InterSystems
Health I Business I Government

Optimizing Your SQL Queries

Steve Mallam
Sales Engineer

Agenda

- Planning for good performance
- Analyzing query performance
  - How do you know what your application is doing?
  - How can you tell which bits are, or **will be,** “slow”?
- What can you do to improve performance?

Planning for good performance

- “Volume breaks stuff!”
- Test performance early and often
- Realistic test data
  - Size
  - Quality (or lack thereof…)
- What does “good” look like?
- How do we *know* it’s good enough?

Analyzing Query Performance

What is my query *doing*?!

Execute | Show Plan | Show History | Query Builder | Display Mode | Max 1000 more

```
SELECT StockTrade.BrokerId, Broker.Name, Broker.Office, count(*) as BuyCount
FROM StockTrade, Broker
WHERE StockTrade.TransactionType = 'buy'
  AND Broker.ID = StockTrade.BrokerID
GROUP BY StockTrade.BrokerId
ORDER BY BuyCount desc
```

Row count: **20** Performance: **15.457** seconds **24995989** global references **184970047** lines executed **0** disk read latency (ms) Cached Query: %sqlcq.IRIS.cls2 Last update: 2018-09-18 11:44:43.869

BrokerId	Name	Office	BuyCount
103	Jung,Brendan K.	Fargo	251291
106	Gibbs,Rob I.	Boston	250787
104	Eno,Milhouse W.	Boston	250500
115	Jones,Laura V.	San Antonio	250318
110	Uberoth,Barbara W.	Fargo	250308
108	Xerxes,Quentin L.	Boston	250063
116	Jaynes,Rob X.	Boston	250046
119	Quine,Christen S.	Fargo	250033
101	Umansky,Charles G.	San Antonio	250008
114	Xavier,Terry I.	San Antonio	249945

 |

What is my query *doing*?!

The execution plan is displayed below:

Statement Text

```
SELECT StockTrade . BrokerId , Broker . Name , Broker . Office , COUNT ( * ) AS BuyCount FROM StockTrade , Broker WHERE StockTrade . TransactionType = ?
AND Broker . ID = StockTrade . BrokerID GROUP BY StockTrade . BrokerId ORDER BY BuyCount DESC
```

Query Plan

Relative Cost = 139931400

- Call module B, which populates temp-file A.
- Call module D, which populates temp-file B.
- Read temp-file B, looping on count(rows) and a counter.
- For each row:
  - Output the row.

Module: B

- Read bitmap index SQLUser.StockTrade.TransactionTypeIndex, using the given %SQLUPPER(TransactionType), and looping on ID.
- For each row:
  - Read master map SQLUser.StockTrade.IDKEY, using the given idkey value.
  - Read master map SQLUser.Broker.IDKEY, using the given idkey value.
  - Check distinct values for BrokerId using temp-file A, subscripted by a hashing of these values.
  - For each distinct row:
    - Add a row to temp-file A, subscripted by the hashing, with node data of BrokerId, Name, and Office.
  - Update the accumulated count(rows) in temp-file A, subscripted by the hashing.

- Read master map SQLUser.StockTrade.IDKEY, looping on ID.
- For each row:
  - Read master map SQLUser.Broker.IDKEY, using the given idkey value.

