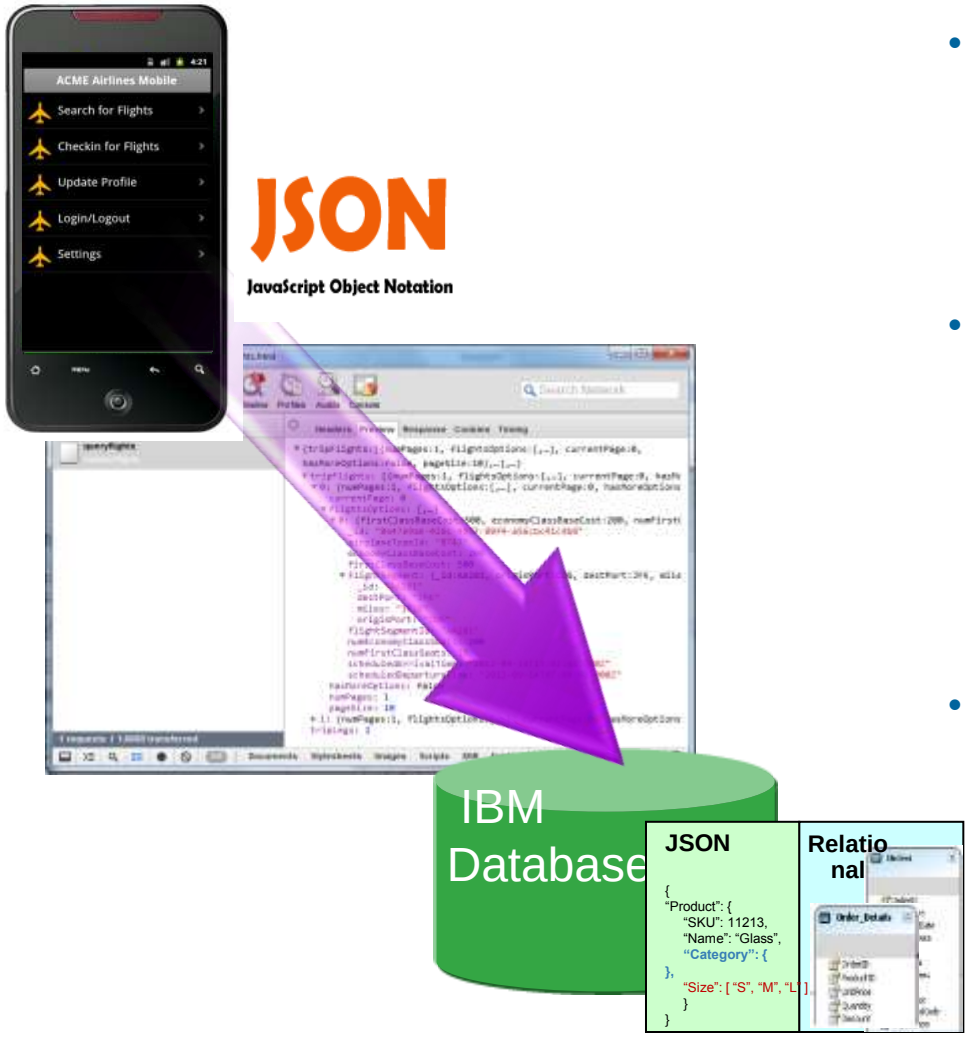
Delete all posts of Joe:
db.posts.remove({author:"Joe"})

Typical JSON Open Source Datastore Attributes

- Logging is often turned off to improve performance
- By default, no return code on insert (a.k.a. “fire and forget”)
  - App must verify update was performed
- Data is sharded for scalability
- Shards are replicated asynchronously for availability
  - Queries to replica nodes can return back-level data sometimes…
- No concept of commit or rollback
  - Each JSON update is independent
  - Applications have to implement compensation logic to update multiple documents with ACID properties
- JSON documents are stored in collections
  - But no “join” across collections
- No document-level locking
  - App must manage a “revision” tag to detect document update conflicts
- No document-level or tag-level security
- No built-in temporal or geo-spatial query support

