You'll make breakthroughs

INTERSYSTEMS®

Optimizing SQL Performance (2015)

Brendan Bannon – Support Manager

3 most important things in real estate?

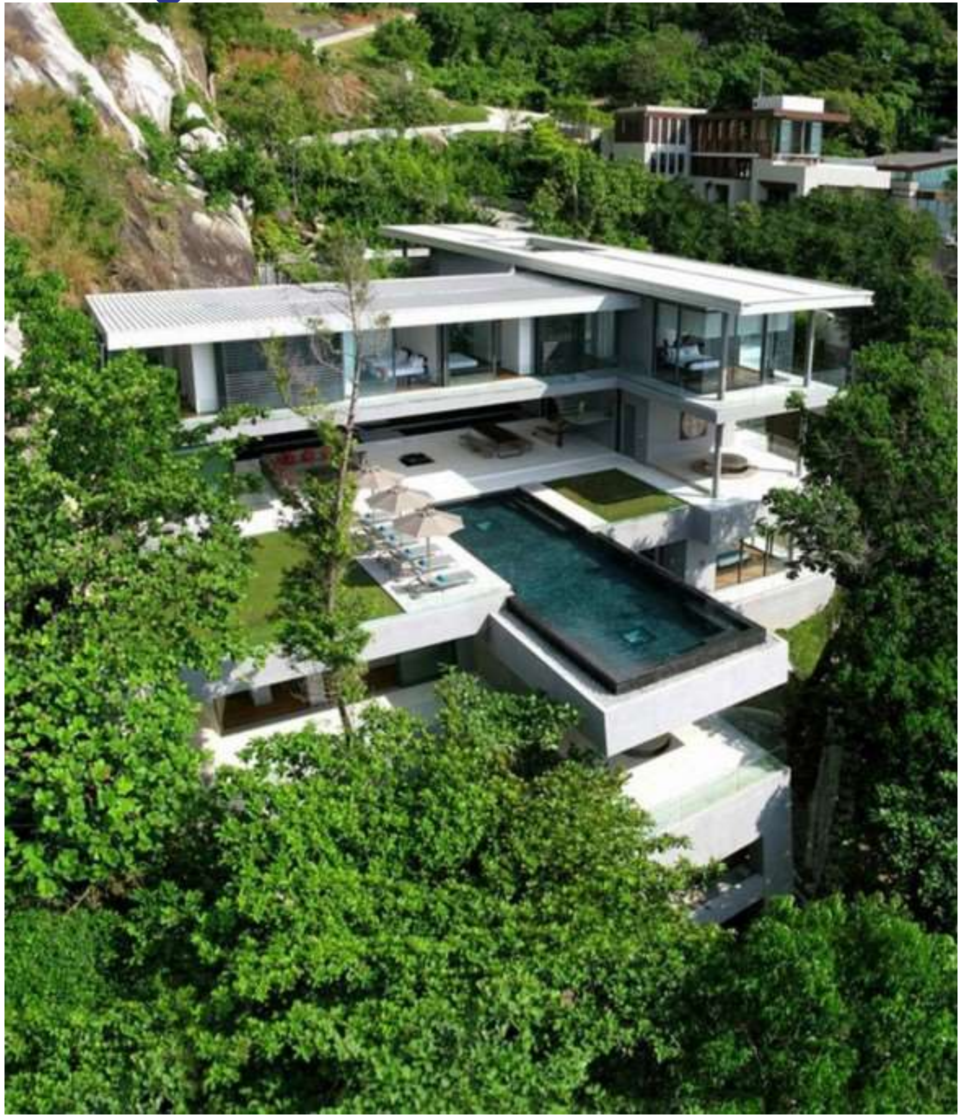
- ?
- ?
- ?

3 most important things in real estate?

- Location

3 most important things in real estate?

- Location
- Location



3 most important things in real estate?

- Location
- Location
- Location

Three most important things for optimizing Cache SQL?

- **TuneTable**
- **TuneTable**
- TuneTable

TuneTable - ExtentSize

- The *ExtentSize* value for a table is simply the number of rows (roughly) stored within the table.
- Used by query Optimizer to determine order of tables.
- Larger ExtentSize means larger Query Cost.

