モジュール5

インデックスとチューニング

InterSystems®
Health | Business | Government

このモジュールの主な目標

- このモジュールでは、インデックスを使用してSQLパフォーマンスの最適化を行うために必要な調査方法、クエリプランの読み方を習得いただくため、以下の内容についてご説明します。
  - SQLのパフォーマンス測定方法
  - クエリプランの読み方
    - テーブル定義に沿った実データ（グローバル変数）の構造を知る
  - テーブル・データに基づく最適化
  - テーブル定義の最適化
  - クエリの最適化

最適化といっても・・・・

列データの種類数に
大幅な変化がないか

- テーブル・データに基づく最適化はどうか
  - テーブルのチューニングをした方がいい？
- テーブル定義の最適化はどうか
  - インデックスは最適？

どんなインデックスを
追加したらいいのか

- クエリの最適化はどうか
  - 現状のプランからもっと効率よくできないか？
- 構成の最適化はどうか
  - そもそも、メモリ（主にデータベースキャッシュ）は足りてるか？

^mgstatルーチンを使用してキャッシュ効率を確認します。
詳細は、公開ガイド内P8～をご参照ください。

SQLの実行時統計情報の収集

- 以下の2種類の方法があります。
  - 自動で収集される統計情報
    - 管理ポータル > システムエクスプローラ > SQL 画面の「SQLステート」または「テーブルのSQL文」画面で確認できます。
    - 設定は特にありません。
  - デフォルトで無効化されている統計情報
    - 管理ポータル > システムエクスプローラ > ツール > SQL パフォーマンス・ツール を使用します。
    - 自動で収集される統計データより詳細な情報を収集できます。
    - パフォーマンスの調査の流れで使用することがあります。
    - 収集を開始したいとき、有効化する必要があります。

自動で収集される統計情報

- 日常的にどのようなSQLが何回実行されているのか、どのようなプランが使用されているかを確認するには、管理ポータルのSQL画面内「SQLステートメント」または「テーブルのSQL文」メニューが便利です。
  - ドキュメントでは「SQL文」と表示されています。

テーブル単位に表示させる場合
テーブル選択
＞（右画面）カタログの詳細
＞ テーブルのSQL文

システム > SQL
フィルタ Training.* 適用先 すべて
システム 非推奨
スキーマ Training
テーブル
Training.Capability
Training.Employee
ビュー
プロシージャ
クエリキャッシュ

ウィザード » アクション » テーブルを開く ツール » ドキ
カタログの詳細 クエリ実行 参照 SQLステートメント
テーブル: Training.Employee テーブル情報 フィールド マップ/インデックス トリガ 制約 クエリ・キャッシュ テーブルの SQL 文

#	テーブル	プランの状態	新しいプラン	実行回数	実行回数/日	合計時間	平均時間	StdDev 時間	行数	行数/日	実行されたコマンド	実行コマンド数/日	場所	ステートメント
1	Training.Employee	Unfrozen		1	1	0.0040220	0.0040220	0	2	2	92645	92645	%sqlcq.USER.cls9.1	DECLARE QRS CURSOR FOR SELECT
2	Training.Employee	Unfrozen											Training.Employee.1	INSERT INTO TRAINING . EMPLOYEE
3	Training.Employee	Unfrozen		2	2	0.0089260	0.0044630	0.0028480	10000	10000	410086	410086		DECLARE GETROWS CURSOR FOR S
4	Training.Employee	Unfrozen		1	1	0.0010930	0.0010930	0	226	226	99641	99641	%sqlcq.USER.cls8.1	DECLARE QRS CURSOR FOR SELECT

ネームスペース全体のSQL文を参照する場合

SQL文の詳細画面に移動します（次ページ参照）。

SQL文の詳細画面

プランを凍結 | プランを凍結解除 | SQL 統計情報をクリア | エクスポート | ページを更新 | 閉じる

ステートメント詳細

SQLステートメントID	176	ステートメント・ハッシュ	34cHvjFlJSsdX8boKYsM01bIxaQ=	最初に見た日付	2025-05-26
プランの状態	凍結解除	合計時間	0.0040220	行数	2
実行回数	1	平均時間		行数/日	2
実行回数/日	1	StdDev 時間	0	実行されたコマンド	92645
				実行コマンド数/日	92645

コンパイル設定

選択モード	Runtime	デフォルト・スキーマ	SQLUSER	バージョン	2025.1.0.225
ユーザ名	SuperUser	スキーマ・パス		タイムスタンプ	2025-05-26 09:25:48
クライアントIPアドレス	172.21.0.1	クライアント名	localhost	クライアントアプリケーション	CSPa24.so

コールスタック GetQueryParameters+14^%CSP.UI.Portal.SQL.Utils.1,PrepareQuery+4^%CSP.UI.Portal.SQL.Home.1,InvokeClassMethod

ステートメントは以下のルーチンで定義されています

ルーチン	タイプ	最終コンパイル日時	実行時ルーチン	
%sqlcq.USER.cls9.1	Class Method	05/26/2025 09:25:48	%sqlcq.USER.cls9.1	-

ステートメントは以下のリレーションを使用します

テーブルまたはビューの名前	タイプ	最終コンパイル日時	クラス名	
Training.Employee	Table	05/26/2025 09:25:34.074626721	Training.Employee	クラスをコ

ステートメントテキストとクエリプラン

ステートメント

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND
```

クエリプラン

相対コスト = 145800

- Divide master map Training.Employee.IDKEY into subranges of IDs.
- Call module A in parallel on each subrange, piping results into temp-file A.
- Read temp-file A, looping on a counter.
- For each row:
  - Output the row.

Module: A

- Read master map Training.Employee.IDKEY, looping on the subrange of ID.
- For each row:
  - Test the = condition on %SQLUPPER(Location), the = condition on %SQLUPPER(Name), the NOT NULL condition on %SQLUPPER(Location), and the NOT NULL condition on %SQLUPPER(Name).
  - Add a row to temp-file A, subscripted by a counter, with node data of Dept, EmpID, ID, Location, Name, and Tel.

この画面に表示されているSQL 実行時統計情報は常に収集されOffにすることができません。統計情報の収集をできる限り効率化するために、統計情報は設定された間隔でのみ書き込まれます。（最大30分間隔があくこともあります）
同じ情報を、INFORMATION_SCHEMAスキーマ以下の3つのテーブルでも確認できます。
STATEMENTS
STATEMENT_DAILY_STATS
STATEMENT_HOURLY_STATS

デフォルトで無効化されている統計情報

- SQLパフォーマンス・ツール以下にあるSQL実行時統計情報画面は「今この瞬間の情報を取得したい」場合に最適です。
  - 自動で収集される統計情報より詳細な情報を収集します。
  - ユーザが統計データの収集開始、停止を指定できるため、指定期間に多く実行されたクエリ、遅いクエリを特定したい場合に適したツールです。

デフォルトで無効化されている統計情報利用方法

- 管理ポータルの「SQL実行時統計」を利用してクエリ実行時のパフォーマンスを測定できます。
  - 管理ポータル→ツール→SQLパフォーマンス・ツール→SQL実行統計情報

デフォルトでは無効化しているため「設定」タブで情報を収集するように設定します。

SQL 実行時統計情報

このツールでSQL 実行時統計情報の SQL 実行時パフォーマンス詳細を収集して表示します。

設定 | クエリテスト | 統計情報表示

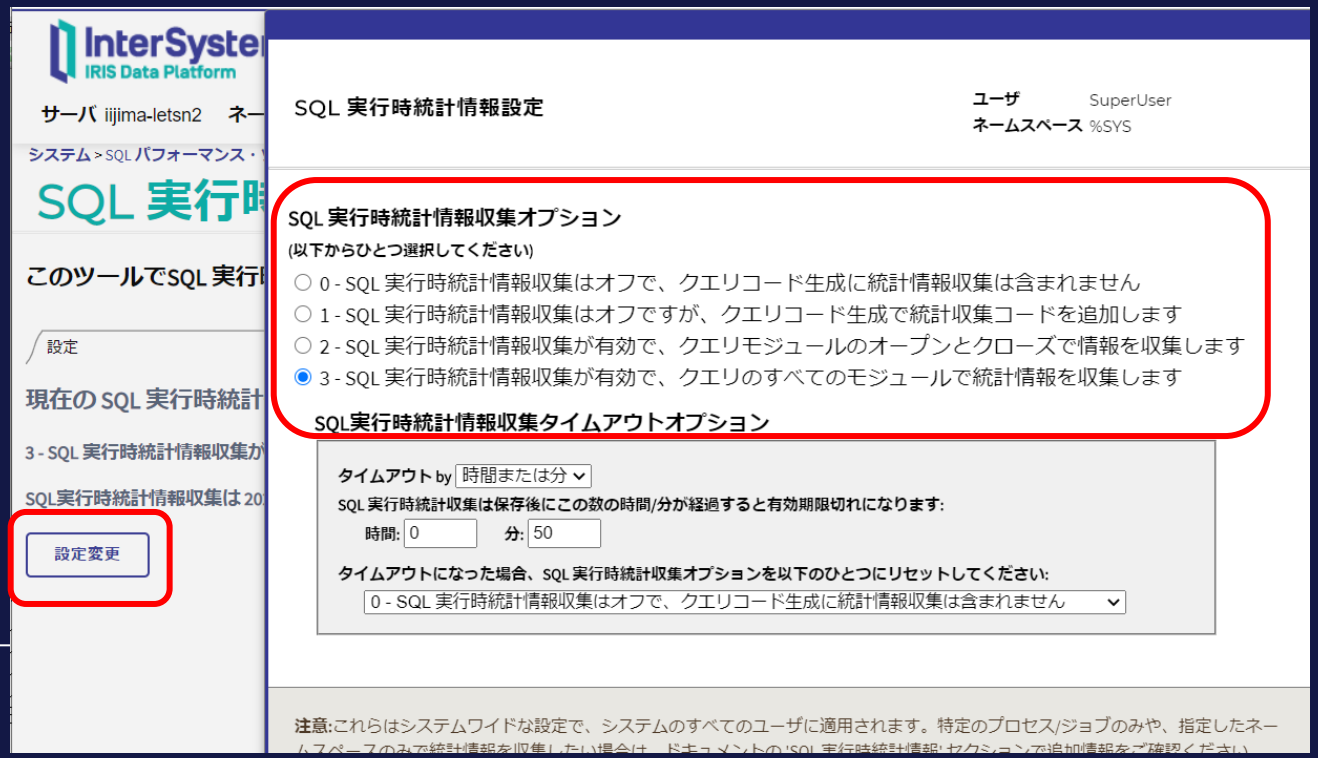
以下のテーブルには SQL 統計情報を含むすべてのクエリの情報が表示されています。 テーブルの再読み込み

ページサイズ: 0 最大行数: 1000 結果: 50 ページ: 1 の 1

ルーチン	カーソル	実行回数	平均行	平均グローバル参照	平均コマンド	平均ディスク待ち	平均時間	SQL文
%sqlcq.USER.cls6.1	QRS0	1	5.00	122.00	1921.00	1.00	0.00181	SELECT I
%sqlcq.USER.cls7.1	QRS0	1	500.00	9578.00	141565.00	0.00	0.03648	SELECT I
%sqlcq.USER.cls8.1	QRS0	1	71.00	1841.00	26647.00	0.00	0.00596	SELECT I
%sqlcq.USER.cls9.1	QRS0	1	12.00	238.00	3645.00	0.00	0.00142	SELECT I
%sqlcq.USER.xE1NVoOgpCgFiWicMOsqQsdZ0qkA.1	1A							INSERT I
%sqlcq.USER.xE723I97a3eYis6W99bFykXaTeWM.1	1A							INSERT I

SQL実行時統計情報　手順

1. 設定タブで統計コードの生成を有効にする
2. 統計コードを生成させるため、クエリキャッシュの削除かソースコードのコンパイルを行う
   - クエリキャッシュを削除することで統計コードの生成を開始します。
     - （2019.1以前）埋め込みSQLについてはクエリが記載されたソースコードをコンパイルすることで情報収集を開始します。



例

参照したいSQL文を選択します

ページサイズ: 0　最大行数: 1000　結果: 50　ページ: 1 の 1

	ルーチン	カーソル	実行回数	平均行	平均グローバル参照	平均コマンド	平均ディスク待ち	平均時間	SQL文
	%sqlcq.USER.cls6.1	QRS0	1	5.00	122.00	1921.00	1.00	0.00181	SELECT ID
»	%sqlcq.USER.cls7.1	QRS0	1	500.00	9578.00	141565.00	0.00	0.03648	SELECT ID
	%sqlcq.USER.cls8.1	QRS0	1	71.00	1841.00	26647.00	0.00	0.00596	SELECT ID
	%sqlcq.USER.cls9.1	QRS0	1	12.00	238.00	3645.00	0.00	0.00142	SELECT ID
	%sqlcq.USER.xF1NVoOapCaFiWicMOsaOsdZ0akA.1	1A							INSERT INT

実行回数	モジュール名	平均モジュール実行回数	平均行	平均グローバル参照	平均コマンド	平均ディスク待ち	平均時間
1	INFO	1.00	500.00	9578.00	141565.00	0.00	0.04
1	MAIN	1.00	500.00	9578.00	141565.00	0.00	0.04
1	FIRST	1.00		500.00	28493.00	0.00	0.01
1	B	1.00		501.00	22518.00	0.00	0.00

クエリプラン

相対コスト = 2413600

Module	Rows Returned	Performance (secs)	Global Refs	Lines Exec	Read Latency (ms)
MAIN	500	0.036482	9,578	141,565	0

Module	Rows Returned	Performance (secs)	Global Refs	Lines Exec	Read Latency (ms)
FIRST		0.005373	500	28,493	0

Module	Rows Returned	Performance (secs)	Global Refs	Lines Exec	Read Latency (ms)
B		0.003689	501	22,518	0

- Read master map Training.Employee.IDKEY, looping on ID.
- For each row:
  - Output the row.

Read master map

クエリプランとは

- クエリがどのように実行されるか処理内容を表記したもの（英語）
- 出てくる用語
  - **map（マップ）**
    - テーブルが持つデータ構造を示す（データ or インデックス）
    - 1つのテーブルが1つ以上のマップを持つ
  - **module（モジュール）**
    - 一時テーブルを作る処理
  - **temp-file n（一時テーブル）**
    - ソート、結合などで必要になる一時テーブルで、moduleで作成される

クエリプランを参照してみよう！

- 管理ポータル→システムエクスプローラ→SQL の画面のクエリ実行から
- 管理ポータル→システムエクスプローラ
  →ツール→SQL実行時統計情報→クエリのテスト から

相対コスト = 2831.9

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

  Output the row.

