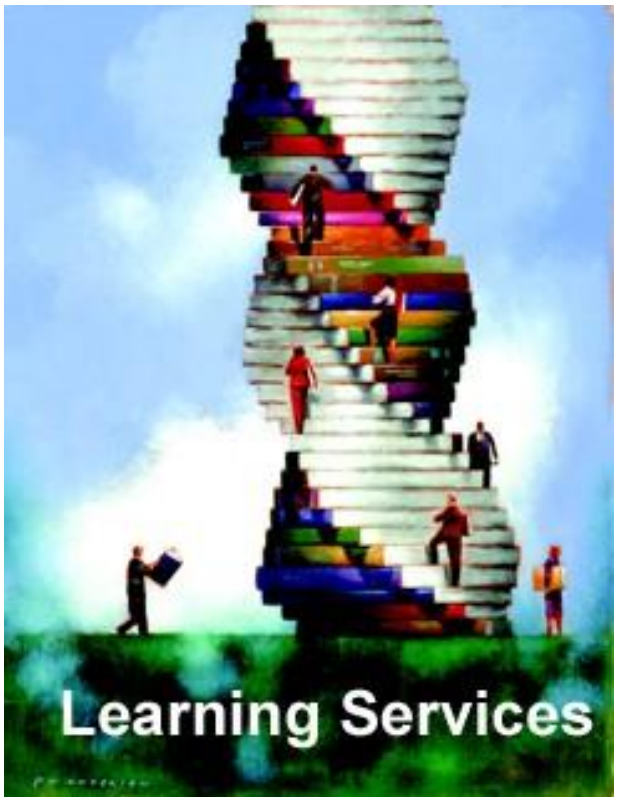


IRIS SQL機能ガイド

（IRIS Version2024.1ベース）

V1.0

2024年12月

インターシステムズジャパン株式会社

内容

1. はじめに

本ガイドは、IRISのSQL機能に関する入門書です。

以下の説明では、作業場所としてIRISをインストールすると標準で作成されるUSERネームスペースを使用することを前提としています。

もちろん適当なネームスペースを作成して作業を行うこともできます。

本ガイドを習得すれば、IRIS SQL機能をお使いいただく上において必要十分な知識が得られます。

さらに詳しい情報については、オンライン・ドキュメント等をご参照ください。

なお、本ガイドでは管理ポータル内SQL操作用画面を「SQLポータル」として記述しています。

2. SQLポータル

IRISのSQL基本機能を確認するには、SQLポータルを使用します。

2.1. SQLポータルの使用方法

SQLポータルへは、管理ポータルから移動します。

SQLポータルに移動後、参照したいデータが存在するネームスペースに切り替えます。

管理ポータル→システムエクスプローラ→SQL

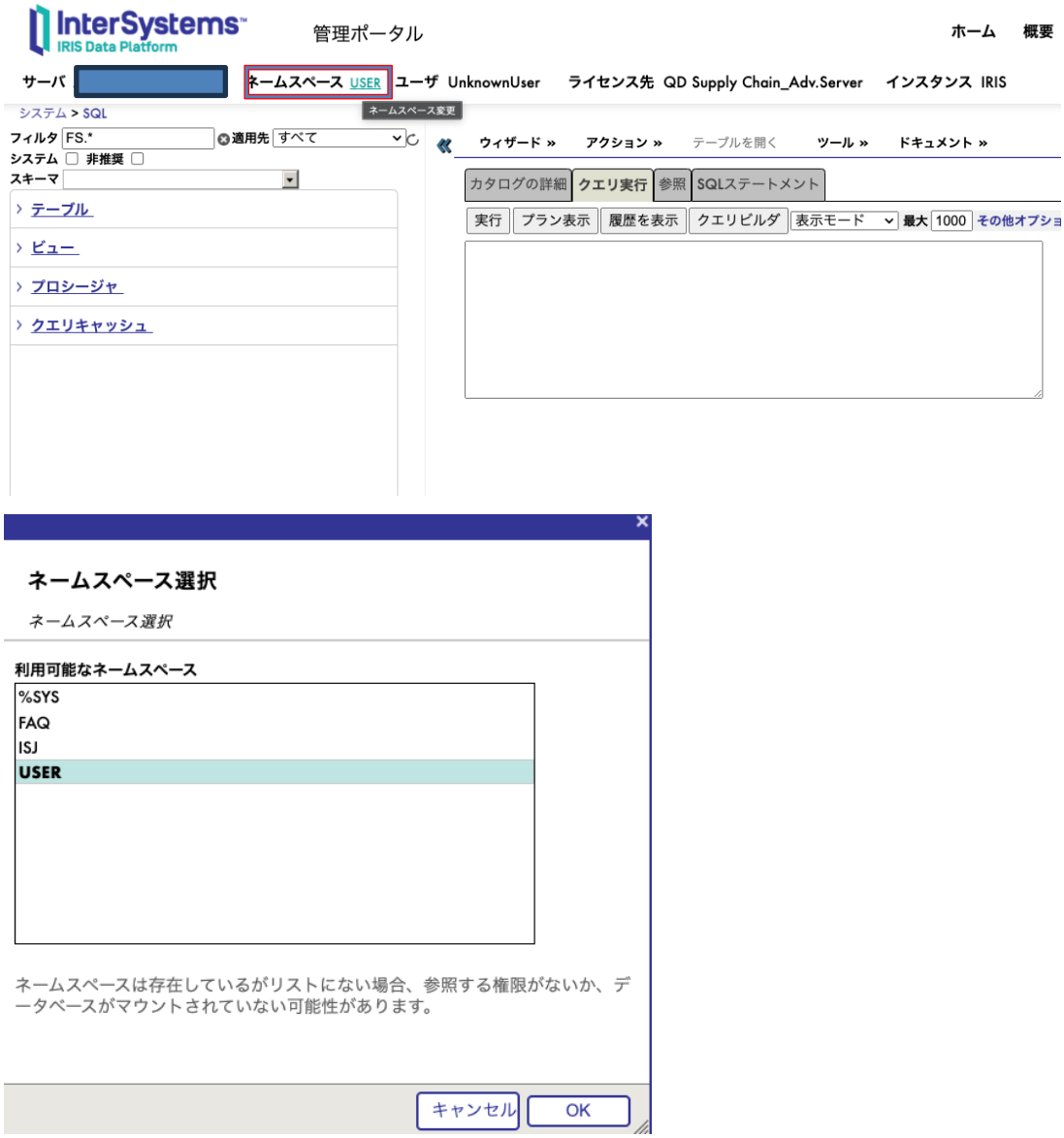


図1 SQL ポータルの使い方

「スキーマ」のドロップダウンにスキーマー覧が表示されます。

2.2. サンプルデータの生成

SQLポータルの基本機能を確認するためにサンプルデータを生成します。

＜このサンプルは、step1フォルダにあります。＞

まず step1フォルダにあるクラス定義step1.xml をインポートします。

なお、本ガイドでは、USERネームスペースの利用を前提に説明しています。

インポートは、管理ポータル>エクスプローラ>クラスを選択し、インポートボタンを押してその該当ファイルの場所を指示します。

表示されたインポートダイアログボックスのOKボタンを押してください。

インポートおよびそれらのクラスのコンパイルが実行されます。

コンパイル終了後、テストデータの生成を行います。

テストデータ生成に関して前準備が必要です。

住所を生成するために郵便番号データが必要です。

サンプルデータがstep1フォルダにあります。

サンプルデータ用ファイルstep1.gsaを、管理ポータルからインポートします。

管理ポータル→グローバル→左ペインでUSERネームスペースを選択→インポートより、
step1.gsaを選択しインポートします。

データの生成は、IRISターミナルを使用し、インポートしたクラス定義に対して、Populate()メソッドを実行すること自動生成されます。

IRISキューブからターミナルを選択して下さい。

以下の様なウィンドウが表示されます。

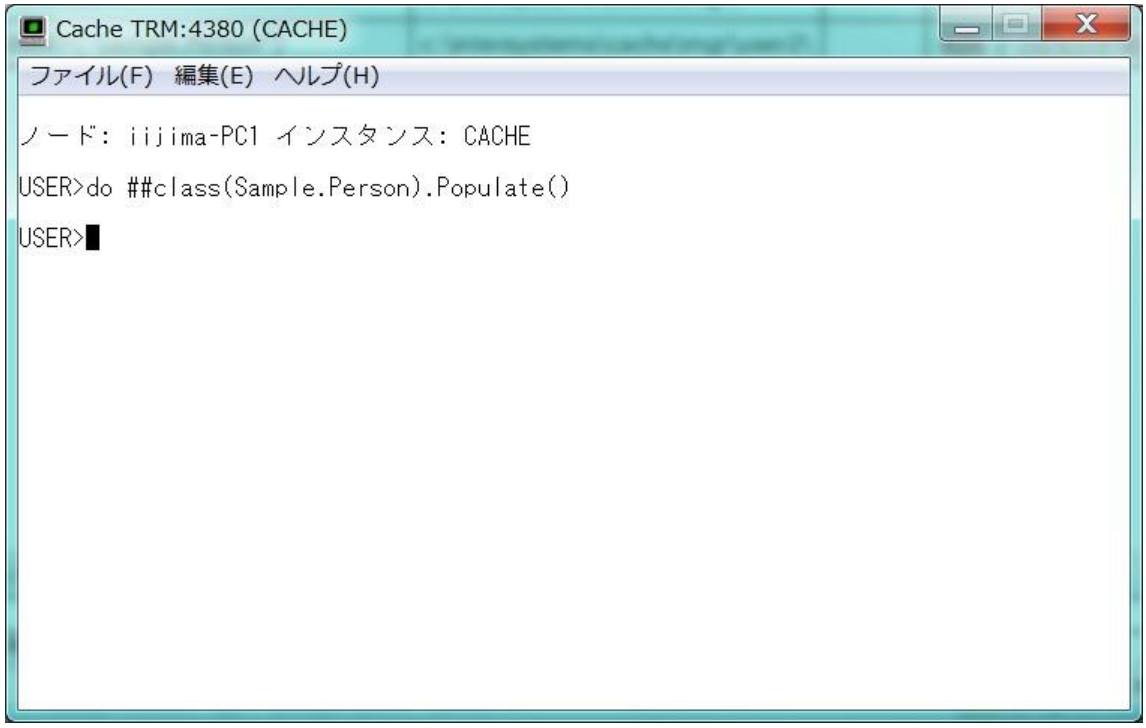


図5 ターミナルでのデータ自動生成 Populate()実行

Linux,Macではターミナルソフトから以下のコマンドを実行してください。

iris session iris

Populate()メソッドは、%Library.Populate クラスが継承されたクラスで実行できる自動生成用メソッドです。

図5 ターミナルでのデータ自動生成 Populate()実行にあるように、

Do ##class(パッケージ名.クラス名).Populate()

を実行するだけで、デフォルト10件のデータが作成できます。

また、件数を指定したい場合は、Populate()メソッドの引数に件数を指定できます。

例） Do ##class(Sample.Person).Populate(5)

それでは、作成されたデータをSQLポータルから参照してみましょう。

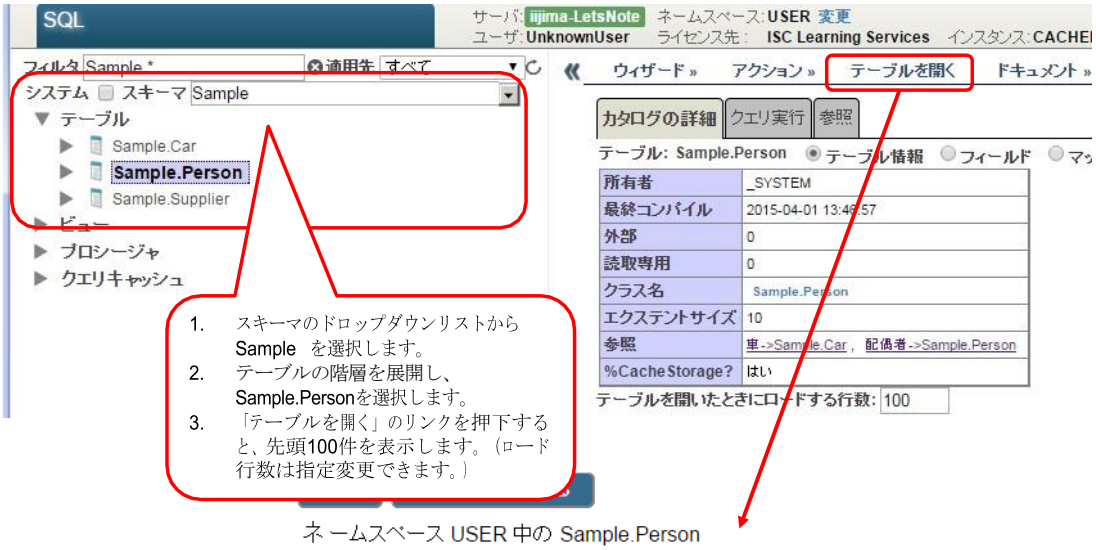
管理ポータルから SQLポータルを選択してください。

ネームスペース：USERを選択します。

インポートクラスは、Sampleパッケージで作成されています。

SQLポータルでは、パッケージ名は、スキーマ名として参照できるので、Sampleスキーマが表示されます。

Sample スキーマを選択すると、テーブル一覧が表示されます。



ネームスペース USER 中の Sample.Person

#	ID	年令	車	誕生日	好きな色	名前	配偶者	自宅_都市	自宅_都道府県	自宅_地名	自宅_郵
1	1	29		07/23/1985	オレンジ	日高 敏明		知多郡武豊町	愛知県	川尻	470235
2	2	80		06/06/1934	紫	林 徹		豊橋市	愛知県	西浜町	441015
3	3	14		08/02/2000	青	川島 博美		大分郡庄内町	大分県	渕	879542
4	4	11		11/07/2003	黒	乗口 博史		白河市	福島県	四ツ谷	961096
5	5	1		09/16/2013		大崎 紀子		天竜市	静岡県	青谷	431342
6	6	6		02/05/2009	青	柏木 裕香		熊谷市	埼玉県	高柳	360085
7	7	42		01/09/1973		大谷 敏哉		大垣市	岐阜県	横曽根	503094
8	8	86		07/20/1928	青	関口 恵美		印旛郡富里町	千葉県	高野	286021
9	9	46		05/11/1968	黒	山原 俊子		北葛城郡王寺町	奈良県	明神	636002
10	10	27		02/29/1988	紫	松尾 克道		諫早市	長崎県	中田町	859031

完了

図6 Sampleスキーマのテーブル一覧

左画面でテーブル名を選択した状態で、右画面の「テーブルを開く」のリンクを押下し、レコードを参照できます。（デフォルトで先頭から 100 件のデータが参照できます。）

続いて、SQL文の実行メニューから、SQL文でSample.Personのデータを参照します。

SQL文の記入が終わったら、クエリ実行ボタンを押下すると、SQL文の実行が開始されます。

今回用意した、テーブルには、フィールド「好きな色」があり、このフィールドは、好きな色を1～n 個登録できるように、Listコレクションとして定義しています。

IRIS内部では、Listコレクションの定義がある場合、特別な関数（$LISTBUILD()関数）を使用して、n個の要素をリストとして登録しています。

SQLポータルでは、SQL文実行時「表示モード」を切り替えることもできます。

9	46		05/11/1968	黒	山原俊子		北葛城郡王寺町	奈良県	明神	6360022	松江市	島根県	意宇町	6900027
10	27		02/29/1988	紫	松尾克道		諫早市	長崎県	中田町	8590313	湯沢市	秋田県	山谷	0120821
11	25		04/24/1989	緑	宮本麻美	9	茅ケ崎市	神奈川県	美住町	2530023	鯖江市	福井県	深江町	9160052
12	57		05/25/1957	緑 オレンジ	山原孝	8	能代市	秋田県	後谷地	0160863	徳島市	徳島県	沖浜町	7708051
13	87		03/30/1928	緑 緑	根本保之	1	双葉郡大熊町	福島県	以下に掲載がない場合	9791300	飯田市	長野県	浜井町	3950022
14	88		01/24/1927	黒	岩淵いずみ	6	新冠郡新冠町	北海道	本町	0592401	明石市	兵庫県	中崎	6730883
15	46		02/17/1969	赤	清水馨	3	橋本市	和歌山県	橋谷	6480095	芦屋市	兵庫県	以下に掲載がない場合	6590000