IBM NoSQL : Delivering the Best of Both Worlds

JSON Agility with a Trusted Foundation



- **Interoperate seamlessly with modern applications**
  - Flexible schemas allow rapid delivery of applications
- **Preserve traditional DBMS Capabilities, leverage existing skills and tools:**
  - Multi-statement Transactions
  - Management/Operations
  - Security
  - Scale, performance,high availability
- **Extend with Advanced features (future)**
  - Temporal semantics
  - Full Text search
  - Multi-collection joins
  - Combine with Enterprise RDBMS data

JSON API Details

address New Era Application Development

- **IBM DB2 10.5 FP1 debuted a JSON Document Store API consisting of the following contents:**

  - **IBM provided Java Driver for JSON API**
    - Java Driver supporting JSON API for data access layer
    - Transactions
    - Parametric SQL statements (Delete, select)
    - Temporal tables

  - **CLP-Like Command Shell**
    - Ad-hoc updates / queries
    - Administration commands

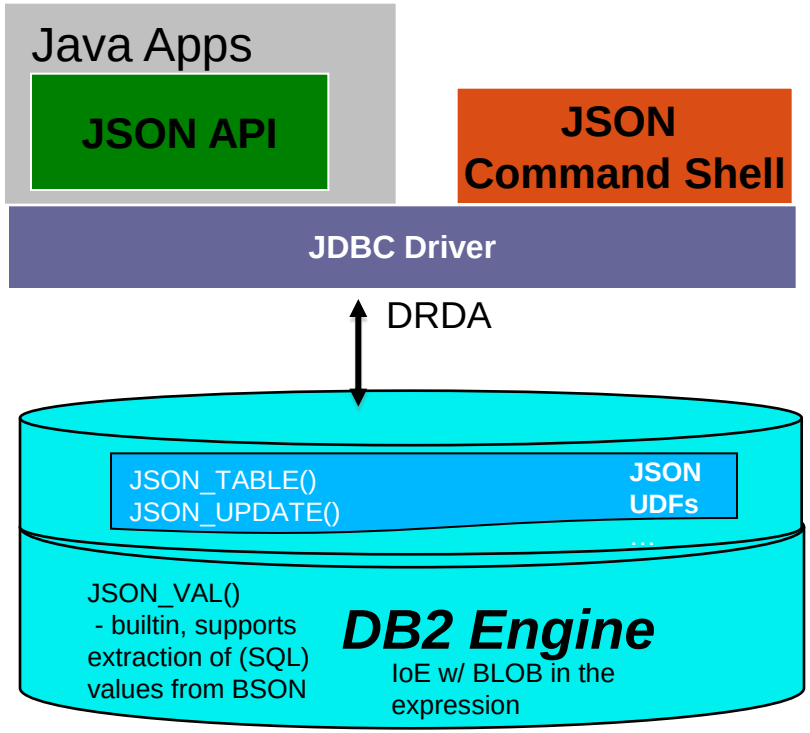
  - **Open Source Driver Wire Listener**
    - Leverage NoSQL community drivers for data access layer

