Advanced software technologies for breakthrough applications

Caché SQL パフォーマンスチューニング

テクニカルコンサルタント
上中進太郎
インターシステムズジャパン

InterSystems

Agenda

- Caché SQL 概要
  - データ・インデックスグローバルの構造
  - 埋め込みSQL、動的SQLが処理される仕組み
  - クエリのロジック
- SQLパフォーマンスチューニング
  - クエリプラン
  - チューニング
    - TuneTable、Index追加
  - クエリキーワード
- SQLパフォーマンス計測
  - %SYS.PToolsユーティリティ

InterSystems

2

Caché SQL概要

- CachéのSQLはどうやって動いてる？
  - サンプルクラス定義
  - グローバル構造（データ、インデックス）
  - 埋め込みSQLの仕組み
  - 動的SQLの仕組み
  - SQLロジックの基本的な考え方

3

Caché SQL概要

サンプルクラス定義

```
Class Symposia.Person Extends %Persistent
  {

  Property RegID As %Integer;

  Property Name As %String;

  Property CompanyName As %String;

  Index RegIDIndex On RegID [ Unique ];

  Index CoNameIndex On CompanyName [ Type =
  bitmap ];
}
```

4

Caché SQL概要

データグローバル

^Symposia.PersonD=10　← IDを採番する

^Symposia.PersonD(1)=$lb(“”,774,“山田太郎","ロイヤル薬品証券")

^Symposia.PersonD(2)=$lb(“”,894,“鈴木花子”,”サイバー技研興業“)

- データは$List形式で格納されている
  - $Listの構造

    ②Ⓝ17⑫ⓉInterSystems⑤ⓉTokyo

    ↑各データの前にデータ長、データ型がバイナリ形式で格納されている

5

Caché SQL概要

インデックスグローバル

- 標準インデックス

^Symposia.PersonI("RegIDIndex",774,1)=""

^Symposia.PersonI("RegIDIndex",894,2)=""

グローバル名(インデックス名,対象プロパティ1,・・,ID)

=$LB(インデックスデータ)

- ビットマップインデックス

^Symposia.PersonI(“CoNameIndex“,” ロイヤル薬品証券”,1)

=$c(150,1,1,0,1,0)/*$bit(01)*/

グローバル名(インデックス名,対象データ,chunk)

=010000001000. . . . . .

1 chunkに最大64000bit

6

Caché SQL概要　クエリを実行したらどのように処理されているのか？

- 埋め込みSQL、クラスクエリは
  - そのルーチンやクラスにクエリが展開される

SQL1.mac

```
&sql(select count(*) into :cnt from Symposia.Person where CompanyName %STARTSWITH 'ロイヤル')
```

SQL1.int

7

Caché SQL概要　クエリを実行したらどのように処理されているのか？

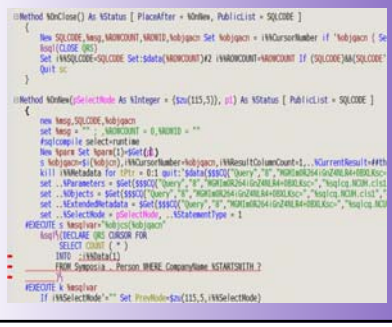
- 動的SQLや外部からのxDBC接続では
  - クエリキャッシュができる
  - クエリキャッシュをクラス等で参照するには SMP→構成→SQL設定→一般SQL設定→クエリキャッシュ-ソース保存 = はい　に変更

xDBC経由のSQLや%ResultSet API の場合
%sqlcq.[NS name].[n].mac
新しい%SQL.Statement APIの場合
%sqlcq.[NS name].cls[n].cls
がクエリキャッシュとして生成される

*使用しているバージョンによって
異なるクエリキャッシュ名となる可能性がある。

```
set sql="select count(*) from Symposia.Person where CompanyName %STARTSWITH ?"
set stmt=##class(%SQL.Statement).%New()
set tSC=stmt.%Prepare(sql)
set rs=stmt.%Execute("ロイヤル")
set tSC=rs.%Next()
set cnt=rs.%GetData(1)
w cnt
```

SQL2mac



8

Caché SQL概要

クエリのロジックを予想してみる

Select RegID from Symposia.Person
where CompanyName = 'AAA' Order by Name

^Symposia.PersonD=93
^Symposia.PersonD(1)=$lb("",774,"山田太郎","ロイヤル薬品証券")
^Symposia.PersonD(2)=$lb("",894,"鈴木花子","サイバー技研興業")
・・・
^Symposia.PersonI("CoNameIndex"," AAA",1)
=$bit(00010000010000001)
^Symposia.PersonI("CoNameIndex"," BBB",1)
=$bit(00001) ・・・