図7 Sample.Person のデータ表示

試しに、「論理モード」に切り替え、実行ボタンを押下すると、文字化けしたような表示になります。

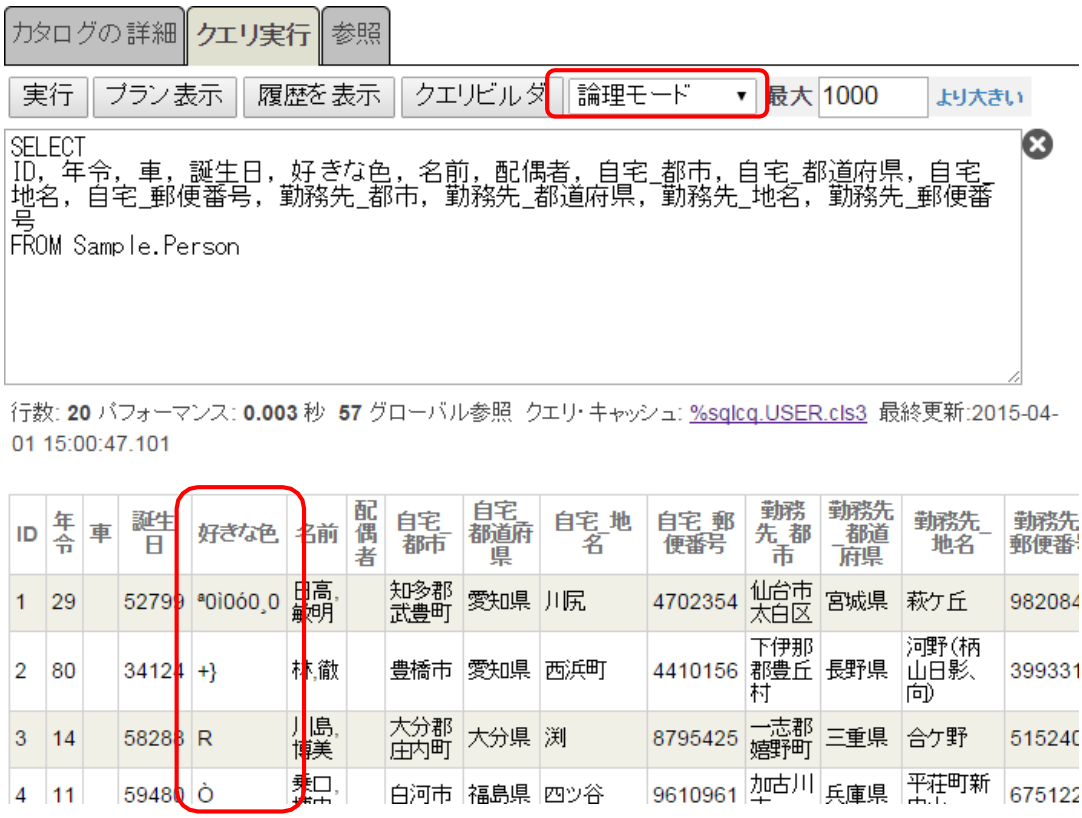


図8 Sample.Person；論理モードでの表示

論理モードでも、$LISTTOSTRING()関数でフィールド「好きな色」を括れば、表示モードと同じ表示結果が得られます。

図9 Sample.Person；論理モードでの表示 $LISTTOSTRING()の利用

$LISTTOSTRING()では、Listの要素が複数ある場合は、カンマで区切ります。

いくつかのSQL文を入力し、同様のことを行い、結果がどうなるか確認してみてください。

また、すでにお気づきかもしれませんが、Address1クラスに該当するテーブルは存在していません。

その理由はAddress1クラスが%Persistentクラスを継承していないためです。

3. 統一データアーキテクチャ

IRISでは、クラス定義からオブジェクトアクセスおよびSQLアクセスに必要な実行コードを自動生成し、それらは同一の多次元データ構造にアクセスします。

この仕組みにより、見かけ上はまったく異なるアクセス手法でありながら、同一のデータに対して操作が可能になっています。

また定義を変更する必要が発生してもクラス定義を変更しさえすれば、その変更は自動的にオブジェクトアクセス、SQLアクセスに反映できるようになっています。

この仕組みを**統一データアーキテクチャ**と呼んでいます。

とはいえ、オブジェクト指向とリレーショナルデータベースの間には考え方の違いがあり、オブジェクト指向アクセスのほうが、より豊富なデータ表現ができます。

IRISでは、それらをSQLに投影するため、SQLアクセスに関して通常のRDBMSにはない、いくつかの拡張を行っています。

3.1. クラス定義をテーブルにプロジェクションする

以下の表でオブジェクトクラスの各要素が SQLのテーブル要素にどの様にマッピングされるかを示します。

オブジェクトクラス表現	リレーショナル表現
パッケージ	SQL スキーマ
クラス	テーブル
インスタンス	行
オブジェクト識別子	主キー
リテラルプロパティ	カラム
永続オブジェクトへの参照	外部キー
埋め込みオブジェクト	個別カラム
リレーションシップ	外部キー／依存リレーションシップ
List コレクション	リストフィールドを持つカラム
Array コレクション	子テーブル
ストリーム	BLOB
インデックス	インデックス
クエリ	ストアドプロシジャ、ビュー
クラスメソッド	ストアドプロシジャ

表1 オブジェクトとリレーショナル：要素のマッピング

リレーショナルモデルは、クラスパラメータ、多次元プロパティ、インスタンスメソッドという概念を持たないので、これらは、リレーショナル表現には投影されません。

一方IRISは、リレーショナルモデルにしか存在しないトリガーもサポートします。

3.2. 継承

通常のリレーショナルデータベースには、継承という概念はありません。

しかし、IRISのクラス定義で表現した継承の概念をSQLアクセスに投影することができます。

以下の様なクラス階層を想定します。

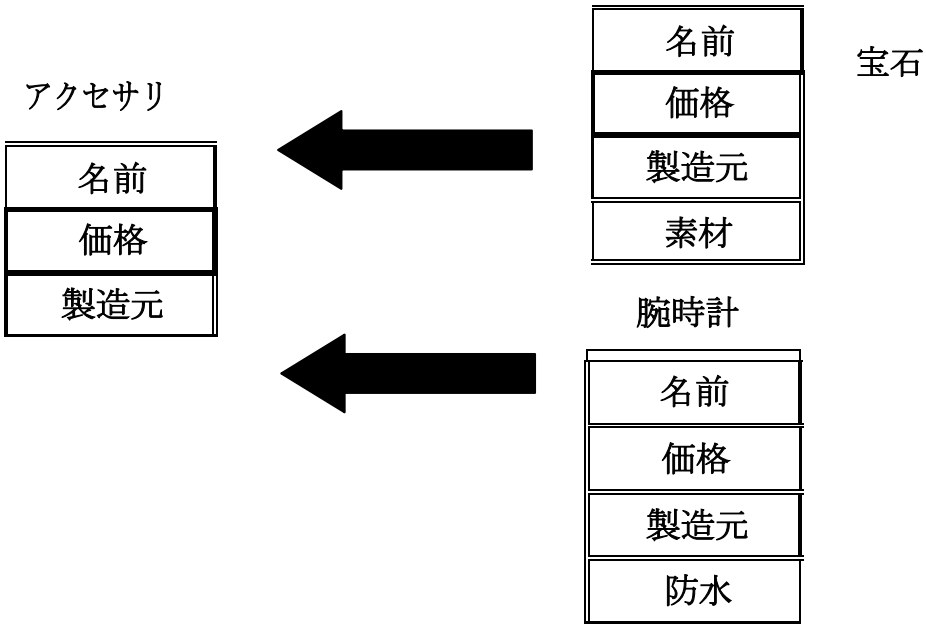


図10 継承のクラス例

[図10 継承のクラス例]のクラス階層は、アクセサリというスーパクラスがあり、そのサブクラスとして宝石クラスと腕時計クラスがあるということを示しています。

そして、宝石クラスには素材というプロパティが追加されているのに対して、腕時計クラスには、防水プロパティが追加されています。

では早速、サンプルで確認してみましょう。

＜このサンプルは、step2フォルダにあります。＞

step2ディレクトリにあるstep2.xmlを管理ポータルでインポートします。

データの作成にはターミナルを起動し、Sample.Accesary クラスに用意したデータ自動生成用メソッド CreateAll()を実行します。

```
do ##class(Sample.Accesary).CreateAll()
```

デフォルトで、合計20件のアクセサリー（Sample.Accesary）を作成します。
（ランダムに宝石クラスと腕時計クラスのデータを作成しています。）

データ作成が完了したら、SQLポータル→SQL 文の実行 から以下 SQL文を入力し、実行します。

```
SELECT * FROM Sample.Accesary

SELECT * FROM Sample.Jewel

SELECT * FROM Sample.Watch
```

アクセサリ（Sample.Accesary）を問い合わせ対象とした場合には、宝石（Sample.Jewel）と腕時計（Sample.Watch）を合わせたすべての行が表示されます。

宝石を問い合わせ対象とした時には、宝石クラスにのみある素材フィールドの値が表示され、腕時計を問い合わせ対象とした時には、腕時計クラスにのみある防水フィールド（Waterproof）の値が表示されることがおわかりいただけるかと思います。

3.3. データタイプ

IRIS Object のデータタイプとそのメソッドは、そのままリレーショナルのプロジェクションに反映されます。

つまり、データタイプが持つ変換処理や検証処理がそのまま使用でき、クラスのパラメータが使用可能です。

例えば、PersonクラスのNameプロパティは、JNAMEというデータタイプとして定義されています。

そこで、JNAMEデータタイプの定義を参照すると、%NAMEを継承し、以下のメソッドが定義されております。

```
ClassMethod LogicalToDisplay(%val As %Name = "") As %String
{
    Quit $Piece(%val,",",1)_" "_$Piece(%val,",",2)
}
```

これは、内部表現（姓,名前）を表示表現（姓　名前）に変換します。

これと逆に表示表現を内部表現に変換するDisplayToLogical()メソッドも定義されていることがわかるでしょう。

SQL文でNameフィールド（プロパティ）を取得する様に設定し、SQLポータルからクエリを実行するとNameの値が表示表現、つまり（姓　名前）の形式で表示されることがわかると思います。

3.4. 計算プロパティ

計算プロパティをSQLにプロジェクションするためには、クラス定義中の該当プロパティに SQLComputedを設定し、SqlComputeCodeにプロパティ値を求めるコードを記述します。

PersonクラスのAgeプロパティの定義をVisual Studio Codeから参照してみてください。

現日付から誕生日を引いて日数計算し、それから年齢を取得しているのがわかるでしょう。

```
SqlComputeCode =
    { Set {年令}=$select({誕生日}="":"",1:$zdate($horolog,8)-$zdate({誕生日},8)¥10000)}
```

3.5. 永続オブジェクトへの参照

IRISでは、クラスのプロパティのタイプに他永続クラスのタイプを指定することによりオブジェクト参照を表現できます。

IRIS SQLでは、他のテーブルへの外部キーとしてプロジェクトされます。

では、またサンプルで確認しましょう。

<このサンプルは、step3フォルダにあります。>

step3のstep3.xmlを管理ポータルからインポートおよびコンパイルして下さい。

次に管理ポータルのグローバルのインポートメニューにてstep3.gsaからデータ（グローバル）をインポートしてください。

次に SQLポータルの"SQL文の実行"から以下のSQL文を発行して下さい。
SELECT * FROM SAMPLE.注文

プロパティCustomer1にCustomerクラスのidが入っているのが確認できると思います。

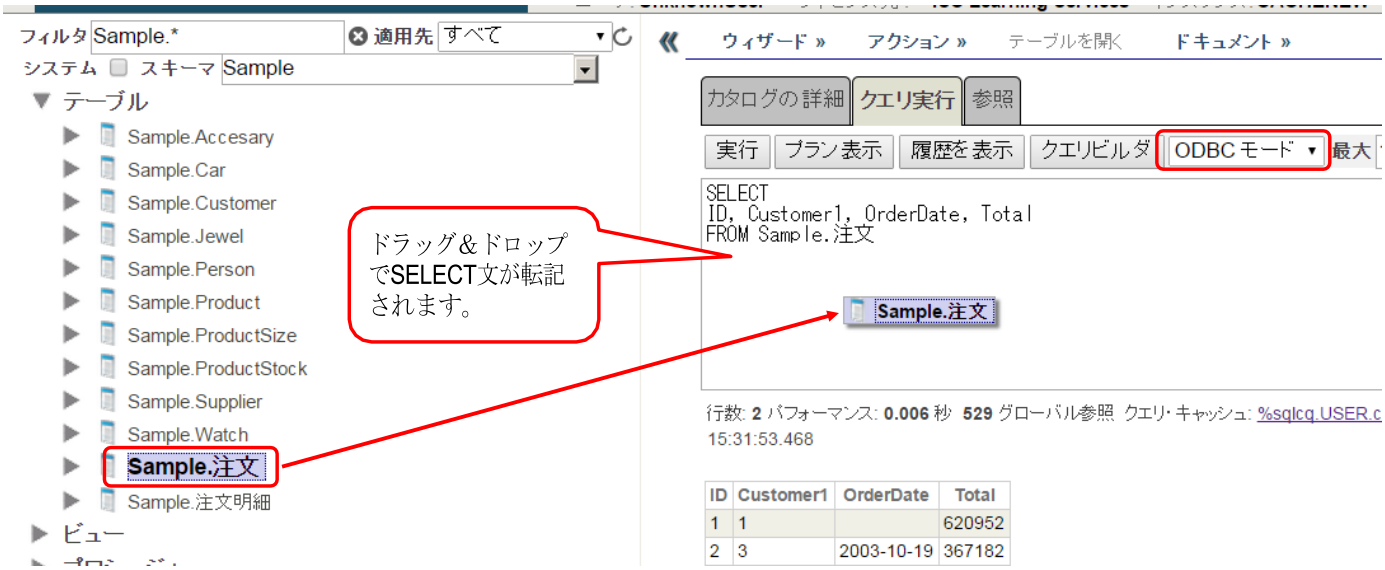


図11　永続オブジェクトへの参照を持つプロパティの表示（Sample.注文テーブル）

3.6. 埋め込みオブジェクト

埋め込みオブジェクトのプロパティは、IRIS SQLでは個々のフィールドとしてプロジェクトされます。

そしてその名前は、埋め込みオブジェクトクラス名とそれに関連するプロパティ名をアンダースコアで連結した名前になります。

それでは、サンプルで確認しましょう。

SQLポータルの"SQL文の実行"から以下のSQL文を発行してみてください。

```
SELECT * FROM SAMPLE.CUSTOMER
```

CustomerクラスのAddress1プロパティは、埋め込みオブジェクトAddress2であり、その個々のフィールドCity、Pref、Streetが SQLのフィールドとしてどの様に表現されているか確認して下さい。

図12 埋め込みオブジェクトを持つプロパティの表示（Sample.Customer）

メモ：図では、テーブル名をクエリ記入欄にドラッグ＆ドロップした状態で検索しています。

3.7. リレーションシップ

永続オブジェクト間のリレーションシップは、IRIS SQLでは他のテーブルへの外部キーとしてマップされます。

その外部キーは、単一のオブジェクト参照を持つ側にのみ、表示されます。
（つまり、それぞれ 1 対多の多側、親子の子側）

それでは、サンプルで確認しましょう。

SQLポータルの“SQL文の実行”から、以下のSQL文を発行してみてください。

Order（注文）クラスと OrderItem（注文明細）クラスには親子リレーションシップがあります。

そして、子の OrderItem（注文明細）クラスからTheOrderプロパティを参照することにより、親 id が取得できます。

なお、Order（注文）クラスおよび OrderItem（注文明細）クラスは、クラスのキーワード SQLTableName を使用して、クラス名と別名のテーブル名を指定しています。