Module: D

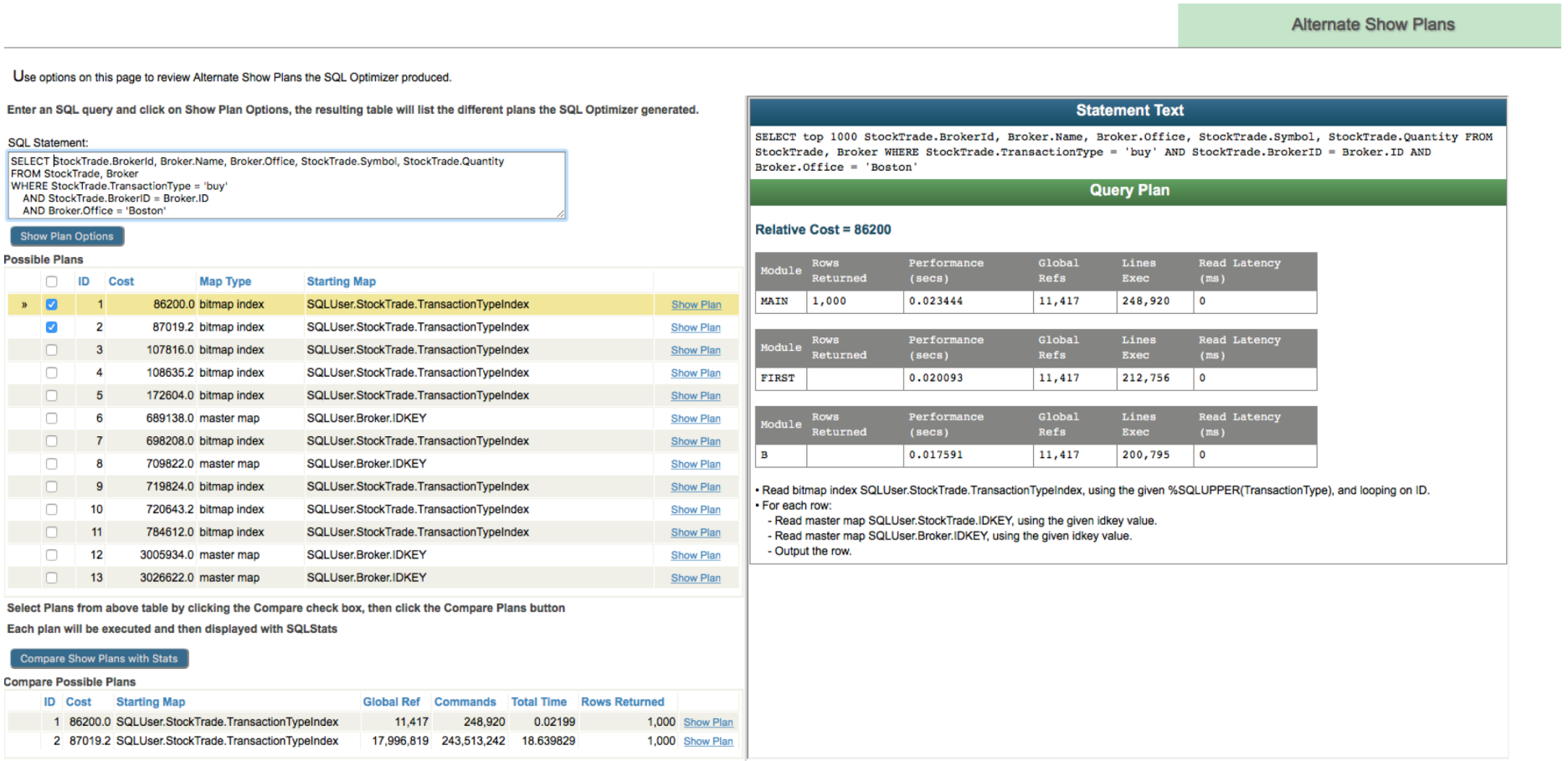
- Read temp-file A, looping on the hashing subscript.
- For each row:
  - Add a row to temp-file B, subscripted by count(rows) and a counter, with node data of BrokerId, Name, and Office.

Note: Don’t HAVE to execute the query before viewing the plan!

 |

Comparing Query Plans

System Explorer -> Tools -> SQL Performance Tools -> Alternate Show Plans



Alternate Show Plans

Use options on this page to review Alternate Show Plans the SQL Optimizer produced.

Enter an SQL query and click on Show Plan Options, the resulting table will list the different plans the SQL Optimizer generated.

SQL Statement:

```
SELECT StockTrade.BrokerId, Broker.Name, Broker.Office, StockTrade.Symbol, StockTrade.Quantity
FROM StockTrade, Broker
WHERE StockTrade.TransactionType = 'buy'
   AND StockTrade.BrokerID = Broker.ID
   AND Broker.Office = 'Boston'
```

Show Plan Options

Possible Plans

		ID	Cost	Map Type	Starting Map	
»	☑	1	86200.0	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☑	2	87019.2	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	3	107816.0	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	4	108635.2	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	5	172604.0	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	6	689138.0	master map	SQLUser.Broker.IDKEY	Show Plan
	☐	7	698208.0	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	8	709822.0	master map	SQLUser.Broker.IDKEY	Show Plan
	☐	9	719824.0	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	10	720643.2	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	11	784612.0	bitmap index	SQLUser.StockTrade.TransactionTypeIndex	Show Plan
	☐	12	3005934.0	master map	SQLUser.Broker.IDKEY	Show Plan
	☐	13	3026622.0	master map	SQLUser.Broker.IDKEY	Show Plan

Select Plans from above table by clicking the Compare check box, then click the Compare Plans button

