Make Applications More Valuable

Introduction to Indexing

Brendan Bannon
Support Manager

Innovations by InterSystems

Agenda

1. The Basics
2. Tune Table
3. Indices
   1. Standard
   2. Bitmap
   3. Extent
   4. Bitslice
   5. BuildValueArray

2. The Basics

- Query speed is almost always constrained by disk I/O speed, not cpu or network
  - – Disk can be 1000 times slower than cpu operation
- Cache SQL tuning and optimizer are based on that
- 95/5 Rule: Worry about the 5% of queries that you use very often and are slow, not the 95%

3. Tune Table : Why?

- TuneTable generates statistics on your tables that are used by the Query Optimizer to pick the best query path:
  - The size of one table compared to another.
    - “ExtentSize”
  - How selective an index is for a given property.
    - “Selectivity”
- Which is better to use when querying
    - … where Patient.Sex=“M” and Doctor.Zip=91521…
  - Sex index of Patients
  - Zip index of Doctors

Tune Table : How?

- How to run TuneTable?
  - – Use $SYSTEM.SQL.TuneTable()
  - – Use TuneTable from the SQL Manager/System Management Portal.
  - – Set Selectivity and ExtentSize manually.

TuneTable : When?

- Having good statistics for table cardinality and column cardinalities is crucial for the Optimizer.
- When/How Often?
  - As soon as you have a stable database design and some representative data
  - When you get your first ‘real’ database
  - If you install at a site with atypical data distributions
  - If you think data ratios have changed a lot
  - Before calling ISC Support !
  - (Not needed) just because DB has grown larger

5. Indices

- Most of ‘Tuning’ is about Indices:
  - Define the right indices
  - Make sure the queries use them correctly
- Indices are used for
  - Fast Access Paths (minimize disk access)
  - Selection criteria (WHERE …)
  - Table JOINs
  - Grouping results (GROUP BY, ORDER BY)

Query Tuning - Indices

- **Why is Index search better?**
  - – Data is sorted in known order
  - – Size of index is smaller than data map
  - – More rows in memory at same time
  - – Less I/O than a table scan of data map
  - – If you can make your query access ONLY the index, it will be very fast

Types of Caché Indices

- Standard
  - Unique
  - IDKey
  - SQL Primary Key
  - Compound
- Bitmap
- Extent
- Bitslice

Standard Index

- In a standard index you can also store additional information as data in the global.
- Example:
  - Property Name as %String;
  - Index NIdx On Name [Data = Name];
- Stored:

```
^User.PI("NIdx"," KRATZ,SAM S.",2)=$LB(,Kratz,Sam S.)
^User.PI("NIdx"," MALKO,ELVIRA E.",3)=$LB(,Malko,Elvira E.)
```

- Used:

```
Select Name from P where Name %Startswith ‘KR’
```

Types of Standard Indexes

- **Unique** – Used to make sure that each row has a unique value for a given field or combination of fields.
- **ID Key** – The field is unique, collation is Exact, and it is add only. This is the value we use to retrieve a row from the disk.
- **SQL Primary Key** – Projected to SQL tools as the Primary key, must be Unique, can be changed on an UPDATE.

Bitmap Index

- Uses a series of bit strings to represent the set of ID Key values that correspond to a given indexed value.
- Does not support additional data storage, there is no place to put it!
- This is what the global looks like for a Bitmap Index:

```
                                                   Id → 0123456789…
^User.PI("BossIdx"," LAROCCA,DANIEL Y.",1)=$BIT( 00100101111101111011)
^User.PI("BossIdx"," LUBBAR,JOHN X.",1)=   $BIT( 01011010000010000100)
```

Bitmap vs. Standard Index

- Bitmap index is NOT slow to update, in fact it can be faster (smaller)
- Bitmap index only if IDKey is positive Integer
- ISC Rule of Thumb: If you have less than 10,000 distinct values you should bitmap. But …
- There are some things Bitmaps are very good for.
  - – SELECT Count….
  - – complex WHERE clause with AND and OR

Bit Extent

- A bit Extent is a special bitmap.
- Instead of a bit string for the different values of a field it has one bit string for the whole table.
- If a bit is 1 the row is defined.
- If the bit is 0 the row was deleted.
- Automatically maintained with Cache Storage if there are any bit-map indices

^User.PI(“$Person",1)=$BIT(01011011111111111101)

[records 2, 5 and 18 have been deleted]

Bitslice Index

- A way to index numbers so that they can be summed or averaged quickly. (SUM, AVE)
- Defined in Studio
  - Index PIdx On Price [Type = bitslice];
- Cache will update. The value is broken down into its binary value and then indexed on each bit of that value.
  - Data:
    - ^User.BSD(1) = {^Yang,Nellie H.^3}
    - ^User.BSD(2) = {^Chadwick,Usha U.^4}
  - Bitslice Index:
    - ^(PIdx,1,1) = 0101110001100000000000000000000...
    - ^(PIdx,2,1) = 0100011010000000000000000000000…
    - ^(PIdx,3,1) = 0010000101000000000000000000000...

Using Bitslice Index

- Bitslice indices should be used to solve very specific problems.
- Slower on INSERT UPDATE and DELETE.
- SQL will use for some queries (more later)
  - Select SUM(Amount) from Orders
- You can use bitslice indices in your Cache Script

BuildValueArray

- Indexing on Lists and Arrays
  - Class

    Property FavoriteColors As list Of %String;

    Index Color On FavoriteColors(ELEMENTS);

    - (Needs Delimited Identifiers)
  - Query

    SELECT Name,FavoriteColors FROM Person
    WHERE FOR SOME %ELEMENT(FavoriteColors)

    (%Key=2 and %Value = 'Red')
  - Returns rows where ‘Red’ is the second color
    - FavoriteColors returns as $lb(value,value,…)
  - Roll-your-own multi-valued index from any Property
    - See **BuildValueArray**

Compound Index

- When defining an index you can base it on one or more fields.
- Order is important.

```
Index NameSexIndex On (Name, Sex);
```

Is not the same as:

```
Index SexNameIndex On (Sex, Name);
```

Compound Indices

- If you have a query where the only restriction for a table is the AND of two or more ranges.
- The order of the columns in the index can be important.
  - Example: Two date columns d1 and d2 with conditions d1 < ? And d2 > ? .
    - Both parameters are recent dates.
    - Compound Index on (d1,d2), the d1 condition will read most of the index while the d2 condition will only read a small amount.
    - It would be much better to have the index be (d2,d1) rather than (d1,d2)

Multi Index Solution

- Caché supports the use of 2 indices to resolve one query.
  - This reduces the need for a Compound index.
  - Two indices, one on Name and one on Gender will yield results almost as fast as a Compound index on Name and Gender
  - Two single indices are more flexible than one Compound index : reusable.
- For both Standard and Bitmap indices
- Can often replace Compound Indices with multiple simple indices, standard or (better) bitmap

Indices – How Many ?

- As many as you would like
  - (but not more than you need)
- Don’t be afraid of indices, Cache loves them