4D ODBC

リファレンス

Windows® and Mac™OS

4D

4D ODBC リファレンス
Windows® and Mac™ OS

このマニュアルに記載されている事項は、将来予告なしに変更されることがあり、いかなる変更に関しても 4D SA は一切の責任を負いかねます。このマニュアルで説明されるソフトウェアは、本製品に同梱の License Agreement（使用許諾契約書）のもとでのみ使用することができます。

ソフトウェアおよびマニュアルの一部または全部を、ライセンス保持者がこの契約条件を許諾した上での個人使用目的以外に、いかなる目的であれ、電子的、機械的、またどのような形であっても、無断で複製、配布することはできません。

4th Dimension、4D Server、4D、4D ロゴ、4D ロゴ、およびその他の 4D 製品の名称は、4D SA の商標または登録商標です。

Microsoft と Windows は Microsoft Corporation 社の登録商標です。

Apple, Macintosh, Mac, Power Macintosh, Laser Writer, Image Writer, ResEdit, QuickTime は Apple Computer Inc.の登録商標または商標です。

その他、記載されている会社名，製品名は、各社の登録商標または商標です。

注意

このソフトウェアの使用に際し、本製品に同梱の License Agreement（使用許諾契約書）に同意する必要があります。ソフトウェアを使用する前に、License Agreement を注意深くお読みください。

目次

目次

序文

4D ODBCとは、4th DimensionデータベースがMacintoshあるいはWindowsプラットフォーム上でODBCデータベースと通信できる4th Dimensionのプラグインセットです。4D ODBCを用いることで、4th DimensionデータベースはODBCデータベースにストアされたデータを表示、操作、修正することができます。

このマニュアルについて

このマニュアルでは、4th Dimensionデータベースで、ODBCによりアクセス可能なデータソースの実行法、使用法、および修正法を説明します。

このマニュアルは4thDimension言語、およびODBCのSQL言語に精通したユーザのために書かれています。新規ユーザは両言語を習熟した後、このマニュアルを読むことを推奨します。

このマニュアルは10章に分かれています。

- 第1章「はじめに」では4D ODBCの主要概念を紹介し、フロントエンド開発の際に必要なデザイン選択のいくつかを説明します。
- 第2章から第4章の「ユーザリファレンス」では、データ管理法を説明します。
  第1の方法は、データソース中のデータと4th Dimension中のデータをリンクするために、コンテキストを使用するものです。第2の方法は、データソースのSQL文を送信、実行するために、下位コマンドを使用するものです。
- 第5章から第10章の「言語リファレンス」では、4D ODBC言語の各コマンドについて説明しています。
- さらに、付録には追加情報があります。付録A「4D ODBCエラーコード」では、4D ODBCで起こりうるエラーコードをリストしています。付録B「SQL Getinfo機能」では、このSQL機能の利用について説明しています。付録C「設定環境」では、ユーザ環境へ4D ODBCをインストールするための追加情報を説明しています。

クロスプラットフォーム

このマニュアルでは、MacintoshとWindows両方での4D ODBCの使用について説明しています。4D ODBC両バージョンの概念、および機能はほとんど同一ですが、マニュアルでは、必要なあらゆる違いについて説明します。そのような違いには、グラフィカルユーザインタフェース、およびキーボードコマンドを含みます。

このマニュアルには、Macintosh環境、Windows環境の両方を説明するグラフィックスを含みます。

4th Dimension、4D Server、および4D ODBC

4D ODBCは、4th Dimension、または4D Serverとともに使用されます。4D ODBCとともに使用する場合、4th DimensionはODBCデータソースのクライアントとなりうるデータベースを作成できます。データベースを持つ各ユーザは、ODBCデータベースへの同時通信、同時使用を行うことができます。

4D Serverによって、複数開発者によるデータベースアプリケーションを作成することができます。4D ODBCとともに使用する場合、4D Serverによって、複数の開発者がODBCデータベースに接続可能です。

