InformationOnDemand2013

November 3 – 7
Mandalay Bay | Las Vegas, NV

#ibmiod

NoSQL DEEP DIVE

Relational + JSON = Simply Powerful

Jef Treece, IBM Product Manager jtreece@us.ibm.com
John F. Miller III, IBM Software Architect miller3@us.ibm.com
Keshava Murthy, IBM Software Architect rkeshav@us.ibm.com

IBM

Agenda

- NoSQL Business Drivers
- Live Demo
- JSON Store Overview
- Relational + JSON = Simply Powerful

InformationOnDemand2013

November 3 – 7
Mandalay Bay | Las Vegas, NV

#ibmiod

NoSQL BUSINESS DRIVERS

Jef Treece, IBM Product Manager

jtreece@us.ibm.com

@JefTreece

IBM

What is Driving IT Demand?

Advanced predictive analytics

Explosion of mobile devices

Cyber security

Real-time sensor data

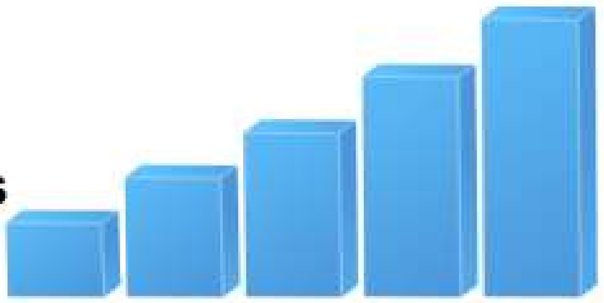
Infrastructure optimization – cloud computing

Growth of social media

Business Trends Driving NoSQL Adoption

Explosion of mobile devices

- **Applications must support mobile**
  - Interoperate with modern applications with agility
  - Enterprise infrastructure
- **Ability to scale to big data**
  - Commodity hardware and software
  - Use case are driving big data
  - Data in motion
- **Strategy: more interactions with customers**
  - Systems of engagement needed!
  - 71% CIOs see move toward social/digital collaboration
  - New class of applications are based on NoSQL



Global C-suite Study, http://www-935.ibm.com/services/us/en/c-suite/csuitestudy2013/

NoSQL Landscape

Key Value Store
- Couchbase
- Riak
- Citrusleaf
- Redis
- BerkeleyDB
- Membrain
- ...

Document
- MongoDB
- CouchDB
- RavenDB
- Couchbase
- ...

Column
- Cloudera
- HBase
- Hypertable
- Cassandra
- ...

Graph
- OrientDB
- DEX
- Neo4j
- GraphBase
- ...

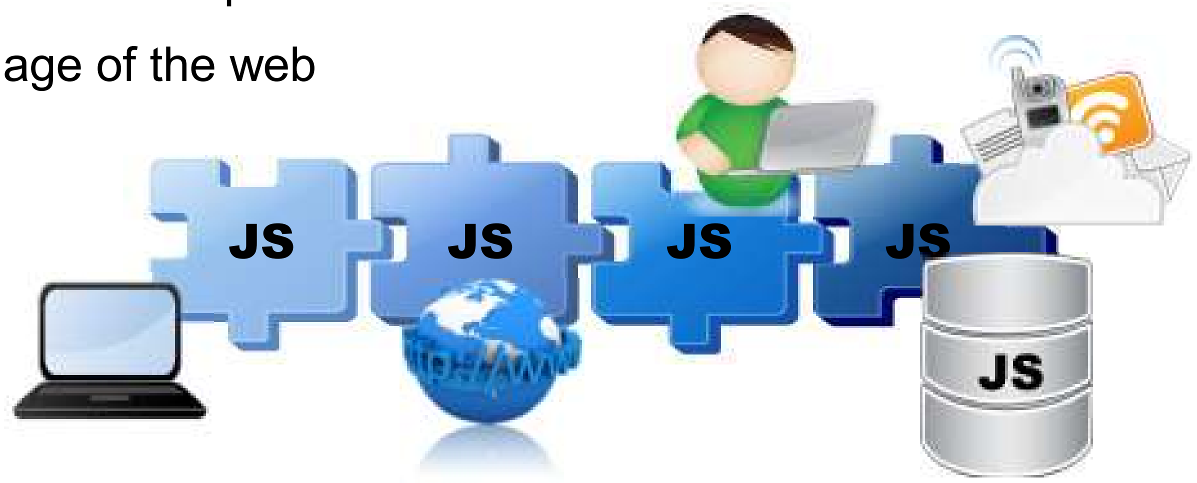
Martin Fowler says:
"aggregate-oriented"
What you're most likely to access as a unit.

Characteristics of a NoSQL Document Store

- **Ability to manage humongous data**
- **Enable rapid development**
  - Flexible schema development
  - Tap into the existing ecosystem – JSON, BSON, drivers, developers, modern applications
- **Capture real-time interactions**
  - Varied data sources

JavaScript Everywhere

- **JavaScript client development now dominant**
  - JavaScript and HTML5 for browser presentation
  - JavaScript mobile applications
- **JSON: JavaScript Object Notation**
  - End-to-end JavaScript
  - The language of the web



Traditional Relational Database Still Required

- Relational model works best for transactional data
- Enterprise data exists in relational databases
- Relational database preferred for most analytics

You need both
at the same time!

NoSQL + Relational

E-commerce, social commerce

Explosion of mobile devices – gaming and social apps

Smart Devices

Machine data and real-time operational decisions

Advertising: serving ads and real-time bidding

Social networking, online communities

Business Value of NoSQL + RDBMS

- **Level 1: Hybrid storage**
  - JSON and relational tables in same storage engine
  - Different apps, same database

    Reduces cost and complexity!

    Performance!

- **Level 2: Hybrid applications**
  - A single application brings together RDBMS and NoSQL
  - Business insight from bringing the two different types of data and the two different requirements together

    New business patterns!

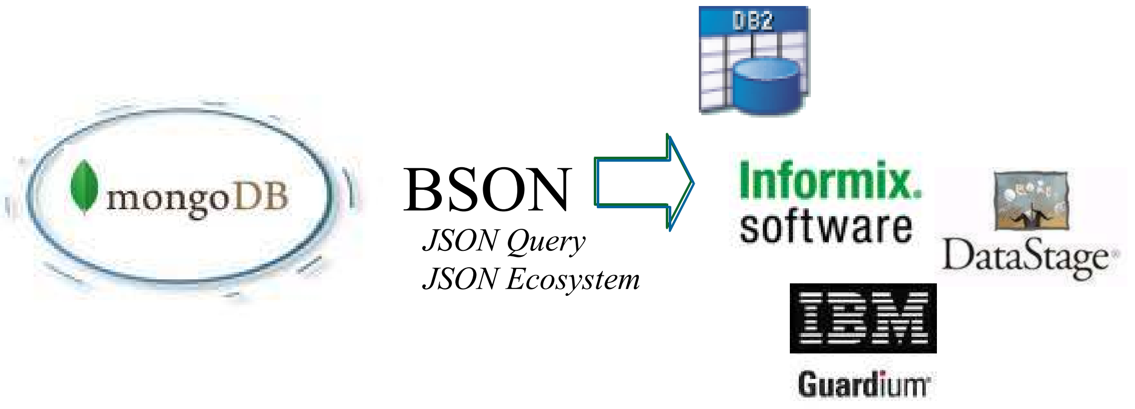
IBM/MongoDB Partnership

MongoDB and IBM announced a partnership in June 2013

ComputerWeekly.com — IBM and 10Gen collaborate on database standard for enterprise mobile

SILICON VALLEY BUSINESS JOURNAL — IBM and 10Gen team up to dominate the database market

ZDNet — IBM, 10gen partner to bring mobile to the enterprise



Enter IBM Informix 12.10

Informix. software Simply powerful for both!

- **Now you have the right tool for the job – all in one toolbox**
  - System of record: Informix RDBMS
  - System of engagement: Informix NoSQL
  - Hybrid storage, hybrid applications

Thank You!

Next Up...

Demo

John F. Miller III, IBM Software Architect

miller3@us.ibm.com

Information On Demand 2013

November 3 – 7
Mandalay Bay | Las Vegas, NV

#ibmiod

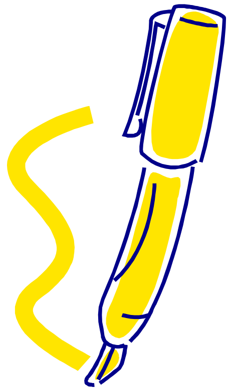
BUILDING A REAL LIFE APPLICATION

IOD Attendee Photo Application

Allow conference attendee to take and share photo!

- Web application geared for smart devices allowing attendees to take and view photos
- View the most popular pictures
- View the pictures you took
- See what pictures are trending
- Allow users to ask for more information

Technology Highlights



- Create a hybrid application using NoSQL, traditional SQL, timeseries mobile web application
  - Utilizing both JSON collections, SQL tables and timeseries
  - Utilize IBM Dojo Mobile tools to build a mobile application
  - Leverage new mongo client side drivers for fast application development and delployment
- Demonstrate scale-out using sharding with over **100 nodes**
- Cloud based solution using Amazon Cloud
  - Can be deployed on PureFlex or SoftLayer
- Provide real-time analytics on all forms of data
  - Leverage existing popular analytic front-end IBM-Congos
  - Utilize an in-memory columnar database accelerator to provide real-time trending analytics on data