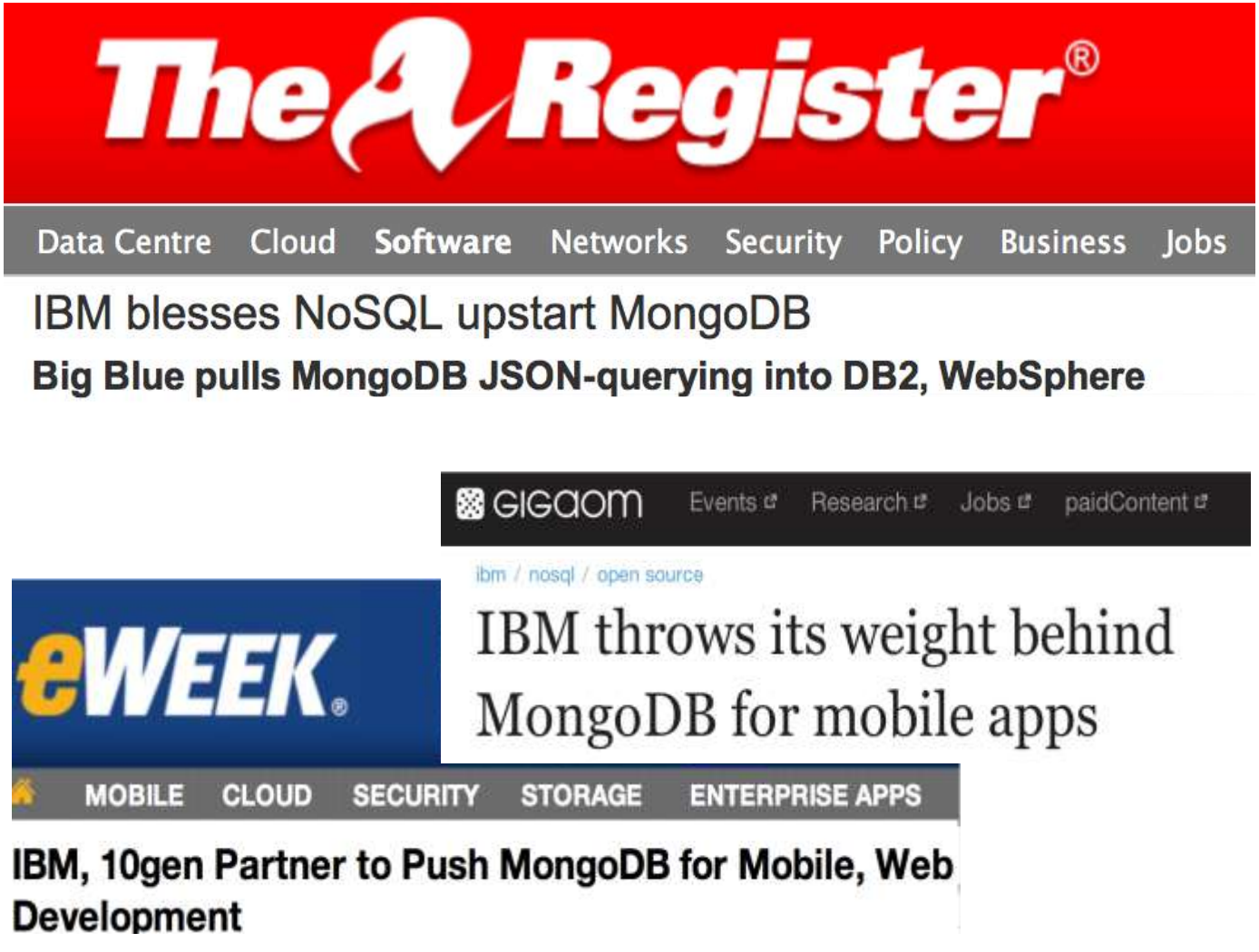
  - **DB2 enablement:**
    - Index on Expression
      - allows indexing of JSON document fields
    - Scalar function and UDF extensions
      - Start to form the base for official SQL/JSON support

DB2 JSON API

Java Driver *that translates API calls to SQL + function invocations*

- Supports Transactions
- Batches insertions
- Fire-forget inserts (fast)
- Indexing
- Time travel query
- Smart Query re-write
- Good performance with Inline LOBS
- Java command line





PROFIT.CA /READER

IBM throws its weight behind MongoDB for mobile apps

NoSQL JSON API and equivalent SQL

1) Create a customer collection / table.

```
db.createCollection("customers")
```

```
CREATE TABLE customers (_id VARBIN(12) data BLOB(16M))
```

