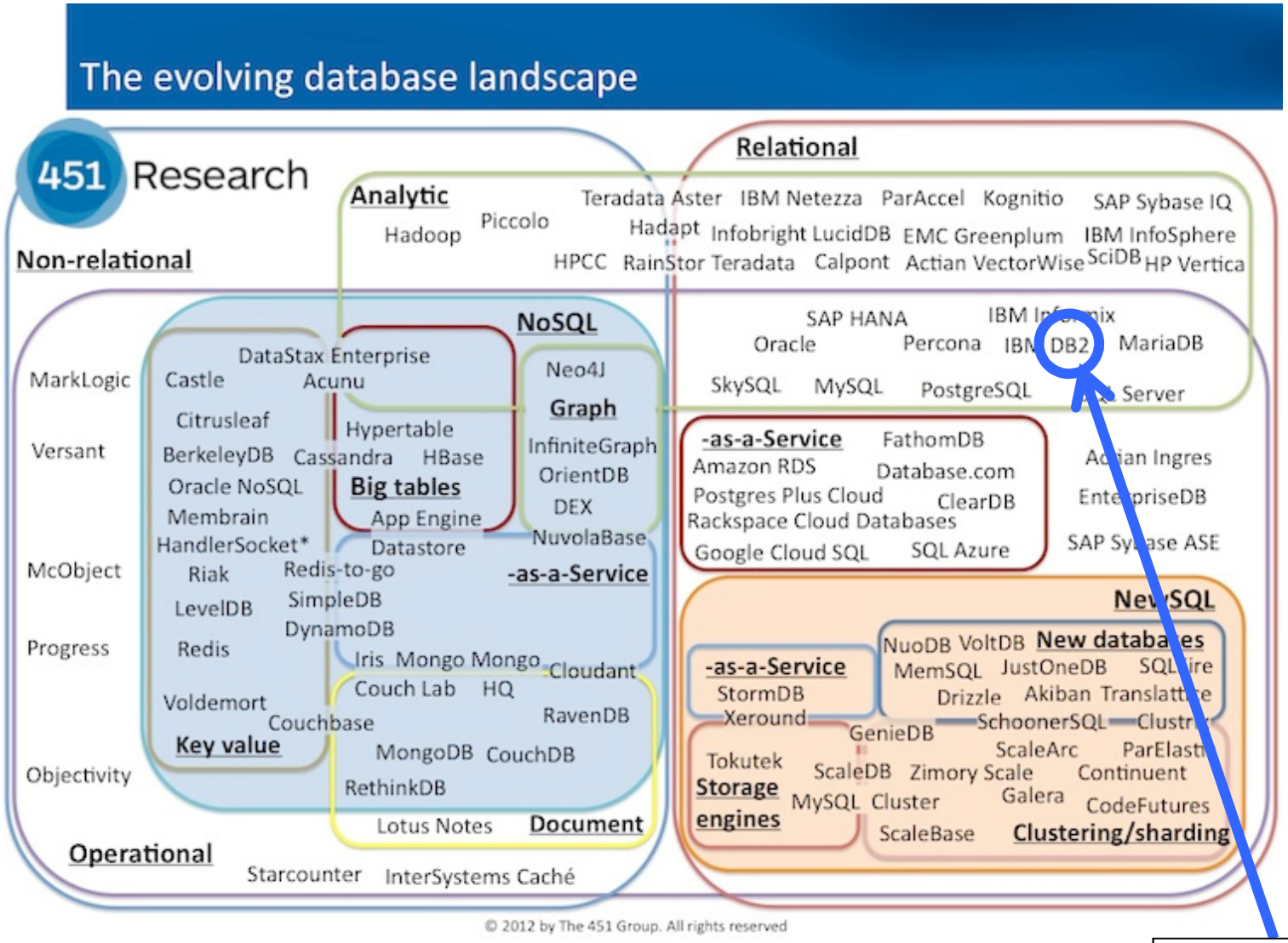
NoSQL – Is This The End Of SQL?

Chris Eaton
Worldwide Information Management Technical Specialist
IBM Toronto Lab
ceaton@ca.ibm.com

The Database Market and Choice



Agenda

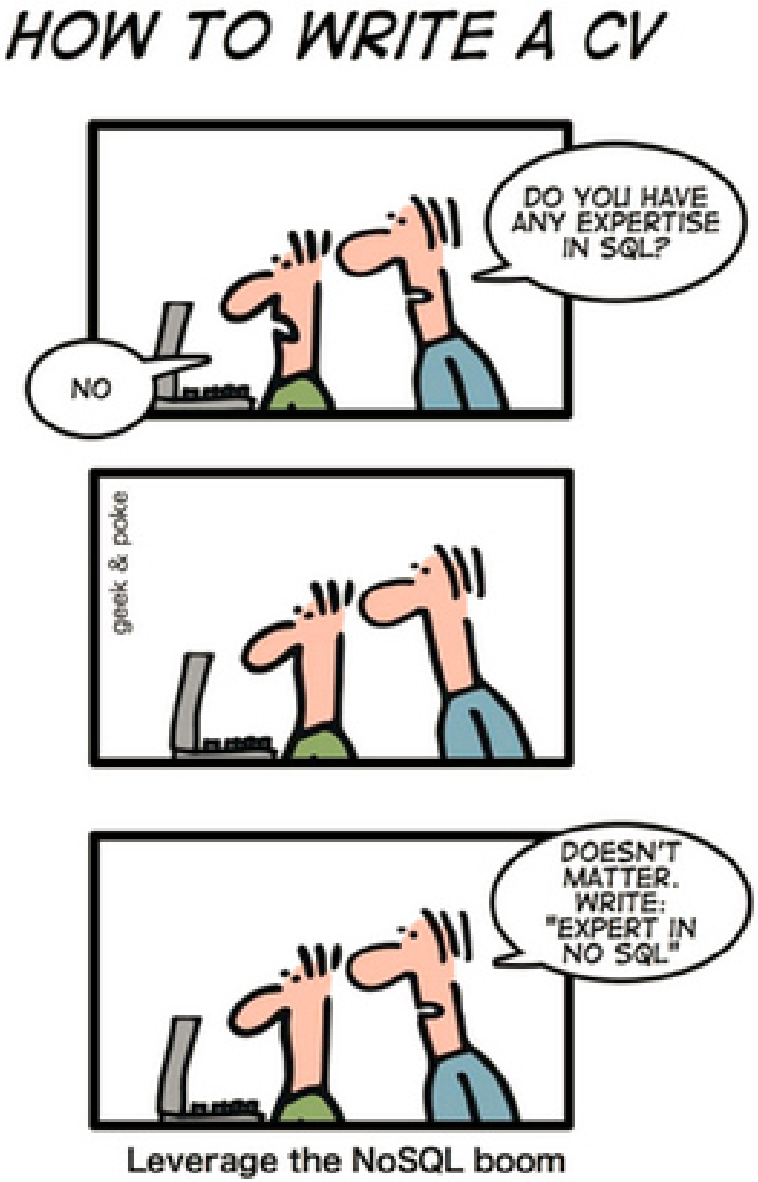
- **The NoSQL World**
- **What is JSON and Why Should I Care**
- **JSON in the Enterprise**
- **The Biggest Thing In the NoSQL World**
  - psst – it's your specialty

The NoSQL World

What is NoSQL?

