

The Couchbase Blog

Couchbase, the NoSQL Database

SQL for Documents (N1QL): Brief introduction to query planning.



Keshav Murthy on January 2, 2017

SQL for Documents: Brief introduction to query planning.

I wrote [earlier](#) on the need for new kind of SQL and introducing SQL For Documents (N1QL). In this blog, we'll discuss query planning phase of N1QL in the upcoming Couchbase Sherlock release. N1QL itself has gone through multiple developer previews. Checkout the online tutorial at <http://query.couchbase.com> ; Download developer preview of Couchbase Sherlock release, which includes SQL for Documents(N1QL).

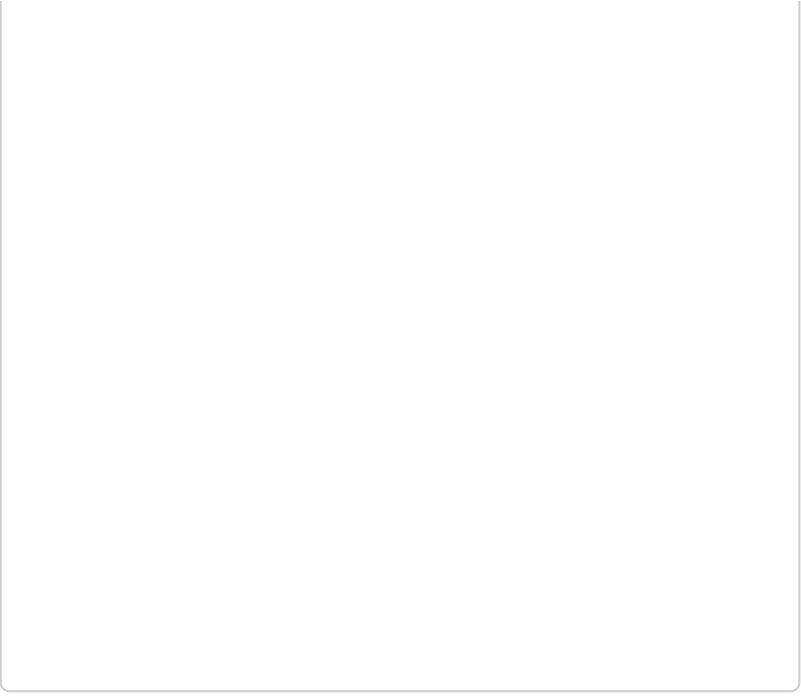
What's N1QL? It is a modern query processing engine designed to provide SQL for documents (e.g. JSON) on modern distributed architecture with flexible data model. Modern databases are deployed on massive clusters, use flexible data model (JSON, Column family, etc). N1QL supports enhanced SQL on top of this data model to make query processing easier.