```
select ad.City, co.AmountIn, co.CurrencyIn, co.AmountOut, co.CurrencyOut,
co.DateStamp, co.TS
from FCE.CurrencyOrder co
 left outer join FCE.ATM atm on atm.%ID=co.ATM
 left outer join FCE.Branch b on atm.Branch=b.%ID
 left outer join FCE.RealEstate ad on b.Address=ad.%ID
where co.Status='Pending' and (co.CurrencyIn = 'USD' or CurrencyOut='USD')
order by ad.City
```

- Read master map FCE.CurrencyOrder.IDKEY, looping on ID.
- For each row:

  Read master map FCE.ATM.IDKEY, using the given idkey value.
  Generate a row padded with NULL for table FCE.ATM if no row qualified.
  Read master map FCE.Branch.IDKEY, using the given idkey value.
  Generate a row padded with NULL for table FCE.Branch if no row qualified.
  Read master map FCE.RealEstate.IDKEY, using the given idkey value.
  Generate a row padded with NULL for table FCE.RealEstate if no row qualified.
  Add a row to temp-file A, subscripted by %SQLUPPER(City) and a counter,
  with node data of AmountIn, AmountOut, CurrencyIn, CurrencyOut, DateStamp, TS, and City.

全レコードをループして参照している

Read master map スキーマ名.テーブル名.IDKEY, looping on IDを理解するための

レコードデータの実データ(グローバル)の構造

- クラス定義初回コンパイル時に以下の命名規則でグローバル変数の名称が確定し、同時にデータ部にプロパティ(=フィールド)値を格納する順番も決定します。
  - Storage定義に登録されます。
  - プロパティ(フィールド)追加を行うと、末尾に格納されるように定義されます。
- メモ:CREATE TABLE文で作成したテーブルに対応するグローバル変数名は以下表とは異なる形式で初回コンパイル時定義されます。
  - 詳細は「テーブル定義のデータが格納されるグローバル変数名について」をご参照ください。

データ	^スキーマ名.テーブル名**D=$LB(フィールドデータ)** (フィールド値は**$L**IST**B**UILD()関数を使用した内部形式で登録されます。) 【例】 ^Training.Employee**D**
インデックス	^スキーマ名.テーブル名**I** 【例】 ^Training.Employee**I**
ストリーム BLOB/CLOB	^スキーマ名.テーブル名**S** 【例】 ^Training.Employee**S**

Read **master** mapは このグローバルのこと

Read **index** mapは このグローバルのこと

Read master map スキーマ名.テーブル名.IDKEY, looping on IDを理解するための

レコードデータの実データ(グローバル)の構造 Storage定義

```
Class Training.Employee Extends %Persistent
{

Property EmpID As %String [ SqlComputeCode = { set {*}="EMP"_$TR($J({%%ID},4)," ","0")
}, SqlComputed, SqlComputeOnChange = %%INSERT ];

Property Name As %String;

Property Dept As %String;

Property Location As %String;

Property Tel As %String;

Index EmpIDIdx On EmpID [ Unique ];

Index LocationIdx On Location;
```

```
^Training.EmployeeD=20
^Training.EmployeeD(1)=$lb("","EMP0001","飯塚","","徳島県","08-4259-2162")
^Training.EmployeeD(2)=$lb("","EMP0002","石原","","福岡県","0644-93-8387")
^Training.EmployeeD(3)=$lb("","EMP0003","浅野","","石川県","04-6801-1348")
^Training.EmployeeD(4)=$lb("","EMP0004","石渡","","熊本県","06-7775-4888")
^Training.EmployeeD(5)=$lb("","EMP0005","石塚","","鹿児島県","04-6491-4090")
^Training.EmployeeD(6)=$lb("","EMP0006","石渡","営業部","埼玉県","04-6436-254
^Training.EmployeeD(7)=$lb("","EMP0007","石川","","埼玉県","03-9666-4058")
^Training.EmployeeD(8)=$lb("","EMP0008","相原","","長崎県","07-4325-9277")
^Training.EmployeeD(9)=$lb("","EMP0009","市川","","香川県","05-6221-8456")
^Training.EmployeeD(10)=$lb("","EMP0010","新井","教育部","三重県","03-9460-95
```

```
Storage Default
  {
 <Data name="EmployeeDefaultData">
  <Value name="1">
    <Value>%%CLASSNAME</Value>
  </Value>
  <Value name="2">
    <Value>EmpID</Value>
  </Value>
  <Value name="3">
    <Value>Name</Value>
  </Value>
  <Value name="4">
    <Value>Dept</Value>
  </Value>
  <Value name="5">
    <Value>Location</Value>
  </Value>
  <Value name="6">
    <Value>Tel</Value>
  </Value>
 </Data>
 <DataLocation>^Training.EmployeeD</D
 <DefaultData>EmployeeDefaultData</Def
 <ExtentSize>20</ExtentSize>
 <IdLocation>^Training.EmployeeD</IdLo
 <IndexLocation>^Training.EmployeeI</I
 <Property name="%%CLASSNAME">
  <AverageFieldSize>1</AverageFieldSiz
```

Read master map スキーマ名.テーブル名.IDKEY, looping on ID とはこの**サブスクリプト(レコードID)**でループすること

インデックスデータの実データ(グローバル)の構造

```
1:  ^Training.EmployeeI("LocationIdx"," 三重県",10) = ""
2:  ^Training.EmployeeI("LocationIdx"," 三重県",14) = ""
3:  ^Training.EmployeeI("LocationIdx"," 北海道",11) = ""
4:  ^Training.EmployeeI("LocationIdx"," 埼玉県",6) = ""
5:  ^Training.EmployeeI("LocationIdx"," 埼玉県",7) = ""
6:  ^Training.EmployeeI("LocationIdx"," 山形県",19) = ""
7:  ^Training.EmployeeI("LocationIdx"," 岡山県",20) = ""
8:  ^Training.EmployeeI("LocationIdx"," 徳島県",1) = ""
9:  ^Training.EmployeeI("LocationIdx"," 愛媛県",12) = ""
10: ^Training.EmployeeI("LocationIdx"," 愛媛県",15) = ""
11: ^Training.EmployeeI("LocationIdx"," 愛媛県",18) = ""
12: ^Training.EmployeeI("LocationIdx"," 熊本県",4) = ""
13: ^Training.EmployeeI("LocationIdx"," 石川県",3) = ""
14: ^Training.EmployeeI("LocationIdx"," 石川県",17) = ""
15: ^Training.EmployeeI("LocationIdx"," 福岡県",2) = ""
16: ^Training.EmployeeI("LocationIdx"," 長崎県",8) = ""
17: ^Training.EmployeeI("LocationIdx"," 長崎県",16) = ""
18: ^Training.EmployeeI("LocationIdx"," 静岡県",13) = ""
19: ^Training.EmployeeI("LocationIdx"," 香川県",9) = ""
20: ^Training.EmployeeI("LocationIdx"," 鹿児島県",5) = ""
```

埼玉県には、ID=6と7の人が住んでいることがわかります

ID	Name	Loc	Location	ID
1	飯塚	徳島県	三重県	10
2	石原	福岡県	三重県	14
3	浅野	石川県	北海道	11
4	石渡	熊本県	埼玉県	6
5	石塚	鹿児島県	埼玉県	7
6	石渡	埼玉県	山形県	19
7	石川	埼玉県	岡山県	20
8	相原	長崎県	徳島県	1
9	市川	香川県	愛媛県	12
10	新井	三重県	愛媛県	15
11	市川	北海道	愛媛県	18
12	石原	愛媛県	熊本県	4

レコードデータ　インデックスデータ

```
Index LocationIdx On Location;
```

インデックス名

Locationが北海道の従業員を検索する場合

```
^Training.EmployeeD=20
^Training.EmployeeD(1)=$lb("","EMP0001","飯塚","","徳島県","08-4259-2162")
^Training.EmployeeD(2)=$lb("","EMP0002","石原","","福岡県","0644-93-8387")
^Training.EmployeeD(3)=$lb("","EMP0003","浅野","","石川県","04-6801-1348")
^Training.EmployeeD(4)=$lb("","EMP0004","石渡","","熊本県","06-7775-4888")
^Training.EmployeeD(5)=$lb("","EMP0005","石塚","","鹿児島県","0
^Training.EmployeeD(6)=$lb("","EMP0006","石渡","営業部","埼玉県
^Training.EmployeeD(7)=$lb("","EMP0007","石川","","埼玉県","03-
^Training.EmployeeD(8)=$lb("","EMP0008","相原","","長崎県","07-
^Training.EmployeeD(9)=$lb("","EMP0009","市川","","香川県","05-6221-845
^Training.EmployeeD(10)=$lb("","EMP0010","新井","教育部","三重県","03-940 9535"
)
^Training.EmployeeD(11)=$lb("","EMP0011","市川","カスタマーサポート部","北海道"
"01-4382-8775")
^Training.EmployeeD(12)=$lb("","EMP0012","石原","","愛媛県","0976-88-5864")
```

レコードIDでループしながらデータ部のLocationをチェックします