Mobile Device Application Architecture

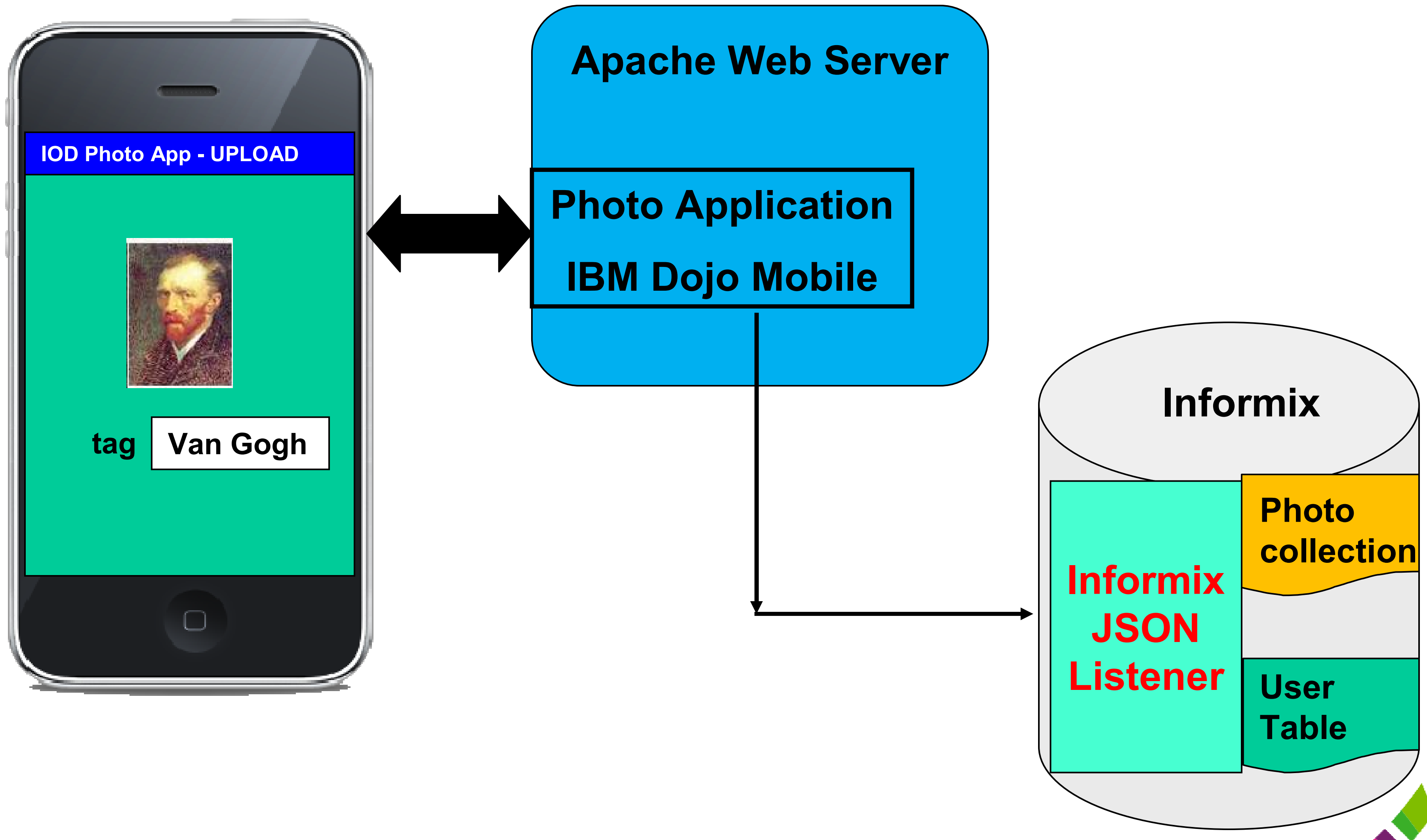
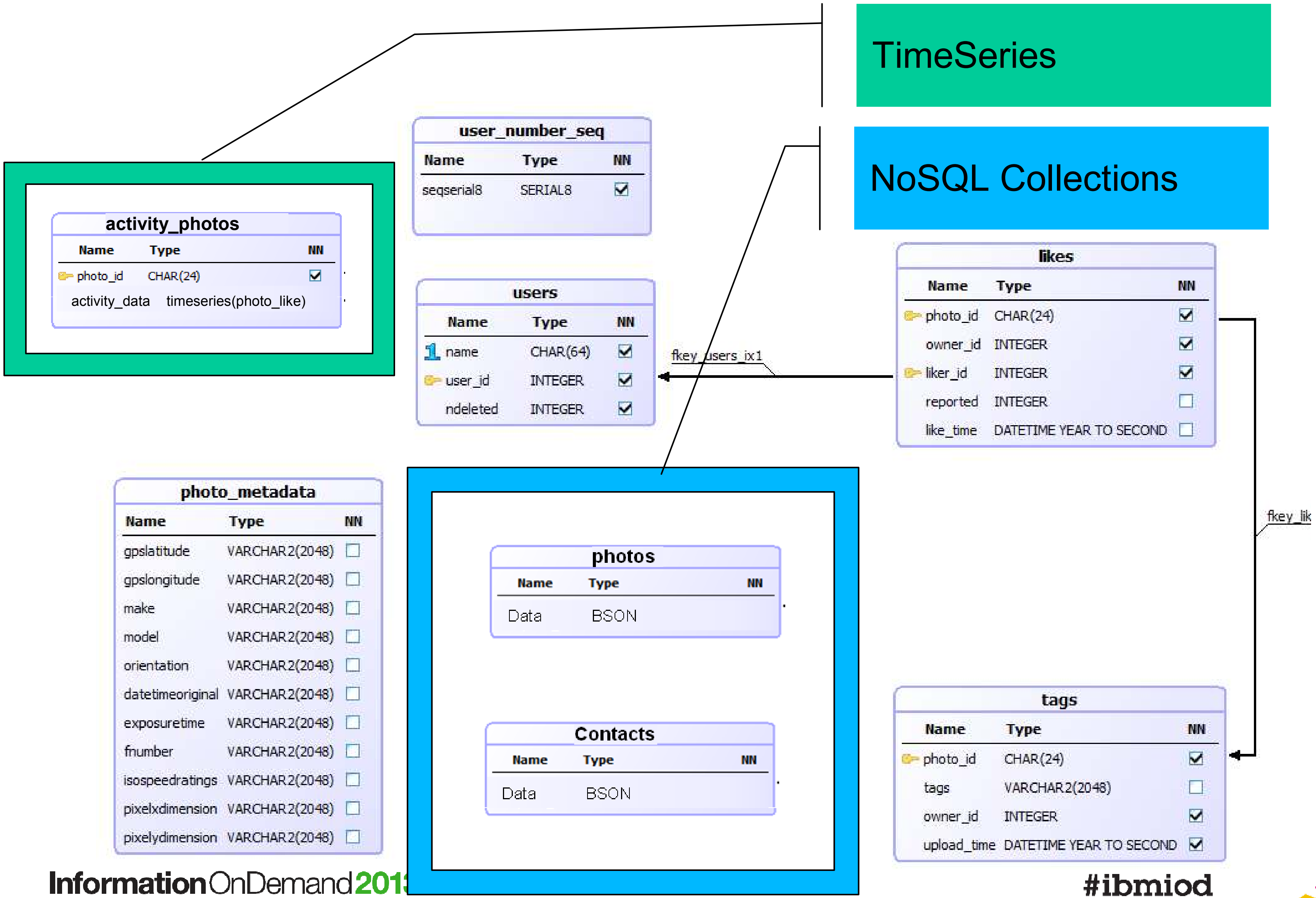


Photo Application Schema



activity_photos

Name	Type	NN
photo_id	CHAR(24)	✓
activity_data	timeseries(photo_like)	

user_number_seq

Name	Type	NN
seqserial8	SERIAL8	✓

users

Name	Type	NN
name	CHAR(64)	✓
user_id	INTEGER	✓
ndeleted	INTEGER	✓

likes

Name	Type	NN
photo_id	CHAR(24)	✓
owner_id	INTEGER	✓
liker_id	INTEGER	✓
reported	INTEGER	☐
like_time	DATETIME YEAR TO SECOND	☐

photo_metadata

Name	Type	NN
gpslatitude	VARCHAR2(2048)	☐
gpslongitude	VARCHAR2(2048)	☐
make	VARCHAR2(2048)	☐
model	VARCHAR2(2048)	☐
orientation	VARCHAR2(2048)	☐
datetimeoriginal	VARCHAR2(2048)	☐
exposuretime	VARCHAR2(2048)	☐
fnumber	VARCHAR2(2048)	☐
isospeedratings	VARCHAR2(2048)	☐
pixelxdimension	VARCHAR2(2048)	☐
pixelydimension	VARCHAR2(2048)	☐

photos

Name	Type	NN
Data	BSON	

Contacts

Name	Type	NN
Data	BSON	

tags

Name	Type	NN
photo_id	CHAR(24)	✓
tags	VARCHAR2(2048)	☐
owner_id	INTEGER	✓
upload_time	DATETIME YEAR TO SECOND	✓

Application Considerations

Flexible Schema

- Photo meta-data varies from camera to camera
- A Picture and all its meta data are stored in-document
- Pictures are stored in a JSON collection
- Pre-processing on the phone ensures only reasonable size photos are sent over the network.

Example of Live JSON Photo Data

JSON Data

{"_id":ObjectId("526157c8112c2fe70cc06a75"), "Make":"NIKON CORPORA
TION","Model":"NIKON D60","Orientation":"1","XResolution":"300","
YResolution":"300","ResolutionUnit":"2","Software":"Ver.1.00 ","D
ateTime":"2013:05:15 19:46:36","YCbCrPositioning":"2","ExifIFDPoi
nter":"216","ExposureTime":"0.005","FNumber":"7.1","ExposureProgr
am":"Not defined","ISOSpeedRatings":"100",

"Contrast":"Normal","Saturation":"Normal","Sharpness":"Normal",
"SubjectDistanceRange":"Unknown","name":"DSC_0078.JPG","img_d
ata":"data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABcQ

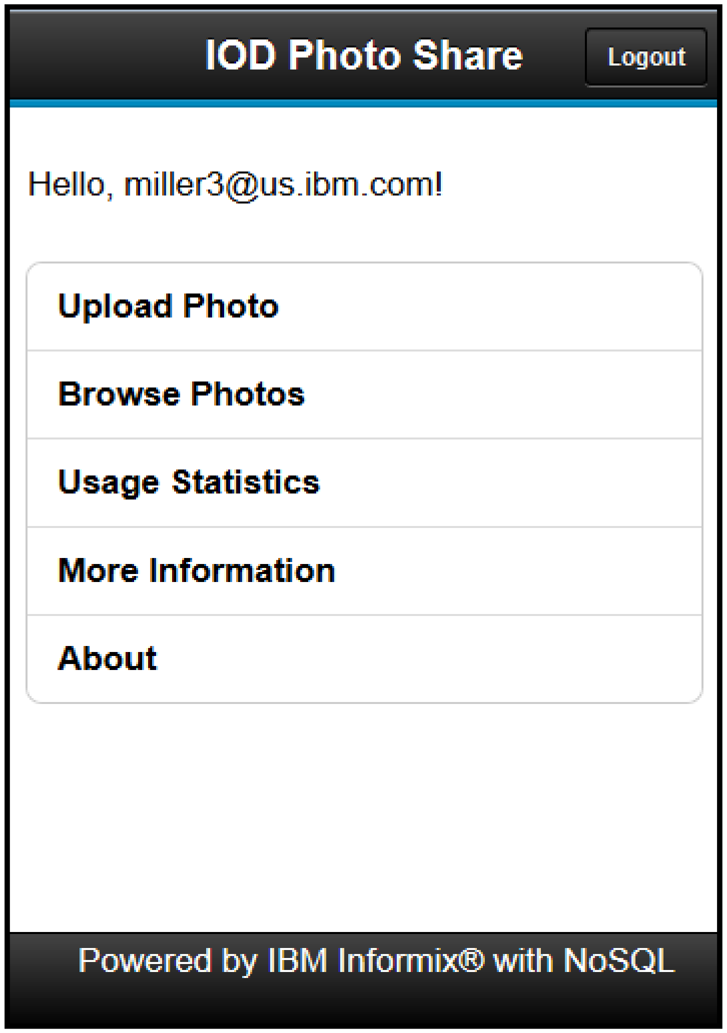
Motivation of Sharding

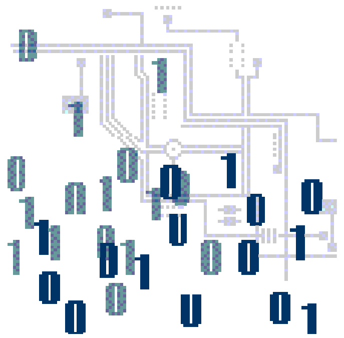
- Enables horizontal scaling (partitioning)
- The application strategy in step with business
  - Start small and grow with commodity hardware as the business grows
  - Grow as you go
- Economics of solution
  - 4 nodes each of 4 cores
  - 1 node of 16 cores

Code Snippets

- Used PHP and mongo PHP API

- Example showing
  - Inserting
  - Retrieving data
  - Deleting JSON documents and SQL rows
  - Executing Stored Procedures





Basic PHP Programming Overview Information

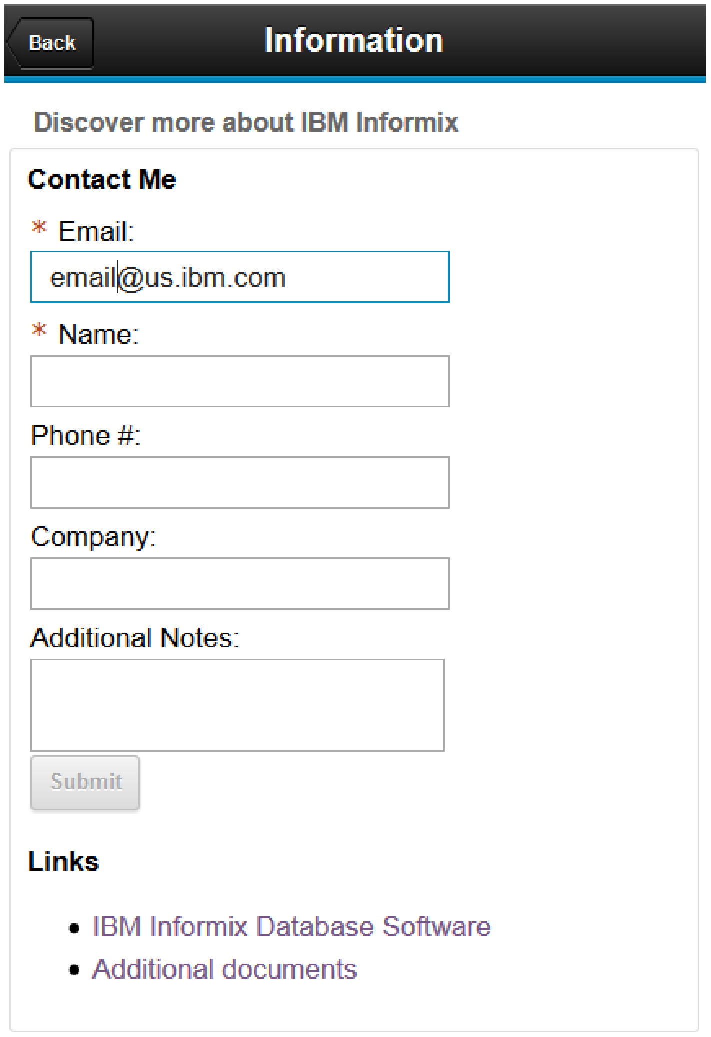
- List of NoSQL collection names and SQL tables names
- Function to set the active database and return the Collection

```
 private $conn;

 private $dbname = "photo_demo";
 private $photoCollectionName = "photos";
 private $contactsCollectionName = "contacts";
 private $sqlCollectionName = 'system.sql';
 private $userTableName = "users";
 private $tagsTableName = "tags";
 private $likesTableName = "likes";

 private $photoQueryProjection = array("_id" => 1, "tags" => 1,
                                       "user_id" => 1, "img_data" => 1);
/**
 * Get collection by name
 * @param MongoCollection $collectionName
 */
private function getCollection($collectionName) {
   return $this->conn->selectDB($this->dbname)->selectCollection ($collectionName);
}
```

Insert Example



Back

Information

Discover more about IBM Informix

Contact Me

* Email:

email@us.ibm.com

* Name:

Phone #:

Company:

