IBM

The Power of Hybrid – Informix & JSON

Scott Pickett
WW Informix Lab Services
For questions about this presentation, email to: spickett@us.ibm.com

© 2021 IBM Corporation

Agenda

- The Power of Hybrid – Informix and JSON
- Informix & JSON: Use Cases
- Informix & JSON: "Nothing Like the Present"
- NoSQL – Feature Checklist
- The Wire Listener
- Sharding
- MongoDB Utilities & Informix
- Useful Informix Capabilities Over Collections
- Informix NoSQL Aggregation Framework
- MongoDB Application Source Drivers & Informix
- Additional JSON Enhancements
- MQTT & Informix
- Appendicies

© 2021 IBM Corporation

The Power of Hybrid – Informix and JSON

© 2021 IBM Corporation

Why JSON

- **Rapid Application Prototyping**
  - Schema rides as objects inside the JSON Script, not as a separate item. Allows for constant unit test and retest as logic expands
    - Schemas change quite quickly in modern apps as business needs change.
    - Apps have shorter production lives and even shorter life-cycles as a result
    - Knows and understands JSON natively.
    - No need to normalize per se.
    - 1 to 1 and many to 1 RDBMS relationships do not apply here.
  - Less bureaucratic
    - Stripped down IS organizations, lean and mean
    - Programmer can be the DBA, system administrator as well.
- **Scalability**
  - Scalability is not by database tuning per se and adding resources, but rather by sharding (fragmenting the data by a key) the data when necessary across cheap, inexpensive commodity network hardware
    - Automatic data rebalancing across the shards by key occurs during the shard operation.
- **Costs of initial "up and running" are less**
- **Quicker project completion times**

© 2021 IBM Corporation

Informix as a Hybrid Data Storage Database

- **Many of NoSQL customers are enterprise that have both relational and non-relational data sitting in the same facility and need to make sense of the data they have for business purposes.**

- **Some NoSQL databases have no transaction capabilities in the SQL sense**
  - Informix will enforce transactions on all application statements, with consistency assured at the end of the transaction
  - Informix is transactional, encrypted, and handles SQL and NoSQL data in the same instance, all at once

© 2017 IBM Corporation

Informix as a Hybrid Data Storage Database

- **Joining noSQL and SQL based queries in the same statement is possible within Informix**
  - Many customers using NoSQL in production circumstances with large document store databases need to join this data to their **existing** SQL based database repositories for business purposes, often stored, housed and queried separately:
    - Informix can join the two kinds of data
      - Document store portion stored natively in a BSON as a single row data type within Informix, accessed as JSON, with a set full set of functions to access the document store data.

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Informix & JSON: Use Cases

© 2021 IBM Corporation

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

© 2017 IBM Corporation

Typical NoSQL Use Cases - Interactive Web Mobile

- **Online/Mobile Gaming**
  - Leaderboard (high score table) management
  - Dynamic placement of visual elements
  - Game object management
  - Persisting game/user state information
  - Persisting user generated data (e.g. drawings)
- **Display Advertising on Web Sites**
  - Ad Serving: match content with profile and present
  - Real-time bidding: match cookie profile with ad inventory, obtain bids, and present ad
- **Communications**
  - Device provisioning
- **Social Networking/Online Communities**
- **E-commerce/Social Commerce**
  - Storing frequently changing product catalogs
- **Logging/message passing**
  - Drop Copy service in Financial Services (streaming copies of trade execution messages into (for example) a risk or back office system)
- **Dynamic Content Management and Publishing (News & Media)**
  - Store content from distributed authors, with fast retrieval and placement
  - Manage changing layouts and user generated content

© 2017 IBM Corporation

Why NoSQL?

- **Non-traditional data management requirements driven by Web 2.0 companies**
  – Document stores, key-value stores, graph and columnar DBMS
- **The Three Vs:**
  – Velocity – high frequency of data arrivals
  – Volume – BigData
  – Variability – unstructured data, flexibility in schema design

- **New data interchange formats**
  – JSON (JavaScript Object Notation)
  – BSON (Binary JSON)
  – Both implemented to industry standards
- **Scale-out requirements across heterogeneous environment**
  – Cloud computing

© 2017 IBM Corporation

IBM

What is a NoSQL Database?

- **Not Only SQL or NOt allowing SQL**
- **A non-relational database management systems**
  - Does not require a fixed schema
  - Avoids join operations
  - Scales horizontally
  - No ACID (eventually consistent)
- **Good with distributing data and fast application development**

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

© 2017 IBM Corporation

IBM Use Case Characteristics for JSON

Characteristic	Details
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,…) • Commonly available hardware (Windows & Linux,…)
Reduced administration and maintenance	• Easy of deployment • Install, configure add to exiting environment in minutes

© 2017 IBM Corporation

Why Most Commercial Relational Databases cannot meet these Requirements

Requirement	Why RDBMS falls short
Schema flexibility and development agility	•Relational schemas are inherently rigid •Database design needs to be done before application is developed •Different rows cannot have a different structures
Dynamic elasticity	•Not a natural fit for RDBMS, due to requirement for strong consistency •Scale-out requires partition tolerance, that increases latency
Continuous availability	•Requirement can be met, but at a significant cost •Typically rolling version upgrades are complex in clustered RDBMS
Consistent low latency, even under high loads	•ACID requirements inherently introduce write latency •There is no latency-consistency tradeoff knobs available •Possible, but at a much higher cost (hardware, software or complexity)
Low cost infrastructure	•Distributed RDBMS typically require specialized hardware to achieve performance •Popular relational databases typically require several DBAs for maintenance and tuning
Reduced administration and maintenance	•Often requires a significant DBA resources

© 2017 IBM Corporation

NoSQL Database Philosophy Differences

- **No ACID**
  - No ACID (Atomicity, Consistency, Isolation, Durability)
  - An eventual consistence model
- **No Joins**
  - Generally single row/document lookups
- **Flexible Schema**
  - Rigid format

© 2017 IBM Corporation

Informix NoSQL Key Messages

- **Mobile computing represents one of the greatest opportunities for organizations to expand their business, requiring mobile apps to have access to all relevant data**
- **Informix has embraced the JSON standard and up to version 4.2 of the MongoDB API to give mobile app developers the ability to combine critical data managed by Informix with the next generation mobile apps**
- **Informix JSON lets developers leverage the abilities of relational DBMS together with document store systems**

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Informix & JSON: Nothing Like the Present ……..

© 2021 IBM Corporation

Informix NoSQL High Level Functionality Comparison (1)

- SQL Data Native Types
- TimeSeries Native Type
- JSON & BSON Native Types
- 2 GB Maximum Document Size
- Extended JSON Date Support
- Hybrid Storage
- Hybrid Applications
- Sharding
- Type-less JSON Indexes
- Typed JSON Indexes
- Hybrid Storage as native types
- REST Compatible
- Intelligent Buffer Pool & Read Ahead
- Single Statement Transactions
- Multi-Statement Transactions
- Partial-Statement Transactions
- Row Lock Granularity
- Join Collections to SQL Tables
- Join Collections to Collections
- Join Collections to TimeSeries
- SPL Execution w/JSON & SQL Types
- Text Search
- Table & Row Level Auditing
- Data Compression
- Web Based Graphical Admin

© 2021 IBM Corporation

Informix NoSQL High Level Functionality Comparison (2)

- Cost Based Optimizer
- Run SQL Analytics Tools (i.e. Cognos) on JSON Document and SQL Tables
- BLU Acceleration on JSON data
- OLAP analytics on JSON Collections
- Distributed Access to Tables or Collections
- Triggers on Collections or Tables
- Truncate Collections or Tables
- Enterprise Level Security
- Online Index Maintenance
- Native JSON Command Line Processing
- Rolling Upgrades
- Single Point in Time Online Backups
- Read-Only Secondary/Slave Servers
- Read-Write Secondary/Slave Servers
- Diskless Secondary/Slave Servers
- Multiple Secondary/Slave Servers
- Gridded Tables or Collections
- Policy Based Failover
- Automatic Connection Re-Direction
- Shard Tables or Collections
- MQTT Supported
- Encryption at Rest
- Heterogenous Table Based Replication

© 2021 IBM Corporation

Informix Security

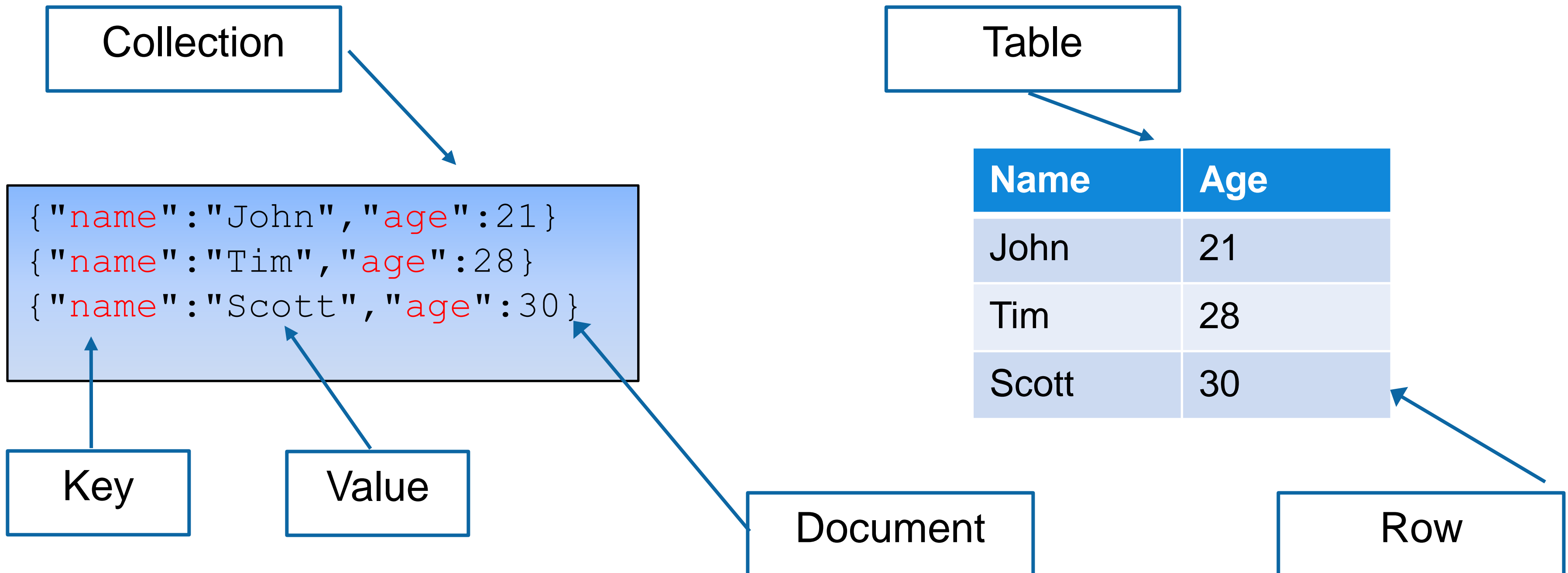
- **Encryption**
  - At rest
  - On the wires
  - Server to server
  - Client to server
  - Backups
  - Replication servers

- **Authentication via O/S**
  - Don't forget PAM for two factor authentication
- **Managed users**
- **Off server security**
- **Roles**
- **DBSA**
- **Security directories**
  - Install

© 2021 IBM Corporation

Basic Translation Terms/Concepts

Mongo/NoSQL Terms	Traditional SQL Terms
Database	Database
Collection	Table
Document	Row
Field	Column



© 2017 IBM Corporation

The SQL to Create a Collection

- **Formal definition of a collection used by the Wire Listener**

```
CREATE COLLECTION TABLE mycollection
                (
                id          char(128),
                modcnt      integer,
                data        bson,
                flags       integer
                );
```

- **All collection tables in Informix that can hold JSON documents, have the same 4 basic column names and data type definitions:**
  - Only the collection table name changes.
  - Standard Table DDL subclauses apply (lock mode, extent size, compressed, etc.)
- **JSON documents are stored whole in binary format in the column '*data*'**
- **Queries from both JSON and SQL can access this data.**

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

NoSQL – Feature Checklist

© 2021 IBM Corporation

JSON Features in Informix

Flexible Schema	▪ Use BSON and JSON data type. ▪ Complete row is stored in a single column ▪ BSON, JSON are multi-rep types and can store up to 2GB.
Access Key Value Pairs within JSON	▪ Translate the Mongo queries into SQL expressions to access key-value pairs. ▪ Informix has added expressions to extract specific KV pairs. ▪ Select bson_new(data, "{id:1, name:1, addr:1}") from tab;
Indexing	▪ Support standard B-tree index via functional indexing. ▪ Type Less Index → ▪ Create index ix_1 on t(bson_extract(data, "id"); ▪ Typed Index → Create index ix_2 on t(bson_value_lvarchar(data, "name"), bson_value_date(data, "expire"); ▪ Informix supports indexing bson keys with different data types.
Sharding	▪ Supports range & hash based sharding. ▪ Informix has built-in technology for replication. ▪ Create identical tables in multiple nodes. ▪ Add meta data for partitioning the data across based on range or hash expressions.

© 2017 IBM Corporation

JSON Features in Informix

Select	▪ Limited support required for sharding, Mongo API disallowing joins. ▪ The query on a single sharded table is transformed into a federated UNION ALL query; includes shard elimination processing.
Updates (single node)	▪ INSERT: Simple direct insert. ▪ DELETE: DELETE statement with ▪ WHERE bson_extract() > ?; or bson_value..() > ? ▪ UPDATE: ▪ bson_update(bson, “update expr”) will return a new bson after applying the bson expression. Simple updates to non_sharded tables will be direct UPDATE statement in a single transaction. ▪ UPDATE tab bsoncol = bson_update(bsoncol, “expr”) where …
Updates (sharded env)	▪ **INSERT** – All rows are inserted to LOCAL shard, replication threads will read logs and replicate to the right shard and delete from local node (log only inserts underway). ▪ **DELETE** – Do the local delete and replicate the “delete statements” to target node in the background ▪ **UPDATE** – Slow update via select-delete-insert.
Transaction	▪ Each statement is a distinct transaction in a single node environment. ▪ The data and the operation is replicated via enterprise replication.

© 2017 IBM Corporation

JSON Features in Informix

Isolation levels	▪ NoSQL session can use any of Informix isolation levels.
Locking	▪ **Application controls only on the node they're connected to.** ▪ **Standard Informix locking semantics will apply when data is replicated and applied on the target shard.**
Hybrid Access **(From MongoAPI to relational tables)**	▪ **INSERT** – All rows are inserted to LOCAL shard, replication threads will read logs and replicate to the right shard and delete from local node (log only inserts underway). ▪ **DELETE** – Do the local delete and replicate the "delete statements" to target node in the background ▪ **UPDATE** – Slow update via select-delete-insert.
Hybrid Access **(From SQL to JSON data)**	▪ **Directly get binary BSON or cast to JSON to get in textual form.** ▪ **Use expressions to extract to extract specific key-value pairs.** NoSQL collections will only have one BSON object in the table. We can "imply" the expressions when the SQL refers to a column. **SELECT t.c1, t.c2 from t; ➔ SELECT bson_extract(t.data, "{c1:1}"), bson_extract(t.data, "{c2:1}") from t;**

© 2017 IBM Corporation

Flexible Schema

- **Clients exchange BSON document with the server both for queries & data.**
- **Thus, BSON becomes a fundamental data type.**
- **The explicit key-value(KV) pairs within the JSON/BSON document will be roughly equivalent to columns in relational tables.**

- **However, there are differences!**
  - The type of the KV data encoded within BSON is determined by the **client**
  - Server is unaware of data type of each KV pair at table definition time.
    - Application must keep track of the data types of the data
  - 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.
  - Due to the limitations of Mongo/NoSQL API, customers typically de-normalize the tables
    - (customer will have customer+customer addr + customer demographics/etc) to avoid joins.

© 2017 IBM Corporation

Flexible Schema – Informix Implementation

- **Informix has a new data type, BSON, to store the data.**
- **Informix also has a JSON data type to convert between binary and text form.**
- **BSON and JSON are abstract data types (like spatial, etc).**
- **BSON and JSON multi-representational types:**
  - Objects up to 4K is stored in data pages.
  - Larger objects (up to 2GB) are stored out of row, in BLOBs.
    - MongoDB limits objects to 16MB
- **This is all seamless and transparent to applications.**

© 2017 IBM Corporation

Flexible Schema – Informix Implementation

- **CREATE TABLE customer (data BSON)**
- **BSON is the binary representation of JSON.**
  - It has length and types of the key-value pairs in JSON.
- **MongoDB drivers send and receive in BSON form.**

```
{"hello":            →   "\x16\x00\x00\x00\x02hello\x00
"world"}                  \x06\x00\x00\x00world\x00\x00"

                         "\x31\x00\x00\x00\x04BSON\x00\x26\x00
{"BSON":                  \x00\x00\x020\x00\x08\x00\x00
["awesome",          →    \x00awesome\x00\x011\x00\x33\x33\x33\x33\x33\x33
5.05, 1986]}              \x14\x40\x102\x00\xc2\x07\x00\x00
                          \x00\x00"
```


© 2017 IBM Corporation

Accessing KV pairs within JSON

- Extract Expressions/functions returning base type

  bson_value_bigint(BSON, “key”);
  bson_value_lvarchar(bsoncol, “key.key2”);
  bson_value_date(bsoncol, “key.key2.key3”);
  bson_value_timestamp(bsoncol, “key”)
  bson_value_double(bsoncol, “key”);
  bson_value_boolean(bsoncol, “key”);
  bson_value_array(bsoncol, “key”);
  bson_keys_exist(bsoncol, “key”);
  bson_value_document(bsoncol, “key”)
  bson_value_binary(bsoncol, “key”)
  bson_value_objectid(bsoncol, “key”)

- Expression returning BSON subset. Used for bson indices.

  bson_extract(bsoncol, “projection specification”)

- Expressions to project out of SELECT statement.

  bson_new(bsoncol, “{key1:1, key2:1, key3:1}”);
  bson_new(bsoncol, “{key5:0}”);

© 2017 IBM Corporation

Accessing Key-Value (KV) pairs within JSON.

Mongo Query	SQL Query
db.customers.find();	SELECT data::json FROM customers
db.customers.find({},{num:1,name:1});	SELECT bson_new(data, '{ "num" : 1.0 , "name" : 1.0}')::bson::json FROM customers
db.customers.find({}, {_id:0,num:1,name:1});	SELECT bson_new(data, '{_id:0.0, "num" : 1.0 , "name" : 1.0}')::bson::json FROM customers
db.customers.find({status:"A"})	SELECT data::json FROM customers WHERE bson_value_lvarchar(data,"status")= "A"
db.customers.find({status:"A"}, {_id:0,num:1,name:1});	SELECT bson_new(data, '{ "_id" : 0.0 , "num" : 1.0 , "name" : 1.0}')::bson::json FROM customers WHERE bson_extract(data, 'name')::bson::json::lvarchar = '{"status":"A"}'

© 2017 IBM Corporation

IBM

Indexing (1)

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

© 2017 IBM Corporation

Indexing (2)

Mongo Index Creation Statement	SQL Query Statement
db.customers.ensureIndex({orderDate:1})	CREATE INDEX IF NOT EXISTS w_x_1 ON w (bson_extract(data,'x') ASC) using bson (ns='{ "name" : "newdb.w.$x_1"}', idx='{ "ns" : "newdb.w" , "key" : {"x" : [1.0 , "$extract"]} , "name" : "x_1" , "index" : "w_x_1"}') EXTENT SIZE 64 NEXT SIZE 64
db.customers.ensureIndex({orderDate:1, zip:-1})	CREATE INDEX IF NOT EXISTS v_c1_1_c2__1 ON v (bson_extract(data,'c1') ASC, bson_extract(data,'c2') DESC) using bson (ns='{ "name" : "newdb.v.$c1_1_c2__1"}', idx='{ "ns" : "newdb.v" , "key" : { "c1" : [1.0, "$extract"] , "c2" : [-1.0 , "$extract"]} , "name" : "c1_1_c2__1" , "index" : "v_c1_1_c2__1"}') EXTENT SIZE 64 NEXT SIZE 64
db.customers.ensureIndex({orderDate:1}, {unique:true)	CREATE UNIQUE INDEX IF NOT EXISTS v_c1_1_c2__1 ON v (bson_extract(data,'c1') ASC, bson_extract(data,'c2') DESC) using bson (ns='{ "name" : "newdb.v.$c1_1_c2__1"}', idx='{ "ns" : "newdb.v" , "key" : { "c1" : [1.0 , "$extract"] , "c2" : [-1.0 , "$extract"]} , "name" :"c1_1_c2__1" , "unique" : true , "index" : "v_c1_1_c2__1"}') EXTENT SIZE 64 NEXT SIZE 64

© 2017 IBM Corporation

Indexing - Examples

- **Mongo Index Creation Statement**

```
> mongo
MongoDB shell version: 2.4.9
connecting to: test
> use newdb
switched to db newdb
> db.customers.ensureIndex({orderdate:1})
```

- **Informix SQL Query Statement**

```
create index customers_orderdate_1 on customers (bson_extract(data, 'orderdate') )
using bson (ns='{ "name" : "newdb.customers.$orderdate_1"}',
     idx='IfxDBObject { "ns" : "newdb.customers" ,
                       "key" : {"orderdate" : [ 1.0 , "$extract"]} ,
                       "name" : "orderdate_1" ,
                       "index" : "customers_orderdate_1"}'
);
```


© 2017 IBM Corporation

Indexing - Examples

- **Mongo Index Creation Statement**

```
> mongo
MongoDB shell version: 2.4.9
connecting to: test
> use newdb
switched to db newdb
> db.customers.ensureIndex({orderdate:1,zip:-1})
```

- **Informix SQL Query Statement**

```
create index customers_orderdate_1_zip__1 on customers (
  bson_extract(data,'orderdate') ,bson_extract(data,'zip')  desc  )
using bson (ns='{ "name" : "newdb.customers.$orderdate_1_zip__1"}',
idx='IfxDBObject { "ns" : "newdb.customers" ,
                         "key" : { "orderdate" : [ 1.0 , "$extract"] , "zip" : [ -1.0 , "$extract"]} ,
                         "name" : "orderdate_1_zip__1" ,
                         "index" : "customers_orderdate_1_zip__1"}'
);
```


© 2017 IBM Corporation

Indexing - Examples

- **Mongo Index Creation Statement**

```
> mongo
MongoDB shell version: 2.4.9
connecting to: test
> use newdb
switched to db newdb
> db.customers.ensureIndex({orderdate:1}, {unique:true})
```

- **Informix SQL Query Statement**

```
create unique index customers_orderdate_1 on customers (bson_extract(data,'orderdate') )
using bson (ns='{ "name" : "newdb.customers.$orderdate_1"}', idx=
'IfxDBObject { "ns" : "newdb.customers" ,
               "key" : { "orderdate" : [ 1.0 , "$extract"]} ,
               "name" : "orderdate_1" , "unique" : true ,
               "index" : "customers_orderdate_1"}'
);
```


© 2017 IBM Corporation

Indexing

db.w.find({x:1,z:44},{x:1,y:1,z:1})

- Translates to:

```
SELECT  bson_new( data, '{ "x" : 1.0 , "y" : 1.0 , "z" : 1}
   FROM w
 WHERE ( bson_extract(data, 'x') = '{ "x" : 1.0 }'::json::bson ) AND
            ( bson_extract(data, 'z') = '{ "z" : 44.0 }'::json::bson )
```

Estimated Cost: 2
Estimated # of Rows Returned: 1

1) ifxjson.w: SEQUENTIAL SCAN

Filters: (informix.equal(informix.bson_extract(ifxjson.w.data ,'z'),UDT)
AND (informix.equal(informix.bson_extract(ifxjson.w.data ,'x'),UDT))

© 2017 IBM Corporation

Indexing

- **Creating Mongo index ….**

```
db.w.ensureIndex({x:1,z:1})
```

- **…. Informix functional Index is built on bson expressions**

```
 create index w_x_1_z_1 on w (.bson_extract(data, 'x') ,bson_extract(data,'z') )
 using bson (ns='{ "name" : "newdb.w.$x_1_z_1"}',
                idx='IfxDBObject { "ns" : "newdb" ,
                               "key" : { "x" : [ 1.0 , "$extract"] , "z" : [ 1.0 , "$extract"]} ,
                               "name" : "x_1_z_1" ,
                               "index" : "w_x_1_z_1"}'
);
```

- **Listener is aware of the available index and therefore generates right predicates**

© 2017 IBM Corporation

Indexing

- Functional Index is built on bson expressions

```
CREATE INDEX IF NOT EXISTS w_x_1 ON
        w (bson_extract(data,'x') ASC) using bson
        (ns='{ "name" :  "newdb.w.$x_1"}',
        idx='{ "ns" : "newdb.w" ,
        "key" : {"x" : [ 1.0 , "$extract"]} ,
        "name" : "x_1" , "index" : "w_x_1"}')
        EXTENT SIZE 64 NEXT SIZE 64
```

- Listener is aware of the available index and therefore generates right predicates:

```
db.w.find({x:1});
```

gets translated to

```
SELECT SKIP ? data FROM w WHERE bson_extract(data, 'x') = ?
```


© 2017 IBM Corporation

Indexing - sqexplain

```
db.w.find({x:5,z:5}, {x:1,y:1,z:1})

Translates to:

SELECT bson_new( data, ‘ifxDBObject{ "x" : 1.0 , "y" : 1.0 , "z" : 1 .0}'):bson
  FROM w
 WHERE ( bson_extract(data, 'x') = '{ "x" : 5.0 }'::json::bson )
     AND ( bson_extract(data, 'z') = '{ "z" : 5.0 }'::json::bson )

Estimated Cost: 1
Estimated # of Rows Returned: 1

  1) ifxjson.w: INDEX PATH

    (1) Index Name: ifxjson.w_x_1_Z_1
      Index Keys: informix.bson_extract(data,'x') informix.bson_extract(data,'z')
(Serial, fragments: ALL)
      Lower Index Filter: (informix.equal(informix.bson_extract(ifxjson.w.data
,'z' ),UDT )
       AND informix.equal(informix.bson_extract(ifxjson.w.data ,'x' ),UDT ))
```


© 2017 IBM Corporation

Transactions - Nosql Informix extension

- **Informix Nosql transactional syntax for non-sharded queries:**
  - **enable**
    - Enable transaction mode for the current session in the current database
      - **db.runCommand({transaction: “enable” }).**
  - **disable**
    - Disable transaction mode for the current session in the current database.
      - **db.runCommand({transaction: “disable” }).**
  - **status**
    - Print status information to indicate whether transaction mode is enabled and if transactions are supported by the current database.
      - **db.runCommand({transaction: “status” }).**
  - **commit**
    - If transactions are enabled, commits the current transaction. If transactions are disabled, an error is shown.
      - **db.runCommand({transaction: “commit” }).**
  - **rollback**
    - If transactions are enabled, rolls back the current transaction. If transactions are disabled, an error is shown.
      - **db.runCommand({transaction: “rollback” }).**

© 2017 IBM Corporation

Transactions - Nosql Informix extension

- **execute**
  - This optional parameter runs a batch of commands as a single transaction.
  - If transaction mode is not enabled for the session, this parameter enables transaction mode for the duration of the transaction.
  - Command documents include **insert**, **update**, **delete**, **findAndModify**, and **find**
    - **insert**, **update**, and **delete** command documents, you cannot set the **ordered** property to **false**.
    - Use a **find** command document to run SQL & NoSQL queries, but not commands.
    - A **find** command document can include the **$orderby, limit, skip**, and **sort** operators.
    - The following example deletes a document from the inventory collection and inserts documents into the archive collection:

```
db.runCommand({"transaction" : "execute", "commands" : [ {"delete":"inventory",
"deletes" : [ { "q" : { "_id" : 432432 } } ] }, {"insert" : "archive", "documents" : [ {
"_id": 432432, "name" : "apollo", "last_status" : 9} ] } ] })
```

  - The **execute** parameter has an optional **finally** argument if you have a set of command documents to run at the end of the transaction regardless of whether the transaction is successful.

© 2017 IBM Corporation

Transactions - Nosql Informix extension

- **Example:**

  db.runCommand({"transaction" : "execute", "commands" : [{"find" : "system.sql", "filter" : {"$sql" : "SET ENVIRONMENT USE_DWA 'ACCELERATE ON'" } }, {"find" : "system.sql", "filter" : {"$sql" : "SELECT SUM(s.amount) as sum FROM sales AS s WHERE s.prid = 100 GROUP BY s.zip" } }], "finally" : [{"find":"system.sql", "filter" : {"$sql" : "SET ENVIRONMENT USE_DWA 'ACCELERATE OFF'" } }] })

- **All transaction commands for Informix work on non-sharded servers only.**

© 2017 IBM Corporation

Transactions - Nosql Informix extension (1)

```
switched to db stores_demo
mongos> db.runCommand({transaction: "enable"})
{ "ok" : 1 }
mongos> db.runCommand({transaction: "status"})
{ "enabled" : true, "supported" : true, "ok" : 1 }
mongos> db.city_info2.insert({ "city" : "RAINBOWVILLE", "loc" : [-99.999999, 40.012343 ], "pop" :
9999, "state" : "CA" })
mongos> db.runCommand({transaction: "status"})
{ "enabled" : true, "supported" : true, "ok" : 1 }
mongos> db.runCommand({transaction: "commit"})
{ "ok" : 1 }
mongos> db.city_info2.insert({ "city" : "RAINBOWVILLE", "loc" : [-99.999999, 40.012343 ], "pop" :
9999, "state" : "MA" })
mongos> db.city_info2.find( { city: { $eq: "RAINBOWVILLE" } } )
{ "code" : null, "ok" : 0, "errmsg" : "invalid operator: $eq" }
mongos> db.city_info2.find( { city: "RAINBOWVILLE" } )
{ "_id" : ObjectId("533b02f51ab6204de9174503"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  4
0.012343 ], "pop" : 9999, "state" : "CA" }
mongos> db.runCommand({transaction: "commit"})
{ "ok" : 1 }
mongos> db.city_info2.find( { city: "RAINBOWVILLE" } )
{ "_id" : ObjectId("533b02f51ab6204de9174503"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  4
0.012343 ], "pop" : 9999, "state" : "CA" }
{ "_id" : ObjectId("533b03c91ab6204de9174504"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  4
0.012343 ], "pop" : 9999, "state" : "MA" }
mongos> db.city_info2.insert({ "city" : "RAINBOWVILLE", "loc" : [-99.999999, 40.012343 ], "pop" :
9999, "state" : "ZA" })
mongos> db.runCommand({transaction: "rollback"})
{ "ok" : 1 }
mongos> db.city_info2.find( { city: "RAINBOWVILLE" } )
{ "_id" : ObjectId("533b02f51ab6204de9174503"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  4
0.012343 ], "pop" : 9999, "state" : "CA" }
{ "_id" : ObjectId("533b03c91ab6204de9174504"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  4
0.012343 ], "pop" : 9999, "state" : "MA" }
```


© 2017 IBM Corporation

Transactions - Nosql Informix extension (2)

```
mongos> db.runCommand({transaction: "enable"})
{ "ok" : 1 }
mongos> db.city_info2.insert({ "city" : "RAINBOWVILLE", "loc" : [-99.999999, 40.012343 ], "pop" : 9999, "state" : "ZA" })
mongos> db.runCommand({transaction: "commit"})
{ "ok" : 1 }
mongos> db.runCommand({transaction: "status"})
{ "enabled" : true, "supported" : true, "ok" : 1 }
mongos> db.city_info2.find( { city: "RAINBOWVILLE" } )
{ "_id" : ObjectId("533b12d41ab6204de9174506"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "MN" }
{ "_id" : ObjectId("533b02f51ab6204de9174503"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "CA" }
{ "_id" : ObjectId("533c4612385ecd075e57df09"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "ZA" }
{ "_id" : ObjectId("533b03c91ab6204de9174504"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "MA" }
mongos> db.city_info2.update( { state: "ZA" }, { $set: { state: "NY" } } )
mongos> db.runCommand({transaction: "commit"})
{ "ok" : 1 }
mongos> db.city_info2.find( { city: "RAINBOWVILLE" } )
{ "_id" : ObjectId("533b12d41ab6204de9174506"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "MN" }
{ "_id" : ObjectId("533b02f51ab6204de9174503"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "CA" }
{ "_id" : ObjectId("533c4612385ecd075e57df09"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "NY" }
{ "_id" : ObjectId("533b03c91ab6204de9174504"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "MA" }
mongos> db.city_info2.remove({ "$and" : [{ state : "NY", city: "RAINBOWVILLE" }] })
mongos> db.runCommand({transaction: "commit"})
{ "ok" : 1 }
mongos> db.city_info2.find( { city: "RAINBOWVILLE" } )
{ "_id" : ObjectId("533b12d41ab6204de9174506"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "MN" }
{ "_id" : ObjectId("533b02f51ab6204de9174503"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "CA" }
{ "_id" : ObjectId("533b03c91ab6204de9174504"), "city" : "RAINBOWVILLE", "loc" : [  -99.999999,  40.012343 ], "pop" : 9999, "state" : "MA" }
Mongos>
```


© 2017 IBM Corporation

Create and Drop Index

- **mongos> db.city_info2.ensureIndex({ pop: 1 })**
  - Returns nothing if successful

- **mongos> db.city_info2.dropIndex({ "pop" : 1 })**
  { "ok" : 1, "nIndexesWas" : 2 }

© 2017 IBM Corporation

What Indexes Do You Have on Your Collection?