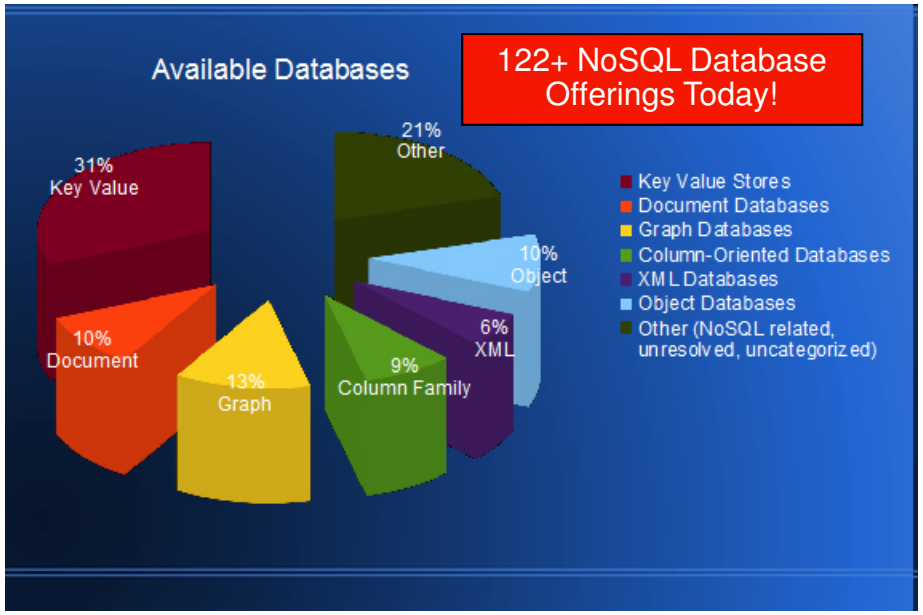
- **Over 122+ open source NoSQL databases at last count**
  - Four dominant ‘flavors’
  - Key|Value, Document, and Columnar focused on what folks familiar with RDBMSs do for BI and OLTP
  - Graph stores doesn’t do your typical RDBMS OLTP or BI, but it’s pretty pervasive because it has some powerful analytics and reasoning engine capabilities

- **Some have a fair amount of traction, most are on the decline and aren’t going anywhere (if there’s a 122 of something, they are not all going to survive)**



Leverage the NoSQL boom

What is NoSQL?



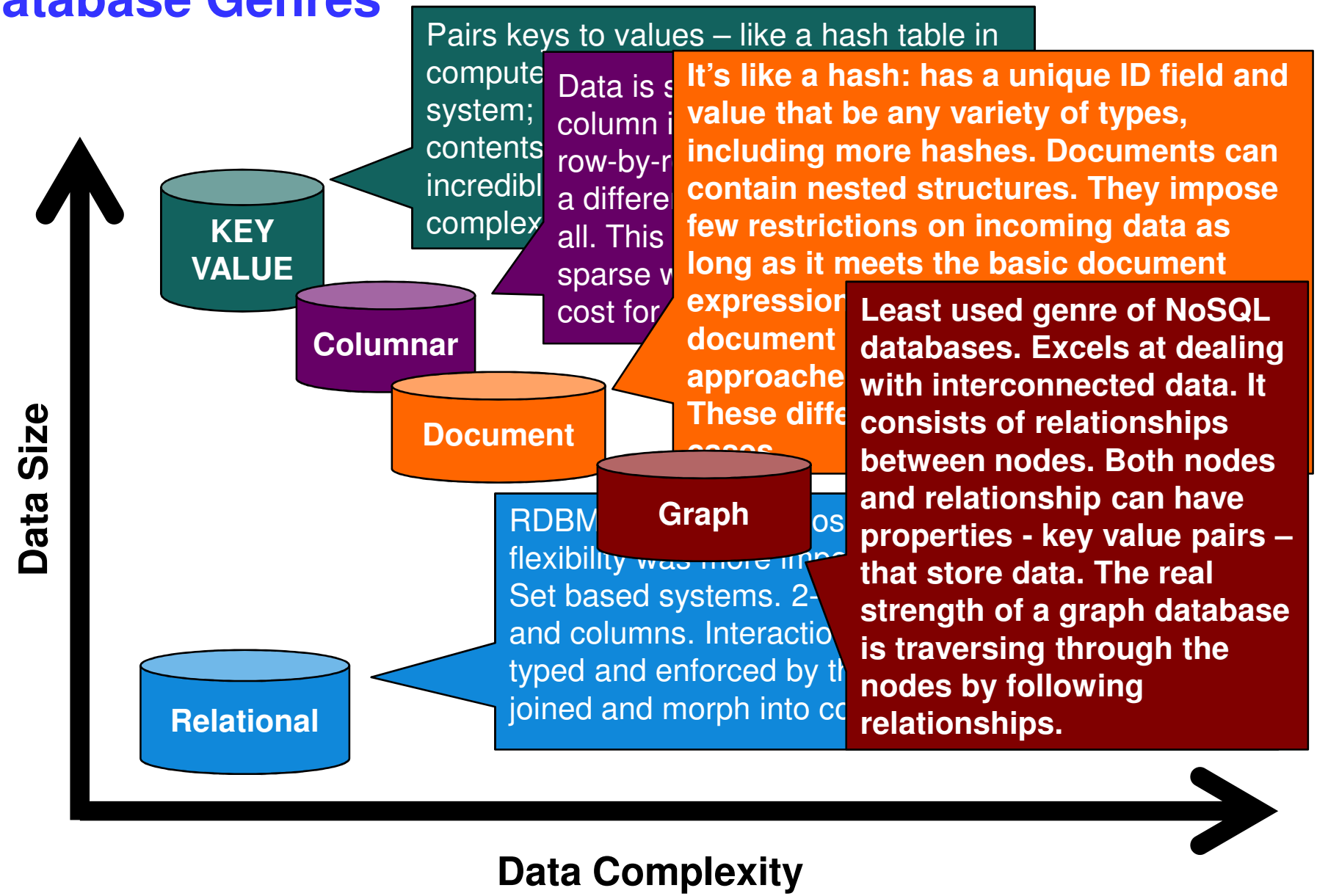
Motivation

- 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, Cloudent*
- **Column Family**
  - Each storage block contains data from only one column/column set
  - Examples : *HBase*, *Cassandra, BLU Acceleration*
- **Graph Store**
  - Key-values are related through graph structure
  - Common Model : *RDF*
  - Examples : *Jena, Sesame*, DB2 RDF store