TuneTable - Selectivity

- The *Selectivity* value for a column is the percentage of rows within a table that would be returned as a result of query searching for a typical value of the column.
- Used by the Query Optimizer to help select what indices should be used.
- 1 is the best selectivity value, fields defined a Unique.
- All other field Selectivities are expressed as a percentage, the lower the percentage the better.

TuneTable

New Feature - Outlier Selectivity

- When calculating Selectivity for a property we now keep track of how often different values come up. If one value is statistically very different from the rest it is an outlier. For example, in this app 99.5747% of all problems are closed so that value would be an outlier for Problem Status. This means the optimizer might generate a different plan for:

WHERE Status = ('Closed')

As apposed to:

WHERE Status = 'Open'

TuneTable

New Feature - Outlier Selectivity

From the Storage of User.Problem

```
<Property name="Status">
   <OutlierSelectivity>.995747:"Closed"</OutlierSelectivity>
   <Selectivity>0.2118%</Selectivity>
   </Property>
```

TuneTable

New Feature - Outlier Selectivity

SELECT * FROM SQLUser.Problem WHERE Status = ('Closed')

cost value=31109040

```
Read master map SQLUser.Problem.IDIndex, looping on ID.
For each row:
    Output the row.
```

TuneTable

New Feature - Outlier Selectivity

SELECT * FROM SQLUser.Problem WHERE Status = 'Open'

cost value=248606

```
Call module B, which populates temp-file A.

Read temp-file A, looping on ID.

For each row:

    Read master map SQLUser.Problem.IDIndex, using the given idkey value.

    Output the row.

Module B

Read index map SQLUser.Problem.Work, using the given %SQLUPPER(Status), and
    looping on %SQLUPPER(Priority), ProblemOwner, and ID.

For each row:

    Add a row to temp-file A, subscripted by ID,

        with no node data.
```

TuneTable

New Feature – Block Count

- In addition to calculating ExtentSize and Selectivity TuneTable will now also count the number of blocks a map will take up on disk
- If TuneTable is not run then the class compiler will still put in an estimated value for the Block Count
- The size of all Caché SQL map blocks is 2048 bytes (2K bytes)
- Run on Local Databases

TuneTable

New Feature – Block Count

```
SAMPLES>d $SYSTEM.SQL.TuneTable("GS.Outliers",1,1)
 TABLE: GS.Outliers:
EXTENTSIZE:
        CURRENT =     500000
        CALCULATED =  500001
        SAMPLESIZE    2121          Table & Class Definition Updated.
…
BLOCKCOUNT of MAP IDKEY
        CURRENT =     14728
        MEASURED =    14728
 BLOCKCOUNT of MAP SalaryIndex
        CURRENT =     9667
        MEASURED =    2188            Table & Class updated
```

TuneTable

- Run TuneTable on ALL your tables.
- You only need to run TuneTable once, on a good database.
- Tables will generally grow relative to one another so as the DB grows there is no need to rerun TuneTable
- Test query performance after running TuneTable.
- Ship tables with default ExtentSize and Selectivity.

INTERSYSTEMS®

Indices

Indices

Bitmap - ^My.ClassI("Gender",value,chunk)="bit string"

- Very fast
- Compact – we store 64,000 IDs per global node
- Uses bit math with multiple indices
- Use when you have < 10,000 distinct values
- No performance hit for INSERT, UPDATE or DELETE
- Performance can slow on volatile systems (large number of INSERTs and DELETEs)

InterSystems®

Indices

Bit Extent - ^My.ClassI("$Class",chunk)="bit string"

- Special case of a bitmap
- No sort field
- Just chunks of IDs
- Used for COUNT(*) and some JOINs

Indices

Standard - ^My.ClassI("AccountNum",value,ID)="data"

- Flexible
- No restriction on distinct values
- Can have extra data
- One global node per indexed value
- Will not degrade over time

Indices

```
Bit slice - ^My.ClassI("Salary",Binary 1,chunk)="bit string"
            ^My.ClassI("Salary",Binary 2,chunk)="bit string"
            …
            ^My.ClassI("Salary",Binary n,chunk)="bit string"
```

- Very fast aggregate calculations
- Instead of storing a number we convert to binary and store the bits
- Much more expensive than Bitmaps or Standard indices
- Currently limited usage in Caché SQL

Indices

Multi Index Solutions

- Caché has the ability to use multiple indices from one table to resolve a query.
- For the following query what is the best Index?

```
SELECT Name
FROM SQLUser.Employee
WHERE Deleted = 0
AND NASCLeader = 1
```

Indices

Multi Index Solutions

- The fastest way to resolve this query would be with a Compound Index
- Index SuperFast On (NASCLeader, Deleted) [Data = Name];

Row count: **5** Performance: **0.002** seconds **69** global references

Read index map SQLUser.Employee.SuperFast, using the given NASCLeader and Deleted, and looping on ID.

