



Sistemas aKública



# THE GREAT PERFORMANCE OF PROGRESS OPENEDGE 12 WITH SSJ (SERVER-SIDE JOIN)



## THE GREAT PERFORMANCE OF OPENEDGE 12 WITH SERVER-SIDE JOIN

### Are you aware of the new Server-Side Join (SSJ) introduced in OpenEdge version 12.0?

The SSJ functionality was introduced in OpenEdge 12.0 with the FOR statement, now with 12.1 you can use it with dynamic FORWARD-ONLY, NO-LOCK queries. In this tutorial we will show you how to implement this functionality.

The multi-threaded Database Server of OpenEdge 12 processes remote client requests at the same time, improving DB performance significantly, without requiring changes to application coding. For better performing queries OpenEdge 12 performs Server-Side Joins with FOR-EACH queries that join multiple tables. These queries are now resolved server-side rather than client-side, and require no code changes.

SSJ supports up to 10 tables joined in the same logical DB. When a query cannot be resolved on the server side it continues to work as usual.

Applications using an n-tier architecture connect to the DB via network parameters. In this scenario the client can be a GUI, TTY, or a PASOE process agent.

Finally, SSJ processing improves performance by resolving queries on the server and reducing data sent over the network, and is default for joins that meet all of the following conditions:

1. Joins in a client / server environment.
2. Bind using NO-LOCK with the FOR statement, or the FORWARD-ONLY dynamic query.
3. Joins up to 10 tables in the same logical DB.

It is implemented as follows:

### PROSERVE PARAMETERS

The SSJ functionality is controlled by the DB parameter -ssj. By default the database uses -ssj 1. The use of SSJ can be disabled by specifying -ssj 0 as the startup parameter when starting the broker with PROSERVE. MTS (multi-threaded server) is also enabled from the start.

| Parameters                  | Server-Side Join   |
|-----------------------------|--|
| -threadedServer 1<br>-ssj 1 | Enabled (default)<br>Multi-threaded server (MTS) with server-side join (SSJ) |
| -threadedServer 1<br>-ssj 0 | Disabled   |
| -threadedServer 0<br>-ssj 1 | -ssj 1 is ignored because the multi-threaded server is not enabled.          |
| -threadedServer 0<br>-ssj 0 | Disabled   |

## SSJ PROCESSING WITH A FOR STATEMENT

This program shows a join using the FOR statement. The connection to the DB is made via network parameters.



```
1 etime(yes).
2 output to report.txt.
3
4=for each customer no-lock,
5     each order no-lock
6         where order.custnum = customer.custnum
7             and promisedate = 05/28/2018,
8     each orderline no-lock
9         where orderline.ordernum = order.ordernum:
10    put customer.custnum format ">>>>9" customer.name skip.
11 end.
12
13 output close.
14 display etime.
15
16 pause.
17 quit.
18 |
```

These are the steps to run an example program using SSJ on a system with OpenEdge 12.1 installed:

1. prodb sports2020 sports2020
2. proserve sports2020 -S 20000
3. mpro sports2020 -S 20000 -p SSJ1.p

The connection to the database is done using the network parameter "-S 20000". The client / server connection is established. The SSJ1 .p program runs and displays the elapsed time for the query.

### Notes:

- The performance depends on the data and the reduction of the data sent over the network.
- There are no changes to the index selection rules when you run the SSJ functionality.
- If needed, you can specify USE-INDEX to ensure that a certain index is used for optimal query resolution.

## PERFORMANCE TEST DISABLING SSJ

To compare the execution speed of the program by disabling SSJ you can do the following:

1. `prodb sports2020 sports2020 -ssj 0`
2. `proserve sports2020 -S 20000`
3. `mpro sports2020 -S 20000 -p SSJ1.p`

The difference with the previous steps is that the option “-ssj 0” disables SSJ.

When you run the performance test, you will need to close and restart the DB broker so that the query enhancements are not associated with the caching of blocks or records.