このマニュアルでは、4th Dimensionとして、両製品の動作が異なる場合を除き、4th Dimensionと4D Serverの両方を参照しています。

表記方法について

このマニュアルでは、内容が判りやすいように特定の規則を設けてます。

- 次のような説明注釈が使用されます。

  注：このように強調されたテキストによって、4th Dimensionをより効率的に使用できるよう注釈や近道を与えます。

  警告：このような警告によって、データが失われるかもしれない状態を警告します。

- **コマンド**：このマニュアルでは、4D ODBCのコマンドは、特別なフォントを用いて、すべて大文字で印刷されています。例えば、**OC DEBUG MESSAGE**などです。4th Dimensionに追加された4D ODBCのコマンドにはすべて、OCという文字が頭についています。
- **機能**：4D ODBCの機能は、頭文字が大文字で示されています。例えば、**OC Set SQL in cursor**などです。4D ODBCの機能にはすべて、OCという文字が頭についてます。
- **ファイル名**：さらに、フィールド名、レイアウト名、および他の項目とファイル名とを区別しやすくするため、すべてのファイル名は本文中でカッコの中に記述されています。例えば、会社のファイルは、［会社］ファイルと記述します。

はじめに

このマニュアルは、統合4D ODBCのデザイナ、管理者、およびユーザに対するリファレンスガイドとして役立ちます。このマニュアルでは、統合アーキテクチャ、ODBCデータソースの性能に精通し、4th Dimensionのプログラミング言語、およびODBCドライバ上で利用可能な機能を熟知したユーザを想定しています。

4th Dimensionは、Macintosh、Windows用の強力なデータ管理ツールです。4D ODBCで開発されたアプリケーションは、マイクロコンピュータの関係データベースの出力とグラフィカルインタフェースを結びつけます。

4D ODBCによって、4th DimensionとODBCデータベース両方の出力を利用するアプリケーションを開発することが可能です。4D ODBCを利用して、SQLデータベースにストアされたデータに、4th Dimensionからアクセスすることが可能です。

ODBCアーキテクチャ

オープン・データベース・コネクティビティ（ODBC）はストラクチャード・クエリ・ランゲージ（SQL）を利用して、4th Dimentionのようなアプリケーションがデータベース・マネジメント・システム（DBMS）にアクセス可能にする機能ライブラリを定義します。ODBCインタフェースは、異なるデータベース・マネジメント・システムに対し、ベンダー・ニュートラルなアクセスを提供します。

ODBCアーキテクチャは4つの構成要素を持ちます。

1. アプリケーション
2. ドライバマネージャ
3. ドライバ
4. データソース

ODBCドライバが提供する主要な機能は、次のものを含みます。

- DBMSとの通信、および解除
- クエリの実行、および記憶エリアとクエリ結果の提供
- オンライントランザクション処理の考慮
- ODBCインタフェース形式の機能（DBMSの具体的機能）
- ドライバマネージャは、異なるODBC機能に単一のエントリポイントを提供しながら、ドライバをロードする動的リンクライブラリ（DLL）です。

通信選択

4D ODBCアプリケーションを設計する第一段階は、通信するデータベースの選択です。ODBCは、利用可能なデータベースを知らせ、データベースのタイプを記述し、通信の確立する多くの機能を提供しています。

アプリケーションは、特定の目標データベースで設計できます。例えば、経理課はORACLEデータベースにレコードを記録するかもしれません。ドキュメントシステムを設計する場合には、ORACLEドライバが必要と知らなければなりません。データベースのタイプ、データベースとの通信に必要な属性についても知っていなければなりません。アプリケーションを目的のデータベースのために設計する場合、DBMSやドライバで提供される具体的機能を利用できます。

代わりに、データベースを操作するために、アプリケーションを設計する必要があり得ます。あらかじめ使用するドライバ、通信するデータベースは分からないことが考えられます。この場合、すべてのODBCデータソースに共通な特徴を使うだけのために、開発者は警告を使わねばなりません。4D ODBCによって、開発者は2つのシナリオのうちどちらかのためにアプリケーションを開発できます。

The Preferences File（4D ODBC v6）

4D ODBC preferences fileの名前は、4D ODBCバージョン1.5と6.0との同時利用を考慮するため、変更されてきました。新しい名称は、Windowsでは「ODBV6Prf.RSR」、Macintoshでは「ODBV6Prf」です。