For each row:

Output the row.

Indices

Multi Index Solutions

- The most flexible solution is to define 2 bitmaps:

```
Index Bitmap1 On NASCLeader [ Type = bitmap ];
Index Bitmap2 On Deleted [ Type = bitmap ];
```

Row count: **5** Performance: **0.002** seconds **71** global references

Generate a stream of idkey values using the multi-index combination:

```
 ((bitmap index SQLUser.Employee.Bitmap1)
 INTERSECT (bitmap index SQLUser.Employee.Bitmap2))
For each idkey value:

 Read master map SQLUser.Employee.Emp, using the given idkey
value.
 Output the row.
```

You'll make breakthroughs

INTERSYSTEMS®

Bitmap Performance

Why is my Bitmap slowing down?

- One global node contains bits for 64,000 rows of a table
- One node takes up to 8K
- When you DELETE 1 row we change 1 bit to 0
- When all the bits in a chunk 0 the node is no longer needed
- When a query is looking for a defined row we need to look at all these zero chunks for a 1
- These zero chunks build up for every distinct value that is part of a Bitmap

Compacting Bitmaps

%SYS.Maint.Bitmap

- New in Cache 2014.2
- This utility can be run on a live system
- Will compact or remove bitmap nodes
- Needs to be run against local databases
- Methods:
  - d ##class(%SYS.Maint.Bitmap).Namespace("Samples",1,1,"2014-01-17 09:00:00")
  - d ##class(%SYS.Maint.Bitmap).OneClass("BitMap.Test",1,1)

Compacting Bitmaps

```
SAMPLES>d ##class(GS.Compact).Populate(1000000)
```

Global - GS.CompactI   Blocks - 34   Bytes - 250,901

1. DELETE FROM GS.Compact WHERE ID < 750000

Global - GS.CompactI   Blocks - 46   Bytes - 344,747

```
SAMPLES>d ##class(%SYS.Maint.Bitmap).OneClass("GS.Compact",1,1)

Class:  GS.Compact Start Time:  2014-03-13 14:45:29
    Global:  ^GS.CompactI("$Compact") was compressed:  93.87 %
    Global:  ^GS.CompactI("GenderIdx") was compressed:  70.48 %
Compression time in seconds:  0
```

Global - GS.CompactI   Blocks - 12   Bytes - 80,066

You'll make breakthroughs

INTERSYSTEMS®

Show Plan

Show Plan – Things we don't like

- Bad

  Read index map DatesRUs.Profile.CustomerPointerIndex, looping on CustomerPointer and ID

- Add a row to temp-file A, subscripted by %SQLUPPER(Name) and ID, with node data of Name.

- The worst

  Read master map DatesRUs.Employee.IDKEY, looping on ID

Show Plan – Things we like

- Good

  Read index map DatesRUs.Customers.NameIndex, using the given %SQLUPPER(Name) and ID

- Better

  (((bitmap index DatesRUs.Profile.EyeI) INTERSECT (bitmap index DatesRUs.Profile.GenderI)) INTERSECT (bitmap index DatesRUs.Profile.ActiveI))

You'll make breakthroughs

INTERSYSTEMS®

Examples

Table Scans

SELECT name FROM DatesRUs . Employee WHERE Active = 1 ORDER BY Name

Time in Module MAIN =	0.029
Module Execution Count =	4
Global References =	82,597
Commands Executed =	83,028
Number of Rows =	3

Read master map DatesRUs.Employee.IDKEY, looping on ID.

For each row:

Add a row to temp-file A, subscripted by %SQLUPPER(Name) and ID,

with node data of Name.

Table Scans – What’s wrong?