- **Below, one index is unique, the others are not:**

```
mongos> db.city_info2.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : [
                                1,
                                "$string"
                        ]
                },
                "ns" : "stores_demo.city_info2",
                "name" : "_id_",
                "unique" : true,
                "index" : "city_info2__id_"
        },
        {
                "ns" : "stores_demo.city_info2",
                "key" : {
                        "pop" : [
                                1,
                                "$extract"
                        ]
                },
                "name" : "pop_1",
                "index" : "city_info2_pop_1"
        },
        {
                "ns" : "stores_demo.city_info2",
                "key" : {
                        "state" : [
                                1,
                                "$extract"
                        ]
                },
                "name" : "state_1",
                "index" : "city_info2_state_1"
        }
]
```


© 2017 IBM Corporation

Isolation levels

- Default isolation level is DIRTY READ.
- Change this directly or sysdbopen()
- You can also use USELASTCOMMITTED variable in the database server configuration file ($ONCONFIG).
- If you're using procedures for executing multi-statement transactions, you can set it within your procedure.

© 2017 IBM Corporation

Locking

- **Page level locking is the default.**
- **You can change it to ROW level locks easily.**
  - **ALTER TABLE jc MODIFY lock mode (row);**
  - **DEF_TABLE_LOCKMODE** server configuration file variable.

- **SET LOCK MODE can be set via sysdbopen()**
- **Each statement is executed with auto-commit and locking semantics will apply there.**

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

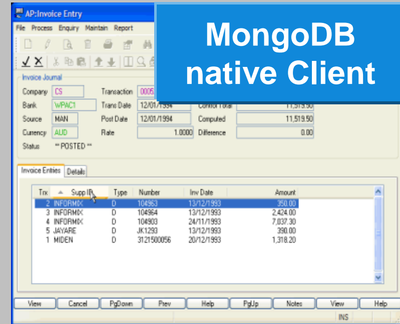
The Wire Listener – How Informix Communicates with MongoDB

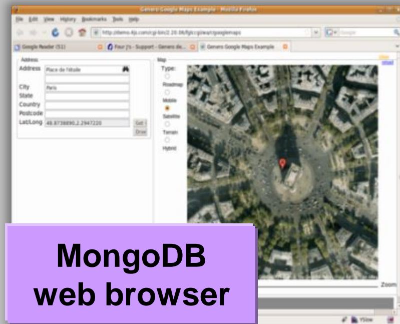
© 2021 IBM Corporation

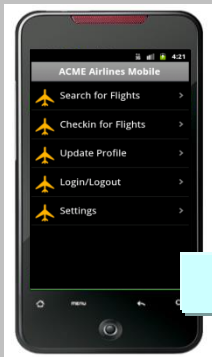
Client Applications & The Wire Listener

- Wire Listener supports existing MongoDB drivers
- Connect to MongoDB or Informix with same application!
- $INFORMIXDIR/etc/jsonListener.jar

Applications







MongoDB driver

MongoDB Wire Protocol

MongoDB

Informix 14.1

```
informixva:/data/opt/informix/bin # java -jar $INFORMIXDIR/bin/jsonListener.jar -config $INFORMIXDIR/etc/jsonListener.properties -start
starting listener on port 27017
[main] INFO com.ibm.nosql.informix.server.LwfJsonListener - JSON server listening on port: 27017, 0.0.0.0/0.0.0.0
```


© 2017 IBM Corporation

Starting and Stopping the Wire Listener via sysadmin

- EXECUTE FUNCTION task("stop json listener");

  (expression) stop requested for localhost:27017

- EXECUTE FUNCTION task("start json listener");

  (expression) JSON listener launched. For status check /opt/IBM/informix_12.10_1/jsonListener.log

- cat $INFORMIXDIR/jsonListener.log
  2014-04-01 19:16:39 [JsonListener-1] INFO com.ibm.nosql.informix.server.LwfJsonListener - JSON server listening on port: 27017, 0.0.0.0/0.0.0.0

- Both JSON and Informix SQL commands can pass through this.

© 2017 IBM Corporation

Starting and Stopping via the command Line

- java –jar $INFORMIXDIR/bin/jsonListener.jar –config $INFORMIXDIR/etc/jsonListener.properties –start
- java –jar $INFORMIXDIR/bin/jsonListener.jar –config $INFORMIXDIR/etc/jsonListener.properties –stop
- The standard editable jsonListener.properties file is located in $INFORMIXDIR/etc
- Sample jsonListener.properties contents:

```
listener.port=27017
url=jdbc:informix-sqli://localhost:17875/sysmaster:INFORMIXSERVER=lo_informix1210_1;USER=ifxjson;PASSWORD=pj0D$XsMiFG
security.sql.passthrough=true
sharding.enable=true
update.client.strategy=deleteInsert:
```


© 2017 IBM Corporation

Notes on the Wire Listener

- **Parameters in this file are one per line and settings are not dynamic at this time.**
- **To enable sharding collections in JSON based apps using "MongoDB", you must turn on "sharding.enable=true"**
  - If **sharding.enable=true** then you are required to set one more parameter as well, you must set **update.client.strategy=deleteInsert.**
- **To enable Informix SQL pass thru in JSON based apps using "MongoDB", you must turn on "security.sql.passthrough=true"**
- **User name & password are optional**
  - O/S authentication used in its place.
- **INFORMIXSERVER is usually a reserved port, not the main server instance port, and is found in /etc/services and sqlhosts.**

© 2017 IBM Corporation

Three Server Cluster sqlhosts file – ol_informix1410_1

- **cat $INFORMIXSQLHOSTS**

```
dr_informix1410_1    drsoctcp   cte2        dr_informix1410_1
lo_informix1410_1    onsoctcp   127.0.0.1   lo_informix1410_1
g_ol_informix1410_1  group      -           -                   i=1
ol_informix1410_1    onsoctcp   cte2        ol_informix1410_1   g=g_ol_informix1410_1
g_ol_informix1410_2  group      -           -                   i=2
ol_informix1410_2    onsoctcp   cte2        26311               g=g_ol_informix1410_2
g_ol_informix1410_3  group      -           -                   i=3
ol_informix1410_3    onsoctcp   cte2        10756               g=g_ol_informix1410_3
```


© 2017 IBM Corporation

Three Server Cluster sqlhosts file – ol_informix1410_2

- cat $INFORMIXSQLHOSTS

```
dr_informix1410_2     drsoctcp    cte2       dr_informix1410_2
lo_informix1410_2     onsoctcp    127.0.0.1  lo_informix1410_2
g_ol_informix1410_1   group       -          -                   i=1
ol_informix1410_1     onsoctcp    cte2       31374               g=g_ol_informix1410_1
g_ol_informix1410_2   group       -          -                   i=2
ol_informix1410_2     onsoctcp    cte2       26311               g=g_ol_informix1410_2
g_ol_informix1410_3   group       -          -                   i=3
ol_informix1410_3     onsoctcp    cte2       10756               g=g_ol_informix1410_3
```


© 2017 IBM Corporation

Three Server Cluster sqlhosts file – ol_informix1410_3

- cat $INFORMIXSQLHOSTS

```
dr_informix1410_3    drsoctcp   cte2       dr_informix1410_3
lo_informix1410_3    onsoctcp   127.0.0.1  lo_informix1410_3
g_ol_informix1410_1  group      -          -              i=1
ol_informix1410_1    onsoctcp   cte2       31374          g=g_ol_informix1410_1
g_ol_informix1410_2  group      -          -              i=2
ol_informix1410_2    onsoctcp   cte2       26311          g=g_ol_informix1410_2
g_ol_informix1410_3  group      -          -              i=3
ol_informix1410_3    onsoctcp   cte2       10756          g=g_ol_informix1410_3
```


© 2017 IBM Corporation

/etc/services for 3 Server cluster

```
ol_informix1410_2    26311/tcp
dr_informix1410_2    20669/tcp
lo_informix1410_2    29368/tcp
ol_informix1410_3    10756/tcp
dr_informix1410_3    15685/tcp
lo_informix1410_3    31498/tcp
ol_informix1410_1    31374/tcp
dr_informix1410_1     6498/tcp
lo_informix1410_1    17875/tcp
```


© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Sharding

© 2021 IBM Corporation

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**

© 2017 IBM Corporation

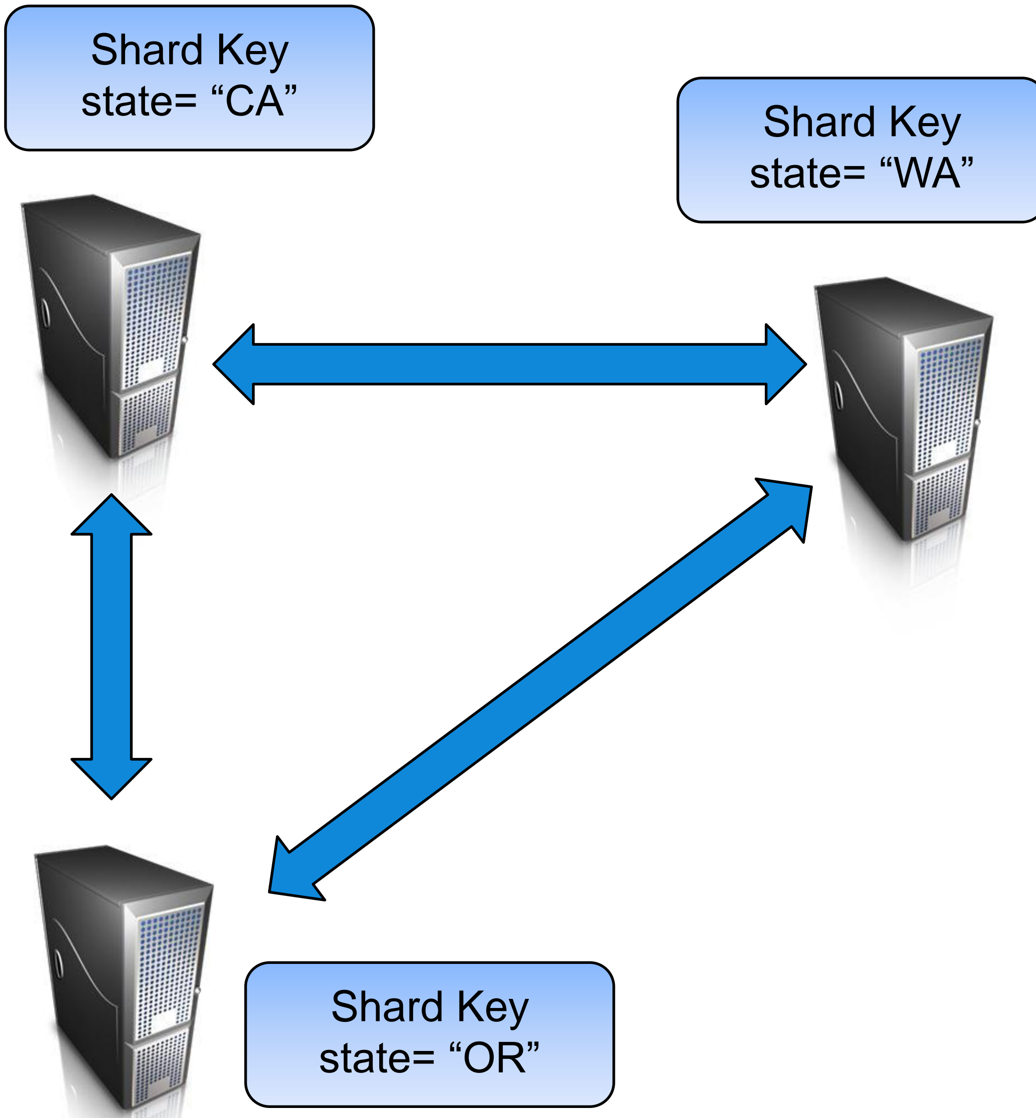
Why Scale Out Instead of Up?

- **Scaling Out**
  - Adding more servers with less processors and RAM
  - Advantages
    - Startup costs much less
    - Can grow instep with the application
    - Individual servers cost less
      - Several less expensive server rather than fewer high cost servers
      - Redundant servers cost more
    - Greater chance of isolating catastrophic failures
    - Dynamic – instance does not go down

- **Scaling Up**
  - Adding more processors and RAM to a single server
  - Advantages
    - Less power consumption than running multiple servers
    - Less infrastructure (network, licensing,..)

© 2017 IBM Corporation

Difference between Sharding Data vs Replication



Sharding	Replication
Each node hold 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
Actions are shipped to applicable nodes	Work on local copy and modification are propagated

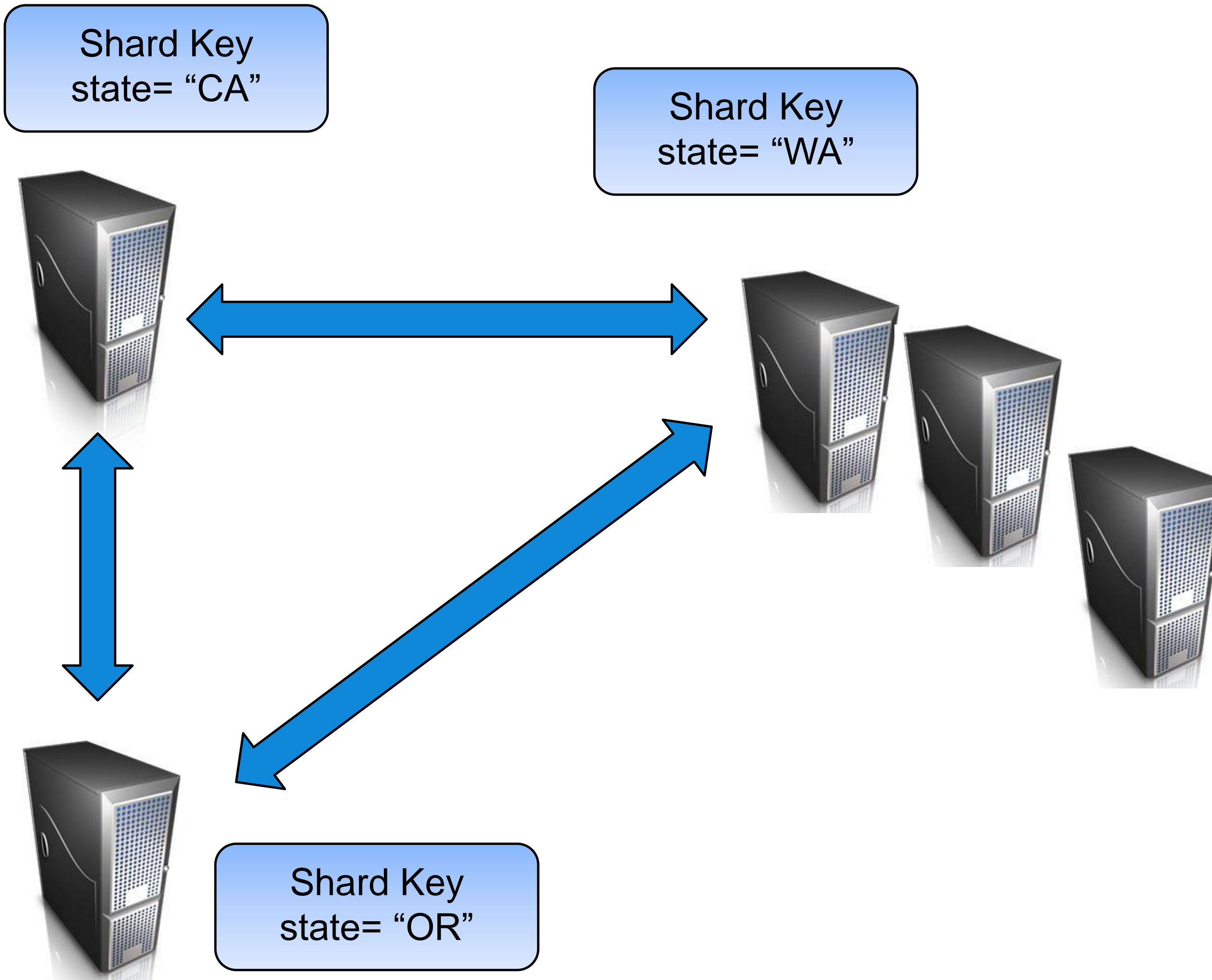
© 2017 IBM Corporation

Mongo 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 MACH, and additional nodes secondary
- **With Informix the secondary server can:**
  - Provide high availability
  - Scale out
    - Execute select
    - Allow Insert/Update/Deletes on the secondary servers
    - Share Disks with the master/primary node

© 2017 IBM Corporation

Mongo Sharding is not for Data Availability



© 2017 IBM Corporation

Basic Data Distribution/Replication Terms

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	MACH 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.	Secondary Server Remote Secondary

© 2017 IBM Corporation

Mongo Sharding

mongo

app server | app server | app server

mongos | mongos | mongos

Shard 1		
config		
mongod primary	mongod secondary	mongod secondary
RAID 10	RAID 10	RAID 10

Shard 2		
config		
mongod primary	mongod secondary	mongod secondary
RAID 10	RAID 10	RAID 10

Shard 3		
config		
mongod primary	mongod secondary	mongod secondary
RAID 10	RAID 10	RAID 10

© 2017 IBM Corporation

Scaling Out Using Sharded Queries

Shard Key
state= "CA"

Shard Key
state= "WA"

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

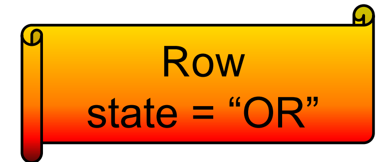
Shard Key
state= "OR"

© 2017 IBM Corporation

Scaling Out Using Sharded Inserts

Shard Key
state= “CA”

Shard Key
state= “WA”

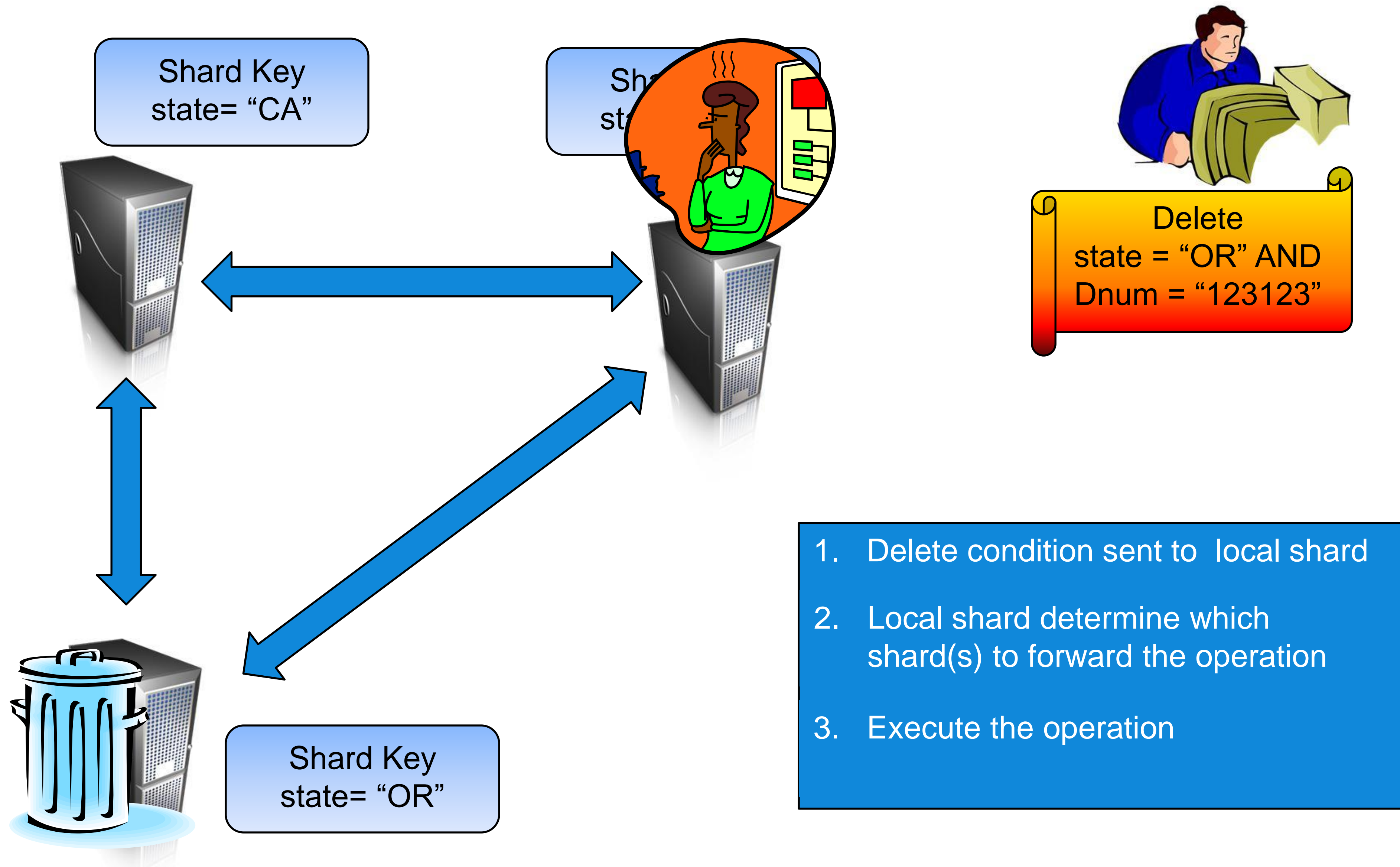


Shard Key
state= “OR”

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

© 2017 IBM Corporation

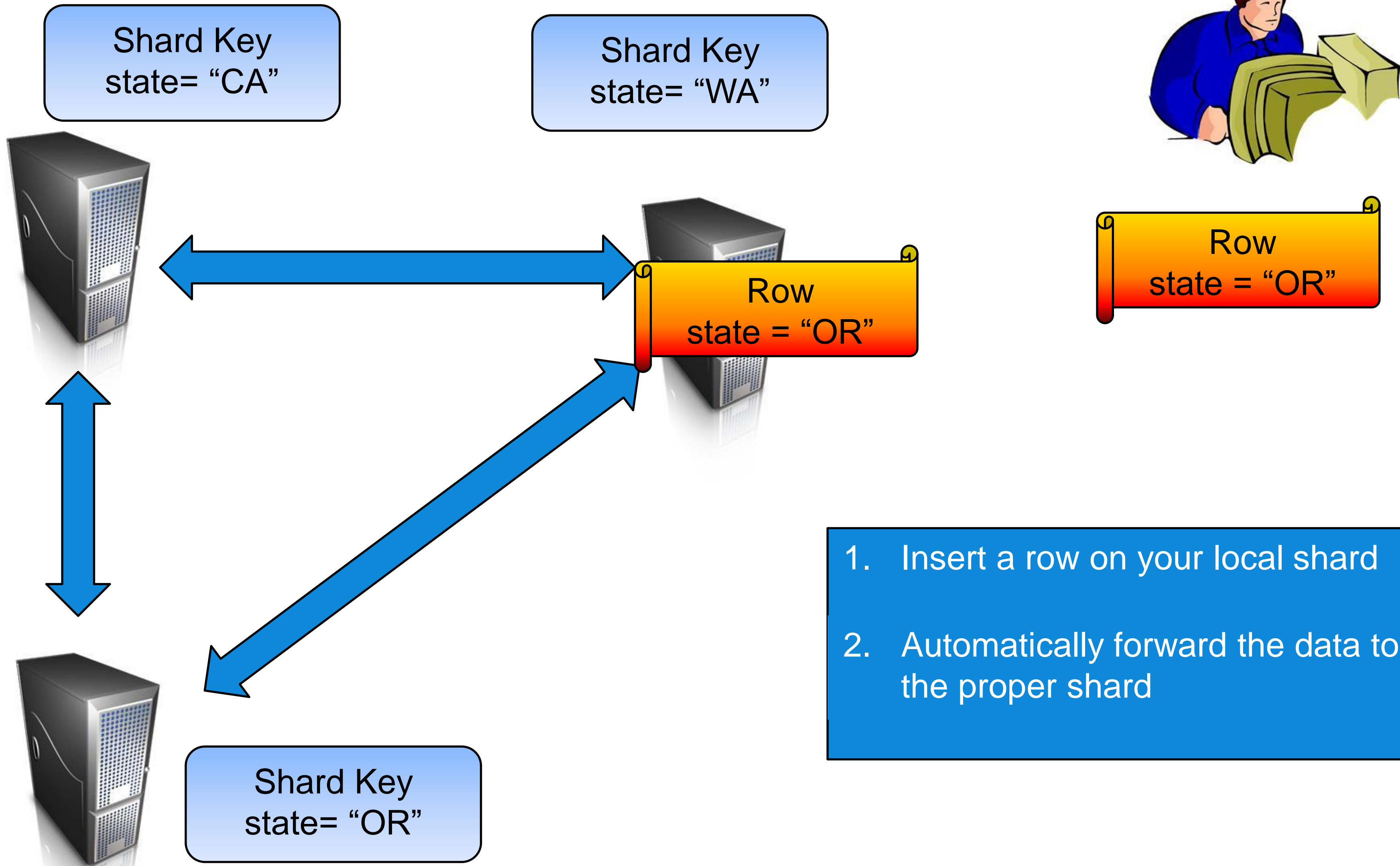
Scaling Out Using Sharded Delete



1. Delete condition sent to local shard
2. Local shard determine which shard(s) to forward the operation
3. Execute the operation

© 2017 IBM Corporation

Scaling Out Using Sharded Update



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

© 2017 IBM Corporation

Scaling Out Adding a Shard

Shard Key
state= "CA"

Shard Key
state= "WA"

Command
Add Shard "NV"

Shard Key
state= "OR"

Shard Key
state= "NV"

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

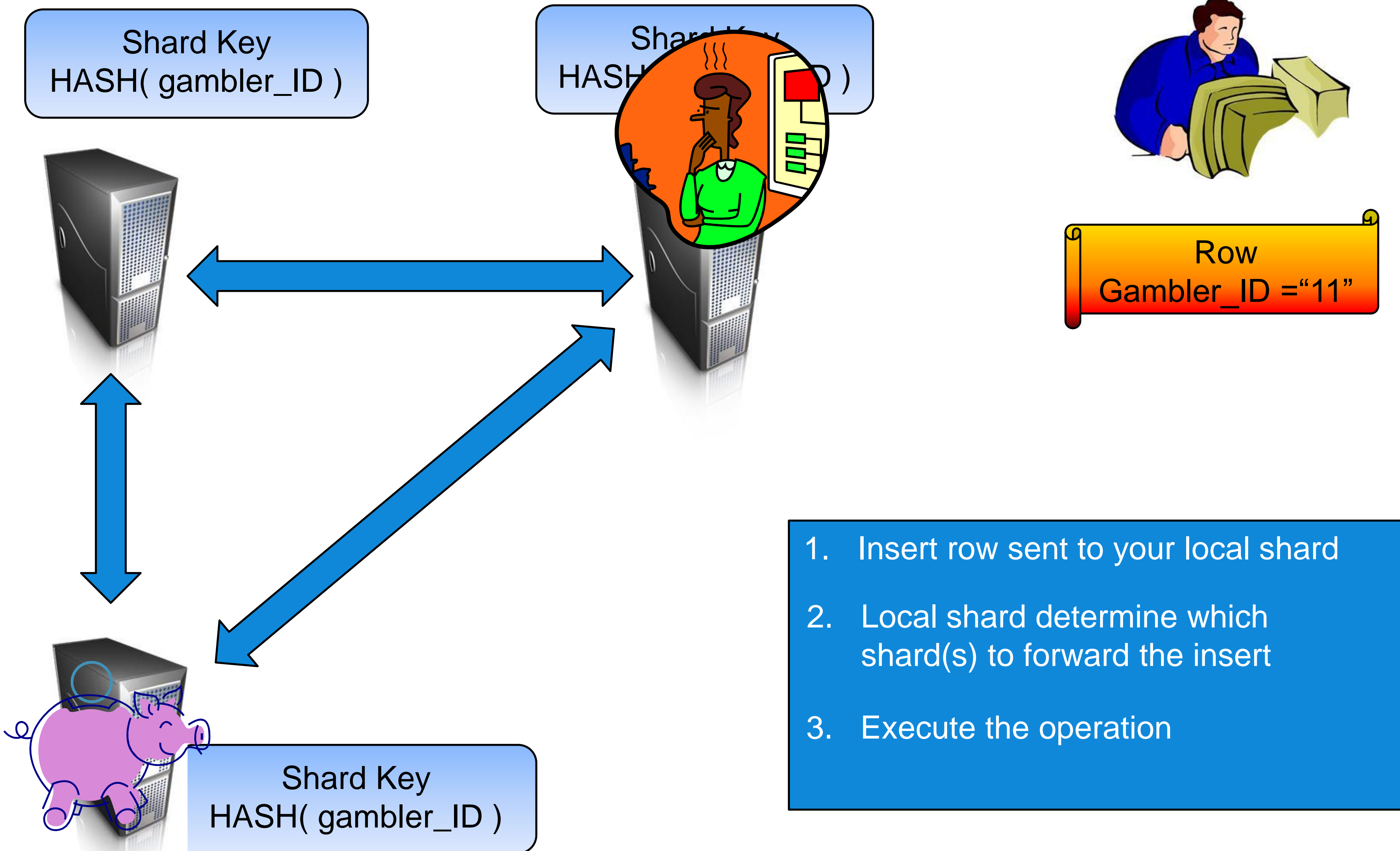
© 2017 IBM Corporation

Sharding with Hash

- **Hash based sharding simplifies the partitioning of data across the shards**
- **Pros**
  - No data layout planning is required
  - Adding additional nodes is online and dynamic
- **Cons**
  - Adding additional node requires data to be moved
  - Electricity/hardware costs if taken to extremes
- **Data automatically broken in pieces**