2) Insert all your customers as JSON documents. For example, one insert might contain this document:

```
{ name:"Joe", age:25, phone:["555-666-7777", "444-789-1234"],
        address: {  street:"ABC st",
                    zipcode:"95141" } }
```

```
db.customers.insert({name:"Joe"…)
```

```
INSERT INTO customers
       VALUES ( <binary JSON> )
```

3) Look for customers in zipcode 95141.

```
db.customers.find(
      {"address.zipcode":'95141'})
```

```
SELECT DATA FROM customers
WHERE JSON_VAL
       (json_data,'address.zipcode','s:5')
       ='95141'
```

4) Improve performance by creating index on zipcode.

```
db.customers.ensureIndex
   ({"address.zipcode"});
```

```
CREATE INDEX idx1
ON customers
 (JSON_VAL(json_data,'address.zipcode','s:5')
```

DB2 NoSQL/JSON API from Java

```
/*Set up Conn. and Database handle*/
Context ctx = new InitialContext();
DataSource ds =
   (DataSource)ctx.lookup("jdbc/myDB2");
Connection conn = ds.getConnection();

Database db = new Database(conn);
DBCollection shop =
  db.getCollection("shop");

/*Create JSON objects and insert*/
BasicDBObject cart = new
  BasicDBObject();
BasicDBObject amtDue = new
  BasicDBObject();

cart.put("sid", "176");
cart.put("customer", "Bill")';
amtDue.put("subtotal", 50.07);
amtDue.put("tax", 4.26);
amtDue.put("total", 54.33)
cart.put("amtDue", amtDue);
shop.insert(cart);
```

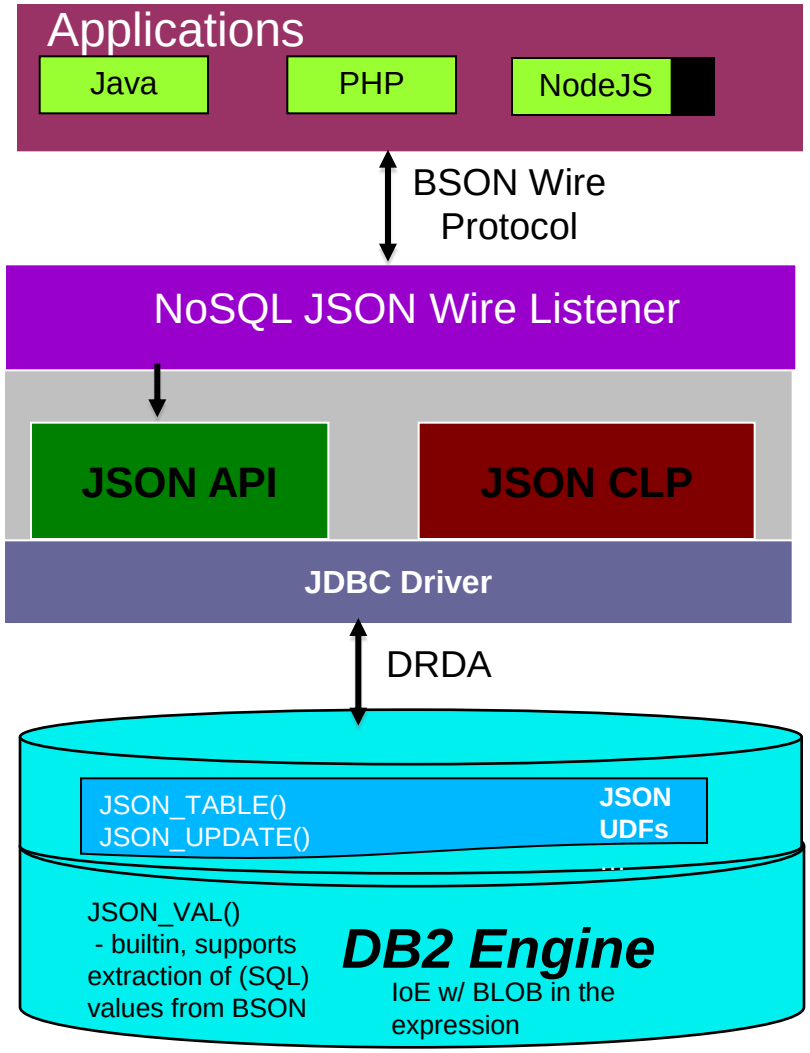
```
/* Use cursor to fetch back the JSON */

DBCursor cursor = shop.find(new
  BasicDBObject("customer", "Bill"));

try {
   while(cursor.hasNext()) {
      DBObject obj = cursor.next();
      doSomething(obj);
   }
} finally{
     cursor.close(); //close the cursor
  no matter what.
}
```