Database Genres



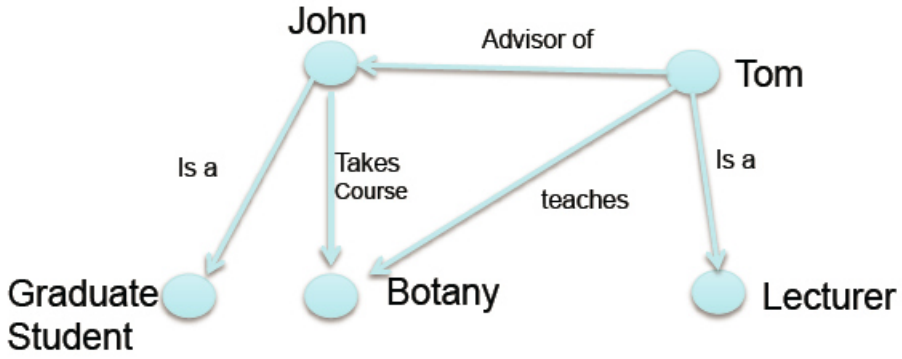
What is a Graph Store – RDF Example

- **RDF provides a general method to decompose any information into pieces called triples.**
  - Each triple is of the form ‘Subject’ , ‘Predicate’, ‘Object’.
    - Subject and Object are the names for 2 things in the world
    - Predicate, the relationship between them

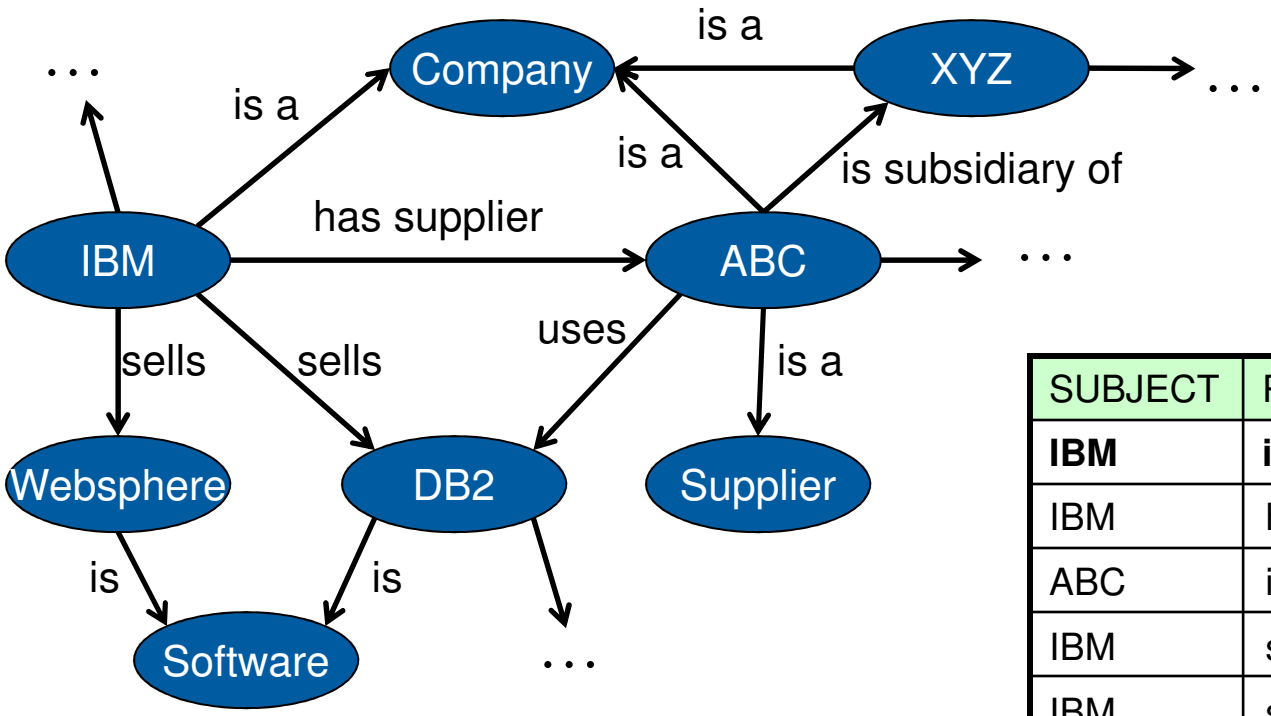
Subject	Predicate	Object
Tom	is a	Lecturer
Tom	teaches	Botany

  - Subject, Predicate, Object are given as URI’s
    - stand-ins for things in the real world
  - Object can additionally be raw text

In technical terms a labeled directed graph, where each edge is a triple



RDF Example

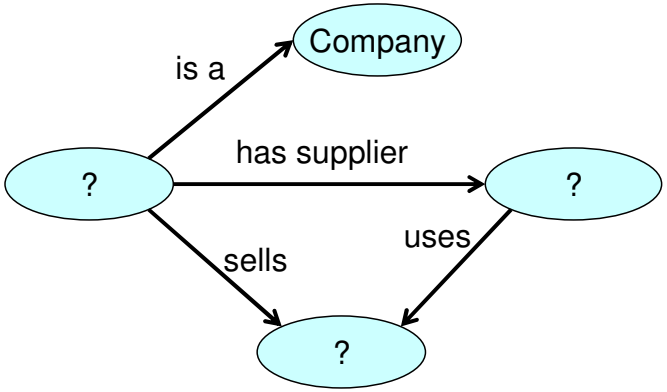


SUBJECT	PREDICATE	OBJECT
IBM	**is a**	**Company**
IBM	has supplier	ABC
ABC	is a	Company
IBM	sells	DB2
IBM	sells	Webshere
ABC	uses	DB2
ABC	is subsidiary of	XYZ
XYZ	is a	Company
…	…	…