© 2017 IBM Corporation

Scaling Out with Hash Sharding - Insert



© 2017 IBM Corporation

Informix NoSQL Cluster Architecture Overview

two independent copies of the data, but three servers to share the workload (two servers share the same disks). Read/Write activity supported on all servers

Shard Key state= “CA”

two independent copies of the data and two servers to share the workload. Read/Write activity supported on all servers

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= “OR”

Shard Key state= “WA”

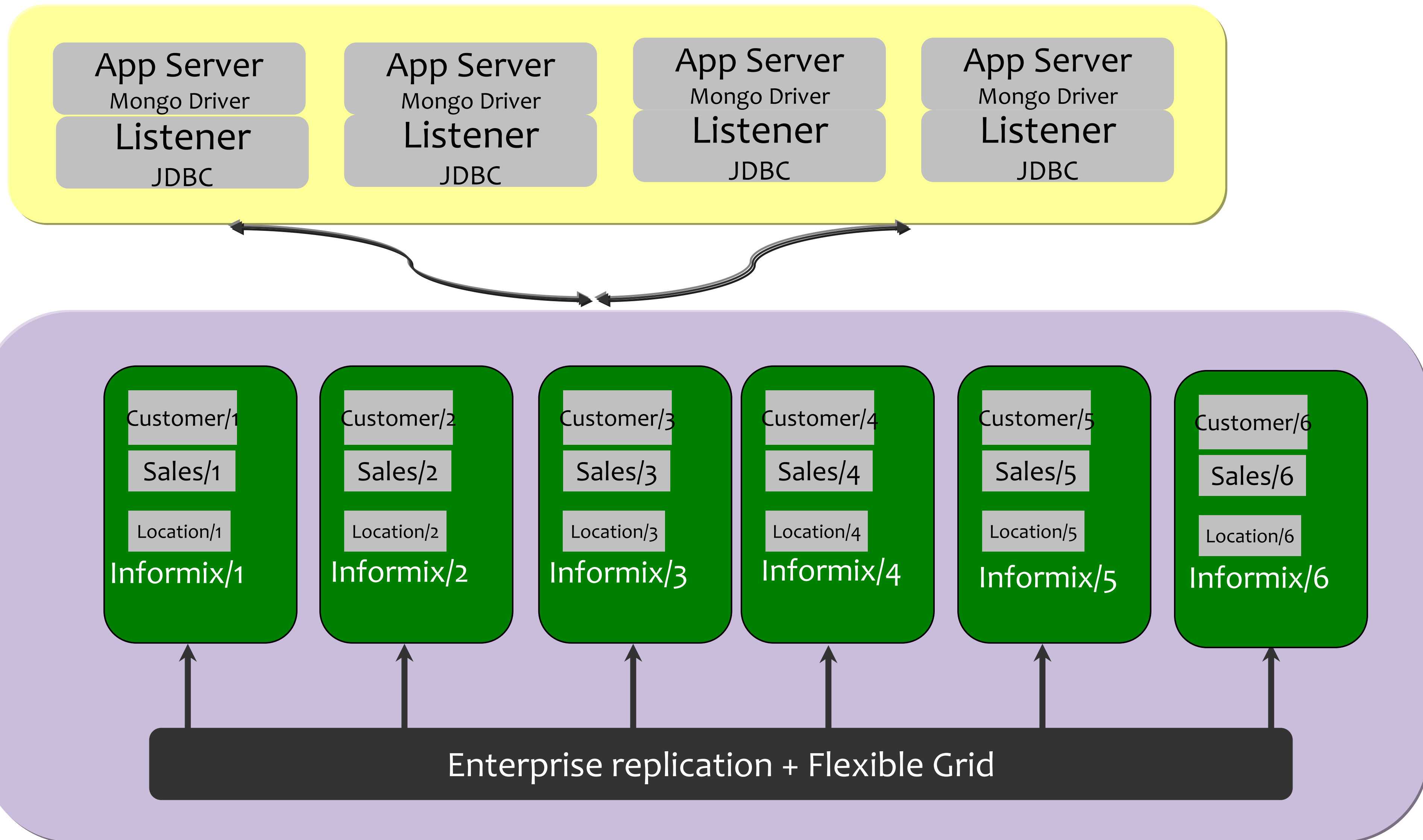
© 2017 IBM Corporation

MongoDB SHARDING details (roughly)

- **Shard a single table by range or hashing.**
- **Mongos will direct the INSERT to target shard.**
- **Mongos tries to eliminate shards for update, delete, selects as well.**
- **FIND (SELECT) can happen ONLY a SINGLE table.**
- **Mongos works as coordinator for multi-node ops.**
- **Once a row is inserted to a shard, it remains there despite any key update.**
- **No transactional support on multi-node updates.**
  - Each document update is unto its own.

© 2017 IBM Corporation

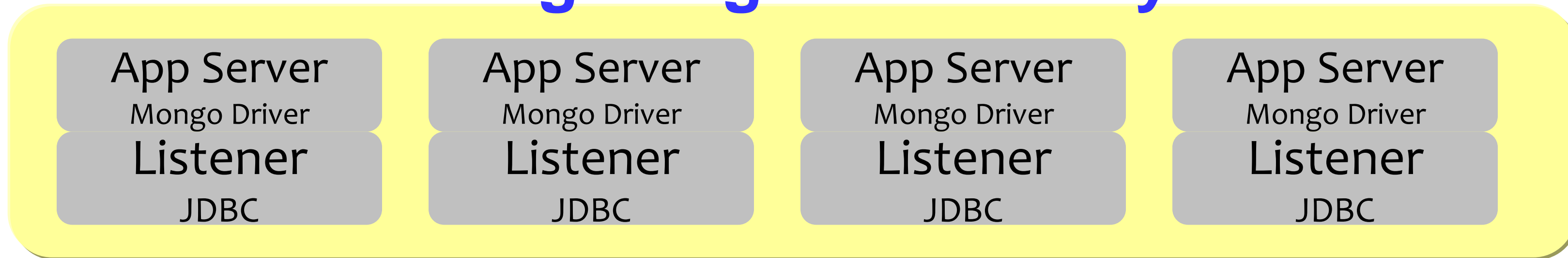
Informix Sharding



© 2017 IBM Corporation

IBM

Informix Sharding + High Availability



Informix/1 Primary
Informix/1 SDS/HDR
Informix/1 RSS

Informix/2 Primary
Informix/2 SDS/HDR
Informix/2 RSS

Informix/3 Primary
Informix/3 SDS/HDR
Informix/3 RSS

Informix/4 Primary
Informix/4 SDS/HDR
Informix/4 RSS

Informix/5 Primary
Informix/5 SDS/HDR
Informix/5 RSS

Informix/6 Primary
Informix/6 SDS/HDR
Informix/6 RSS

Enterprise replication + Flexible Grid

© 2017 IBM Corporation

Sharding – Informix Implementation

- **Shard a single table by range or hashing.**

```
cdr define shard myshard mydb:usr1.mytab \
–type=delete –key=”bson_get(bsoncol, ‘STATE’)” –strategy=expression \
versionCol=version \
servA “in (‘TX’, ‘OK’)” \
servB “in (‘NY’,’NJ’) “ \
servC “in (‘AL’,’KS’) “ \
servD  remainder

cdr define shard myshard mydb:usr1.mytab \
–type=delete –key=state –strategy=hash --versionCol=version \
servA servB servC servD
```

- **MongoDB shard a single table by hashing.**
  – sh.shardCollection("records.active", { a: "hashed" })

© 2017 IBM Corporation

Show Sharded Collections – All Collections

- **user1@cte2:~> cdr list shardCollection**

```
Shard Collection:sh_stores_demo_city_info2 Version:0 type:hash key:bson_value_lvarchar(data, '_id')
Version Column:modCount
Table:stores_demo:ifxjson.city_info2
g_ol_informix1210_1     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  = 0
g_ol_informix1210_2     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (1, -1)
g_ol_informix1210_3     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (2, -2)

Shard Collection:sh_stores_demo_city_info Version:1 type:hash key:bson_value_lvarchar(data, '_id')
Version Column:modCount
Table:stores_demo:ifxjson.city_info
g_ol_informix1210_1     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  = 0
g_ol_informix1210_2     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (1, -1)
g_ol_informix1210_3     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (2, -2)

Shard Collection:sh_stores_demo_lowercase Version:0 type:hash key:bson_value_lvarchar(data, '_id')
Version Column:modCount
Table:stores_demo:ifxjson.lowercase
g_ol_informix1210_1     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  = 0
g_ol_informix1210_2     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (1, -1)
g_ol_informix1210_3     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (2, -2)
```


© 2017 IBM Corporation

Listing Sharded Collections – Single Collection

- **user1@cte2:~> cdr list shardCollection stores_demo.city_info2**

```
Shard Collection:sh_stores_demo_city_info2 Version:0 type:hash key:bson_value_lvarchar(data, '_id')

Version Column:modCount
Table:stores_demo:ifxjson.city_info2

g_ol_informix1210_1     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  = 0
g_ol_informix1210_2     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (1, -1)
g_ol_informix1210_3     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (2, -2)

Shard Collection:sh_stores_demo_city_info Version:1 type:hash key:bson_value_lvarchar(data, '_id')
Version Column:modCount

Table:stores_demo:ifxjson.city_info

g_ol_informix1210_1     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  = 0
g_ol_informix1210_2     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (1, -1)
g_ol_informix1210_3     mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3)  in (2, -2)
```


© 2017 IBM Corporation

onstat –g shard

- onstat -g shard
  Your evaluation license will expire on 2014-06-26 00:00:00

  IBM Informix Dynamic Server Version 12.10.FC3TL -- On-Line -- Up 4 days 07:05:02 -- 208516 Kbytes

  sh_stores_demo_city_info2 stores_demo:ifxjson.city_info2 key:bson_value_lvarchar(data, '_id')
  HASH:DELETE SHARD OPTIMIZATION:ENABLED

  Matching for delete:modCount
  g_ol_informix1210_1 (65550) mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3) = 0
  g_ol_informix1210_2 (65551) mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3) in (1, -1)
  g_ol_informix1210_3 (65552) mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3) in (2, -2)

  sh_stores_demo_city_info stores_demo:ifxjson.city_info key:bson_value_lvarchar(data, '_id')
  HASH:DELETE SHARD OPTIMIZATION:ENABLED

  Matching for delete:modCount
  g_ol_informix1210_1 (65547) mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3) = 0
  g_ol_informix1210_2 (65548) mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3) in (1, -1)
  g_ol_informix1210_3 (65549) mod(ifx_checksum(bson_value_lvarchar(data, '_id')::LVARCHAR, 0), 3) in (2, -2)

© 2017 IBM Corporation

Sharding – Informix Implementation

- **Shard a single table by range or hashing.**
- **Sharding is transparent to application.**

- **Each CRUD statement will only touch a single table.**
  - Limitation of MongoDB…Makes it easier for Informix.
  - Lack of joins is a big limitation for SQL applications.

- **Lacks transactional support for distributed update.**

© 2017 IBM Corporation

Informix Sharding

- **Identical table is created on each node and meta data is replicated on each node.**
- **Schema based replication is the foundation for our sharding.**

- **CRUD operations can go to any node.**
  - We'll use replication and other techniques to reflect the data in the target node, *eventually.*
  - Replication is asynchronous in Informix presently.
  - Informix also has synchronous replication with HDR … not used for sharding now.

© 2017 IBM Corporation

SELECT on sharded tables

- **The query can be submitted to any of the nodes via the listener.**
- **That node acts as the "coordinator" for the distributed query.**
- **It also does the node elimination based on the query predicate.**
- **After that, the query is transformed to UNION ALL query**

```
SELECT SKIP ? bson_new( data, '{"_id":0.0 ,"num":1.0 ,"name" : 1.0}')::bson
   FROM customers@rsys1:db
 WHERE bson_extract(data, 'name') = "A" or bson_extract(data, 'name') = "X"
```

is transformed into:

```
SELECT SKIP ? bson_new( data, '{"_id":0.0 ,"num":1.0 ,"name" : 1.0}')::bson
   FROM customers@rsys1:db
 WHERE bson_extract(data, 'name') = "A" or bson_extract(data, 'name') = "X"
UNION ALL
SELECT SKIP ? bson_new( data, '{"_id":0.0 ,"num":1.0 ,"name" : 1.0}')::bson
   FROM customers@rsys1:db
 WHERE bson_extract(data, 'name') = "A" or bson_extract(data, 'name') = "X"
```


© 2017 IBM Corporation

INSERT: Single node

- **If necessary, automatically create database & table (collection) on the application INSERT**

- **Collections: CREATE TABLE t(a GUID, d BSON);**

- **GUID column is needed to ensure unique row across the SHARD's:**
  - Also used as a Primary Key (PK) in Enterprise Replication.

- **Client application inserts JSON, client API converts this into BSON, generates '_id' (object id) if necessary & sends to server over JDBC/ODBC.**

- **Server saves the data into this table as BSON, with an automatically generated GUID**

© 2017 IBM Corporation

DELETE: Single node

- Mongo remove are translated to SQL DELETE
- Will always remove all the qualifying rows.
- WHERE clause translation is same as SELECT

© 2017 IBM Corporation

IBM

UPDATE: Single node

- **Simple set, increment, decrement updates are handled directly by the server.**

```
Mongo: db.w.update({x: {$gt:55}}, {$set:{z:9595}});
SQL: UPDATE w
           SET data = bson_update(data, "{$set:{z:9595}}")
        WHERE bson_extract(data, 'x') > "{x:55}"::bson ?
```

- **bson_update is a built-in expression updating a BSON document with a given set of operations.**

- **Complex updates are handled via select batch, delete, insert.**

- **Inform*ix* always updates all the rows or no rows, under a transaction.**

© 2017 IBM Corporation

INSERT: shard implementation

- **If the table is sharded:**
  - Insert the data into local table as usual.
  - Replication threads in the background will evaluate the log record to see which rows will have moved.
  - Replication thread will move the necessary row to target and delete from the local table.
  - For each inserted row ending up in non-local shard, simply generate the logical log and avoid insert into local table & indices.

© 2017 IBM Corporation

DELETE : shard implementation

- **Application delete could delete rows from any of the shards in the table.**

- **DELETE will come in as shard_delete() procedure.**

  execute procedure shard_delete(tabname, delete_stmt);

  – This procedure will issue delete locally.
  – It will then **INSERT** the delete statement into a "special" shard delete table (single for all tables).
  – Enterprise replication will propagate the delete to applicable target systems.

© 2017 IBM Corporation

UPDATE: shard implementation

- **When the update have to be done on multiple nodes:**
  - **Client application does UPDATE and CLIENT API converts that into three operations:**
    - **SELECT, DELETE & INSERT**
    - Inserts **JSON**, client API converts this into **BSON** & sends to server.
    - ListenerFile needs to be configured properly for this:
      - if **sharding.enable=true** in the listener file then you are required to set **update.client.strategy=deleteInsert.**

- **GUID column is needed to ensure unique row across the SHARD**

© 2017 IBM Corporation

Transactions (single node)

- **Mongo does not have the notion of transactions.**
  - Each document update is atomic, but not the app statement
- **For the first release of Informix-NoSQL**
  - By default, **JDBC** listener simply uses **AUTO COMMIT** option
  - Each server operation **INSERT, UPDATE, DELETE, SELECT** will be automatically be committed after each operation.
    - No locks are held across multiple application operations.

© 2017 IBM Corporation

Transactions (sharded environment)

- **In sharded environment, mongo runs databases via two different instances: mongos and mongod.**
  - **Mongos** simply redirects operations to the relevant **mongod.**
  - No statement level transactional support.

- **Informix**
  - Does not have the 2-layer architecture
  - The server the application connected to becomes the transaction coordinator
  - Now has 2-phase commit protocol transaction support
  - **SELECT** statement goes thru distributed query infrastructure(ISTAR) as always
  - **INSERT, UPDATE, DELETE** direct thru Enterprise Replication as always.
    - Wire Listener must be configured for this, if the parameter **sharding.enable=true** is set then you are required to set **update.client.strategy=deleteInsert.**

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

IBM

MongoDB Utilities

© 2021 IBM Corporation

mongoexport

- **A command-line utility that produces a JSON or CSV (comma separated values) export of data stored in a MongoDB instance.**
  - "Do not use **mongoimport** and **mongoexport** for full-scale backups because they may not reliably capture data type information."
  - "Use **mongodump** and **mongorestore** as described in *MongoDB Backup Methods* for this kind of functionality."

- **By default, mongoexport will write one JSON document on output for every MongoDB document in the collection.**

- **The other output format possibility presently is via CSV; you must specify this separately with the "- - csv" option and use the "- - fields" option, to specify the fields within the document to use.**

- **The MongoDB web site contains lots of page references for methods to employ to get a complete export that fall short of complete.**

© 2017 IBM Corporation

IBM

mongoexport

- **Below is the command line to dump to a JSON file (default) for the city_info collection:**

```
user1@cte2:~> mongoexport -d stores_demo -c city_info2 -o mongoexport/city_info2.out
connected to: 127.0.0.1
exported 29470 records
```

- **mongoexport requires a collection to be specified:**

```
user1@cte2:~> mongoexport -d stores_demo -o mongoexport/all.out
connected to: 127.0.0.1
no collection specified!
Export MongoDB data to CSV, TSV or JSON files.
```

- **What does the JSON output look like:**

```
{ "_id" : "70001", "city" : "METAIRIE", "loc" : [ -90.16951299999999, 29.987138 ], "pop" : 39554,
"state" : "LA" }
{ "_id" : { "$oid" : "533b12d41ab6204de9174506" }, "city" : "RAINBOWVILLE", "loc" : [ -99.999999,
40.012343 ], "pop" : 9999, "state" : "MN" }
{ "_id" : "70047", "city" : "NEW SARPY", "loc" : [ -90.373982, 29.96579 ], "pop" : 10472, "state"
: "LA" }
{ "_id" : "70052", "city" : "GRAMERCY", "loc" : [ -90.69018199999999, 30.052711 ], "pop" : 2765, "
state" : "LA" }
{ "_id" : "70058", "city" : "HARVEY", "loc" : [ -90.06725900000001, 29.872535 ], "pop" : 36824, "s
tate" : "LA" }
{ "_id" : "70067", "city" : "LAFITTE", "loc" : [ -90.05663300000001, 29.562194 ], "pop" : 0, "stat
e" : "LA" }
{ "_id" : "70070", "city" : "LULING", "loc" : [ -90.36926099999999, 29.925116 ], "pop" : 11956, "s
tate" : "LA" }
```


© 2017 IBM Corporation

IBM

mongoexport

- **CSV output:**

```
user1@cte2:~/mongoexport> mongoexport -d stores_demo -c city_info2 --csv -f city,loc,pop,state -o
mongoexport/city_info2.export.out
connected to: 127.0.0.1
exported 29470 records
user1@cte2:~/mongoexport>
```

- **What does it look like:**

```
city,loc,pop,state
"METAIRIE","[ -90.16951299999999, 29.987138 ]",39554,"LA"
"RAINBOWVILLE","[ -99.999999, 40.012343 ]",9999.0,"MN"
"NEW SARPY","[ -90.373982, 29.96579 ]",10472,"LA"
"GRAMERCY","[ -90.69018199999999, 30.052711 ]",2765,"LA"
"HARVEY","[ -90.06725900000001, 29.872535 ]",36824,"LA"
"LAFITTE","[ -90.05663300000001, 29.562194 ]",0,"LA"
"LULING","[ -90.36926099999999, 29.925116 ]",11956,"LA"
"MERAUX","[ -89.92143299999999, 29.933494 ]",7196,"LA"
"VENICE","[ -89.347776, 29.261812 ]",458,"LA"
```

- **A comma actually being a piece of data could be a problem here…..**
- **Note: the "_id" field was not selected and does not appear here. This is expected behavior.**
  – Exporting to a JSON file (default) makes it appear.

© 2017 IBM Corporation

mongoimport – Migration tool ????

- mongoimport of a JSON file into a collection table, the collection city_info3 did not exist within Informix, prior to the import of the 29470 records:

```
user1@cte2:~/mongoexport> mongoimport -d stores_demo -c city_info3 < city_info2.out
connected to: 127.0.0.1
Thu Apr  3 14:21:13.131                         4600    1533/second
Thu Apr  3 14:21:17.463                         7000    1000/second
Thu Apr  3 14:21:21.348                         11600   1054/second
Thu Apr  3 14:21:25.791                         12700   846/second
Thu Apr  3 14:21:30.695                         15000   750/second
Thu Apr  3 14:21:33.027                         18400   800/second
Thu Apr  3 14:21:36.865                         24400   938/second
Thu Apr  3 14:21:39.201                         29100   1003/second
Thu Apr  3 14:21:39.207 check 9 29470
Thu Apr  3 14:21:40.721 imported 29470 objects
user1@cte2:~/mongoexport>
```

```
Table Name          city_info3
Owner               ifxjson
Row Size            4240
Number of Rows      29470
Number of Columns   4
Date Created        04/03/2014
```

- During its operation, mongoimport reports the time every 4 seconds or so on average, and how many records to the second have been imported, and the average load time per second.

- To import 29470 objects took roughly 27 seconds inside of a virtualized image on laptop.

© 2017 IBM Corporation

mongoimport

- **A file exported with mongoexport should be imported with mongoimport.**

- **A collection table, if it does not exist in Informix or MongoDB, will be created for you with name given with the "-c" option, at time of import.**

- **There are other options here as well**
  - Specifying whether the load file is -**csv** or -**tsv** oriented, and as a sub-requirement for this option, the field names (**-f** option) to be used, comma separated (with no spaces in between fields)
  - Ability to drop an existing collection before import (**--drop**)
  - To do Upserts (**--upsert**)

- **update statistics works on collection tables and should be run immediately and in the usual way.**

© 2017 IBM Corporation

mongodump – Single Collection Dump

- **Utility to backup a database.**
- **Can backup**
  - Entire Database Collections
  - Entire Single Collection
  - Can accept a query to do a partial collection backup.
  - Point in Time Collection Backup
  - And other things as well.

- **user1@cte2:~> mongodump --collection city_info2 --db stores_demo**

```
connected to: 127.0.0.1
Wed Apr  2 15:29:35.062 DATABASE: stores_demo    to
dump/stores_demo
Wed Apr  2 15:29:35.082         stores_demo.city_info2 to
dump/stores_demo/city_info2.bson
Wed Apr  2 15:29:35.953                 29470 objects
Wed Apr  2 15:29:35.954         Metadata for stores_demo.city_info2 to
dump/stores_demo/city_info2.metadata.json
user1@cte2:~>
```


© 2017 IBM Corporation

mongodump – Single Collection Dump (cont'd)

- **Absent any other argument, mongo creates a subdirectory in the your current directory called dump, and another subdirectory under dump called database_name.**

- **The output here under database_name is in binary format and unreadable, absent any argument to the contrary. The file(s) contained here are by collection_name.**

© 2017 IBM Corporation

mongodump – Entire Database Dump

- **user1@cte2:~> mongodump --db stores_demo**

```
connected to: 127.0.0.1
Wed Apr  2 15:38:23.440 DATABASE: stores_demo    to     dump/stores_demo
Wed Apr  2 15:38:23.472         stores_demo.city_info to dump/stores_demo/city_info.bson
Wed Apr  2 15:38:23.957                 10067 objects
Wed Apr  2 15:38:23.958         Metadata for stores_demo.city_info to dump/stores_demo/city_info.metadata.json
Wed Apr  2 15:38:23.958         stores_demo.city_info2 to dump/stores_demo/city_info2.bson
Wed Apr  2 15:38:24.563                 29470 objects
Wed Apr  2 15:38:24.587         Metadata for stores_demo.city_info2 to dump/stores_demo/city_info2.metadata.json
Wed Apr  2 15:38:24.587         stores_demo.deliveries1 to dump/stores_demo/deliveries1.bson
Wed Apr  2 15:38:24.631                 1 objects
Wed Apr  2 15:38:24.631         Metadata for stores_demo.deliveries1 to dump/stores_demo/deliveries1.metadata.json
Wed Apr  2 15:38:24.631         stores_demo.lowercase to dump/stores_demo/lowercase.bson
Wed Apr  2 15:38:24.665                 2 objects
Wed Apr  2 15:38:24.665         Metadata for stores_demo.lowercase to dump/stores_demo/lowercase.metadata.json
user1@cte2:~>
```

- **All of the above objects are collections.**

- **mongodump can only backup collections, including sharded collections. Since this is all it can do:**
  - It is unlikely that all user tables in a hybrid database would be collections
  - The system catalogs will never be collections (see next slide).
  - So don't use this to backup Informix based databases.

© 2017 IBM Corporation

mongodump – Entire Database Dump

- mongodump can presently only backup collections. Below are some of the objects missed ….
  mongos> db.systables.find({ tabid: { $gte: 1 } }, { tabname: 1 })
  { "tabname" : "systables" }
  { "tabname" : "syscolumns" }
  { "tabname" : "sysindices" }
  { "tabname" : "systabauth" }
  { "tabname" : "syscolauth" }
  { "tabname" : "sysviews" }
  { "tabname" : "sysusers" }
  etc.

- So if you need a tool to work with your collections only, then this will do the job.

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Useful Informix Capabilities Over Collections

© 2021 IBM Corporation

Compression (1) (and repack and shrink too)

- **Works as user Informix within the listener file on a collection:**

```
user1@cte2:~> su - informix
Password:
informix@cte2:~> source mongoDB_env.ksh
informix@cte2:~> mongo
MongoDB shell version: 2.4.5
connecting to: test
mongos> use stores_demo
switched to db stores_demo
mongos> db.getCollection("system.sql").findOne({ "$sql": "execute function
sysadmin@ol_informix1210_1:'informix'.task ('table compress repack
shrink','city_info2','stores_demo','ifxjson')" })
{
        "(expression)" : "Succeeded: table compress repack shrink  stores_demo:ifxjson.city_info2"
}
mongos>
```

- **Listener file:**

```
informix@cte2:/opt/IBM/informix_12.10_1/etc> cat jsonListener.properties
listener.port=27017
url=jdbc:informix-
sqli://localhost:17875/sysmaster:INFORMIXSERVER=lo_informix1210_1;USER=informix;PASSWOR
D=informix
security.sql.passthrough=true
sharding.enable=true
update.client.strategy=deleteInsert
```


© 2017 IBM Corporation

Compression (2) (and repack and shrink too)

```
informix@cte2:~> dbschema -d stores_demo -t city_info2 -ss
Your evaluation license will expire on 2014-06-26 00:00:00
DBSCHEMA Schema Utility       INFORMIX-SQL Version 12.10.FC3TL

{ TABLE "ifxjson".city_info2 row size = 4240 number of columns = 4 index size = 164 }

create collection "stores_demo.city_info2" table "ifxjson".city_info2
  (
    id                  char(128) not null ,
    data                "informix".bson,
    modcount            bigint,
    flags               integer,
    primary key (id)
  ) in datadbs1 extent size 64 next size 64  compressed lock mode row;
```


© 2017 IBM Corporation

estimate_compression

- **Usual rules apply:**

```
mongos> db.getCollection("system.sql").findOne({ "$sql": "execute function
sysadmin@ol_informix1210_1:'informix'.task ('table
estimate_compression','city_info3','stores_demo','ifxjson')" })
{
        "(expression)" : "est   curr  change partnum   coloff   table/index\n----- ----- ------ ---------- -
---- ---------------------------\n64.9%  0.0%  +64.9 0x00900007   -1
stores_demo:ifxjson.city_info3\n\nSucceeded: table estimate_compression
stores_demo:ifxjson.city_info3"
}
mongos> db.getCollection("system.sql").findOne({ "$sql": "execute function
sysadmin@ol_informix1210_1:'informix'.task ('table
estimate_compression','city_info4','stores_demo','ifxjson')" })
{
        "(expression)" : "est   curr  change partnum   coloff   table/index\n----- ----- ------ ---------- -
---- ---------------------------\nPartition 100236 does not contain enough rows to build a
compression dictionary.\nThe partition must contain at least 2000 rows to perform this
operation.\nFAILED: table estimate_compression  stores_demo:user1.city_info4"
}
```


© 2017 IBM Corporation

defragment

```
mongos> db.getCollection("system.sql").findOne({ "$sql": "execute function
sysadmin@ol_informix1210_1:'informix'.task
('defragment','stores_demo:informix.sysprocbody')" })
{ "(expression)" : "OK" }
mongos> db.getCollection("system.sql").findOne({ "$sql": "execute function
sysadmin@ol_informix1210_1:'informix'.task
('defragment','stores_demo:informix.syscolumns')" })
{ "(expression)" : "OK" }
```

```
select tabname[1,18], nrows, npused, npdata, nptotal, nextns
  from sysptprof, sysptnhdr
 where sysptprof.partnum = sysptnhdr.partnum
    and dbsname = "stores_demo"
    and tabname in ("sysprocbody","syscolumns")
  order by 6 desc
```

```
tabname  syscolumns
nrows    844
npused   32
npdata   19
nptotal  32
nextns   1

tabname  sysprocbody
nrows    12511
npused   2266
npdata   1833
nptotal  2304
nextns   1
```

Obviously, the output is after defragmentation.
sysprocbody had 15 extents prior.

© 2017 IBM Corporation

Backups – ontape

- **Output is below:**

```
informix@cte2:~> mongo
MongoDB shell version: 2.4.5
connecting to: test
mongos> db.getCollection("system.sql").findOne({ "$sql": "execute function
sysadmin@ol_informix1210_1:'informix'.task ('ontape archive directory level 0','/backup/','512'
)" })
{
        "(expression)" : "Your evaluation license will expire on 2014-06-26 00:00:00\n10 percent
done.\n100 percent done.\nFile created: /backup/cte2_215_L0\n\nPlease label this tape as
number 1 in the arc tape sequence. \nThis tape contains the following logical logs:\n\n
178\n\nProgram over."
}
mongos> exit
bye
informix@cte2:~> ls -l /backup
total 122884
-rw-rw---- 1 informix informix 125829120 Apr 10 09:07 cte2_215_L0
informix@cte2:~>
```

- **This operation took no more than 1 minute start to finish, a small database. Yes…. The command output is a bit messy ……**

© 2017 IBM Corporation

Backups (cont'd)

- **onbar and onpsm do not yet work thru the Mongo interface (or the Informix SQL Admin API for that matter for restores); nor do onbar and onpsm work through the supplied MongoDB "run()" command.**

- **onbar and onpsm work normally to backup collections and tables and in the normal way.**

- **Make sure TAPEDEV, LTAPEDEV and BAR_BSALIB_PATH are set**

- **onpsm (minimally):**
  - **onpsm -D add /opt/IBM/informix_12.10_1/backups/logs -g LOGPOOL -p HIGHEST -t FILE**
  - **onpsm -D add /opt/IBM/informix_12.10_1/backups/spaces -g DBSPOOL -p HIGHEST -t FILE**
  - Substitute, if you like a more suitable directory storage capacity wise above. **$INFORMIXDIR** is likely not the best place for this.