そのため、SQL 文を実行する場合には、SQLTableName に指定した名称を指定する必要がありますので、ご注意ください。

（クラス名とテーブル名を異なる名前にした理由は、Orderが、SQLの予約語に指定されているためです。）

```
SELECT TheOrder FROM SAMPLE.注文明細
```

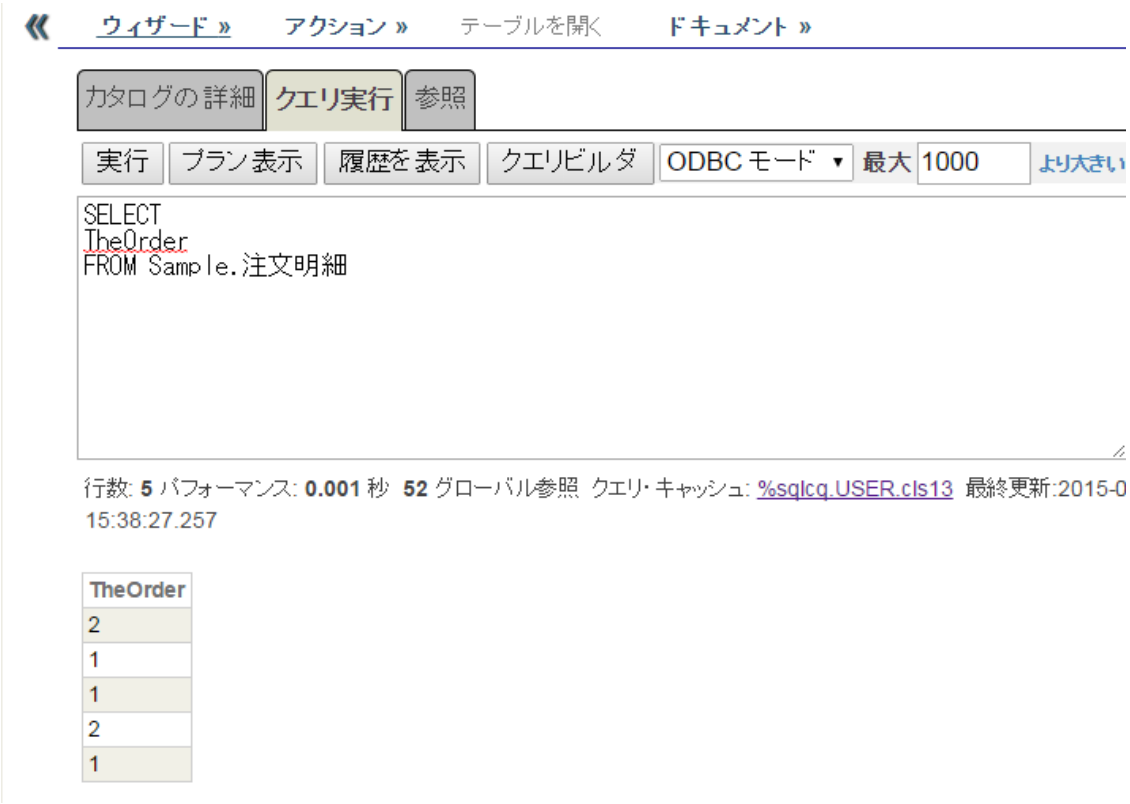


図13　リレーションシップ定義による外部キーの確認（Sample.注文明細テーブル）

3.8. Listコレクション

Listコレクションは、コレクションの値を持つ1つのリストを含む単一フィールドとしてプロジェクトされます。

PersonクラスのFavoriteColorがテーブルのフィールド'好きな色'にプロジェクトされています。SQLポータルから以下の SQL文を発行して確認してみて下さい。

（フィールド "好きな色" は Listコレクションのため、画面中の表示を「論理モード」にしている場合は、フィールドを$LISTTOSTRING()関数で括ることで正しく表示されるようになります。）

```
SELECT * FROM SAMPLE.PERSON
```

3.9. Arrayコレクション

Arrayコレクションは、外部キーを使い、主テーブルにつながる子テーブルとして表現されます。

子テーブルの名前は、**主テーブル名_そのコレクション名**より構成されます。

では、またサンプルで確認しましょう。

＜このサンプルは、step44フォルダにあります。＞

step4の step4.xmlを管理ポータルからインポートおよびコンパイルして下さい。

Userパッケージ配下に2つのクラスと、ルーチンArrayPopulate.macがインポートされます。

次にＩＲＩＳターミナルで該当ネームスペースにログインし、以下のコマンドを実行してください。

```
Do ^ArrayPopulate()
```

SQLポータルからSQLUser.tblNewClass1とSQLUser.tblNewClass2および
tblNewClass1_tblNewClass2を開いて内容を確認して下さい。

（SQLポータル→スキーマ：SQLUserを選択した後、それぞれのテーブルの“テーブルを開く”のリンクからデータを確認します。）

4. SQLアクセス法

IRISからSQLを使用してデータベースにアクセスする方法が、いくつか用意されています。
そのアクセスは、問い合わせ、レポーティングだけでなく性能とセキュリティの要求されるオンライントランザクション処理にも利用可能です。

4.1. ANSI標準SQL

IRISは、SQL ,SQL92 ANSI 標準の以下の全要素をサポートしています。

- データ問い合わせ言語（DQL）
- データ操作言語（DML）
- データ定義言語(DDL)
- トランザクション管理言語(TCL)
- データ制御言語(DCL)

4.2. 埋め込みSQLアクセス

データベースアプリケーション開発を容易にするため、IRISは、メソッドとルーチン内にSQL文の埋め込みができます。
そして、それをObjectScriptの中で以下の用途で使用することができます。

- 複雑なデータベース問い合わせを行うため
- 結果をObjectScript言語に結びつけるため

埋め込みSQL文は、&sql()プリプロセッサ機能を使用します。

＜このサンプルは、step5フォルダにあります。＞

それでは、まずstep5のstep5.xml を管理ポータルからインポートし、コンパイルして下さい。
Sample.Utilsクラスがインポートされます。

このクラスには、Sample.Personに対して、クラスクエリの実行や、ダイナミックSQL の実行、埋め込みSQLへのアクセスなどを確認する、クラスメソッドが定義されています。

次に、ターミナルから以下クラスメソッドを実行してください。

```
do ##class(Sample.Utils).PersonSelect()
```

スタジオで Sample.UtilsクラスのPersonSelect()メソッドを開き、コードを確認します。

単純なSQL文を実行し、返ってきた値をObjectScriptの変数にバインドしているのが確認できます。

PersonSelect()では、1 行しか答えが返ってこないため、データの取得は、一回のみでOKですが、複数行返ってくる場合は、どうしたら良いでしょう。

そのためには、カーソルを使用します。

ターミナルから以下クラスメソッドを実行してください。

```
do ##class(Sample.Utils).PersonSelectAll()
```

今度は複数行返ってくるのが確認できると思います。

スタジオで Sample.UtilsクラスのPersonSelectAll()メソッドを開き、処理を確認して下さい。カーソルを開き、Fetchを繰り返し行っていることが確認できると思います。

4.3. ResultSetオブジェクト

前述の埋め込みSQL文に対して、よりオブジェクト指向的にSQLを実行する手段がIRISには用意されています。

そのためには、ResultSetオブジェクトを使用します。

(このオブジェクトクラスは現在非推奨機能となっています。)

4.3.1. クラスクエリの実行

ResultSetオブジェクトは、クラスに定義されたクエリを実行することができます。

ターミナルから以下のコマンドを入力し、実行して下さい。

```
do ##class(Sample.Utils).PersonResultSet("山")
```

'山'で始まる名前の人のIDと名前が表示されるのが確認できるはずです。

Visual Studio CodeでSample.Utils クラスのPersonResultSet()メソッドを開きコードを確認します。

ResultSetオブジェクトを%New()メソッドでインスタンス化する際、クエリ名を指定しているのが確認できます。

4.4. ダイナミックSQL

プログラム実行時に、ダイナミック（動的）にSQL文を指定できる「ダイナミックSQL」の実行方法もあります。

ダイナミックSQLを利用するには、%SQL.Statementクラスを使用して、SQL文のコンパイル、実行を行います。

（このクラスが先ほどのResultSetクラスの後継になります）

ターミナルから以下のコマンドを実行して下さい。

```
do ##class(Sample.Utils).PersonDynamicSQL()
```

Sample.Person の全員の名前が表示されるのが確認できるはずです。

Visual Studio CodeでSample.Personクラスの PersonDynamicSQL()メソッドを開き、コードを確認します。

ダイナミックSQLを実行するため、%SQL.Statementクラスを%New()でインスタンス化し、%Prepare()メソッドの引数に、実行対象のSQL文を指定し、コンパイルを実行しています。

その後、%Execute()メソッドの実行で、結果セットオブジェクト（%SQL.StatementResult）が返るため、結果セットオブジェクトを前方に移動し、SELECTに指定したカラム値を取得しています。

5. SQL拡張

IRIS SQLでは、標準SQLにはないいくつかの独自拡張機能があります。

以下にそれぞれにつき簡単に説明します。

5.1. IRIS演算子

以下表は、標準SQLにはないIRIS特有の演算子です。

シンボル	説明
=*	一方向外部結合
_,#	文字列結合、モジュロ除算
?,[	比較演算子、パターンマッチ、包含
&,|	CacheObjectScript AND, OR
]	Follow演算子
%STARTSWITH	開始演算子

図14 IRIS特有の演算子

さらにIRIS SQLでは、リテラルの識別にシングルクォートだけでなくダブルクォートの使用が可能です。

5.2. 追加IRIS関数

組み込み機能としてIDカラムの実際の名前に関わらず、IRIS SQLは、常にIDカラムを表す%ID仮カラムを提供します。

さらに追加の関数をプログラマが定義することが可能です。

任意のクラスメソッドにSqlprocキーワードを使い、ストアドプロシジャとして宣言することにより、それをSQL関数として使用できます。

それでは、SQLポータルを起動し、以下のSQL文を入力して実行してみて下さい。

```
select sample.Stored_Procedure_Test(名前) as jname from sample.person
```

先ほど作成したPerson10人分の行が返り、列 jnameに姓<sp sp>名前が返ってきているのが確認できます。

次にVisual Studio Codeから Sample.Personクラスを開き、StoredProcTest()クラスメソッドの内容を確認して下さい。

入力された文字列から ',' を除去しているのが確認できると思います。

5.3. リスト

IRIS SQLでは、複数の値を持ったフィールドの定義が可能です。

この概念は、ObjectScriptでは Listとして知られています。

SQLでは、リストは、シリアライズされた1つの文字列としてプロジェクトされます。

この文字列は2つの形式で格納できます。

- List形式

  ObjectScriptの$List 関数で指定される形式です。

  自動的に管理されます。

- 区切り文字列

  開発者が定義した区切り文字により各要素が区切られたユーザ管理リスト形式です。

  リストコレクションは、リレーショナルプロジェクションでは、リストフィールドとして表現されます。

それでは、ターミナルから以下のコマンドを実行して下さい。

```
do ##class(Sample.Utils).PersonSQLList()
```

好きな色を複数持つ人がいるのを確認できると思います。

Visual Studio CodeからSample.Utilsクラスの PersonSQLList()メソッドを開いて、コードを確認してください。

$List関数を使い、好きな色が複数ある場合にそれを1つ1つの色に分解しているのが確認できると思います。

5.4. 結合

IRIS SQLは、標準SQLにはない2つの結合タイプ拡張があります。

5.4.1. 一方向外部結合

一方向外部結合では、2つ目のテーブルに対応する行がなくても、1つ目のテーブルの全行を含みます。

それでは、SQLポータルから以下のSQL文を入力し、実行してみて下さい。

```
Select * from sample.supplier, sample.product
where sample.supplier.id =* sample.product.supplier
```

上記の結果と以下のSQL文の実行結果の違いを確認して見てください。

```
select * from sample.supplier, sample.product
where sample.supplier.id = sample.product.supplier
```

一方向外部結合の場合には、1つ目のテーブル行は、対応する行が 2つ目のテーブルにあるなしにかかわらず表示されていることが確認できると思います。

5.4.2. 暗黙結合

暗黙結合は、SQLクエリには明示的に指定せずに2つのテーブルの結合を指示します。

その結合は、内部的にデータベース内のキーに暗黙的に関係付けられています。

IRIS SQLは、暗黙結合の 2つの異なった形式、参照と従属リレーションシップをサポートします。

5.4.2.1. 参照

参照は、参照されるテーブルのデータレコードの主キー（ID）を参照側のテーブルが持っている時に使用できます。

IRIS SQLは、暗黙結合用に特別なシンタックスを提供します。

そのシンタックスを使用して参照をたどっていくことができます。

今回インポートしていただいた以下3つのテーブルを利用して暗黙結合用のシンタックスを練習します。（各テーブルの詳細はスタジオを利用してご確認ください）

Sample.Personテーブル、Sample.Carテーブル、SampleSupplierテーブル

Sample.Personテーブルのフィールド：配偶者（Spouse）は同じSample.Personテーブルを参照しています。

Sample.Personテーブルのフィールド：車（Car）はSample.Carテーブルを参照しています。

次に、Sample.Carテーブルのフィールド：Makerは Sample.Supplierテーブルを参照しています。

Sample.Personテーブル、Sample.Supplierテーブルは既にデータ自動生成やインポートを行っていますが、手順により、まだそれぞれの参照を持ったデータが現在存在していません。

以下の手順で上記 3テーブルに対してデータ自動生成を行います。

(1) Sample.Personテーブルの全データを削除します。

ターミナルを起動し対象ネームスペースにて以下実行してください。

```
do ##class(Sample.Person).%KillExtent()
```

(2) Sample.Carテーブルのデータを自動生成します。

```
do ##class(Sample.Car).Populate()
```

(3) 次に Sample.Personテーブルのデータを自動生成します。

```
do ##class(Sample.Person).Populate()
```

この時点で3テーブルにデータが存在しますが、Sample.Personテーブルのフィールド：配偶者はSample.Personテーブル自身を参照しているため、現時点では参照情報がありません。

ここで、もう一度、Sample.Personテーブルに対してデータ自動生成を行います。

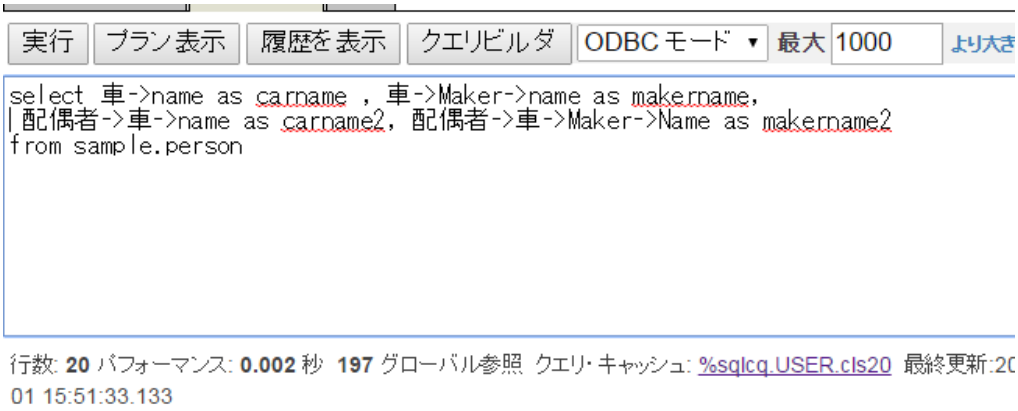
（必ず２回行ってください。）

```
do ##class(Sample.Person).Populate()
```