Additional Notes:

Submit

Links

- IBM Informix Database Software
- Additional documents

- Information is placed in the contacts collection

Insert Data into a Collection

- Very simple to insert JSON data into a collection using the MongoAPIs

```
/**
 * Insert user's contact information into contacts table.
 */
public function insertContact( $json ) {
    if (! is_array ( $json )) {
            return "Contact info not in JSON format.";
    }
    try {
        $result=$this->getCollection($this->contactsCollectionName)->insert($json);
        if ($result ["ok"] != 1) {
                return $result ["err"];
        }
    } catch ( MongoException $e ) {
            return $e->getMessage ();
    }
    return "ok";
}
```

Retrieve Collection Information

- Very simple to retrieve data from a collection using the MongoAPIs
- Data is returned as a JSON document

```
/**
 * Get all contact info
 */
 public function adminContacts() {
    $contactsCollection = $this->getCollection($this-
>contactsCollectionName);
    $cursor = $contactsCollection->find();
    $results = $this->getQueryResults($cursor);
    return $results;
}
```

Naughty Pictures

- Allow users to flag naughty pictures
- Have naughty pictures automatically removed

Delete a Photo and its Information

- Deleting from SQL Tables and NoSQL Collection is exactly the same

```
/**
 * Delete photo
 */
public function deletePhoto($id) {
       try {
           // First delete from likes and tags tables
           $query = array('photo_id' => $id['_id']);
           $result = $this->getCollection($this->likesTableName)->remove($query);
           if ($result ["ok"] != 1) {
                  return $result["err"];
           }
           $result = $this->getCollection($this->tagsTableName)->remove($query);
           if ($result ["ok"] != 1) {
                   return $result["err"];
           }

           // Then delete the photo from the collection
           $query = array('_id' => new MongoId($id['_id']));
           $result = $this->getCollection ( $this->photoCollectionName )->remove ( $query );
           if ($result ["ok"] != 1) {
                return $result["err"];
           }
      } catch ( MongoException $e ) {
           return $e->getMessage();
      }
      return "ok";
}
```

Executing a Stored Procedure in MongoAPI

```
/**
 * Get the user_id for a particular user name (email address).
 *
 * Calls a stored procedure that will insert into the users table if the
 * user does not exist yet and returns the user_id.
 *
 * @param string $username
 * @return int $user_id
 */
 public function getUserId($username) {
    $username = trim($username);

    try {
        $sql = "EXECUTE FUNCTION getUserID('" . $username . "')";
        $result = $this->getCollection($this->sqlCollectionName)->findOne(array('$sql'=>$sql));
        if (isset($result['errmsg'])) {
               return "ERROR. " . $result['errmsg'];
        }
        return $result['user_id'];
    } catch (MongoException $e) {
        return "ERROR. " . $e->getMessage();
    }
}
```

Real Time Analytics

- Customer Issues
  - Several different models of data (SQL, NoSQL, TimeSeries/Sensor)
  - NoSQL is not strong building relations between collections
  - Most valuable analytics combine the results of all data models
  - Most prominent analytic system written using standard SQL
  - ETL & YAS (Yet Another System)

- Solution

Provide a mapping of the required data in SQL form

  - Enables common tools like Cognos

Analytics on a Hybrid Database

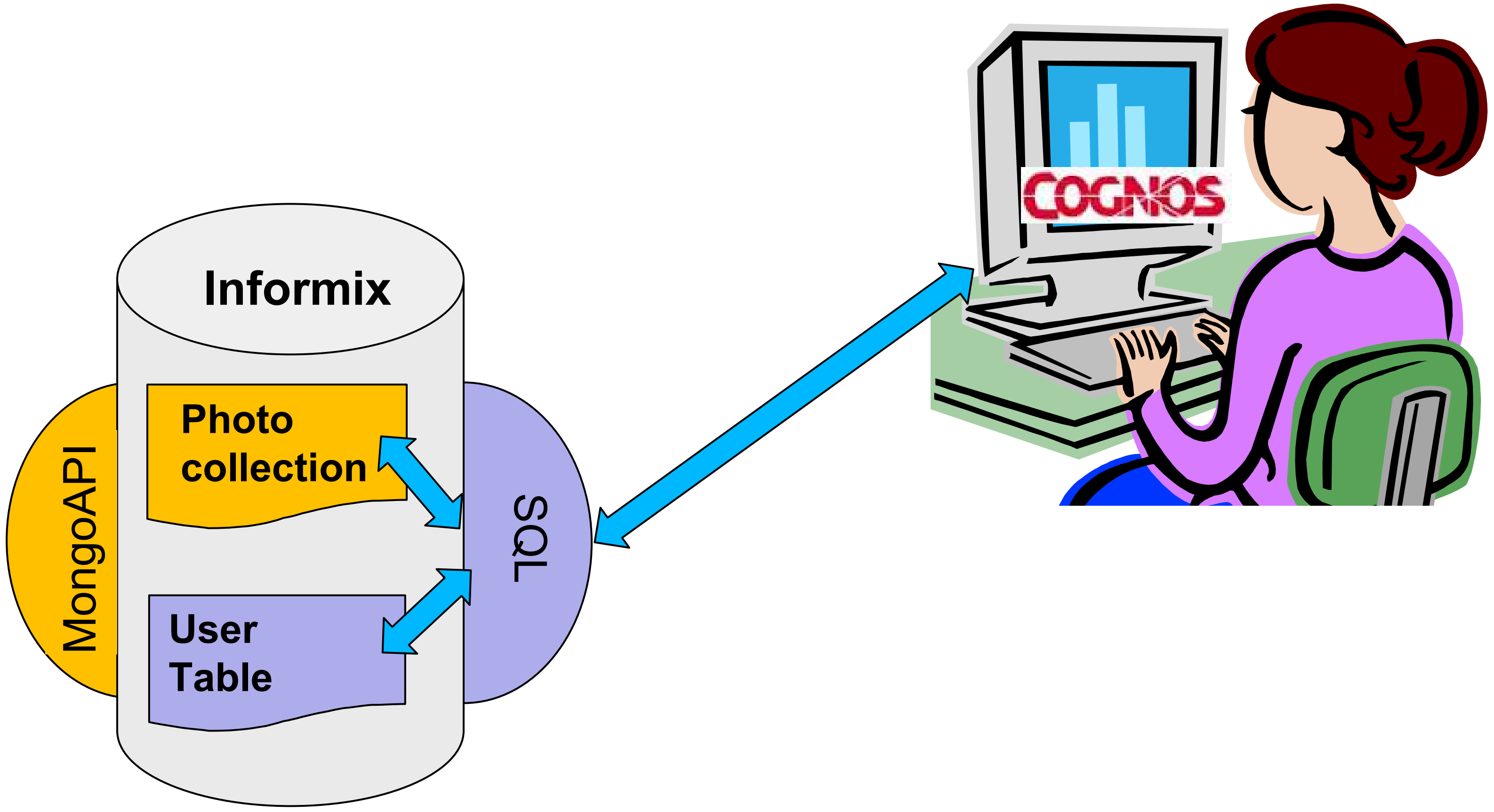
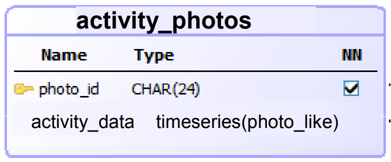
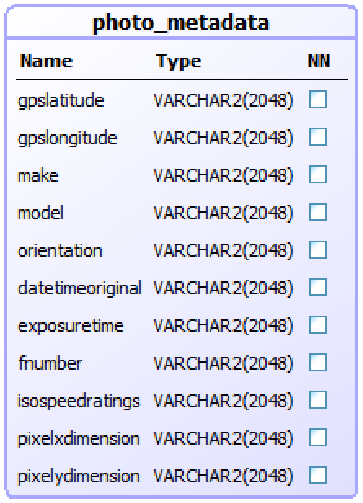


Photo Application
SQL Mapping of NoSQL PHOTO Collection



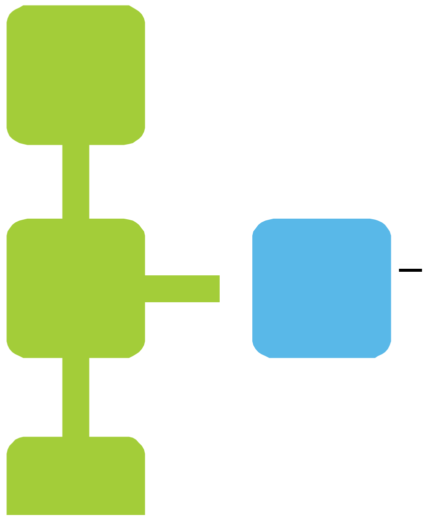
activity_photos

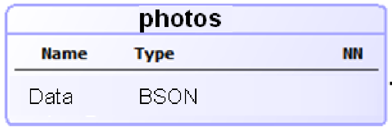
Name	Type	NN
photo_id	CHAR(24)	✓
activity_data	timeseries(photo_like)	



photo_metadata

Name	Type	NN
gpslatitude	VARCHAR2(2048)	☐
gpslongitude	VARCHAR2(2048)	☐
make	VARCHAR2(2048)	☐
model	VARCHAR2(2048)	☐
orientation	VARCHAR2(2048)	☐
datetimeoriginal	VARCHAR2(2048)	☐
exposuretime	VARCHAR2(2048)	☐
fnumber	VARCHAR2(2048)	☐
isospeedratings	VARCHAR2(2048)	☐
pixelxdimension	VARCHAR2(2048)	☐
pixelydimension	VARCHAR2(2048)	☐





photos

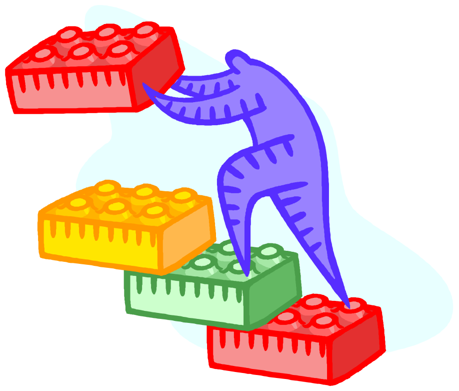
Name	Type	NN
Data	BSON	

Mapping A Collection To A SQL Table

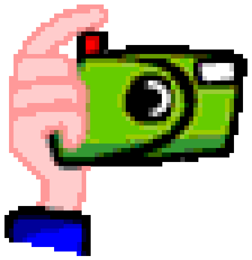
```
CREATE VIEW photo_metadata (gpslatitude, gpslongitude,
            make, model, orientation, datetimeoriginal,
            exposuretime, fnumber, isospeedratings,
            pixelxdimension, pixelydimension)
AS SELECT BSON_VALUE_LVARCHAR ( x0.data , 'GPSLatitude' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'GPSLongitude' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'Make' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'Model' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'Orientation' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'DateTimeOriginal' ) ,
          BSON_VALUE_LVARCHAR ( x0.data , 'ExposureTime'),
          BSON_VALUE_LVARCHAR ( x0.data , 'FNumber' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'ISOSpeedRatings' ),
          BSON_VALUE_LVARCHAR ( x0.data , 'PixelXDimension' ) ,
          BSON_VALUE_LVARCHAR ( x0.data , 'PixelYDimension')
   FROM photos x0;
```

Configure Informix on Amazon Cloud Simple

- Instantiate the Amazon image
- Setup the storage
- Install the product
- Start the system
- Configure sharding

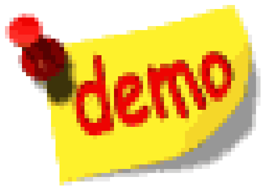


All under 3 minutes



What happens in Vegas

is always recorded!!



Please Visit

www.nosqldemo.com

And have Fun

InformationOnDemand2013

November 3 – 7
Mandalay Bay | Las Vegas, NV

#ibmiod

NOSQL, JSON AND BSON OVERVIEW

Technical Opportunities/ Motivation

What are NoSQL Databases?

Quick overview of JSON

What is sharding?

IBM

New Era in Application Requirements

- Store data from web/mobile application in their native form
  - New web applications use JSON for storing and exchanging information
  - Very lightweight – write more efficient applications
  - It is also the preferred data format for mobile application back-ends

- Move from development to production in no time!
  - Ability to create and deploy flexible JSON schema
  - Gives power to application developers by reducing dependency on IT

Ideal for agile, rapid development and continuous integration

What is a NoSQL Document Store?

- Not Only SQL or NOt allowing SQL

- A non-relational database management systems
  - Flexible schema
  - Avoids join operations
  - Scales horizontally
  - Eventually consistent (no ACID)

- Good with distributing data and fast application development

Provides a mechanism for storage and retrieval of data while providing horizontal scaling.