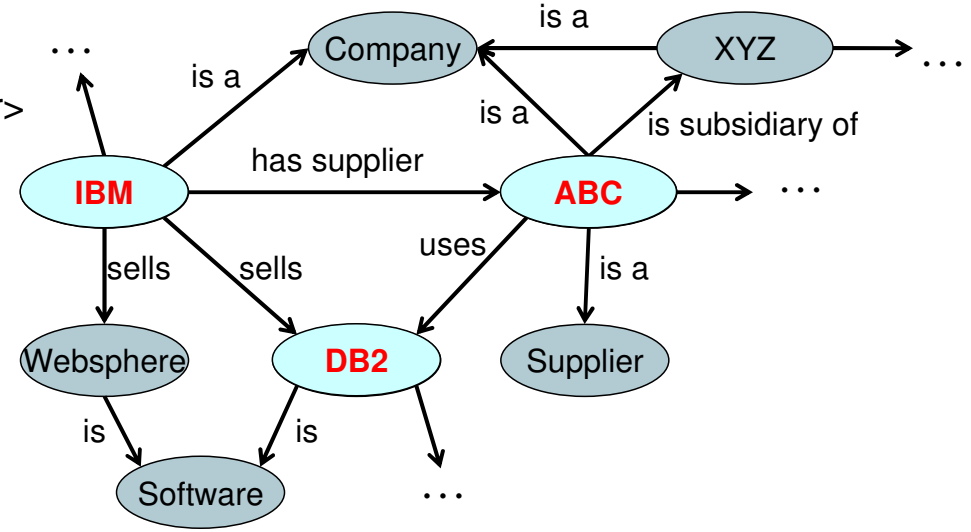
SPARQL: *SPARQL Protocol and RDF Query Language*

- **SPARQL : A subgraph pattern matching query language**

- **Example:**
  "*Find all companies that sell a product to a supplier*"




```
SELECT ?comp, ?product, ?supplier
WHERE  {        ?comp    <isA> <Company>
                ?comp    <sells> <?product>
                ?comp    <hasSupplier>  <?supplier>
                ?supplier <uses> <?product>
                   }
```



- **Result:**

?comp – IBM
?product – DB2
?supplier – ABC

New Era Application Characteristics

- **Today's applications evolve rapidly in a social-mobile-cloud world in order to keep pace with the Internet users they serve**

- **Developers want nearly continuous integration of app changes**
  - Performance to a DBA : SLA – how fast is it running
  - Performance to a Developer : How fast can I build my application

- **Developers 'resist' solutions that require delays to sync up with change windows**
  - NoSQL JSON stores appeal to these developers since they can evolve the app rapidly without DB or data modeler intervention
  - Objects like "Shopping Cart" aren't used outside Web app so why the need to interlock with the enterprise data mode

Session Store

Managing session information using relational technology has been a pain point for many Web application developers, especially as applications have grown in scale. In those cases, a global session store—i.e., one that manages session information for each user who visits the Website—is the right approach, and NoSQL has emerged as one of the best options for storing Web app session information. This is due in part to the key value storing properties of NoSQL databases: The unstructured nature of session data is easier to store in a schema-less document than in a structured (and more rigid) RDBMS record. In addition, low-latency access to session data is critical for ensuring a great user experience.

eWEEK

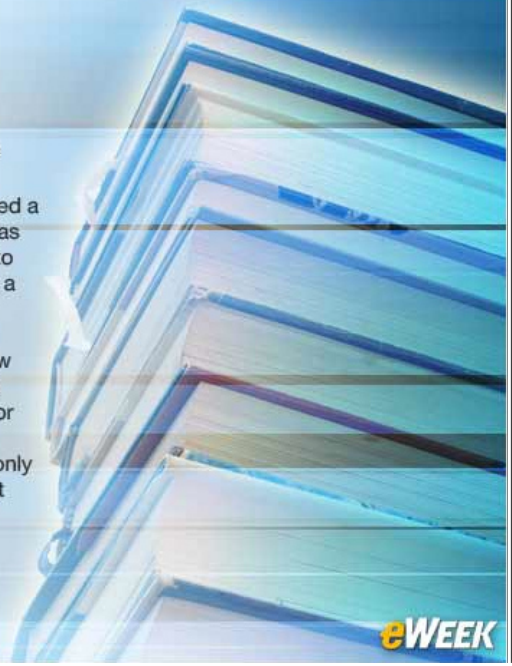
User Profile Store

All Web applications require user profiles and the ability to log in. A global user profile store is another example of where the key value characteristics of NoSQL come into play. A NoSQL database can store the user IDs, user preferences, multiple ID mappings and additional user information so that the app can quickly look up a user and authenticate access. Given the importance of this functionality to any Web app, the "always on" and scale-out characteristics of NoSQL are essential. TuneWiki recently drafted a blog post about how it uses NoSQL as a user profile store.

eWEEK

Content and Metadata Store

Companies such as McGraw-Hill need a place to store text-heavy data such as digital content, articles and ebooks to integrate different learning tools into a single platform. For content-driven applications, metadata is the most heavily accessed data that needs low response times. Using NoSQL—and particularly document databases—for building custom content-driven applications gives the flexibility not only to store a wide variety of content but also to provide fast access to it.



eWEEK

Mobile Applications

App developers' ability to update and enhance mobile apps—quickly and without service disruption—is critical to user adoption and loyalty. Because NoSQL databases can store user information and application content in a schema-less format, developers can quickly modify apps without major database infrastructure changes. That means users experience no interruption to application uptime. Some popular companies that take advantage of NoSQL for their mobile apps are Kobo and Playtika, both of which serve millions of users across the globe.

eWEEK

Characteristics of NoSQL: Data Model Flexibility

- **Amazon web site is made up of dozens of applications that appear a single web page**
- **1000s of developers work on these applications and all are empowered to make code changes without checks and balances**
  - No approval needed to integrate your code: just pass regression test and new code is auto-deployed across all servers running Amazon.com
    - Code first lightly deployed to a few servers: if all is well, aggressive deployment
    - After a few days, the 1000s of Amazon.com servers have completed changes
- **Amazon makes changes dozens to hundreds of times a day**
  - Don't have a freeze period during peak shopping periods such as Thanksgiving
  - On the night before Christmas when all through labs….
- **Compare approach to typical development shops (i.e. IBM)**
  - This approach not allowed
  - SVT and FVT check a developer's work, a decision board decides when the right time is to add the enhancements to the code base, +++

Wait a Minute! What About Stability?

- **Amazon.com experiences an average of 5 issues for every 1000 changes!**
  - This is a much higher success ratio than almost any development team I've come across

- **Many CIOs are looking at Amazon, Google, +++ and know they have much lighter volumes and HA requirements: leaves them wondering why can't they use this approach**
  - Can't really do this in RDBMS world since the continuous integration will likely involves a schema schema change
    - Need DBA to do the `ALTER` or `CREATE` table statement, built indexes, +++
    - DBA is going to want to wait for a change window

- **In NoSQL world, developers make their own code changes and can likely change schema without getting a DBA involved**
  - Developer simply covers 'all their bases'
    - Make sure the find all places in code with dependency on schema change and mix it into the code change for continuous integration
    - Check in immediate rules (48 hours) keep the code 'fresh' in mind

50% of Mobile Apps use JSON Data Store. What is JSON?