上記手順でデータを自動生成すると3テーブルが参照情報を持つようになります。

それでは、SQLポータルから以下の暗黙SQL文を入力し、実行してみて下さい。

select 車->name as carname , 車->Maker->name as makername, 配偶者->車->name as carname2, 配偶者->車->Maker->Name as makername2
from sample.person



carname	makername	carname2	makername2
フィラシューズ	メトロゴールド		
フラミンゴサルン	ボールスミス		
フォワード	ボールスミス		
フィラシューズ	メトロゴールド		
フラミンゴサルン	ボールスミス		
フォワード	ボールスミス		
フィールドライン	フィラシューズ		
フィラシューズ	メトロゴールド		
フィラシューズ	メトロゴールド		
フィラシューズ	メトロゴールド		
メトロゴールド	ボーダメイド	フラミンゴサルン	ボールスミス
フィラシューズ	メトロゴールド	フィラシューズ	メトロゴールド
フォワード	ボールスミス	フィラシューズ	メトロゴールド
フォワード	ボールスミス	フィラシューズ	メトロゴールド
ミリオンエア	ボーダメイド	フラミンゴサルン	ボールスミス

図15 暗黙結合 実行例

最初の10件に対してcarname2、makename2が表示されない理由は、フィールド：配偶者は同じSample.Personテーブルを参照しているので、初回の Sample.Personテーブルの自動生成では参照情報が設定されません。

Sample.Personテーブルのデータを作成した後の2回目の自動生成で参照情報が割り当てられことになるので上記表示になります。

5.4.2.2. 従属リレーションシップ

従属リレーションシップは、テーブル（子テーブル）の行の存在が他のテーブル（基本テーブル）に依存しているテーブル間の結合です。

つまり、基本テーブルの行は、子テーブルの行に対して、1対他のリレーションシップを持ちます。

子テーブルは、常にその基本テーブルを参照しますので、関係は、暗黙結合と考えることができます。

5.4.2.2.1. 子テーブルから基本テーブルへの参照

上記にも記述した様に、暗黙結合を使い、参照することができます。

それでは、SQLポータルから以下のSQL文を入力して、実行してみて下さい。

```
select TheOrder, TheOrder->OrderDate, TheOrder->Total, ID, Quantity
from sample.注文明細  where Quantity < 5
```

上記 SQL文を通常の結合条件で書き直すと、

```
select sample.注文.id, sample.注文.orderdate, sample.注文.Total, sample.注文明細.id, sample.注文明細.Quantity from sample.注文,sample.注文明細
where sample.注文明細.Quantity < 5 and sample.注文.id = sample.注文明細.theorder
```

暗黙結合を使用すると、SQL文をシンプルに表現できることが理解できると思います。

6. 外部SQL連携

IRIS　SQLサーバ機能を使い、外部のツールとＩＲＩＳを連携させることができます。

6.1. ODBC接続

Windows環境では、ODBCを使い、様々なツール（問い合わせツール、帳票ツール、データ分析ツールなど）と連携が可能です。

6.1.1. DSN設定

IRISサーバにODBC接続するためには、DSNを設定する必要があります。

IRISをWindows環境にインストールすると、IRISのODBCドライバは自動的にインストールされます。

この ODBCドライバを使用し、DSN の設定を行います。

まず、コントロールパネル→管理ツール→データソース（ODBC）を起動します。
32bitと64bitが別々に用意されています。

ODBCクライアントツールに合わせて選択してください

以下の様な画面が表示されます。

ここで、システムDSNタブを選択すると、既にIRIS用のDSNが1つ（Userネームスペース用）が定義済みであるのが確認できるはずです。

これをそのまま利用することも可能ですが、ここでは新しいDSNを作成してみましょう。

図16　インストールで用意されるDSN名

上記の画面で追加ボタンを押します。

そうすると以下の様にODBCドライバを選択する画面がでてきます。

ここで InterSystems IRIS ODBC35を選択して、完了ボタンを押して下さい。

図17 ODBC ドライバーの選択

次に以下の画面が表示されます。

画面中、認証方法にパスワードとKerberosの2つがありますが、このガイドではパスワード認証を採用します。

ここで、設定したいDSN内容に基づき、設定項目を入力します。

名前	: DSNを識別するための名前
説明	: 説明情報
ホスト（IP アドレス）	: IRISサーバの IP アドレスポート
	: 通常 1972
Cache ネームスペース	: 接続先の IRISネームスペース名認証方
法	: パスワード
ユーザ名	: _system
パスワード	: SYS

図18 IRISへ接続のためのDSN設定

入力が終わったら、「テスト接続」ボタンを押下し、正しく接続できるか確認してみます。

図19 ODBC接続テスト

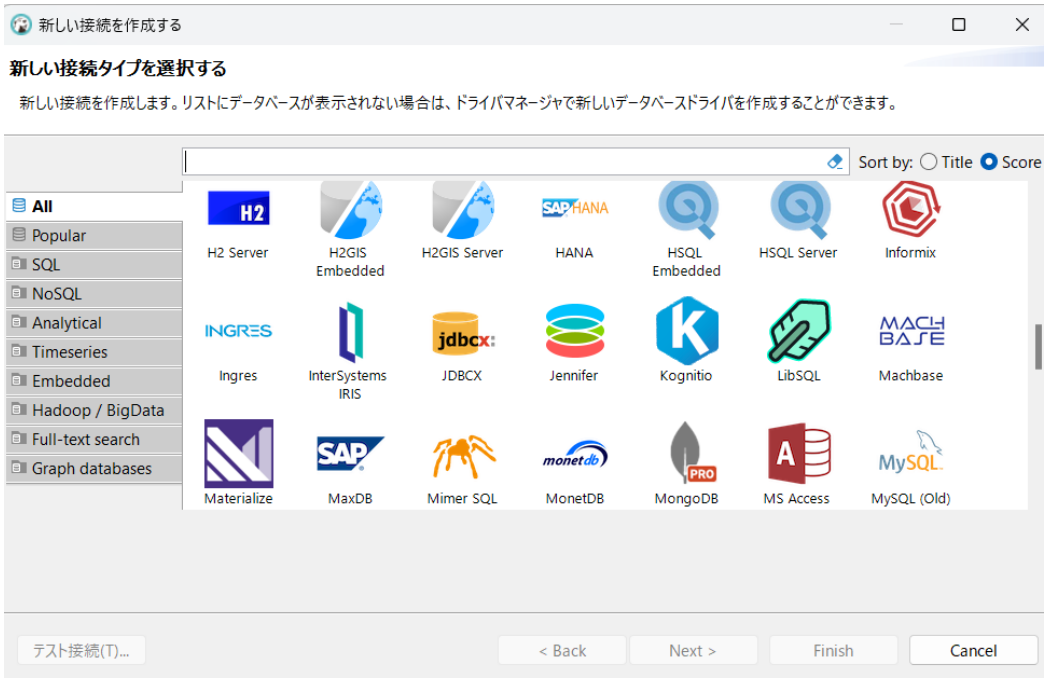
接続テストが完了したら、OKボタンを押下し、設定を保存します。

6.1.2. ODBC/JDBC対応ツールからのアクセス

前述（6.1.1）のDSN設定後、任意のODBC対応ツールからIRISサーバにODBC接続が可能になります。

ここではフリーツールであるDBeaverを使った例を示します。(これはJDBCを使用)
(https://dbeaver.io/download/)

データベース>新しい接続をクリック



InterSystems IRISをクリック

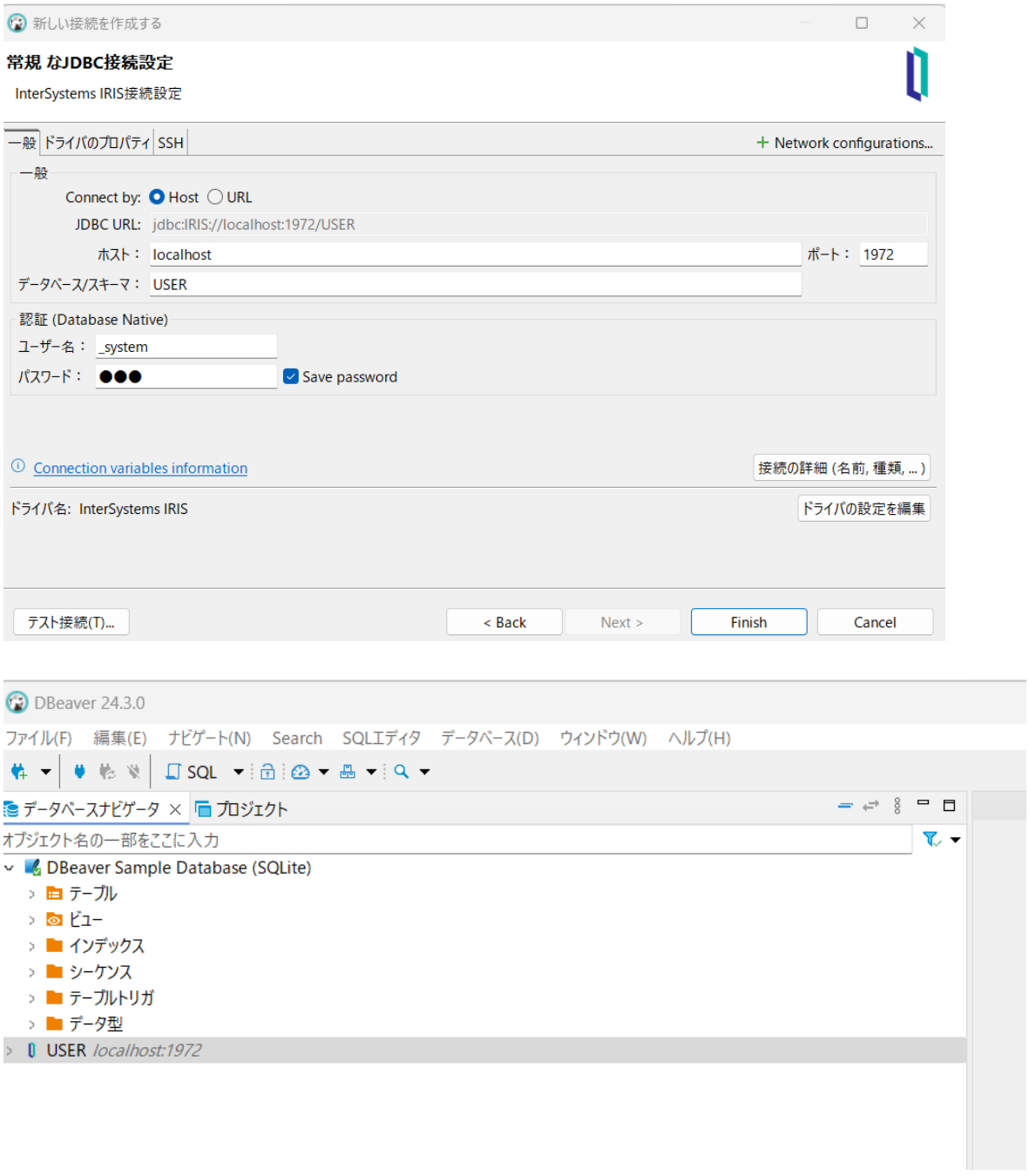
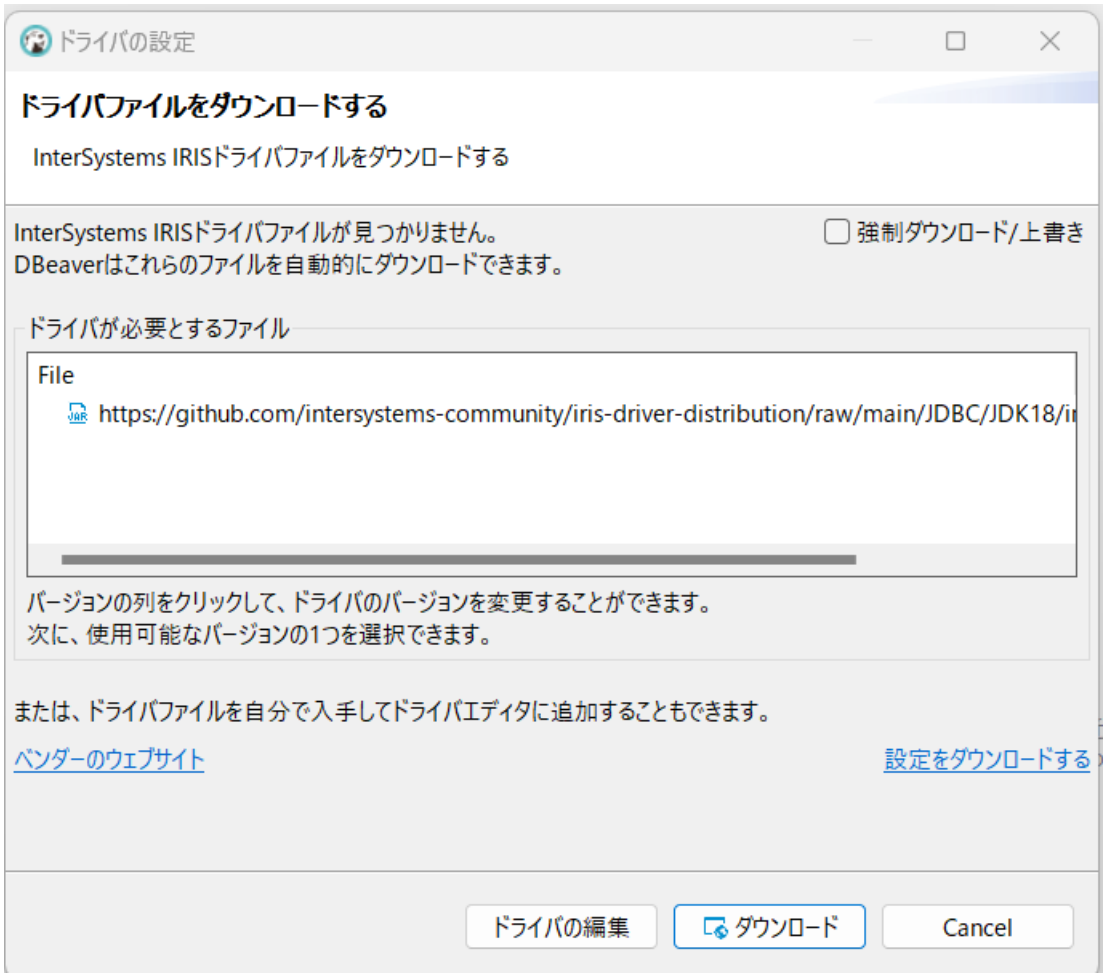
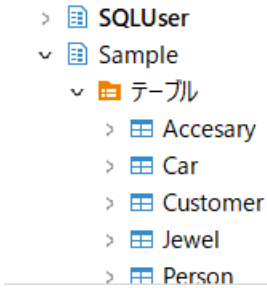