CompanyNameもIndex指定してあるので、
^Symposia.PersonI("CoNameIndex","AAA",n)
を参照して該当するデータグローバルのID、Name取得し、その後ソートのためにNameをキーにテンポラリグローバル
^CacheTemp(Name,id)=RegIDに一度格納してから、最終的にデータを取りだす

9

SQLパフォーマンスチューニング

パフォーマンス調査の第一歩

- いちいちロジックを予想したり、ルーチンを見たりはしていられないので・・・
- クエリプランをみよう！
  - 管理ポータルのSQL→SQL文の実行→クエリプランの表示
    - クエリを実行しなくても表示できる
  - 管理ポータルのSQL→スキーマ→クエリキャッシュ
    - 動的SQL、xDBCクエリのプランが確認できる
  - STUDIOでクエリを選択して右クリック→SQL文に対するプラン表示

10

SQLパフォーマンスチューニング

クエリプランの具体的な見方

- 英語で処理内容を表記
  - 次ページでキーワード紹介
- 相対コストとは
  - 同一クエリ同士で比較できる相対的なコスト。主にインデックス追加前後の効果を測定できる。相対コストが低いプランのほうが良い。
  - 異なるクエリ同士のコスト比較は意味がない。

11

SQLパフォーマンスチューニング

クエリプラン キーワード

- Call module n
  - この表記がある場合は、そのmodule部分を先に呼び出している

実行プランが以下に表示されます:

クエリ文字列

```
SELECT RegID FROM Symposia . Person WHERE CompanyName = ? ORDER BY Name
```

クエリプラン

相対コスト = 91808

- Call module B, which populates temp-file A.
- Read temp-file A, looping on %SQLUPPER(Name) and ID.
- For each row:

  Output the row.

module B

- Read bitmap index Symposia Person CoNameIndex, using the given %SQLUPPER(CompanyName), and looping on ID.
- For each row:

  Read master map Symposia Person IDKEY, using the given idkey value.
  Add a row to temp-file A, subscripted by %SQLUPPER(Name) and ID,
  with node data of RegID.

12

SQLパフォーマンスチューニング

クエリプラン キーワード

キーワード	意味
Read master map	データグローバルを参照
Read index map	インデックスグローバルを参照
using the given yyy	主にクエリのパラメータとして与えられたyyyを使用してインデックス or データ本体の値を取得
looping on xxx	xxxでインデックス or データ本体をループ
with a %STARTSWITH range condition	前方一致条件でループ

13

SQLパフォーマンスチューニング

クエリプラン キーワード

キーワード	意味
Add ID bit to bitmap temp-file A	各モジュールでの検索結果をテンポラリ領域にビットマップ形式で保存
Add a row to temp-file A, subscripted by %SQLSTRING(AAA) and ID,with node data of BBB.	各モジュールの検索結果をテンポラリ領域に サブスクリプト にAAAとIDを配列形式で、データ部にBBBを保存　例： ^temp(AAA,ID)=BBB
Accumulate the max(xxx).	xxxを計算する。Maxの場合は、比較、Sumの場合は足し算など
(((index map INDEXNAME) UNION (bitmap temp-file A)) UNION (bitmap temp-file B))	INDEXあるいはテンポラリ領域の複数の結果をUNION処理

SQLパフォーマンスチューニング

クエリプランで特に注意する点

- **Read master map Symposia.Table1.IDKEY, looping on ID.**
  - これは全データグローバルをループして参照しているということなので、データ件数の多いテーブルの場合はよくない。
  - Read master map Symposia.Table1.IDKEY, using the given idkey value. は問題ない。
- **Add a row to temp-file A**
  - テンポラリグローバルに書いているということなので、件数にもよりますが、あまりよくない。この内容がインデックス追加のヒントになることもあります。

15

SQLパフォーマンスチューニング

クエリチューニング

- まずはTuneTable！
  - 管理ポータル→SQL→SQLスキーマを参照
    - スキーマを表示してスキーマ全体のチューニング
    - テーブルを個別に表示してテーブルのチューニング
  - $SYSTEM.SQL.TuneTable(tablename)
- TuneTableは以下の二つを計測している
  - 選択性（SELECTIVITY）
  - エクステントサイズ（EXTENTSIZE）

16

SQLパフォーマンスチューニング

選択性 (SELECTIVITY)

