IBM DB2 for i: JSON Store Technology Preview

Storing web documents in DB2 for i

John W. Eberhard

June 05, 2015

The JavaScript Object Notation (JSON) Store Technology Preview for IBM i allows the storage and retrieval of JSON documents in database tables. This article describes how to enable the JSON store on IBM® DB2® for i. It also illustrates the use of the DB2 JSON command line and the DB2 JSON Java™ application programming interface (API) to store and retrieve JSON documents. Because this support is similar to the support provided by DB2 for Linux, UNIX, and Windows® and IBM DB2 for z/OS, this article refers the reader to the documentation provided by those products and describes the minor differences with the support provided by DB2 for i.

The latest DB2 program temporary fix (PTF) groups (SF99701 for 7.1 and SF99702 for 7.2), released on the same date as Technology Refresh 10 for IBM i 7.1 and Technology Refresh 2 for IBM i 7.2, include the JSON Store Technology Preview. This new feature allows JSON documents to be stored and retrieved using DB2 for i database tables. This feature adds JSON support that is comparable to the support added for DB2 for Linux, UNIX, and Windows and DB2 for z/OS. The JSON support on IBM i includes support for the DB2 JSON command line processor and the DB2 JSON Java API. The purpose of the article is to describe the JSON functionality on IBM i and refer the reader to other IBM developerWorks® articles containing more details. The JSON Store technical capabilities are described solely by developerWorks articles.

JSON

As explained in the DB2 JSON capabilities, Part 1: Introduction to DB2 JSON article, JSON is a lightweight format to exchange information. JSON documents are easily created and consumed by JavaScript programs running on web browsers. Because of its simple structure, JSON has become a common means of information interchange for web applications. It is popular because it is much lighter weight than the alternative approach of XML. More details about JSON and its benefits are explained in the Part 1 article.

There are three ways to utilize JSON on IBM i.

- db2nosql -- The DB2 JSON command line processor
- The DB2 JSON Java API

- SYSTOOLS.BSON2JSON -- A function to retrieve a JSON document stored within a DB2 for i table.

Storing JSON documents in DB2 for i provides excellent reliability and availability. The DB2 JSON Store does not include query engine indexing support and does not support shredding and publishing JSON using SQL statements.

db2nosql: DB2 JSON command line processor

The DB2 JSON command line processor is included in IBM i within the integrated file system. The full IFS path name containing the processor is /QIBM/ProdData/OS/SQLLIB/bin/db2nosql. On IBM i, the DB2 JSON command line processor is used from the IBM i QSHELL environment. To enter the QSHELL environment, run the `Start QSHELL (STRQSH)` command.

The DB2 JSON command line processor can also be used on a client system by downloading the following files from the /QIBM/ProdData/OS/SQLLIB/bin directory.

- db2nosql
- db2nosql.bat
- jline-0.9.93.jar
- js.jar
- mongo-2.8.0.jar
- nosqljson.jar
- servlet-api.jar

You also need to obtain a version of the jt400.jar file, which can be found in the /QIBM/ProdData/os400/jt400/lib/java6 directory. Place these files in the same directory on the client and run db2nosql (on UNIX systems) or db2nosql.bat (on Windows systems) to start the command line processor. An easy way to set up a Microsoft® Windows® system (with FTP installed) is to start a Windows command prompt and run the following commands, where MYIBMI is the name of your IBM i partition.

Listing 1. Windows client setup

```
C:\>mkdir nosql
C:\>cd nosql
C:\nosql>ftp MYIBMI
ftp> bin
ftp> cd /QIBM/ProdData/OS/SQLLIB/bin
ftp> mget *
ftp> cd /QIBM/ProdData/OS400/jt400/lib/java6
ftp> get jt400.jar
ftp> quit
```

Starting db2nosql on IBM i

To use the command line processor on IBM i, use STRQSH to start QSHELL and run /QIBM/ProdData/OS/SQLLIB/bin/db2nosql. When used on IBM i, the command line processor uses the native JDBC driver to connect to the current partition using the current user's credentials. If you need to connect using another user's credentials, you can specify those credentials using the `-user` and `-password` parameters of `db2nosql` as shown in the following example.