## SAMPLE CODE USING FORWARD-ONLY DYNAMIC QUERIES

```
SSJ2.p
1 Define variable qh as widget-handle no-undo.
2 define variable lSuccess as logical no-undo.
3
4 etime(yes).
5 output to report.txt.
6
7 create query qh.
8
9 qh:forward-only = true.
10 qh:set-buffers(buffer customer:handle,
11               buffer order:handle,
12               buffer orderline:handle).
13
14 qh:query-prepare(
15     "for each customer no-lock," +
16     "  each order no-lock " +
17     "    where order.custnum = customer.custnum" +
18     "    and promisedate = 05/28/2018," +
19     "  each orderline no-lock " +
20     "    where orderline.ordernum = order.ordernum"
21 ).
22 qh:query-open.
23
24 repeat while not qh:query-off-end:
25     lSuccess = qh:get-next().
26     if lSuccess then
27     do:
28         put customer.custnum format ">>>>>9"
29         order.ordernum " "
30         orderline.linenum skip.
31     end.
32 end.
33
34 qh:query-close().
35 delete object qh.
36
37 output close.
38 display proversion etime.
39 pause.
40 quit.
41
```

1. prodb sports2020 sports2020
2. proserve sports2020 -S 20000
3. mpro sports2020 -S 20000 -p SSJ2.p

This example program uses FORWARD-ONLY Dynamic query with NO-LOCK.