- 選択性はデータの分散度合いを抽出して計測している
  - N %
  - 1　→　ユニークであることを表している

＊性別のフィールドの選択性は５０%になる。

→実際のデータ分布は関係ない

- 例えばデータ上は男性７０%、女性３０%であったとしても、「男性」、「女性」というデータが抽出されたのであれば、５０%となる。
- ９９%のデータが０の「削除フラグ」でも、削除フラグ＝１のデータが抽出されると５０%となる

→ 管理ポータルから値を指定することも可能

17

SQLパフォーマンスチューニング

エクステントサイズ

- テーブルの総データ件数を計測
- エクステントサイズは複数テーブル間のJOINが発生する場合に、どのテーブルから絞り込みを開始するか、決定するのに使用される。

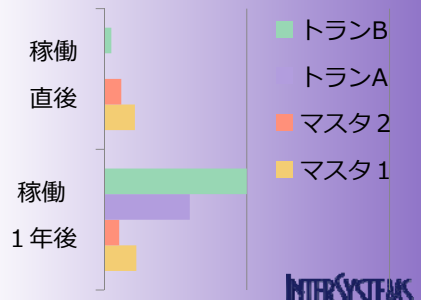
```
SELECT
   count(*)
FROM
   Symposia.Table2,Symposia.Table3,Symposia.Table4
WHERE
   T2Key=T3Key  AND  T2Key=T4Key AND
   T2Value %STARTSWITH 'A' AND
   T3Value %STARTSWITH 'B' AND
   T4Value %STARTSWITH 'C'
```

18

SQLパフォーマンスチューニング

TuneTableはいつ行う？

- システム稼働前に実行し、空のテーブルについては予測値を入れておく。
- 通常は定期的に行う必要はない。パフォーマンスに問題が発生していない場合は特に行う必要がない。
- 各テーブルの選択性やエクステントサイズのテーブル間の比率が大きく変わった場合は効果あり
  - 注意点
    - TuneTable実行中はシステムに負荷がかかります。
    - TuneTable実行直後、関連するテーブルのすべてのクエリキャッシュがパージされます→次回実行時に新たに生成されるため負荷がかかります。



19

SQLパフォーマンスチューニング

インデックスを追加しよう

- どんなインデックスを追加する？
  - Q. 標準インデックス？ビットマップ？
  - A. 対象プロパティの個別値が10000～20000個を超える場合には標準インデックスを使用したほうが良い。
- データの分布を考えてインデックスを選択するとさらに良い。
  - 10万件に1件しかない「削除フラグ＝１」を検出したい→標準
- 条件にAND 検索やOR検索が多い→ビットマップ
- COUNT()クエリを高速化したい→ビットマップ
  - 条件なしCount()の高速化→エクステントビットマップ

20

SQLパフォーマンスチューニング

インデックスを追加しよう

- 対象テーブルにフィールドが多い場合はインデックスデータを活用
- ある特定のクエリに対して専用インデックスを設定して高速化したい→標準インデックス
- クエリプランの Add a row to temp-file X をヒントにしてみる

```
SELECT P1 FROM Symposia.Table1
WHERE P2 = 'V2411' ORDER BY P3,P4
→   Index SpecialIndex On (P2, P3, P4) [Data = P1];
```

21

SQLパフォーマンスチューニング

インデックスを使わせる

- クエリプロセッサがインデックスを使わない方法を選択することがある。
  - レコード件数が少ないので、インデックスを見るよりデータを直接見たほうが早いと判断する
  - 恣意的にインデックスを使わせたい場合は選択性、エクステントサイズなどを変更する
- 条件となるフィールドが関数を使用しているとインデックスを使用できない
  - 例：TimeStamp型フィールド(SalesDate)を日付(2000-01-01)と比較

SELECT * FROM Symposia.Table5 where SalesDate > '2000-01-01 00:00:00' **OK**

SELECT * FROM Symposia.Table5 where {fn CONVERT(SalesDate,SQL_DATE)} > '2000-01-01' **NG**

SELECT * FROM Symposia.Table5 where SalesDate > {fn CONVERT('2000-01-01',SQL_TIMESTAMP)} **OK**

22

SQLパフォーマンスチューニング

インデックスを使わせる

- 最近実際にあったケース
  - Oracle からデータをSQL Gatewayで移行
  - Oracle のDATE型は時間情報まで含むので、Cachéは TIMESTAMP型に変換
  - クエリは 日付をTO_DATE関数で指定しているので、インデックスを使わない！
- 解決策

  案1. 指定をTO_DATEではなくTO_TIMESTAMPに変更

  Where visitdate = TO_DATE('2011/06/03','yyyy/mm/dd')

  Where visitdate = TO_TIMESTAMP('2011/06/03','yyyy/mm/dd')