Query execution: Overview

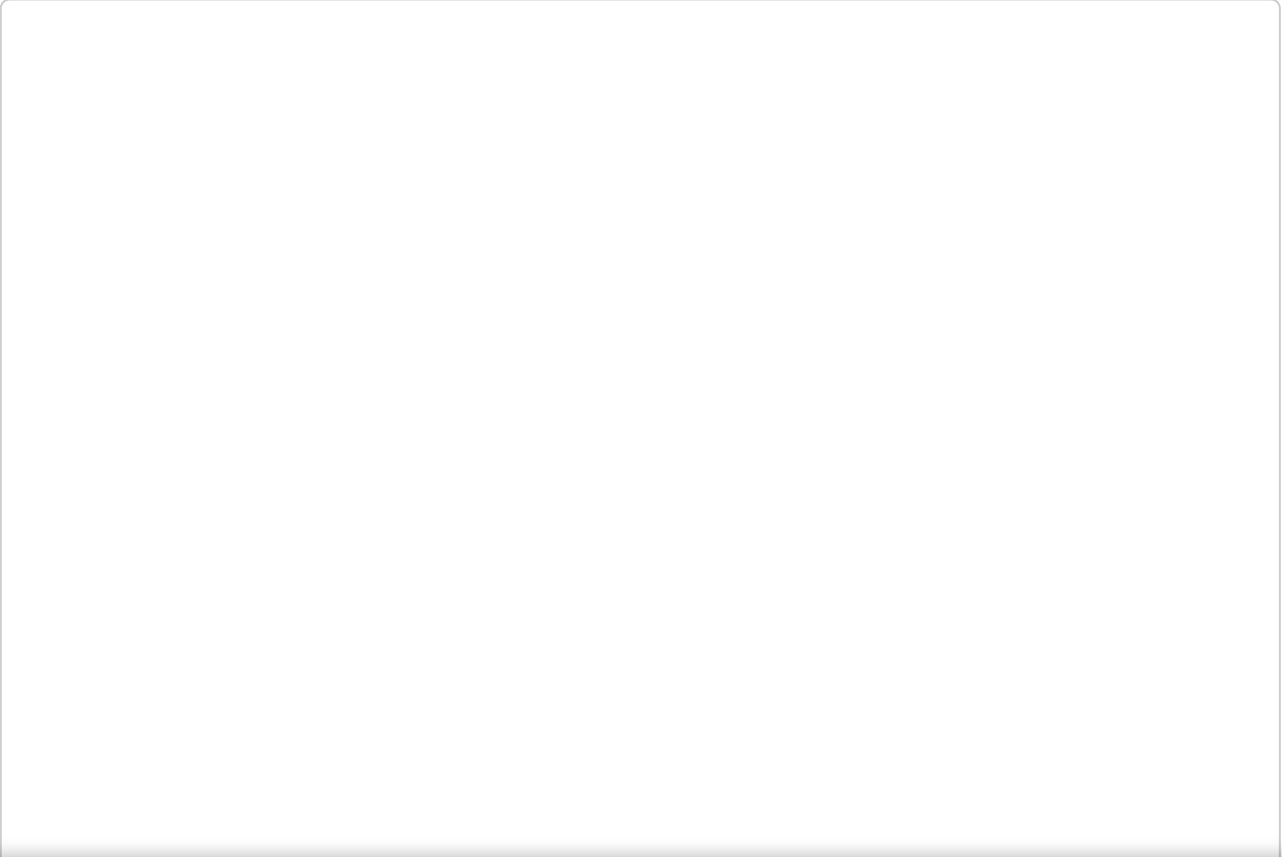
Applications and their drivers submit the N1QL query to one of the available query nodes. The Query node analyzes the query, uses meta data on underlying objects to figure out the optimal execution plan, which it then executes. During execution, depending on the query, using applicable indices, query node works with index and data nodes to retrieve and perform the select-join-project operations. Since, Couchbase is a clustered database, you scale out data, index and query nodes to fit your performance and availability goals.

Next →

Using Jil for custom JSON S...



This figure (thanks, to @N1QL architect Gerald Sangudi for the figures) shows all the possible phases a query goes through to return the results. Not all queries need to go thru every phase, some go thru many of these phases multiple times. For example, sort phase can be skipped when there is no ORDER BY clause in the query; scan-fetch-join phase multiple times to perform multiple joins.



Next →

Using Jil for custom JSON S...

N1QL analyzes the query and available access path options for each keypace (table or bucket) in the query to create a query plan and execution infrastructure. The planner needs to first select the access path for each bucket, determine the join order and then determine the type of the join. Once the big decisions are made, planner then creates the infrastructure needed to execute the plan. Some operations like query parsing and planning is done serially, while other operations like fetch, join, sort are done in parallel. **In this blog, we'll focus on the brief overview of query planning phase using examples.**

Access Path Selection:

Keyspace (Bucket) access path options

- a. Keyscan access. When specific document IDs (keys) are available, Keyscan access method retrieves documents for those IDs. Any filters on that keypace is applied after that. Keyscan access method can be used when a keypace is queried by itself or during join processing. The keyscan is commonly used to retrieve qualifying documents on for the inner keypace during join processing.
- b. PrimaryScan access: This is equivalent of full table scan in relational database systems. Documents IDs are not given and no qualifying secondary access methods are available for this keypace. N1QL will apply applicable filters on each document. This access method is quite expensive and the average time to return results increases linearly with number of documents in the bucket.
- c. IndexScan access: A qualifying secondary index scan is used to first filter the keypace and determine the qualifying documents IDs. It then retrieves the documents from the data store. In Couchbase, the secondary index can be a VIEW index or a global secondary index (GSI).