- **onbar –b –L0**

© 2017 IBM Corporation

onbar Backups (partial results, obviously)

```
user1@cte2:/opt/IBM/informix_12.10_1/backups> cd spaces
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces> ls -l
total 4
drwxr-x--- 15 informix informix 4096 Apr 10 14:00 ol_informix1210_1
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces> cd ol_informix1210_1
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces/ol_informix1210_1> ls -l
total 52
drwxr-x--- 6 informix informix 4096 Apr 10 14:00 critical_files
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 data8dbs1
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 data8dbs2
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 data8dbs3
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 datadbs1
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 datadbs2
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 datadbs3
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 ifx_cdrdbs1
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 ifx_qdatasbspace1
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 llog
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 plog
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 rootdbs
drwxr-x--- 3 informix informix 4096 Apr 10 14:00 sbspace1
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces/ol_informix1210_1> cd datadbs1
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces/ol_informix1210_1/datadbs1> ls -l
total 4
drwxr-x--- 2 informix informix 4096 Apr 10 14:00 0
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces/ol_informix1210_1/datadbs1> cd 0
user1@cte2:/opt/IBM/informix_12.10_1/backups/spaces/ol_informix1210_1/datadbs1/0> ls -l
total 2668
-rw-r----- 1 informix informix 2730217 Apr 10 14:00 ol_informix1210_1.1.1
```

```
2014-04-10 14:00:51 7450  7448 /opt/IBM/informix_12.10_1/bin/onbar_d complete, returning 131 (0x8
```


© 2017 IBM Corporation

onbar Cold Restores – bar_act.log (in part, obviously) onbar

- onbar –r (as command line informix) …….
  (skipped a large amount of successful output)

.

.

2014-04-16 10:27:38 4968 4966 Begin restore logical log 199 (Storage Manager copy ID: 0 53).
2014-04-16 10:27:47 4968 4966 Completed restore logical log 199.
2014-04-16 10:27:47 4968 4966 Begin restore logical log 200 (Storage Manager copy ID: 0 58).
2014-04-16 10:27:47 4968 4966 Completed restore logical log 200.
2014-04-16 10:27:47 4968 4966 Begin restore logical log 201 (Storage Manager copy ID: 0 59).
2014-04-16 10:27:47 4968 4966 Completed restore logical log 201.
2014-04-16 10:27:59 4968 4966 Completed logical restore.
2014-04-16 10:27:59 4968 4966 Informix PSM session 77 closed
2014-04-16 10:28:02 4968 4966 /opt/IBM/informix_12.10_1/bin/onbar_d complete, returning 0 (0x00)

- Testing afterward showed intact chunks (onstat –d), good schemas (dbschema –d stores_demo –t city_info3 –ss) and error free output from both the standard informix and mongo environments.

- ontape had similar successful restore results.

- In both cases, ER came up as well. Automatically on reboot.

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Some MongoDB & JSON stuff you can do while connected to Informix

© 2021 IBM Corporation

IBM

JSON & MongoDB (1)

- informix@cte2:~> mongo stores_demo --eval "printjson(db.getCollectionNames())"
  MongoDB shell version: 2.4.5
  connecting to: stores_demo
  [
  "city_info",
  "city_info2",
  "city_info3",
  "city_info4",
  "deliveries1",
  "lowercase"
  ]

- informix@cte2:~> mongo stores_demo --eval "print(db.getCollectionNames())"
  MongoDB shell version: 2.4.5
  connecting to: stores_demo
  city_info,city_info2,city_info3,city_info4,deliveries1,lowercase

- informix@cte2:~> mongo cte2:27017
  MongoDB shell version: 2.4.5
  connecting to: cte2:27017/test
  mongos>

© 2017 IBM Corporation

JSON & MongoDB (2)

- informix@cte2:~> mongo cte2:27017/stores_demo
  MongoDB shell version: 2.4.5
  connecting to: cte2:27017/stores_demo
  mongos>

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Informix NoSQL Aggregation Framework

© 2021 IBM Corporation

Description

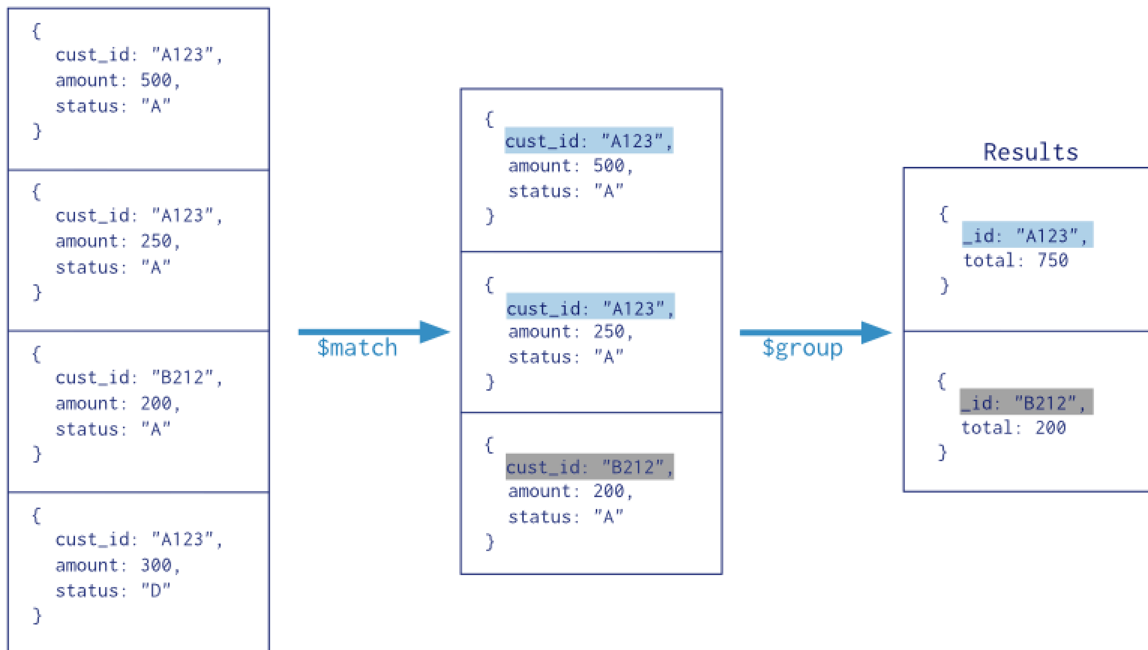
- **Informix support for Mongo's Aggregation Pipeline**
  - Allow users to run aggregation queries on BSON documents
  - Enables users to do more than just retrieve back the same documents they inserted. Now they can run queries to aggregate and summarize groups of documents in a collection.

© 2017 IBM Corporation

Mongo's Aggregation Pipeline

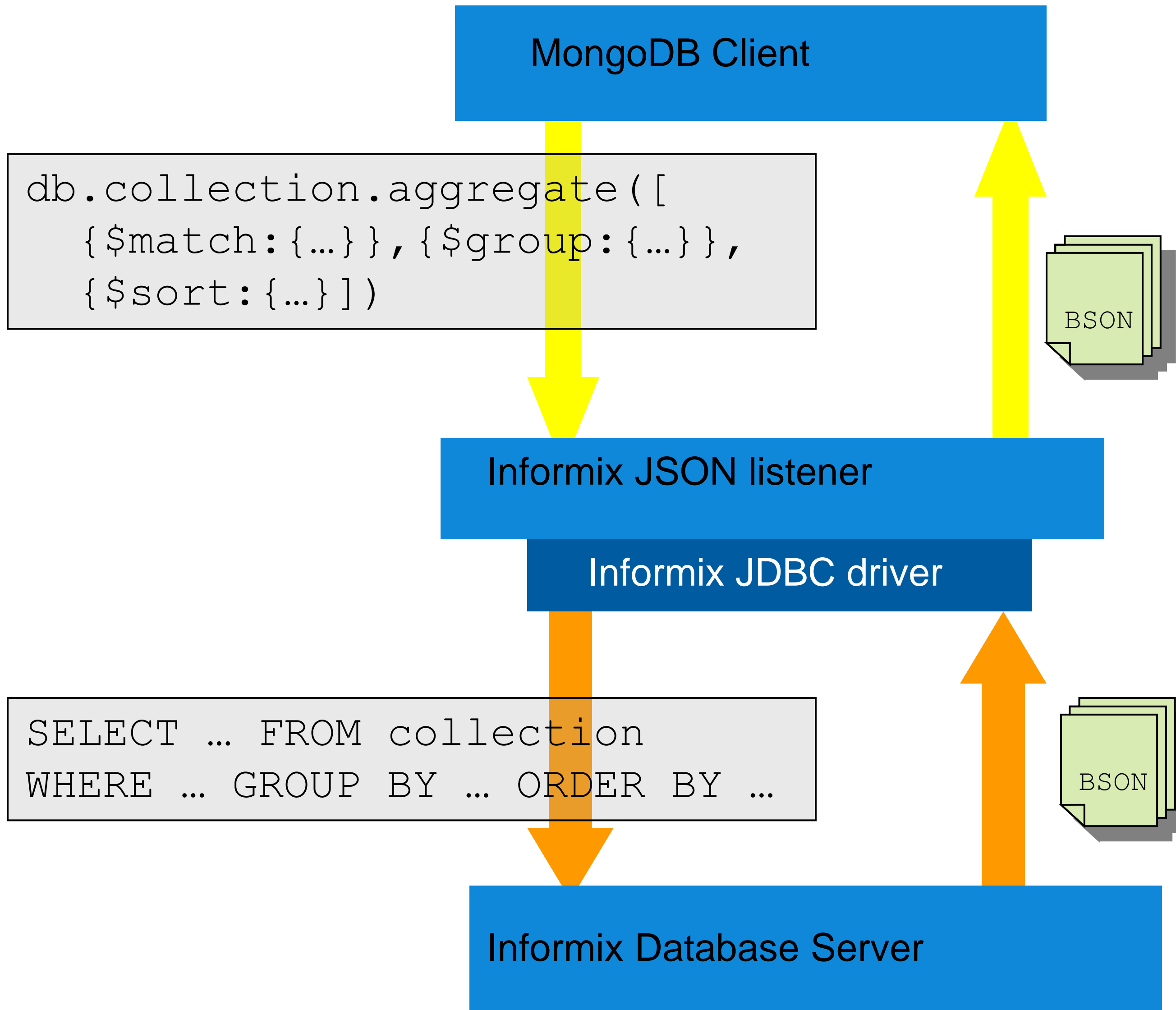
http://docs.mongodb.org/manual/core/aggregation-pipeline/

"The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated results."



© 2017 IBM Corporation

Informix NoSQL Aggregation Queries



The JSON listener will handle pipelining by nesting various derived table queries inside of each other.

© 2017 IBM Corporation

Aggregation Pipeline Operations

$match	**WHERE clause.** Uses standard Mongo query operators that are already supported in xC2.
$group	**GROUP BY clause.** See next slide for group (aggregation) operators.
$unwind	New Informix function **bson_unwind** was created. Takes an array of documents and returns them as a stream of documents
$project	**Projection clause** Support specifying which fields of the documents to include in the projection. Mongo also has a set of projection operators to add new computed fields, as well as rename or rearrange fields in your documents.
$limit	**LIMIT keyword** in projection clause.
$skip	**SKIP keyword** in projection clause.
$sort	**ORDER BY clause.**

Mongo also has a **$geoNear** pipeline operator, which is not supported at this time.

© 2017 IBM Corporation

Group aggregation operators

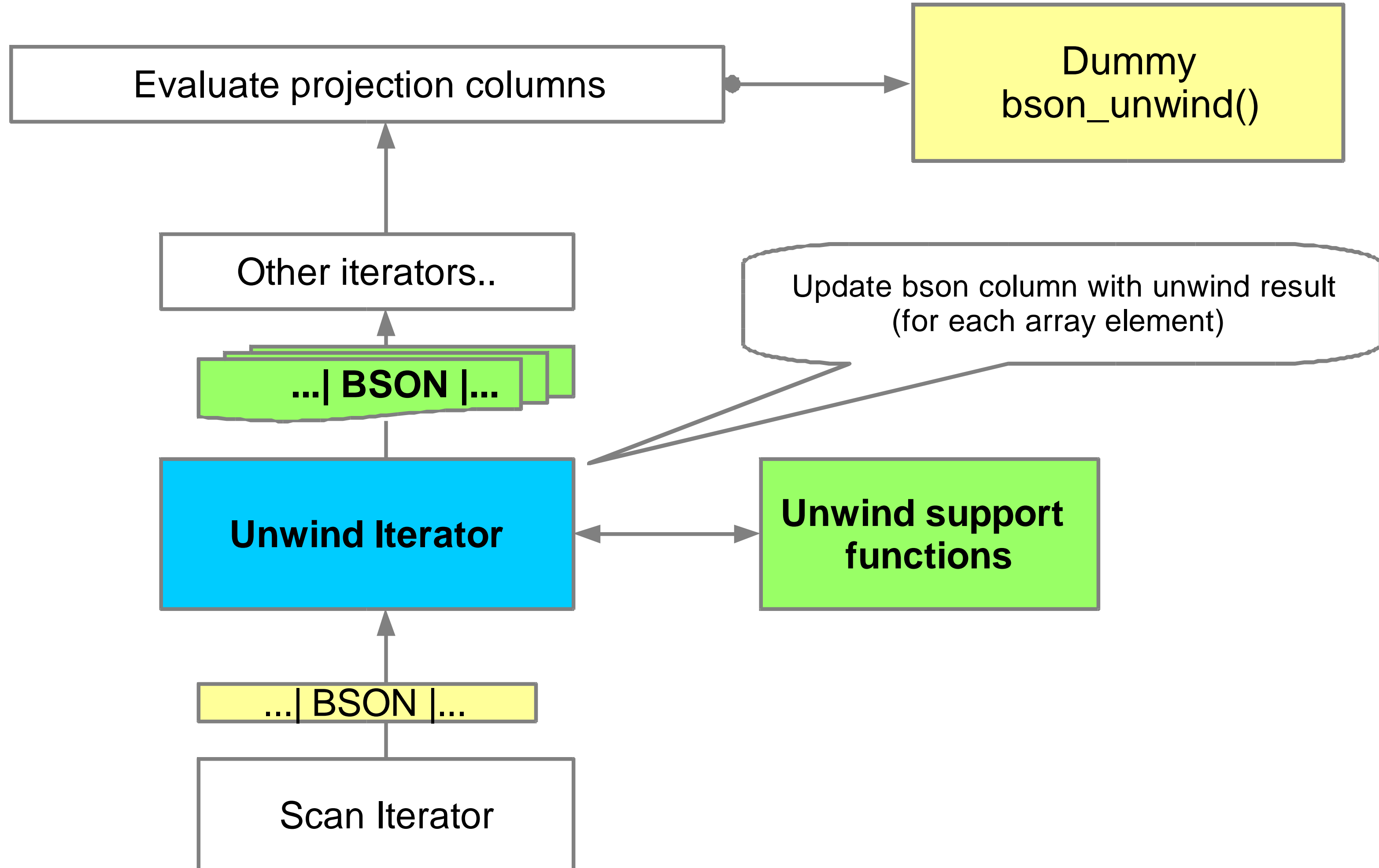
$sum	Use **bson_value_double** to extract value and **SUM**
$avg	Use **bson_value_double** to extract value and compute average
$max	Use **bson_extract** and then apply **MAX** function directly on BSON
$min	Use **bson_extract** and then apply **MIN** function directly on BSON
$addToSet	New Informix **addToSet** function that returns an array of all the *unique* values for the selected field among for each document in that group.
$push	New Informix **push** function that returns an array of *all* values for the selected field among for each document in that group.

Mongo also has **$first** and **$last** group operators, which are not supported at this time.

© 2017 IBM Corporation

bson_unwind

Select _id, **bson_unwind(data, “$tags”)** from tab1;



© 2017 IBM Corporation

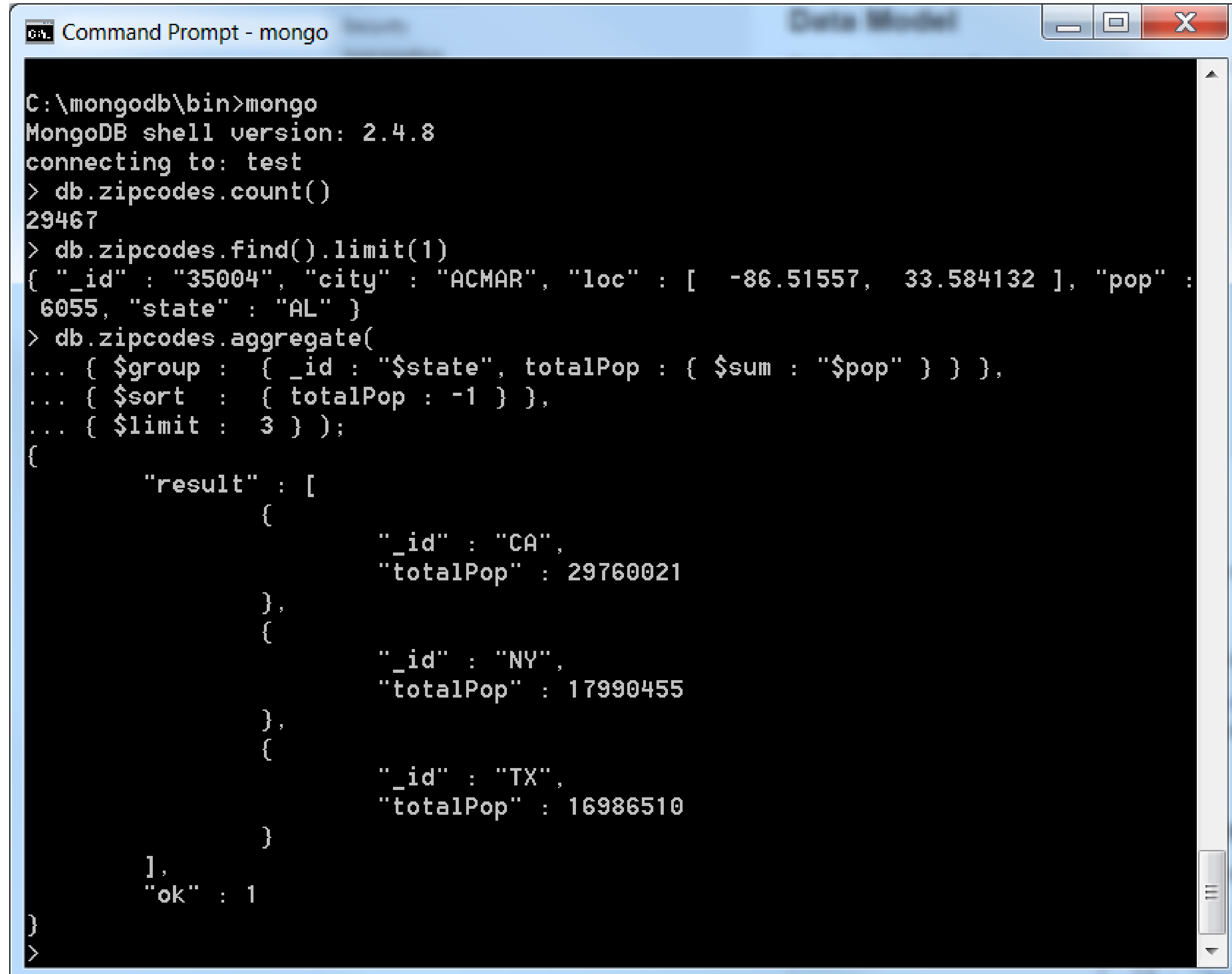
bson_unwind()

- One bson_unwind() per query block
- Unwind one array per query block
- Support Skip M / First N on unwind iterator
- bson_unwind() is allowed only on column references (e.g bson_unwind(bson_concat(...), "$tag") not allowed)
- Materialize a Derived Table with bson_unwind()
- PDQ is disabled

© 2017 IBM Corporation

Use Cases

$group and $sum example: Return 3 states with the largest population



© 2017 IBM Corporation

Use Cases

$unwind example:

- Each article has an array of languages that it is available in.
- Aggregation query returns the number of articles available for each language

Select Command Prompt - mongo

```
> db.articles_collection.count()
14
> db.articles_collection.find().limit(1)
{ "_id" : 1, "title" : "Compare the Informix Version 12 editions", "type" : "article", "date" : "2013-10-01", "tags" : [       "Informix Dynamic Server Version 12",    "IDS",    "Informix",     "Informix Dynamic Server",       "comparison" ], "source" : "developerWorks", "languages" : [  "en",  "cn",  "ru",  "br" ] }
>  db.articles_collection.aggregate(
... {$unwind : "$languages"},
... {$group  : { _id : "$languages" , count : {$sum:1} } },
... {$sort   : { count : -1 } } )
{
        "result" : [
                {
                        "_id" : "en",
                        "count" : 10
                },
                {
                        "_id" : "cn",
                        "count" : 8
                },
                {
                        "_id" : "br",
                        "count" : 4
                },
                {
                        "_id" : "ru",
                        "count" : 3
                },
                {
                        "_id" : "vn",
                        "count" : 2
                },
                {
                        "_id" : "es",
                        "count" : 1
                }
        ],
        "ok" : 1
}
> _
```


© 2017 IBM Corporation

IBM

Use Cases

$addToSet example:

- Return an array of all students in each grade level

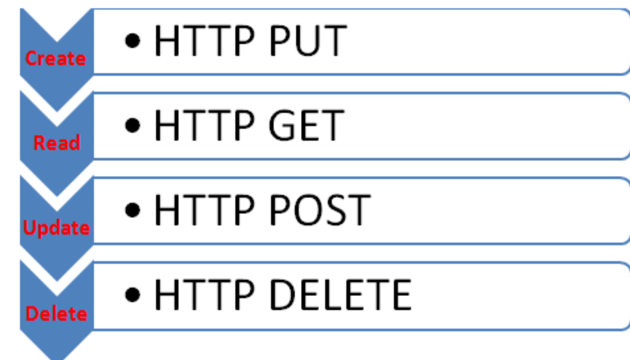
```
Select Command Prompt - mongo
> db.students.find().limit(1)
{ "_id" : ObjectId("52fd3eb5017883651 49a0519"), "name" : "Ben", "grade" : 7, "ph
one" : "555-9876" }
> db.students.aggregate(
... {$group : { _id : "$grade", children : { $addToSet : "$name" } } },
... {$sort  : { _id : 1 } } )
{
        "result" : [
                {
                        "_id" : 6,
                        "children" : [
                                "Susie",
                                "Ava",
                                "Joey",
                                "Janelle",
                                "Jocelyn",
                                "Allison",
                                "Parker",
                                "Kieran"
                        ]
                },
                {
                        "_id" : 7,
                        "children" : [
                                "Ben",
                                "Eddie",
                                "Anne",
                                "Gwen",
                                "Lindsay",
                                "Charlie"
                        ]
                },
                {
                        "_id" : 8,
                        "children" : [
                                "Alex",
```


© 2017 IBM Corporation

Enhanced JSON Compatibility

- **Create and store Timeseries data:**
  - With the REST API or the MongoDB API
- **Access BSON data from JDBC client applications**
- **Quickly export relational tables to BSON or JSON documents**
  - Allows for moving data between environments / architectures

Not Only SQL OR SQL



© 2017 IBM Corporation

Text Search of JSON Documents (NoSQL)

- **Use the Basic Text Search (BTS) database extension to:**
  - Perform basic text search on columns that have **JSON** or **BSON** data types
  - Create the BTS index on **JSON** or **BSON** data types through SQL with the **CREATE INDEX** statement or on **BSON** data types through the Informix extension to MongoDB with the **createTextIndex** command
  - Control how **JSON** and **BSON** columns are indexed by including **JSON** index parameters when you create the basic text search index.
  - Run a basic text query on **JSON** or **BSON** data with the **bts_contains()** search predicate in SQL queries or the **$ifxtext** query operator in **JSON** queries.

© 2017 IBM Corporation

Include JSON documents in timeseries

- **Schema flexibility for Timeseries applications**
  - Add key value pairs as data elements
  - Create a time series that contains **JSON** documents
  - **JSON** documents are unstructured data
  - No schema changes required, if the structure of the data changes
- **Easy to load and query the data**
- **Provides performance benefit over classic timeseries implementation**
- **Allows for multiple timeseries for each device**

© 2017 IBM Corporation

Include JSON documents in timeseries (contd)

- **Use MongoDB and REST API clients**
  - Can load JSON documents directly from MongoDB and REST API clients without formatting the data.
- **Application compatibility**
  - No need for schema migration, therefore minimal changes required for the application.
- **Variable schema**
  - Storing data as unstructured data in JSON documents prevents the need to update your schema or your application.
  - For example, if you have sensors that monitor every machine in a factory, when you add a machine, the new sensors might collect different types of data than existing sensors. Greater flexibility.
- **Simplified schema**
  - If schema for time-based data includes more columns than each record typically uses, or if data records typically contain many NULL values, storing it as unstructured JSON documents makes perfect sense.
  - For example, if there are 50 different measurements but each sensor collects only 5 of those measurements, each record will have 45 NULL values.

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

MongoDB Application Source Drivers & Informix

© 2021 IBM Corporation

MongoDB Drivers – The Power of Hybrid

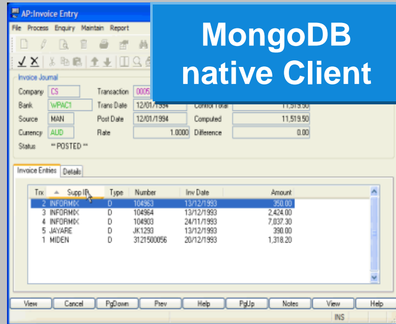
- **One of the big secrets here is what the MongoDB company, formerly known as 10Gen, does not have to do.**
- **Lots of user community support for the product and its drivers.**
- **With the new MongoDB client interface to Informix as a hybrid data store, there are a series of MongoDB officially supported drivers formally listed as the MongoDB "ecosystem" and linked to on the website for MongoDB.**
- **These have support from MongoDB directly as a corporation and also broad support from the MongoDB user community**
- **And now, those users can be part of the Informix "ecosystem" as well.**
- **A whole new universe:**
  - to go, where Informix has not gone before.

© 2017 IBM Corporation

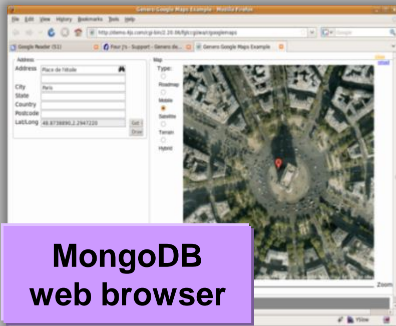
Client Applications

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

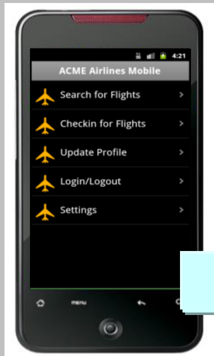
Applications



MongoDB native Client

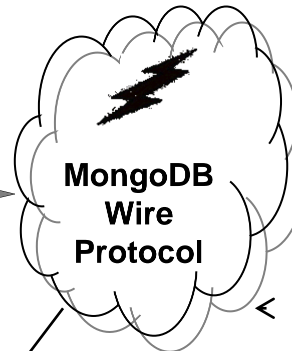


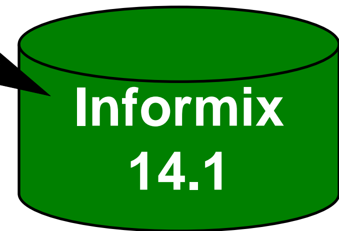
MongoDB web browser



Mobile

MongoDB driver





```
informixva:/data/opt/informix/bin # java -jar $INFORMIXDIR/bin/jsonListener.jar -config $INFORMIXDIR/etc/jsonListener.properties -start
starting listener on port 27017
[main] INFO com.ibm.nosql.informix.server.LwfJsonListener - JSON server listening on port: 27017, 0.0.0.0/0.0.0.0
```


© 2017 IBM Corporation

MongoDB Drivers – Officially Supported (1)

- **With the new MongoDB client interface, with Informix as a hybrid data store, there comes a whole new world of drivers necessary for programmers to access and interface Informix, thru mongo:**

- **C**
  - C Driver Library
- **C++**
  - C++ Driver Library
  - Download and Compile C++ Driver
  - Getting Started with the C++ Driver
  - SQL to mongo Shell to C++
  - C++ BSON Helper Functions
  - C++ BSON Array Examples
- **C#**
  - CSharp Language Center
  - CSharp Community Projects
  - Getting Started with the CSharp Driver
  - CSharp Driver LINQ Tutorial
  - Serialize Documents with the CSharp Driver
  - CSharp Driver Tutorial
- **Erlang**
  - Erlang Language Center

© 2017 IBM Corporation

MongoDB Drivers – Officially Supported (2)

- **Java**
  - Java Language Center
  - Java Types
  - Java Driver Concurrency
  - Java Driver Replica Set Semantics
  - Getting Started with Java Driver
  - Java Driver and Aggregation Framework
  - Java DBObject to Perform Saves
- **JavaScript**
  - JavaScript Language Center
- **Node.js**
  - Node.js Language Center
- **Perl**
  - Perl Language Center
  - Contribute to the Perl Driver
- **PHP**
  - PHP Language Center
  - PHP Libraries, Frameworks and Tools
- **Python**
  - Python Language Center
  - Write a Tumblelog Application with Flask and MongoEngine
- **Ruby**
  - Ruby Language Center
  - Ruby External Resources
  - MongoDB Data Modeling and Rails
  - Getting Started with Rails
  - Getting Started with Rails 3
- **Scala**
  - Scala Language Center

© 2017 IBM Corporation

MongoDB Drivers – Community Supported (1)

- **There is a large and vast MongoDB user community for the support and maintenance of various drivers used in interfacing MongoDB not officially supported by MongoDB.**

- **They are programmers or dba's controlling a database in many cases and are interested in getting solutions to complex programming and database technical issues.**

- **The community experiments with these new technologies, as such represent a new frontier for Informix:**
  - Supporting the drivers used/created thru the standard modern day Internet forums, chat rooms, code repositories, etc.
  - These ecosystems outside of the MongoDB official environment are testament to the large user community following it has developed.
  - In part, its why they presently have 25% market share in the document store aspect of the database marketplace.
  - The drivers are numerous ……… worth looking at briefly.
  - Perhaps one of your customers is using one of these …….