```
1:  ^Training.EmployeeI("LocationIdx"," 三重県",10) = ""
2:  ^Training.EmployeeI("LocationIdx"," 三重県",14) = ""
3:  ^Training.EmployeeI("LocationIdx"," 北海道",11) =
4:  ^Training.EmployeeI("LocationIdx"," 埼玉県",6)  = ""
5:  ^Training.EmployeeI("LocationIdx"," 埼玉県",7)  = ""
6:  ^Training.EmployeeI("LocationIdx"," 山形県",19) = ""
7:  ^Training.EmployeeI("LocationIdx"," 岡山県",20) = ""
8:  ^Training.EmployeeI("LocationIdx"," 徳島県",1)  = ""
9:  ^Training.EmployeeI("LocationIdx"," 愛媛県",12) = ""
10:
11:
12:
13:
14:  Training.EmployeeI("LocationIdx"," 石川県",17) =
15: ^Training.EmployeeI("LocationIdx"," 福岡県",2)  = ""
```

インデックスを利用する場合
第1サブスクリプト：LocationIdx
（＝インデックス名）
第2サブスクリプト：北海道
に紐付く第3サブスクリプトのIDを探すだけ

グローバル変数のサブスクリプトはキーとなるような値が格納されているため、素早く取得したい情報に辿りつけます。

インデックス未使用／使用時のプラン

```
SELECT ID, Dept, EmpID, Location, Name, Tel
FROM Training.Employee
where Location='北海道'
```

クエリプラン

相対コスト = 2413600 ★

インデックス未使用

- Read master map Training.Employee.IDKEY, looping on ID.
- For each row:
  - Output the row.

LocationIdxを使用して LocationのデータとIDをループし、条件に合う内容を取得しています

相対コスト = 85000 ★

- Read index map Training.Employee.LocationIdx, using the given %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

インデックス使用

クエリプランの相対コスト

- 同じクエリ（SQL）で比較できる相対的なコストで、主にインデックス追加前後の効果を測定できます。
  - この場合、相対コストが低いプランの方が良いプランといえます。
- ※注意点※
  異なるクエリ同士のコスト比較は意味がありません。

相対コストのほかにも、**グローバル参照数**
クエリ実行時間、を前後比較しながら、良いプランであるどうかを判断します。

なぜ、グローバル参照数を確認すべきなのか

ユーザプロセス
（ターミナルやプログラム）
ID＝1のデータをSELECT
^Training.EmployeeD(1)
$lb("","EMP0001","飯塚",
"","徳島県","08-4259-2162")

データベースキャッシュ
（共有メモリ）
^Training.EmployeeD(1)=
$lb("","EMP0001",
"飯塚","","徳島県",
"08-4259-2162")

データベースキャッシュに
グローバルが存在しない
場合、ディスクを参照し、
データベースキャッシュに
対象のブロック（8K）全体
をロードする

グローバル変数へのアクセスはディスクI/Oを伴う処理であるため、できるだけグローバル変数への参照回数が少なくなるようなプランとなるよう調整します。

※ グローバル参照数が多くても対象グローバルがデータベースキャッシュ上に存在していれば、ディスクI/Oを伴いません。

※ 同一グローバル変数の参照回数をできるだけ減らすことでの内部処理が減り、パフォーマンス劣化を防ぐ意味があります。

再掲：インデックス未使用／使用時のプラン

実行 | プラン表示 | 履歴を表示 | クエリビルダ | 表示モード | 最大 1000 | その他オプシ

```
SELECT ID, Dept, EmpID, Location, Name, Tel
FROM Training.Employee,
where Location='北海道'
```

行数: 10 パフォーマンス: 0.004 秒 827 グローバル参照 7945 実行されたコ
終更新:2020-11-28 10:51:21.286

相対コスト = 2413600 ★

インデックス未使用

- Read master map Training.Employee.IDKEY, looping on ID.
- For each row:
  - Output the row.

実行 | プラン表示 | 履歴を表示 | クエリビルダ | 表示モード | 最大 1000 | その

```
SELECT ID, Dept, EmpID, Location, Name, Tel
FROM Training.Employee,
where Location='北海道'
```

グローバル参照回数が　827→347　に減りました

行数: 10 パフォーマンス: 0.004 秒 347 グローバル参照 4037 実行されたコマンド 0 ディスク読み込みレイテンシ

相対コスト = 85000 ★

- Read index map Training.Employee.LocationIdx, using the given %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

インデックス使用

クエリプランのキーワード

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
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配列を作成しサブスクリプトAAAとIDを設定し、データ部にBBBを保存
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

悪いケース:テーブルスキャン

- Read master map Training.Employee.IDKEY, looping on ID.
  - このプランは、全データグローバルをループして参照しているため、<u>データ件数の多いテーブルの場合は良くない</u>
- ただし、
  Read master map Training.Employee.IDKEY, **using the given idkey value.**
  は、問題なし。
  - Training.EmployeeのLocationインデックスを使用したプランで、Locationインデックスから得られたID番号を使用してmaster mapからデータを取得しているため問題なし。

相対コスト = 85000

- Read index map Training.Employee.LocationIdx, using the given %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

悪いケース：一時テーブルの作成

- 一時テーブルを作成している場合、<u>件数によりますが</u>最適ではない場合もあります。

クエリ例

```
SELECT
ID,Name,Dept,Tel
FROM Training.Employee
Where Location='北海道' and Name %Startswith '青'
```

相対コスト = 22087

- Call module C once, which populates bitmap temp-file A.
- Generate a stream of idkey values using the multi-index combination:
  ((index map Training.Employee.LocationIdx) INTERSECT (bitmap temp-file A))
- For each idkey value:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

Module: C

- Read index map Training.Employee.NameIdx, looping on %SQLUPPER(Name) (with a %STARTSWITH range condition) and ID.
- For each row:
  - Add ID bit to bitmap temp-file A.

WHERE節にAND条件を含む場合、**<u>複合インデックスを利用することで、2種類のインデックスをあわせる作業がいらなくなります。</u>**
【メモ】条件に使用するフィールドの選択性（後述）によっては複合条件としない方が良い場合もあります。

この例では、NameIdxインデックスを使用して temp-file Aを作成し、その後、LocationIdxインデックスの結果と合わせる作業をしています。

良いケース:インデックスを使用している

- 前頁と同じクエリを 複合インデックスを定義し、さらに、インデックスデータを指定した後で実行した場合のプランは以下の通りです。
  - 2種類のインデックスの結果を合わせる作業がなくなったのと、一時テーブルの作成がなくなっています。
  - <u>複合インデックス定義時、フィールドを指定する順序が重要で、選択性を確認して決定します。</u>

クエリ例

```
SELECT
ID,Name,Dept,Tel
FROM Training.Employee
Where Location='北海道' and Name %Startswith '青'
```

相対コスト = 37674

- Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(Location), and looping on %SQLUPPER(Name) (with a %STARTSWITH range condition) and ID.
- For each row:
  - Output the row.

```
Index NameLocationIdx On (Name, Location) [ Data = (Dept, Tel, Name) ];
```

```
^Training.EmployeeI("NameLocationIdx"," 青木"," 兵庫県",169)  = $lb("","広報部","05-3929-5964","青木")
^Training.EmployeeI("NameLocationIdx"," 青木"," 兵庫県",329)  = $lb("","総務部","0258-15-9551","青木")
^Training.EmployeeI("NameLocationIdx"," 青木"," 北海道",61)   = $lb("","広報部","0886-24-2030","青木")
^Training.EmployeeI("NameLocationIdx"," 青木"," 北海道",409)  = $lb("","カスタマーサポート部","0203-60-9174","青木")
^Training.EmployeeI("NameLocationIdx"," 青木"," 千葉県",144)  = $lb("","総務部","0874-40-1181","青木")
^Training.EmployeeI("NameLocationIdx"," 青木"," 和歌山県",369) = $lb("","総務部","0750-79-4337","青木")
^Training.EmployeeI("NameLocationIdx"," 青木"," 和歌山県",463) = $lb("","","0532-42-8058","青木")
```

良いケース:ビットマップインデックスの例

- ビットマップインデックスのビット演算で一致するレコードを特定しているようなインデックス

```
24  Index LocationIdx On Location [ Type = bitmap ];
25
26  Index DeptIdx On Dept [ Type = bitmap ];
27
28  Index NameIdx On Name [ Type = bitmap ];
```

実行 | プラン表示 | 履歴を表示 | クエリビルダ | 表示モード