  案2. 時間情報はいらない場合、移行プログラムを作成してCaché側でもDATE型を利用する

23

SQLパフォーマンスチューニング

ストアド関数をうまく使う

- 複雑な条件をもつクエリをストアド関数を使って簡略化する。
  - SELECT count(*) FROM Symposia.Table2 where T2Value like '%12%' or T2Value like '%23%' or T2Value like '%34%' or T2Value like '%45%' or T2Value like '%56%' or T2Value like '%67%' or T2Value like '%78%' or T2Value like '%89%' or T2Value like '%90%'

条件に合致するストアド関数を用意する　→

SELECT count(*) FROM Symposia.Table2 where Symposia.Table2_MyFunction1(T2Value)=1

* ストアド関数を使う場合、相対コストは余りあてにならない

```
ClassMethod MyFunction1(value)
{
    if value["12" Quit 1
    if value["23" Quit 1
    if value["34" Quit 1
    if value["45" Quit 1
    if value["56" Quit 1
    if value["67" Quit 1
    if value["78" Quit 1
    if value["89" Quit 1
    if value["90" Quit 1

    Quit 0
}
```

24

SQLパフォーマンスチューニング

クエリキーワード1

- SQLにキーワードを付与して、動作をコントロールすることができます
- INSERT/UPDATE/DELETE用キーワード
  - %NOLOCK
    - データ追加・更新時にレコードのロックを実行しない。レコードの競合が発生しないことが分かっている場合に使用可能。例：深夜のオンラインユーザ切断中のバッチ処理など
  - %NOCHECK
    - 外部キーの参照整合性チェックやデータの妥当性検証を行わない。例：外部システムやH/Wから出力されたデータの追加など
  - %NOINDEX
    - インデックスを生成しない。全レコード追加後にまとめて%BuildIndices()メソッド等を使って自分で生成する必要あり
  - %NOTRIGGER
    - トリガを実行しない。

25

SQLパフォーマンスチューニング

クエリキーワード2

- SELECT用キーワード(FROM句の後につける)
  - %IGNOREINDICES
    - 指定されたインデックスを使用しない

    SELECT * FROM %IGNOREINDICES Symposia.Table6.FlagIndex Symposia.Table6 where Flag=0 and P1 ='A'

    ほとんどのデータでFlag=0なのでインデックスを参照しないように指定する
  - JOINするテーブルの順序をコントロールするキーワード
    - %FULL、%INORDER、%STARTTABLE
  - 各クエリ最適化オプションを実行しないように指定するキーワード
    - %NOFLATTEN、%NOMERGE、%NOSVSO、%NOTOPOPT、%NOUNIONOROPT

26

SQLパフォーマンス計測

パフォーマンスを計測しよう

- %SYS.PTools.SQLStats
  - クエリの実行時間を測定し、データベースに保存する
  - 有効にすると、クエリキャッシュ内に計測用のコードが埋め込まれる
- 設定方法
  - Do $SYSTEM.SQL.SetSQLStats(n)
    - 0：計測用コード生成を無効にする
    - 1：計測用コードを生成するが、測定は行わない
    - 2：計測用コードを生成し、クエリの開始・終了を測定する
    - 3：計測用コードを生成し、クエリの開始・終了、モジュール毎の時間も測定する

27

SQLパフォーマンス計測

測定する項目・結果確認

- 測定する項目
  - グローバル参照数
  - コードの実行行数
  - クエリ・モジュールの呼び出された回数
  - クエリ・モジュールの実行時間
  - クエリによって返された行数
- 結果の確認
  - SQLStatsクラス内に定義されている、View [%SYS_PTools.SQLStatsView] を使用するのが良い。
- 利用する可能性がある場合は $SYSTEM.SQL.SetSQLStats(1)を設定しておくとよい
  - PTools未設定状態から、有効にする場合はクエリキャッシュをパージする必要あり。

28

SQLパフォーマンス計測

結果確認クエリ１

```
SELECT RoutineName, ModuleName, ModuleCount,
  GlobalRefs, LinesOfCode, TotalTime, RowCount,
  QueryType, StartTime, QueryText
      FROM %SYS_PTools.SQLStatsView
          WHERE Namespace= 'SYMPOSIA'
```