IBM Use Case Characteristics for JSON

Schema flexibility and development agility	• Application not constrained by fixed pre-defined schema • Ability to handle a mix of structured and unstructured data
Dynamic elasticity	• Rapid horizontal scalability • Ability to add or delete nodes dynamically in the Cloud/Grid • Application transparent elasticity
Continuous availability	• 24x7x365 availability • Online maintenance operations • Ability to upgrade hardware or software without down time
Consistent low latency, even under high loads	• Ability to handle thousands of users • Typically millisecond response time
Low cost infrastructure	• Commonly available hardware (Windows & Linux,…)
Reduced administration and maintenance	• Ease of deployment • Install, configure add to exiting environment in minutes

Example of Supported JSON Types

- There are 6 types of JSON Values
- Example of each JSON type
- Mongo-specific JSON types in blue
  - date

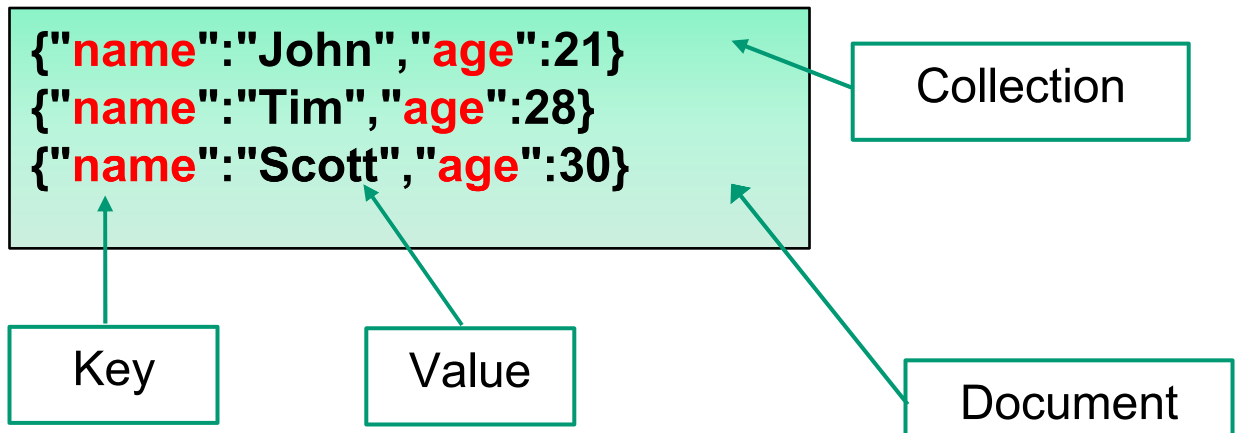
```
{
"string":"John",
"number":123.45,
"boolean":true,
"array":[ "a", "b", "c" ],
"object: { "str":"Miller", "num":711 },
"value": NULL,
"date": ISODate("2013-10-01T00:33:14.000Z")
}
```

The Power of JSON Drives Flexible Schema

- JSON key value pair enables a flexible schema
- Flexible schema simplifies deployment of new/upgraded applications
- Fast/Agile application development
  - Minimal to no schema management
- Adept at variable attribute management
  - Easy to add new parts or objects
- No transformation of data to match schema

Basic Translation Terms/Concepts

Mongo/NoSQL Terms	Traditional SQL Terms
FIND	SELECT
SAVE	INSERT
REMOVE	DELETE
UPDATE	UPDATE
ensureIndex	CREATE INDEX
SORT()	ORDER BY
LIMIT	LIMIT/FIRSTN



Name	Age
John	21
Tim	28
Scott	30

Simple Code Example

```
use mydb
db.posts.insert( ‘{“author”:”John”, “date”,”2013-04-20”,”post”,”mypost…”}’ )
```

- Creates the database “my*db*” if it does not exists
- Creates the collection “*posts*” if the it does not exists
- Insert a record into a blog post by user John

```
db.posts.find ( ‘{ “author”:”John” }’ )
```

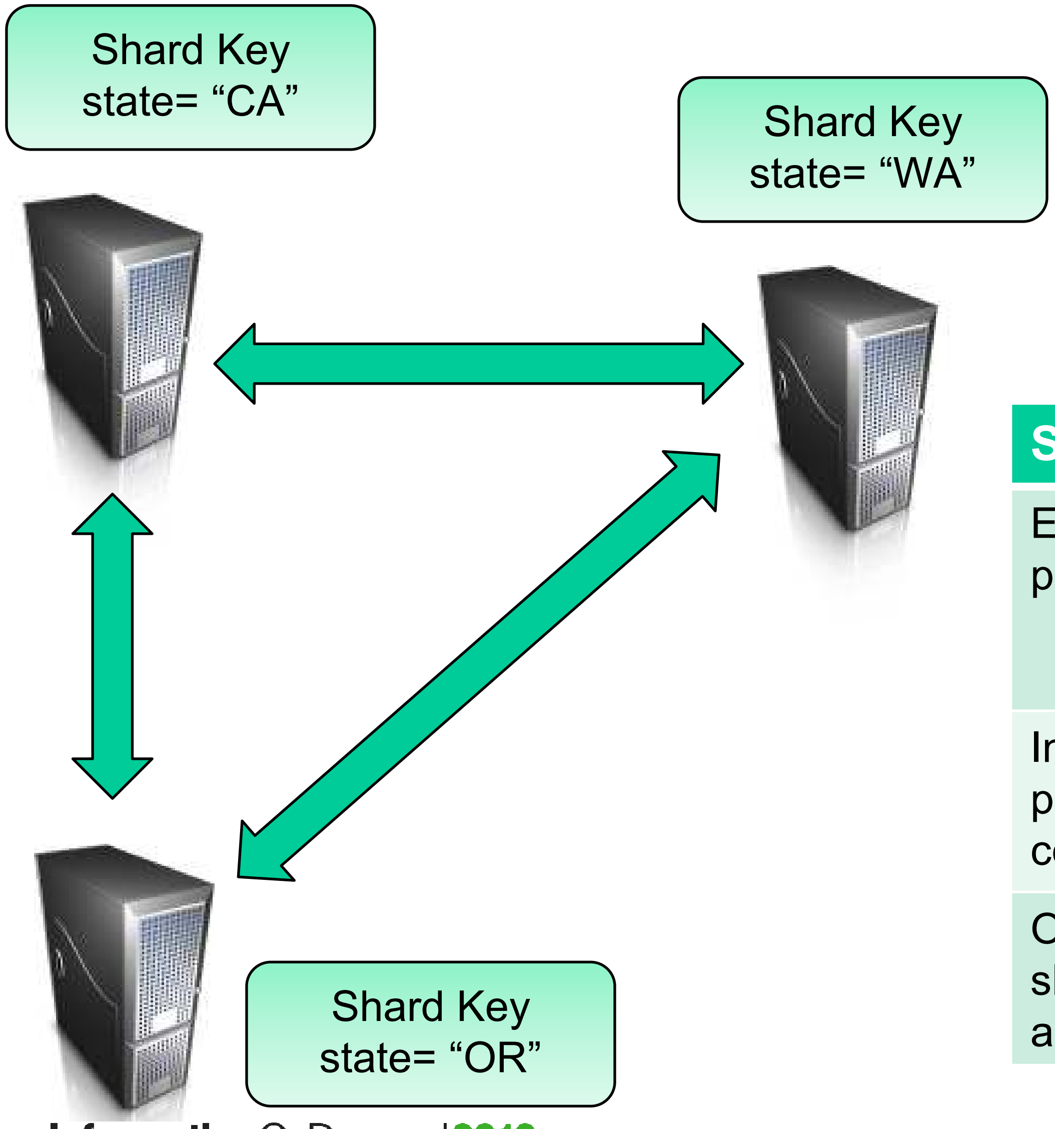
- Retrieve all posts by user John

Dynamic Elasticity

- Rapid horizontal scalability
  - Ability for the application to grow by adding low cost hardware to the solution
  - Ability to add or delete nodes dynamically
  - Ability rebalance the data dynamically
- Application transparent elasticity

Sharding

Difference between Sharding Data VS Replication



Sharding	Replication
Each node holds a portion of the data • Hash • Expression	Same data on each node
Inserted data is placed on the correct node	Data is copied to all nodes
Operations are shipped to applicable nodes	Work on local copy and modification are propagated

InformationOnDemand2013

November 3 – 7
Mandalay Bay | Las Vegas, NV

#ibmiod

NEW INFORMIX NOSQL/JSON CAPABILITIES

IBM Informix High Level Solution

- Flexible schema, native JSON and BSON Data Types
- Super scale out & elasticity
- Provide Mongo compatible programming
- Leveraging Informix for NoSQL

Two New Data Types JSON and BSON

- Native JSON and BSON data types
- Index support for NoSQL data types
- Native operators and comparator functions allow for direct manipulation of the BSON data type
- Database Server seamlessly converts to and from
  - JSON ←→ BSON
  - Character data ←→ JSON

Informix JSON Store Benefits

- Informix provides
  - Row locking on the individual JSON document
    - MongoDB locks the database
  - Large documents, up to 2GB maximum size
    - MongoDB limit is 16MB
  - Ability to compress documents
    - MongoDB currently not available
  - Ability to intelligently cache commonly used documents
    - MongoDB currently not available

Flexible Schema

- Applications use JSON, a set of key-value pairs
- JSON is text , BSON is the binary representation.
- The explicit key-value pairs within the JSON/BSON document will be roughly equivalent to columns in relational tables.
- Applications typically denormalize the schema
  - Customer, customer address, customer contacts all in a single JSON

Flexible Schema

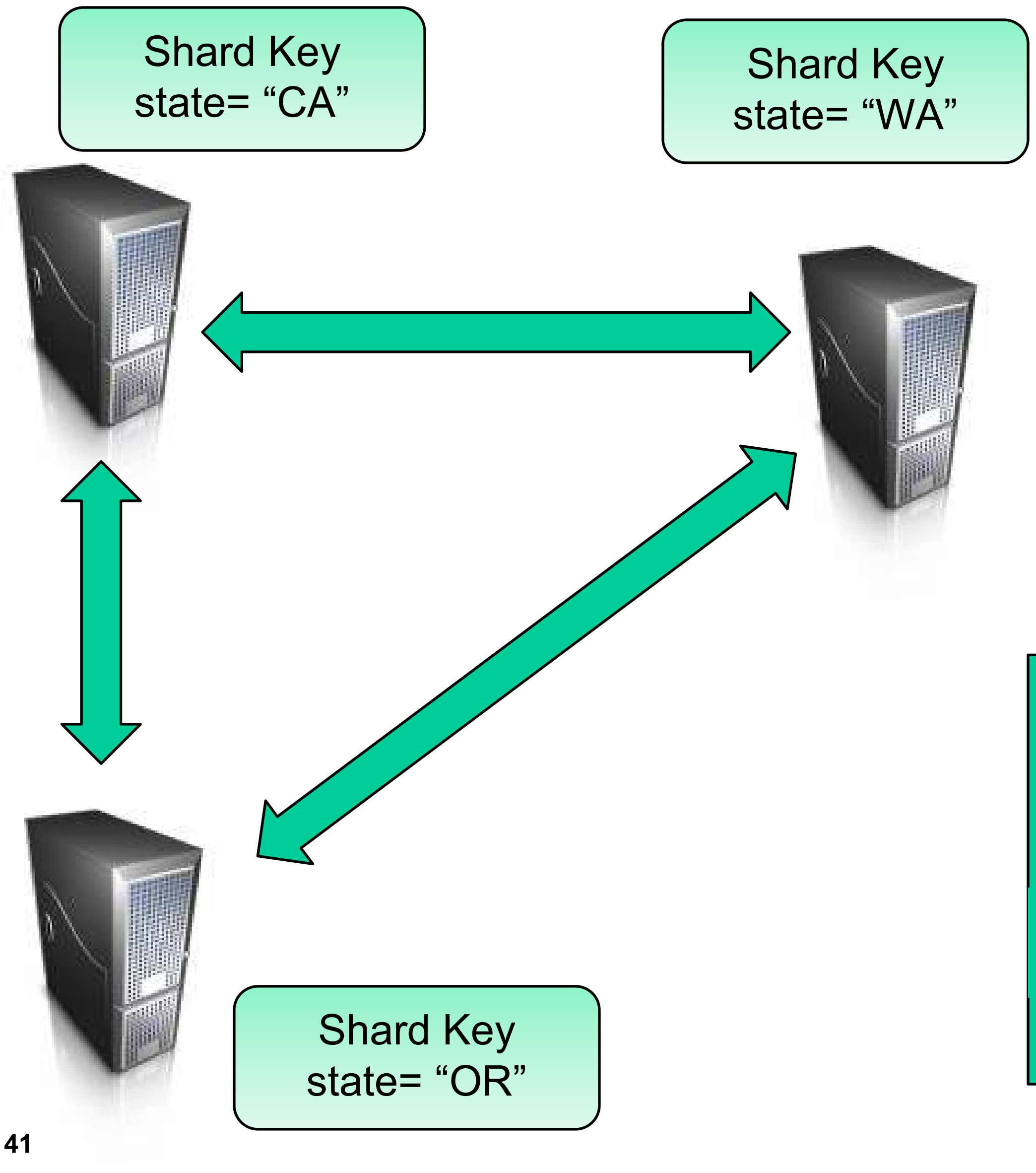
- However, there are differences!
  - The type of the Key Value data encoded within BSON is determined by the client
  - Server is unaware of data type of each Key Value pair at table definition time.
  - No guarantees that data type for each key will remain consistent in the collection.
  - The keys in the BSON document can be arbitrary
  - While customers exploit flexible schema, they're unlikely to create a single collection and dump ***everything under the sun*** into that collection.
  - Developers typically denormalize the tables (a JSON document will contain customer+customer addr + customer demographics + …) to avoid joins.