```
SELECT ID, Dept, EmpID, Location, Name, Tel
FROM Training.Employee
where Location='北海道' and (Name='荒井' or Dept='広報部')
```

相対コスト = 75264

- Generate a stream of idkey values using the multi-index combination:
  ((bitmap index Training.Employee.LocationIdx) INTERSECT ((bitmap index Training.Employee.NameIdx) UNION (bitmap index Training.Employee.DeptIdx)))
- For each idkey value:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

ビットマップインデックスについて

- ビットマップインデックスとは、インデックスをビット列で表すインデックスです。
  - 各インデックスエントリーは、64000ビット(1か0)のビット文字列で、64000のテーブル行に対応します。
  - ビット文字列は、圧縮されているので、64000ビットのストレージ（8000バイト）が、8KBより少なくて済みます。
  - ビット列でインデックスデータを表しているため、ANDやORの論理演算や集計など、高速に処理できます。

ビットマップインデックス 動作原理(1)

レコード番号

	1	2	3	4	5
2) Gender = Male	0	1	0	1	0
3) Gender = Female	1	0	1	0	1
4) City = Boston	0	0	0	1	1
5) Age = 20	1	0	0	1	1
6) …..etc …..					

条件

INSERT Person (Age,City,Gender) VALUES(20,'Boston','M')

ビットマップインデックス
動作原理(2)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	.....
2) Gender = Male	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	1	.....
3) Gender = Female	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1	0	.....
4) City = Boston	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	1	.....
5) Age = 20	1	0	0	1	1	0	0	1	0	0	0	1	0	1	1	1	………
6) …..etc ….																	

条件

データ増加に伴い、各フィールドの種類が**250,000通り**存在し、データが**10,000,000件**存在する場合、**25万行×1千万列の表ができあがるイメージ**

ビットマップインデックス
動作原理(3)

2)Gender=Male	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	1	…
3)Gender=Female	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1	0	…
4)City = Boston	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	1	…
5)Age = 20	1	0	0	1	1	0	0	1	0	0	0	1	0	1	1	1	…..
6)…..etc …..																	
Temp	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	…

例えば
クエリ : ボストン在住の20歳の男性は何人いるか？
処理　: Temp=条件2,4,5のAND
TempのBit(1)を数える → 2
これだけ！

標準 vs. ビットマップインデックス

- ビットマップインデックスは、インデックス対象フィールドの値の種類が少なく（**10000種類以下**）1つの値に多くのエントリを含むようなデータに向いています。
  - 選択性が大きいほど、1つのインデックス値に多くのエントリが登録されているため、ビットマップインデックスに向いています（選択性については後述します）。
- 選択性が小さい場合は、ビットマップインデックスより標準インデックスを使用します。
  - ユニークフィールドに対しては、標準インデックスを利用します（ビットマップインデックスは定義できません）。

選択性とは？

- 選択性＝フィールドのユニークデータの割合
  - **Where フィールド="値"** で全体の何%のレコードを抽出するか（概数）
- クエリ実行時にどのインデックスを使用するかの手がかりとなる値で以下のように利用されます。
  - 選択性＝1　はユニークフィールドを示し、優先的に使用されます。
  - それ以外のフィールドについては、<u>選択性の小さいものから使用されます。</u>

FCE.CurrencyOrder

テーブルチューニングは、テーブルのデータを参照して、エクステントサイズ(テーブルに何行
た、各 SQL マップのブロックカウントも計算します。

閉じる

選択性　マップ・ブロックカウント

現在のテーブルのエクステント・サイズ: 30 編集　テーブルチューニング

クラスを最新状態に保つ:

ページサイズ: 20　最大行数: 1000　結果: 18　ページ: 1 の 1

フィールド名	選択性	備考	外れ値の選択性	外れ値	平均フィールドサイ
AccountHolder	7.6923%				1.3
AmountIn	3.3333%				6.7
AmountOut	3.3333%				6.87
ATM	7.1429%				5.73
CurrencyIn	10.0000%				3
CurrencyOut	10.0000%				3
DateStamp	3.3333%				5
Employee	7.6923%				1.2
ExchangeRate	3.3333%				3.77
Fee	3.3333%				6.6
ID	1	RowID field			1.73
Payment	1				1.7
Requestor	3.3333%				16.6
RequestorEmai	3.3333%				4.9
Satisfaction	20.0000%				1
Status	14.2857%				7.3
TS	3.3333%				4.87
x__classname		Hidden field			

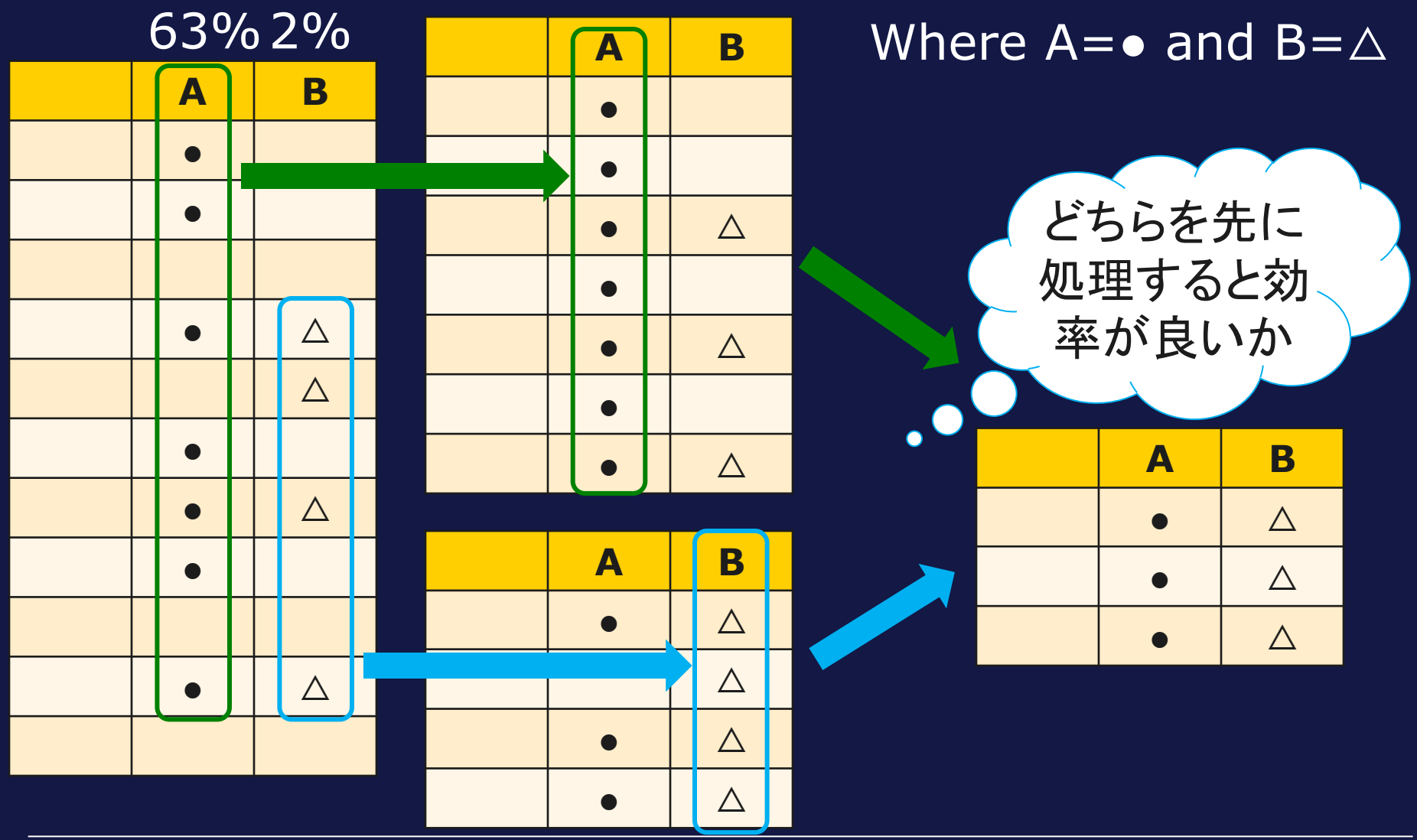
選択性の計算方法（2022.1以降）

- 自動的に実行されます。
  - 一度も実行したことがないテーブルに対して最初のクエリを実行する際、自動実行されます。
- 手動でも実行できます。
  - 管理ポータル→SQL→対象テーブルを選択→アクション→テーブルチューニング情報
  - $SYSTEM.SQL.Stats.Table.GatherTableStats(テーブル名)
    - 第1引数には、スキーマ名.テーブル名を指定します。
    - 全テーブルを対象とする場合は、第1引数に ＊ を指定します。
- テーブルチューニングでは、選択性の他に、エクステントサイズ（レコード総数）も算出されます。
  - エクステントサイズはテーブル結合を行う場合にどのテーブルから処理するかを決める手がかりになる情報です。
  - エクステントサイズが大きい＝コストが大きい を意味します。

ご参考：選択性の計算方法（～2021.1以前）

- 選択性は、「テーブルチューニング」を実施したときに計算されます。
  - 管理ポータル→SQL→対象テーブルを選択→アクション→テーブルチューニング情報
  - $system.SQL.TuneTable(”パッケージ名.クラス名”,1,1)
- テーブルチューニングでは、選択性の他に、エクステントサイズ（レコード総数）も算出されます。
  - **エクステントサイズはテーブル結合を行う場合にどのテーブルから処理するかを決める手がかりになる情報**です。
  - エクステントサイズが大きい＝コストが大きい　を意味します。

選択性によるインデックス決定の流れ



複数テーブルの結合の流れ

社員ID	名前	年齢	部署ID
EMP0001	山田太郎	25	3
EMP0002	鈴木次郎	48	2
EMP0003	浦島三郎	39	4
・・・			

←1000行

部署ID	部署名	グループ名
1	総務	東日本
2	営業	西日本
3	人事	西日本
4	経理	東日本
...		

↑
50行

- 部署が営業 で 40歳以上の人 を検索するとき、どちらのテーブルから処理するでしょうか？

エクステントサイズ×選択性の値が小さい方から処理されます

選択性　再計算のタイミング

- 選択性は、データの種類が急激に変化したりデータが急増することがない限り、再計算の必要はありません。
- データが大幅に増えたりデータの種類が急激に変化しても、実行しているクエリのパフォーマンスが悪化していなければ、再計算の必要はありません（悪化した場合にのみ、再計算を行ってください）。
  - 例）　データの種類が急激に変化する例
    0歳～7歳までのデータを扱っていたところ、0歳～80歳までのデータも取り扱うようになった。
- 性能調査の最初のステップとして選択性の計算を行ってください。
  - 理由：
    選択性が古い状態（または、未計算状態）のままであると、最適なクエリ実行プランではない可能性があります。

[外れ値の選択性]と[外れ値]

- フィールドに値がほとんど登録されていない(Null)や、特定の値がほとんどを占める場合、その値を**[外れ値]**として選択性計算時に算出されるようになりました。
  - 外れ値が全レコードの何%を占めているかの値は **[外れ値の選択性]** として記録します。
  - 外れ値の選択性により、クエリプランが変わる場合もあります。
  - 以下の選択性の計算結果では、FavoriteColors に未登録(Null)の値も多く含まれていることが検出されています(35.5 % が Null)。
    選択性の計算の中で、外れ値を検出した場合、以下の情報を表示します。
    [外れ値]にNull　[外れ値の選択性]に (Nullの件数/全体の件数)×100

テーブル: Sample.Person　テーブル情報　フィールド　インデックス　トリガー　制約　クエリキャッシュ

列	データタイプ	列 #	必須	ユニーク	照合	隠し	最大長	BLOB	コンテナ	選択性	xDBC型	参照先	バージョン列	外れ値の選択性	外れ値
ID	%Library.Integer	1	Yes	Yes		No		No		1	INTEGER		No		
Age	%Library.Integer	2	No	No		No		No		1.2048%	INTEGER		No		
DOB	%Library.Date	3	No	No		No		No		0.5025%	DATE		No		
FavoriteColors	%Library.String	4	No	No	SQLUPPER	No	50	No		1.4333%	VARCHAR		No	35.5%	<Null>
Home	Sample.Address	5	No	No		Yes		No		0.5000%	VARCHAR		No		

[外れ値の選択性]と[外れ値] Null以外の例

- 選択性の計算でNull以外の外れ値が算出されるケースもあります。

例えば、FCE.CurrencyOrderのStatusのほとんどに**"Completed"**が登録されている場合

FCE.CurrencyOrder　ネームスペース:USER　テーブルチューニング

テーブルチューニングは、テーブルのデータを参照して、エクステントサイズ(テーブルに何行含まれるか)と、各フィールドの選択性(特定の値に一致す
の割合)を返します。また、各 SQL マップのブロックカウントも計算します。

閉じる

選択性　マップ・ブロックカウント

現在のテーブルのエクステント・サイズ: 1030 編集 テーブルチューニング

クラスを最新状態に保つ:

ページサイズ: 20　最大行数: 1000　結果: 15　ページ: 1 の 1

	フィールド名	選択性	備考	外れ値の選択性	外れ値
	AmountIn	0.0971%			
	AmountOut	0.0971%			
	ATM	6.6577%			
	CurrencyIn	9.9996%			
	CurrencyOut	9.9996%			
	DateStamp	0.1085%			
	ExchangeRate	0.1210%			
	Fee	0.0971%			
	ID	1	RowID field		
	Requestor	0.0971%			
	RequestorEmail	0.0971%			
	Satisfaction	20.0000%			
»	Status	0.0992%		97.9167%	"Completed"
	TS	0.0971%			
	x__classname	0.0971%	Hidden field	99.2708%	<Null>

詳細

詳細を表示するアイテムを選択

Status

保存

選択性
0.0992%
選択性の値を入力してください

外れ値の選択性
97.9167%
> 0 から 99.999% のパーセントを入力し
だい。

外れ値
"Completed"
データの値を入力してください。文字列の
合は二重引用符を使用します

クエリオプティマイザがRTPCを使用する場合（2023.1以降）

- 2023.1以降では、RTPC（Runtime Plan Choice）＝実行時プラン選択の利用を推奨しています。
  - **2021.2以降のバージョンに追加された機能ですが2023.1以降での利用を推奨しています。**
- オプティマイザは**実行時にプランを選択**します。
  - WHEREの条件に外れ値ではない値を使用している場合、オプティマイザはインデックスを利用するプランを考慮します。
  - WHEREの条件に外れ値を使用している場合、オプティマイザはインデックスを使用しないプランを考慮します。
- どちらのプランもクエリキャッシュに保存されます。
- 特定のクエリだけ、RTPCを無効化する場合は**%NORUNTIME制約**キーワードをSELECT文に指定します。
- RTPCを無効化する場合は、管理ポータルの以下メニューを使用します。
  - **システム管理 > 構成 > SQLとオブジェクトの設定 > SQL >**

アダプティブモードをオフにして実行時間計画の選択、自動チューニング、クエリ計画の凍結/アップグレードを無効化 ☐

2024.1+

アダプティブモードをオフにして実行時プラン選択と自動チューニングを無効にする ☐

2023.1

クエリオプティマイザがRTPCを使用しない場合（または2021.1以前）

- 外れ値がNullの場合
  - Statusに値がほとんど登録されていないような場合、Nullが外れ値に検出されます。
  - クエリに「where Status is null」の条件が含まれていると、オプティマイザはほとんどの場合、Status用インデックスを使用せず結果を算出します。
- 外れ値がNull以外の場合（2023.1以降）
  - Stautsの値のほとんどが "Completed" である場合、"Completed" が外れ値に検出されます。
  - **ダイナミックSQLとODBC／JDBC経由のクエリで、条件に指定する値が外れ値で、オプティマイザにインデックスを使用させたくない場合は、指定値を (()) で囲う必要があります。**
    例） where Status = **((‘Completed’))**
    - クラス・クエリ、埋め込み **SQL** クエリ、またはビューにあるクエリについては、外れ値の選択性を常に使用する動作となるため、上記対応のような特別なコーディングは必要ありません。

並列クエリ

- オプティマイザは、クエリを分割実行するほうが良いプランとなるかどうか判断します。
  - 並列実行により効果が得られると考えられるSELECTクエリが対象です。
  - 複数のプロセッサを持つシステムでは、結果行は別々のプロセスで処理されます。
- （2019.1以降）無効に設定しない限り有効化モードで動作します。
  - 無効化は管理ポータルの以下メニューで行います。
    システム管理 > 構成 > SQLとオブジェクトの設定 > SQL > [単一プロセス内でクエリを実行]
- 2018.1以前では、並列クエリは %parallelキーワードを使用して有効化します。

ここまで確認できたこと

- 日々のクエリ実行統計データは自動的に収集されいるため、実行回数やクエリプランをいつでも確認できます。
  - 瞬間的な統計データを収集するツールも別途用意があるので遅い、重いクエリの特定には別ツールでの収集が可能です。
- アダプティブモードが有効化されている場合、実行時プラン選択（RTPC）が利用できるため、外れ値を検出した場合など最適なプラン選択をオプティマイザに任せることができます。

例えば、これ以上データが増えることがないので現状プランを使い続けたい場合や、選択性再計算があっても指定するまで今までのプランを使用したい場合、どうしたらいいでしょうか。

プランを変更しないように設定する「**凍結プラン**」が利用できます。

凍結プランとは

- 現在使用しているクエリプランを変更しないように凍結したプランを「凍結プラン」と呼びます。
- 凍結しないとどうなるのか：
  - インデックス追加など、定義変更に伴うコンパイルにより対象テーブルに対するクエリプランを含めたクエリキャッシュが破棄されます。
  - 選択性の再計算によりクエリプランが変わる可能性もあります。
  - InterSystems 製品アップグレード前後で別のクエリプランが作成される場合もあり、現状プランではないプランが利用されることもあります。
- アップグレード前後でプランの良し悪しを見極めたい場合などクエリプランを「凍結」させることで現プランの継続利用、また新プランとの比較が行えます。
  - プランのエクスポート／インポートも行えます。

自動的に凍結プランが作成されるケース（ある条件でのアップグレードのタイミングで作成されます）

- **2022.2より前バージョンからのアップグレード、**または**アダプティブモードを無効**に設定している環境でInterSystems製品のアップグレードを行うと**自動的に現プランは凍結されます。**

アップグレード前

#	テーブル名	プランの状態	新しいプラン	実行回数	実行回数/日	合計時間	平均時間	StdDev時間	行数	行数/日	実行されたコマンド	実行コマンド数/日	場所	ステートメント
1	Training.Employee	Unfrozen											%sqlcq.USER.cls3.1	DECLARE QRS CURSOR FOR SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL F
2	Training.Employee	Unfrozen											Training.Employee.1	INSERT INTO TRAINING . EMPLOYEE VALUES :val ( ) /*#OPTIONS {"IsolationLevel":0} *
3	Training.Employee	Unfrozen											%sqlcq.USER.cls2.1	DECLARE QRS CURSOR FOR SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL F

テーブル: Training.Employee ○テーブル情報 ○フィールド ○マップ/インデックス ○トリガ ○制約 ○クエリ・キャッシュ ◉テーブルの SQL 文

アップグレード後

アップグレード前のプランを自動的に凍結しています。
Frozen/Upgradeを表示されます。

#	テーブル名	プランの状態	新しいプラン	実行回数	実行コマンド数/日	場所	ステートメント
1	Training.Employee	Frozen/Upgrade					DECLARE QRS CURSOR FOR SELECT ID , DEPT , EMPID , LOCATION , NAME ,
2	Training.Employee	Unfrozen				Training.Employee.1	INSERT INTO TRAINING . EMPLOYEE VALUES :val ( ) /*#OPTIONS {"IsolationLev
3	Training.Employee	Frozen/Upgrade					DECLARE QRS CURSOR FOR SELECT ID , DEPT , EMPID , LOCATION , NAME ,

Frozen/Upgradeの例

SQLステートメントID 168
ステートメント・ハッシュ 34cHvjFIJSsdX8boKYsM01blxaQ=
最初に見た日付
プランの状態 凍結/アップグレード
合計時間
凍結プランが異なる いいえ
実行回数
平均時間
行数
実行回数/日 0
StdDev 時間
行数/日 0
実行されたコマンド
実行コマンド数/日 0

凍結プランと現プランが異なる場合「はい」と表示されます。（例は「いいえ」の表示）

コンパイル設定

選択モード Runtime
デフォルト・スキーマ SQLUSER
バージョン 2024.1.4.512
ユーザ名 SuperUser
スキーマ・パス
タイムスタンプ 2025-05-25 22:01:24
クライアントIPアドレス 172.21.0.1
クライアント名 localhost
クライアントアプリケーション CSPa24.so
コールスタック GetQueryParameters+14^%CSP.UI.Portal.SQL.Utils.1,PrepareQuery+4^%CSP.UI.Portal.SQL.Home.1,InvokeClassMethod+

ステートメントは以下のルーチンで定義されています

ルーチン	タイプ	最終コンパイル日時	実行時ルーチン
結果がありません			

ステートメントは以下のリレーションを使用します

テーブルまたはビューの名前	タイプ	最終コンパイル日時	クラス名	
Training.Employee	Table	05/25/2025 21:59:52.435955791	Training.Employee	クラスをコンパイル

凍結したクエリプランが表示されます。

ステートメントテキストとクエリプラン

ステートメント・テキスト

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ?
/*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"} */
```