Each plan will be executed and then displayed with SQLStats

Compare Show Plans with Stats

Compare Possible Plans

	ID	Cost	Starting Map	Global Ref	Commands	Total Time	Rows Returned	
	1	86200.0	SQLUser.StockTrade.TransactionTypeIndex	11,417	248,920	0.02199	1,000	Show Plan
	2	87019.2	SQLUser.StockTrade.TransactionTypeIndex	17,996,819	243,513,242	18.639829	1,000	Show Plan

Statement Text

```
SELECT top 1000 StockTrade.BrokerId, Broker.Name, Broker.Office, StockTrade.Symbol, StockTrade.Quantity FROM
StockTrade, Broker WHERE StockTrade.TransactionType = 'buy' AND StockTrade.BrokerID = Broker.ID AND
Broker.Office = 'Boston'
```

Query Plan

Relative Cost = 86200

Module	Rows Returned	Performance (secs)	Global Refs	Lines Exec	Read Latency (ms)
MAIN	1,000	0.023444	11,417	248,920	0

Module	Rows Returned	Performance (secs)	Global Refs	Lines Exec	Read Latency (ms)
FIRST		0.020093	11,417	212,756	0

Module	Rows Returned	Performance (secs)	Global Refs	Lines Exec	Read Latency (ms)
B		0.017591	11,417	200,795	0

- Read bitmap index SQLUser.StockTrade.TransactionTypeIndex, using the given %SQLUPPER(TransactionType), and looping on ID.
- For each row:
  - Read master map SQLUser.StockTrade.IDKEY, using the given idkey value.
  - Read master map SQLUser.Broker.IDKEY, using the given idkey value.
  - Output the row.

 |

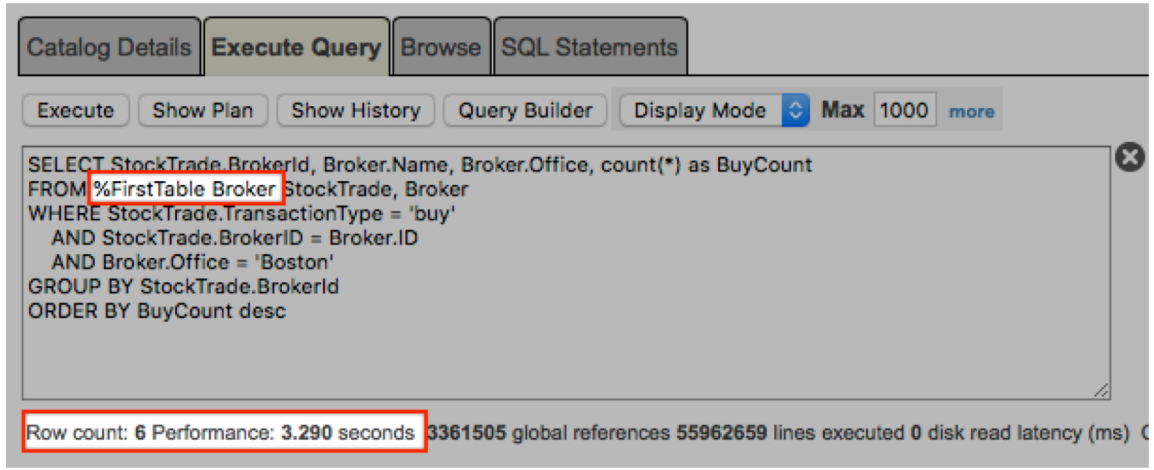
Improving Query Performance

Before we start…

- TuneTables
  - We'll come back to this…
  - …but I have colleagues who would be very upset if I didn't list this first!

- Freeze Plans
  - "Taking Control of Your SQL Performance" by Kyle Baxter
  - Search for "Freeze Plans" on learning.intersystems.com

Fixing the join order



- %FirstTable
- %StartTable
- %InOrder

Can’t the optimizer figure that out?