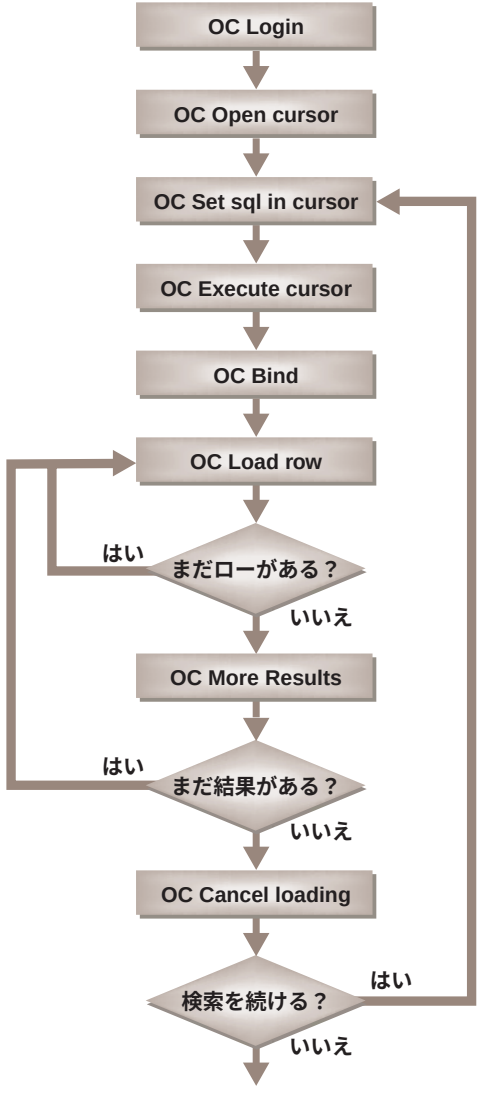
下位レベルコマンドとコントロールコマンド　2

この章では、4D ODBCの下位コマンドおよびコントロールコマンドを用いて、データソースにアクセスする重要な局面を検討します。これらはODBC API呼び出しと名称、文法、機能が非常によく似ています。この章ではODBC処理、機能について詳細な分析は行いません。ここでは、次の4D Drawドキュメントを管理する基本的な事柄について説明します。

個々のODBC機能は7つのグループに分類できます。これらの個々の手続き群によって、異なるコミュニケーションの場所で、データソースと相互作用します。

- 接続
- カーソル
- SQL文の実行
- 結果の受け取り
- 処理の取り消し
- エラー操作
- トランザクションコントロール

次のダイヤグラムでは、4D ODBCのルーチンを用いて、4th Dimensionとデータソース間の通信を説明しています。



通信

ODBCデータソースへの初期通信はログインコマンドで操作されます。これらのコマンドはデータソースへの通信を作成し、与えられたユーザネームとパスワードでログインします。

4D ODBCは通信の作成に2つの方法を与えます：

- **OC Login dialog**
- **OC Login**

OC Loginダイアログの機能はダイアログボックスを表示して、ユーザがデータソースを選択し、ログインネームとパスワードを指定することを可能にします。

OC Loginはユーザ名、パスワード、データソース名を送信します。このコマンドを使用して、要求された値をハードコードするか、ユーザに値を要求するために自信のインタフェースを生成します。

同時に同種あるいは異種のデータソースに対して、複数の接続を確立することはできません。

カーソル

一旦コネクションが確立すれば4th DimensionとODBC間のダイアログを開始できます。カーソルは2つのアプリケーション間の通信を管理するハイレベルのメカニズムです。

カーソルとはクエリに起因する個々の列を一度に1つ処理できるメカニズムです。カーソルはODBCデータソース上のプロセスと見なすことができます。各々のカーソルはODBCデータソース上のデータを選択、挿入、更新、削除するSQL文と生成された後に接続されます。複数のカーソルは同時に生成、使用できますが、多くのカーソルのオープンはパフォーマンスの低下を生じさせます。