警告

- The following System-wide SQL Configuration Setting, which impacts the SQL Query Plan, has been changed from its default value: AdaptiveMode: 0 (Disabled) [DEFAULT: 1 (Enabled)]
- Table Training.Employee is not tuned.

凍結したクエリ・プラン

Frozen Plan

相対コスト = 551.2

- Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(Name) and %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

ステートメント・テキスト

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ?
/*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"} */
```

警告

- The following System-wide SQL Configuration Setting, which impacts the SQL Query Plan, has been changed from its default value: AdaptiveMode: 0 (Disabled) [DEFAULT: 1 (Enabled)]
- Table Training.Employee is not tuned.

クエリプラン

相対コスト = 551.2
- Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(Name) and %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.

アダプティブモードが無効化されている＋Training.Employeeに対する選択性が未計算であることが警告として表示されています。（2024.3以降で追加された改善案の提示）

プランを凍結する方法

- SQL文の詳細画面の「プランを凍結」ボタンを利用します。

プランを凍結 | プランを凍結解除 | SQL 統計情報をクリア | エクスポート | ページを更新 | 閉じる

プランをエクスポートすることができます。
（凍結解除した後で元の凍結プランに戻したいときエクスポートファイルを利用してインポートできます。）

ステートメント詳細

SQLステートメントID	191	ステートメント・ハッシュ	34cHvjFlJSsdX8boKYsM01bIxaQ=		
プランの状態	凍結/明示	合計時間	0.033623	凍結プランが異なる	いいえ
実行回数	5	平均時間	0.006725	行数	6
実行回数/日	5	StdDev 時間	0.009367	行数/日	6
				実行されたコマンド	356695
				実行コマンド数/日	356695

コンパイル設定

選択モード	Runtime	デフォルト・スキーマ	SQLUSER	バージョン	2025.1.0.225
ユーザ名	SuperUser	スキーマ・パス		タイムスタンプ	2025-05-26 10:13:04
クライアントIPアドレス	172.21.0.1	クライアント名	localhost	クライアントアプリケーション	CSPa24.so

コールスタック GetQueryParameters+14^%CSP.UI.Portal.SQL.Utils.1,PrepareQuery+4^%CSP.UI.Portal.SQL.Home.1,InvokeClassMethod

ステートメントは以下のルーチンで定義されています

ルーチン	タイプ	最終コンパイル日時	実行時ルーチン	
%sqlcq.USER.cls2.1	Class Method	05/26/2025 10:13:04	%sqlcq.USER.cls2.1	-

ステートメントは以下のリレーションを使用します

テーブルまたはビューの名前	タイプ	最終コンパイル日時	クラス名	
Training.Employee	Table	05/26/2025 09:25:34.074626721	Training.Employee	クラスをコンパイル

このプランはインデックス未使用のプランです。

ステートメントテキストとクエリプラン

ステートメント・テキスト

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ?
/*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"} */
```

凍結したクエリ・プラン

Frozen Plan

相対コスト = 145800

- Divide master map Training.Employee.IDKEY into subranges of IDs.
- Call module A in parallel on each subrange, piping results into temp-file A.
- Read temp-file A, looping on a counter.
- For each row:
  - Output the row.

Module: A

- Read master map Training.Employee.IDKEY, looping on the subrange of ID.
- For each row:
  - Test the = condition on %SQLUPPER(Location), the = condition on %SQLUPPER(Name), the NOT NULL condition on %SQLUPPER(Location), and the NOT NULL condition on %SQLUPPER(Name).
  - Add a row to temp-file A, subscripted by a counter, with node data of Dept, EmpID, ID, Location, Name, and Tel.

ステートメント・テキスト

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ?
/*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"} */
```

クエリプラン

相対コスト = 145800

- Divide master map Training.Employee.IDKEY into subranges of IDs.
- Call module A in parallel on each subrange, piping results into temp-file A.
- Read temp-file A, looping on a counter.
- For each row:
  - Output the row.

Module: A

- Read master map Training.Employee.IDKEY, looping on the subrange of ID.
- For each row:
  - Test the = condition on %SQLUPPER(Location), the = condition on %SQLUPPER(Name), the NOT NULL condition on %SQLUPPER(Location), and the NOT NULL condition on %SQLUPPER(Name).
  - Add a row to temp-file A, subscripted by a counter, with node data of Dept, EmpID, ID, Location, Name, and Tel.

凍結プラン以外の新プランがある場合の表示

- [凍結プランが異なる]が「はい」と表示され画面下に両方のプランが表示されます。

プランを凍結 | プランを凍結解除 | SQL 統計情報をクリア | エクスポート | ページを更新 | 閉じる

ステートメント詳細

SQLステートメントID	191	ステートメント・ハッシュ	34cHyjFUSedX8bcKYsM01bkxQ=	最初に見た日付	2025-05-26
プランの状態	凍結/明示	合計時間	0.033623	凍結プランが異なる	はい
実行回数	5	平均時間	0.006725	行数	6
実行回数/日	5	StdDev 時間	0.009367	行数/日	6
				実行されたコマンド	356695
				実行コマンド数/日	356695

コンパイル設定

選択モード	Runtime	デフォルト・スキーマ	SQLUSER	バージョン	2025.1.0.225
ユーザ名	SuperUser	スキーマ・パス		タイムスタンプ	2025-05-26 10:13:04
クライアントIPアドレス	172.21.0.1	クライアント名	localhost	クライアントアプリケーション	CSPa24.so

コールスタック GetQueryParameters+14^%CSP.UI.Portal.SQL.Utils.1,PrepareQuery+4^%CSP.UI.Portal.SQL.Home.1,InvokeClassMethod

ステートメントは以下のルーチンで定義されています

ルーチン	タイプ	最終コンパイル日時	実行時ルーチン
結果がありません			

ステートメントは以下のリレーションを使用します

テーブルまたはビューの名前	タイプ	最終コンパイル日時	クラス名	
Training.Employee	Table	05/26/2025 11:18:48.649193764	Training.Employee	クラスをコンパイル

ステートメントテキストとクエリプラン

ステートメント・テキスト

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ?
/*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"} */
```

警告

- Table Training.Employee is not tuned.

凍結したクエリ・プラン

Frozen Plan

相対コスト = 145800

- Divide master map Training.Employee.IDKEY into subranges of IDs.
- Call module A in parallel on each subrange, piping results into temp-file A.
- Read temp-file A, looping on a counter.
- For each row:
  - Output the row.

Module: A

- Read master map Training.Employee.IDKEY, looping on the subrange of ID.
- For each row:
  - Test the = condition on %SQLUPPER(Location), the = condition on %SQLUPPER(Name), the NOT NULL condition on %SQLUPPER(Location), and the NOT NULL condition on %SQLUPPER(Name).
  - Add a row to temp-file A, subscripted by a counter, with node data of Dept, EmpID, ID, Location, Name, and Tel.