It can! (If you let it…)

- *ExtentSize*: row count for each table used within the query
- *Selectivity*: percentage of distinct values for each column used by the query
- *BlockCount*: count for each SQL map used by the query
- *Outlier Selectivity*: when a single property value is much more frequent than others
  - *e.g. “Home State” in a local Doctor’s office, or “Closed” in an issue tracking system*

Can be set manually, or with **TuneTables**

Adding an index to improve a query

```
Class User.Broker Extends User.Person [ DdlAllowed ]
{

Property Salary As %Float(POPSPEC = "Float(50000,250000,2)");

Property Office As %String(POPSPEC = "##class(User.Broker).Office()");

Index OfficeIDX on Office;

/// Return a random city name.
ClassMethod Office() As %String
{
 set t1=$lb("Boston","Chicago","New York","Fargo","San Antonio")
```

- Create the Index in Atelier / Studio or with DDL (if [DdlAllowed])
  - CREATE INDEX OfficeIDX ON SQLUser.Broker(Office)
- Rebuild Indices through Management Portal
  - Or ##class(User.Broker).%BuildIndices()
- Considerations for live systems
  - $SYSTEM.SQL.SetMapSelectability
  - Purge Cached Queries

 |

Indexing multiple fields

Compound Index

- Index SuperFastIDX on (TransactionType, Symbol) [Data = Quantity];
- Fastest / Inflexible

Multi Index

- Index TransactionTypeIDX on TransactionType [Type = bitmap];
- Index SymbolIDX on Symbol [Type = bitmap];
- Fast (not quite AS fast) but more flexible

Indexing multiple fields

```
SELECT * FROM SQLUser.StockTrade
WHERE Symbol = 'FFLM'
  AND TransactionType = 'buy'
```

The execution plan is displayed below:

Statement Text

```
SELECT * FROM SQLUser . StockTrade WHERE Symbol = ? AND TransactionType = ?
```

Query Plan

Relative Cost = 256500400

- Read bitmap index SQLUser.StockTrade.TransactionTypeIndex, using the given %SQLUPPER(TransactionType), and looping on ID.
- For each row:
  - Read master map SQLUser.StockTrade.IDKEY, using the given idkey value.
  - Output the row.

Row count: **443** Performance: **6.498** seconds **4998741** global references **80024718**

```
Index SymbolIndex on Symbol [ type = bitmap];
```

The execution plan is displayed below:

Statement Text

```
SELECT * FROM SQLUser . StockTrade WHERE Symbol = ? AND TransactionType = ?
```

Query Plan

Relative Cost = 261399

- Generate a stream of idkey values using the multi-index combination:
  ((bitmap index SQLUser.StockTrade.SymbolIndex) INTERSECT (bitmap index SQLUser.StockTrade.TransactionTypeIndex))
- For each idkey value:
  - Read master map SQLUser.StockTrade.IDKEY, using the given idkey value.
  - Output the row.

Row count: **443** Performance: **0.013** seconds **764** global references **56981**

But don't "SELECT *"!!!!

When should I use the different Index types?

- "Standard" Index
  - Flexible
  - No restriction on values
  - Can store additional data
- Bitmap Index
  - Highly compressed and optimized
  - IDs must be positive integer
  - Ideal for data with well defined set of distinct values (e.g. "State")
    - Use when you have < 10,000 distinct values
- Bitslice Index
  - Store values as bit strings
  - Very fast aggregate calculations
  - Very expensive!

Dealing with ”outliers” in your application

Run Time Plan Choice (RTPC)

- System-wide SQL configuration option
- Detects whether a query contains conditions on a field with an outlier
- Defers optimization until run-time
  - Selects the best plan for the query at that time
- Applies to Dynamic SQL and cursor-based Embedded SQL

For more details, refer to documentation

Summary

What have we talked about?

- The need to plan for and test performance
- Reading Query Plans
  - Table scan v. Index
- Cached Queries
- SQL Statements
- SQL Runtime Statistics
  - Gathering SQLStats data
- Alternate Show Plans
- %Parallel
- %FirstTable, %StartTable, %InOrder
- Tuning Statistics
- TuneTable
- Index Analyzer
- Adding an Index
  - $SYSTEM.SQL.SetMapSelectability
- Index with Data
- Complex Indices
- Index Types
- Run-Time Plan Choice

More on SQL & Scalability @ Global Summit

Other sessions at the Solution Developer Conference

- InterSystems IRIS Roadmap – Monday 3PM
- Optimizing your SQL Queries – Monday 3PM - Wednesday 1:30PM
- Getting Sharded with InterSystems IRIS – Monday 4PM - Wednesday 1:30PM
- An Outlook on Scaling Out – Monday 5PM
- InterSystems IRIS Reference Architectures – Tuesday 4PM
- Multi-Model Development – Wednesday 12PM

And even more at the Tech Exchange, on http://learning.intersystems.com and http://community.intersystems.com/

 |

Questions?

Please complete the survey in the event app or text “summit” to 878787

Session recording and slides will be available at:
Learning.InterSystems.com
Search for “Global Summit 2018”

Visit the Tech Exchange to meet with a subject expert