JSON Cheat Sheet for IBM® Informix® Version 12.10.xC4

The information in this quick reference lists commonly used JSON features supported by Informix. This cheat sheet assumes that you are using the MongoDB shell. All MongoDB drivers are supported. For an online version of this information with links to full argument descriptions, see http://www.ibm.com/support/docview.wss?uid=swg27041825.

Send comments about this card or suggestions for additional quick reference topics to docinf@us.ibm.com.

How are JSON collections different from relational tables?

A JSON collection holds BSON (binary JSON) data. BSON documents have a flexible schema, which is a property of unstructured data, The structure and contents of BSON documents can differ from one document to another. With relational tables, all rows must following the same predefined structure.

How do MongoDB features map to SQL features?

MongoDB collection methods	Informix SQL statements
`find`	**`SELECT`**
`save`	**`INSERT`**
`remove`	**`DELETE`**
`update`	**`UPDATE`**
`ensureIndex`	**`CREATE INDEX`**
`sort`	**`ORDER BY`**
`limit`	**`LIMIT/FIRST`**

What are the components of the Informix JSON solution?

There are three main components: the Informix server, the JSON wire listener, and a MongoDB client. The wire listener is a mid-tier gateway server that enables communication between MongoDB client and the Informix server.

Commonly customizable JSON wire listener properties

The properties that control the wire listener and the connection between the client and database server are set in the `%INFORMIXDIR%\etc\jsonListener.properties` file. The url parameter is required, but all other parameters are optional. Here are the commonly customized parameters.

`url`

This required parameter specifies the host name, database server, user ID, and password that are used in connections to the Informix database server.

`authentication.enable`

This optional parameter indicates whether to enable user authentication. The default value is false.

`database.locale.default`

This optional parameter specifies the default locale to use when a new database is created. The default value is en_US.utf8.

`listener.port`

This optional parameter specifies the port number to listen on for incoming connections from MongoDB clients. The default value is 27017.

`security.sql.passthrough`

This optional parameter indicates whether to enable support for issuing SQL statements by using JSON documents. The default value is false.

`sharding.enable`

This optional parameter indicates whether to enable the use of commands and queries on sharded data. The default value is false.

Starting the JSON wire listener from the command line

If the wire listener is not already running, you can start it by using a system command. For example:

```
java -jar $INFORMIXDIR/bin/jsonListener.jar -config
/home/jdoe/business/2014/projects/NoSQL/jsonListener_frodo.properties -start -loglevel debug
```

Create, read, update, and delete (CRUD) operations on collections and tables

These standard CRUD operations are supported by Informix:

- **`insert`**
- **`find`**
- **`update`**
- **`remove`**

This table shows an example of MongoDB operations and comparable SQL statements against relational tables. In the example, the retirement age of a customer is queried:

MongoDB operation	Informix SQL statement
`db.customer.insert( { name: “John", age: 65 } )`	`INSERT INTO customer (name, age) VALUES (“John”,65)`
`db.customer.find()`	`SELECT * FROM customer`
`db.customer.find( {age: { $gt:65 } } )`	`SELECT * FROM customer WHERE age > 65`
`db.customer.drop()`	`DROP TABLE customer`
`db.customer.ensureIndex( { name : 1, age : -1 } )`	`CREATE INDEX idx_1 on customer(name , age DESC)`
`db.customer.remove( {age: { $gt:65 } } )`	`DELETE FROM customer where age > 65`
`db.customer.update( { age: { $gt: 64 } }, { $set: { status: “Retire" } }, { multi: true } )`	`UPDATE customer SET status = “Retire" WHERE age > 64`

Implicit operations for JSON collections and databases

If you insert into a non-existent JSON collection, a collection is implicitly created.

If you create a JSON collection in a non-existent database, a database is implicitly created.

Creating and listing indexes

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”] } )
```

You can list indexes by running the MongoDB getIndexes command.

Accessing multiple databases per connection

In standard Informix JDBC connections, you must specify the database name on the connection string and you must create one connection per database. In MongoDB, all messages include a fully qualified namespace that includes the database name and the collection. MongoDB connections are not associated with a particular database and each individual message or command specifies the intended database. A single MongoDB connection can switch between databases.

Moving data to and from collections and tables

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 from MongoDB to Informix.

Viewing usage statistics

You can run the MongoDB serverStatus command to get the wire listener status information, including:

- Uptime
- Number of active and available connections
- Number of open cursors
- Total number of requests
- Counters for the number operations (queries, inserts, updates, deletes, commands, etc)