NoSQL JSON Wire Listener

- Built on JSON API
- Leverage community
- Immediate reach to more applications and developers
- Presence in “New style apps”
- (Future) Extend existing community drivers with DB specific features:
  - Mulit-statement commit scope
  - Temporal
  - Geo-spatial



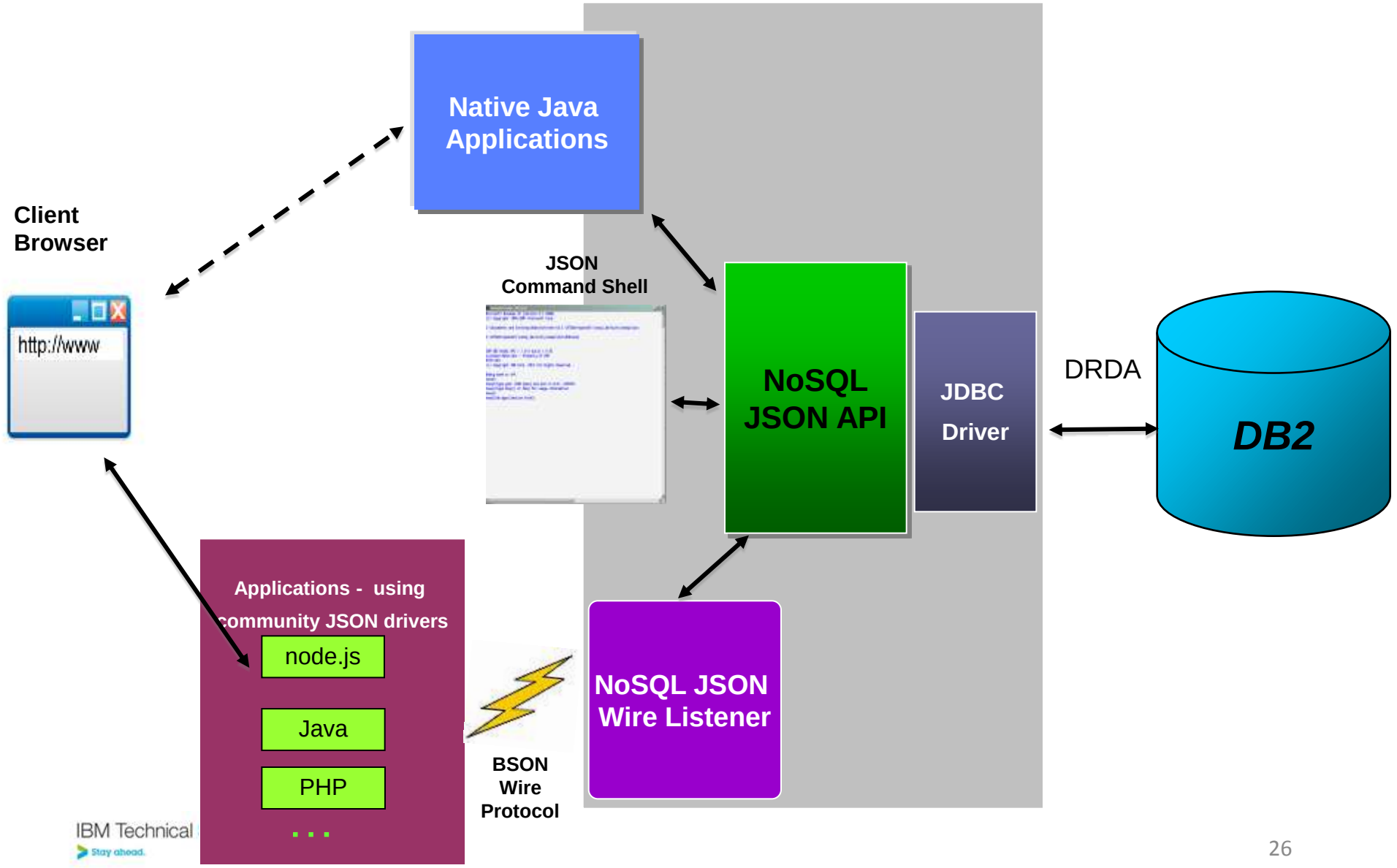
Node.js code sample

```
var databaseUrl = "shop"
var collections = ["cart"]
var db = require("mongojs").connect(databaseUrl, collections);

…
Db.users.save({sid: "176", customer: "shopper99@yahoo.com",
               amtDue: {subtotal: 50.07, tax: 4.26, total: 54.33}},
               function(err, saved) {
                 if (err || !saved )
                   console.log("cart not saved");
                 else
                   console.log("cart saved"); });

…
Db.cart.find({customer: "shopper99@yahoo.com"},
              function(err. carts) {
                if (err || !carts)
                  console.log("No carts found");
                else carts.forEach( function(iCart) {
                  console.log(iCart);});});
```