ステートメント

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM T[illegible] LOYEE WHERE LOCATION = ? AND NAME = ?
/*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"Dy[illegible]List":"1,1"} */
```

警告

- Table Training.Employee is not tuned.

クエリプラン

相対コスト = 551.2

- Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(Name) and %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

新プランの相対コストが低い数値に変わっています。

ご参考：プランを凍結解除した場合の表示

プランを凍結 | プランを凍結解除 | SQL 統計情報をクリア | エクスポート | ページを更新 | 閉じる

ステートメント詳細

SQLステートメントID	191	ステートメント・ハッシュ	34cHvjFIJSsdX8boKYsM01bIxaQ=	最初に見た日付	2025-05-26
プランの状態	凍結解除	合計時間	0.033623	行数	6
実行回数	5	平均時間	0.006725	行数/日	6
実行回数/日	5	StdDev 時間	0.009367	実行されたコマンド	356695
				実行コマンド数/日	356695

コンパイル設定

選択モード	Runtime	デフォルト・スキーマ	SQLUSER	バージョン	2025.1.0.225
ユーザ名	SuperUser	スキーマ・パス		タイムスタンプ	2025-05-26 10:13:04
クライアントIPアドレス	172.21.0.1	クライアント名	localhost	クライアントアプリケーション	CSPa24.so

コールスタック GetQueryParameters+14^%CSP.UI.Portal.SQL.Utils.1,PrepareQuery+4^%CSP.UI.Portal.SQL.Home.1,InvokeClassMethod

ステートメントは以下のルーチンで定義されています

ルーチン	タイプ	最終コンパイル日時	実行時ルーチン
結果がありません			

ステートメントは以下のリレーションを使用します

テーブルまたはビューの名前	タイプ	最終コンパイル日時	クラス名	
Training.Employee	Table	05/26/2025 11:18:48.649193764	Training.Employee	クラスをコンパイル

ステートメントテキストとクエリプラン

ステートメント・テキスト

```
SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ? /*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"
```

警告

- Table Training.Employee is not tuned.

クエリプラン

相対コスト = 551.2

- Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(Name) and %SQLUPPER(Location), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Output the row.

新プランがあるかどうかをチェックする方法

- 以下SELECT文の**FrozenDifferentの値が1**の場合、新プランが存在します。

```
SELECT Frozen,FrozenDifferent,Timestamp,Statement
FROM INFORMATION_SCHEMA.STATEMENTS
WHERE Frozen=1 OR Frozen=2
```

メモ：未凍結プランの場合の文は Frozen=0 または Frozen=3

- 例）前ページの新プランがある場合の実行結果

実行 | プラン表示 | 履歴を表示 | クエリビルダ | 表示モード | 最大 1000 | その他オプション

```
SELECT Frozen,FrozenDifferent,Timestamp,Statement
FROM INFORMATION_SCHEMA.STATEMENTS
WHERE Frozen=1 OR Frozen=2
```

行数: **1** パフォーマンス: **0.0029** 秒 **760** グローバル参照 **4275** 実行されたコマンド **0** ディスク読み込みレイテンシ (ms) クエリ:%sqlcq.USER.cls13 最終更新:2025-05-26 17:33:43.673

印刷

Frozen	FrozenDifferent	Timestamp	Statement
1	1	2025-05-26 16:07:39.703511591	DECLARE QRS CURSOR FOR SELECT ID , DEPT , EMPID , LOCATION , NAME , TEL FROM TRAINING . EMPLOYEE WHERE LOCATION = ? AND NAME = ? /*#OPTIONS {"IsolationLevel":0} */ /*#OPTIONS {"DynamicSQLTypeList":"1,1"} */

プランのエクスポート／インポート

- クエリプランはエクスポート／インポートが行えます。
  - 凍結解除後に前のプランを再度利用したい場合にエクスポートしたプランをインポートすることで対応できます。
- 管理ポータルでは以下のメニューで行います。
  - エクスポートは**SQL文の詳細画面のエクスポートボタン**を利用します。
  - インポートは、管理ポータルのSQL画面内 **[アクション] > [ステートメントをインポート]** を利用します。
- メソッドでの実行は以下の通りです。
  - エクスポート：$SYSTEM.SQL.Statement.ExportFrozenPlans()
    - 第1引数：ファイル名
    - 第2引数：ハッシュ値（SQL文詳細画面で確認できます）
    - 戻り値　：%Status
  - インポート：$SYSTEM.SQL.Statement.ImportFrozenPlans()
    - 第1引数：ファイル名
    - 戻り値　：%Boolean
  - メソッドでは全凍結プランのエクスポートも行えます。

$SYSTEM.SQL.Statement.ExportAllFrozenPlans()

ご参考：プランの状態

- アダプティブモードが無効化された状態での InterSystems 製品アップグレードにより、自動的に凍結されたプランや未凍結プランなど「プランの状態」は、以下のように表示されます。
  - Unfrozen ：凍結されていませんが、凍結できます。
  - Unfrozen/Parallel ：凍結されておらず、凍結することもできません。
  - Frozen/Explicit ：ユーザによって凍結されていますが、未凍結にすることができます。
  - Frozen/Upgrade ：InterSystems 製品バージョンのアップグレードによって凍結されていますが、凍結解除できます。

新プランを試したいとき：%NOFPLAN

- 凍結プランがあっても **%NOFPLANキーワード** を利用することで新しいプランを試すことができます。

```
SELECT %NOFPLAN ID, Dept, EmpID, Location, Name, Tel
FROM Training.Employee
where Location='北海道' AND Name ='高木'
```

実行プランが以下に表示されます:

ステートメント・テキスト

```
SELECT %NOFPLAN ID , Dept , EmpID , Location , Name , Tel FROM Training . Employee WHERE Location = ? AND Name = ? /*#OPTIONS
{"DynamicSQLTypeList":"1,1"} */
```

警告

• Table Training.Employee is not tuned.

クエリプラン

相対コスト = 551.2

• Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(Name) and %SQLUPPER(Location), and looping on ID.
• For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.

Frozen Plan

相対コスト = 145800

• Divide master map Training.Employee.IDKEY into subranges of IDs.
• Call module A in parallel on each subrange, piping results into temp-file A.
• Read temp-file A, looping on a counter.
• For each row:
  - Output the row.

%NOFPLAN未使用時のプラン（凍結プラン）

Module: A

• Read master map Training.Employee.IDKEY, looping on the subrange of ID.
• For each row:
  - Test the = condition on %SQLUPPER(Location), the = condition on %SQLUPPER(Name), the NOT NULL condition on %SQLUPPER(Location), and the NOT NULL condition on %SQLUPPER(Name).
  - Add a row to temp-file A, subscripted by a counter, with node data of Dept, EmpID, ID, Location, Name, and Tel.

別プランを試したいとき:
[ツール] > [別のプランを表示]メニューを利用

SQL ステートメントに対してSQLオプティマイザが生成する代替えプランをレビューするためにこのページのオプションを使用してください。

SQL ステートメントを入力して 'プランのオプションを表示' をクリックしてください。結果テーブルにSQLオプティマイザが生成した異なるプランが表示されます。

☐ **'プラン表示オプション' または '比較' をバックグラウンドで実行**
(実行に長時間を要する大きなクエリに対して強く推奨します)

SQL 文:

```
SELECT ID, Dept, EmpID, Location, Name, Tel
FROM Training.Employee
where Location='北海道' AND Name ='高木'
```

プラン表示のオプション | 履歴を表示

利用可能なプラン
(以下のテーブルから複数のIDを選択して、'プランの比較' ボタンをクリックすることができます。)

	☐	ID	Cost	マップ・タイプ	開始マップ		
	☐	1	544.6	index map	Training.Employee.NameLocationIdx	プラン表示	統計情報付きでプランを表示
	☐	2	1363.8	index map	Training.Employee.NameLocationIdx	プラン表示	統計情報付きでプランを表示
»	☐	3	13430.0	index map	Training.Employee.NameIdx	プラン表示	統計情報付きでプランを表示
	☐	4	14249.0	index map	Training.Employee.NameIdx	プラン表示	統計情報付きでプランを表示
	☐	5	53952.0	index map	Training.Employee.LocationIdx	プラン表示	統計情報付きでプランを表示
	☐	6	54771.0	index map	Training.Employee.LocationIdx	プラン表示	統計情報付きでプランを表示
	☐	7	145800.0	master map	Training.Employee.IDKEY	プラン表示	統計情報付きでプランを表示
	☐	8	146619.0	master map	Training.Employee.IDKEY	プラン表示	統計情報付きでプランを表示

Stats でプラン表示を比較

利用可能なプランを比較

	ID	Cost	開始マップ	グローバル参照	Commands	合計時間	返された行数	
	1	544.6	Training.Employee.NameLocationIdx	2	94	0.000035	1	統計情報付きでプランを表示
	3	13430.0	Training.Employee.NameIdx	2	94	0.000042	1	統計情報付きでプランを表示
»	5	53952.0	Training.Employee.LocationIdx	4	105	0.000064	1	統計情報付きでプランを表示

ステートメント・テキスト

SELECT ID , Dept , EmpID , Location , Name , Tel FROM Training . Employee WHERE Location = ? AND Name = ?

クエリプラン

相対コスト = 13430

Module	Rows Returned	Performance (secs)	Global Refs	Commands Exec	Read Latency (ms)
MAIN	0	0.000015	0	39	0

Module	Rows Returned	Performance (secs)	Global Refs	Commands Exec	Read Latency (ms)
FIRST		0	0	0	0

Module	Rows Returned	Performance (secs)	Global Refs	Commands Exec	Read Latency (ms)
B		0.000002	0	18	0

- Read index map Training.Employee.NameIdx, using the given %SQLUPPER(Name), and looping on ID.
- For each row:
  - Read master map Training.Employee.IDKEY, using the given idkey value.
  - Test the = condition on %SQLUPPER(Location) and the NOT NULL condition on %SQLUPPER(Location).
  - Output the row.

ここからは・・・

- SQL実行時統計情報から問題となるクエリが特定でき、テーブルチューニングの実施（または実施無）を確認できた後でできる事をご説明します。
  - インデックス追加によるクエリの最適化
  - クエリ最適化オプション（＝オプティマイザ・ヒント）の利用

インデックス定義方法

- 新規プロパティウィザードでインデックスまたはユニークオプションをチェックするとプロパティ名に合わせてインデックスが自動的に作成されます。

  Index IndexName on PropertyName;

- 作成済みプロパティに対してインデックスを追加するには、ウィザードを使用するか、エディタウィンドウに直接定義を入力します。
- SQLのCreate Index文でクラスにインデックスを作成できます。
  - クラス定義に対して実行する場合は、Create Index文実行前に、クラス定義のDdlAllowed属性を追加する必要があります。
  - Create Index文での作成は、<u>既存データに対するインデックス構築も同時に行います。</u>

    CREATE [UNIQUE | BITMAP] INDEX IndexName
    ON TableName (ColumnName , ...)

インデックス定義方法

- クラス定義の{}内にIndex定義文を追加します。
  - Unique Indexはユニーク属性を付けたいときに利用すると定義が簡単です。