© 2017 IBM Corporation

MongoDB Drivers – Community Supported (2)

- **ActionScript3**
  - http://code.google.com/p/jmcnet-full-mongo-flex-driver/
- **C**
  - libmongo-client
- **ColdFusion**
  - https://github.com/marcesher/cfmongodb/
- **D**
  - Port of the MongoDB C Driver for D
  - Dart
- **Delphi**
  - mongo-delphi-driver
  - pebongo
  - TMongoWire
  - Alcinoe TALMongoClient
  - Synopse mORMot framework
- **Djongo**
  - https://github.com/nesdis/djongo
- **Elixir**
  - Elixir Driver
  - Elixir Driver Alternate
  - Ecto adapter
- **Entity**
  - Entity driver for mongodb on Google Code, included within the standard Entity Library
- **Erlang**
  - Erlang Driver
- **Factor**
  - Factor
- **Fantom**
  - afMongo
  - afMorphia
  - FantoMongo
- **F#**
  - F#
- **Go**
  - mgo (GlobalSign fork)

© 2017 IBM Corporation

MongoDB Drivers – Community Supported (3)

- **Groovy**
  - gmongo
  - Grails Bates:
    - Grails business audit trails plugin
- **Haskell**
  - MongoDB driver for Haskell
- **Javascript**
  - Narwhal
- **Kotlin**
  - KMongo
- **LabVIEW**
  - mongo-labview-driver
- **Lisp**
  - https://github.com/fons/cl-mongo
- **Lua**
  - LuaMongo on Google Code
  - LuaMongo fork on Github
- **Mathematica**
  - MongoDBLink
- **MatLab**
  - mongo-matlab-driver
- **Objective C**
  - NuMongoDB
  - ObjCMongoDB
- **OCaml**
  - Mongo.ml
- **Opa**
  - Opa Standard Library MongoDB Driver
- **PHP**
  - Asynchronous PHP driver using libevent|

© 2017 IBM Corporation

MongoDB Drivers – Community Supported (4)

- **PowerShell**
  - mosh Powershell provider for MongoDB
  - mdbc module cmdlets using the officially supported .NET driver
- **Prolongo**
- **Python**
  - MongoEngine
  - MongoKit
  - Django-nonrel
  - Django-mongodb
  - Django-mongonaut
- **R**
  - rmongodb
- **Racket (PLT Scheme)**
  - mongodb.plt
- **Scala**
  - Reactive-Mongo
- **Smalltalk**
  - Squeaksource MongoTalk
- **Swift**
  - Mongokitten

© 2017 IBM Corporation

Nothing Like the Present …….

- **In addition to connectivity thru the wire listener, we are offering meaningful enhancements to Informix for JSON capabilities:**
  - Native JSON and BSON data storage types
    - With functions necessary to manipulate the data as well, in and out.
  - Hybrid Applications
    - Execute SQL within JSON apps
    - Execute JSON within SQL based apps.
  - Hybrid database
    - Store Relational and non-relational data within the same database
    - Join both data organization types within SQL
  - Standard MongoDB <u>application</u> performance scaling techniques via sharding
    - We do it too, and have for years.
    - We call it Enterprise Replication, and now we store on a node (shard) by key value and load balance the storage across the shards as does MongoDB for scalability.
  - Use standard Informix utilities and techniques on both.
    - Backup/restore
    - Enterprise Replication
    - Compression
    - Time Series,
    - Etc.

© 2017 IBM Corporation

What Data Store Format to Use?

- **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

© 2017 IBM Corporation

Data Normalization - Choose the Right Solution

```
create table department (
    dept char(3),
    deptname varchar(40),
    manager varchar(20),
    empno integer,
    projno integer);

create table employee(
    empno integer
    lastname varchar(20)
    edlevel smallint );

create table project(
    projno integer,
    projname varchar(20),
    respemp  integer);
```

Unique indexes on department.dept, employee.empno, project.projno
Composite index on department (dept, empno, projno)

Constraints (minimally) :

Between department.empno and employee.empno,
Between department .projno and project.projno

- **Possibility exists of mistyped data with the NoSQL schema.**
- **Application joins a must**

- **No mismatched data types on Informix.**
- **No Application Joins.**

NoSQL JSON - Two approaches

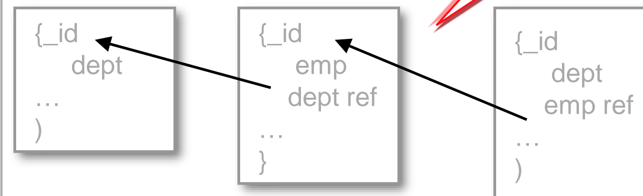
Embedded (de-normalized)

```
{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

Using references

Requires application-side join



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

© 2017 IBM Corporation

Using Schema-less Model In a Traditional Environment

- **During prototyping, early development, validation**
  - As system stabilizes, stable schema parts could be moved to relational columns
- **For cases where web message or document will be retrieved as-is**
  - Yet retrieval on internal fields is needed
- **When parts of the schema will always be dynamic**

© 2017 IBM Corporation

JSON and Informix – Complementary Technologies

- **Does NoSQL mean NoDBA? No Informix?**
  - Definitely not - the relational database isn't going away anytime soon
  - IBM sees JSON as becoming a complementary technology to relational
- **Transactional atomicity is essential for mission critical business transactions**
  - Informix JSON solution brings commits, transaction scope
- **Informix was the first DBMS to store relational and non-relational datatypes side by side 19+ years ago and allow hybrid apps.**
  - Acquisition of Illustra
- **Choice for developers.**

© 2017 IBM Corporation

JSON and Informix – The Hybrid Apps/DB Solution

- **What do we offer a customer/developer – Think Hybrid:**
  - Hybrid database and applications, SQL & NoSQL
  - Single/multi-instance, single/multi-database machines
  - Multi, single statement and partial transactions, two-phase commit
  - Encryption at Rest and on the wires, Backups Encrypted, Compression
  - **Free** backup/restore utilities and storage manager to disk, tape or remote
  - Heterogeneous Table/Whole Instance Replication, Rolling Upgrades, Sharding
  - **Free** Connection Managers to all server types
  - Scale up/out secondary servers, Shared Disk
  - Master/Slave or Master/Master Replication
  - User Defined, Geospatial, Spatial Data Types
  - Time Series, Basic Text Search Data Types
  - Native JSON/BSON, MQTT, REST support
  - Configurable Dynamic Database Server Autonomics, Graphical Administration
  - **Free** Informix Developer/Innovator-C Edition
  - Hosted on Multiple Clouds and BYOL to Cloud
  - In-memory Informix Warehouse Accelerator
  - Legendary Up-Time ‘**5 9’s**’, Stable, Easy “stand-up” time measured in years

- **All of this on one or <u>possibly</u> two physical machines (if replicating), one user database in one instance, one electricity bill, one cost**

© 2017 IBM Corporation

Informix vs Oracle Competitive

- **Informix has lower TCO:**
  - Oracle is notorious for giving the product away for free for a year and then more than making up for it in immediate years ahead.
    - Lots of dbas needed in Oracle as well.
      - One Informix customer has 44,000+ instances and 5 dbas
      - Doubtful Oracle can run 44,000+ Oracle instances with 5 dbas
    - Large S & S bills and hard to get away due to serious database conversion issues

- **Vendor LOCK: Oracle locks in their users, once converted, for life by converting their in house apps to Oracle Apps if applicable**
  - There are no known methods of converting the Oracle database and Oracle Apps once both are in place at a customer site:
    - Presently Oracle Cloud is in its infancy, and growing, and so it is giving that away too
    - Informix on Cloud is growing too, we don't lock in our users for life, we don't have App Suites
- **Web apps based on NoSQL DB's are the growing segment**
  - Oracle sells a separate product for these NoSQL apps, and logically separate hardware is needed as well
  - With Informix you can do it all in server and not incur those extra electricity and other operating costs with a minimum number of dbas.

© 2017 IBM Corporation

Informix vs SQL Server Competitive

- **SQL Server just announced a Linux release that came out in late 2017**
  - Informix had multiple Linux releases for 10 years
  - If SQL Server is so good on Windows why Linux now?

- **It basically only runs on Windows**
  - Has a reputation for not being able to handle large scale Enterprise workloads
  - Good for small to lower mid size workloads
  - Which is why it costs less at small to mid-size

- **No good virtualization licensing model, buy in 2's**
  - Minimum purchase of 4 core physical and virtualized

- **Client based encryption of everything (disk, network, memory, backups) – leads to overheated clients**
  - Informix encrypts at rest disk storage and also backups and restores (storage and logical log backups) thru configuration and there is an option to do partial storage encryption.
  - Network communications encryption from server to server and client to server are serviced thru a wide variety of methodologies and ciphers

© 2017 IBM Corporation

IBM

JSON and Informix – User Experience Quotes

- **“This is just the beginning of the possibilities for this hybrid development structure. I believe this structure will be more common in the future of app creation than simply choosing only SQL or only NoSQL. The more tools are given to developers, the more creative they can be and more choices they have for tackling their problems. That is what developers want, reliable and varied tools to solve their problems. Informix is working hard to provide that with the JSON Listener. This technology will help drive the next generation of web based applications in an industry where time is precious, rapid prototyping is preferred and scalability is key. With NoSQL capabilities infused to IDS, Informix is off to a great start.” –**
  Hector Avala – Software Developer and College Student

© 2017 IBM Corporation

Links, Helpful Information and Resources

- **Informix CTP wiki**
- **Informix Developer Edition free download:**
  https://www.ibm.com/account/reg/us-en/signup?formid=urx-32091Informix Virtual Appliance
  - Requires login
- **Informix Innovator-C Edition free download:**
  https://mrs-ux.mrs-prod-7d4bdc08e7ddc90fa89b373d95c240eb-0000.us-south.containers.appdomain.cloud/marketing/iwm/platform/mrs/assets/packageList?source=ifxdl&lang=en_US
  - Requires login
- **MongoDB site**
- **MongoDB download: http://www.mongodb.org/downloads**
- **MongoDB drivers**
- **Informix NoSQL Compatibility Guide**

© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

REST and Informix

© 2017 IBM Corporation

REST Definition

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

- Roy Fielding

© 2017 IBM Corporation

REST Architecture

- **Distributed communication architecture**
- **Widely popular in cloud environments**
- **REST**
  – An architectural style for web based communication
  – Permits clients to communicate with servers in a unique manner
  – Represents resources (databases in this case) as URI's
  – Architecture uses **HTTP** protocol
  – A set of operations (**GET**/**POST**/**PUT**/**DELETE**) permit manipulation of resources
- **RESTful architectures are stateless**
  – Server does not maintain any client context between transactions
  – Transaction must contain all information necessary to satisfy the particular request.
  – Makes **REST**ful architectures more reliable and also helps to expand their scalability.

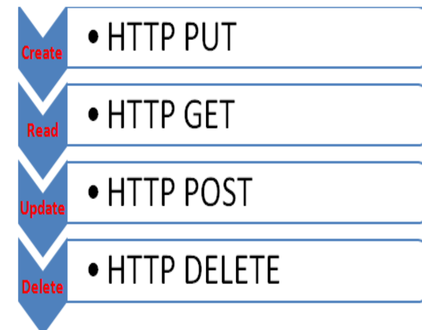
The strength of REST
REST is an architectural style, not a protocol or an implementation. REST has some core principles, but in the end, it's an abstraction, not a specific implementation.
(Source: http://www.ibm.com/developerworks/library/os-understand-rest-ruby/)

© 2017 IBM Corporation

Access Informix from REST API Clients

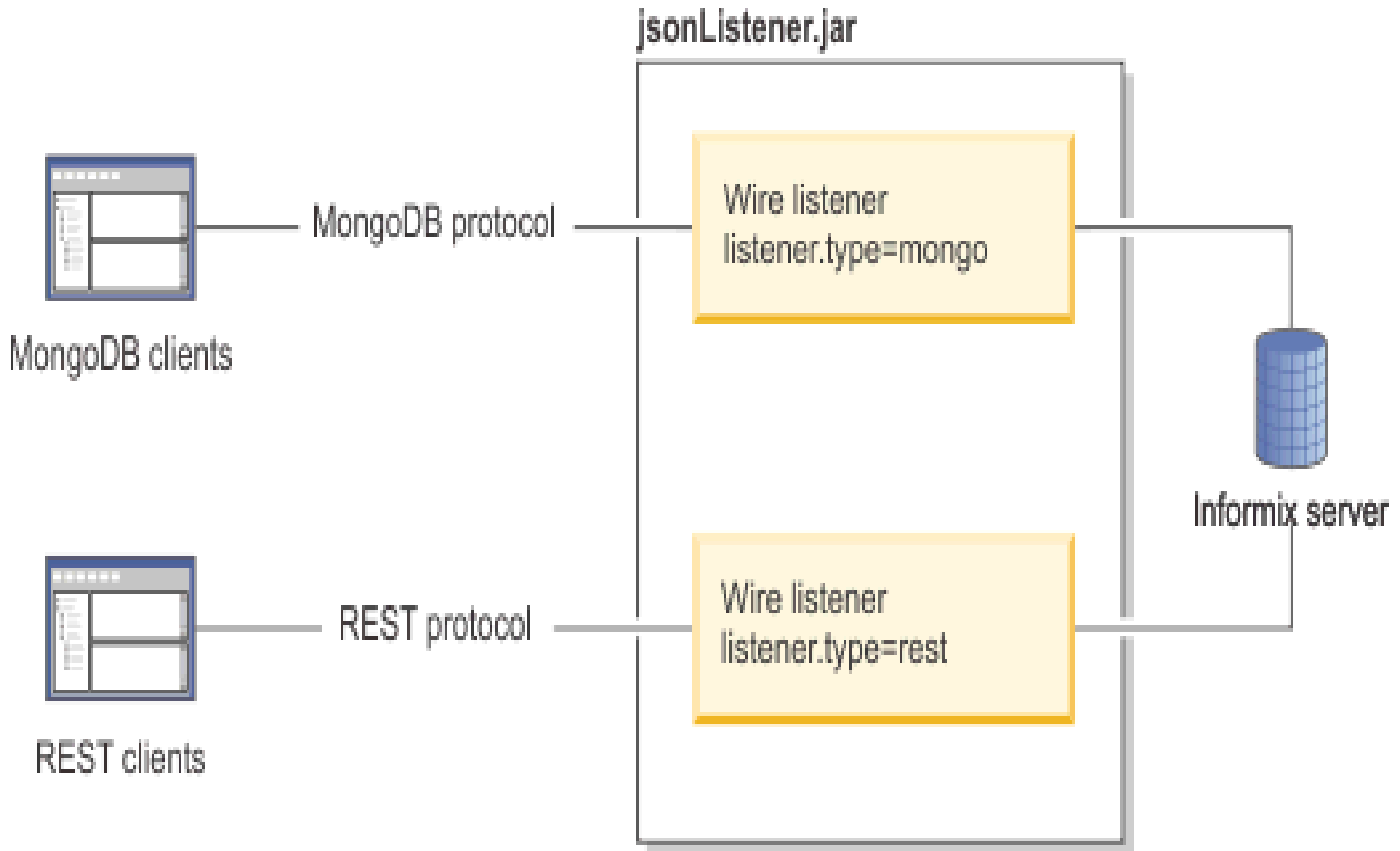
- **Directly connect applications or devices that communicate through the REST API to Informix**
  - No client side drivers needed, freedom from client dependency
  - Web based applications can connect seamlessly to the database using HTTP protocol
  - Create connections by configuring the wire listener for the REST API
  - Use MongoDB and SQL queries against JSON and BSON document collections, traditional relational tables, and time series data
  - The REST API uses MongoDB syntax and returns JSON documents
  - Widely popular in Cloud / IOT architectures
  - Simplify web application development in Cloud environments

- **A subset of the HTTP protocol (GET / POST / DELETE / PUT) supported**
  - **POST** method maps to mongo db insert or create command
  - **GET** method maps to mongo db query command
  - **PUT** method maps to mongo db update command
  - **DELETE** method maps to mongo db delete command



© 2017 IBM Corporation

Access Informix from REST API Clients (contd)



© 2017 IBM Corporation

Wire Listener & REST (1)

- **The wire listener is a mid-tier gateway server that enables communication between MongoDB applications and the Informix® database server.**

- **The wire listener is provided as an executable JAR file that is named $INFORMIXDIR/bin/jsonListener.jar. The JAR file provides access to the MongoDB API and REST API.**

- **You can connect to a JSON collection by using the REST API.**

- **When a client is connected to the wire listener by using the REST API, each database is registered; session events such as create or drop a database.**

- **If a REST request refers to a database that exists but is not registered, the database is registered and a redirect to the root of the database is returned.**

© 2017 IBM Corporation

Wire Listener & REST (2)

- **The JSONListener.properties file has an optional parameter called listener.type It specifies the type of wire listener to start:**
  - The default is **mongo** which connects the wire listener to the MongoDB API
    - **listener.type=mongo**
- **To connect to a REST API, connect to the wire listener, connect the wire listener to the REST API using the following parameter value which must be specified to use the REST API:**
  - **listener.type=rest**
- **There are some new REST related optional parameters for the JSONListener.properties file which may be necessary for use.**

© 2017 IBM Corporation

Multiple wire listeners configuration (1)

- **Run multiple wire listeners at the same time to access both Mongo and REST data, by creating a properties file for each:**
  - Create each properties file in the **$INFORMIXDIR/etc** directory using the **$INFORMIXDIR/etc/jsonListener-example.properties** file as a template.
  - Customize each properties file and assign a unique name:
    - The **url** parameter **must** be specified, either in each individual properties file or in the file that is referenced by the include parameter.
      - Optional: Specify the **include** parameter to reference another properties file.
      - The properties file path can be relative or absolute.
    - If you have multiple properties files, you can avoid duplicating parameter settings in the multiple properties files by specifying a subset of shared parameters in a single properties file, and the unique parameters in the individual properties files.
  - Start the wire listeners.

© 2017 IBM Corporation

Multiple wire listeners configuration (2) - Example

- The same url, authentication.enable, and security.sql.passthrough parameters are used to run two separate wire listeners:

- Create a properties file named shared.properties that includes the following parameters
  - :url=jdbc:informix-sqli://localhost:9090/sysmaster: INFORMIXSERVER=lo_informix1210; authentication.enable=true security.sql.passthrough=true

- Create a properties file for use with the MongoDB API that is named mongo.properties, with the parameter setting include=shared.properties included:
  - include=shared.properties listener.type=mongo listener.port=27017

- Create a properties file for use with the REST API that is named rest.properties, with the parameter setting include=shared.properties included:
  - include=shared.properties listener.type=rest listener.port=8080

© 2017 IBM Corporation

Multiple wire listeners configuration (3) - Example

- **Start the wire listeners by using the command line:**
  - **java -jar jsonListener.jar -start -config json.properties -config rest.properties**

© 2017 IBM Corporation

HTTP: POST

Method	Path	Description
POST	/	Create a database
POST	/databaseName	Create a collection **databaseName** – database name
POST	/databasename/collectionName	Create a document **databaseName** – database name **collectionName** – collection name

© 2017 IBM Corporation

HTTP: POST – Create a database

- **With the locale specified.**

- **Request: Specify the POST method:**
  - POST / Data:

- **Specify database name mydb and an English UTF-8 locale:**
  - {name:"mydb",locale:"en_us.utf8"}

- **Response: The following response indicates that the operation was successful:**
  - Response does not contain any data.

© 2017 IBM Corporation

HTTP: POST – Collection Creation

- Creates a collection in the mydb database.
- Request: Specify the POST method and the database name as mydb:
  - POST /mydb
- Data: Specify the collection name as bar:
  - {name:“bar”}
- Response: The following response indicates that the operation was successful:
  - {"msg":"created collection mydb.bar","ok":true}

© 2017 IBM Corporation

HTTP: POST – Relational Table Creation

- This example creates a relational table in an existing database.

- Request: Specify the POST method and stores_mydb as the database:
  - POST /stores_mydb

- Data: Specify the table attributes:
  - { name: "rel", columns: [{name:"id",type:"int",primaryKey:true,}, {name:"name",type:"varchar(255)"}, {name:"age",type:"int",notNull:false}]}

- Response: The following response indicates that the operation was successful:
  - {msg: "created collection stores_mydb.rel" ok: true}

© 2015 IBM Corporation

HTTP: POST – Insert a Single Document

- **Inserts a document into an existing collection.**
- **Request: Specify the POST method, mydb database, and people collection:**
  - **POST /mydb/people**
- **Data: Specify John Doe age 31:**
  - **{firstName:"John",lastName:"Doe",age:31}**
- **Response: Because the _id field was not included in the document, the automatically generated _id is included in the response. Here is a successful response:**
  - **{"id":{"$oid":"537cf433559aeb93c9ab66cd"},"ok":true}**

© 2017 IBM Corporation

HTTP: POST – Insert Multiple Documents

- This example inserts multiple documents into a collection.
- Request: Specify the POST method, mydb database, and people collection:
  - POST /mydb/people
- Data: Specify John Doe age 31 and Jane Doe age 31:
  - [{firstName:"John",lastName:"Doe",age:31}, {firstName:"Jane",lastName:"Doe",age:31}]
- Response: Here is a successful response:
  - {ok: true}

© 2015 IBM Corporation

HTTP: GET

- **The GET method maps to the MongoDB query command.**

Method	Path	Description
GET	**/**	List all databases
GET	**/databaseName**	List all collections in a database **databaseName** – database name
GET	**/databasename/collectionName?*queryParameters***	Query a collection **databaseName** – database name **collectionName** – collection name ***queryParameters*** - The query parameters. The supported Informix ***queryParameters*** are: **batchSize**, **query, fields**, and **sort**. These map to the equivalent MongoDB **batchSize**, **query**, **fields**, and **sort** parameters.

© 2017 IBM Corporation

HTTP: GET – List All Databases on the Server

- Specify the GET method and forward slash (/):
  - GET /

- Data: None.

- Response: Here is a successful response:
  - ["mydb" , "test"]

© 2017 IBM Corporation

HTTP: GET – List All Collections in a Database

- **Request: Specify the GET method and mydb database:**
  - **GET /mydb**

- **Data: None.**

- **Response: Here is a successful response:**
  - **["bar"]**

© 2017 IBM Corporation

HTTP: GET – Show Sorted Data from a Collection

- **This example sorts the query results in ascending order by age.**

- **Request: Specify the GET method, mydb database, people collection, and query with the sort parameter.**
  - The sort parameter specifies ascending order **(age:1),** and filters id (**_id:0**) and last name (**lastName:0**) from the response
  - **GET /mydb/people?sort={age:1}&fields={_id:0,lastName:0}**

- **Data: None.**

- **Response: The first names are displayed in ascending order with the _id and lastName filtered from the response:**
  - **[{"firstName":"Sherry","age":31}, {"firstName":"John","age":31}, {"firstName":"Bob","age":47}, {"firstName":"Larry","age":49}]**

© 2017 IBM Corporation

HTTP: PUT

- **The PUT method maps to the MongoDB update command.**

Method	Path	Description
PUT	**/databasename/collectionName ?*queryParameters***	Update a document **databaseName** – database name **collectionName** – collection name **queryParameters -** The supported Informix ***queryParameters*** are **query**, **upsert**, and **multiupdate.** These map to the equivalent MongoDB query, insert, and multi query parameters, respectively.**-**

© 2017 IBM Corporation

HTTP: PUT – Document Update in a Collection

- **Update the value for Larry in an existing collection, from age 49 to 25:**
  - [{"_id":{"$oid":"536d20f1559a60e677d7ed1b"},"firstName":"Larry" ,"lastName":"Doe","age":49},{"_id":{"$oid":"536d20f1559a60e677d7ed1c"} ,"firstName":"Bob","lastName":"Doe","age":47}]

- **Request: Specify the PUT method and query the name Larry:**
  - PUT /?query={name:"Larry"}

- **Data: Specify the MongoDB $set operator with age 25:**
  - {"$set":{age:25}}

- **Response: Here is a successful response:**
  - {"n":1,"ok":true}

© 2017 IBM Corporation

HTTP: DELETE

- **The DELETE method maps to the MongoDB delete command.**

Method	Path	Description
DELETE	**/**	Delete all databases
DELETE	**/databaseName**	Delete a database **databaseName** – database name
DELETE	**/databasename/collectionName**	Delete a collection **databaseName** – database name **collectionName** – collection name
DELETE	**/databasename/collectionName ?queryParameter**	Delete a document **databaseName** – database name **collectionName** – collection name ***queryParameter*** - The query parameter. The supported Informix ***queryParameter*** is **query**. This maps to the equivalent MongoDB query parameter.

© 2017 IBM Corporation

HTTP: DELETE (1) – Database Deletion

- **Delete a database called mydb.**
- **Request: Specify the DELETE method and the mydb database:**
  - DELETE /mydb
- **Data: None.**
- **Response: Here is a successful response:**
  - {msg: "dropped database"ns: "mydb"ok: true}

© 2017 IBM Corporation

HTTP: DELETE (2) – Collection deletion

- **This example deletes a collection from a database.**
- **Request: Specify the DELETE method, mydb database, and bar collection:**
  - **DELETE /mydb/bar**
- **Data: None.**
- **Response: Here is a successful response:**
  - **{"msg":"dropped collection""ns":"mydb.bar""ok":true}**

© 2017 IBM Corporation

Some Selected JSON Improvements

© 2021 IBM Corporation

Agenda – JSON

- **Manipulate JSON & BSON data via SQL**
- **High Availability for MongoDB and REST clients**
- **Wire Listener configuration enhancements**
- **Wire Listener query support**
- **Enhanced user account management through the wire listener**

© 2017 IBM Corporation

High Availability for MongoDB and REST clients

- **To provide high availability to client applications:**
  - REST clients use a reverse proxy for multiple wire listeners.
  - MongoDB clients use a HA cluster configuration for Informix database servers.

- **Each database server in the cluster has a directly connected wire listener on the same computer as the database server that the wire listener is connected to and all wire listeners run on port 27017.**
  - **http://docs.mongodb.org/meta-driver/latest/legacy/connect-driver-to-replica-set/**

- **To provide high availability between the wire listener and the Informix database server, use one of the following methods:**
  - Route the connection via the Connection Manager between the wire listener and the database server.
    - Known methods
  - Configure the **url** parameter in the wire listener configuration file to use one of the Informix JDBC Driver methods of connecting to a high-availability cluster, via a dynamic reading of the **sqlhosts** file.
    - Has been enhanced in 12.10.xC5

© 2017 IBM Corporation

Wire Listener Configuration File Enhancements

- **The wire listener configuration file can be any name, and there can be many of them created in a HA cluster, as long as each file is created in $INFORMIXDIR/etc and has a required .properties file name suffix.**
  - Use **$INFORMIXDIR/etc/jsonListener-example.properties** as a template.
    - Copy it first; DON'T edit it directly.

- **To include parameters in the wire listener, you must uncomment the row and customize parameters with the default values in the copy of the original template file.**

- **The url parameter is required. All other parameters are optional.**
  - Review the defaults for the following parameters and verify that they are appropriate for your environment:
    - authentication.enable
    - listener.type
    - listener.port
    - listener.hostName

© 2017 IBM Corporation

Wire Listener Configuration File – url Parameter

- Specifies the host name, database server, user ID, and password that are used in connections to the database server.

- You must specify the sysmaster database in the url parameter; the wire listener uses sysmaster for administrative purposes.

  >>-url=--jdbc:informix-sqli://*hostname*:*portnum*--/sysmaster:-----> >--+-----------------------------+----------------------->< '-USER=*userid*;--PASSWORD=*password*-'

- You can now include additional JDBC properties, each semi-colon ';' separated with a semi-colon in the url parameter such as:
  - INFORMIXCONTIME
  - INFORMIXCONRETRY
  - LOGINTIMEOUT
  - IFX_SOC_TIMEOUT

© 2017 IBM Corporation

listener.hostName Wire Listener Parameter

- **Specifies the host name of the wire listener. The host name determines the network adapter or interface that the wire listener binds the server socket to.**

- **To enable the wire listener to be accessed by clients on remote hosts, turn on authentication by using the authentication.enable parameter.**

```
                         .--localhost--.
>>-listener.hostName=--+-hostname--+-----------------------><
                         '-*-------------'
```

- **localhost**
  - Bind the wire listener to the **localhost** address. The wire listener is not accessible from clients on remote machines. Default value.
- **hostname**
  - The host name or IP address of host machine where the wire listener binds to.
- *****
  - The wire listener can bind to all interfaces or addresses.

© 2017 IBM Corporation

collection.informix.options Wire Listener Parameter (1)

- **Specifies table options for shadow columns or auditing to use when creating a JSON collection.**

```
                                  .-,-----------------.
                                  V                   |
>>-collection.informix.options=[-----+-------------+----+---]----->< 
                                  +-"audit"--------+
                                  +-"crcols"-------+
                                  +-"erkey"--------+
                                  +-"replcheck"----+
                                  '-"vercols"------'
```


© 2017 IBM Corporation

collection.informix.options Wire Listener Parameter (2)

- **audit**
  - Uses *the **CREATE*** **TABLE** statement **AUDIT** option to create a table to be included in the set of tables that are audited at the row level if selective row-level is enabled.
- **crcols**
  - Uses the **CREATE TABLE** statement **CRCOLS** option to create the two shadow columns that Enterprise Replication uses for conflict resolution.
- **erkey**
  - Uses the **CREATE TABLE** statement **ERKEY** option to create the **ERKEY** shadow columns that Enterprise Replication uses for a replication key.
- **replcheck**
  - Uses the **CREATE TABLE** statement **REPLCHECK** option to create the **ifx_replcheck** shadow column that Enterprise Replication uses for consistency checking.
- **vercols**
  - Uses the **CREATE TABLE** statement **VERCOLS** option to create two shadow columns that Informix uses to support update operations on secondary servers.

© 2017 IBM Corporation

command.listDatabases.sizeStrategy (1)

- **Wire listener parameter specifying a strategy to calculate the size of your database when the MongoDB listDatabases command is run.**
- **The listDatabases command estimates the size of all collections and collection indexes for each database:**
  - Relational tables and indexes are excluded from this size calculation.
- **Performs expensive and CPU-intensive computations on the size of each database in the database server instance.**
  - You can decrease the expense by using the **command.listDatabases.sizeStrategy** parameter.

```
                                         .---estimate-------------.
>>-command.listDatabases.sizeStrategy=---+-{estimate:n}----------+---><
                                         +-compute---------------+
                                         +-none------------------+
                                         '-perDatabaseSpace-'
```


© 2017 IBM Corporation

command.listDatabases.sizeStrategy (2)

- **estimate**
  - Estimate the database size by sampling documents in every collection; this is the default value.
  - This strategy is the equivalent of {estimate: 1000}, which takes a sample size of 0.1% of the documents in every collection; this is the default value.

  command.listDatabases.sizeStrategy=estimate

- **estimate: n**
  - Estimate the database size by sampling one document for every **n** documents in every collection. The following example estimates the collection size by using sample size of 0.5% or 1/200th of the documents:

  command.listDatabases.sizeStrategy={estimate:200}

© 2017 IBM Corporation

command.listDatabases.sizeStrategy (3)

- **compute**
  - Compute the exact size of the database.

  command.listDatabases.sizeStrategy=compute

- **none**
  - List the databases but do not compute the size.
  - The database size is listed as 0.

  command.listDatabases.sizeStrategy=none

- **perDatabaseSpace**
  - Calculates the size of a tenant database created by multi-tenancy feature by adding the sizes for all dbspaces, sbspaces, and blobspaces that are assigned to the tenant database.

© 2017 IBM Corporation

fragment.count Wire Listener Parameter

- **Specifies the number of fragments to use when creating a collection.**
  - **0**
    - •The database server determines the number of collection fragments to create. Default.
  - ***fragment_num*** > 0,
    - •Number of collection fragments created at collection creation.