プロセスの確立を要求する時間は次の機能に従って変化します：

- オペレーションシステムの使用
- データサーバの構成
- サーバに接続しているユーザ数

4D ODBCのルーチンは、2つの異なる方法でカーソルの処理を管理します。1つはカーソルを独自生成するもので、もう1つは既にあるカーソルを要求するものです。

SQL文の実行

SQL文の実行には準備実行と直接実行の2つの方法があります。アプリケーションにSQL文を並列的に実行させるなら準備実行を使用します。準備実行によってパラメータの値を変更することができます。クエリの実行に用いられる最適化情報を準備して、ルーチンが実行される毎の時間を修正する必要がないので、SQL文が並列的に実行される時、準備実行は効率的です。

アプリケーションがSQLのリクエストの完成の前に結果について情報を要求しない時、アプリケーションにSQLリクエストを一度だけ処理させるならば直接実行を使用します。

SQL文の使用準備

ODBCを利用する時、SQL文中にパラメータを埋め込むことで動的なSQL文を構成できます。動的なSQL文はSQL文中にパラメータマーカーを埋め込んで構成されています。SQLパラメータマーカーの1つはホスト変数としてされるクエスチョンマーク（?）です。

同じSQL文の実行を繰り返す必要がある時は動的なSQLのパラメータやカーソル、及び**OC Execute cursor**中の**OC Set SQL**の組み合わせを使用しなくてはなりません。

動的なSQLの利用にはいくつかの利点があります：

- パラメータを知る以前にSQLコマンドを構成してもかまいません。
- 値を変更する場合、カーソル中の**OC Set SQL**によって代入変数を用いたSQLコマンドを再分析する必要がありません。
- SQLの最適化情報は実行ごとに保存されます。

次の方法はSQL挿入分を生成するため、配列からの値を入力パラメータとして用いています。

この例では、配列のような4Dインタフェースオブジェクトを ODBC SQL関数的に組み合わせることにより、達成した最適化を示します。

```
C_STRING (35;vState)
ARRAY STRING (30;arState;10)
arState{1}:="U.S.A."
arState{2}:="France"
arState{3}:="Great Britain"
arState{4}:="Mexico"
arState{5}:="Spain"
arState{6}:="Hong Kong"
arState{7}:="Japan"
arState{8}:="Germany"
arState{9}:="Canada"
arState{10}:="Austria"
$sql:="insert into markets values (?)"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind parameter (cursor_id;1;"vState";1)

For ($i;1;10)
   vState:=arState{$i}
   $res:=OC Execute cursor (cursor_id)
End for
```

SQL文の直接使用

OC Execute Direct cursorは、1つの単体ルーチンの中でカーソル及び**OC Execute cursor**のOC Set SQL機能を結合しています。**OC Execute Direct cursor**はその場限りのSQL文及び一回のみ呼び出される実行文の生成に役立ちます。DBMSは実行完了後、SQL文の実行時に使用されたあらゆる最適化情報を破棄します。

▼ 次の例ではSQL削除コマンドを実行するために、直接SQL実行を用いています。

```
$sql:="delete from sales where stor_id = '6380'"
$result:=OC Execute direct cursor (cursor_id;$sql)
```

結果の受け取り

データソースからデータを直接検索するには**OC Bind**を使用します。多くの4D ODBCルーチンは**OC Bind**をベースとしています。**OC Bind**は4th Dimensionオブジェクトとデータソースオブジェクトをバインドしています。この機能は4th Dimensionデータベースの変数およびフィールドとテーブルXの列Aをバインドするために使用できます。

OC Bindは2つのオブジェクト間のバインド、すなわちODBC側のSQL列と4th Dimension側のフィールド、変数、配列とのオブジェクト間バインドを定義します。一度オブジェクト間でバインドすれば**OC Local row**コマンドによってODBCデータソースから関連のある4th Dimensionオブジェクトへと選択した変数を転送できます。**OC Load row**はより詳細な結果が利用かを示す値を返します。

データ検索オペレーションには3つのフェーズがあります：

- コマンドの送信及び実行
- 結果の宛先の決定
- 結果の検索

OC Bindの機能によってSQLサーバの列を3種類の4th Dimensionのオブジェクトとバインドができます：

- 変数
- 配列
- フィールド

変数のバインディング

変数のバインドはユーザが選択した特定の列上の詳細な情報を検索するように、一度につきデータ1列のみを検索するために適した方法です。この方法は最も単純な方法ですが、列が複数ある場合、結果を格納または再利用できません。この方法は、メッセージ生成、列の表示、及びレコードへデータを格納する際の一時記憶に適しています。