- 82,000 global refs to return 3 rows.
- SELECT COUNT(*) from DatesRUs.Employee returns 105 rows.
- What is causing all the extra work?

Table Scans – Problems

- DatesRUs.Employee Extends DatesRUs.Person so all the employees are stored in the same global as Person. (and DatesRUs.Customer)
- No Index on Active.
  - ➢ Standard or Bitmap index?

Table Scans

- New Index

  Index ActiveNames On (Active,Name) [Data = Name];

```
Time in Module MAIN =                0.000
Module Execution Count =                 4
Global References =                     34
Commands Executed =                    306
Number of Rows =                         3
```

```
Read index map DatesRUs.Employee.ActiveNames, using the given
   Active, and looping on %SQLUPPER(Name) and ID.
For each row:
    Output the row.
```

Multi Index Solution

```
SELECT ID, Description
FROM DatesRUs.Profile
WHERE Active=1
and (Gender = :sex or :sex IS NULL)
AND (Eye = :eye OR :eye IS NULL)
AND (Hair = :hair OR :hair IS NULL)
And (Hobbies %INLIST :hobby or :hobby IS NULL)
```

Multi Index Solution

- Only have conditions in the WHERE that the user provided a value, greatly simplifies the query

```
SELECT ID, Description
FROM DatesRUs.Profile
WHERE Active=1 AND Gender = :sex
AND Eye = :eye AND Hair = :hair
AND Hobbies %INLIST :hobby
```

One Compound Index Vs. Several simple Indices

- One Compound with several subscripts.
  - Index Compound On (Active, Hair, Eye, Gender);
  - Fastest option for specific query
  - Least flexible
- Read index map DatesRUs.Profile.Compound, using the given Active, %SQLUPPER(Hair), %SQLUPPER(Eye), and %SQLUPPER(Gender), and looping on ID.

One Compound Index Vs. Several simple Indices

- Individual bitmaps for each field
  - Index GenderIndex On Gender [Type = bitmap];
    Index HairIndex On Hair [Type = bitmap];
    Index EyeIndex On Eye [Type = bitmap];
    Index ActiveIndex On Active [Type = bitmap];
  - Almost as fast for specific query
  - Much more flexible for other queries
- ((((bitmap index DatesRUs.Profile.EyeIndex) INTERSECT (bitmap index DatesRUs.Profile.HairIndex)) INTERSECT (bitmap index DatesRUs.Profile.GenderIndex)) INTERSECT (bitmap index DatesRUs.Profile.ActiveIndex))

Table Joins

```
SELECT C2.Name, S.Comments, S.Rating
FROM DatesRUs.Customers C1
JOIN DatesRUs.Survey S ON C1.ID = S.SurveyOf
JOIN DatesRUs.Customers C2 ON C2.ID = S.SurveyBy
WHERE C1.Name = 'Adam,Alice J.‘
```

```
Time in Module MAIN =              1.951
Module Execution Count =             109
Global References =              153,000
Commands Executed =            3,100,851
Number of Rows =                     108
```

- Read master map DatesRUs.Survey.IDKEY, looping on ID.

Table Joins

- Add index to support JOIN

  Index SurveyOfIndex On SurveyOf;

Time in Module MAIN =	0.061
Module Execution Count =	107
Global References =	859
Commands Executed =	9,384
Number of Rows =	106

- Read index map DatesRUs.Customers.NameIndex, using the given %SQLUPPER(Name), and looping on ID.

You'll make breakthroughs

Location?

Now what about that house?

TuneTable

- Assumes
  - Evenly distributed data
  - All values equally likely

TuneTable - Evenly Distributed Data

```
SELECT Status, COUNT(*)
FROM DatesRUs.Customers
GROUP BY Status
```

Status	Count
Available	27
Closed	450
Dating	17
Married	6

TuneTable – All Values Equally Likely

- The property Status has 4 values: Available, Closed, Dating and Married
- For our application are these Equally likely?
- Status has a Selectivity of 16%, is that reflective of how we will be using it?

TuneTable – Modify Selectivity by Hand

- Change Selectivity from
  - The Portal
  - Studio – View Storage
- Set Property Parameter CalcSelectivity = 0

You'll make breakthroughs

INTERSYSTEMS®

Questions?