```
                    .-0--------------------.
>>-fragment.count=--+-fragment_num-+-------------------------><
```


© 2017 IBM Corporation

jdbc.afterNewConnectionCreation

- **Wire listener parameter specifies one or more SQL commands to run after a new connection to the database is created.**

```
                                     .-,-------------.
                                      V              |
>>-jdbc.afterNewConnectionCreation=[---"sql_command"-+-]-------><
```

- **For example, to accelerate queries run through the wire listener by using the Informix Warehouse Accelerator:**

  jdbc.afterNewConnectionCreation=["SET ENVIRONMENT USE_DWA 'ACCELERATE ON'"]

© 2017 IBM Corporation

IBM

authentication.enable Wire Listener Parameter (1)

- **Specifies whether to enable user authentication.**
- **Authentication of MongoDB clients occurs in the wire listener, not in the database server.**
  - –Privileges are enforced by the wire listener.
- **All communications that are sent to the database server originate from the user that is specified in the url parameter, regardless of which user was authenticated.**
- **User information and privileges are stored in the system_users collection in each database.**
- **MongoDB authentication is done on a per database level, whereas Informix authenticates to the instance.**

© 2017 IBM Corporation

authentication.enable Wire Listener Parameter (2)

```
                           .-false-.
>>-authentication.enable=--+-true--+--------------------------><
```

- **false**
  - Do not authenticate users.
  - This is the default value.
- **True**
  - Authenticate users.
  - Use the **authentication.localhost.bypass.enable** parameter to control the type of authentication.

© 2017 IBM Corporation

Wire Listener Logging – Default Logback Mechanism (1)

- **The wire listener can output trace, debug, informational messages, warnings, and error information to a log.**

- **Logback is pre-configured and installed along with the JSON components.**

- **If you start the wire listener from the command line, you can specify the amount of detail, name, and location of your log file by using the -loglevel and -logfile command-line arguments.**
  - If you have customized the **Logback** configuration or specified another logging framework, the settings for **-loglevel** and **-logfile** are ignored.

© 2017 IBM Corporation

Wire Listener Logging – Default Logback Mechanism (2)

- **If the wire listener is started automatically after you create a server instance or if you run the task() or admin() function with the start json listener argument, errors are sent to a log file:**
  - UNIX:
    - The log file is in **$INFORMIXDIR/jsonListener.log**.
  - Windows:
    - The log file is named ***servername*_jsonListener.log** and is in your home directory. **C:\Users\ifxjson\ol_informix1210_5_jsonListener.log.**

© 2017 IBM Corporation

Enhanced Account Management Via the Wire Listener

- **Control user authorization to Informix databases through the wire listener by locking and unlocking user accounts or individual databases via the new Informix JSON lockAccount and unlockAccounts commands.**

© 2017 IBM Corporation

JSON – lockAccounts – Lock a Database/User Account

- If you specify the lockAccounts:1 command without specifying a db or user argument, all accounts in all databases are locked.
- Run this command as instance administrator.

- Syntax:

```
>>-lockAccounts:----1,-+-----------------------------------------------------------------------------+----------><
                       +-db:-+-"database_name"---------------------------------------------------+-"---+
                       |     |   .-,---------------------------.                                  |     |
                       |     |   V                             |                                  |     |
                       |     +-[---"database_name"-+-]-----------------------------------------+     |
                       |     +-{"$regex":"json_document"}----------------------------------+     |
                       |     |   .-,--------------------------------------------------------------|    |     |
                       |     |   V                                                                |    |     |
                       |     '-{---+-"include":-+-"database_name"----------------+-+-+-}-'     |
                       |           |            |  .-,-------------------------.         | |          |
                       |           |            |  V                           |         | |          |
                       |           |            +-[---"database_name"-+-]---------+ |          |
                       |           |            '-{"$regex":"json_document"}---' |          |
                       |           '-"exclude":-+-"database_name"----------------+--'          |
                       |                        |  .-,---------------------------.     |          |
                       |                        |  V                             |     |          |
                       |                        +-[---"database_name"-+-]----+              |
                       |                        '- {"$regex":"json_document"}-'              |
                       '-user:-+-"user_name"------+--------------------------------------------------------'
                               '-"json_document"-'
```


© 2017 IBM Corporation

JSON – lockAccounts – Lock a Database/User Account

- **lockAccounts:1**
  - Required parameter locks a database or user account.

- **db**
  - Optional parameter specifies the database name of an account to lock.
  - For example, to lock all accounts in database that is named **foo**:

```
db.runCommand({lockAccounts:1,db:"foo"})
```

- **exclude**
  - Optional parameter specifies the databases to exclude.
  - For example, to lock all accounts on the system except those in the databases named **alpha** and **beta**:

```
db.runCommand({lockAccounts:1,db:{"exclude":["alpha","beta"]})
```


© 2017 IBM Corporation

JSON – lockAccounts – Lock a Database/User Account

- **include**
  - Optional parameter specifies the databases to include.
  - To lock all accounts in the databases named **delta** and **gamma**:

```
db.runCommand({lockAccounts:1,db:{"include":["delta","gamma"]})
```

- **$regex**
  - Optional evaluation query operator selects values from a specified JSON document.
  - To lock accounts for databases that begin with the character **a.** and end in **e**:

```
db.runCommand({lockAccounts:1,db:{"$regex":"a.*e"})
```

- **user**
  - Optional parameter specifies the user accounts to lock.
  - For example, to lock the account of all users that are not named **alice**:

```
db.runCommand({lockAccounts:1,user:{$ne:"alice"}});
```


© 2017 IBM Corporation

JSON – unlockAccounts – Unlock a Database/User Account

- If you specify the unlockAccounts:1 without specifying a db or user argument, all accounts in all databases are unlocked.
- Run this command as instance administrator.

```
>>-unlockAccounts:------1,-+-------------------------------------------------------------------------------+----><
                           +-db:-+-"database_name"-----------------------------------------------+-"-+
                           |     |  .-,---------------------------.                              |   |
                           |     |  V                             |                              |   |
                           |     +-[---"database_name"-+-]-------------------------------------+   |
                           |     +-{"$regex":"json_document"}----------------------------------+   |
                           |     |  .-,---------------------------------------------------------.  |   |
                           |     |  V                                                           |  |   |
                           |     '-{---+-"include":-+-"database_name"------------+-+-+-}-'       |
                           |           |          |  .-,---------------------------.       |  |           |
                           |           |          |  V                             |       |  |           |
                           |           |          +-[---"database_name"-+-]-----   -+  |           |
                           |           |          '-{"$regex":"json_document"}----' |           |
                           |           '-"exclude":-+-"database_name"-----------+-'           |
                           |                        |  .-,--------------------------.      |           |
                           |                        |  V                            |      |           |
                           |                        +-[---"database_name"-+-]---+           |
                           |                        '-{"$regex":"json_document"}-'          |
                           '-user:-+-"user_name"-----+-----------------------------------------------'
                                   '-"json_document"-'
```


© 2017 IBM Corporation

JSON – unlockAccounts – Unlock a Database/User Account

- **unlockaccounts:1**
  - Required parameter unlocks a database or user account.

- **db**
  - Optional parameter specifies the database name of an account to unlock.
  - To unlock all accounts in database that is named **foo:**

```
db.runCommand({unlockAccounts:1,db:"foo"})
```

- **exclude**
  - Optional parameter specifies the databases to exclude.
  - To unlock all accounts on the system except those in the databases named **alpha** and **beta:**

```
db.runCommand({unlockAccounts:1,db:{"exclude":["alpha","beta"]})
```


© 2017 IBM Corporation

JSON – unlockAccounts – Unlock a Database/User Account

- **include**
  - Optional parameter specifies the databases to include.
  - To unlock all accounts in the databases named **delta** and **gamma**:

```
db.runCommand({unlockAccounts:1,db:{"include":["delta","gamma"]})
```

- **$regex**
  - Optional evaluation query operator selects values from a specified JSON document.
  - To unlock accounts for databases that begin with the character **a.** and end in **e:**

```
db.runCommand({unlockAccounts:1,db:{"$regex":"a.*e"})
```

- **user**
  - This optional parameter specifies the user accounts to unlock.
  - For example, to unlock the account of all users that are not named **alice**:

```
db.runCommand({unlockAccounts:1,user:{$ne:"alice"}});
```


© 2017 IBM Corporation

Manipulation of JSON & BSON Data Types with SQL

- **JSON and BSON data types, allowed in local and distributed queries, are Informix built-in data types accessible and manipulatible with SQL statements.**
- **By calling BSON value functions within SQL statements it is possible to retrieve specific key values from JSON or BSON data columns.**
- **It is possible to define indexes on key values within a JSON or BSON column.**

© 2017 IBM Corporation

High Availability for MongoDB and REST Clients

- **MongoDB and REST clients can be provided High Availability functionality via running a wire listener on each server in an Informix high-availability cluster.**

- **Provide high availability between the wire listener and the Informix database server:**
  - Connect the wire listener to the database server through the Connection Manager
  - Specify an **sqlhosts** file via the **url** parameter in the wire listener properties file.

© 2017 IBM Corporation

Wire Listener Configuration Enhancements

- **These new or updated parameters can be set in the wire listener properties file:**
  - **url** parameter can include JDBC environment variables.
  - **listener.hostName** parameter can specify the listener host name to control the network adapter or interface to which the wire listener connects.
  - **collection.informix.options** parameter specifies table options to automatically add shadow columns or enable auditing during JSON collection creation.
  - **command.listDatabases.sizeStrategy** parameter can specify a strategy for computing the database size when **listDatabases** is executed.
  - **fragment.count** parameter can specify the number of fragments to create for a collection.
  - **jdbc.afterNewConnectionCreation** parameter can specify SQL statements, such as **SET ENVIRONMENT**, to run after connecting to the database server.

© 2017 IBM Corporation

Wire Listener Query Support

- **The wire listener now supports these types of queries:**
  - Join queries on;
    - JSON data
    - Relational data or
    - Both JSON and relational data.
  - Array queries on JSON data with the **$elemMatch** query operator:
    - Ratings, for example, must be an arrayed column in the inventory collection.

      db.inventory.find({ ratings: { $elemMatch: { $gt: 25, $lt: 90 } } })
  - **$first** and **$last** group operators

© 2017 IBM Corporation

IBM

Wire Listener Query Support (1)

- **Join query support is an important part of the hybrid SQL/NoSQL value proposition of Informix.**
- **The JSON listener now supports the following running joins by querying against a new pseudo system.join table:**
  - Collection-to-collection
  - Relational-to-relational
  - Collection-to-relational
- **Join queries are done by running a "find" query against the new pseudo system table called system.join.**
- **For, example, in the Mongo shell, you'd run a query like this:**

> db.system.join.find({ *join query document* })

© 2017 IBM Corporation

Wire Listener Query Support (2)

- **Join Query Document:**

```
{ $collections :
      {
          "tabName1" :  { join_table_specification },
          "tabName2" :  { join_table_specification },
          …
      },
  "$condition" : { join_condition_specification }
}
```

- **Required:**
  - **$collections** and **$condition** fields to run a find query against **system.join**.
  - The **$collections** field must map to a document that includes two or more collections or relational tables to be joined.
  - The **$condition** specifies how to join the collections/tables. No other query operators are supported in the top level of the join query document. (over)

© 2017 IBM Corporation

IBM

Wire Listener Query Support (3)

- **The join_table_specification for each collection/table must include the required $project field; can have an optional $where query document:**

  {"**$project**" : { … }, "$where": { … }}

  - The **$project** field follows the same projection syntax as regular Mongo queries.
  - The optional **$where** field and uses the same query syntax as regular Mongo queries.

- **The join_condition_specification is a document of key-value pairs that define how all of the tables specified are joined together. These conditions can be specified in two ways:**
  - A key-string value pair to map a single table's column to another table's column:

    "tabName1.column1": "tabName2.column2"

  - As a key-array pair to map a table's column to multiple other table columns.

    "tabName1.column1":
    ["tabName2.column2", "tabName3.column3", ….]

© 2017 IBM Corporation

Wire Listener Query Support – Implementation Details

- **Join queries work:**
  - With the **sort, limit, skip, and explain** options that can be set on a Mongo cursor
  - With listener cursoring

- **Collection-to-Collection joins:**
  - The listener will look up if there are matching typed BSON indexes on the join fields for each collection.
    - If so, it will use that **bson_value_*** function in the join condition to take advantage of the index.

  - If the join was on **customer.customer_num** and **orders.customers_num** and there were **bson_value_int** indexes on both **customer.customer_num** and **orders.customer_num**, then the listener SQL join condition would be:
    - **bson_value_int(customer.data, “customer_num”) = bson_value_int(orders.data, “customer_num”)**

© 2017 IBM Corporation

Wire Listener Query Support – Implementation Details

- If there are no matching indexes using the same **bson_value_*** function, then the listener defaults to the **bson_get** function for the join condition:
  - **bson_get(customer.data, “customer_num”) = bson_get(orders.data, “customer_num”)**

▪ Collection-to-Relational joins:

- For collection-to-relational joins, the data type of the relational column determines the **bson_value_*** function that is used:
  - If joining a collection field to a **character relational** column, the **bson_value_lvarchar** function is used.
  - If joining a collection field to a **numeric relational** column, the **bson_value_double** function is used, etc.

▪ Relational-to-Relational joins:

- No type conversions in the SQL query itself are necessary.
- The SQL condition is as expected:
  - **tab1.col1 = tab2.col2**

© 2017 IBM Corporation

Wire Listener Query Support Examples (1)

- For all these examples, the tables can be collections, relational tables, or a combination of both; the syntax is the same.

- Example 1: Get the customers orders that totaled more than $100. Join the customers and orders collections/tables on the customer_num field/column where the order total > 100.

```
{ "$collections" :
      {
            "customers" :
{ "$project": { customer_num: 1, name: 1, phone: 1 } },
            "orders" :
{ "$project": { order_num: 1, nitems: 1, total: 1, _id: 0 },
  "$where" : { total : { "$gt": 100 } } }
      },
  "$condition" :
{ "customers.customer_num" : "orders.customer_num" }
}
```


© 2017 IBM Corporation

Wire Listener Query Support Examples (2)

- **Get the IBM locations in California and Oregon.**
- **Join the companies, sites, and zipcodes collections/tables where company name is "IBM" and state is "CA" or "OR".**

```
{ $collections :
      {
            "companies" :
{ "$project": { name: 1, _id: 0 }
 "$where" : { "name" : "IBM" } },
            "sites" :
{ "$project": { site_name: 1, size: 1, zipcode: 1, _id:0 } },
            "zipcodes" :
{ "$project": { state: 1, city: 1, _id:0 }
 "$where" : { "state": { "$in", ["CA", "OR"] } } }
      },
 "$condition" :
{ "companies._id" : "sites.company_id",
 "sites.zipcode" : "zipcodes.zipcode" }
}
```


© 2017 IBM Corporation

IBM

Wire Listener Query Support Examples (3)

- Use array syntax in the condition.
- Get the order info, shipment info, and payment info for order number 1093.

```
{ $collections :
      {
            "orders" :
{ "$project": { order_num: 1, nitems: 1, total: 1, _id: 0 },
 "$where" : { order_num : 1093 } },
            "shipments" :
{ "$project": { shipment_date: 1, arrival_date: 1 } },
            "payments" :
{ "$project": { payment_method: 1, payment_date: 1 } }
      },
 "$condition" :
{ "orders.order_num" :
[ "shipments.order_num", "payments.order_num" ] }
```


© 2017 IBM Corporation

IBM

BSON_UPDATE Enhancements

- The BSON_UPDATE SQL function allows SQL users to update the contents of a BSON column within a JSON or BSON document with one or more MongoDB update operators.

- Prior to 12.10.xC7, the BSON_UPDATE function allowed for the following Mongo_DB update operators to be used:
  - $set
  - $inc
  - $unset

- Now, the following are additionally allowed from the database server:

MongoDB array update operators	MongoDB array update operator modifiers
$addToSet	$each
$pop	$position
$pullAll	$slice
$push	$sort

© 2017 IBM Corporation

BSON_UPDATE Usage

- **BSON_UPDATE is a function to update a BSON document with the supported MongoDB API update operators.**

- **Update operations are run sequentially:**
  - 1) The original document is updated
  - 2) The updated document is then the input to the next update operator.
  - 3) The **BSON_UPDATE** function returns the final **BSON** document.
    - Which then needs to be converted to **JSON** if to be human read

- **To include JavaScript expressions in update operations, evaluate the expressions on the client side and supply the final result of the expression in a field-value pair.**

© 2017 IBM Corporation

Some Notes

- **The MongoDB array update operators $pull, $pushAll work within wire listener based operations and use a different strategy to perform document updates that do not involve the BSON_UPDATE command or the database server; JSON operations that do not use the database server.**

- **It would appear the MongoDB API update operator '$' is not supported currently, as you receive the following error message:**
  - "9659: The server does not support the specified **UPDATE** operation on **JSON** documents."

© 2017 IBM Corporation

Wire Listener Processing Change

- **The wire listener now sends document updates to the database server first by default.**
- **Previously, the wire listener processed document updates by default. The default setting of the update.mode parameter in the wire listener configuration file is now mixed instead of client.**
  - The wire listener falls back to the **client** in **mixed** mode when the database server cannot process the updates.

© 2017 IBM Corporation

Quickly Add or Remove Shard Servers With Consistent Hashing

- **Quickly add or remove a shard server by using the new consistent hashing distribution strategy to shard data.**

- **With consistent hash-based sharding, the data is automatically distributed between shard servers in a way that minimizes the data movement when you add or remove shard servers.**

- **The original hashing algorithm redistributes all the data when you add or remove a shard server.**

- **You can specify the consistent hashing strategy when you run the cdr define shardCollection command.**

© 2017 IBM Corporation

Consistent Hash-based Sharding

- **When a consistent hash-based sharding definition is created, Informix uses a hash value of a specific defined column or field to distribute data to servers of a shard cluster in a consistent pattern.**

- **If a shard server is added or removed, the consistent hashing algorithm redistributes a fraction of the data.**

- **Specify how many hashing partitions to create on each shard server:**
  – The default is **3**.

- **If more than the default number of hashing partitions are created, the more evenly the data is distributed among shard servers.**
  – If more than 10 hashing partitions are specified, the resulting SQL statement to create the sharded table might fail because it exceeds the SQL statement maximum character limit.

© 2017 IBM Corporation

cdr define shardCollection (1)

- Below is a sharding definition that is named collection_1. Rows that are inserted on any of the shard servers are distributed, based on a consistent hash algorithm, to the appropriate shard server.

- The b column in the customers table that is owned by user john is the shard key. Each shard server has three hashing partitions.

```
cdr define shardCollection collection_1 db_1:john.customers --type=delete
--key=b --strategy=chash --partitions=3 --versionCol=column_3
g_shard_server_1 g_shard_server_2 g_shard_server_3
```

- ER verifies a replicated row or document was not updated before the row or document can be deleted on the source server.

- Each shard server has a partition range calculated on the server group name and data distributed according to the following sharding definition which is very data dependent: (over)

© 2017 IBM Corporation

Consistent Hashing Index Example

- **To create three partitions on each shard server:**
  cdr define shardCollection collection_1 db_1:informix.customers --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3 g_shard_server_1 g_shard_server_2 g_shard_server_3

- **Change dynamically the number of hashing partitions per shard server by running the cdr change shardCollection command.**
  cdr change shardCollection collection1 - –partitions=4

© 2017 IBM Corporation

cdr define shardCollection (2)

```
g_shard_server_1 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 4019 and 5469) or
                 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5719 and 6123) or
                 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2113 and 2652)
g_shard_server_2 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 6124 and 7415) or
                 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5470 and 5718) or
                 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 7416 and 7873)
g_shard_server_3 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2653 and 3950) or
                  mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) >= 7874 or
                  mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) < 2113 or
                 (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 3951 and 40
```


© 2017 IBM Corporation

MongoDB 3.2,3.4,3.6,3.8,4.0,4.2 API Support

- **Informix is compatible with the MongoDB 3.2-4.2 API's versions.**

- C Driver Compatibility
  - MongoDB C Driver
- C++ Driver Compatibility
  - C++ version 11 0.3.0 &
- C#/.Net Driver Compatibility
  - Versions 2.12 & 2.11
- Java Driver Compatibility
  - Version 3.11 thru 4.2
- Node.js Driver Compatibility
  - Version 3.3. thru 3.6
- PHP Driver Compatibility
  - PHPLIB 1.0 + mongodb-1.1
  - ext 1.6 + lib 1.5 thru ext 1.9 + lib 1.8
- Python Driver Compatibility
  - Use PyMongo 3.9 thru 3.11
  - Compatible with Python 2.7, 3.4-3.7
- Ruby Driver Compatibility
  - MongoDB Ruby Driver Version 2.2
- Scala Driver Compatibility
  - Version 2.7-4.2

All versions should be checkout with Informix Support and documentation. Versioning shown above is based on Mongo API 4.2, the latest supported API for Informix. Not all MongoDB supported drivers are shown above.

© 2017 IBM Corporation

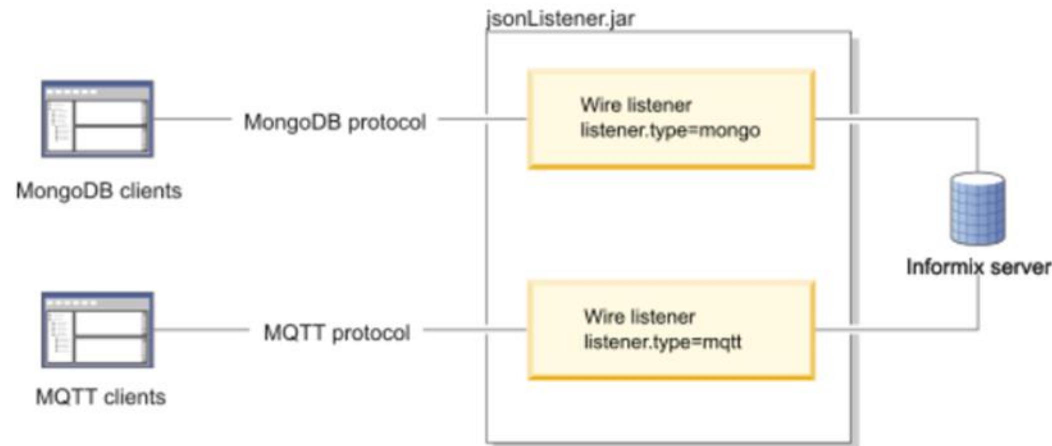
MQTT and Informix

© 2021 IBM Corporation

MQTT & JSON – Internet of Things (IoT) (1)

- **Load JSON documents with the MQTT protocol by defining a wire listener of type MQTT.**

- **The MQTT protocol is a light-weight messaging protocol that you can use to load data from devices or sensors:**
  - Use the MQTT protocol with Informix to publish data from sensors into a time series that contains a BSON column.



© 2017 IBM Corporation

MQTT & JSON – Internet of Things (IoT) (2)

- **Informix supports the following operations for MQTT:**
  - CONNECT
  - PUBLISH (equivalent to insert)
  - DISCONNECT

- **Configure an MQTT wire listener by setting the listener.type=mqtt parameter in the wire listener configuration file.**
  - From an MQTT client, you send PUBLISH packets to insert data.
  - You can authenticate MQTT client users through the wire listener.
  - Network Connections via TCP/IP or TLS and WebSocket; **not UDP** [1]
    - The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes from the Client to Server and Server to Client.
      - The transport protocol used to carry MQTT 3.1 & 3.1.1 is TCP/IP
      - TCP ports 8883 (TLS) and 1883 (non TLS) communications respectively are used

© 2017 IBM Corporation

MQTT & JSON – Connect & Publish

- **Connect (*database_name.user_name)***
  - Include a **Connect** packet to identify the client user.
  - If authentication is enabled in the MQTT wire listener with the **authentication.enable=true** setting, specify a user name and password.
    - User name includes the database name in the following format: ***database_name.user_name***.
    - Example: connect to the database **mydb** as user **joe** with the password **pass4joe**:
      - **CONNECT(mydb.joe, pass4joe)**

- **Publish *(topicName, { message }***
  - The **Publish** packet maps to the MongoDB **insert** or **create** command.
  - The ***topicName*** field must identify the target database and table in the following format: ***database_name/table_name***,
  - The ***message*** field must be in JSON format.
    - If you are inserting data into a relational table, the field names in the JSON documents must correspond to column names in the target table.
  - The following example inserts a JSON document into the sensordata table in the mydb database:
    - **PUBLISH(mydb/sensordata, { "id": "sensor1234", "reading": 87.5})**

© 2017 IBM Corporation

MQTT & JSON – Internet of Things (IoT) (4)

- **Prerequisites:**
  - Cannot presently create a time series through the MQTT wire listener
  - Create a JSON time series with the REST API, the MongoDB API, or SQL statements
  - TimeSeries row type consists of a time stamp and BSON columns
  - Example: In SQL, with row type, accompanying table, and storage container with record insert for the data:

```
CREATE ROW TYPE ts_data_j2( tstamp datetime year to fraction(5), reading
BSON);
CREATE TABLE IF NOT EXISTS tstable_j2( id INT NOT NULL PRIMARY KEY, ts
timeseries(ts_data_j2) ) LOCK MODE ROW;
EXECUTE PROCEDURE TSContainerCreate('container_j', 'dbspace1', 'ts_data_j2',
512, 512);
INSERT INTO tstable_j2 VALUES(1, 'origin(2014-01-01 00:00:00.00000),
calendar(ts_15min), container(container_j), regular, threshold(0), []');
```

  - Create a virtual table with a JSON time series