Indexing

- Supports B-Tree indexes on any key-value pairs.
- Typed indices could be on simple basic type (int, decimal,)
- Type-less indices could be created on BSON and use BSON type comparison
- Informix translates ensureIndex() to CREATE INDEX
- Informix translates dropIndex() to DROP INDEX

Mongo Operation	SQL Operation
`db.customers.ensureIndex( {orderDate:1, zip:-1})`	`CREATE INDEX IF NOT EXISTS v_customer_2 ON customer (bson_extract(data,‘orderDate') ASC, bson_extract(data,‘zip') DESC) USING BSON`
`db.customers.ensureIndex( {orderDate:1},{unique:true})`	`CREATE UNIQUE INDEX IF NOT EXISTS v_customer_3 ON customer (bson_extract(data,'c1') ASC USING BSON`

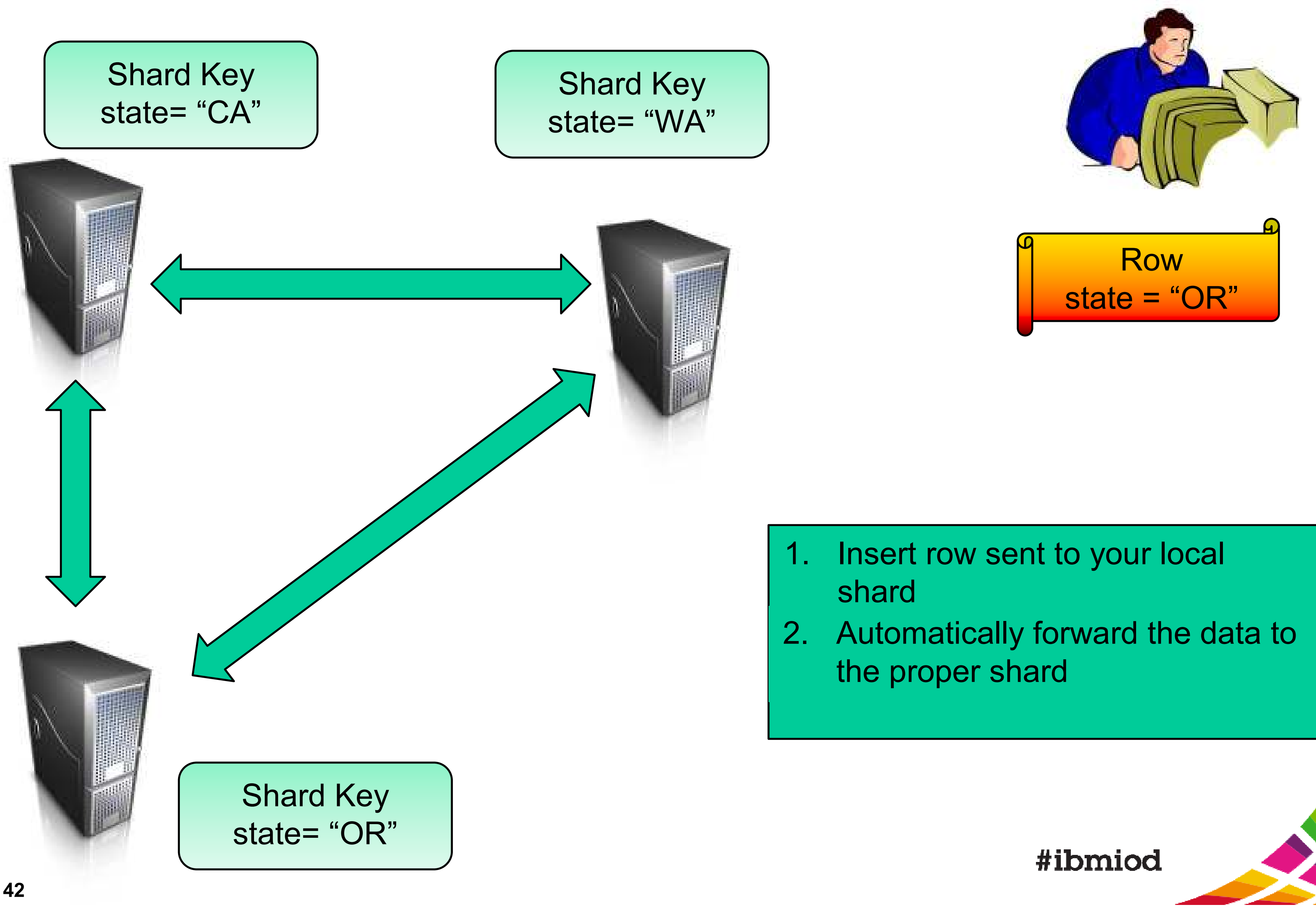
Scaling Out Using Sharded Queries



Find sold cars for all states

1. Request data from local shard
2. Automatically sends request to other shards requesting data
3. Returns results to client

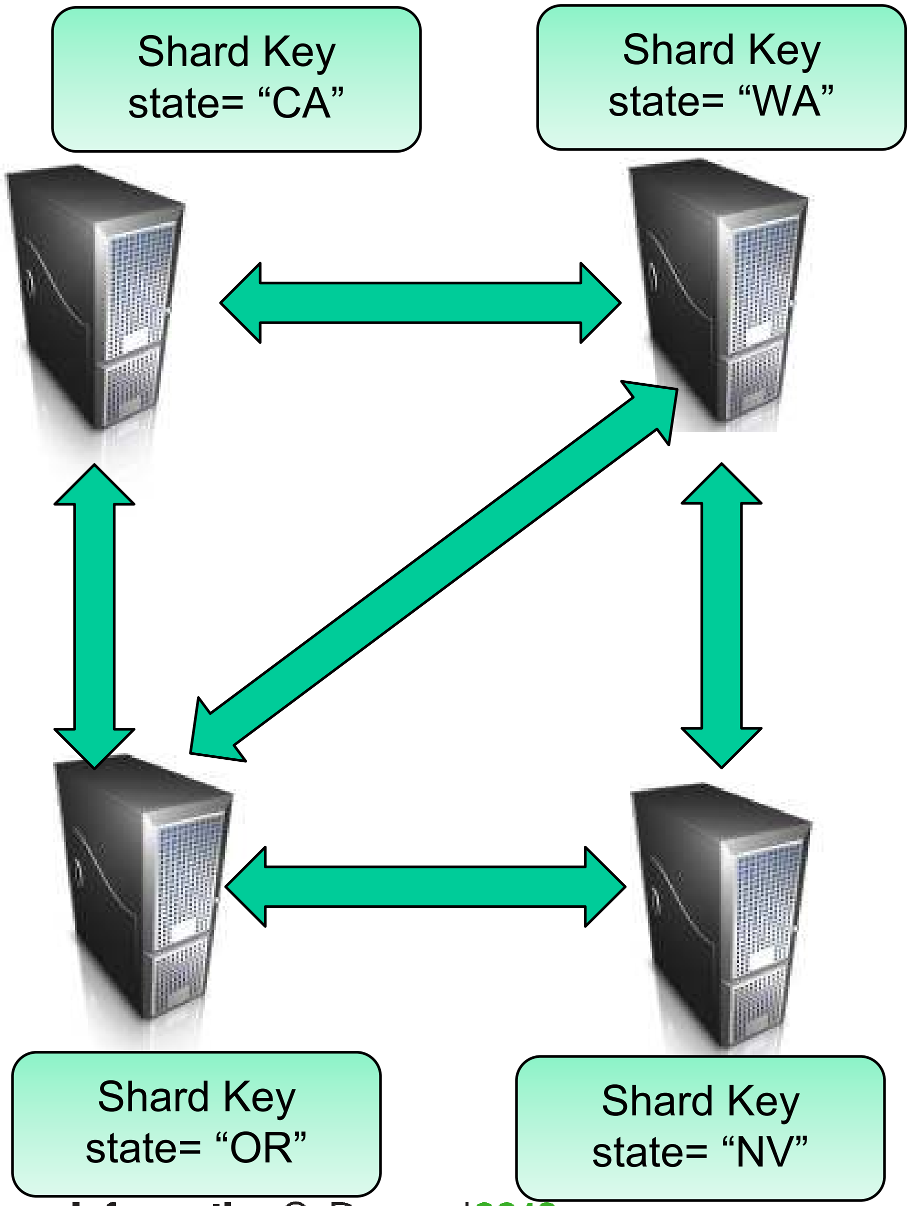
Scaling Out Using Sharded Inserts



1. Insert row sent to your local shard
2. Automatically forward the data to the proper shard

#ibmiod

Scaling Out Adding a Shard



Command
Add Shard “NV”

1. Send command to local node
2. New shard dynamically added, data re-distributed (if required)

Sharding with Hash

- Hash based sharding simplifies the partitioning of data across the shards
- Benefits
  - No data layout planning is required
  - Adding additional nodes is online and dynamic
- Cons
  - Adding additional node requires data to be moved
- Data automatically broken in pieces

Mongo API Command to add a shard in Informix

- Add just a single shard

```
db.runCommand({"addShard":"hostname1:port1"})
```

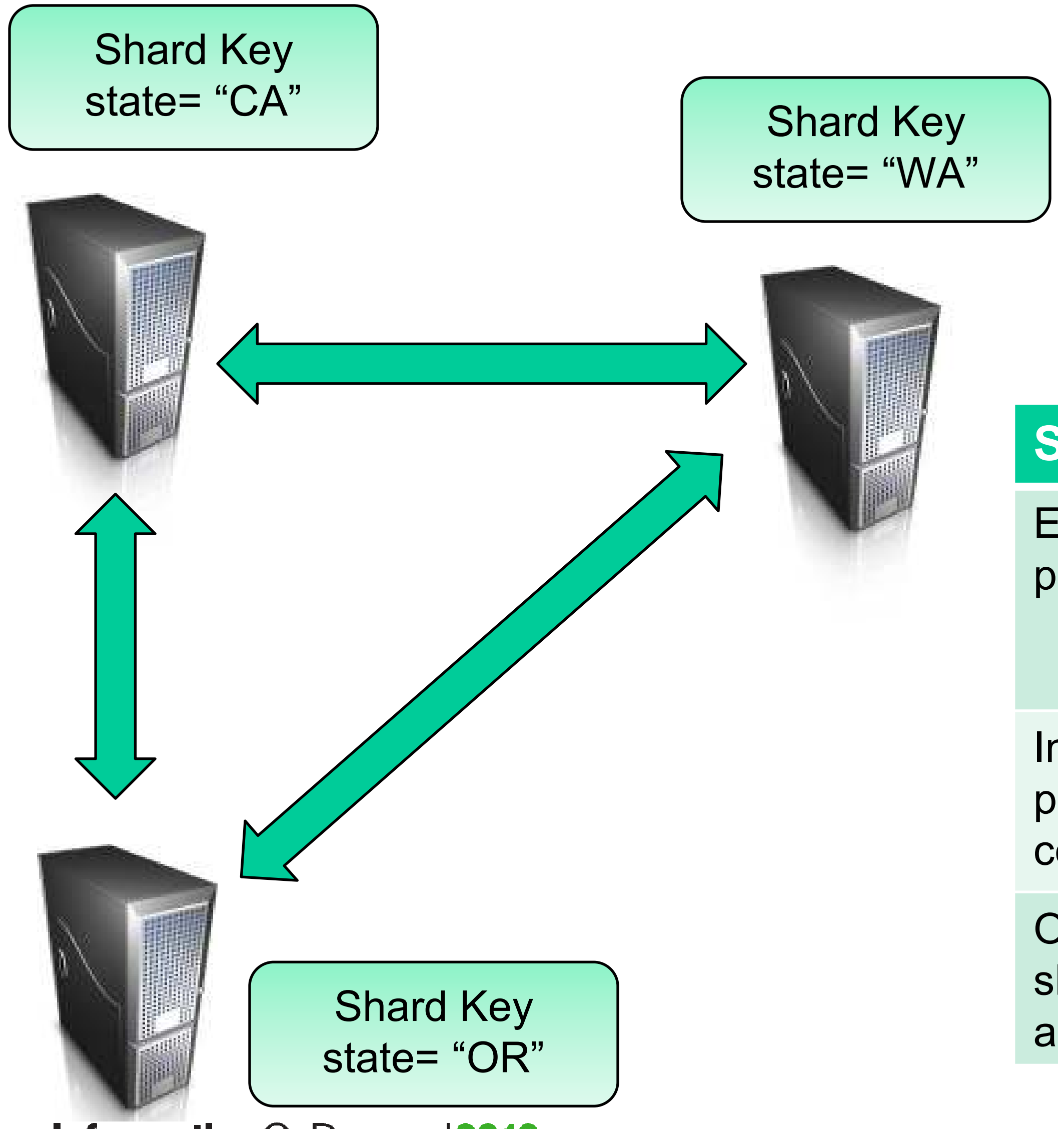
- Add multi shard in a single command
  - Informix only syntax

```
db.runCommand({"addShard":["hostname2:port2", "hostname3:port3",
"hostname4:port4"]})
```