SQLCODE: 100 行数: 13　パフォーマンス: 0.003 秒　183 グローバル参照

#	RoutineName	ModuleName	ModuleCount	GlobalRefs	LinesOfCode	TotalTime	RowCount	QueryType	StartTime	QueryText
1	%sqlcq.NCUH.1	MAIN	1	156	2550	.003186	16	Unknown	2011-04-26 15:50:34	SELECT * FROM Symposia . Table5 WHERE Sale
2	%sqlcq.NCUH.11	MAIN	1	6	459	.000306	0	Unknown	2011-04-26 15:54:08	SELECT RoutineName , ModuleName , ModuleCo
3	%sqlcq.NCUH.11	MAIN	1	6	408	.00029	0	Unknown	2011-04-26 15:54:12	SELECT RoutineName , ModuleName , ModuleCo
4	%sqlcq.NCUH.11	MAIN	1	147	3417	.004586	11	Unknown	2011-04-26 15:54:20	SELECT RoutineName , ModuleName , ModuleCo
5	%sqlcq.NCUH.16	MAIN	1	2027	6375	.002389	1	SELECT	2011-04-26 15:50:14	SELECT P1 FROM Symposia . Table1 WHERE P2
6	%sqlcq.NCUH.17	MAIN	1	254	2652	.00132	16	SELECT	2011-04-26 15:50:55	SELECT * FROM Symposia . Table5 WHERE (fn
7	%sqlcq.NCUH.18	MAIN	1	2120	12699	.010762	38	SELECT	2011-04-26 15:51:04	SELECT * FROM Symposia . Table1 WHERE $ EX
8	%sqlcq.NCUH.19							SELECT		SELECT RoutineName , ModuleName , ModuleCo
9	%sqlcq.NCUH.2	MAIN	1	156	2550	.00131	16	Unknown	2011-04-26 15:50:48	SELECT * FROM Symposia . Table5 WHERE Sale
10	%sqlcq.NCUH.4	MAIN	1	2423375	8380320	2.853697	1	Unknown	2011-04-26 15:49:43	SELECT COUNT (*) FROM Symposia . Table2 ,
11	%sqlcq.NCUH.5	MAIN	1	112	2448	.001299	8	Unknown	2011-04-26 15:51:28	SELECT RoutineName , ModuleName , ModuleCo
12	%sqlcq.NCUH.cls11.1							SELECT		DECLARE QRS CURSOR FOR SELECT COUNT (
13	SQL1	MAIN	1	14	102	.000186	1	SELECT	2011-04-26 15:48:52	select count(*) into :cnt from Symposia.Person

完了

29

SQLパフォーマンス計測

結果確認クエリ２

```
SELECT RoutineName as クエリキャッシュ名,
  ModuleName as モジュール名, SUM(ModuleCount) AS
  クエリ実行回数, AVG(TotalTime) AS 平均実行時間,
  SUM(TotalTime) AS 合計実行時間, AVG(GlobalRefs) AS
  平均グローバル参照数, AVG(LinesOfCode) AS 平均コー
  ド実行行数, QueryText as クエリテキスト
FROM %SYS_PTools.SQLStatsView
  WHERE NameSpace = 'SYMPOSIA'
      GROUP BY RoutineName, ModuleName
          ORDER BY 合計実行時間 DESC
```

＊クエリ単位で集計し、合計実行時間でソートすることでシステムへの影響が大きいクエリを洗い出すことができる。

30

SQLパフォーマンス計測

SQLStatsの注意点

- Select 句の場合、クエリの開始からFetchの終了までを計測しています。

  →クライアントでFetch毎に処理をしている場合は、その時間も含まれます。＝サーバでかかった時間を計測しているわけではありません。

- システムワイドの設定です。
- １度クエリを実行するごとに複数回レコードの更新が実行されます。システムに負荷がかかります。

31

SQLパフォーマンス計測

クエリパラメータが必要な場合

- 同じクエリのパフォーマンスに大きな差がある場合は、クエリパラメータにパフォーマンスが依存していることが考えられます。
- クエリパラメータはView [%SYS_PTools.Query_With_Parameters]を参照して取得することができます（2011.1~）
  - 直接の参照関係はないので、上記ViewのRunTimeとSQLStatsViewのStartTimeを比較します。

32

SQLパフォーマンス計測

SQLベンチマークを実行する

- 収集したクエリ情報、クエリパラメータを利用して、再度SQLを順番に実行することができます。(2011.1~)

```
##class(%SYS.PTools.SQLBenchMarkQueries).Run(ns,1)
```

実行結果の確認

```
SELECT QueryId->QueryText,* FROM
        %SYS_PTools.SQLBenchMarkResults
                WHERE QueryId->NameSpace='SYMPOSIA'
```

*クエリ毎に結果を集計してまとめているわけではないので注意。クエリ毎に見るには、 QueryId->QueryText等でのGroup byが必要。

INTERSYSTEMS

33