Listing 2. Using db2nosql with another user's credentials

```
$ /QIBM/ProdData/OS/SQLLIB/bin/db2nosql  -user myuserid-password mypassword
JSON Command Shell Setup and Launcher.
Type db2nosql -help to see options
…


IBM DB2 NoSQL JSON API 1.1.0.0 build 1.4.7
Licensed Materials - Property of IBM
(c) Copyright IBM Corp. 2013,2015 All Rights Reserved.

nosql>Type your JSON query and hit <ENTER>
nosql>Type help() or help for usage information. All commands are case sensitive.
```

Starting db2nosql on other platforms

The `db2nosql` command can be used on client systems to interactively work with JSON data within DB2 for i. The target IBM i partition is specified using the `-hostname` or `-url` parameters, where the URL is a JDBC-style URL. For IBM i, the JDBC URL must be of the form *jdbc:as400:partitionName*. The following examples use these options, where the partition name is `MYIBMI`.

Listing 3. Using the hostname and url parameters of db2nosql

```
C:\nosql>db2nosql -hostname MYIBMI -user json –password PASSWORD
JSON Command Shell Setup and Launcher.
db2nosql.bat is launched with several options.
...
> db2nosql -url jdbc:as400:MYIBMI -user json –password PASSWORD
JSON Command Shell Setup and Launcher.
Type db2nosql -help to see options
...
```

Note: `db2nosql` requires that a version of Java is available on the command path. The version of Java used should be at least Java 6.

Setup

Before using JSON on an IBM i partition, a one-time setup must be performed. This is accomplished by running db2nosql with the `-setup enable` option, as shown in the following example. This setup needs to be done only once per IBM i partition.

Listing 4. Using db2nosql to enable JSON support

```
$ /QIBM/ProdData/OS/SQLLIB/bin/db2nosql -setup enable
JSON Command Shell Setup and Launcher.
Type db2nosql -help to see options

IBM DB2 NoSQL JSON API 1.1.0.0 build 1.4.7
Licensed Materials - Property of IBM
(c) Copyright IBM Corp. 2013,2015 All Rights Reserved.

Executing SQL...
CDJSN1209I Database artifacts created successfully.
```

On IBM i, JSON documents are stored using binary large object (BLOB) columns of DB2 for i tables. If commitment control is used, IBM i requires the journaling to be active when using those

tables. To assure that this condition is met, it is recommended that JSON collections are created in SQL schemas, which automatically activates journaling for tables in the schema. An SQL schema is created using the "CREATE SCHEMA" SQL statement. Using a JSON collection in an IBM i library without journaling results in SQL7008, "File not valid for operation", errors when using commitment control to manipulate those tables.

Using db2nosql

For detailed information about using db2nosql, refer to the DB2 JSON capabilities, Part 2: Using the command-line processor developerWorks article. In the current article we provide samples of what can be accomplished using db2nosql on IBM i.

Any JSON *collection* maps to a single DB2 for i table. The table is located in whichever schema is in use by JSON and is dedicated to only containing JSON documents. To specify the current schema, the db2nosql `use` command is used. For example, the following command causes db2nosql to use the JSONLIB schema.

```
nosql>use JSONLIB
CDJSN1214I Switched to schema: "JSONLIB"
```

As mentioned above, this schema should be a schema created using the `CREATE SCHEMA` SQL statement. If the schema hasn't been created, you can use the `db.sqlUpdate()` command to run the `CREATE SCHEMA` SQL statement. Here is an example of using db2nosql to create the JSONLIB schema and then to use it.

Listing 5. Creating and using a schema

```
nosql>db.sqlUpdate("CREATE SCHEMA JSONLIB")
0
nosql>use JSONLIB
CDJSN1214I Switched to schema: "JSONLIB"
nosql>
```

You can see information about the current database using the `db` command.

```
nosql>db
Database: jdbc:db2:localhost Schema: JSONLIB
```

In order to store JSON documents, a JSON collection must exist to contain the documents. A JSON collection stores JSON documents using a DB2 for i table. The sole purpose of such a table is to house JSON documents accessed by db2nosql or the DB2 Java JSON API. A table containing a JSON collection contains two columns named ID and DATA. The ID column contains a unique identifier for the JSON document that is inserted into the table. The DATA column is a BLOB column which contains the JSON document [represented in the Binary JSON (BSON) format]. The maximum size of a JSON document is approximately 16 megabytes (MB).