```
Ind
  Index
}  Index                                   Unique Index
```

```
11    /// Description
12    Index IndexName On property [Unique];
```

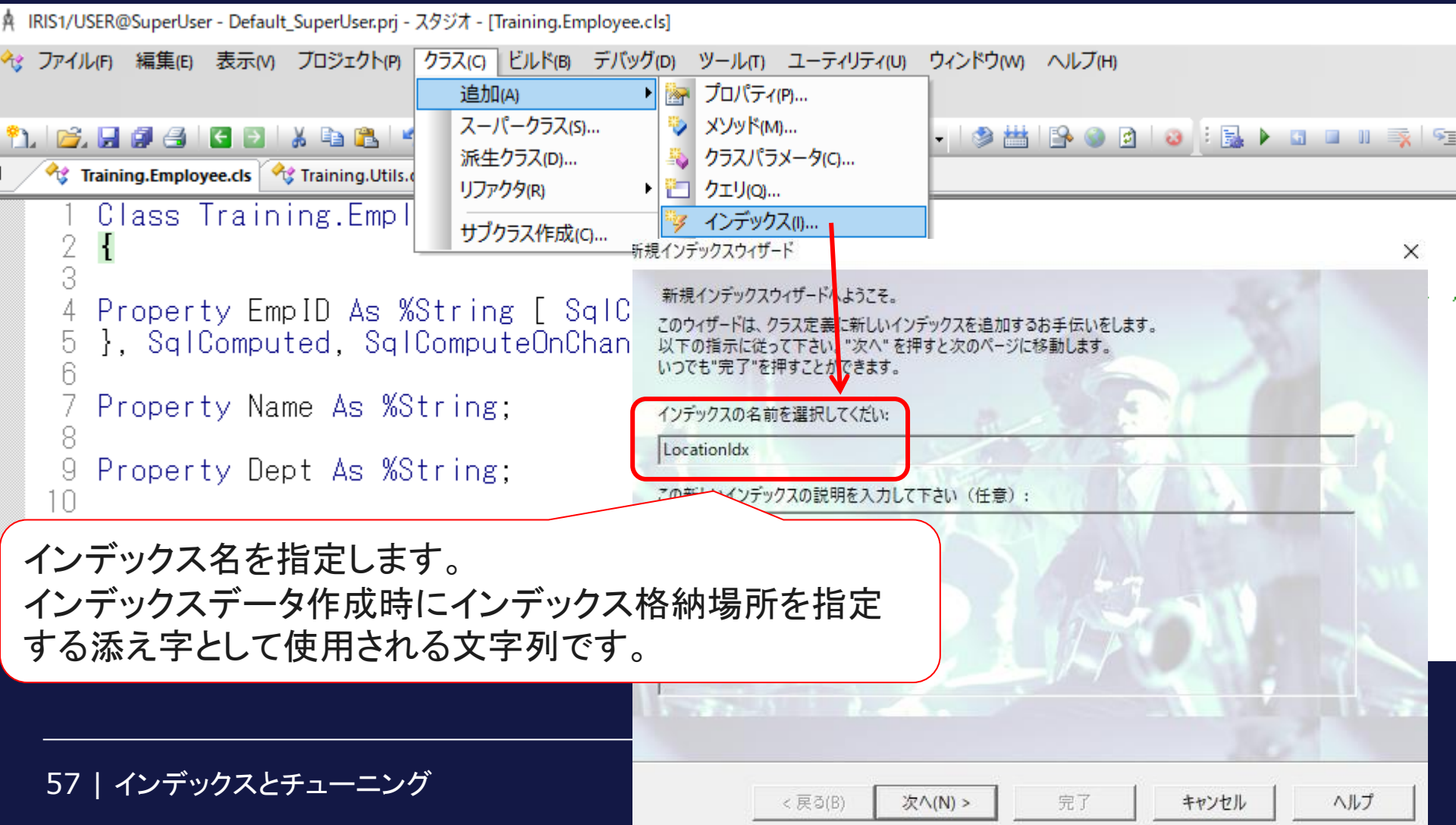
Uniqueインデックスを指定した場合、Unique属性が付与されます。

インデックス名を指定します。インデックスデータ作成時にインデックス格納場所を指定する添え字として使用される文字列です。

インデックス対象プロパティをOnの後ろに指定します。

スタジオ：インデックス定義方法（ウィザードの使用　1）

- インデックスをウィザードで追加する方法は以下の通りです。



インデックス名を指定します。
インデックスデータ作成時にインデックス格納場所を指定する添え字として使用される文字列です。

スタジオ：インデックス定義方法（ウィザードの使用　2）

- インデックスの種類や、インデックス対象プロパティを選択します。

新規インデックスウィザード

インデックスタイプ

このインデックスは:

通常: 1 つ以上のプロパティのインデックスを維持するために使用さ

これはユニークなインデックスです

これはこのクラスのIDKEYです

これはこのクラスのSQL主キーです

エクステント: エクステント内のこのクラスの全オブジェクトのインデック

その他

このインデックスの実装:

標準インデックス

ビットマップインデックス

ビットスライスインデックス

< 戻る(B)　次へ(N) >　完了

新規インデックスウィザード

インデックスプロパティ

このインデックスの基礎となる 1 つ以上のプロパティのリストを入力して下さい.

プロパティ	照合	照合パラメータ
Location		

属性

プロパティ: Location

照合:

照合パラメータ:

OK　キャンセル

< 戻る(B)　次へ(N) >　完了　キャンセル　ヘルプ

インデックスの維持管理

- インデックスの追加とコンパイルを行った後は、オブジェクト／行の追加、変更作業で自動的にインデックスデータが更新されます。
- **既存データがあるクラス定義に対して途中からインデックスを追加した場合は、インデックスの再構築が必要です。**
  - アプリケーションの停止時間など（ユーザがいない時間）に行います。
    - 補足（後述）
      %IndexBuilderクラスを使用することで、アプリケーション使用中であっても、複数のプロセスを使用してインデックスの再構築が行えます。
- **【注意】Create Indexを使用した場合は、既存データに対するインデックス構築も同時に行います。**

インデックスの再構築

- インデックスの再構築を管理ポータルで行う方法は以下の通りです。
  - テーブル内全インデックスを再構築する方法
    システムエクスプローラ→SQL→スキーマを選択→テーブルを選択→アクション→インデックス再構築
  - テーブルの1つのインデックスを再構築する方法
    システムエクスプローラ→SQL→スキーマを選択→テーブルを選択→ラジオボタン：インデックス→（該当するインデックスの）インデックス再構築
- ターミナルでの実行方法は以下の通りです。
  do ##class(Package.Class).%BuildIndices(list)
  - 引数には、ObjectScriptの$LISTBUILD関数（省略系：$LB()）で作成したインデックス名のリストを指定します。
  - 例） NameIndexとPhoneIndexを再構築したい場合

do ##*class(Package.Class)*.%BuildIndices($LB(”NameIndex”,”LocationIndex”))

  - インデックス名を指定してインデックスデータを削除したい場合
    do ##class(Package.Class).%PurgeIndices(list)
- SQLで実行する場合はBUILD INDEXも利用できます（2024.1以降）。

インデックスの再構築（管理ポータル）

- 管理ポータルでインデックス一覧を参照／再構築が行えます。
  - **システムエクスプローラ→SQL→スキーマ選択→テーブル選択→マップ／インデックス**

カタログの詳細 | クエリ実行 | 参照 | SQLステートメント

テーブル: Training.Employee ○テーブル情報 ○フィールド ◉マップ/インデックス ○トリガ ○制約 ○クエリ・キャッシュ ○

インデックス名	SQL マップ名	列	ブロックカウント	マップを継承?	グローバル	
$Employee	$Employee					
CapabilityIdx	CapabilityIdx	Capability	t 6 (Estimated)	No	^Training.Employee(("$Employee")	インデックス再構築
DeptIdx	DeptIdx	$$SQLUPPER({	4 (Measured)	No	^Training.Employee(("CapabilityIdx")	インデックス再構築
EmplDIdx	EmplDIdx	$$SQLUPPER({	375 (Estimated)	No	^Training.Employee(("DeptIdx")	インデックス再構築
IDKEY	IDKEY	ID	8 (Measured)	No	^Training.Employee(("EmplDIdx")	インデックス再構築
LocationIdx	LocationIdx	$$SQLUPPER({	16 (Measured)	No	^Training.EmployeeD	インデックス再構築
NameIdx	NameIdx	$$SQLUPPER({	8 (Measured)	No	^Training.Employee(("LocationIdx")	インデックス再構築
NameLocationIdx	NameLocationIdx	$$SQLUPPER({	4 (Measured)	No	^Training.Employee(("NameIdx")	インデックス再構築
			20 (Measured)	No	^Training.Employee(("NameLocationIdx")	インデックス再構築

インデックス再構築の注意点

- インデックス定義を追加し、クラス定義をコンパイルすると該当クラスのクエリキャッシュも同時に削除されます。
  - 動的発行のSQLが実行されると、新しいインデックス定義を利用してクエリキャッシュが再作成されます。（※インデックスを使うかどうかはクエリオプティマイザが判断するため、使用されない場合もあります。）
  - インデックス再構築前では、既存データに対してインデックスデータは存在しません。 つまり

オプティマイザがインデックスを使用する経路を選択した場合、検索結果0件の状態が発生します。

- 対策
  - インデックス定義前に、クエリオプティマイザから定義予定のインデックス名を隠します。
  - インデックス定義のコンパイル＋再構築が完了してから、クエリオプティマイザに新インデックスを見せるように変更します。

インデックス再構築例

- インデックス再構築途中の不完全なデータ参照を防ぐため、クエリオプティマイザにインデックスを使用させないように設定してから再構築を行います。
- バージョン別で利用方法が異なるため、以降のページではバージョン別再構築手順例を示します。
  - 2024.1以降の方法
  - 2022.1～2023.1までの方法
  - 2021.1以前の方法

クエリオプティマイザにインデックスを使用させない方法 DEFERオプションの利用 《2024.1以降》

- CREATE INDEXのDEFERオプションを使用します。
  - 注意1：DEFERオプションを付けないCREATE INDEX文では、作成時にインデックスの再構築も同時に行われます。
  - 注意2：**永続クラス定義はDDL文の発行がデフォルトで許可されていません。**クラス定義文の属性**DdlAllowed**を追加することでDDL文を実行できるようになります。
    - 例）Class Training.Employee Extends %Persistent [**DdlAllowed**]
- DEFERオプションを使用することで、インデックス定義は追加されますが、クエリオプティマイザがそのインデックスを使用しないように**「選択不可能」**と設定されます。
  - 管理ポータルで確認できます。

CREATE INDEX LocationIdx On Training.Employee (Location) **DEFER**

カタログの詳細 | クエリ実行 | 参照 | SQLステートメント

テーブル: Training.Employee ○ テーブル情報 ○ フィールド ◉ マップ/インデックス ○ トリガ ○ 制約 ○ クエリ・キャッシュ ○ テーブルの SQL 文

インデックス名	SQL マップ名	列	タイプ	ブロックカウント	マップを継承?	グローバル	ステータス	
CapabilityIdx	CapabilityIdx	Capability	Index	28 (Measured)	No	^Training.EmployeeI("CapabilityIdx")	選択可能	インデックス再構築
EmpIDIdx	EmpIDIdx	$$SQLUPPER({Training.Employee.EmpID})	Unique	48 (Measured)	No	^Training.EmployeeI("EmpIDIdx")	選択可能	インデックス再構築
IDKEY	IDKEY	ID	Data/Master	164 (Measured)	No	^Training.EmployeeD	選択可能	インデックス再構築
LocationIdx	LocationIdx	$$SQLUPPER({Training.Employee.Location})	Index	60 (Measured)	No	^Training.EmployeeI("LocationIdx")	選択不可能	インデックス再構築

クエリオプティマイザにインデックスを使用させない方法 DEFERオプション：つづき 《2024.1以降》

- DEFERオプションを付けて定義したインデックスに対して<u>BUILD INDEX</u>文を利用してインデックスを再構築します。
  - BUILD INDEXを利用することで、再構築が終了すると同時に追加したインデックスが<u>**「選択可能」**</u>に自動的に設定されます。

BUILD INDEX FOR TABLE Training.Employee INDEX **LocationIdx**

テーブル: Training.Employee ○ テーブル情報 ○ フィールド ◉ マップ/インデックス ○ トリガ ○ 制約 ○ クエリ・キャッシュ ○ テーブルの SQL 文

インデックス名	SQL マップ名	列	タイプ	ブロックカウント	マップを継承?	グローバル	ステータス	
CapabilityIdx	CapabilityIdx	Capability	Index	28 (Measured)	No	^Training.EmployeeI("CapabilityIdx")	選択可能	インデックス再構
EmpIDIdx	EmpIDIdx	$$SQLUPPER({Training.Employee.EmpID})	Unique	48 (Measured)	No	^Training.EmployeeI("EmpIDIdx")	選択可能	インデックス再構
IDKEY	IDKEY	ID	Data/Master	164 (Measured)	No	^Training.EmployeeD	選択可能	インデックス再構
LocationIdx	LocationIdx	$$SQLUPPER({Training.Employee.Location})	Index	60 (Measured)	No	^Training.EmployeeI("LocationIdx")	選択可能	インデックス再構

クエリオプティマイザにインデックスを使用させない方法 DEFERオプション：手順まとめ 《2024.1以降》

1. 永続クラスを作成している場合は、クラス定義文にDdlAllow属性を追加＋コンパイルを実行し、DDL文を発行できるように準備します。

定義例：Class Training.Employee Extends %Persistent [**DdlAllowed**]

2. 追加したいインデックス定義をCREAT INDEX文のDEFERオプションを利用して追加します。

CREATE INDEX LocationIdx On Training.Employee (Location) **DEFER**

3. BUILD INDEX文を利用してインデックスの再構築を行います。

BUILD INDEX FOR TABLE Training.Employee INDEX **LocationIdx**

クエリオプティマイザにインデックスを使用させない方法《2022.1～2023.1》

1. 新インデックス名をクエリオプティマイザが使用しないように以下メソッドを実行します。