- Shard the table phot_demo.photos by hash

```
sh.shardCollection("photo_demo.photos", {"_id": "hashed"})
```

Difference between Sharding Data VS Replication



Sharding	Replication
Each node holds a portion of the data • Hash • Expression	Same data on each node
Inserted data is placed on the correct node	Data is copied to all nodes
Operations are shipped to applicable nodes	Work on local copy and modification are propagated

Sharding is not for Data Availability

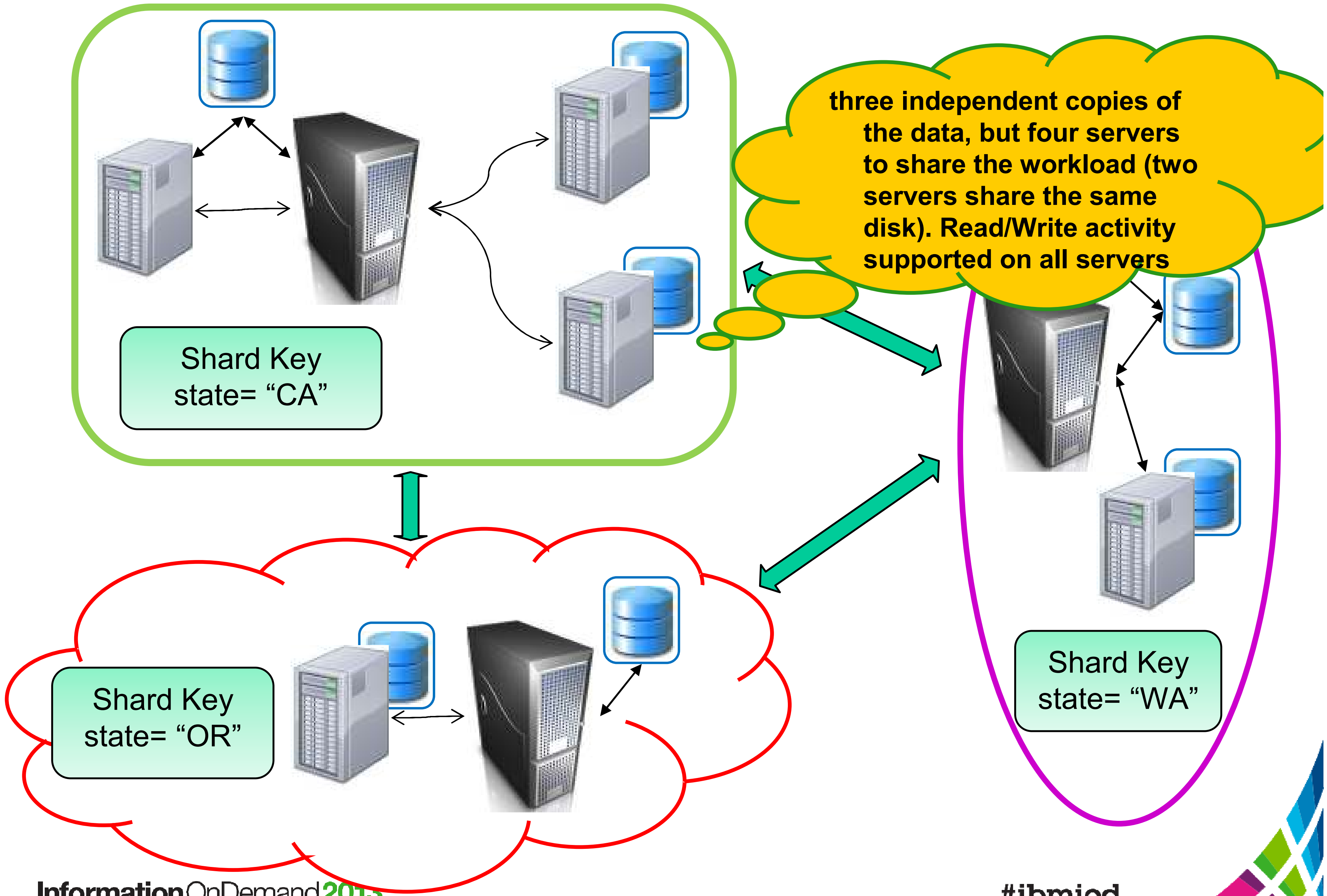
- Sharding is for growth, not availability
- Redundancy of a node provides high availability for the data
  - Both Mongo and Informix allow for multiple redundant nodes
  - Mongo refers to this as Replica Sets and the additional nodes slaves
  - Informix refers to this as H/A, and additional secondary nodes

Term	Description	Informix Term
Shard	A single node or a group of nodes holding the same data (replica set)	Instance
Replica Set	A collection of nodes contain the same data	HA Cluster
Shard Key	The field that dictates the distribution of the documents. Must always exist in a document.	Shard Key
Sharded Cluster	A group shards were each shard contains a portion of the data.	Grid/Region
Slave	A server which contains a second copy of the data for read only processing.	HA Secondary Server

Informix Secondary Servers

- Features of Informix secondary server:
  - Provide high availability
    - Can have one or more secondary servers
    - Synchronous or asynchronous secondary servers
    - Automatic promotion upon server failure
  - Scale out
    - Execute select
    - Allow Insert/Update/Deletes on the secondary servers
    - Secondary server can have their own disk or share disks with the master node
  - Connection manager routes users connection based on policies and server availability

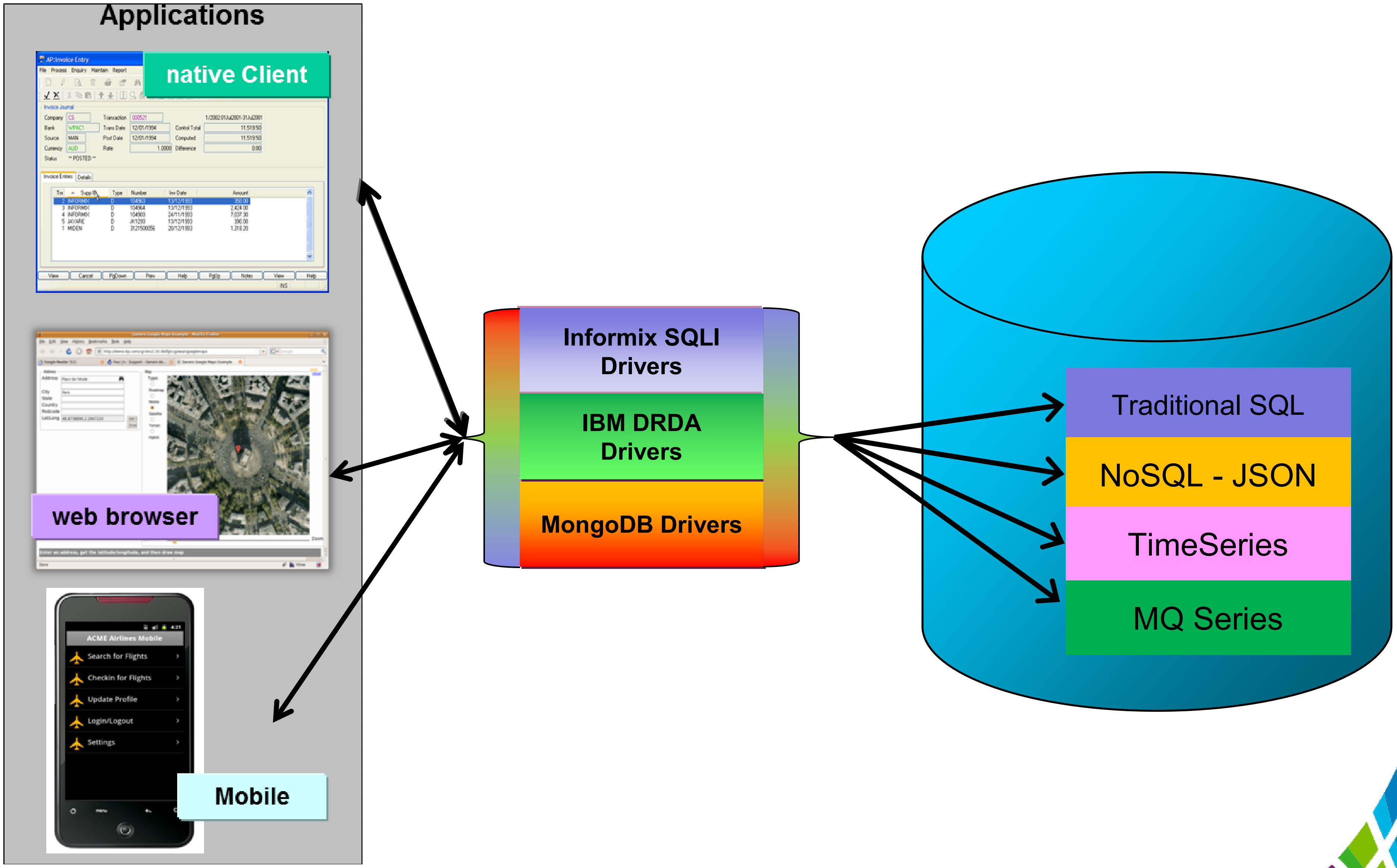
Informix NoSQL Cluster Architecture Overview


three independent copies of the data, but four servers to share the workload (two servers share the same disk). Read/Write activity supported on all servers
Shard Key state= “CA”
Shard Key state= “OR”
Shard Key state= “WA”

High Level Solution

- Flexible schema, native JSON and BSON
- Super scale out & elasticity
- Provide Mongo compatible programming
- Leveraging Informix for NoSQL

Ability for All Clients to Access All Data Models



Application Development Tools

The MEAN Stack

mongoDB

{ name: mongo, type: DB }

express

Web dev framework for NodeJS

ANGULARJS by Google

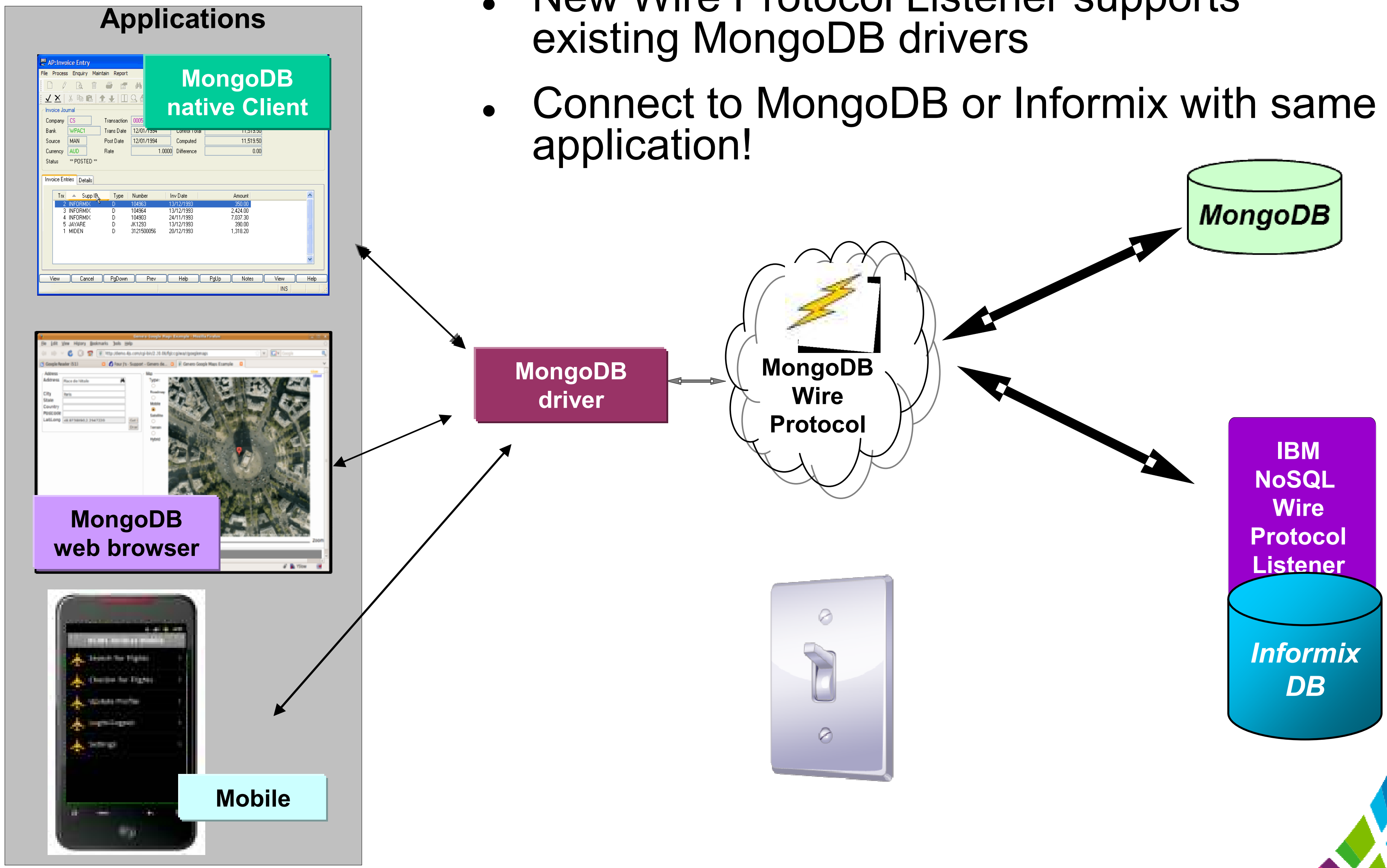
Superheroic frontend framework

node JS

Event-based concurrency environment

Client Applications

- New Wire Protocol Listener supports existing MongoDB drivers
- Connect to MongoDB or Informix with same application!