図20　外部ソフトウェアからのODBC/JDBC接続



DBMSドライバーのダウンロードスクリーンが表示されるのでダウンロードボタンをクリック

以下のように表示されます



Personを選択して、右クリックし、View Dataをクリックします

Person 結果をフィルタリングするSQL式を入力します（Ctrl + Spaceを使用）

	ID	年令	車	誕生日	好きな色	名前	配偶者	自宅_都市	自宅_都道府県	自宅_地名	自宅_郵便番号
1	1	52	7	1972-04-02	オレンジ,オレンジ	金沢,孝子	[NULL]	幡豆郡一色町	愛知県	松木島	4440403
2	2	35	5	1989-01-06	紫,紫	橋本,誠一	[NULL]	十日町市	新潟県	中条乙	9498616
3	3	42	7	1982-10-30	緑,紫	高岡,真紀	[NULL]	八幡市	京都府	橋本尻江	6148312
4	4	52	3	1972-10-16	黒,黄色	三好,伊代	[NULL]	碧南市	愛知県	松原町	4470079
5	5	16	8	2008-05-07	オレンジ	波内,康之	[NULL]	竹原市	広島県	下野町	7250012
6	6	94	5	1930-07-25	[NULL]	山崎,信弘	[NULL]	天草郡倉岳町	熊本県	宮田	8616403
7	7	40	3	1984-09-02	[NULL]	大野,芳郎	[NULL]	須崎市	高知県	横町	7850014
8	8	45	3	1979-01-28	黒	荒川,博康	[NULL]	金沢市	石川県	香林坊	9200961
9	9	10	2	2014-08-17	白,紫	伊藤,恵	[NULL]	茨木市	大阪府	大同町	5670844
10	10	30	2	1994-01-27	[NULL]	柳,孝子	[NULL]	金沢市	石川県	桜田町	9200057
11	11	41	9	1983-02-03	[NULL]	小島,陽子	2	土佐市	高知県	蓮池	7811105
12	12	24	6	2000-04-03	黄色	細田,雄三	8	勝浦郡上勝町	徳島県	以下に掲載がない場合	7714500
13	13	98	1	1926-02-19	[NULL]	金子,洋子	8	飯石郡頓原町	島根県	頓原村	6903202
14	14	89	9	1935-12-10	オレンジ	伊藤,エミリ	1	雨竜郡北竜町	北海道	碧水	0782503
15	15	99	1	1925-01-20	[NULL]	河野,幸博	8	秋田市	秋田県	金足下刈	0100114
16	16	32	3	1992-02-19	オレンジ,白	奥山,正一	1	富山市	富山県	田刈屋新町	9300894
17	17	15	7	2008-12-25	黄色	砂川,裕香	3	大原郡加茂町	島根県	加茂中	6991106
18	18	68	9	1956-06-13	白,青	吉村,高志	1	七尾市	石川県	魚町	9260864
19	19	73	4	1951-10-21	黄色,白	山本,誠一	2	大津市	滋賀県	木下町	5200812
20	20	61	3	1963-11-12	黄色	藤木,秀美	10	上川郡東川町	北海道	東９号北	0711439

図21 ODBC/JDBC 経由でのSQL文実行

6.2. IRIS Managed Provider for .NETの利用

IRIS Managed Providerでは、ADO.NETの APIを利用して、IRISにアクセスすることができます。

step6には、VB.NETで作成したDataReader、DataSetを使った、簡単な検索処理を確認するサンプルがあります。

7. IRIS SQLチューニング

データ量が増えてくると、当然ながらSQL処理にも時間がかかる様になります。

処理スピードを最適化するためにいくつかのことを考慮しなければなりません。

7.1. インデックス

最適化においてまず検討すべきことは、クエリに対して適切なインデックスが設定されているかということです。

IRISでは、インデックスのタイプとして、標準インデックスとビットマップインデックスの2つのタイプをサポートしています。

ビットマップインデックスは、ビジネスインテリジェンス等の複雑な条件でクエリを実行する際に威力を発揮するインデックスです。

一般にビットマップインデックスは、検索は高速ですが、作成、更新には時間がかかるといわれます。

しかしIRISのビットマップインデックスは、作成、更新に関しても、通常のインデックスと同等の速度で処理できます。

7.1.1. インデックスの設定法

Visual Studio Codeを起動し、Sample.Personクラスを開いて下さい。
このクラスは、既に以下の様な 2つのインデックスが設定されています。

```
/// Define an index for <property>Name</property>.
Index NameIDX On Name [ Data = Name ];

/// Define an index for embedded object property <b>ZipCode</b>.
Index ZipCode On Home.Zip [ Type = bitmap ];
```

7.2. インデックスの構築、再構築

インデックス定義終了後、SQLアクセスおよびオブジェクトアクセスにてデータの挿入、更新および削除に関して、システムが自動的にインデックスの維持管理を行います。

しかし、インデックスの定義前に既に存在していたデータに対してインデックスを作成およびインデックスをきれいに再構築するために、再構築する手段も用意されています。

インデックスを定義設定するには、SQLポータル→（操作したいネームスペースへ移動）→ Sampleスキーマを選択→Personテーブルを選択します。

テーブル全体で再構築を行うか、インデックス定義単体の構築か選択できます。

以下の図は、インデックス単体での再構築です。

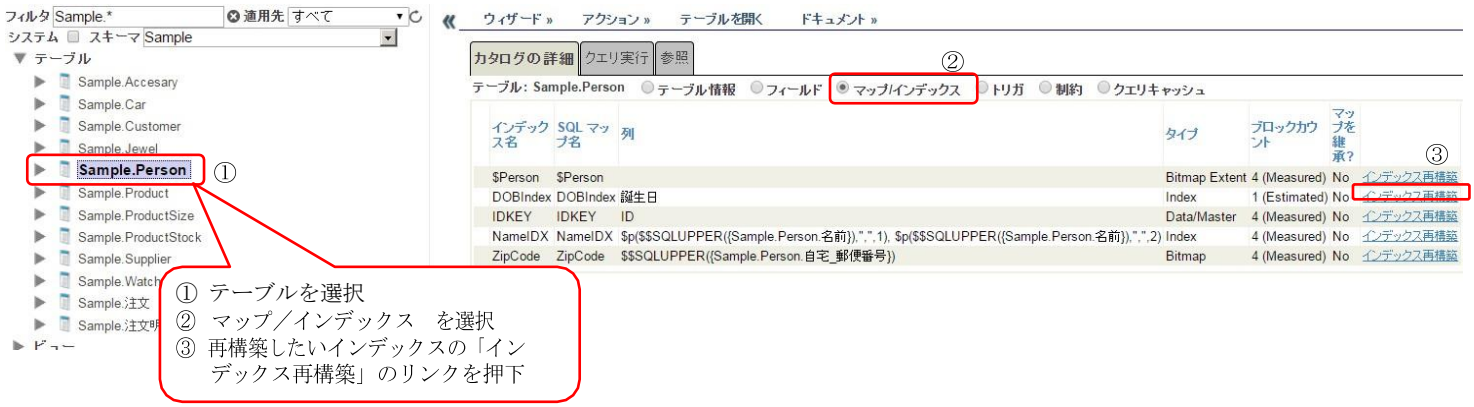
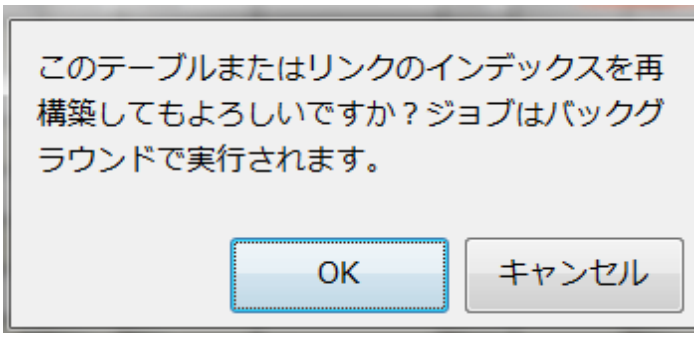


図 25 インデックスの再構築（インデックス定義単体の構築：管理ポータル）

テーブル全体での再構築は、アクション→インデックス再構築のメニューから行います。

図26　インデックス再構築（テーブル全体の再構築：管理ポータル

インデックスを再構築すると以下のようなダイアログボックスがでてきます。OKボタンを押下し、再構築を開始します。



SQLポータルでの再構築実行以外にも、各永続クラスのクラスメソッド%BuildIndices()を実行する方法も用意されています。

```
Do ##class(Sample.Person).%BuildIndices()
```

7.3. TuneTable

テーブル結合が発生するクエリを発行した場合には、クエリの内部的な実行順番で処理時間に大きな差がでます。

最適な実行順を決定するためには、現状のテーブル（クラス）のエクステント（行数、インスタンス数）を分析することが不可欠です。

この分析を行う処理をテーブルチューニング（TuneTable）と言い、SQLポータルまたは、IRISのコマンド行として実行可能です。

まず、SQLポータルから実行する方法を以下に示します。

「テーブルチューニング」を押下します。

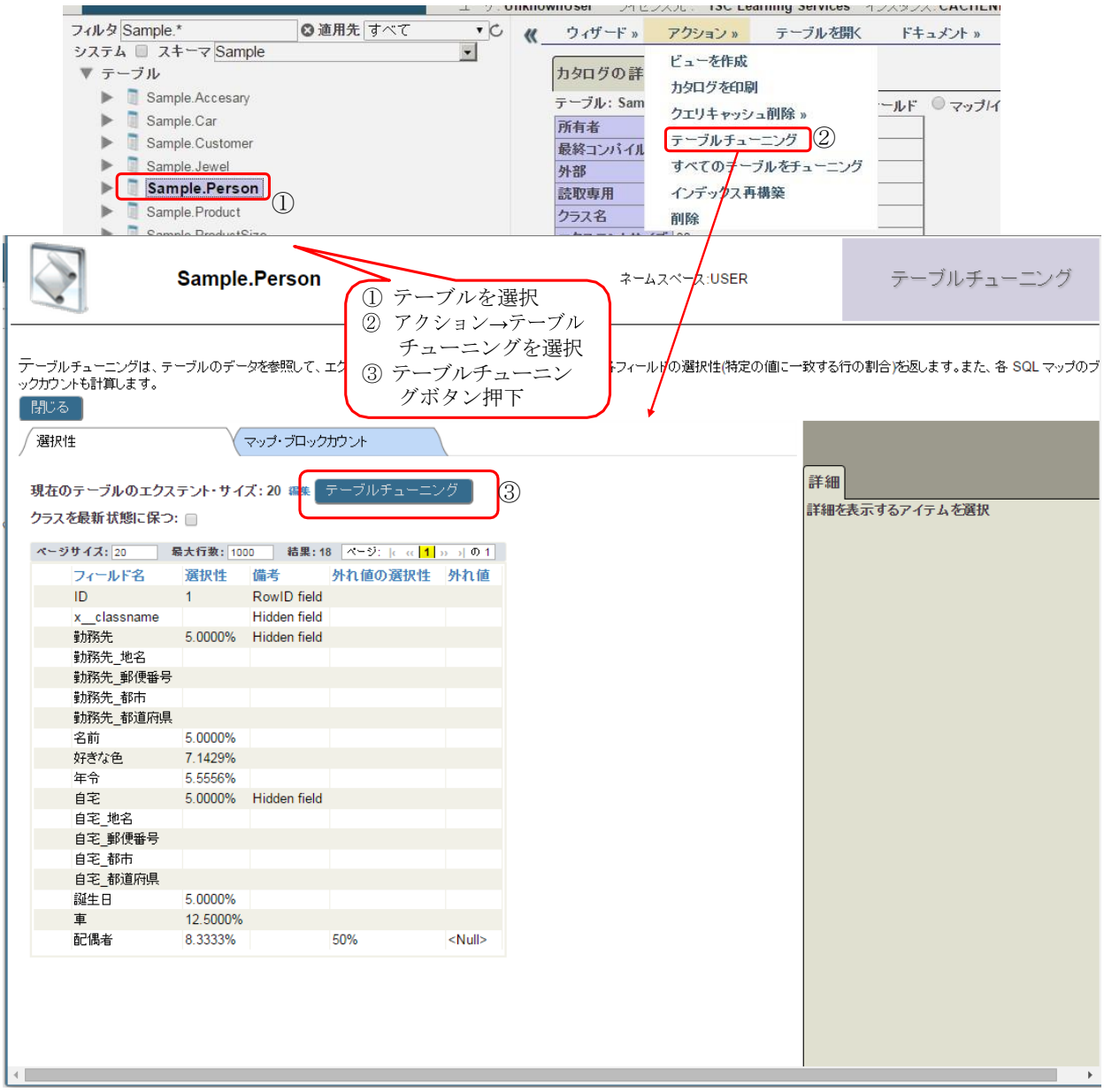


図27 テーブルチューニング（SQL ポータル）

テーブルチューニングは、テーブルサイズが大きいと時間がかかる可能性があります。

以下、%SYSTEM.SQLクラスから提供されているTuneTable()メソッドからも、実行でもできます。TuneTable()メソッドの第1引数は、テーブル名（スキーマ名.テーブル名）、第2引数は、計算の反映可否を（反映する：1、反映しない:0）指定します。

```
Do $System.SQL.TuneTable("Sample.Person",1)
```

チューニングを行うと、計算結果となる選択性の数値が表示されます。

列	データタイプ	列#	必須	ユニーク	照合	隠し	最大長	最大値	最小値	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
ID	%Library.Integer	1	Yes	Yes		No				No		1	INTEGER		No	
年令	%Library.Integer	2	No	No		No				No		3.7037%	INTEGER		No	
車	%Library.Integer	3	No	No		No				No		10.0000%	INTEGER	Sample.Car	No	
誕生日	%Library.Date	4	No	No		No				No		3.3333%	DATE		No	
好きな色	%Library.String	5	No	No	SQLUPPER	No	50			No		5.5556%	VARCHAR		No	
自宅	Sample.Address1	6	No	No		Yes				No		3.3333%	VARCHAR		No	
名前	Sample.JNAME	7	Yes	No	SQLUPPER	No	25			No		3.3333%	VARCHAR		No	
勤務先	Sample.Address1	8	No	No		Yes				No		3.3333%	VARCHAR		No	
配偶者	%Library.Integer	9	No	No		No				No		8.3333%	INTEGER	Sample.Person	No	
x__classname	%Library.CacheString	10	No	No		Yes				No		100.0000%	VARCHAR		No	
自宅_都市	%Library.String	11	No	No	SQLUPPER	No	80			No	自宅	3.3333%	VARCHAR		No	
自宅_都道府県	%Library.String	12	No	No	SQLUPPER	No	10			No	自宅	4.7619%	VARCHAR		No	
自宅_地名	%Library.String	13	No	No	SQLUPPER	No	80			No	自宅	3.3333%	VARCHAR		No	
自宅_郵便番号	%Library.String	14	No	No	SQLUPPER	No	7			No	自宅	3.3333%	VARCHAR		No	
勤務先_都市	%Library.String	15	No	No	SQLUPPER	No	80			No	勤務先	3.3333%	VARCHAR		No	
勤務先_都道府県	%Library.String	16	No	No	SQLUPPER	No	10			No	勤務先	5.0000%	VARCHAR		No	
勤務先_地名	%Library.String	17	No	No	SQLUPPER	No	80			No	勤務先	3.3333%	VARCHAR		No	
勤務先_郵便番号	%Library.String	18	No	No	SQLUPPER	No	7			No	勤務先	3.3333%	VARCHAR		No	

図28　テーブルチューニングの結果（選択性数値の表示）

7.4. クエリプラン

同じ結果を得るクエリであっても、実行効率が異なることが起こり得ます。

効率の良いクエリを投入することで処理速度が向上したり、リソースの消費を抑えたりできます。クエリが内部的にどの様に処理されるかを確認するためにクエリプランの表示機能があります。

クエリプランの表示も、SQLポータルから行います。

SQLポータル→Sampleスキーマをクリックします。

表示される画面で、"クエリキャッシュ"をクリックします。

図29 SQL ポータル：クエリキャッシュ一覧画面

クエリキャッシュ一覧から参照したいキャッシュ名を選択すると、右画面に詳細が表示されます。キャッシュのクエリプランを参照するためには、"プランの表示" を押下します。