Technical Preview – Demo Components and Architecture



What's behind the API?

- **1 to 1 mapping between collection and DB2 table**
  - Table contains a BLOB column
  - Each row contains single document
  - (table name – collection name)
  - Side column for primary ID field

- **Possible additional side columns for optional features**
  - Bi-temporal (future)
  - Fine grained access control (future)

- **User-defined Functions, scalar function to operate on fields inside the JSON document**

Indexes

- **Simple Index**

```
db.collection.ensureIndex({sid:{1, "$int"}}); //create ascending integer index on 'sid'.
db.collection.ensureIndex({"customer":1}}); //create ascending varchar(50) (default type)
  index on 'customer' field.
```

- **Composite index containing multiple fields**

```
db.collection.ensureIndex({customer:[1, "$string", 20], total:{-1, "$int"}});
//create compound index with two fields: customer ascending with type varchar(20), and total
  descending as integer.
```

- **Index on nested object**

```
db.collection.ensureIndex({amtDue.total:{1, "$int"});
```

- **How does JSON field indexing work?**
  - Indexes are created on fields within the JSON document

```
CREATE INDEX CUSTNDX ON JSON_VAL(JSONBLOB, "customer", ":i")
```

  - Subsequent queries searching on customer will use same functional expression in a predicate

What is JSON's Role in the Enterprise?

- Flexible Schema is agile, liberating for application developers
- But will we abandon years of expertise in data modeling / normalization theory?
  - How to maintain control in an enterprise, mission critical DBMS?
- Identification of appropriate applications is critical
- Application deployment procedures need to adapt
  - New controls to prevent schema chaos
  - Application Development Groups need to implement controls
- When combining with application that uses relational schema
  - Identify portions that need to remain dynamic
  - Allocate / accommodate space for that as JSON
  - Future – combination of SQL and JSON will make this easier

"If I have seen further, it is by standing on the shoulders of giants"

- Sir Isaac Newton

What data store format makes sense for your application?