MongoDB Application Driver Compatibly

- Ability to use any of the MongoDB client drivers and frameworks against the Informix Database Server
  - Little to no change required when running MongoDB programs
  - Informix listens on the same default port as mongo, no need to change.

- Leverage the different programming languages available

All Support Languages	
C	Perl
C#	PHP
Erland	Python
Java	Ruby
JavaScript	Scala
Node.js	

- Other Community Drivers are also available

High Level Solution

- Flexible schema, native JSON and BSON
- Super scale out & elasticity
- Provide Mongo compatible programming
- Leveraging Informix for NoSQL

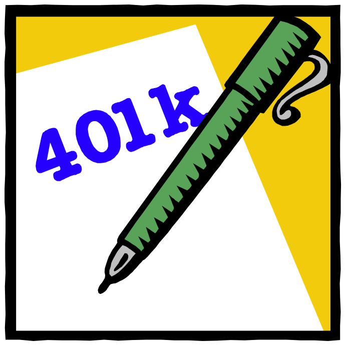
Hybrid Access between Relational & JSON Collections

	Relational Table	JSON Collections
SQL API	Standard ODBC, JDBC, .NET, OData, etc. Language SQL. ✓	?
MongoDB API (NoSQL)	?	Mongo APIs for Java, Javascript, C++, C#,... ✓

Why do you need hybrid access?

Data model should not restrict Data Access

Benefits of Simply Powerful

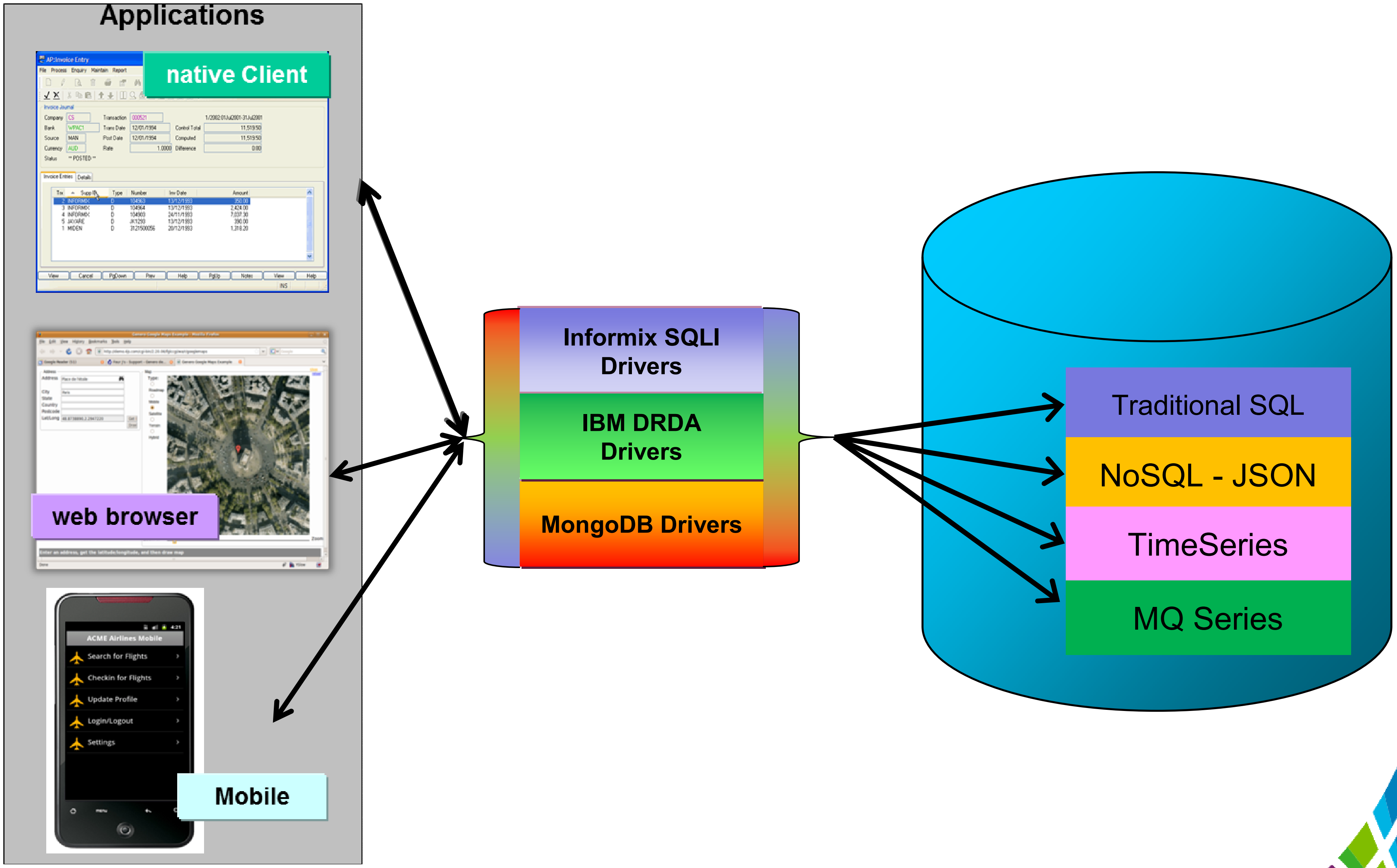


- Access consistent data from its source
- Avoid ETL, continuous data sync and conflicts.
- Exploit the power of SQL, MongoAPI seamlessly
- Exploit the power of RDBMS technologies in MongoAPI:
  - Informix Warehouse accelerator (Blu technologies)
  - Cost based Optimizer
  - R-tree indices for spatial, Lucene text indexes, and more.
- Access all your data thru any interface: MongoAPI or SQL.
- Store data in one place and efficiently transform and use them on demand.
- Existing SQL based tools and APIs can access new data in JSON

Hybrid Access between Relational & JSON Collections

	Relational Table	JSON Collections
SQL API	Standard ODBC, JDBC, .NET, OData, etc. Language SQL. ✓	Direct SQL Access. Dynamic Views Row types
MongoDB API (NoSQL)	Mongo APIs for Java, Javascript, C++, C#,...	Mongo APIs for Java, Javascript, C++, C#,... ✓

Ability for All Clients to Access All Data Models

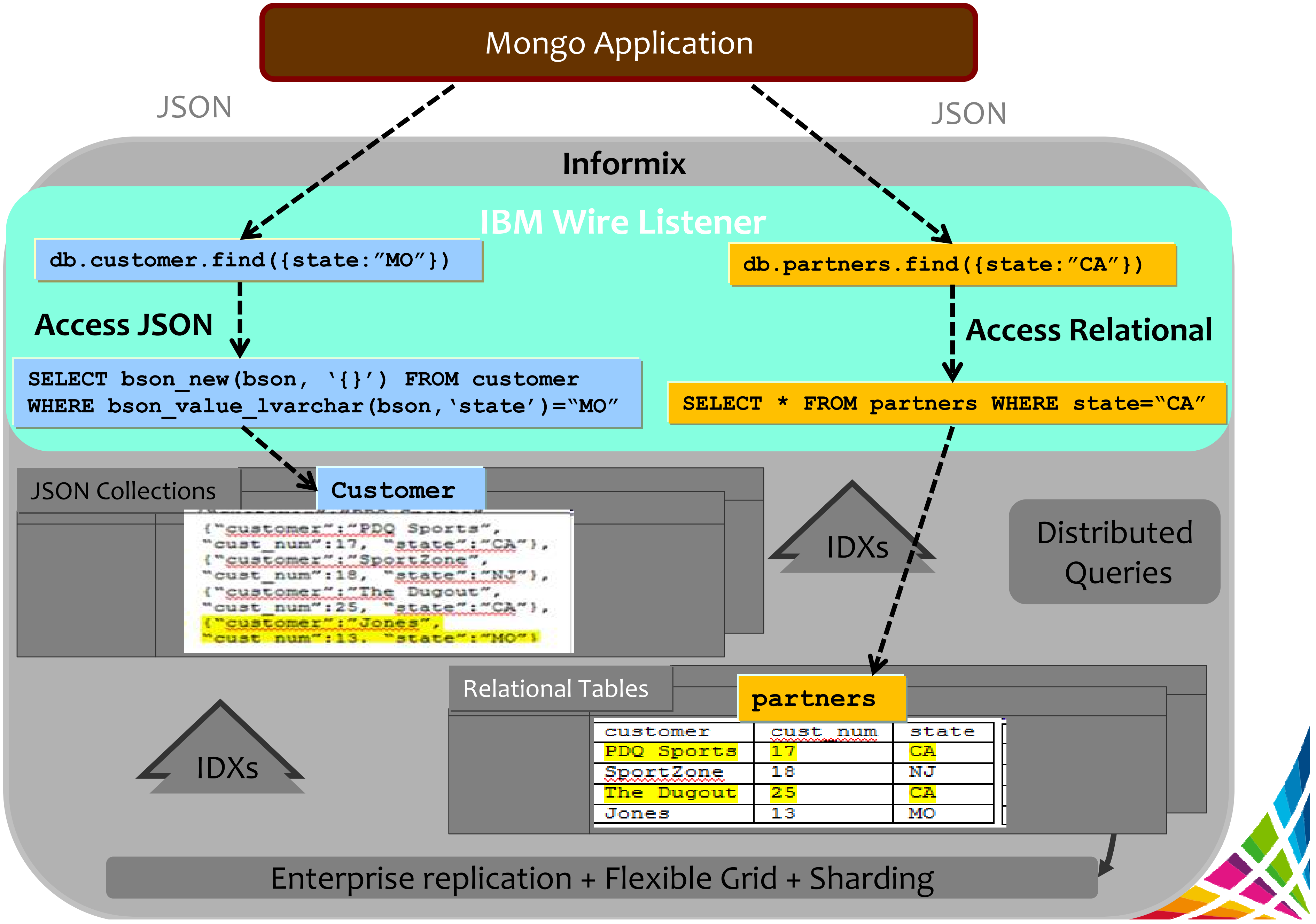


Hybrid access: From MongoAPI to relational tables.

You want to develop an application with MongoAPI, **but…**

1. You already have relational tables with data.
2. You have views on relational data
3. You need to join tables
4. You need queries with complex expressions. E.g. OLAP window functions.
5. You need multi-statement transactions
6. You need to exploit stored procedure
7. You need federated access to other data
8. You have timeseries data.

MongoAPI Accessing Both NoSQL and Relational Tables



```
db.customer.find({state:"MO"})
```

```
SELECT bson_new(bson, '{}') FROM customer
WHERE bson_value_lvarchar(bson,'state')="MO"
```

```
{"customer":"PDQ Sports",
"cust_num":17, "state":"CA"},
{"customer":"SportZone",
"cust_num":18, "state":"NJ"},
{"customer":"The Dugout",
"cust_num":25, "state":"CA"},
{"customer":"Jones",
"cust_num":13. "state":"MO"}
```

```
db.partners.find({state:"CA"})
```

```
SELECT * FROM partners WHERE state="CA"
```

customer	cust_num	state
PDQ Sports	17	CA
SportZone	18	NJ
The Dugout	25	CA
Jones	13	MO

How to Convert Relational Data as JSON Documents

- Relational data can be treated as structured JSON documents; column name-value becomes key-value pair.

➢ `SELECT partner, pnum, country from partners;`

```
 partner     pnum   Country
  Pronto     1748   Australia
  Kazer      1746   USA
   Diester   1472   Spain
   Consultix 1742   France
```

```
{parnter: “Pronot”, pnum:”1748”, Country: “Australia”}
{parnter: “Kazar”, pnum:”1746”, Country: “USA”}
{parnter: “Diester”, pnum:”1472”, Country: “Spain”}
{parnter: “Consultix”, pnum:”1742”, Country: “France”}
```

- Informix automatically translates the results of a relational query to JSON/BSON form.