```
EXECUTE PROCEDURE TSCreateVirtualTab("sensordata", "tstable_j2");
```


© 2017 IBM Corporation

Questions

© 2021 IBM Corporation

Appendix A - MongoDB Shell Supported db.collection commands – released 12.10xC8

© 2021 IBM Corporation

Appendix A – MongoDB Shell Supported db.collection commands (1)

MongoDB Command	JSON Collection	Relational Tables	Details
aggregate	no	no	
count	yes	yes	
createIndex	yes	yes	You can use the ***MongoDB createIndex*** syntax to create an index that works for all data types. For example: ***db.collection.createIndex({ zipcode: 1 })*** ***db.collection.createIndex({ state: 1, zipcode: -1})*** You can use the Informix ***createIndex*** syntax to create an index for a specific data type. For example: ***db.collection.createIndex({ zipcode : [1, “$int”] })*** ***db.collection.createIndex({ state: [1, “$string”], zipcode: [-1, “$int”] })*** **Tip:** If you are creating an index on a field that has a fixed data type, you can get better query performance by using the Informix ***createIndex*** syntax.
dataSize	yes	no	
distinct	yes	yes	
drop	yes	yes	

© 2017 IBM Corporation

Appendix A – MongoDB Shell Supported db.collection commands (2)

MongoDB Command	JSON Collections	Relational Tables	Details
dropIndex	yes	yes	
dropIndexes	yes	no	
ensureIndex	yes	yes	You can use the ***MongoDB ensureIndex*** syntax to create an index that works for all data types. For example: ***db.collection.ensureIndex({ zipcode: 1 })*** ***db.collection.ensureIndex({ state: 1, zipcode: -1})*** You can use the Informix ***ensureIndex*** syntax to create an index for a specific data type. For example: ***db.collection.ensureIndex({ zipcode : [1, "$int"] })*** ***db.collection.ensureIndex({ state: [1, "$string"],zipcode: [-1, "$int"] })*** **Tip:** If you are creating an index on a field that has a fixed data type, better query performance can be had by using the Informix ***ensureIndex*** syntax.
find	yes	yes	
findAndModify	yes	yes	

© 2017 IBM Corporation

Appendix A – MongoDB Shell Supported db.collection commands (3)

MongoDB Command	JSON Collections	Relational Tables	Details
findOne	yes	yes	
getIndexes	yes	no	
getShardDistribution	no	no	
getShardVersion	no	no	
getIndexStats	no	no	This is deprecated in MongoDB 3.0
group	no	no	
indexStats	no	no	This is deprecated in MongoDB 3.0
insert	yes	yes	
isCapped	yes	yes	This command returns false because capped collections are not directly supported in Informix.
mapReduce	no	no	

© 2017 IBM Corporation

Appendix A – MongoDB Shell Supported db.collection commands (4)

MongoDB Command	JSON Collections	Relational Tables	Details
reIndex	no	no	
remove	yes	yes	The **justOne** option is not supported. This command deletes all documents that match the query criteria.
renameCollection	no	no	
save	yes	no	
stats	yes	no	
storageSize	yes	no	
totalSize	yes	no	
update	yes	yes	The multi option is supported only if ***update.one.enable=true*** is set in the **wire listener** file. Multi-parameter is ignore for relational table; all docs are updated meeting the query criteria. If ***update.one.enable=false***, all documents that match the query criteria are updated.
validate	no	no	

© 2017 IBM Corporation

IBM

Appendix B - MongoDB & Informix Command Support – released 12.10xC8

© 2021 IBM Corporation

Appendix B - MongoDB Operations and Informix

- **MongoDB read and write operations on existing relational tables are run as if the table were a collection.**

- **The wire listener determines whether the accessed entity is a relational table and converts the basic MongoDB operations on that table to SQL, and then converts the returned values back into a JSON document.**

- **The initial access to an entity results in an extra call to the Informix server:**
  - However, the wire listener caches the name and type of an entity so that subsequent operations do not require an extra call.

- **MongoDB operators are supported on both JSON collections and relational tables, unless explicitly stated otherwise.**

© 2017 IBM Corporation

IBM

Appendix B - MongoDB Command Support (1)

- **User Commands - Aggregation Commands**

MongoDB Command	JSON Collections	Relational tables	Details
aggregate	yes	yes	The wire listener supports version 2.4 of the MongoDB aggregate command, which returns a command result.
count	yes	yes	Count items in a collection
distinct	yes	yes	
group	no	no	
mapReduce	no	no	

- **User Commands - Geospatial Commands**

MongoDB Command	JSON Collections	Relational Tables	Details
geoNear	yes	no	Supported by using the GeoJSON format. The MongoDB legacy coordinate pairs are not supported.
geoSearch	no	no	
geoWalk	no	no	

© 2017 IBM Corporation

IBM

Appendix B - MongoDB Command Support (2)

- **User Commands - Query and Write Operation Commands**

MongoDB Command	JSON Collections	Relational Tables	Details
delete	yes	yes	
eval	no	no	
findAndModify	yes	yes	
getLastError	yes	yes	For relational tables, the findAndModify command is only supported for tables that have a primary key, a serial column, or a rowid.
getPrevError	no	no	
insert	yes	yes	
resetError	no	no	
text	no	no	Text queries are supported by using the $text or $ifxtext query operators, not through the text command
update	yes	yes	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (3)

- **Database Operations – Authentication Commands**

MongoDB Command	Supported	Details
authenticate	yes	
authSchemaUpgrade	yes	
getonce	yes	
logout	yes	

- **Database Operations – Diagnostic Commands**

MongoDB Command	Supported	Details
buildInfo	yes	Whenever possible, the Informix output fields are identical to MongoDB. There are additional fields that are unique to Informix.
collStats	yes	The value of any field that is based on the collection size is an estimate, not an exact value. For example, the value of the field '***size***' is an estimate.
connPoolStats	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (4)

▪ Database Operations – Diagnostic Commands (cont'd)

MongoDB Command	Supported	Details
cursorInfo	no	
dbStats	yes	The value of any field that is based on the collection size is an estimate, not an exact value. For example, the value of the field '***dataSize***' is an estimate.
features	yes	
getCmdLineOpts	yes	
getLog	no	
hostInfo	yes	The ***memSizeMB, totalMemory***, and ***freeMemory*** fields indicate the amount of memory available to the Java virtual machine (JVM) that is running, not the operating system values.
indexStats	no	Deprecated in MongoDB 3.0
listCommands	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (5)

- **Database Operations – Diagnostic Commands (cont'd)**

MongoDB Command	Supported	Details
listDatabases	yes	The value of any field that is based on the collection size is an estimate, not an exact value. For example, the value of the field '***sizeOnDisk***' is an estimate. The listDatabases command performs expensive and CPU-intensive computations on the size of each database in the Informix instance. You can decrease the expense by using the sizeStrategy option.
ping	yes	
serverStatus	yes	
top	no	
whatsmyurl	yes	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (6)

- **Database Operations – Instance Administration Commands**

MongoDB Command	JSON Collections	Relational Tables	Details
clone	no	no	
cloneCollection	no	no	
cloneCollectionAsCapped	no	no	
collMod	no	no	
compact	no	no	
convertToCapped	no	no	.
copydb	no	no	
create	yes	no	Informix does not support the following flags: • capped • autoIndexID • size • max
createIndexes	yes	yes	
drop	yes	yes	Informix does not lock the database to block concurrent activity.

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (7)

- **Database Operations – Instance Administration Commands (cont'd)**

MongoDB Command	JSON collections	Relational Tables	Details
dropDatabase	yes	yes	
dropIndexes	yes	no	The MongoDB ***deleteIndexes*** command is equivalent.
filemd5	yes	yes	
fsync	no	no	
getParameter	no	no	
listCollections	yes	yes	The **includeRelational** and **includeSystem** flags are supported to include or exclude relational or system tables in the results. Default is **includeRelational=true** and **includeSystem=false.**
listindexes	yes	yes	
logRotate	no	no	
reIndex	no	no	
renameCollection	no	no	.
repairDatabase	no	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (8)

- **Database Operations – Instance Administration Commands (cont'd)**

MongoDB Command	JSON collections	Relational Tables	Details
setParameter	no	no	
shutdown	yes	yes	The ***timeoutSecs*** flag is supported. In the Informix, the ***timeoutSecs*** flag determines the number of seconds that the wire listener waits for a busy client to stop working before forcibly terminating the session. The force flag is not supported.
touch	no	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (9)

- **Replication Commands Support**

Name	Supported
isMaster	yes
replSetFreeze	no
replSetGetStatus	no
replSetInitiate	no
replSetMaintenance	no
replSetReconfig	no
replSetStepDown	no
replSetSyncFrom	no
Resync	no

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (10)

▪ Sharding Commands

MongoDB Command	JSON Collections	Relational tables	Details
addShard	yes	yes	The ***MongoDB maxSize*** and ***name*** options are not supported. In addition to the MongoDB command syntax for adding a single shard server, you can use the Informix specific syntax to add multiple shard servers in one command by sending the list of shard servers as an array.
enableSharding	yes	yes	Action is not required for Informix.
flushRouterConfig	no	no	
isdbgrid	yes	yes	
listShards	yes	yes	Equivalent Informix command is ***cdr list server***.
movePrimary	no	no	.
removeShard	no	no	
shardCollection	yes	yes	The equivalent Informix command is ***cdr define shardCollection***. The MongoDB ***unique*** and ***numInitialChunks*** options are not supported.
shardingstate	no	no	
split	no	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (11)

- **Query Selectors - Array query operations**

MongoDB Command	JSON Collections	Relational Tables	Details
$elemMatch	yes	no	
$size	yes	no	Supported for simple queries only. The operator is supported only when it is the only condition in the query document.

- **Comparison query operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$all	yes	yes	Supported for primitive values and simple queries only. The operator is only supported when it is the only condition in the query document.
$gt	yes	yes	Greater than
$gte	yes	yes	Greater than or equal to
$in	yes	yes	
$lt	yes	yes	Less than
$lte	yes	yes	Less than or equal to
$ne	yes	yes	Not equal

© 2017 IBM Corporation

IBM

Appendix B - MongoDB Command Support (12)

- **Comparison query operators (cont'd)**

MongoDB Command	JSON Collections	Relational Tables	Details
$all	yes	yes	Supported for primitive values and simple queries only. The operator is only supported when it is the only condition in the query document. Not in operator or not exists
$gt	yes	yes	
$gte	yes	yes	
$in	yes	yes	
$lt	yes	yes	
$lte	yes	yes	
$ne	yes	yes	
$nin	yes	yes	
$query	yes	yes	Returns documents matching its argument

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (13)

▪ Element query operators

MongoDB Command	JSON Collections	Relational Tables	Details
$exists	yes	no	Boolean, yes or no (1 or 0)
$type	yes	no	

▪ Evaluation query operators

MongoDB Command	JSON Collections	Relational Tables	Details
$mod	yes	yes	
$regex	yes	yes	The only supported value for the **$options** flag is **i**, which specifies a case-insensitive search.
$text	yes	yes	The $text query operator support is based on MongoDB version 2.6. You can customize your text index and take advantage of additional text query options by creating a basic text search index with the createTextIndex command.
$where	no	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (14)

- **Geospatial query operators**
  - Geospatial queries are supported by using the GeoJSON format. The legacy coordinate pairs are not supported.

MongoDB Command	JSON Collections	Relational Tables	Details
$geoWithin	yes	no	
$geoIntersects	yes	no	
$near	yes	no	
$nearSphere	yes	no	

- **JavaScript query operators**
  - These are currently (as of 12.10.xC8) not supported.

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (15)

- **Logical Query Operators**

MongoDB Command	JSON Collections	Relational tables	Details
$and	yes	yes	
$or	yes	yes	
$not	yes	yes	
$nor	yes	yes	

- **Java Script query operators are not supported as of this time.**

- **Projection Operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$	no	no	
$elemMatch	yes	no	
$meta	yes	yes	
$slice	no	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (16)

- **Projection Operators - Comparison query operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$comment	no	no	
$explain	yes	yes	
$hint	yes	no	
$orderby	yes	yes	Not supported for sharded data.

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (17)

- **Array Update Operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$	no	no	
addToSet	yes	no	Supported for primitive values only. The operator is not supported on arrays and objects.
$pop	yes	no	
$pullAll	yes	no	Supported for primitive values only. The operator is not supported on arrays and objects.
$pull	yes	no	Supported for primitive values only. The operator is not supported on arrays and objects.
$pushAll	yes	no	
$push	yes	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (18)

- **Array Update Operators Modifiers**

MongoDB Command	JSON Collections	Relational Tables	Details
$each	yes	no	
$slice	yes	no	
$sort	yes	no	
$position	no	no	

- **Bitwise Update Operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$bit	yes	no	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (19)

- **Field Update Operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$inc	yes	yes	
$rename	yes	no	
$setOnInsert	yes	no	
$set	yes	yes	
$unset	yes	yes	

- **Isolation update operators**
  - The isolation update operators are not supported.

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (20)

- **Aggregation Framework Operators – Pipeline Operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$geoNear	yes	no	Supported by using the GeoJSON format. The MongoDB legacy coordinate pairs are not supported. Geospatial query operators are not supported for sharded data. You cannot use dot notation for the distanceField and includeLocs parameters.
$group	yes	yes	
$limit	yes	yes	
$match	yes	yes	
$out	no	no	
$project	partial	partial	You can use $project to include fields from the original document, for example { $project : { title : 1 , author : 1 }}. You cannot use $project to insert computed fields, rename fields, or create and populate fields that hold subdocuments. Projection operators are not supported.

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (21)

- **Aggregation Framework Operators – Pipeline Operators (cont'd)**

MongoDB Command	JSON Collections	Relational Tables	Details
$redact	no	no	
$skip	yes	yes	
$sort	yes	yes	Command supported only for collections that are not sharded.
$unwind	yes	no	

- **Expression Operators - $group operators**

MongoDB Command	JSON Collections	Relational Tables	Details
$addtoSet	yes	no	
$first	no	no	
$last	no	no	
$max	yes	yes	
$min	yes	yes	

© 2017 IBM Corporation

Appendix B - MongoDB Command Support (22)

- **Expression Operators - $group operators (cont'd)**

MongoDB Command	JSON Collections	Relational Tables	Details
$avg	yes	yes	
$push	yes	no	
$sum	yes	yes	

© 2017 IBM Corporation

IBM

Appendix B - MongoDB Command Support (23)

- **User management commands**

Name	Supported	Details
createUser	Yes	Supported for MongoDB API ver 2.6 or higher
dropAllUsersFromDatabase	Yes	Supported for MongoDB API ver 2.6 or higher
dropUser	Yes	Supported for MongoDB API ver 2.6 or higher
grantRolesToUser	Yes	Supported for MongoDB API ver 2.6 or higher
revokeRolesFromUser	Yes	Supported for MongoDB API ver 2.6 or higher
updateUser	Yes	Supported for MongoDB API ver 2.6 or higher
usersInfo	Yes	Supported for MongoDB API ver 2.6 or higher

© 2021 IBM Corporation

Appendix B - MongoDB Command Support (24)

- **Roles Management commands**

Name	Supported	Details
createRole	Yes	Supported for MongoDB API ver 2.6 or higher
dropAllRolesFromDatabase	Yes	Supported for MongoDB API ver 2.6 or higher
dropRole	Yes	Supported for MongoDB API ver 2.6 or higher
grantPrivilegesToRole	Yes	Supported for MongoDB API ver 2.6 or higher
grantRolesToRole	Yes	Supported for MongoDB API ver 2.6 or higher
invalidateUserCache	No	
rolesInfo	Yes	Supported for MongoDB API ver 2.6 or higher
revokePrivilegesFromRole	Yes	Supported for MongoDB API ver 2.6 or higher
revokeRolesFromRole	Yes	Supported for MongoDB API ver 2.6 or higher
updateRole	Yes	Supported for MongoDB API ver 2.6 or higher

© 2021 IBM Corporation

Appendix C – Extensions to MongoDB Supported – released 12.10.xC8

© 2021 IBM Corporation

Agenda

- **Query Operators**
  - $ifxtext
  - $like
  - $nativecursor
- **createTextIndex**
- **exportCollection**
- **importCollection**
- **killCursors**
- **lockAccounts**
- **runFunction**
- **runProcedure**
- **transaction**
- **unlockAccounts**

© 2017 IBM Corporation

Query operators - $ifxtext & $like

- **Query Operator: $ifxtext**
  - Can be used in all MongoDB functions that accept query operators
  - **$ifxtext** – uses Basic Text Search (BTS)
    - Passes the search string as-is to the **bts_contains()** function.
    - When using relational tables, the MongoDB **$text** and Informix **$ifxtext** query operators both require a column name, specified by **$key**, in addition to the **$search** string.
    - The **$search** string can be a word or a phrase as well as optional query term modifiers, operators, and stopwords.
    - Can include field names to search in specific fields.
    - Search string syntax in the **$ifxtext** query operator is the same as the **bts_contains()** search syntax criteria included in an SQL query.
      - **Example:**
        db.collection.find({ "$ifxtext" : { "$search" : "te?t” } })

- **Query Operator: $like**
  - The **$like** operator tests for matching character strings
  - Maps to the SQL **LIKE** query operator.
  - In the following example, a wildcard search is run for strings that contain Informix:
    db.collection.find({ "$like" : "%Informix%")

© 2017 IBM Corporation

Query modifier - $nativeCursor

- **Holds open a true cursor on the Informix database server during a query**
- **A native cursor requires more wire listener resources because connections and result set objects are tied to a single session, but the cursor guarantees consistent query results**
- **Control the cursor idle timeout with the cursor.idle.timeout wire listener property**
  - For REST API queries, use the **killCursors** command to close the cursor. In the following example, query results are returned in a cursor:

    ```
    db.collection.find().addSpecial("$nativeCursor",1);
    ```


© 2017 IBM Corporation

createTextIndex

- **Basic Text Search (bts) indexes are now possible with MongoDB, JSON and Informix.**
  - Widely used functionality in this environment.

- **createTextIndex –** creates Informix **bts** indexes within JSON.
  - Text Indexes created by using the Informix **createTextIndex** function must be queried by using the Informix **$ifxtext** query operator.
  - Text Indexes created by using the MongoDB syntax must be queried by using the MongoDB **$text** query operator.

```
db.runCommand( { createTextIndex: “mytab”, name:”myidx”,
key:{“title”:”text”, “abstract”:”text”}, options : {} } )
```

- **The following example creates an index named articlesIdx on the articles collection by using the bts parameter all_json_names="yes".**

```
db.runCommand( { createTextIndex: “articles”, name:”articlesIdx”, options
: {all_json_names : "yes"} } )
```


© 2017 IBM Corporation

exportCollection

- **Export JSON collections from the wire listener to a file.**

```
>>-exportCollection:"collection_name",--file:"filepath",-------->

                  .-json-.
>--format:-+-+-"-+--------+-"-+--+--------------------------------------+----+-->
           |  '-"jsonArray"--' |                  .-,------.         |    |
           |                   |                  V        |         |    |
           |                   '-,--fields--:--{--"----filter-+--"--}--'    |
           |                              .-,------.                      |
           |                              V        |                      |
           '-"csv",-fields--:--{--"----filter-+--"--}----------------------'


---+-------------------------------------------+-------------<>
   '-,--query--:--{--"--query_document--"--}-'
```


© 2017 IBM Corporation

exportCollection

- **exportCollection**
  - This required parameter specifies the collection name to export.
- **file**
  - Required parameter specifying host machine output file path where the wire listener is running. For example:
    - UNIX is file:"/tmp/export.out"
    - Windows is file:"C:/temp/export.out"
- **format**
  - This required parameter specifies the exported file format.
- **json**
  - Default. The **.json** file format. a JSON-serialized document per line is exported.
  - The following exports all collection documents named **c** by using the **json** format:

```
> db.runCommand({exportCollection:"c",file:"/tmp/export.out"
,format:"json"}) { "ok":1, "n":1000, "millis":NumberLong(119),
"rate":8403.361344537816 }
```

"n" is the number of documents that are exported,
"millis" is the number of milliseconds it took to export, and
"rate" is the number of documents per second that are exported.

© 2017 IBM Corporation

IBM

exportCollection

- **jsonArray**
  - The **.jsonArray** file format.
    - This format exports an array of JSON-serialized documents with no line breaks.
    - The array format is JSON-standard.
  - Following exports all documents from collection **c** using the **jsonArray** format:

```
> db.runCommand({exportCollection:"c",file:"/tmp/export.out" ,
format:"jsonArray"}) { "ok":1, "n":1000, "millis":NumberLong(81),
"rate":12345.67901234568 }
```

    Where "n" is the number of documents that are exported,
    "millis" is the number of milliseconds it took to export, and
    "rate" is the number of documents per second that are exported.

- **csv**
  - The **.csv** file format.
    - Comma-separated values are exported.
    - You must specify which fields to export from each document.
    - The first line of the **.csv** file contains the fields and all subsequent lines contain the comma-separated document values.

© 2017 IBM Corporation

exportCollection

- **fields**
  - This parameter specifies which fields are included in the output file.
    - Required for the **csv** format, but optional for the **json** and **jsonArray** formats.
  - Following command exports all documents from the collection that is named **c** by using the **csv** format, only output the **"_id"** and "**name**" fields:

```
> db.runCommand({exportCollection:"c",file:"/tmp/export.out"
,format:"csv",fields:{"_id":1 ,"name":"1"}}) { "ok":1, "n":1000,
"millis":NumberLong(57), "rate":17543.859649122805 }
```

  - Where "n" is the number of documents that are exported,
  - "millis" is the number of milliseconds it took to export, and
  - "rate" is the number of documents per second that are exported.
- **query**
  - Optional parameter specifies a query document that identifies which documents are exported. The following example exports all documents from the collection that is named **c** that have a "**qty**" field that is less than 100:

```
> db.runCommand({exportCollection:"c",file:"/tmp/export.out"
,format:"json",query:{"qty":{"$lt":100}}})
{"ok":1,"n":100,"millis":NumberLong(5),"rate":20000}
```


© 2017 IBM Corporation

importCollection

- **Import JSON collections from the wire listener to a file.**

- **>>-importCollection:"*collection_name*",--file:"*filepath*",-------->**

```
                    .-json---------.
>--format:"-+-jsonArray-+-"-----------------------------------><
                    '-csv-----------'
```

- **importCollection**
  – The required parameter specifies the collection name to import.
- **file**
  – This required parameter specifies the input file path.
  – Example,
  file: "/tmp/import.json".
  – Important: The input file must be on the same host machine where the wire listener is running.

© 2017 IBM Corporation

importCollection

- **format**
  - This required parameter specifies the imported file **format.**
  - **json**
    - Default. The **.json** file format.
    - Following example imports documents from the collection that is named **c** by using **json** format:

```
db.runCommand({importCollection:"c",file:"/tmp/import.out"
,format:"json"})
```

  - **jsonArray**
    - The **.jsonArray** file format.
    - The following example imports documents from the collection **c** by using **jsonArray**

```
db.runCommand({exportCollection:"c",file:"/tmp/import.out"
,format:"jsonArray"})
```

  - **csv**
    - The **.csv** file format.

© 2017 IBM Corporation

lockAccounts – Lock a Database/User Account

- If you specify the lockAccounts:1 command without specifying a db or user argument, all accounts in all databases are locked.
- Run this command as instance administrator.

- Syntax:

```
>>-lockAccounts:----1,-+------------------------------------------------------------------------------------+----------><
                       +-db:-+-"database_name"---------------------------------------------------+-“---+
                       |     |   .-,---------------------------.                                 |     |
                       |     |   V                             |                                 |     |
                       |     +-[---"database_name"-+-]-------------------------------------------+     |
                       |     +-{"$regex":"json_document"}----------------------------------------+     |
                       |     |  .-,---------------------------------------------------------------|   |     |
                       |     |  V                                                               |   |     |
                       |     '-{---+-"include":-+-"database_name"---------------+-+-+-}-'             |
                       |           |            |  .-,-------------------------.         | |         |
                       |           |            |  V                           |         | |         |
                       |           |            +-[---"database_name"-+-]---------+ |         |
                       |           |            '-{"$regex":"json_document"}---‘ |         |
                       |           '-"exclude":-+-"database_name"---------------+--'          |
                       |                        |   .-,---------------------------.      |            |
                       |                        |   V                             |      |            |
                       |                        +-[---"database_name"-+-]----+            |
                       |                        '- {"$regex":"json_document"}-'           |
                       '-user:-+-"user_name"------+--------------------------------------------------------'
                               '-"json_document"-'
```


© 2017 IBM Corporation

lockAccounts – Lock a Database/User Account

- **lockAccounts:1**
  - Required parameter locks a database or user account.

- **db**
  - Optional parameter specifies the database name of an account to lock.
  - For example, to lock all accounts in database that is named **foo**:

```
db.runCommand({lockAccounts:1,db:"foo"})
```

- **exclude**
  - Optional parameter specifies the databases to exclude.
  - For example, to lock all accounts on the system except those in the databases named **alpha** and **beta**:

```
db.runCommand({lockAccounts:1,db:{"exclude":["alpha","beta"]}})
```


© 2017 IBM Corporation

IBM

lockAccounts – Lock a Database/User Account

- **include**
  - Optional parameter specifies the databases to include.
  - To lock all accounts in the databases named **delta** and **gamma**:

```
db.runCommand({lockAccounts:1,db:{"include":["delta","gamma"]})
```

- **$regex**
  - Optional evaluation query operator selects values from a specified JSON document.
  - To lock accounts for databases that begin with the character **a.** and end in **e**:

```
db.runCommand({lockAccounts:1,db:{"$regex":"a.*e"})
```

- **user**
  - Optional parameter specifies the user accounts to lock.
  - For example, to lock the account of all users that are not named **alice**:

```
db.runCommand({lockAccounts:1,user:{$ne:"alice"}});
```


© 2017 IBM Corporation

killCursors

- **Close native cursors that were created with the &nativeCursor query modifier in a REST API query.**

```
                                     .-------------.
                                     V             |
>>-killCursors:1--cursorIds:[---cursorID-+-]------------------><
```

- **killCursors**
  - This required parameter closes native cursors.
- **cursorIds**
  - This required parameter lists the native cursor IDs to close.

- **The following command closes the cursor with the ID of 22:**

  GET /dbname/$cmd?query={killCursors:1, cursorsIds:[22]}

© 2017 IBM Corporation

runFunction

- **Run an SQL function through the wire listener. Command is equivalent to the SQL statement EXECUTE FUNCTION.**

```
>>-runFunction:"function_name"--------------------------------->

>--+-----------------------------------------+-------------------------><
   |                              .-,--------.  |
   |                              V          |  |
   '-,"arguments":[---argument-+-]---'
```

- **runFunction**
  - This required parameter specifies the SQL function name to run.
  - Example, a current function returns the current date and time:

    db.runCommand({runFunction:"current"}) {"returnValue": 2016-04-05 12:09:00, "ok":1}

© 2017 IBM Corporation

runFunction

- **arguments**
  - This parameter specifies an array of argument values to the function.
  - You must provide as many arguments as the function requires.
  - For example, an **add_values** function requires two arguments to add together:

```
db.runCommand({runFunction:"add_values", "arguments":[3,6]})
{"returnValue": 9, "ok":1}
```

  - The following example returns multiple values from a **func_return3** function:

```
db.runCommand({runFunction:"func_return3", "arguments":[101]})
{"returnValue": {"serial_num":1103, "name":"Newton", "points":100},
"ok":1}
```


© 2017 IBM Corporation

runProcedure

- **Run an SQL stored procedure through the wire listener.**
  - This command is equivalent to the SQL statement **EXECUTE PROCEDURE**.

```
>>-runProcedure:"procedure_name"------------------------------->

>--+---------------------------------------+--------------------------><
   |                        .-,-------------.     |
   |                        V              |     |
   '-,"arguments":[---argument-+-]-'
```

- **runProcedure**
  - Required parameter specifies the name of the SQL procedure to run.
  - For example, a **colors_list** stored procedure, which uses a **WITH RESUME** clause in its **RETURN** statement, returns multiple rows about colors:

```
> db.runCommand({runProcedure:"colors_list"}) {"returnValue": [
{"color" : "Red","hex" : "FF0000"}, {"color" : "Blue","hex" : "0000A0"},
{"color" :"White","hex" : "FFFFFF"} ], "ok" : 1}
```


© 2017 IBM Corporation

runProcedure

- **arguments**
  - This parameter specifies an array of argument values to the procedure.
  - You must provide as many arguments as the procedure requires.
  - For example, an increase_price procedure requires two arguments to identify the original price and the amount of increase:

    ```
    db.runCommand({runProcedure:"increase_price", "arguments":[101, 10]}) {"ok":1}
    ```


© 2017 IBM Corporation

Extension: MongoDB Authentication Configuration

- **Procedures in order of operation:**
  - Start the MongoDB wire listener with authentication turned off.
  - For each database, add the users that you want grant access.
    - For example, to grant user **bob readWrite** access:
      db.addUser({user:"bob", pwd: "myPass1", roles:["readWrite","sql"]})
  - Stop the wire listener.
  - Set **authentication.enable=true** in the properties file.
  - Restart the wire listener.

- **Based on MongoDB 2.4**

- **After the wire listener is turned back on, each client will have to authenticate.**

© 2017 IBM Corporation

lockAccounts – Lock a Database/User Account

- If you specify the lockAccounts:1 command without specifying a db or user argument, all accounts in all databases are locked.
- Run this command as instance administrator.

- Syntax:

```
>>-lockAccounts:----1,-+-----------------------------------------------------------------------------------+----------><
                       +-db:-+-"database_name"---------------------------------------------------+-“---+
                       |     |   .-,--------------------------.                                  |     |
                       |     |   V                            |                                  |     |
                       |     +-[---"database_name"-+-]-------------------------------------------+     |
                       |     +-{"$regex":"json_document"}----------------------------------------+     |
                       |     |  .-,-----------------------------------------------------------|   |     |
                       |     |  V                                                           |   |     |
                       |     '-{---+-"include":-+-"database_name"---------------+-+-+-}-'          |
                       |           |            |  .-,-------------------------.        | |         |
                       |           |            |  V                           |        |  |        |
                       |           |            +-[---"database_name"-+-]---------+ |              |
                       |           |             '-{"$regex":"json_document"}---‘ |                |
                       |           '-"exclude":-+-"database_name"----------------+--'              |
                       |                        |  .-,--------------------------.     |            |
                       |                        |  V                            |     |            |
                       |                        +-[---"database_name"-+-]----+                     |
                       |                         '- {"$regex":"json_document"}-'                   |
                       '-user:-+-"user_name"------+-------------------------------------------------------'
                               '-"json_document"-'
```


© 2017 IBM Corporation

lockAccounts – Lock a Database/User Account

- **lockAccounts:1**
  - Required parameter locks a database or user account.

- **db**
  - Optional parameter specifies the database name of an account to lock.
  - For example, to lock all accounts in database that is named **foo**:

```
db.runCommand({lockAccounts:1,db:"foo"})
```

- **exclude**
  - Optional parameter specifies the databases to exclude.
  - For example, to lock all accounts on the system except those in the databases named **alpha** and **beta**:

```
db.runCommand({lockAccounts:1,db:{"exclude":["alpha","beta"]})
```


© 2017 IBM Corporation

lockAccounts – Lock a Database/User Account

- **include**
  - Optional parameter specifies the databases to include.
  - To lock all accounts in the databases named **delta** and **gamma**:

```
db.runCommand({lockAccounts:1,db:{"include":["delta","gamma"]})
```

- **$regex**
  - Optional evaluation query operator selects values from a specified JSON document.
  - To lock accounts for databases that begin with the character **a.** and end in **e**:

```
db.runCommand({lockAccounts:1,db:{"$regex":"a.*e"})
```

- **user**
  - Optional parameter specifies the user accounts to lock.
  - For example, to lock the account of all users that are not named **alice**:

```
db.runCommand({lockAccounts:1,user:{$ne:"alice"}});
```


© 2017 IBM Corporation

Extension: Transactions

- **These do not exist in MongoDB natively; optionally, these are Informix extensions that enable or disable transactions for a session.**
  - Binds/unbinds a MongoDB session in database

    enable
    - Enables transactions for the current session in the current db

      db.runCommand ({transaction : "enable" })

    disable
    - Disables transactions for the current session in the current db

      db.runCommand ({transaction : "disable" })

    status
    - Returns status of transaction enablement of current session and whether the current db supports transaction processing.

      db.runCommand ({transaction : "status" })

    commit
    - Commits the transaction for the current session in the current db if transactions are enabled; otherwise an error is produced if disabled.

      db.runCommand ({transaction : "commit" })

    rollback
    - Rolls back a transaction for the current session in the current db if transactions are enabled; otherwise an error is produced if disabled.

      db.runCommand ({transaction : "rollback" })

© 2017 IBM Corporation

Extension: Transactions (cont'd)

- **execute**
  - This optional parameter runs a batch of commands as a single transaction.
  - If transaction mode is not enabled for the session, this parameter enables transaction mode for the duration of the transaction.
  - Command documents include **insert**, **update**, **delete**, **findAndModify**, and **find**
    - **insert**, **update**, and **delete** command documents, you cannot set the **ordered** property to **false**.
    - Use a **find** command document to run SQL & NoSQL queries, but not commands.
    - A **find** command document can include the **$orderby, limit, skip**, and **sort** operators.
    - The following example deletes a document from the inventory collection and inserts documents into the archive collection:

```
db.runCommand({"transaction" : "execute", "commands" : [ {"delete":"inventory",
"deletes" : [ { "q" : { "_id" : 432432 } } ] }, {"insert" : "archive", "documents" : [ {
"_id": 432432, "name" : "apollo", "last_status" : 9} ] } ] })
```

  - The **execute** parameter has an optional **finally** argument if you have a set of command documents to run at the end of the transaction regardless of whether the transaction is successful.

© 2017 IBM Corporation

Extension: Transactions (cont'd)

- **Example:**

```
db.runCommand({"transaction" : "execute", "commands" : [ {"find" : "system.sql", "filter" : {"$sql" : "SET ENVIRONMENT USE_DWA 'ACCELERATE ON'" } }, {"find" : "system.sql", "filter" : {"$sql" : "SELECT SUM(s.amount) as sum FROM sales AS s WHERE s.prid = 100 GROUP BY s.zip" } } ], "finally" : [{"find":"system.sql", "filter" : {"$sql" : "SET ENVIRONMENT USE_DWA 'ACCELERATE OFF'" } } ] })
```

- **All transaction commands for Informix work on non-sharded servers only.**

© 2017 IBM Corporation

unlockAccounts – Unlock a Database/User Account

- If you specify the unlockAccounts:1 without specifying a db or user argument, all accounts in all databases are unlocked.
- Run this command as instance administrator.