One way to create a collection is to use the `db.createCollection()` command. In JSON, the JSON collection names are case sensitive. If lower case letters are used when creating a collection, the tables will be created on the IBM i using delimited names. If you do not wish to have

delimited table names in your schemas, you should create your JSON collections using uppercase names. The following commands create the JSON collections: `JSONA` and `jsonb`.

Listing 6. Creating JSON collections

```
nosql>db.createCollection("JSONA")
CDJSN1006I Collection: "db.JSONLIB."JSONA"" created. Use "JSONA".
nosql>db.createCollection("jsonb")
CDJSN1006I Collection: "db.JSONLIB."jsonb"" created. Use "jsonb".
```

After creating the `JSONA` and `jsonb` collections, we can examine the contents of the JSONLIB library on IBM i. We see the impact of using a lower or mixed case JSON collection name, as `jsonb` created as a FILE object with a delimited name.

Listing 7. JSON file objects

```
Work with Objects
Opt  Object      Type      Library     Attribute   Text
"jsonb"     *FILE     JSONLIB     PF
JSONA      *FILE     JSONLIB     PF
```

Because DB2 for i table names are limited to 128 characters, JSON collection names are limited to 128 if they consist solely of uppercase letters, and are limited to 126 characters if they require delimiting.

After a JSON collection is created, documents can be inserted using the `db.<collectionName>.insert(<document>)` command. If the collection does not exist, the `insert` command will implicitly create the collection. Because of this behavior, it is not necessary to use the `createCollection` command described earlier. Here is an example of inserting a simple JSON document into the JSONA collection.

Listing 8. Inserting a JSON document

```
nosql>db.JSONA.insert({name:"George",age:20,increase:0.12})
CDJSN1000I Command successfully completed.
```

Because the JSON document is converted to the binary format, it is checked to assure that it is valid data. Attempting to insert an invalid JSON document results in an error that might look as shown in the following example.

Listing 9. Attempting to insert invalid JSON

```
nosql>db.JSON.insert({notvalid})
CDJSN0114E Invalid command syntax
Error details shown below, but may not be the source of the problem.
db.JSON.insert({notvalid})
^
Error details: missing : after property id
Column number: 25
```

After the documents are inserted into a JSON collection, you can retrieve them using the `db.<collectionName>.find()` command. Invoking the `find` command without any parameters retrieves all the documents from a collection, as shown in the following example.

Listing 10. Retrieving all JSON documents

```
nosql>db.JSONA.find()
nosql>Row 1:
nosql> {
"_id":{"$oid":"5510844569849c86c5832917"},
"name":"George",
"age":20,
"increase":0.12
}
nosql>CDJSN1206I "1" row(s) returned in "61" milliseconds.
```

The `find` command can also be used with search options. For more details, refer to the Part 2 article. Here are examples of finding the document that was inserted earlier.

Listing 11. Finding a JSON document with name as George

```
nosql>db.JSONA.find({name:"George"})
nosql>Row 1:
nosql> {
"_id":{"$oid":"5510844569849c86c5832917"},
"name":"George",
..
nosql>CDJSN1206I "1" row(s) returned in "240" milliseconds.
```

Listing 12. Finding a JSON document with age as 20

```
nosql>db.JSONA.find({age:20})
db.JSONA.find({age:20})
nosql>Row 1:
nosql> {
"_id":{"$oid":"5510844569849c86c5832917"},
"name":"George",
"age":20,
"increase":0.12
}
nosql>CDJSN1206I "1" row(s) returned in "42" milliseconds.
```

The Part 2 article also describes additional options for querying, updating, aggregating, and administering JSON data. Many features explained in the Part 2 article are currently not available on IBM i. Bi-temporal capabilities and time travel queries are currently not supported. The use of indexes is currently not supported. Also, the use of any `limit()` or `skip()` operations after a `sort()` operation will not produce the expected results because the `limit()` and `skip()` operations are internally applied to the unordered data.

Using the JSON Java API

The JSON Store Technology Preview also includes a JSON Java API that can programmatically work with JSON documents stored in DB2 for i. You can find details about the JSON Java API in the developerWorks article, DB2 JSON capabilities, Part 3: Writing applications with the Java API. This article includes a sample Java program that illustrates some of the features of the Java API.