JSON and XML

- **JSON stands for JavaScript Object Notation**
  - JSON is lightweight text-data interchange format
  - JSON is language independent and object base (how programmers think)
    - JSON uses JavaScript syntax for describing data objects, but JSON is still language and platform independent
    - JSON parsers and JSON libraries exist for many different programming languages

- **Much Like XML**
  - JSON is plain text
  - JSON is "self-describing" and easy to understand (human readable)
  - JSON is hierarchical (values within values)
  - JSON can be parsed by JavaScript and transported using AJAX

- **Much Unlike XML**
  - No end tag
  - Shorter and more compact → Fewer bits to send over the wire in a mobile world
  - Quicker to read and write
  - Can be parsed using built-in JavaScript `eval()`
  - Uses arrays
  - No reserved words

Why is JSON Easier?

- **JSON, unlike XML, doesn't try to be a document markup language and a data exchange language**

- **For AJAX applications, JSON is faster and easier than XML and has an object feel**
  - JSON text format is syntactically identical to the code for JavaScript objects
  - Because of this similarity, instead of using a parser, a JavaScript program can use the built-in `eval()` function and execute JSON data to produce native JavaScript objects

- **Using XML**
  - Fetch an XML document
  - Use XML DOM to loop through the document
  - Extract values and store in variables

- **Using JSON**
  - Fetch a JSON string
  - `eval()` the JSON string

JSON is the Storage Notation

- **JSON syntax is a subset of the JavaScript object notation syntax:**
  - Data is in name/value pairs
    - A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value

```
"firstName" : "John"        → JSON
firstName = "John"          → JavaScript
```

  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

- **JSON values are simple – 6 choices**
  1. A number (integer or floating point)
  2. A string (in double quotes)
  3. A Boolean (true or false)
  4. An array (in square brackets)
  5. An object (in curly brackets)
  6. NULL

Tell Me Some About JSON

- **JSON objects are written inside curly brackets which can contain multiple name/value pairs**
  - As you can see, it's simple to understand

```
{ "firstName":"John" , "lastName":"Doe" }
```

- **JSON arrays are written inside square brackets and can contain multiple objects**
  - Here is the Employees array which contains three objects that represent a person with a first and last name

```
{
"employees": [
{ "firstName":"John" , "lastName":"Doe" },
{ "firstName":"Anna" , "lastName":"Smith" },
{ "firstName":"Peter" , "lastName":"Jones" }
]
}
```

Tell Me Some About JSON

- **Because JSON uses JavaScript syntax, you don't need extra software to work with JSON within JavaScript**
  - Also support on almost every other relevant programming language: node.js, PHP, Python, Ruby, C, C++, Perl, +++

- **JavaScript: create an array of objects and assign data on the fly**

```
var employees = [
{ "firstName":"John" , "lastName":"Doe" },
{ "firstName":"Anna" , "lastName":"Smith" },
{ "firstName":"Peter" , "lastName": "Jones" }
];
```

- **Access the first entry in the JavaScript object array**

```
Employees[0].lastName;
```

- **Modify the data**

```
employees[0].firstName = "Jonathan";
```

JSON Role in an Enterprise

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
  - No document-level locking
  - App must manage a "revision" tag to detect document update conflicts
  - 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 or tag-level security**
- **No built-in temporal or geo-spatial query support**

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? No…**
  - How to maintain control in an enterprise, mission critical DBMS?
- **Identification of appropriate applications is critical**
- **Schema controls moves to application development here**
  - 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

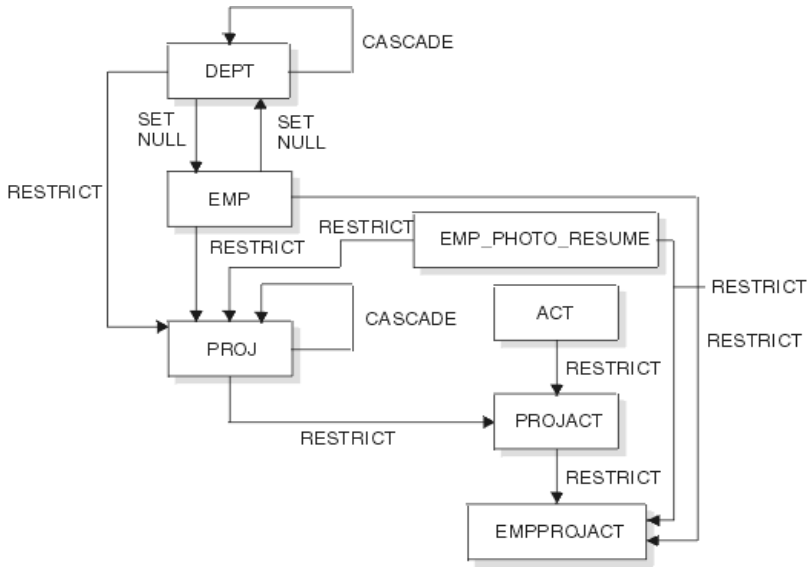
What Data Store Format Makes Sense For Your Application?

- **Consider NoSQL JSON when:**
  - Application and schema subject to frequent changes
  - Prototyping, early stages of application development where the schema is changing a lot and you're trying to figure out what the important fields are
  - 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" such as social media and where eventual consistency is fine
    - Less stringent "CAP" requirements in favor of speed
      - Consistency - all nodes see the same data at the same time
      - Availability - guarantee that every request receives a response on if it was successful or failed
      - Partition tolerance - system continues to operate despite arbitrary message loss or failure
    - Storing entries from social media doesn't have same requirements as transactional data
- **Relational still best suited when these are critical:**
  - Data Normalization to eliminate redundancy and ensure master data consistency
  - Database enforced constraints
  - Database-server JOINs on secondary indexes

Data Normalization - Choose the Right Solution

Relational

Very simple normalized schema (from DB2 SAMPLE database) with relational integrity constraints:



NoSQL JSON - Two approaches:

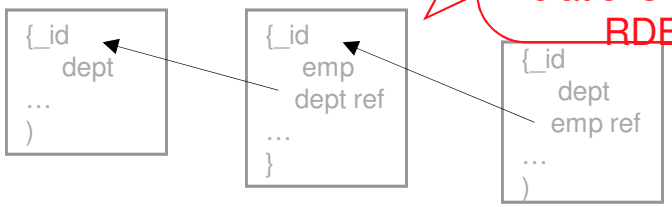
Embedded (de-normalized)

```
{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"}   …
      ]
}
```

Nest all employees in DEPT object creates data redundancy and increases opportunity of out of sync data

Requires application-side join which isn't very efficient for field link relationships like a RDBMS