JOIN methods

N1QL supports nested loop access method for all the joins supports: INNER JOIN and LEFT OUTER JOIN. Here is the simplest explanation of the join method.

```
FROM (ORDERS o INNER JOIN CUSTOMER c ON KEYS o.O_C_ID)
```

For this join, ORDERS become the INNER keypace and CUSTOMER becomes the OUTER keypace. ORDERS keypace is scanned first (using one of the keypace scan options). For each qualifying document on ORDERS, we do a KEYSKAN on CUSTOMER based on the key O_C_D in the ORDERS document.

JOIN order

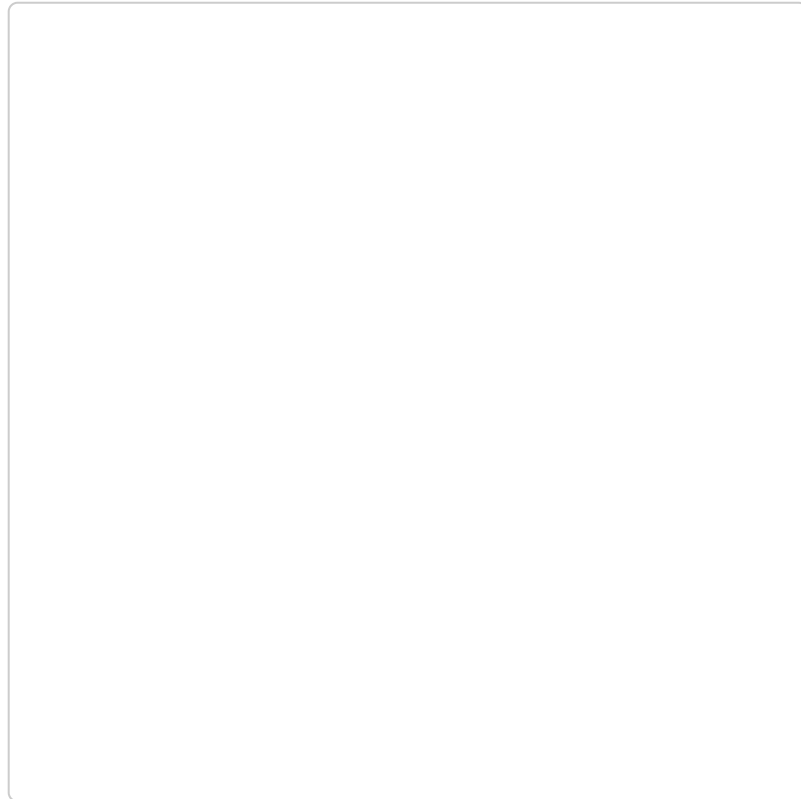
Next →

Using Jil for custom JSON S...

query.

In this blog, let's look at the planner by working thru examples. I'll use 4 keyspaces (buckets) for these. Example documents for these buckets are given at the end of this blog. Those familiar with TBOC

these. Example documents for these buckets are given at the end of this blog. Those familiar with TPCO will recognize these tables.



Each of these keyspaces have primary key index and following secondary indices.

create index CU_ID_D_ID_W_ID on CUSTOMER(C_ID, C_D_ID, C_W_ID) using gsi;

create index ST_W_ID,I_ID on STOCK(S_I_ID, S_W_ID) using gsi;

create index OR_O_ID_D_ID_W_ID on ORDERS(O_ID, O_D_ID, O_W_ID, O_C_ID) using gsi;

create index OL_O_ID_D_ID_W_ID on ORDER_LINE(OL_O_ID, OL_D_ID, OL_W_ID) using gsi;

create index IT_ID on ITEM(I_ID) using gsi;

Example 1:

If you know the document keys, specify with USE KEYS clause for each keyspace. When a USE KEYS clause is specified, the KEYSKAN access path is chosen. Given the keys, KEYSKAN will retrieve the documents from the respective **nodes** efficiently. After retrieving the specific documents, query node applies the filter c.C_STATE = "CA".

```
cbq> EXPLAIN select * from CUSTOMER c USE KEYS ["110192", "120143", "827482"] WHERE  
c.C_STATE = "CA";
```

```
{  
  "requestID": "991e69d2-b6f9-42a1-9bd1-26a5468b0b5f",  
  "signature": "json",  
  "results": [  
    {
```

Next →

Using Jil for custom JSON S...