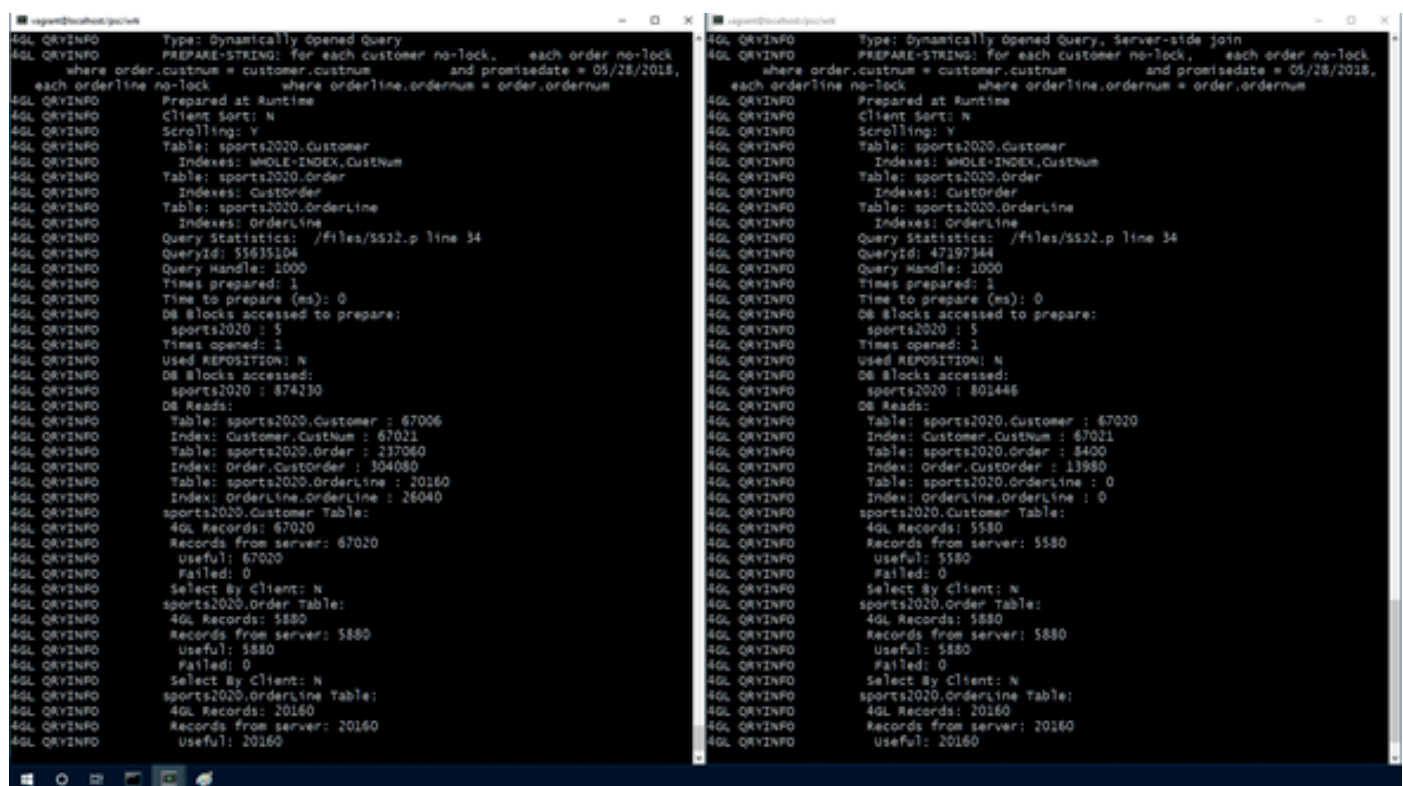
## QRYINFO LOGGING

You can analyze the execution of a query using QryInfo logging option without SSJ, as follows:

1. proserve sports2020 -S 20000 -ssj 0
2. mpro sports2020 -S 20000 -p SSJ2.p \
  - clientlog client.log -logentrytypes QryInfo -logginglevel 3

To run the example using QryInfo logging with SSJ:

1. proserve sports2020 -S 20000
2. mpro sports2020 -S 20000 -p SSJ2.p \
  - clientlog client.log -logentrytypes QryInfo -logginglevel 3



```

4QL QRYINFO Type: Dynamically Opened Query
4QL QRYINFO PREPARE-STRING: for each customer no-lock, each order no-lock
4QL QRYINFO where order.custnum = customer.custnum and promisedate = 05/28/2018,
4QL QRYINFO each orderline no-lock where orderline.ordernum = order.ordernum
4QL QRYINFO Prepared at Runtime
4QL QRYINFO Client Sort: N
4QL QRYINFO Scrolling: Y
4QL QRYINFO Table: sports2020.Customer
4QL QRYINFO Indexes: WHOLE-INDEX,CustNum
4QL QRYINFO Table: sports2020.Order
4QL QRYINFO Indexes: CustOrder
4QL QRYINFO Table: sports2020.OrderLine
4QL QRYINFO Indexes: OrderLine
4QL QRYINFO Query Statistics: /files/SSJ2.p line 34
4QL QRYINFO QueryId: 55635104
4QL QRYINFO Query Handle: 1000
4QL QRYINFO Times prepared: 1
4QL QRYINFO Time to prepare (ms): 0
4QL QRYINFO DB Blocks accessed to prepare:
4QL QRYINFO sports2020 : 5
4QL QRYINFO Times opened: 1
4QL QRYINFO Used REPOSITION: N
4QL QRYINFO DB Blocks accessed:
4QL QRYINFO sports2020 : 874230
4QL QRYINFO DB Reads:
4QL QRYINFO Table: sports2020.Customer : 67006
4QL QRYINFO Index: Customer.CustNum : 67021
4QL QRYINFO Table: sports2020.Order : 237060
4QL QRYINFO Index: Order.CustOrder : 304080
4QL QRYINFO Table: sports2020.OrderLine : 20160
4QL QRYINFO Index: OrderLine.OrderLine : 20400
4QL QRYINFO sports2020.Customer Table:
4QL QRYINFO 4QL Records: 67020
4QL QRYINFO Records from server: 67020
4QL QRYINFO Useful: 67020
4QL QRYINFO Failed: 0
4QL QRYINFO Select By Client: N
4QL QRYINFO sports2020.Order Table:
4QL QRYINFO 4QL Records: 5880
4QL QRYINFO Records from server: 5880
4QL QRYINFO Useful: 5880
4QL QRYINFO Failed: 0
4QL QRYINFO Select By Client: N
4QL QRYINFO sports2020.OrderLine Table:
4QL QRYINFO 4QL Records: 20160
4QL QRYINFO Records from server: 20160
4QL QRYINFO Useful: 20160
  