Using references



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

JSON Use Case – Inheritance of Common Fields

- **Example: Online Store**
  - Sells wide variety of products
- **Documents share a common structure but may have unique variations**
  - This stores sells books and furniture which will have different details
- **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
  - Store both neatly in same collection
- **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
           }
}
```

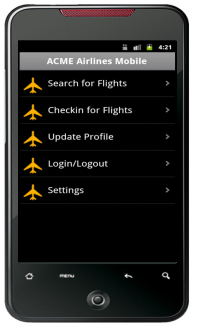
```
{prodnum:"BA9444",
name:"Mahogany Desk",
type:furniture,
price:349.00,
description:"Small Writing Desk",
supplier: "Elegant Wood Designs"

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

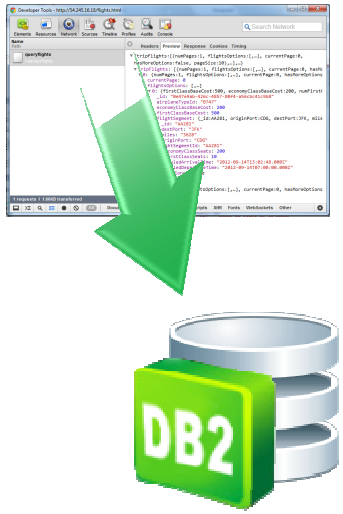
DB2 JSON Support: Agility With a Trusted Foundation

- **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 and high availability
  - Integrity

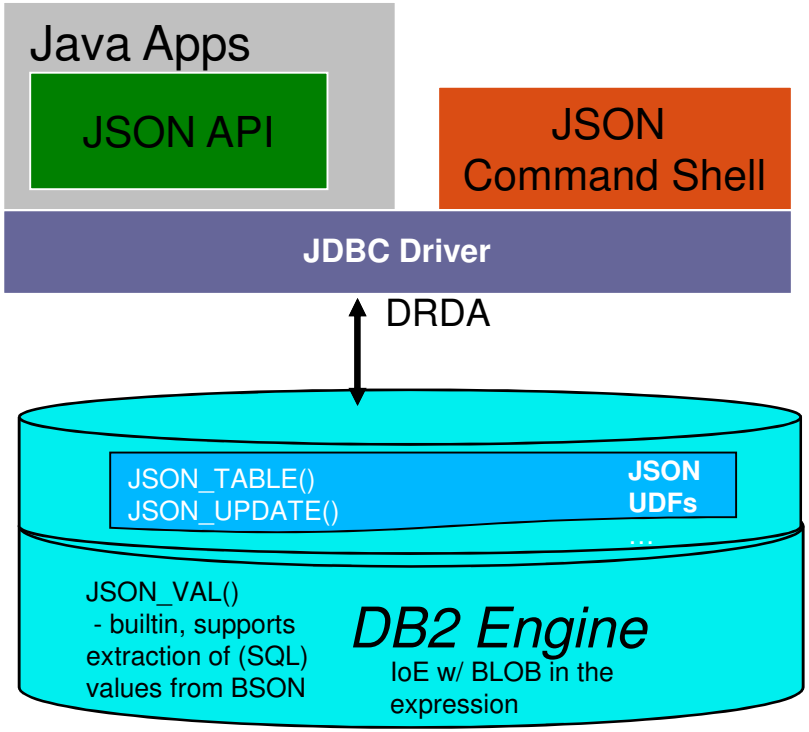


JSON
JavaScript Object Notation



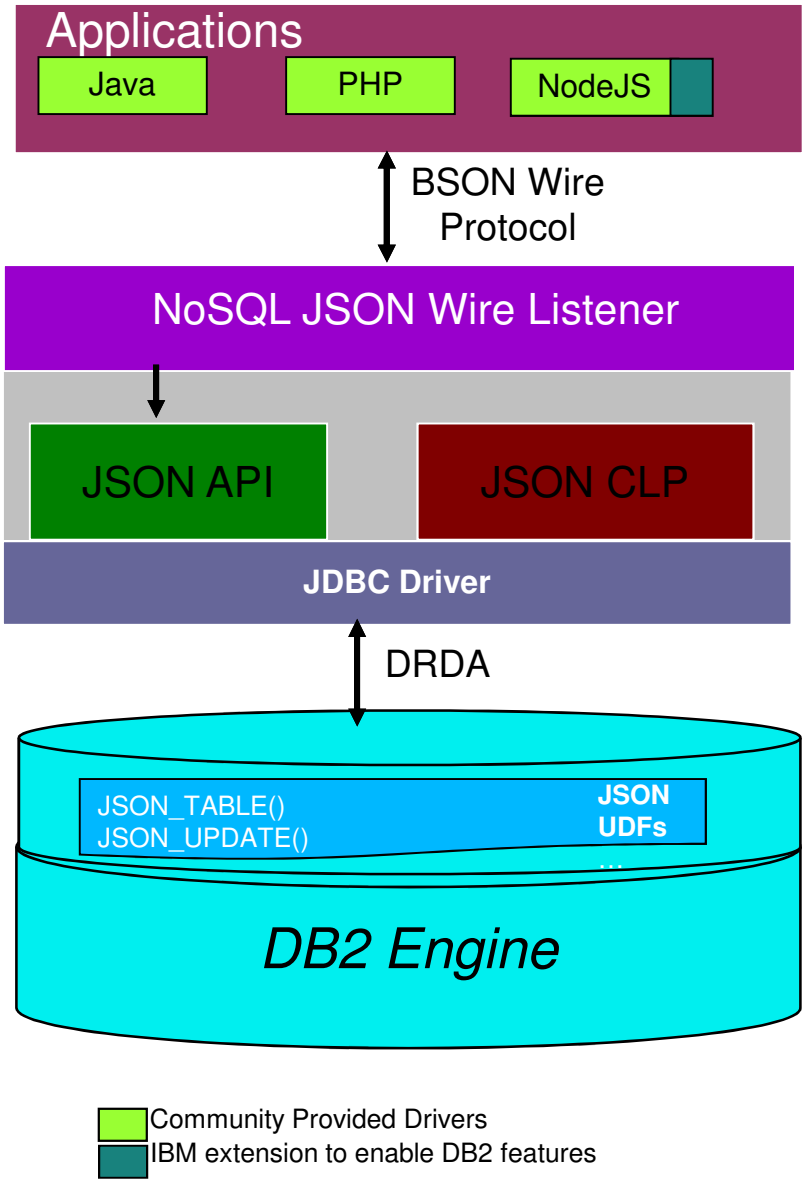
DB2 JSON Java API

- ***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**
- **Java command line**

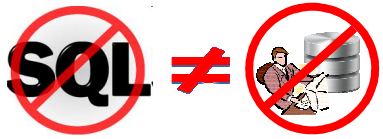


NoSQL JSON Wire Listener

- **Built on JSON API**
- **Leverage community**
- **Immediate reach to more applications and developers**
- **Presence in "New style apps“**