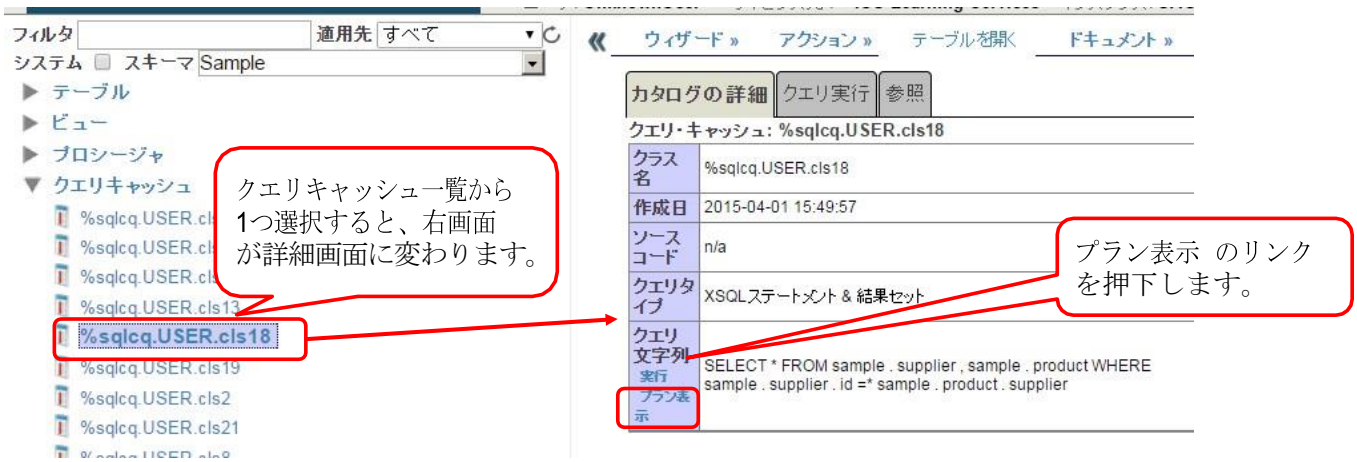


図30 クエリキャッシュ 詳細画面

プラン表示のリンク押下で表示されるプランは以下の通りです。

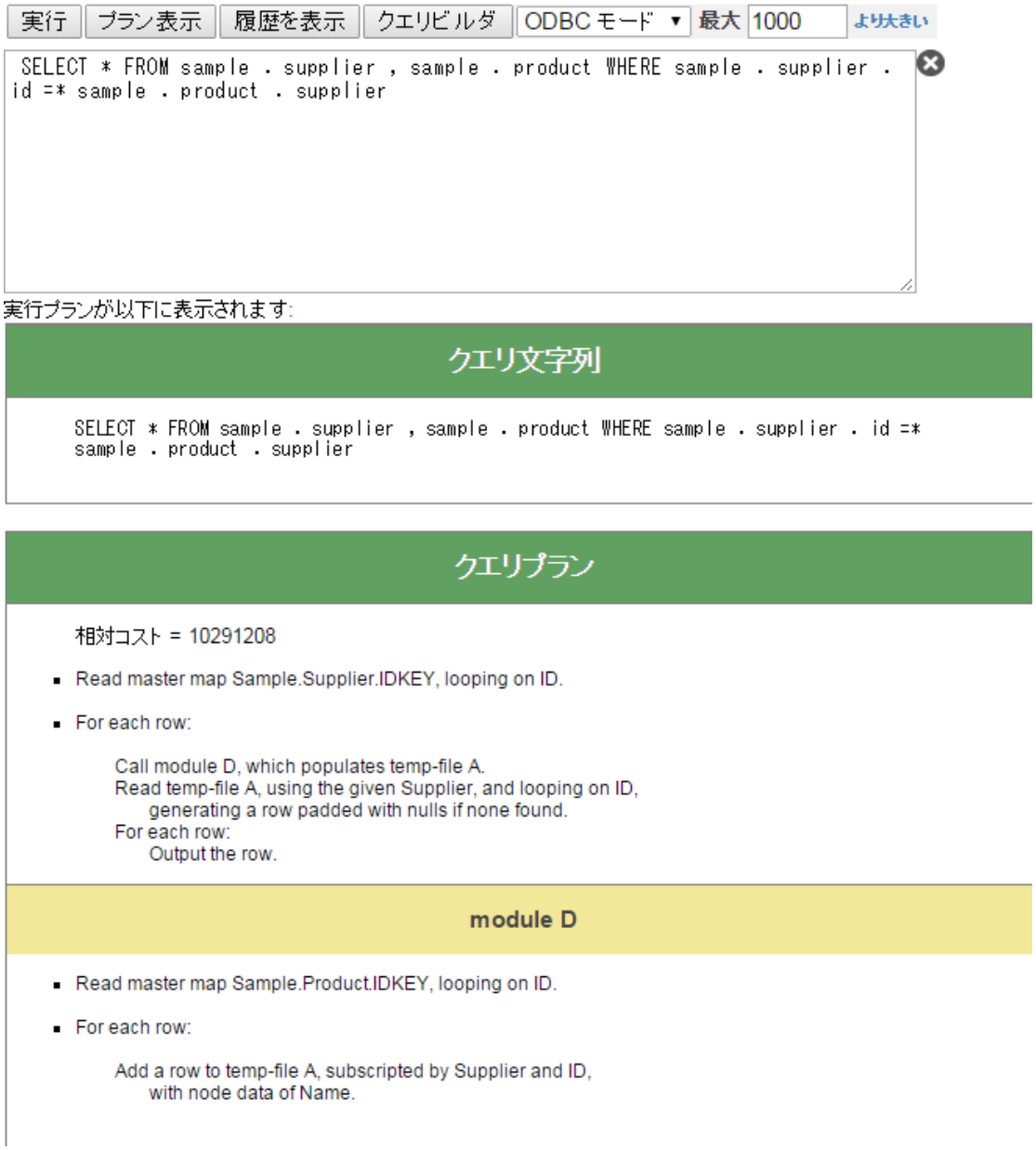


図31 クエリプラン表示

8. 他RDBMS からの移行

他RDBMSのテーブル定義とデータをIRISに移行する方法について説明します。

8.1. DDLの移行方法

他 DBMS の DDL 文が記述されている入力ファイルから、IRISにそのテーブル定義を取り込むことができます。

```
Do $SYSTEM.SQL.DDLImport("Sybase","_SYSTEM","C:¥PT¥Patient.sql")
Do $SYSTEM.SQL.DDLImport("Oracle","DAVE","C:¥DDT¥all_tables.sql",all.log,0,"",";",2)
```

図32 $SYSTEM.SQL.DDLImport() 実行例

第一パラメータには、DBMS 名を入力します。現在、以下の DBMS をサポートしています。

- Informix
- MSSQL
- MSSQLServer （MSSQLと同じ）
- Sybase

DDLImport()メソッド詳細は、クラスリファレンス→%SYSTEM パッケージ→SQLクラスよりご参照ください。

8.2. データの移行

SQLポータルに、データ移行用の「データインポートウィザード」が用意されています。

図33　SQLポータルのウィザード（データインポートウィザードなど）

データインポートウィザードでは、以下 3つの項目を指定します。

インポートファイルのパスおよび名前の指定

インポートするスキーマ名の決定

インポートするテーブル名選択

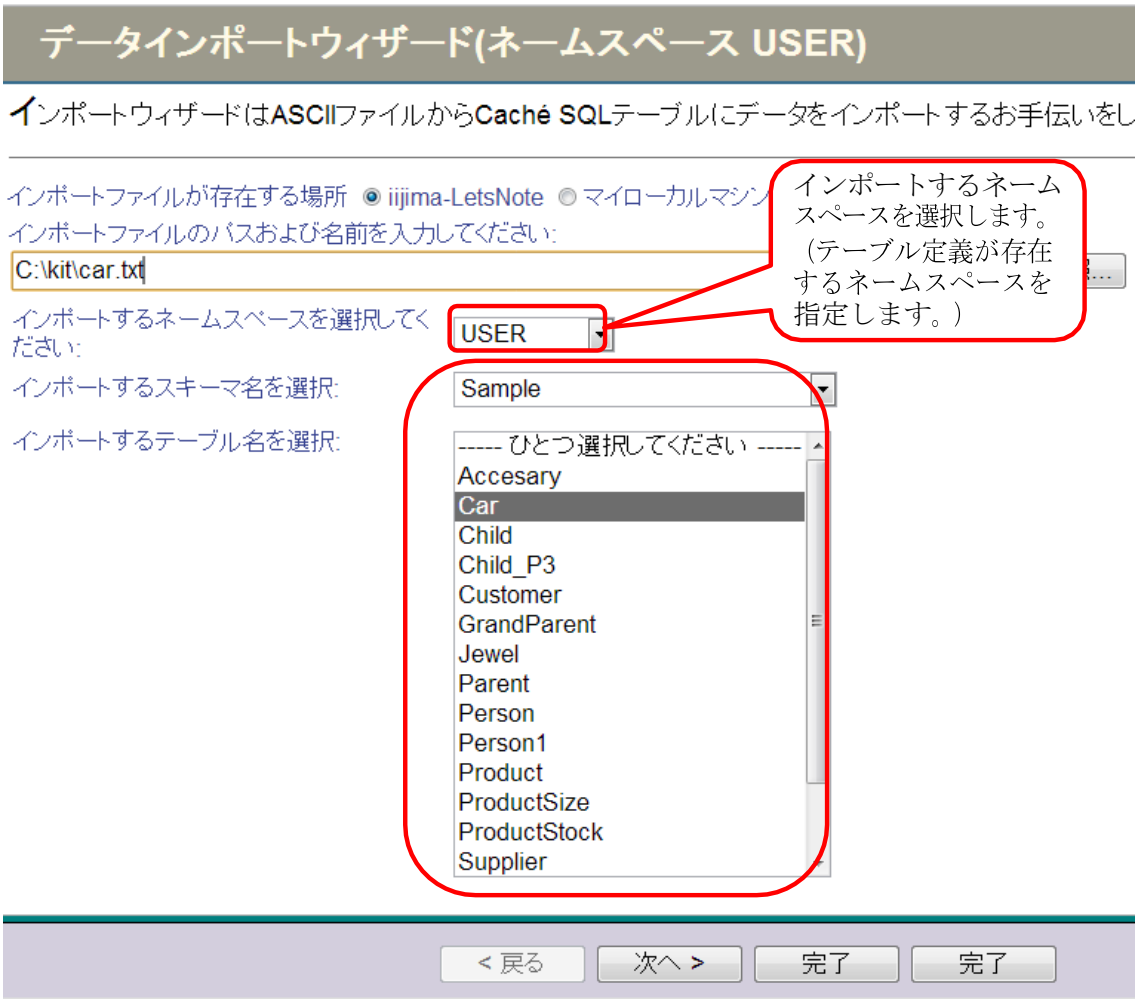


図34 データインポートウィザード スキーマ名.テーブル名の選択選択し終わ

ったら、画面下部の"次へ"ボタンを押下します。

すると、次の画面に遷移します。

この画面で、インポート対象となるカラムを選択します。

インポート対象のカラムを、左側の“利用可能”というボックスから、右側の“選択済み”というボックスに移動させます。移動させるためには、画面中央の矢印を使用します。

次の図は、MakerとNameの両方のカラムを選択済みにした図です。

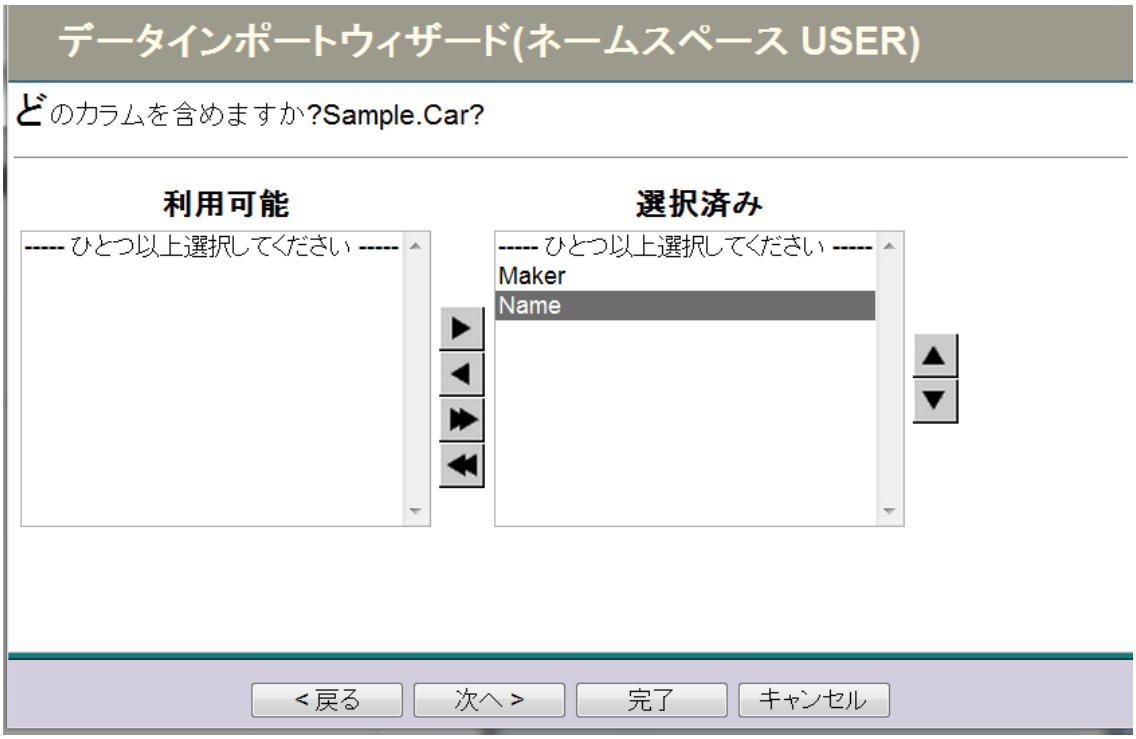


図35　データインポートウィザード：カラム選択

選択処理が終わったら、画面下の“次へ“ボタンを押下します。

すると、以下の画面が表示されます。この画

面上で、以下設定します。

インポートレコードのデリミタ（区切り文字）の指定文字列の

場合の引用符の指定

日付のフォーマットの指定

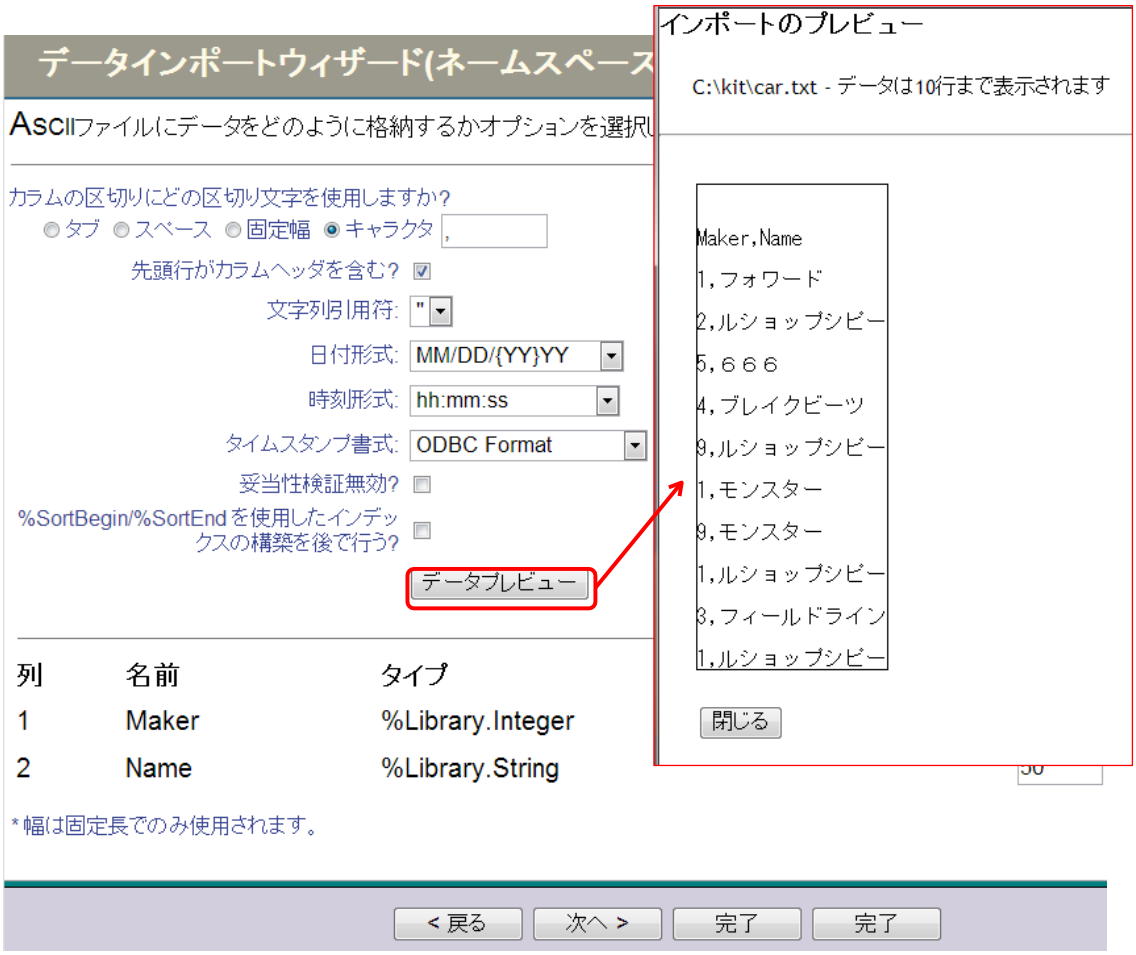


図36 データインポートウィザード：設定確認画面

画面中央の“データプレビュー“ボタン押下で、上記の設定に応じたインポートデータの先頭 10 行を参照できます。

確認ができたら、画面下の“次へ”ボタンを押下します。

すると以下の確認画面が表示されます。

完了ボタンを押下すると、データインポートが実行されます。このインポートはバックグラウンドで行われるため、ボタン押下後には、バックグラウンドタスクページへのリンクが表示されます。