```
$system.SQL.Util.SetMapSelectability("Training.Employee","NewIndex",0)
```

   - 第1引数：クラス名
   - 第2引数：インデックス名（これから指定する新インデックス名を指定します。）
   - 第3引数：隠す場合は0、見せる場合は1
2. クエリオプティマイザからインデックス名を隠している間にインデックス定義を追加＋コンパイルします。
3. インデックスの再構築します。
4. クエリオプティマイザにインデックスを見せるように変更します。

```
$system.SQL.Util.SetMapSelectability("Training.Employee","NewIndex",1)
```

5. クエリキャッシュを削除します。

クエリオプティマイザにインデックスを使用させない方法《2021.1以前》

1. 新インデックス名をクエリオプティマイザが使用しないように以下メソッドを実行します。

```
$system.SQL.SetMapSelectability("Training.Employee","NewIndex",0)
```

   - 第1引数：クラス名
   - 第2引数：インデックス名（これから指定する新インデックス名を指定します。）
   - 第3引数：隠す場合は0、見せる場合は1
2. クエリオプティマイザからインデックス名を隠している間にインデックス定義を追加＋コンパイルします。
3. インデックスの再構築します。
4. クエリオプティマイザにインデックスを見せるように変更します。

```
$system.SQL.SetMapSelectability(" Training.Employee","NewIndex",1)
```

5. クエリキャッシュを削除します。

補足（2022.1以降の記述）
複数プロセスでインデックス再構築を行う方法

- 以下の手順でインデックスを持つクラス定義を修正する必要があります。

1. 再構築処理中にクエリオプティマイザが新しいインデックス定義を使用しないように、オプティマイザから新インデックス定義を隠します（以下メソッドの第3引数に 0 を指定します）。

```
set st=$system.SQL.Util.SetMapSelectability("FCE.CurrencyOrder","NewIndex",0)
```

2. 新インデックスを定義したクラス定義に以下スーパークラスを追加します。
   %IndexBuilder
3. 2のクラス定義で、**INDEXBUILDERFILTERパラメータに再構築予定のインデックス名を登録**します。
4. クラス定義をコンパイルします。
   この時点で 手順1の指示により、**コンパイル後もクエリオプティマイザは新しいインデックス定義を使用しません。**
5. **%ConstructIndicesParallel()** メソッドを使用して、並列処理でインデックスの再構築を行います。

```
set st=##class(FCE.CurrencyOrder).%ConstructIndicesParallel()
```

6. インデックス再構築が終わったら、クエリオプティマイザに新インデックスを利用させるよう、手順1の解除を行います。

```
set st=$system.SQL.Util.SetMapSelectability("FCE.CurrencyOrder","NewIndex",1)
```

ご参考：クエリの最適化オプション（オプティマイザ・ヒント）

- FROM節に指定できるオプションで、クエリオプティマイザに以下のような指示を与えることができます。
  - JOINの順序指定
  - インデックスの指示
  - サブクエリの平坦化抑制　など

例：%IGNOREINDEX

```
24 Index LocationIdx On Location;
25
26 Index NameIdx On Name [ Type = bitmap ];
27
28 Index NameLocationIdx On (Name, Location) [ Data = (Dept, Tel, Name) ];
29
```

インデックスの定義

```
select ID,Name,Dept,Tel from Training.Employee
Where Location='北海道' and Name %Startswith '青'
```

NameLocationIdxを使用しているプラン

行数: 1 パフォーマンス: 0.003 秒 329 グローバル参照 2538 実行されたコマ　　ク読み込みレイテン

相対コスト = 37674

- Read index map Training.Employee.NameLocationIdx, using the given %SQLUPPER(condition) and ID.
- For each row:
  - Output the row.

%IGNOREINDEX
Training.Employee.NameLocationIdx

```
select ID,Name,Dept,Tel
from %IGNOREINDEX Training.Employee.NameLocationidx Training.Emplo
Where Location='北海道' and Name %Startswith '青'
```

行数: 1 パフォーマンス: 0.003 秒 350 グローバル参照 3265 実行されたコマンド 0 ディスク読み込み

相対コスト = 85658

- Call module C once, which populates bitmap temp-file A.
- Generate a stream of idkey values using the multi-index combination:
  ((index map Training.Employee.LocationIdx) INTERSECT (bitmap temp-file A))
- For each idkey value:
  - Read master map Training.Employee.IDKEY, using the given idkey value
  - Output the row.

NameLocationIdxが使用されないプランに変わったことがわかります。

- Read bitmap index Training.Employee.NameIdx, looping on %SQLUPPER(Name) (with a %STARTSWITH range conditi
- For each bitmap chunk:
  - OR the bitmap chunk into bitmap temp-file A.

ご参考:オプティマイザ・ヒント

ヒント	内容
%ALLINDEX	JOIN順の最初のテーブルに対して適用可能な全てのインデックスを使用する。WHERE節に%NOINDEXを指定して明示的に例外を指定可能。
%NOFLATTEN	サブクエリのFROM節に指定してサブクエリをメインクエリの条件に変換することを抑制する。
%NOSVSO	サブクエリのFROM節に指定して"Set-Valued Subquery Optimaization"を抑制する。SVSO: [NOT] EXISTS または [NOT] IN サブクエリの条件から一時インデックスを作成してメインクエリの条件指定に変換する。
%NOUNIONOROPT	OR ⇔ サブクエリでのUNION変換による最適化を抑止する。
%NOTOPOPT	TOP ・・・ ORDER BYにおいて最初の行への時間を優先する最適化を抑止し、全体の結果セット取得時間を優先する。
%FIRSTTABLE	指定したテーブルを最初にJOINする。
%FULL	全てのJOIN順を計算し、最適なプランを算出する(コンパイル時間が長くなる)。ストアドプロシージャ内のクエリに有効な場合がある。
%IGNOREINDEX	指定されたインデックスを使わない。
%INORDER	FROM説に指定されたテーブルの順序でJOINする。
%STARTTABLE	FROM節の最初のテーブルからJOINする
%NOINDEX	指定したWHERE節に対してインデックスを使用しない.

付録

選択性のエクスポート

- 既存の選択性とエクステントサイズのエクスポート方法は以下の通りです。
- **$system.SQL.Stats.Table.Export()**
  - 第1引数：エクスポート時のファイル名をフルパスで指定します。
  - 第2引数：エクスポート対象のスキーマ名を指定します（指定しない場合は全てのスキーマが対象）。
  - 第3引数：エクスポート対象のテーブル名を指定します（指定しない場合は全てのスキーマか第2引数で指定されたスキーマ配下の全テーブルが対象）。
  - 第4引数：結果を表示するかしないかの指定（デフォルトは1：表示する／0：表示しない）

```
USER>do $system.SQL.Stats.Table.Export ("c:¥kit¥emp.xml","Training","Employee",1)
01/16/2018 22:57:00 の SQL テーブルチューニング統計情報をファイル c:¥kit¥emp.xml
にエクスポートしています
テーブル Training.Employee のテーブルチューニング統計情報をエクスポートしています

エクスポートが .019736 秒で完了しました
```

選択性のインポート

ご参考：テーブル統計情報をエクスポートして別環境にインポートする方法

- 選択性とエクステントサイズのエクスポートファイルのインポート方法は以下の通りです。
- **$system.SQL.Stats.Table.Import()**
  - 第1引数：インポートファイル名をフルパスで指定します。
  - 第2引数：結果を表示するかしないかの指定（デフォルトは1：表示する／0：表示しない）
  - 第3引数：インポート対象クラス更新後にクラスの最新状態を維持するかしないかの指定（デフォルトは0：維持しない／1：維持する）

```
USER>do $system.SQL.Stats.Table.Import("c:¥kit¥emp.xml",1,1)

01/16/2018 22:59:55 でファイル c:¥kit¥emp.xml から SQL テーブルチューニング情報
をインポートしています。クラスを最新状態に保つ = 1, 現在のチューニング情報をクリア = 0
テーブル Training.Employee のテーブルチューニング統計情報をインポートしています

インポートが .163012 秒で完了しました
```

ビットマップインデックスの圧縮
圧縮を必要とする例

- 多くのINSERTやDELETEが発生するテーブルに対してビットマップインデックスを使用している場合、徐々に効率が低下する可能性があります。

```
Class MyWork.MonthData Extends (%Persistent, %Populate)
{
/// 満足度
Property Satisfaction As %String(VALUELIST = ",満足,やや満足,やや不満,不満,");
/// 年齢
Property Age As %Integer(MAXVAL = 70, MINVAL = 20);
Index AgeIdx On Age [ Type = bitmap ];
}
```

【INSERT時】

```
^MyWork.MonthDataI("AgeIdx",20,1) = $zwc(401,120,4,75,102,10,＜省略＞ 958)/*$bit(5,76,103,107･･･
^MyWork.MonthDataI("AgeIdx",21,1) = $zwc(407,121,29,178,251,2＜省略＞,732,772,898,960)/*$bit(3･･･
^MyWork.MonthDataI("AgeIdx",22,1) = $zwc(402,96,5,57,74,164,＜省略＞,0,4)/*$bit(20,63,77,92,10･･･
^MyWork.MonthDataI("AgeIdx",23,1) = $zwc(133,116)_$c(0,0,8,0＜省略＞,64,0,4)/*$bit(20,63,77,92･･･
^MyWork.MonthDataI("AgeIdx",25,1) = $zwc(404,119,105,155,235＜省略＞,947)/*$bit(106,156,236,30･･･
^MyWork.MonthDataI("AgeIdx",26,1) = $zwc(128,119)_$c(0,0,0,2,＜省略＞,0,128)/*$bit(26,80,115,1･･･
 ＜以下省略＞
```

【TRUNCATE後】

```
^MyWork.MonthDataI("AgeIdx",20,1)  =  $zwc(145,120)/*$bit()*/
^MyWork.MonthDataI("AgeIdx",21,1)  =  $zwc(151,121)/*$bit()*/
^MyWork.MonthDataI("AgeIdx",22,1)  =  $zwc(146,96)/*$bit()*/
^MyWork.MonthDataI("AgeIdx",23,1)  =  $zwc(133,116)_$c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,･･･
^MyWork.MonthDataI("AgeIdx",24,1)  =  $zwc(131,125)_$c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,･･･
^MyWork.MonthDataI("AgeIdx",25,1)  =  $zwc(148,119)/*$bit()*/
^MyWork.MonthDataI("AgeIdx",26,1)  =  $zwc(128,119)_$c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,･･･
```

ビットマップインデックスの圧縮

- 繰り返し行われるデータの入れ直しによりビットマップ・インデックス用ストレージに不要な情報が残ったり、一括更新作業で効率が低下するようなストレージになった場合、**%SYS.Maint.Bitmapクラスの OneClass()、Namespace()メソッドを使用してビットマップ・インデックスを圧縮(維持管理)できます。**
- 例） 前頁のテーブルデータを1000件作成した後、Tuncateを実行した後の実行例

クラス名

ジャーナルへの記録有:1
記録無:0

```
USER>set st=##class(%SYS.Maint.Bitmap).OneClass("MyWork.MonthData",1,1)

Class:  MyWork.MonthData Start Time:  2018-01-19 18:16:43
    Global:  ^MyWork.MonthDataI("$MonthData")was compressed:  100.00%
        Old Size: 0.000(MB) New Size: 0.000(MB)
    Global:  ^MyWork.MonthDataI("AgeIdx")was compressed:  100.00%
        Old Size: 0.004(MB) New Size: 0.000(MB)
Compression time in seconds:  0
USER>
```

結果の表示有:1
表示無:0