children : [

```
{
```

```
  "generator": "KeyScan"
```

```

      "#operator": "KeyScan",
      "keys": ["110192", "120143", "827482"]
    },
    {
      "#operator": "Parallel",
      "~child": {
        "#operator": "Sequence",
        "~children": [
          {
            "#operator": "Fetch",
            "as": "c",
            "keyspace": "CUSTOMER",
            "namespace": "default"
          },
          {
            "#operator": "Filter",
            "condition": "((c.C_STATE) = 'CA')"
          },
          {
            "#operator": "InitialProject",
            "result_terms": [
              {
                "star": true
              }
            ]
          },
          {
            "#operator": "FinalProject"
          }
        ]
      }
    }
  ]
}
],
"status": "success".

```

Next →

Using Jil for custom JSON S...

```

"executionTime": "10.205523ms",
"resultCount": 1.

```

```

    "resultSize": 1403
  }
}

```

Example 2:

In this case, query is looking to count all of “CA” customers in the bucket CUSTOMER. Since we don’t have an index on key-value, c.C_YTD_PAYMENT, a primary scan of the keyspace (bucket) is chosen.

Filter c.C_YTD_PAYMENT < 100

is applied after the document is retrieved. Obviously, for larger buckets primary scan takes time. As part of planning for application performance, create relevant secondary indices on frequently used key-values within the filters.

N1QL parallelizes many of the phases within the query execution plan. For this query, fetch and filter applications are parallelized within the query execution.

```

cbq> EXPLAIN SELECT c.C_STATE AS state, COUNT(*) AS st_count
      FROM CUSTOMER c
      WHERE c.C_YTD_PAYMENT < 100
      GROUP BY state
      ORDER BY st_count desc;

```

```

"results": [
  {
    "#operator": "Sequence",
    "~children": [
      {
        "#operator": "Sequence",
        "~children": [
          {
            "#operator": "PrimaryScan",
            "index": "#primary",
            "keyspace": "CUSTOMER",
            "namespace": "default",
            "using": "gsi"
          },
          {
            "#operator": "Parallel",
            "~child": {
              "#operator": "Sequence",

```

Next →

Using Jil for custom JSON S...

```

      "#operator": "Fetch",
      "as": "c",

```

```

        "keyspace": "CUSTOMER",
        "namespace": "default"
    },
    {
        "#operator": "Filter",
        "condition": "((c.C_YTD_PAYMENT) u003c 100)"
    },
    {
        "#operator": "InitialGroup",
        "aggregates": [
            "count(*)"
        ],
        "group_keys": [
            "(c.state)"
        ]
    },
    {
        "#operator": "IntermediateGroup",
        "aggregates": [
            "count(*)"
        ],
        "group_keys": [
            "(c.state)"
        ]
    }
]
}
},
{
    "#operator": "IntermediateGroup",
    "aggregates": [
        "count(*)"
    ],
    "group_keys": [
        "(c.state)"
    ]
}

```

Next →

Using Jil for custom JSON S...

```

{
    "#operator": "FinalGroup",

```

$\},$

Using Jil for custom JSON S...

$$\},$$


```

    {
      "#operator": "Parallel",
      "~child": {
        "#operator": "FinalProject"
      }
    }
  ]
}
],

```

Example 3:

In this example, we join keyspace ORDER_LINE with ITEM. For each qualifying document in ORDER_LINE, we want to match with ITEM. The ON clause is interesting. Here, you only specify the keys for the key space ORDER_LINE (TO_STRING(ol.OL_I_ID)) and nothing for ITEM. That's because it's implicitly joined with the document key of the ITEM.

N1QL's FROM clause: ORDER_LINE ol INNER JOIN ITEM i

ON KEYS (TO_STRING(ol.OL_I_ID))

Is equivalent to SQL's: ORDER_LINE ol INNER JOIN ITEM i

ON (TO_STRING(ol.OL_I_ID) = meta(ITEM).id)

If the field is not a string, it can be converted to string using TO_STRING() expression. You can also construct the document key using multiple fields with the document.

e.g. FROM ORDERS o LEFT OUTER JOIN CUSTOMER c

ON KEYS (TO_STRING(o.O_C_ID) || TO_STRING(o.O_D_ID))

Summary is, while writing JOIN queries in N1QL, it's important to understand how the document key is constructed on the keyspace. Corollary is, it's think about these during data modeling.