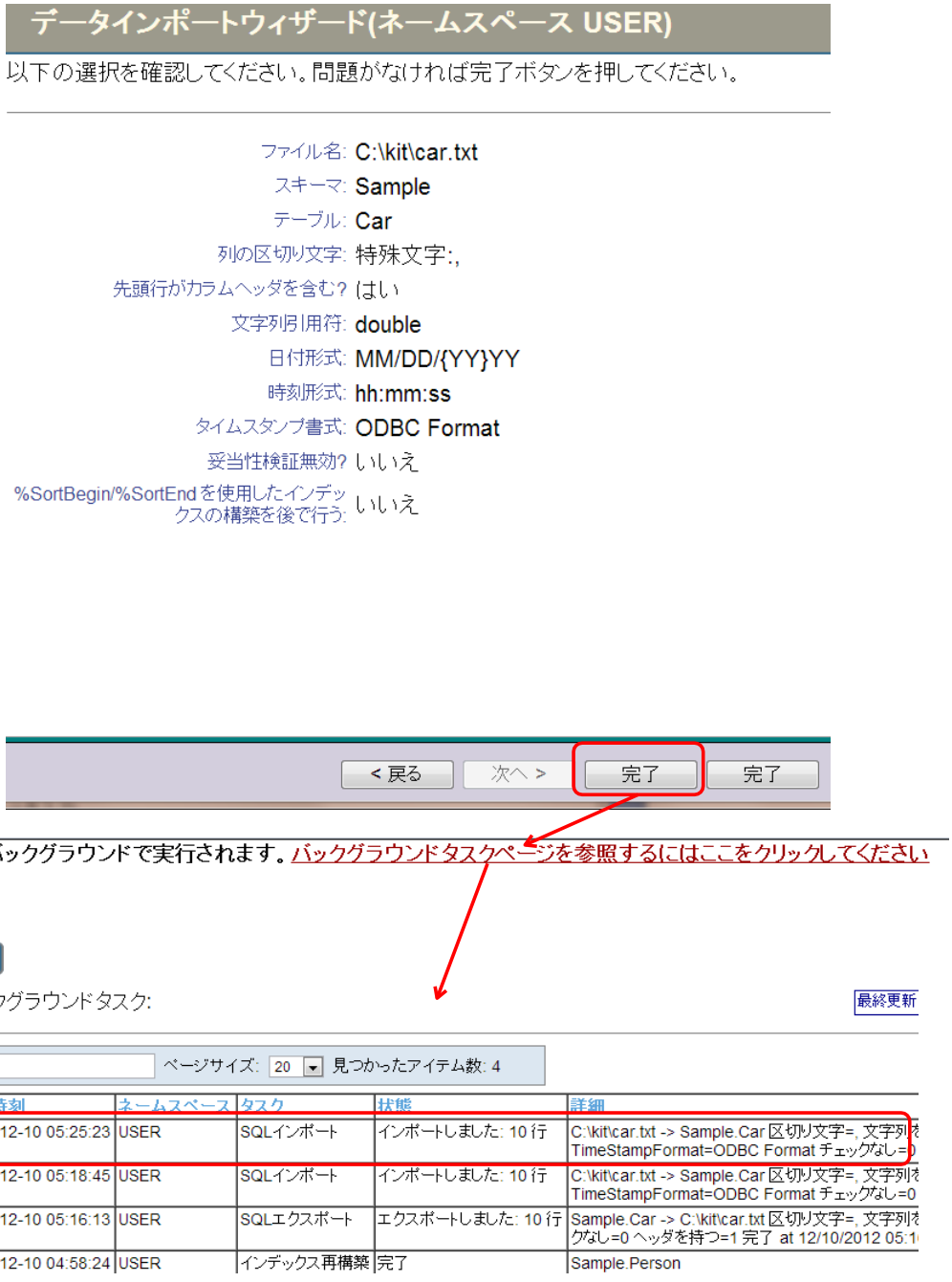


図37 インポートウィザード：インポートの実行

バックグラウンドタスクページで、状態欄が"インポートしました"となっていれば、インポート完了です。

結果は、SQLポータルからインポートしたテーブルの中身を確認してください。

8.3. データ移行ウィザード

SQLポータルには、「データ移行ウィザード」も用意されています。

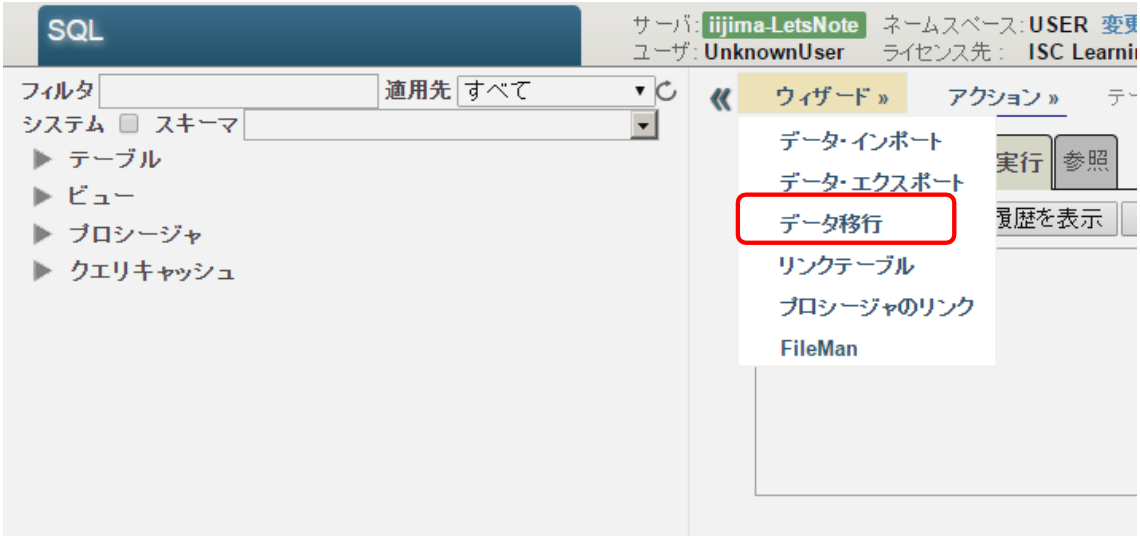


図38 SQL ウィザード（データ移行ウィザード）

これを使用すると、他DBMSのテーブル定義と実際のデータを一度の操作で移行することができます。

この機能を利用するためには、他DBMSシステムに対して、SQLゲートウェイの設定が必要です。

8.3.1. SQLゲートウェイの設定

SQLゲートウェイの設定は、管理ポータルから行います。

管理ポータル→システム管理→構成→接続性→SQLゲートウェイ接続

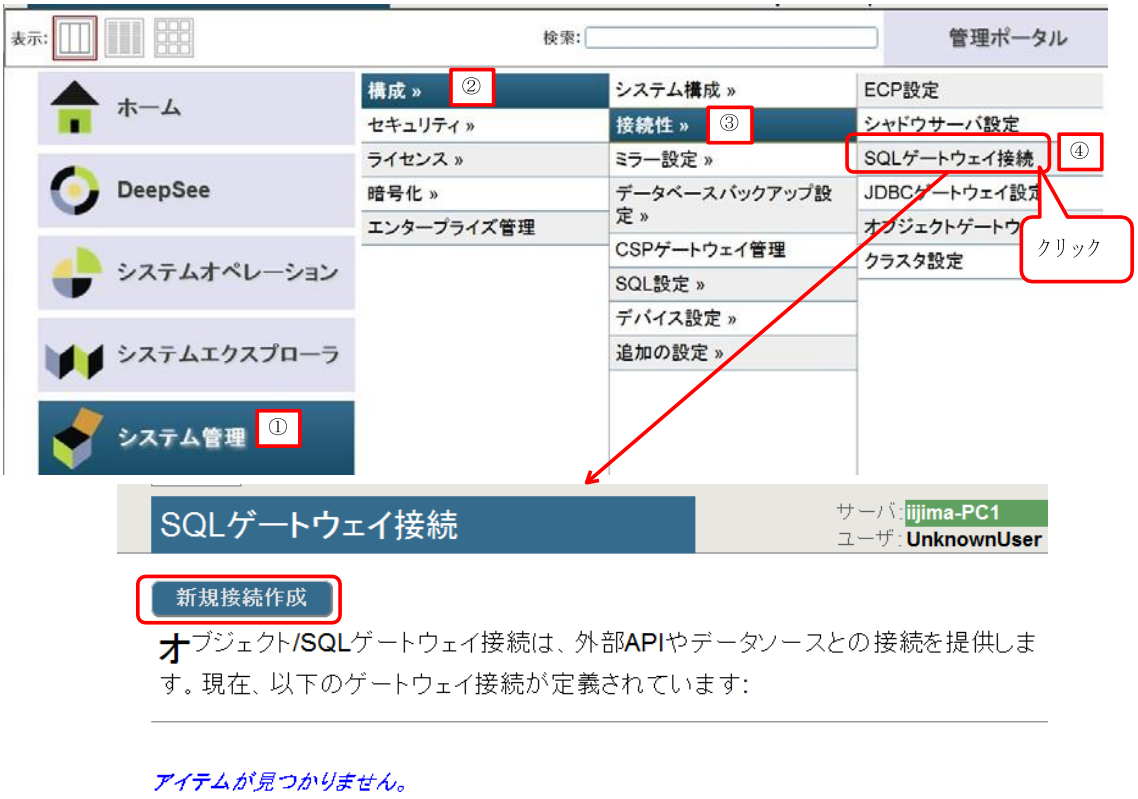


図39 管理ポータル：SQL ゲートウェイ接続メニュー

SQLゲートウェイでは、DSNの指定を行うため、予め作成しておく必要があります。

SQLゲートウェイの新規接続作成画面は、以下の通りです。

画面の「新規接続作成」のリンクを押下すると、ゲートウェイ接続設定画面が表示されます。

ここに事前に定義したDSNと、IRIS内の論理名となる接続名を指定します。

ガイドの例では、IRISに接続していますが、もちろん、IRIS以外の他RDBSを指定することもできます。

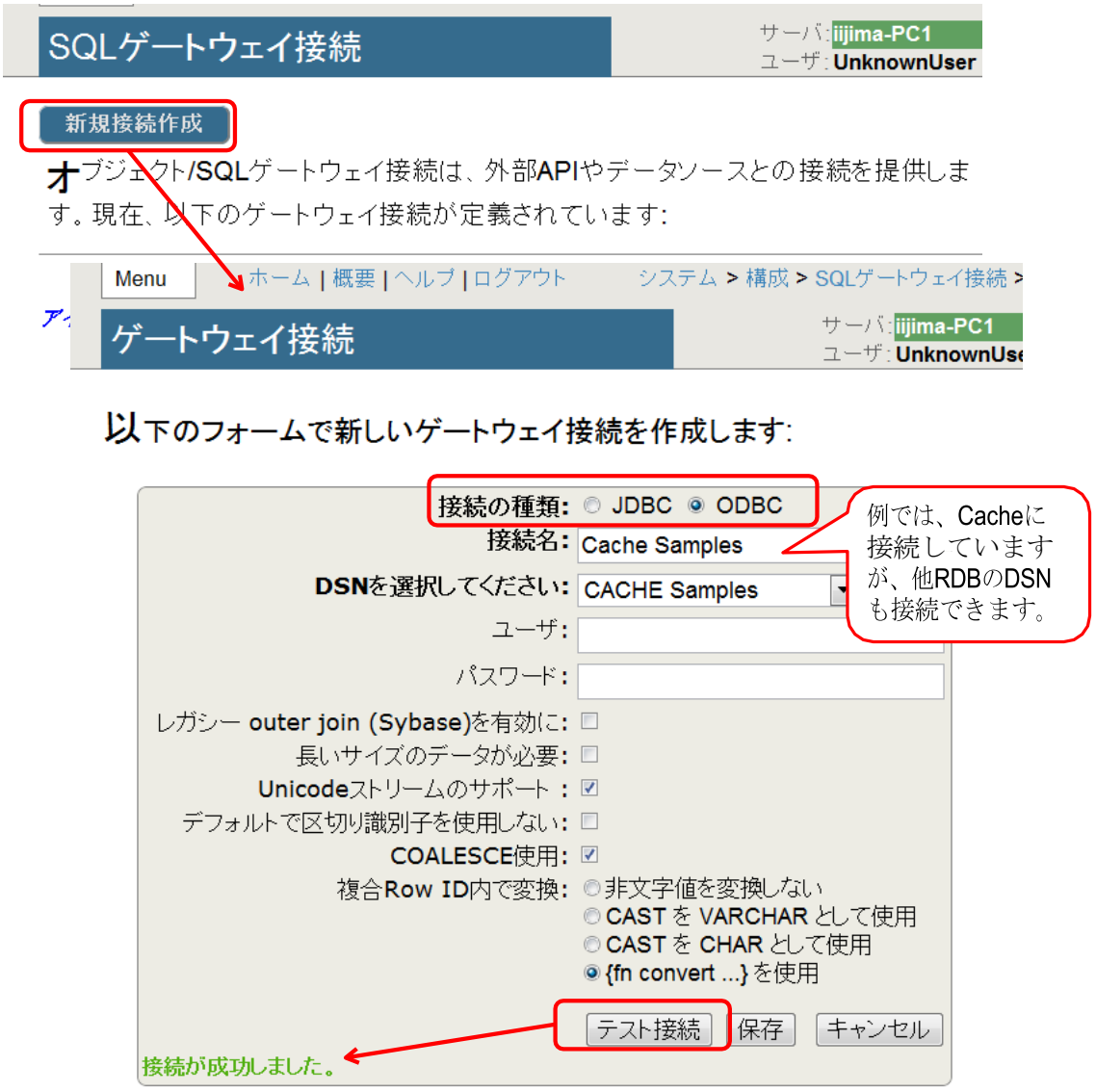


図40 SQL ゲートウェイ 新規接続作成画面

テスト接続ボタンを押下すると、テスト接続が行え、左下にメッセージが表示されます。設定が完了したら、保存ボタンを押下し、設定を保存します。

その後、閉じるボタンを押下すると、1つ前の画面に戻ります。

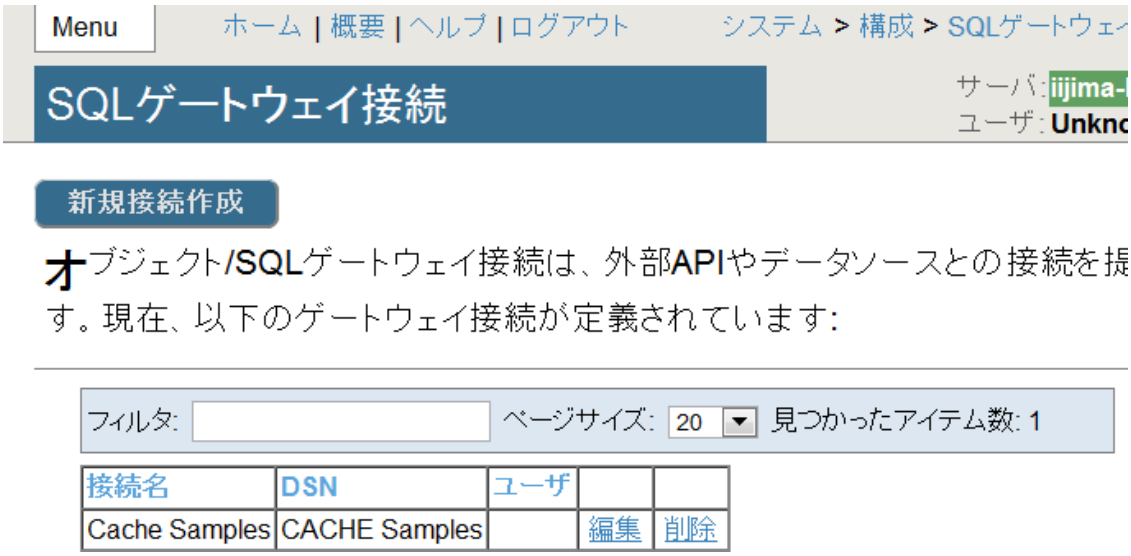


図41 SQLゲートウェイ 新規接続作成後の表示

新規接続を作成するまでは、なにも表示されなかったのが、新たに1件表示されていることがわかります。

8.3.2. データ移行ウィザードの実行

この項目では、SQLゲートウェイ経由の接続確認を行います。

SQLポータルから「データ移行ウィザード」を起動します。

（例では、USER ネームスペースを利用しています。）

画面では、先ほど設定したSQLゲートウェイを選択し、スキーマには[Cinema]を選択します。

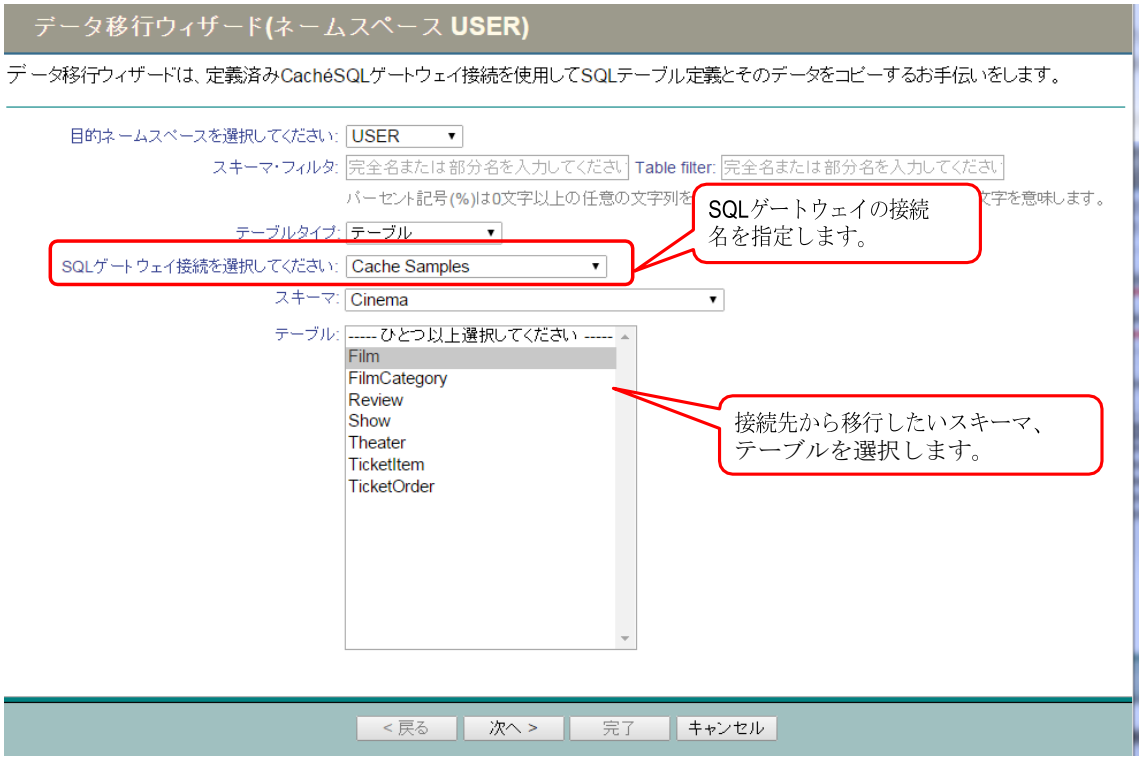


図42　データ移行ウィザード：SQLゲートウェイ接続、スキーマの選択

代表で、Cinemaテーブルを選択して画面下の"次"ボタンを押すと、以下の画面が表示されます。

図43　データ移行ウィザード：テーブルの選択後

完了ボタンを押すと、処理がバックグラウンドで実行されます。

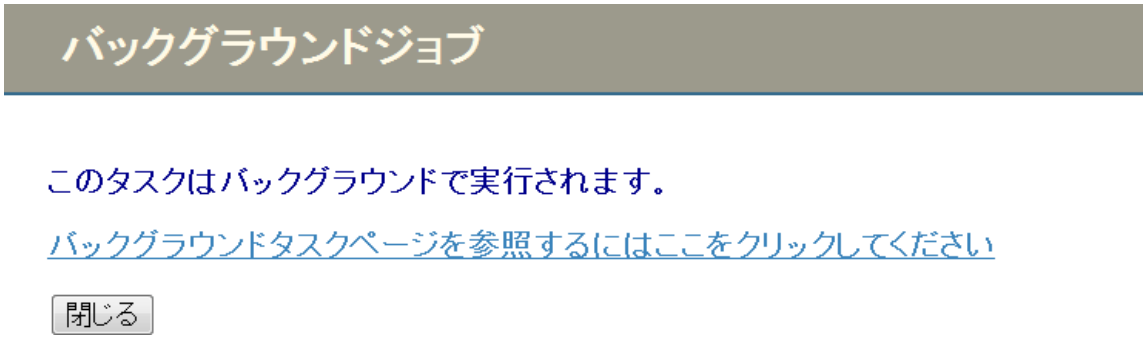


図44 データ移行ウィザード : 完了ボタン押下後

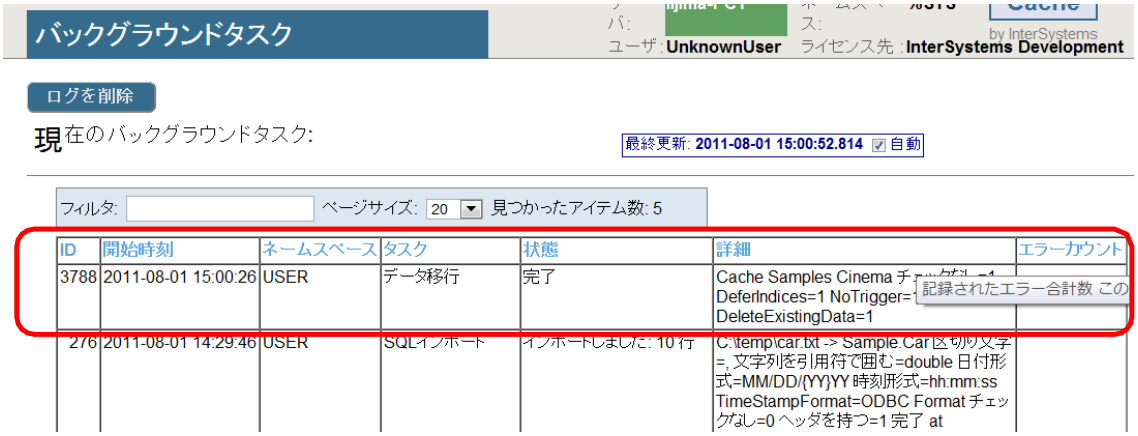
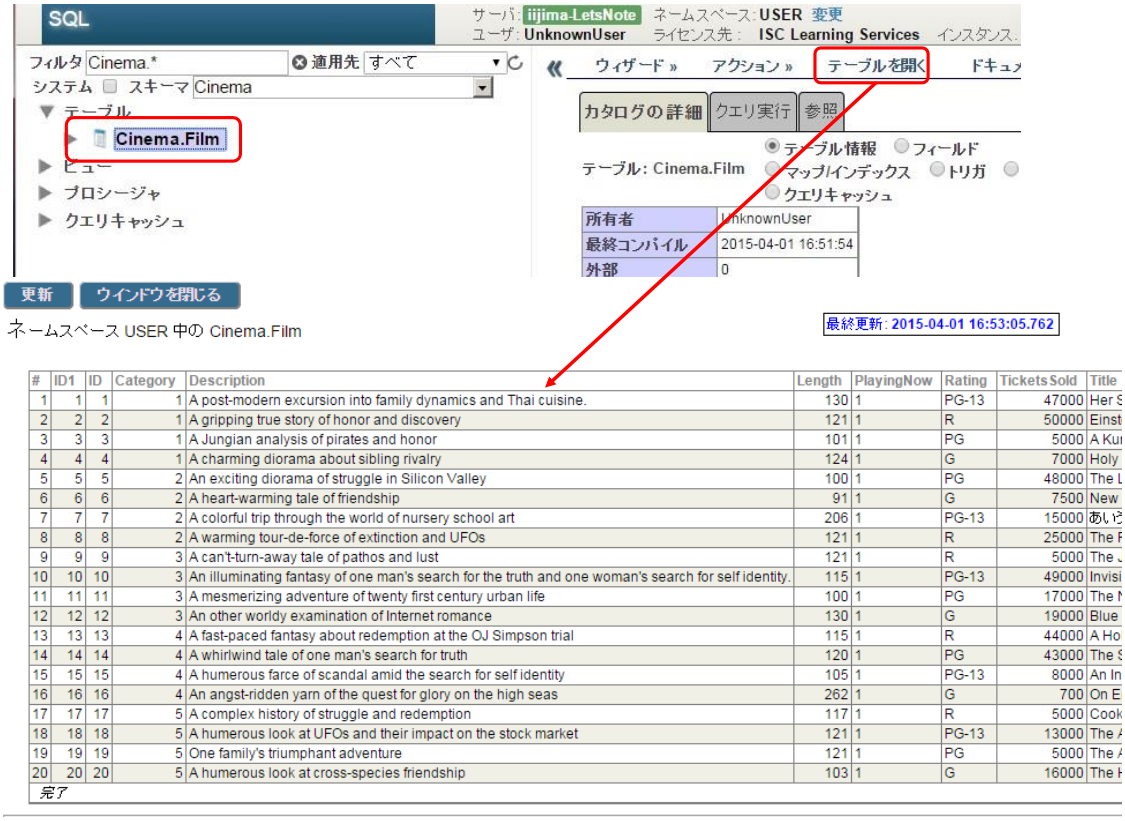


図45 データ移行ウィザード : バックグラウンドタスクのログバックグ

ラウンドのログの状態が"完了"となっていれば接続成功です。

SQLポータルを開いて、Cinemaスキーマにテーブルが作成されていることを確認してください。

この例では、Filmというテーブルが作成されていることを下に示します。



#	ID1	ID	Category	Description	Length	PlayingNow	Rating	Tickets Sold	Title
1	1	1	1	A post-modern excursion into family dynamics and Thai cuisine.	130	1	PG-13	47000	Her S
2	2	2	1	A gripping true story of honor and discovery	121	1	R	50000	Einst
3	3	3	1	A Jungian analysis of pirates and honor	101	1	PG	5000	A Kur
4	4	4	1	A charming diorama about sibling rivalry	124	1	G	7000	Holy
5	5	5	2	An exciting diorama of struggle in Silicon Valley	100	1	PG	48000	The L
6	6	6	2	A heart-warming tale of friendship	91	1	G	7500	New
7	7	7	2	A colorful trip through the world of nursery school art	206	1	PG-13	15000	あいう
8	8	8	2	A warming tour-de-force of extinction and UFOs	121	1	R	25000	The F
9	9	9	3	A can't-turn-away tale of pathos and lust	121	1	R	5000	The J
10	10	10	3	An illuminating fantasy of one man's search for the truth and one woman's search for self identity.	115	1	PG-13	49000	Invisi