The Java API can be found in the com.ibm.nosql.json.api package of the nosqljson.jar file referenced above. The Java API includes the following main objects: the DB object, the DBCollection object, the DBBasicObject object, and the DBCursor object.

The DB object

The JSON DB object represents a JSON database, and corresponds to a SCHEMA on an IBM i partition. As illustrated in the following sample program, a `DB` object can be obtained by passing a JDBC connection and SCHEMA name to the `NoSQLClient.getDB()` method.

Listing 13. Obtaining a DB object using a JDBC connection

```
// Obtain a JDBC connection using the JTOpen JDBC driver
String url = "jdbc:as400://"+system;
Connection con = DriverManager.getConnection(url,
userid,
password);

// get a DB object from a JDBC connection
DB db = NoSQLClient.getDB(con, schema)
```

A DB object can also be obtained by using the JDBC URL, user ID, and password instead of the JDBC connection.

```
// Alternatively, you can get a DB using the JDBC url
db = NoSQLClient.getDB(url, userid, password, schema);
```

The DB object can be used to run SQL statements. This article's sample program uses the `sqlUpdate()` method to run two SQL statements. The first statement ensures that the SCHEMA exists. The second statement deletes the contents of the table which the program will use to store the JSON documents.

```
db.sqlUpdate("CREATE SCHEMA "+schemaName);
db.sqlUpdate("DELETE FROM "+schemaName+".JSONSAMPLE");
```

The DBCollection object

The `DBCollection` object represents a collection of JSON documents. It is obtained from a DB object using the `getCollection()` method. In the following example code, the DB2 for i table JSONSAMPLE is the target of the `getCollection()` method.

```
DBCollection col = db.getCollection("JSONSAMPLE");
```

The table to hold the JSON collection is created when the first item in inserted in the collection. To insert an item into the collection, the `insert()` method is used. The following example inserts a JSON document into the collection.

Listing 14. Inserting a JSON document

```
//
// JSON documents can be inserted directly into the JSON collection.
//
col.insert("{'name':'George','phone':'555-3232'}" );
```

The DBBasicObject object

A `DBBasicObject` contains a JSON document. It can be used to programmatically build JSON documents that are then inserted in a JSON collection. To build a JSON document, just create

a `DBBasicObject` object and then use the append method to add elements to the document. The following sample illustrates the building of a JSON document and inserting it into a JSON collection.

Listing 15. Creating and inserting a DBBasicObject object

```
// Create a BasicDBObject to create an object to be inserted
BasicDBObject json = new BasicDBObject();
json.append("name","Harry");
json.append("phone","555-4364");
col.insert (json);
```

The DBCursor object

The `DBCursor` object is used to return JSON documents from a JSON collection. A `DBCursor` object is returned by the `find()` method of `DBCollection`. The elements of the `DBCursor` object are accessed using the `hasNext()` and `next()` methods as shown in the following example.

Listing 16. Using DBCursor to access JSON documents

```
System.out.println("Querying all");
DBCursor cursor = col.find();
while (cursor.hasNext()) {
DBObject obj = cursor.next();
System.out.println("Retrieved: "+obj);
}
```

You can use the `find()` command to specify the documents to find. This can be done passing the `find()` command either using a JSON value (passed as a string) or a `DBBasicObject` object as shown in the following examples.

Listing 17. Finding JSON documents with name as George

```
cursor = col.find("{'name':'George'}");
while (cursor.hasNext()) {
DBObject obj = cursor.next();
System.out.println("Retrieved: "+obj);
}
```

Listing 18. Finding JSON documents with name as Harry

```
BasicDBObject queryObject = new BasicDBObject();
queryObject.append("name","Harry");
cursor = col.find(queryObject);
while (cursor.hasNext()) {
DBObject obj = cursor.next();
System.out.println("Retrieved: "+obj);
}
```

You can find more details about this object in the Part 3 article. Also, refer to Javadoc for the Java API. When referencing the Part 3 article, the limitations specified in the *db2nosql* section of this article also apply – specifically, indexes cannot be created on JSON collections.

Note: The lack of index support causes queries on JSON collections to run without index optimization.