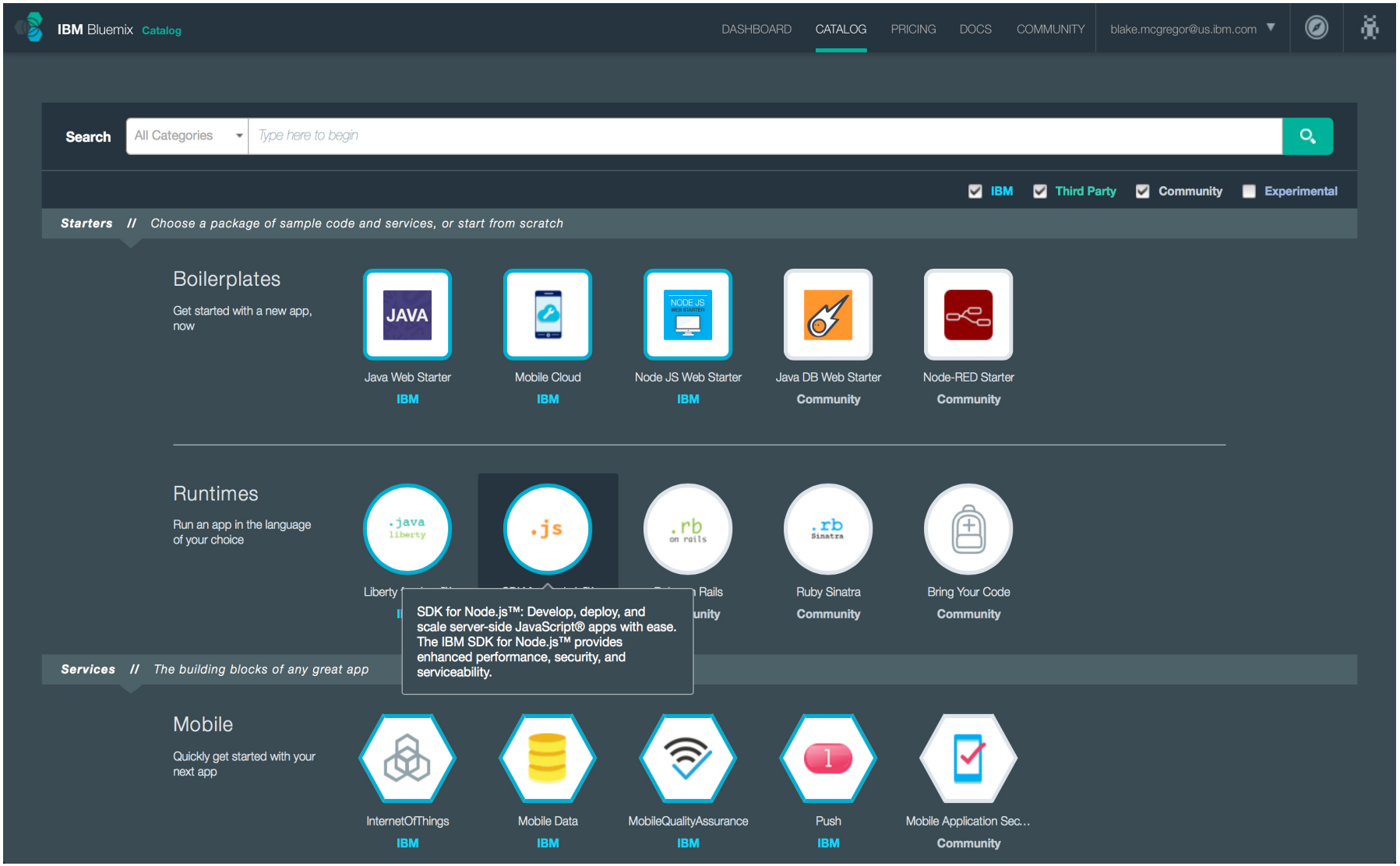
JSON and DB2 – Complementary Technologies



- **Does NoSQL mean NoDBA? NoDB2?**
  - Definitely not - the relational database isn't going away anytime soon
  - We see JSON as becoming a complementary technology to relational
- **Transactional atomicity is essential for mission critical business transactions**
  - DB2 JSON Store solution brings commits, transaction scope

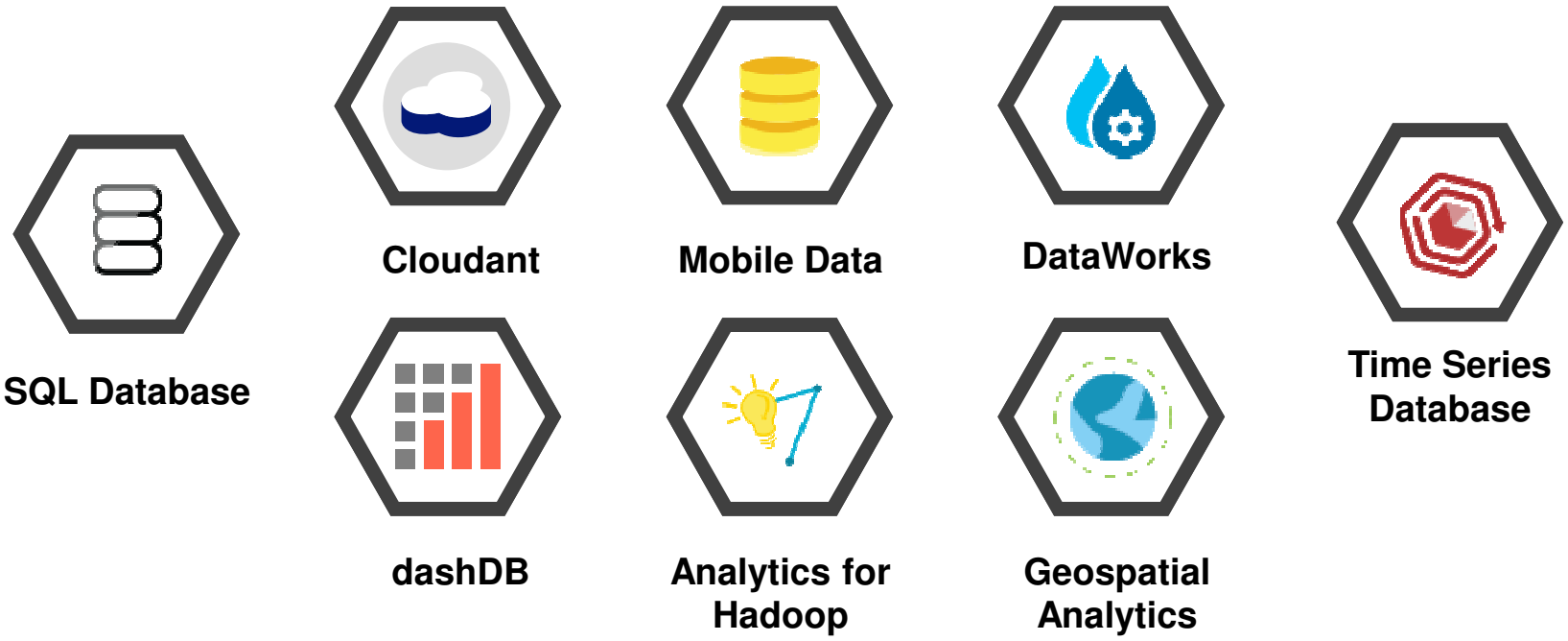
What's the Biggest Thing in NoSQL Lately?