- **Consider NoSQL JSON when:**
  - Application and schema subject to frequent changes
  - Prototyping, early stages of application development
  - De-normalized data has advantages

    Entity / document is in the form you want to save
    - Read efficiency – return in one fetch without sorting, grouping or ORM mapping
  - "Systems of Engagement"

    Less stringent "CAP" requirements in favor of speed
    - Eventual consistency is good enough

    Social media
- **Relational still best suited when these are critical**
  - Data Normalization to

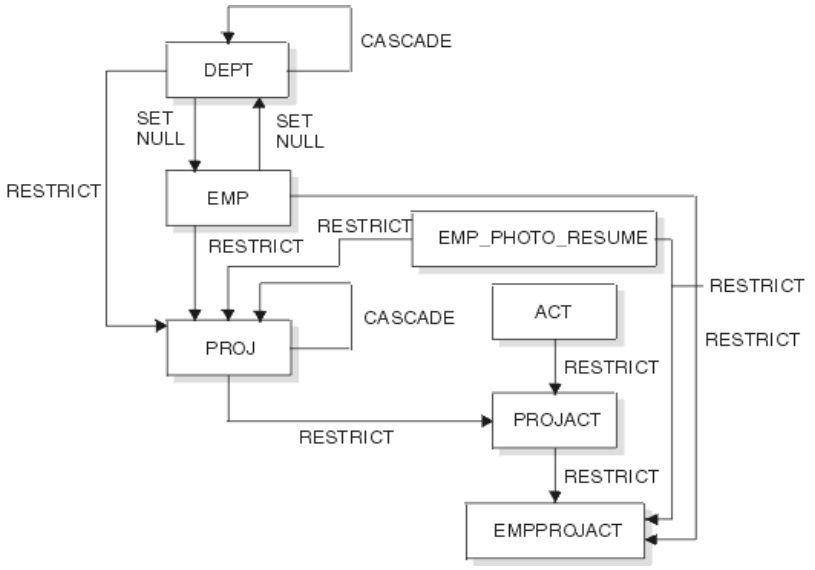
    Eliminate redundancy

    Ensure master data consistency
  - Database enforced constraints
  - Database-server JOINs on secondary indexes

Data Normalization - choose the right solution

Relational

Simple normalized schema (DB2 sample) with relational constraints:



NoSQL JSON - Two approaches:

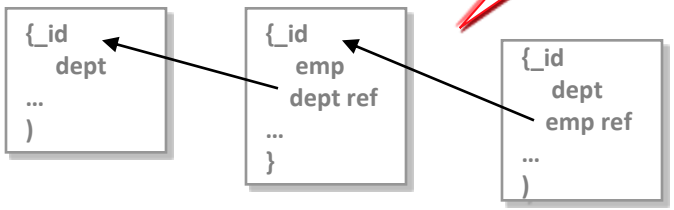
embedded (de-normalized)

```
{dept: "A10",
 deptname:"Shipping",
 manager:"Samuel",
 emp:[
        {empno:"000999",
          lastname:"Harrison",
          edlevel:"16"},
        {empno:"370001",
          lastname:"Davis",
          edlevel:"12"}
       ]
 proj:[
        {projno:"397",
          projname:"Site Renovation",
          respemp:"370001" },
         {projno:"397",
          projname:"Site Renovation",
          respemp:"370001"}  …
       ]
}
```

Chance for data redundancy

Requires application-side join

Using references



If you need normalization and database-enforced constraints, JSON may not be best choice

JSON use case – Inheritance of common fields

- Documents share a common structure but may have unique variations
- Example:
  - website stores product descriptions in single collection
  - All have product number, price, supplier, name, description
  - Different product types have unique fields
  - As new products are introduced they need no database schema change
- Common fields are indexed, others are queryable but not indexed

products

```
{prodnum:"CR2549",
name:"Gulliver's Travels",
type:book,
price:15.97,
description:"Classic novel",
supplier: "Penguin Group"

details : {author:"Jonathan Swift",
           categories:
                [adventure,
                travel,
                fantasy]
            publish_date: 1726
           }
}
```

```
{prodnum:"BA9444",
name:"Mahogany Desk",
type:furniture,
price:349.00,
description:"Small Writing Desk",
supplier: "Elegant Wood Designs"

details : {construction:"veneer",
           weight:80,
           units: pounds
           dimensions:
              {height:29,
               width:48,
                depth: 28,
                units:"inches"
              }
           }
}
```