```

```

4QL QRYINFO Type: Dynamically Opened Query, Server-side join
4QL QRYINFO PREPARE-STRING: for each customer no-lock, each order no-lock
4QL QRYINFO where order.custnum = customer.custnum and promisedate = 05/28/2018,
4QL QRYINFO each orderline no-lock where orderline.ordernum = order.ordernum
4QL QRYINFO Prepared at Runtime
4QL QRYINFO Client Sort: N
4QL QRYINFO Scrolling: Y
4QL QRYINFO Table: sports2020.Customer
4QL QRYINFO Indexes: WHOLE-INDEX,CustNum
4QL QRYINFO Table: sports2020.Order
4QL QRYINFO Indexes: CustOrder
4QL QRYINFO Table: sports2020.OrderLine
4QL QRYINFO Indexes: OrderLine
4QL QRYINFO Query Statistics: /files/SSJ2.p line 34
4QL QRYINFO QueryId: 47197344
4QL QRYINFO Query Handle: 1000
4QL QRYINFO Times prepared: 1
4QL QRYINFO Time to prepare (ms): 0
4QL QRYINFO DB Blocks accessed to prepare:
4QL QRYINFO sports2020 : 5
4QL QRYINFO Times opened: 1
4QL QRYINFO Used REPOSITION: N
4QL QRYINFO DB Blocks accessed:
4QL QRYINFO sports2020 : 801446
4QL QRYINFO DB Reads:
4QL QRYINFO Table: sports2020.Customer : 67020
4QL QRYINFO Index: Customer.CustNum : 67021
4QL QRYINFO Table: sports2020.Order : 8400
4QL QRYINFO Index: Order.CustOrder : 13980
4QL QRYINFO Table: sports2020.OrderLine : 0
4QL QRYINFO Index: OrderLine.OrderLine : 0
4QL QRYINFO sports2020.Customer Table:
4QL QRYINFO 4QL Records: 5580
4QL QRYINFO Records from server: 5580
4QL QRYINFO Useful: 5580
4QL QRYINFO Failed: 0
4QL QRYINFO Select By Client: N
4QL QRYINFO sports2020.Order Table:
4QL QRYINFO 4QL Records: 5880
4QL QRYINFO Records from server: 5880
4QL QRYINFO Useful: 5880
4QL QRYINFO Failed: 0
4QL QRYINFO Select By Client: N
4QL QRYINFO sports2020.OrderLine Table:
4QL QRYINFO 4QL Records: 20160
4QL QRYINFO Records from server: 20160
4QL QRYINFO Useful: 20160
  
```

In this screenshot you can see the log using QryInfo, the left side shows the result with the SSJ option disabled, and the right side shows SSJ enabled (default).

## CONCLUSION

The SSJ processing functionality in OpenEdge 12.x delivers a huge improvement out-of-the-box performance.

### To remember:

- SSJ is enabled by default.
- SSJ is available with the FOR statement and FORWARD-ONLY dynamic queries, NO-LOCK.
- Actual performance gains depend on the data being queried and the reduction of the records sent over the network.
- The QryInfo logging option can be used to analyze the query execution.

# ¿ESTÁS LISTO PARA DAR EL SIGUIENTE PASO?

**Consulta con nosotros todas las posibilidades de  
hacer crecer la innovación en tu empresa**

**Alejandro López**  
81 1077 0559  
alopez@akubica.com

**Arantza Guajardo**  
81 1965 5289  
aguajardo@akubica.com

**contacto@akubica.com**  
**www.akubica.com**

 /sistemasakubica

 company/sistemas-akública