First, to scan ORDER_LINE keyspace, for the given set of filters, based on available access path, planner chooses the index scan on the index **OL_O_ID_D_ID_W_ID**. As we discussed before, access path on the other keyspace in the join is always keyscan using the primary key index. In this plan, we first do the index scan on the ORDER_LINE keyspace pushing down the possible filters to the index scan. Then we retrieve the qualifying document, apply additional filters. If the document qualifies, that document is then joined with ITEM.

cbq> EXPLAIN SELECT COUNT(DISTINCT(ol.OL_I_ID)) AS CNT_OL_I_ID

FROM ORDER_LINE ol INNER JOIN ITEM i **ON KEYS (TO_STRING(ol.OL_I_ID))**

WHERE ol.OL_W_ID = 1

AND ol.OL_D_ID = 10

AND ol.OL_O_ID < 200

AND ol.OL_O_ID >= 100

Next →

Using Jil for custom JSON S...

```

{
  "requestID": "4e0822fb-0317-48a0-904b-74c607f77b2f".

```

.sequence : sequence of results of the query, if the query is a sequence,

“signature”: “json”,

“results”: [

{

“#operator”: “Sequence”,

“~children”: [

{

“#operator”: “IndexScan”,

“index”: “OL_O_ID_D_ID_W_ID”,

“keyspace”: “ORDER_LINE”,

“limit”: 9.223372036854776e+18,

“namespace”: “default”,

“spans”: [

{

“Range”: {

“High”: [

“200”

],

“Inclusion”: 1,

“Low”: [

“100”

]

},

“Seek”: null

}

],

“using”: “gsi”

},

{

“#operator”: “Parallel”,

“~child”: {

“#operator”: “Sequence”,

“~children”: [

{

“#operator”: “Fetch”,

“as”: “ol”,

Next →

Using Jil for custom JSON S...

},

{

```

        "#operator": "Join",
        "as": "i",
        "keyspace": "ITEM",
        "namespace": "default",
        "on_keys": "to_string((o1.OL_I_ID))"
    },
    {
        "#operator": "Filter",
        "condition": "((((((o1.OL_W_ID = 1) and ((o1.OL_D_ID = 10)) and ((o1.OL_O_ID)
u003c 200)) and (100 u003c= (o1.OL_O_ID))) and ((o1.S_W_ID = 1)) and ((i.I_PRICE) u003c 10))"
    },
    {
        "#operator": "InitialGroup",
        "aggregates": [
            "count(distinct (o1.OL_I_ID))"
        ],
        "group_keys": []
    },
    {
        "#operator": "IntermediateGroup",
        "aggregates": [
            "count(distinct (o1.OL_I_ID))"
        ],
        "group_keys": []
    }
]
}
},
{
    "#operator": "IntermediateGroup",
    "aggregates": [
        "count(distinct (o1.OL_I_ID))"
    ],
    "group_keys": []
},

```

Next →

Using Jil for custom JSON S...

```

"aggregates": [
    "count(distinct (o1.OL_I_ID))"

```

```

    ],
    "group_keys": []
  },
  {
    "#operator": "Parallel",
    "~child": {
      "#operator": "Sequence",
      "~children": [
        {
          "#operator": "InitialProject",
          "result_terms": [
            {
              "as": "CNT_OL_I_ID",
              "expr": "count(distinct (o1.OL_I_ID))"
            }
          ]
        },
        {
          "#operator": "FinalProject"
        }
      ]
    }
  }
]
}
]
}
}
],
"status": "success",
"metrics": {
  "elapsedTime": "272.823508ms",
  "executionTime": "272.71231ms",
  "resultCount": 1,
  "resultSize": 4047
}
}

```

Example documents:

Next →

Using Jil for custom JSON S...

```
select meta(CUSTOMER).id as PKID, ... from CUSTOMER limit 1;
```

```
"results": [
```

```
{
```

```

{
  "CUSTOMER": {
    "C_BALANCE": -10,
    "C_CITY": "ttzotwmuivhof",
    "C_CREDIT": "GC",
    "C_CREDIT_LIM": 50000,
    "C_DATA":
"sjlhfnvosawjedregoctclndqzioadurtnlslwvuyjeowzedlvypsudcuerdzvdpsvjfecouyavnyyemivgrcyxxjsjcmkejve
    "C_DELIVERY_CNT": 0,
    "C_DISCOUNT": 0.3866,
    "C_D_ID": 10,
    "C_FIRST": "ujmduarngl",
    "C_ID": 1938,
    "C_LAST": "PRESEINGBAR",
    "C_MIDDLE": "OE",
    "C_PAYMENT_CNT": 1,
    "C_PHONE": "6347232262068241",
    "C_SINCE": "2015-03-22 00:50:42.822518",
    "C_STATE": "ta",
    "C_STREET_1": "deilobyrynukri",
    "C_STREET_2": "goziejuaqbbwe",
    "C_W_ID": 1,
    "C_YTD_PAYMENT": 10,
    "C_ZIP": "316011111"
  },
  "PKID": "1101938"
}
],

```

ITEM

```
select meta(ITEM).id as PKID, * from ITEM limit 1;
```

```

"results": [
  {
    "ITEM": {
      "I_DATA": "dmnjrkhncnrujbtkrirbddknxuxiyfabopmhx",
      "I_ID": 10425,
      "I_ITEM_ID": 1013
    }
  }
],

```

Next →

Using Jil for custom JSON S...

```

},
"PKID": "10425"

```

```
ORDER_LINES
```

```
}
```

```
],
```

```
ORDERS
```

```
select meta(ORDERS).id as PKID, * from ORDERS limit 1;
```

```
“results”: [
```

```
{
```

```
“ORDERS”: {
```

```
“O_ALL_LOCAL”: 1,
```

```
“O_CARRIER_ID”: 2,
```

```
“O_C_ID”: 574,
```

```
“O_D_ID”: 10,
```

```
“O_ENTRY_D”: “2015-03-22 00:50:44.748030”,
```

```
“O_ID”: 1244,
```

```
“O_OL_CNT”: 12,
```

```
“O_W_ID”: 1
```

```
},
```

```
“PKID”: “1101244”
```

```
}
```

```
],
```

```
cbq> select meta(ORDER_LINE).id as PKID, * from ORDER_LINE limit 1;
```

```
“results”: [
```

```
{
```

```
“ORDER_LINE”: {
```

```
“OL_AMOUNT”: 0,
```

```
“OL_DELIVERY_D”: “2015-03-22 00:50:44.836776”,
```

```
“OL_DIST_INFO”: “oiukbnbcazonubtqziuvcdi”,
```

```
“OL_D_ID”: 10,
```

```
“OL_I_ID”: 23522,
```

```
“OL_NUMBER”: 3,
```

```
“OL_O_ID”: 1389,
```

```
“OL_QUANTITY”: 5,
```

```
“OL_SUPPLY_W_ID”: 1,
```

```
“OL_W_ID”: 1
```

```
},
```

```
“PKID”: “11013893”
```

Next →

Using Jil for custom JSON S...

Posted in: Uncategorized

Tagged in: couchbase sql nosql sql for documents n1ql optimizer index index selection

Posted by Keshav Murthy

Keshav Murthy is a Vice President at Couchbase R&D. Previously, he was at MapR, IBM, Informix, Sybase, with more than 20 years of experience in database design & development. He lead the SQL and NoSQL R&D team at IBM Informix. He has received two President's Club awards at Couchbase, two Outstanding Technical Achievement Awards at IBM. Keshav has a bachelors degree in Computer Science and Engineering from the University of Mysore, India, holds eight US patents.

[All Posts](#) [Website](#)



3 Comments

Srikrishna

MARCH 6, 2016 AT 6:06 AM (EDIT)



how can we create GSI on STRING AND REXEXP FUNCATIONS LIKE
CONTAINS,REGEXP_CONTAINS

[Reply](#)

atom992

[Next](#) →

[Using Jil for custom JSON S...](#)

How to understand \"fetch and filter applications are parallelized within the query execution.\"

I think filter should execute after fetch data?

Reply

What is Couchbase Server? How it works? - DevOps Rider

JULY 1, 2019 AT 11:07 AM (EDIT)



[...] more about how couchbase works – <https://blog.couchbase.com/sql-for-documents-n1ql-brief-introduction-to-query-planning/> [...]

Reply

Leave a reply

Comment

☐

Notify me of follow-up comments by email.

☐

Notify me of new posts by email.

Post Comment

Next →

Using Jil for custom JSON S...

Search here...