```
>>-unlockAccounts:------1,-+-------------------------------------------------------------------------+----><
                           +-db:-+-"database_name"-------------------------------------------+-"-+
                           |     |  .-,-------------------------.                            |   |
                           |     |  V                           |                            |   |
                           |     +-[---"database_name"-+-]-----------------------------------+   |
                           |     +-{"$regex":"json_document"}--------------------------------+   |
                           |     |  .-,------------------------------------------------------.  |   |
                           |     |  V                                                      |  |   |
                           |     '-{---+-"include":-+-"database_name"-----------+-+-+-}-'    |
                           |           |          |  .-,-------------------------.     |  |       |
                           |           |          |  V                           |     |  |       |
                           |           |          +-[---"database_name"-+-]-----   -+  |       |
                           |           |          '-{"$regex":"json_document"}----' |       |
                           |           '-"exclude":-+-"database_name"-----------+-'         |
                           |                        |  .-,------------------------.     |         |
                           |                        |  V                          |     |         |
                           |                        +-[---"database_name"-+-]---+         |
                           |                        '-{"$regex":"json_document"}-'        |
                           '-user:-+-"user_name"-----+------------------------------------------'
                                   '-"json_document"-'
```


© 2017 IBM Corporation

unlockAccounts – Unlock a Database/User Account

- **unlockaccounts:1**
  - Required parameter unlocks a database or user account.

- **db**
  - Optional parameter specifies the database name of an account to unlock.
  - To unlock all accounts in database that is named **foo:**

```
db.runCommand({unlockAccounts:1,db:"foo"})
```

- **exclude**
  - Optional parameter specifies the databases to exclude.
  - To unlock all accounts on the system except those in the databases named **alpha** and **beta:**

```
db.runCommand({unlockAccounts:1,db:{"exclude":["alpha","beta"]})
```


© 2017 IBM Corporation

unlockAccounts – Unlock a Database/User Account

- **include**
  - Optional parameter specifies the databases to include.
  - To unlock all accounts in the databases named **delta** and **gamma**:

```
db.runCommand({unlockAccounts:1,db:{"include":["delta","gamma"]})
```

- **$regex**
  - Optional evaluation query operator selects values from a specified JSON document.
  - To unlock accounts for databases that begin with the character **a.** and end in **e:**

```
db.runCommand({unlockAccounts:1,db:{"$regex":"a.*e"})
```

- **user**
  - This optional parameter specifies the user accounts to unlock.
  - For example, to unlock the account of all users that are not named **alice**:

```
db.runCommand({unlockAccounts:1,user:{$ne:"alice"}});
```


© 2017 IBM Corporation

Appendix C - MongoDB API's Supported – Released 12.10.xC8

© 2021 IBM Corporation

Appendix C – MongoDB API's Supported

- **MongoDB community drivers can be used to store, update, and query JSON Documents with Informix as a JSON data store:**
  - Java,
  - C/C++,
  - Ruby,
  - PHP,
  - PyMongo

- **MongoDB API's to version 4.2 are supported by Informix.**

© 2017 IBM Corporation

Appendix D – MongoDB Supported Command Utilities & Tools - 12.10.xC8

© 2021 IBM Corporation

Appendix D – MongoDB Supported Command Utilities & Tools

- **You can use the MongoDB shell and any of the standard MongoDB command utilities and tools.**

- **Supported MongoDB shell versions: 2.4, 2.6, 3.0, 3.2, 3.4, 3.6, 3.8, 4.0, 4.2.**

- **You can run the MongoDB *mongodump* and *mongoexport* utilities against MongoDB to export data from MongoDB to Informix.**

- **You can run the MongoDB *mongorestore* and *mongoimport* utilities against Informix to import data to MongoDB from Informix.**

© 2017 IBM Corporation

Similar Technical Naming – Informix & Mongo DB (1)

MongoDB Concept	Informix Concept	Description
collection	table	This is the same concept. In Informix this type of collection is sometimes referred to as a JSON collection. A JSON collection is similar to a relational database table, except it does not enforce a schema.
document	record	This is the same concept. In Informix, this type of document is sometimes referred to as a JSON document.
field	column	This is the same concept.

© 2017 IBM Corporation

IBM

Similar Technical Naming – Informix & Mongo DB (2)

MongoDB Concept	Informix Concept	Description
master / slave	primary server /secondary server	This is the same concept. However, an Informix secondary server has additional capabilities. For example, data on a secondary server can be updated and propagated to primary servers.
replica set	high-availability cluster	This is the same concept. However, when the replica set is updated, it then sent to all servers, not only the primary server.
sharded cluster	sharded cluster	This is the same concept. In Informix, a sharded cluster consists of servers that are sometimes referred to as shard servers.
sharded key	shard key	This is the same concept.

© 2017 IBM Corporation

Appendix E – Some New Event Alarms

© 2021 IBM Corporation

IBM

Appendix E – New Event Alarms

Class Id	Internal Name	Event ID	Description
24	**ALRMU_85_AUTOTUNE_DAEMON_FAIL**	24013	Auto Tune daemon has insufficient resources to perform tuning operations.
Online log: Performance Advisory			
User action: Restart the server with more resources			

Class Id	Internal Name	Event ID	Description
24	**ALRMU_85_AUTOTUNE_ADD_CPUVP_FAIL**	24014	Auto tuning failed to start a CPU VP
Online log: Performance Advisory			
User action: Restart the server with more resources			

© 2017 IBM Corporation

Appendix E – New Event Alarms (cont'd)

Class Id	Internal Name	Event ID	Description
24	**ALRMU_24_ADD_BP_FAIL**	24015	Auto tuning was unable to extend the buffer pool due to insufficient resources.

Online log: Performance Advisory

User action: Restart the server with more resources

Class Id	Internal Name	Event ID	Description
85	**ALRMU_85_OVER_LLOG**	85001	Auto tuning failed to add another logical log, because adding another log would exceed the maximum log space as defined by configuration parameter AUTO_LLOG.

Online log: Performance Advisory

User action: Increase the maximum amount of log space by changing AUTO_LLOG configuration parameter.

© 2017 IBM Corporation

Appendix E – New Event Alarms (cont'd)

Class Id	Internal Name	Event ID	Description
85	**ALRMU_85_OVER_BPOOL**	85002	Auto tuning failed to extend a bufferpool, because the buffer pool would exceed the maximum amount of memory or extensions as defined by configuration parameter BUFFERPOOL.

Online log: Performance Advisory

User action: Increase the maximum amount defined by the configuration parameter BUFFERPOOL.

© 2017 IBM Corporation

Appendix E – New Event Alarms (cont'd)

Class Id	Internal Name	Event ID	Description
85	**ALRMU_85_OVER_CPU**	85003	Auto tuning failed to add a CPU VP because another CPU VP would exceed the maximum number specified in the configuration parameter VPCLASS or exceed the number of processors on the computer.

Online log: Performance Advisory

User action: If there are more processors available on the computer, increase the maximum number of CPU VPs allowed by changing the VPCLASS configuration parameter.

© 2017 IBM Corporation

Appendix F – Informix Product Limits

© 2021 IBM Corporation

Agenda

- **Unix Limits**
  - System-Level Parameter Limits
  - Table-level parameter limits
  - Access capabilities
  - Informix System Defaults

- **Windows Limits**
  - System-Level Parameter Limits
  - Table-level parameter limits
  - Access capabilities
  - Informix System Defaults

© 2017 IBM Corporation

Informix – Unix/Linux O/S Limits (1)

System-Level Parameters	Maximum Capacity per Computer System
IBM® Informix® systems per computer (Dependent on available system resources)	255
Maximum number of accessible remote sites	Machine specific
Maximum virtual shared memory segment (SHMVIRTSIZE)	2GB (32-bit platforms) or 4TB (64-bit platforms)
Maximum number of Informix shared memory segments	1024
Maximum address space	Machine specific

© 2017 IBM Corporation

Informix – Unix/Linux O/S Limits (2)

Table-Level Parameters (based on 2K page size)	Maximum Capacity per Table
Data rows per page	255
Data rows per fragment	4,277,659,295
Data pages per fragment	16,775,134
Data bytes per fragment (excludes Smart Large Objects (BLOB, CLOB) and Simple Large Objects (BYTE, TEXT) created in blobspaces)	2K page size = 33,818,670,144 4K page size = 68,174,144,576 8K page size = 136,885,093,440 12K page size = 205,596,042,304 16K page size = 274,306,991,168
Binary Large Object BLOB/CLOB pages	4 TB
Binary Large Objects TEXT/BYTE bytes	4 TB
Row length	32,767
Number of columns	32K
Maximum number of pages per index fragment	2,147,483,647
Key parts per index	16
Columns per functional index	102 (for C UDRs) 341 (for SPL or Java™ UDRs)

© 2017 IBM Corporation

Informix – Unix/Linux O/S Limits (3)

Table-Level Parameters (based on 2K page size)	Maximum Capacity per Table
Maximum bytes per index key (for a given page size):	2K page size = 387 4K page size = 796 8K page size = 1615 12K page size = 2435 16K page size = 3254
Maximum size of an SQL statement	Limited only by available memory

© 2017 IBM Corporation

IBM

Informix – Unix/Linux O/S Limits (4)

Access Capabilities	Maximum Capacity per System
Maximum databases per Informix® system	21 million
Maximum tables per Informix system	477, 102, 080
Maximum active users per Informix (minus the minimum number of system threads)	32K user threads
Maximum active users per database and table (also limited by the number of available locks, a tunable parameter)	32K user threads
Maximum number of open databases in a session	32 databases
Maximum number of open tables per Informix system	Dynamic allocation
Maximum number of open tables per user and join	Dynamic allocation
Maximum number of open transactions per instance	32,767
Maximum locks per Informix system and database	Dynamic allocation
Maximum number of page cleaners	128
Maximum number of partitions per dbspace	4K page size: 1048445, 2K page size: 1048314 (based on 4-bit bitmaps)
Maximum number of recursive synonym mappings	16

© 2017 IBM Corporation

Informix – Unix/Linux O/S Limits (5)

Access Capabilities	Maximum Capacity per System
Maximum number of tables locked with LOCK TABLE per user	32
Maximum number of cursors per user	Machine specific
Maximum Enterprise Replication transaction size	4 TB
Maximum dbspace size	131 PB
Maximum sbspace size	131 PB
Maximum chunk size	4 TB
Maximum number of chunks	32,766
Maximum number of 2K pages per chunk	2 billion
Maximum number of open Simple Large Objects (applies only to TEXT and BYTE data types)	20
Maximum number of B-tree levels	20
Maximum amount of decision support memory	Machine specific
Utility support for large files	17 billion GB
Maximum number of storage spaces (dbspaces, blobspaces, sbspaces, or extspaces)	2047

© 2017 IBM Corporation

Informix – Unix O/S Limits (6)

Database characteristic	Informix system default
Table lock mode	Page
Initial extent size	8 pages
Next extent size	8 pages
Read-only isolation level (with database transactions)	Committed Read
Read-only isolation level (ANSI-compliant database)	Repeatable Read
Database characteristic	Informix system default
Table lock mode	Page

© 2017 IBM Corporation

Informix – Windows O/S Limits (1)

System-Level Parameters	Maximum Capacity per Computer System
IBM® Informix® systems per computer (Dependent on available system resources)	255
Maximum number of accessible remote sites	Machine specific
Maximum virtual shared memory segment (SHMVIRTSIZE)	2 GB (32-bit platforms) or 4 TB (64-bit platforms)
Maximum number of Informix shared memory segments	1024
Maximum address space	2.7 GB if 4-gigabyte tuning is enabled: • All Windows versions later than Windows 2003 • Windows 2003 and earlier versions if the boot.ini file contains the /3GB switch 1.7 GB for Windows 2003 and earlier versions if the boot.ini file does not contain the /3GB switch

© 2017 IBM Corporation

Informix – Windows O/S Limits (2)

Table-Level Parameters (based on 2K page size)	Maximum Capacity per Table
Data rows per page	255
Data rows per fragment	4,277,659,295
Data pages per fragment	16,775,134
Data bytes per fragment (excludes Smart Large Objects (BLOB, CLOB) and Simple Large Objects (BYTE, TEXT) created in blobspaces)	2K page size = 33,818,670,144 4K page size = 68,174,144,576 8K page size = 136,885,093,440 12K page size = 205,596,042,304 16K page size = 274,306,991,168
Binary Large Object BLOB/CLOB pages	4 TB
Binary Large Objects TEXT/BYTE bytes	4 TB
Row length	32,767
Number of columns	32K
Maximum number of pages per index fragment	2,147,483,647
Key parts per index	16
Columns per functional index	102 (for C UDRs) 341 (for SPL or Java™ UDRs)

© 2017 IBM Corporation

Informix – Windows O/S Limits (3)

Table-Level Parameters (based on 2K page size)	Maximum Capacity per Table
Maximum bytes per index key (for a given page size):	2K page size = 387 4K page size = 796 8K page size = 1615 12K page size = 2435 16K page size = 3254
Maximum size of an SQL statement	Limited only by available memory

© 2017 IBM Corporation

Informix – Windows O/S Limits (4)

Access Capabilities (1)	Maximum Capacity per System
Maximum databases per IBM® Informix® system	21 million
Maximum tables per IBM Informix system	477, 102, 080
Maximum active users per IBM Informix (minus the minimum number of system threads)	32K user threads
Maximum active users per database and table (also limited by the number of available locks, a tunable parameter)	32K user threads
Maximum number of open databases in a session	8 databases
Maximum number of open tables per IBM Informix system	Dynamic allocation
Maximum number of open tables per user and join	Dynamic allocation
Maximum locks per IBM Informix system and database	Dynamic allocation
Maximum number of page cleaners	128
Maximum number of recursive synonym mappings	16
Maximum number of tables locked with LOCK TABLE per user	32
Maximum number of cursors per user	Machine specific

© 2017 IBM Corporation

IBM

Informix – Windows O/S Limits (5)

Access Capabilities (2)	Maximum Capacity per System
Maximum Enterprise Replication transaction size	4 TB
Maximum dbspace size	131 PB
Maximum sbspace size	131 PB
Maximum chunk size	4 TB
Maximum number of chunks	32 766
Maximum number of 2K pages per chunk	2 billion
Maximum number of open Simple Large Objects (applies only to TEXT and BYTE data types)	20
Maximum number of B-tree levels	20
Maximum amount of decision support memory	Machine specific
Utility support for large files	17 billion GB
Maximum number of storage spaces (dbspaces, blobspaces, sbspaces, or extspaces)	2047
Maximum number of partitions per dbspace	4K page size: 1048445, 2K page size: 1048314 (based on 4-bit bitmaps)

© 2017 IBM Corporation

Informix – Windows Limits

Database characteristic	Informix system default
Table lock mode	Page
Initial extent size	8 pages
Next extent size	8 pages
Read-only isolation level (with database transactions)	Committed Read
Read-only isolation level (ANSI-compliant database)	Repeatable Read

© 2017 IBM Corporation

Appendix G – MongoDB Recommended Security Steps and Informix similarities

© 2021 IBM Corporation

MongoDB Steps for Security

- Enable Access Control and Enforce Authentication
- Configure Role-Based Access Control
- Encrypt Communication
- Encrypt and Protect Data
- Limit Network Exposure
- Audit System Activity
- Run MongoDB with a Dedicated User
- Run MongoDB with Secure Configuration Options
- Request a Security Technical Implementation Guide (where applicable)

© 2017 IBM Corporation

Enable Access Control and Enforce Authentication (1)

- **In the Informix Wire Listener or thru PAM:**
  - You can authenticate users through the wire listener with MongoDB authentication or with the database server, through a pluggable authentication module (PAM) (see speaker notes).

- **For Informix Wire Listener and MongoDB connections:**
  - Set the following parameters in the wire listener configuration file:
    - Enable authentication:
      - Set **authentication.enable=true.**
    - Specify MongoDB authentication:
      - Set **db.authentication=mongodb-cr.**
    - Specify the MongoDB connection pool:
      - Set **database.connection.strategy=mongodb-cr**.
    - Set the MongoDB version:
      - Set mongo.api.version to the version that you want.
    - Optional. Specify the authentication timeout period:
      - Set the **listener.authentication.timeout** parameter to the number of milliseconds for authentication timeout.
  - Restart the wire listener (over)

© 2017 IBM Corporation

Enable Access Control and Enforce Authentication (2)

- **For Informix Wire Listener and MongoDB connections** (cont'd):
  - If necessary, upgrade your user schema by running the **authSchemaUpgrade** command in the **admin** database. For example:

    use admin

    db.runCommand({authSchemUpgrade : 1})

    - The **authSchemaUpgrade** command upgrades the user schema to the MongoDB version that is specified by the **mongo.api.version** parameter in the wire listener configuration file.

© 2017 IBM Corporation

Configure Role-Based Access Control

- **Create a user administrator first, then create additional users. Create a unique MongoDB user for each person and application that accesses the system.**

- **Create roles that define the exact access a set of users needs. Follow a principle of least privilege. Then create users and assign them only the roles they need to perform their operations. A user can be a person or a client application.**

- **In Informix this can be done as well in SQL.**
  - User informix as dba or a dba given such authority can delegate roles and subroles as necessary with scope limited to the current database.

    CREATE ROLE CustomerAdmin;
    - Creates a role for a Customer Admin

    GRANT ALL ON customer to CustomerAdmin;
    - Grants all authorities on the customer table to CustomerAdmin

    GRANT CustomerAdmin to john;

    SET ROLE CustomerAdmin;
    - Activates the role CustomerAdmin for all assigned users.

© 2017 IBM Corporation

IBM

Encrypt Communications (1 of 2)

- **Database server must be configured for SSL first as a prerequisite; see speaker notes at slide bottom.**
- **The connection between the wire listener and the <u>database server</u> can be encrypted using the wire listener configuration file and SSL.**
  - Use the **keytool** utility that comes with your JRE to import a client-side **keystore** database and add the public key certificate to the keystore:

    C:\work>keytool -importcert -file *server_keystore_file* -keystore *client_keystore_name*

    The ***server_keystore_file*** is the name of the server key certificate file.
  - Edit the wire listener properties file to update the **url** property to use the SSL port that you configured for the database server and add the **SSLCONNECTION=true** property to the end of the URL.
  - Start the listener with the **javax.net.ssl.trustStore** and **javax.net.ssl.trustStorePassword** system properties set:

    java -Djavax.net.ssl.trustStore="*client_keystore_path*" -Djavax.net.ssl.trustStorePassword="*password*" -jar jsonListener.jar -config jsonListener.properties -logfile jsonListener.log -start

    The ***client_keystore_path*** is the full path and file name of the client **keystore** file. The ***password*** is the **keystore** password.

© 2017 IBM Corporation

IBM

Encrypt Communications (2 of 2)

- **Encrypt connections between wire listener and client apps with SSL**
  - All client apps must use the same public key certificate file as the wire listener.
- **Create a keystore and certificate for the wire listener**
  - Use a method that best fits the client application and programming language.
    - E.g. you can use IBM® Global Security Kit (GSKit), OpenSSL, or Java keytool tool.

- **Edit the wire listener properties file and restart the listener. Set the following SSL parameters:**
  - **listener.ssl.enable** to **true** to enable SSL.
  - **listener.ssl.keyStore.file** to the full path of the **keystore** file.
  - **listener.ssl.keyStore.password** to the password to unlock the **keystore** file.
  - **listener.ssl.key.alias** to the alias or identifier of the **keystore** entry.
    - If the **keystore** contains only one entry, this parameter does not need to be set.
  - **listener.ssl.key.password** to the password to unlock the entry from the **keystore**
    - If this is not set, the listener uses the **listener.ssl.keyStore.password** parameter.
  - **listener.ssl.keyStore.type** if the keystore is not of type JKS (Java keystore).

- **Configure client applications to connect to the listener over SSL.**

© 2017 IBM Corporation

Encrypt and Protect Data (1 of 2)

- **Informix as of 12.10.xC8 fully supports Encryption at Rest in Informix and it is done at the time of initialization or by a full restore with the encrypt option turned on for ontape or onbar for existing instances.**
  - aes128, aes192, aes256

- **Setting the DISK ENCRYPTION configuration parameter in the Informix configuration file and restarting the server is all you have to do to enable it for existing instances and:**
  - Restoring the database via **ontape** or **onbar** with the **encrypt** option on is all you have to do to encrypt the storage spaces.

- **For new instances, setting the DISK ENCRYPTION configuration parameter in the Informix configuration file. All storage spaces created via onspaces will be encrypted unless the –u option of onspaces is executed.**

© 2017 IBM Corporation

Encrypt and Protect Data (2 of 2)

- **MongoDB introduced Encryption at Rest with its implementation of the Wired Tiger Storage Engine in version 3.2 of MongoDB.**
  - Informix is presently certified to version 4.2 of MongoDB 12.10.xC8 Encryption is compatible with the Wire Listener.
  - MongoDB API's are certified to version 4.2 with Informix as of 14.10.xC6.

© 2017 IBM Corporation

Limit Network Exposure

- **“Ensure that MongoDB runs in a trusted network environment and limit the interfaces on which MongoDB instances listen for incoming connections. Allow only trusted clients to access the network interfaces and ports on which MongoDB instances are available.”**

- **MongoDB talks about security/configuration hardening and the first thing they mention is:**
  - “MongoDB Configuration Hardening - For MongoDB, ensure that HTTP status interface and the REST API are disabled in production to prevent potential data exposure to attackers.
  - Deprecated since version 3.2: HTTP interface for MongoDB”

- **So you can’t use REST protocol in production MongoDB or its HTTP interface in 3.2 or greater due to data exposure risks.**

© 2017 IBM Corporation

Limit Network Exposure

- **Informix requires a hosts.equiv file for its default authentication policy.**
  - File lists the remote hosts and users trusted by the computer on which the database server is located.
    - Trusted users, and users who log-in from trusted hosts, can access the computer without supplying a password.
  - Operating system uses the **hosts.equiv** file to determine user access without specifying a password.
- **Set trusted users or trusted hosts with the REMOTE_USERS_CFG configuration parameter to allow execution of distributed queries.**
- **Authenticating at the wire listener for Informix assures proper access.**
- **Informix provides wire listener level authentication for REST, MongoDB, and MQTT accesses.**

© 2017 IBM Corporation

Audit System Activity

- **Both MongoDB and Informix allow you to audit any or all activities in an instance.**

- **So in this regard they are the same.**
  - These activities can be quite intensive on performance manually intensive on labor if all activity is audited.
  - IBM has Guardium which overcomes this issue for both products.
    - Sits off of the server and does its data collection off of the server as well.

© 2017 IBM Corporation

Run MongoDB with a Dedicated User

- **"Run MongoDB processes with a dedicated operating system user account. Ensure that the account has permissions to access data but no unnecessary permissions."**

- **Ideally that would be either user root or user mongo.**

- **Informix has the same thing, either user informix or user root.**

© 2017 IBM Corporation

IBM

Run MongoDB with Secure Configuration Options

- **MongoDB supports the execution of JavaScript code for certain server-side operations:**
  - **mapReduce, group, and $where.**
    - If these operations aren't used, disable server-side scripting by using the --***noscripting*** option on the command line.

- **Use only the MongoDB wire protocol on production deployments.**
  - Do not enable the following, all of which enable the web server interface:
    - **net.http.enabled**
    - **net.http.JSONPEnabled**
    - **net.http.RESTInterfaceEnabled**

- **Leave these *disabled*, unless required for backwards compatibility.**
  - Deprecated since version 3.2: HTTP interface for MongoDB

- **Keep input validation enabled. MongoDB enables input validation by default through the wireObjectCheck setting.**
  - This ensures that all documents stored by the **mongod** instance are valid ***BSON*****.**

© 2017 IBM Corporation

Request a Security Technical Implementation Guide (STIG) (where applicable)

- **This item, when clicked here on MongoDB's web site, takes you to a web page where it asks you for data on yourself for either a STIG document or again, less noticeably for a Security Reference Architecture document, on an alternate click on the same page.**
  - In both cases, you will be contacted by MongoDB marketing staff.

- **STIG is in reference to DoD (Department of Defense) requirements.**

© 2017 IBM Corporation

Appendix H – Wire Listener Configuration File Options

© 2021 IBM Corporation

Wire Listener Configuration File

- **Settings controlling the wire listener and connection between the client and database server are set in the wire listener configuration file**

- **The default name for the configuration file is $INFORMIXDIR/etc/jsonListener.properties.**
  - You can rename this file, but the suffix must be **.properties**

- **If you create a server instance during the installation process, a configuration file that is named jsonListener.properties is automatically created with default properties, otherwise you must manually create the configuration file**
  - Use the **$INFORMIXDIR/etc/jsonListener-example.properties** file as a template

- **In the configuration file that is created during installation, and in the template file, all of the parameters are commented out by default**
  - To enable a parameter, you must uncomment the row and customize the parameter

© 2017 IBM Corporation

IBM

Wire Listener Configuration File Parameters

- **Required**
  - url

- **Setup and configuration**
  - documentIdAlgorithm
  - include
  - listener.onException
  - listener.hostName
  - listener.port
  - listener.type
  - response.documents.count.default
  - response.documents.count.maximum
  - response.documents.size.maximum
  - sharding.enable
  - sharding.parallel.query.enable

© 2017 IBM Corporation

Wire Listener Configuration File Parameters

- **Command and operation configuration**
  - collection.informix.options
  - command.listDatabases.sizeStrategy
  - update.client.strategy
  - update.mode

- **Database resource management**
  - database.buffer.enable
  - database.create.enable
  - database.dbspace
  - database.locale.default
  - database.log.enable
  - dbspace.strategy
  - fragment.count
  - jdbc.afterNewConnectionCreation

© 2017 IBM Corporation

Wire Listener Configuration File Parameters

- **MongoDB compatibility**
  - compatible.maxBsonObjectSize.enable
  - mongo.api.version
  - update.one.enable

- **Performance**
  - delete.preparedStatement.cache.enable
  - insert.batch.enable
  - insert.batch.queue.enable
  - insert.batch.queue.flush.interval
  - index.cache.enable
  - index.cache.update.interval
  - insert.preparedStatement.cache.enable
  - preparedStatement.cache.enable
  - preparedStatement.cache.size

© 2017 IBM Corporation

Wire Listener Configuration File Parameters

- **Security**
  - authentication.enable
  - authentication.localhost.bypass.enable
  - command.blacklist
  - db.authentication
  - listener.admin.ipAddress
  - listener.authentication.timeout
  - listener.http.accessControlAllowCredentials
  - listener.http.accessControlAllowHeaders
  - listener.http.accessControlAllowMethods
  - listener.http.accessControlAllowOrigin
  - listener.http.accessControlExposeHeaders
  - listener.http.accessControlMaxAge
  - listener.http.headers
  - listener.http.headers.size.maximum
  - listener.rest.cookie.domain
  - listener.rest.cookie.httpOnly
  - listener.rest.cookie.length
  - listener.rest.cookie.name

© 2017 IBM Corporation

Wire Listener Configuration File Parameters

- **Security**
  - listener.rest.cookie.path
  - listener.rest.cookie.secure
  - listener.ssl.algorithm
  - listener.ssl.ciphers
  - listener.ssl.enable
  - listener.ssl.key.alias
  - listener.ssl.key.password
  - listener.ssl.keyStore.file
  - listener.ssl.keyStore.password
  - listener.ssl.keyStore.type
  - listener.ssl.protocol
  - security.sql.passthrough

© 2017 IBM Corporation

IBM

Wire Listener Configuration File Parameters

- **Wire listener resource management**
  - cursor.idle.timeout
  - listener.connectionPool.closeDelay.time
  - listener.connectionPool.closeDelay.timeUnit
  - listener.idle.timeout
  - listener.idle.timeout.minimum
  - listener.input.buffer.size
  - listener.memoryMonitor.enable
  - listener.memoryMonitor.allPoint
  - listener.memoryMonitor.diagnosticPoint
  - listener.memoryMonitor.zeroPoint
  - listener.output.buffer.size
  - listener.pool.admin.enable
  - listener.pool.keepAliveTime
  - listener.pool.queue.size
  - listener.pool.size.core
  - listener.pool.size.maximum
  - listener.socket.accept.timeout
  - listener.socket.read.timeout

© 2017 IBM Corporation

Wire Listener Configuration File Parameters

- **Wire listener resource management**
  - pool.connections.maximum
  - pool.idle.timeout
  - pool.idle.timeunit
  - pool.lenient.return.enable
  - pool.lenient.dispose.enable
  - pool.semaphore.timeout
  - pool.semaphore.timeunit
  - pool.service.interval
  - pool.service.threads
  - pool.service.timeunit
  - pool.size.initial
  - pool.size.minimum
  - pool.size.maximum
  - pool.type
  - pool.typeMap.strategy
  - response.documents.size.minimum
  - timeseries.loader.connections

© 2017 IBM Corporation

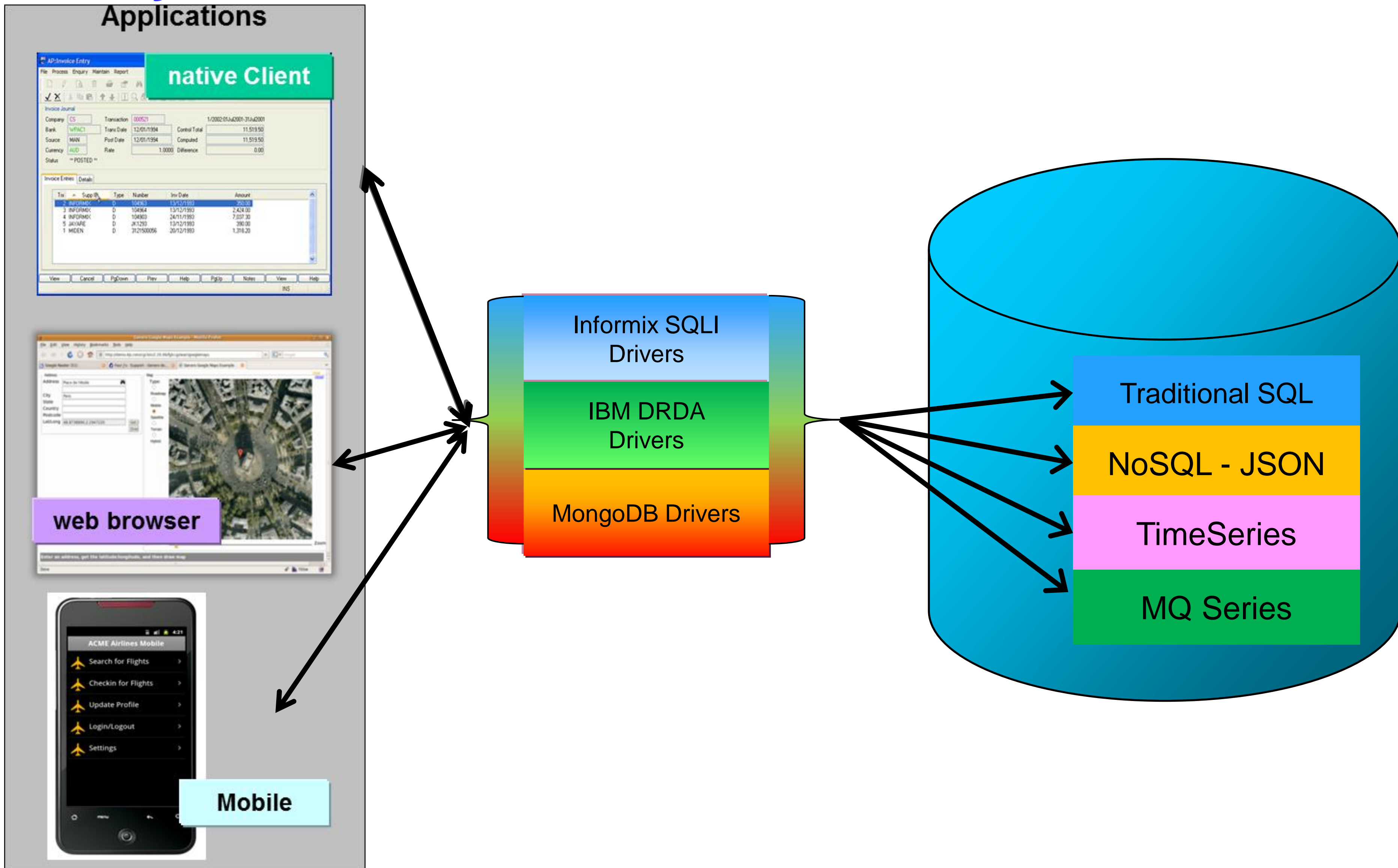
MongoDB Limits – Hard and Soft (2)

- **Speaker notes on this slide have all of the details.**
  - A single MMAPv1 based mongod instance cannot manage a data set that exceeds maximum virtual memory address space provided by the underlying operating system.

	Virtual Memory Limitations	
O/S	Journaled	Not Journaled
Linux	64 TB	128 TB
Windows Server 2012 R2 & Windows 8.1	64 TB	128 TB
Windows (otherwise)	4 TB	8 TB

© 2017 IBM Corporation

Ability for All Clients to Access All Data Models



© 2017 IBM Corporation

Hybrid access: From MongoAPI to relational tables.

- **You want to develop an application with MongoAPI, but…**
  - You already have relational tables with data.
  - You have views on relational data
  - You need to join tables
  - You need queries with complex expressions. E.g. OLAP window functions.
  - You need multi-statement transactions
  - You need to exploit stored procedure
  - You need federated access to other data
  - You have timeseries data.

© 2017 IBM Corporation

MongoAPI Accessing Both NoSQL and Relational Tables

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

© 2017 IBM Corporation

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

{partner: “Pronto”, pnum:”1748”, Country: “Australia”}
{partner: “Kazar”, pnum:”1746”, Country: “USA”}
{partner: “Diester”, pnum:”1472”, Country: “Spain”}
{partner: “Consultix”, pnum:”1742”, Country: “France”}

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

© 2017 IBM Corporation

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**

© 2017 IBM Corporation

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
```


© 2017 IBM Corporation

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”;
```


© 2017 IBM Corporation

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"}
```


© 2017 IBM Corporation

Seamless federated access

- **create database newdb2;**
- **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;**

© 2017 IBM Corporation

Get results from a stored procedure thru a view in Nosql.

```
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" }
```


© 2017 IBM Corporation

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 );
```


© 2017 IBM Corporation

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 }
```


© 2017 IBM Corporation

IBM

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.**

© 2017 IBM Corporation

Analytics

SQL & BI Applications

ODBC, JDBC connections

Informix

JSON Collections

Customer

Orders

Tables & views

Relational Tables

partners

Inventory

CRM

© 2017 IBM Corporation

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**

© 2017 IBM Corporation