MongoAPI Accessing Both NoSQL and Relational Tables

- Typically NoSQL does not involve transactions
  - In many cases, a document update is atomic, but not the application statement
  - Example
    - 7 targeted for deletion, but only 4 are removed
- Informix-NoSQL provides transactions on all application statements
  - Each server operation INSERT, UPDATE, DELETE, SELECT will automatically be committed after each operation.
  - In Informix there is away to create multi-statement transactions is to utilize a stored procedure
- Default isolation level is DIRTY READ
- All standard isolation level support

Accessing Data in Relational Tables

```
CREATE TABLE partners(pnum int, name varchar(32),
                      country varchar(32) );
```

```
db.partners.find({name:”Acme”}, {pnum:1, country:1});

SELECT pnum, country FROM partners WHERE name = “Acme”;
```

```
db.partners.find({name:”Acme”},
{pnum:1, country:1}).sort({b:1})

SELECT pnum,country FROM partners
WHERE name=“Acme” ORDER BY b ASC
```

Accessing data in relational tables.

```
db.partners.save({pnum:1632,name:"EuroTop",Country:"Belgium"});

INSERT into partners(pnum, name, country) values
                   (1632, "EuroTop", "Belgium");
```

```
db.partners.update({country:"Holland"},
          {$set:{country:"Netherland"}}, {multi: true});

UPDATE partners SET country = "Netherland"
                   WHERE country = "Holland";
```

```
db.partners.delete({name:"Artics"});

DELETE FROM PARTNERS WHERE name = "Artics";
```

Views and Joins

- Create a view between the existing *partner* table and a new *pcontact* table

```
create table pcontact(pnum int, name varchar(32), phone
varchar(32));

insert into pcontact values(1748,"Joe Smith","61-123-4821");
```

```
create view partnerphone(pname, pcontact, pphone) as select a.name,
b.name, b.phone FROM pcontact b left outer join partners a on
(a.pnum = b.pnum);
```

- Run the query across the view

```
db.partnerphone.find({pname:"Pronto"})

{ "pname":"Pronto", "pcontact":"Joe Smith", "pphone":"61-123-4821"}
```

Seamless federated access

1. `create database newdb2;`
2. `create synonym oldcontactreport for newdb:contactreport;`

```
> use newdb2
> db.oldcontactreport.find({pname:"Pronto"})
{ "pname" : "Pronto", "pcontact" : "Joel Garner", "totalcontacts" : 2 }
{ "pname" : "Pronto", "pcontact" : "Joe Smith", "totalcontacts" : 2 }
```

```
SELECT data FROM oldcontactreport WHERE
  bson_extract(data, 'pname') = “Pronto”;
```

- `create synonym oldcontactreport for custdb@nydb:contactreport;`

Get results from a stored procedure.

```
create function "keshav".p6() returns int, varchar(32);
define x int; define y varchar(32);
foreach cursor for select tabid, tabname into x,y from systables
        return x,y with resume;
end foreach;
end procedure;
create view "keshav".v6 (c1,c2) as
  select x0.c1 ,x0.c2 from table(function p6())x0(c1,c2);
```

- **db.v6.find().limit(5)**

```
{ "c1" : 1, "c2" : "systables" }
{ "c1" : 2, "c2" : "syscolumns" }
{ "c1" : 3, "c2" : "sysindices" }
{ "c1" : 4, "c2" : "systabauth" }
{ "c1" : 5, "c2" : "syscolauth" }
```

Access Timeseries data

```
create table daily_stocks
  ( stock_id integer,  stock_name lvarchar,
    stock_data timeseries(stock_bar)  );

-- Create virtual relational table on top (view)
EXECUTE PROCEDURE TSCreateVirtualTab('daily_stocks_virt',
'daily_stocks', 'calendar(daycal),origin(2011-01-03
  00:00:00.00000)' );
create table daily_stocks_virt
  ( stock_id integer,
    stock_name lvarchar,
    timestamp datetime year to fraction(5),
    high smallfloat,
    low smallfloat,
    final smallfloat,
    vol smallfloat );
```

Access Timeseries data

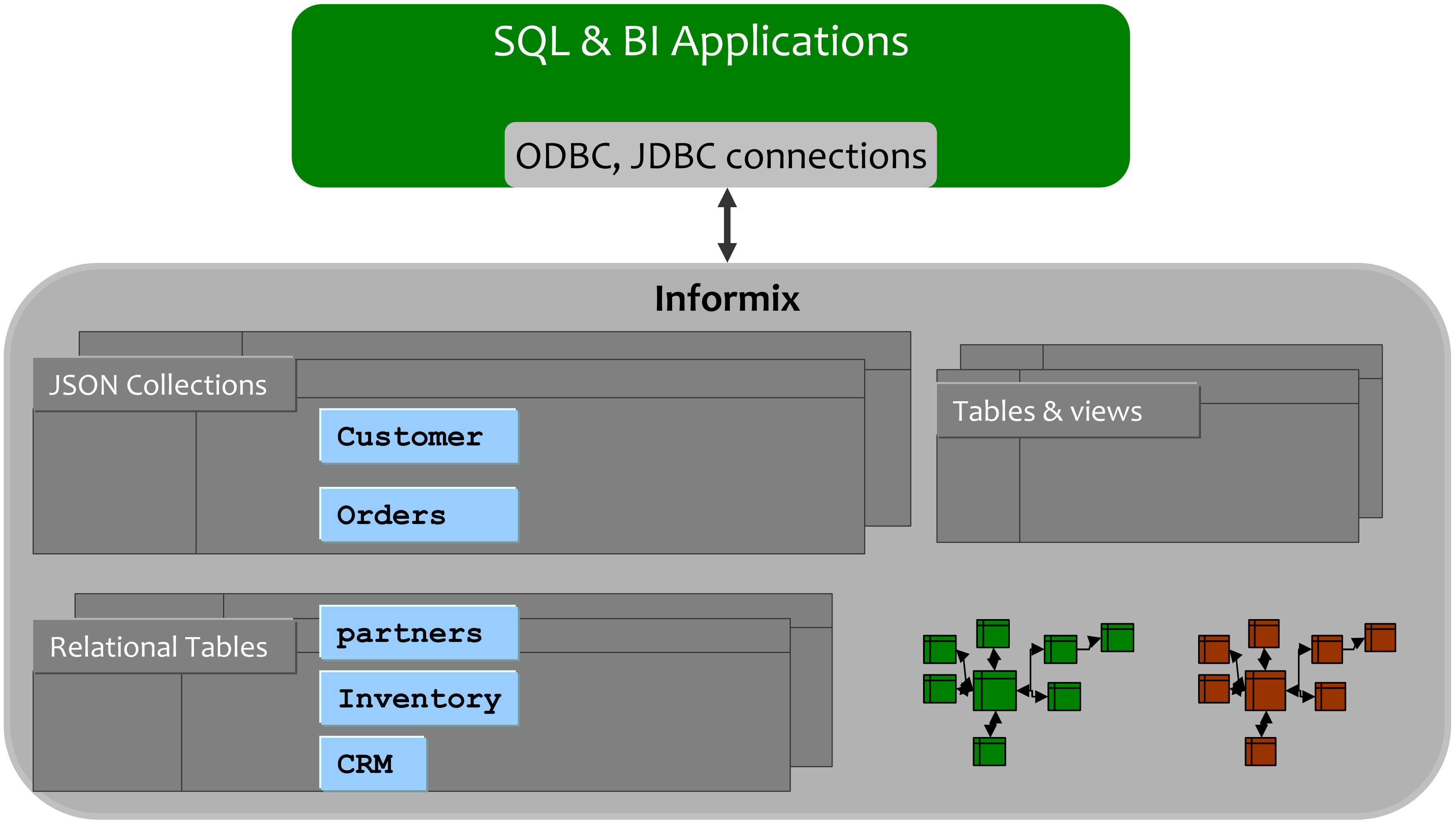
```
db.daily_stocks_virt.find({stock_name:"IBM"})
```

```
{ "stock_id" : 901, "stock_name" : "IBM", "timestamp" : ISODate("2011-01-03T06:0
0:00Z"), "high" : 356, "low" : 310, "final" : 340, "vol" : 999 }
{ "stock_id" : 901, "stock_name" : "IBM", "timestamp" : ISODate("2011-01-04T06:0
0:00Z"), "high" : 156, "low" : 110, "final" : 140, "vol" : 111 }
{ "stock_id" : 901, "stock_name" : "IBM", "timestamp" : ISODate("2011-01-06T06:0
0:00Z"), "high" : 99, "low" : 54, "final" : 66, "vol" : 888 }
```

You want to perform complex analytics on JSON data

- BI Tools like Cognos, Tableau generate SQL on data sources.
- Option 1: Do ETL
- Need to expose JSON data as views so it’s seen as a database object.
  - We use implicit casting to convert to compatible types
  - The references to non-existent key-value pair returns NULL
- Create any combination of views
  - A view per JSON collection
  - Multiple views per JSON collection
  - Views joining JSON collections, relational tables and views.
- Use these database objects to create reports, graphs, etc.

Analytics


SQL & BI Applications
ODBC, JDBC connections
Informix
JSON Collections
Customer
Orders
Tables & views
Relational Tables
partners
Inventory
CRM

Benefits of Hybrid Power

- ☑ Access consistent data from its source
- ☑ Avoid ETL, continuous data sync and conflicts.
- ☑ Exploit the power of SQL, MongoAPI seamlessly
- ☑ Exploit the power of RDBMS technologies in MongoAPI:
  - Informix Warehouse accelerator,
  - Cost based Optimizer & power of SQL
  - R-tree indices for spatial, Lucene text indexes, and more.
- ☑ Access all your data thru any interface: MongoAPI & SQL
- ☑ Store data in one place and efficiently transform and use them on demand.
- ☑ Existing SQL based tools and APIs can access new data in JSON

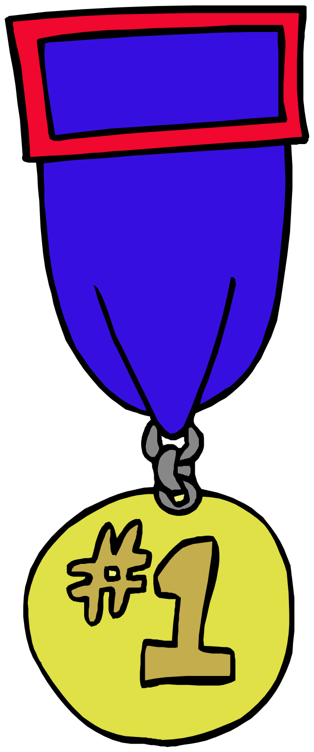
The Hybrid Solution
Informix has the Best of Both Worlds

- **Relational and non-relational data in one system**
- NoSQL/MongoDB Apps can access Informix Relational Tables
- Distributed Queries
- Multi-statement Transactions
- Enterprise Proven Reliability
- Enterprise Scalability
- Enterprise Level Availability

Informix provides the capability to leverage
the abilities of both relational DBMS and document store systems.

Informix Specific Advantages with Mongo Drivers

- Traditional SQL tables and JSON collections co-existing in the same database
- Using the MongoDB client drivers Query, insert, update, delete
  - JSON collections
  - Traditional SQL tables
  - Timeseries data
- Join SQL tables to JSON collections utilizing indexes
- Execute business logic in stored procedures
- Provide a view of JSON collections as a SQL table
  - Allows existing SQL tools to access JSON data
- Enterprise level functionality

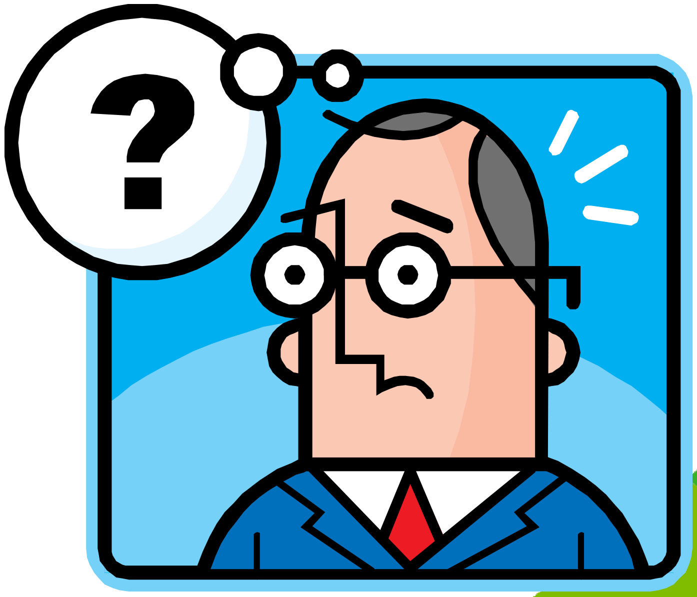


InformationOnDemand2013

November 3 – 7
Mandalay Bay | Las Vegas, NV

#ibmiod

Questions



IBM