Generating JSON document using SQL queries

The JSON Java API can also generate simple JSON documents using SQL queries. These simple JSON documents do not contain JSON arrays or nested JSON documents. To do this, just use `sqlQueryMethod()` of the DB object to run an SQL query. This will return a `DBCursor` object which contains `DBObject` objects representing the rows of the query result. Each `DBObject` object represents a JSON document containing elements, consisting of the column name and column value. Here is a code sample using this approach that queries two columns, named SCHEMA and COL, from the SYSTOOLS.SYSJSON_INDEX table. The SYSTOOLS.SYSJSON_INDEX table is a table used by the DB2 JSON support to track the JSON collections (that is, tables) which exist on IBM i.

Listing 19. Using an SQL query to generate a JSON document

```
cursor = db.sqlQuery("select distinct "+
"COLLECTION_SCHEMA as schema, "+
"COLLECTION as col "+
"from SYSTOOLS.SYSJSON_INDEX "+
"WHERE COLLECTION_SCHEMA='"+schema+"'");
while (cursor.hasNext()) {
DBObject obj = cursor.next();
System.out.println("Retrieved: "+obj);
}
```

On running this code sample, the following output is displayed. In this, SCHEMA and COL are the column names specified in the query and JSONLIB and JSONSAMPLE are values of those columns returned by the query.

```
Retrieved: {"SCHEMA":"JSONLIB","COL":"JSONSAMPLE"}
```

This ability provides Java programs with the ability to easily create JSON documents from SQL queries.

Accessing the JSON documents from SQL

The JSON Store Technology Preview includes a user-defined function (UDF) which can be used to convert JSON documents from their internal binary form into their character form. The SYSTOOLS.BSON2JSON UDF can be used to retrieve the contents of a BSON document as JSON. As mentioned above, JSON documents are stored in the BSON format in the DATA column of the tables that represent JSON collections. Here is an example of using this UDF to view the JSON in a table.

Listing 20. Using BSON2JSON to view JSON documents

```
>select SYSTOOLS.BSON2JSON(DATA) from JSONLIB.JSONSAMPLE
…
{"name":"George","phone":"555-3232"}
{"name":"Harry","phone":"555-4364"}
```

Because the BSON2JSON UDF produces a CLOB, when STRSQL runs the query, the following output is displayed.

```
BSON2JSON
*POINTER
*POINTER
********  End of data  ********
```

To rectify this problem, cast the resultant CLOB to a VARCHAR as shown in the following example.

Listing 21. Using CAST to view JSON documents

```
select CAST(SYSTOOLS.BSON2JSON(DATA) AS VARCHAR(32000)) from JSON.JSONSAMPLE
…
CAST function
{"name":"George","phone":"555-3232"}
{"name":"Harry","phone":"555-4364"}
********  End of data  ********
```

Or better yet, run interactive SQL statements using a modern and strategic interface. Run SQL scripts from System i Navigator. Refer to the IBM i Access page for more details. This preferred interface will result in a much better user experience.

You can find more information about the SYSTOOLS.BSON2JSON UDF in the documentation for DB2 for z/OS.

Limitations of the JSON support

As a reminder, JSON on the IBM i currently does not support the following features that are mentioned in the other developerWorks articles. No temporal functions are supported. The creation of indexes on JSON documents is not supported. The developerWorks articles mention a wire listener. A wire listener is not included with the IBM i JSON support.

Summary

This article has described the JSON Store Technology Preview for IBM i. After the JSON support is enabled using the DB2 JSON command line, the DB2 JSON command line and DB2 JSON Java API are used to insert and retrieve JSON documents from DB2 for i tables. The DB2 JSON Java API can also be used to construct a simple JSON document from SQL queries. Finally, the SYSTOOLS.BSON2JSON UDF can be used in SQL queries to view the JSON documents stored in the DB2 for i tables.

Downloadable resources

Description	Name	Size
	JsonSample	4.76 KB

Related topics

- DB2 JSON capabilities, Part 1: Introduction to DB2 JSON
- DB2 JSON capabilities, Part 2: Using the command-line processor
- DB2 JSON capabilities, Part 3: Writing applications with the Java API
- DB2 NoSQL/JSON API Documentation
- DB2 for z/OS BSON2JSON