IBM Delivering PaaS to Developers: Bluemix



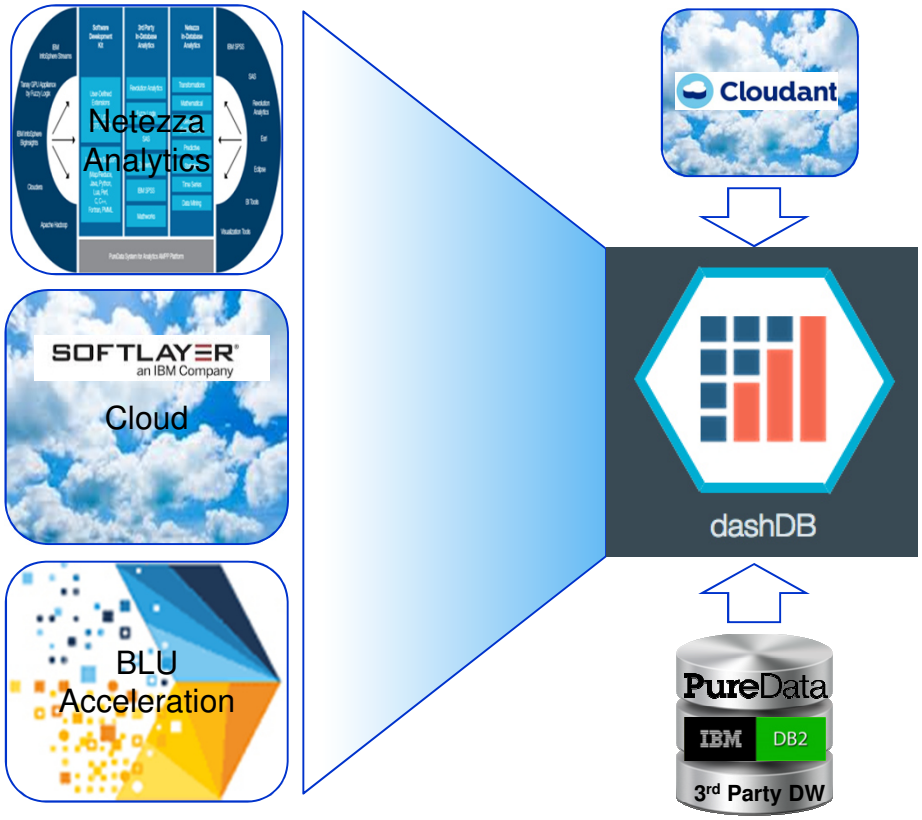
IBM Cloud Data Services

- **Broad portfolio of advanced database capabilities to manage and analyze any data**
- **Quickly provision databases to compose applications**
- **Flexible hybrid deployment models available**



SQLDB: Database as a Service (Relational) – It's DB2!!!

dashDB: Data Warehouse as a Service – It's DB2!!!



Build More

- **Deploy in hours** with rapid cloud provisioning
- **No infrastructure investment** for cloud agility

Grow More

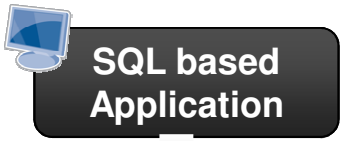
- **Load and Go** with no tuning required
- **Columnar** optimized for analytic workloads
- **Memory optimized** takes analytics beyond in-memory

Know More

- **In-Database analytics** built in
- **R Integration** for predictive modeling
- **Partner Ecosystem** for analytics
- **IBM Watson Analytics** ready


```
select * from hadoop
```

IBM Big SQL
- Best ANSI SQL support
- Federation
- FGAC Security
- Performance
- Oracle Compatibility
- Workload Management



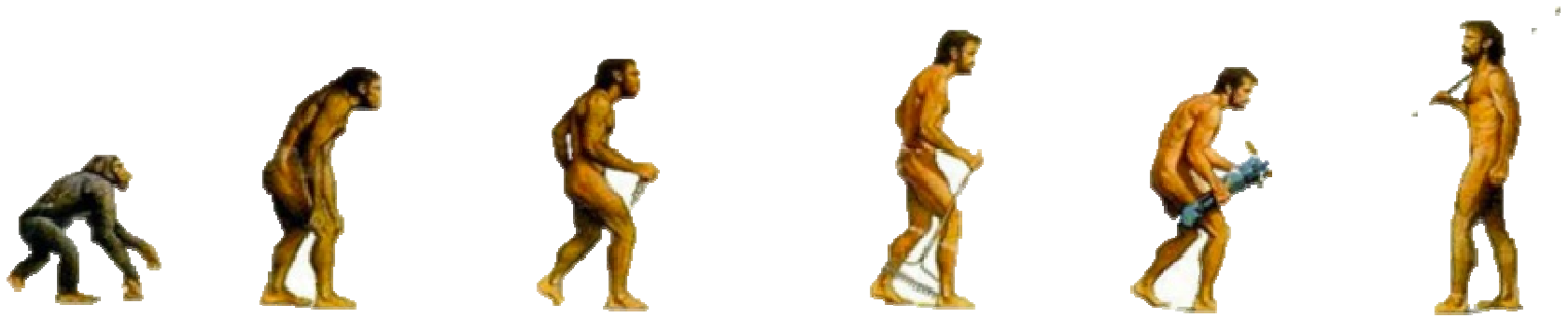
Big SQL
It's DB2!

Native Hadoop Data Sources

CSV	SEQ	Parquet	RC
AVRO	ORC	JSON	Custom

The Push to SQL in NoSQL – It's Your Expertise!

- **Lots of initiatives: Impala, BigSQL, SQL-H, Stinger, HAWQ, +++**
- **All Looking to push SQL into Hadoop and Other NoSQL stores**



Key-Value | Ordered Key-Value | Big Table | Document Full Text Search | Graph | SQL

The Biggest Thing In NoSQL Today is SQL!