JSON Data Store in DB2 10.5 FP1: Summary

Summary of Expected Features and Roadmap

MongoDB Wire Listener
- Leverage NoSQL community drivers for data access layer

IBM provided Java Driver for JSON API
- Java Driver supporting JSON API for data access layer
- Transactions
- Parametric SQL statements(Delete, select)
- Temporal tables

Insert, Update, Delete, Select support
- Select projection list
- Batching, order by, paging queries (API only)
- Fire and forget inserts
- Limited aggregate functions (group-by / having)

Indexing support in API
- Primary index and secondary single value index

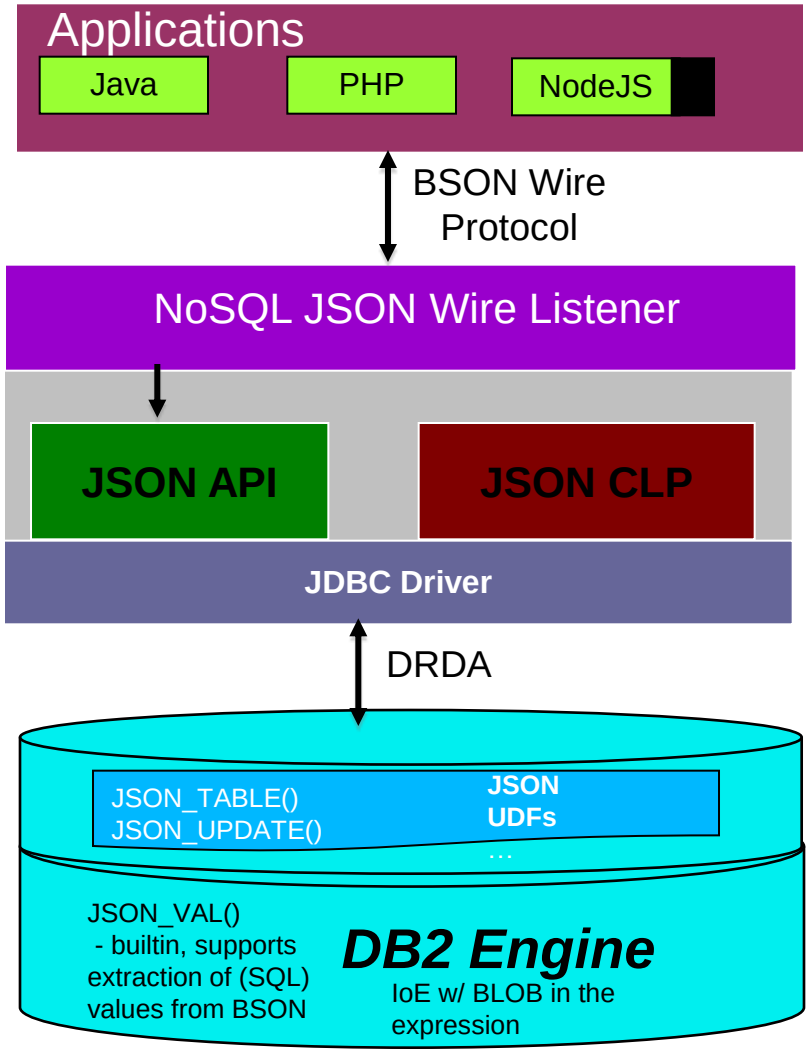
Import/Export
- Import/Export from/to MongoDB export JS-files

Command line tools
- Execute JSON queries and display results

Install
- Files and scripts that are part of server and DS Driver

DB2 Server Capabilities
- JSON_VAL Built-in function
- Index on Expression with BLOB input

Expected Platforms:
Tech Preview: Serial on Windows&RedHat
GA: Expand to all platforms



Questions

Innovate2013
The IBM Technical Summit

Acknowledgements and disclaimers

Availability: References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Thank You