11	11	11	3	A mesmerizing adventure of twenty first century urban life	100	1	PG	17000	The M
12	12	12	3	An other worldy examination of Internet romance	130	1	G	19000	Blue
13	13	13	4	A fast-paced fantasy about redemption at the OJ Simpson trial	115	1	R	44000	A Ho
14	14	14	4	A whirlwind tale of one man's search for truth	120	1	PG	43000	The S
15	15	15	4	A humerous farce of scandal amid the search for self identity	105	1	PG-13	8000	An In
16	16	16	4	An angst-ridden yarn of the quest for glory on the high seas	262	1	G	700	On E
17	17	17	5	A complex history of struggle and redemption	117	1	R	5000	Cook
18	18	18	5	A humerous look at UFOs and their impact on the stock market	121	1	PG-13	13000	The A
19	19	19	5	One family's triumphant adventure	121	1	PG	5000	The A
20	20	20	5	A humerous look at cross-species friendship	103	1	G	16000	The H

完了

図46　データ移行ウィザード：完了後のデータ確認

8.4. 他社ツールを使ったデータ移行

現在では、データベース間の移行を意図した様々なツールがあります。

それらのツールは、ほとんどODBCをサポートしていますので、ODBC経由で IRISにDBMSの定義、データを取り込むことが可能です。

たとえば、SQL Serverには、データ変換サービスがあり、Enterpriseマネージャから起動可能です。

データのインポート、エクスポートがサポートされています。

このツールは、SQL Serverへのデータ移行を意図したものですが、インポート、エクスポートとも入力ソース、出力ソースを自由に選択することができます。

従って、このツールを使用して Oracleデータベースから IRISへの移行を行うことも可能です。

9. トランザクション属性

データベースにとってトランザクションは、避けて通れないものです。

IRISでトランザクションを実装するには、アクセス手法（オブジェクトアクセス、SQL アクセス、ダイレクトアクセス）による条件の違いを考慮しなければなりません。

ここでは、SQLアクセス時のトランザクション属性について説明したいと思います。

以下では、トランザクションの ACID属性の個々の観点から説明します。

9.1. Atomicity（最小性）

この属性は、いわゆるAll or Nothingの状態を保障することです。

ObjectScript にはトランザクションを制御するコマンド（Tstart,Tcommit,Trollback）があり、一連のデータベースの更新をこれらコマンドのペアで囲むことにより、最小性を保障します。

更新を伴う SQL 命令（Insert,Update,Delete）は、個々の命令に対して内部的には、暗黙のトランザクション単位となります。

従って、複数の更新系のSQL命令を1まとめにしてトランザクション単位とした場合には、入れ子トランザクションとなります。

つまり、個々の SQL命令でコミットが発行され、一度データベース更新が確定したのち、上位のトランザクションが異常終了した場合には、その確定した更新もロールバックされます。

9.2. Consistency（一貫性）

これは、データの整合性を保障する属性です。

トランザクションの例で必ず例として引用される銀行口座間の振替処理でいえば、引き出し作業によって引き出された金額と、振り込み作業によって振り込まれた金額は同じにならなければならないというものです。

これを保障するためには、多分にアプリケーションでの対応が必要な部分があり、データベースの機能だけでは、これを保障できるものではありません。

IRISは、上記の最小性、整合性制約のサポートを通して、この属性の保障を後方支援します。

9.3. Isolation（分離）

マルチユーザ環境でトランザクション処理を平行稼動する場合には、その個々のトランザクションの分離レベルにより同時実行性、望ましくない現象の発生が変わってきます。

IRISは、現バージョンでは、ISOが定めている分離レベルの内、Read UncommitedとRead Commitedの2つのレベルをサポートしています。

Repeatable ReadとSerializableに関しては、分離レベルでのサポートをしていません。

9.4. Durability（存続性）

これは、一度コミットされたデータは、その後システム障害等が発生しても決して紛失しないという属性です。

これは、IRISのジャーナル機能により、保証されます。

この属性に関しては、SQLアクセスによる特別な要素はありません。