使用例：

```
C_STRING (30;var1)
C_STRING (30;var2)
C_STRING (30;var3)
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind (cursor_id;1;"var1")
$res:=OC Bind (cursor_id;2;"var2")
$res:=OC Bind (cursor_id;3;"var3")
$res:=OC Execute cursor (cursor_id)
$res:=OC Load row (cursor_id)
```

配列のバインディング

列のバインドは、リスト中で必要な複数列を検索するのに適した方法です。多くの4D Connectivityディベロッパーは4D出力設計に相当するものを生成するためにこの方法を利用します。また、この方法はスクロール可能エリア、リスト、ポップアップメニューといった4th Dimensionインタフェースツールのサポートに適しています。例えば、形式的、且つ独特なキーのコンポーネントを検索して、続いてより読みやすい形態へと書き換えるために使用されます。配列をバインドする際、文字列配列や整数配列といった配列コマンドを使用した配列をあらかじめ定義する必要があります。最適な4th Dimensionデータタイプを用いて配列を定義してください。

配列をバインドして**OC Load row**を複数回呼ぶ時、4D ODBCは自動的に新しい要素を配列に追加し、それらの要素に値をロードします。できる限りテキスト配列（ARRAY TEXT）より長さが固定された文字列配列（ARRAY STRING）を使用してください。

使用例：

```
ARRAY STRING (30;ar1;0)
ARRAY STRING (30;ar2;0)
ARRAY INTEGER (ar3;0)
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind (cursor_id;1;"ar1")
$res:=OC Bind (cursor_id;2;"ar2")
$res:=OC Bind (cursor_id;3;"ar3")

$res:=OC Execute cursor (cursor_id)
$res:=OC Load row (cursor_id)

While ($res=1)
   $res:=OC Load row (cursor_id)
End while
```

一方、直接配列へ結果を格納するという単純な目的なら、**OC Query exec**はSQL Serverデータを配列にロードするためにより有効な方法を提供します。**OC Query exec**はSQLコマンド実行のために直接実行方式を用います。

使用例：

```
ARRAY STRING (30;ar1;0)
ARRAY STRING (30;ar2;0)
ARRAY INTEGER (ar3;0)
   ` 配列の宣言
$sql:="select * from authors where city = 'Oakland'"
$col:=OC Query exec (Login_ID;$sql;-1;"ar1";"ar2";"ar3")
```

フィールドのバインディング

4th Dimensionフィールドの直接使用は、レポートおよびラベルの次の項目に対して中間データ記憶を提供します。それは4th Dimensionの核に近いものです。そのような場合、レコードの作製及びエントリをプログラムに基づいて管理しなければなりません。

```
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind (cursor_id;1;"[authors]ID")
$res:=OC Bind (cursor_id;2;"[authors]Last")
$res:=OC Bind (cursor_id;3;"[authors]First")
$res:=OC Execute cursor (cursor_id)
$res:=OC Load row (cursor_id)
```

結果の分析

結果を生成するSQLコマンドを実行後、それらの結果の種類を分析するために4D ODBCルーチンを使用できます。

OC Number rowsプロセスと**OC Number of colmuns**を使用すれば、コマンドによって処理される列の数及び結果セットで選択される列の数を測定できます。

OC Describe column及び**OC Column**アトリビュートは、結果セットで検索される列の種類の情報を提供します。

おそらく最も役立つものは**OC Load row**機能の戻り値でしょう。**OC Load row**は列が追加されていれば1を返し、そのままならば0を返します。この情報を使用すれば、結果セットから全ての列を返すプロシージャを設計することが可能です。

▼ 次の例は、4D配列に列をロードするために**OC Load row**コマンドを使用したものです。Whileループに注目してください。残りの行がないかをテストしています。

```
ARRAY STRING (30;ar1;0)
ARRAY STRING (30;ar2;0)
ARRAY INTEGER (ar3;0)
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind (cursor_id;1;"ar1")
$res:=OC Bind (cursor_id;2;"ar2")
$res:=OC Bind (cursor_id;3;"ar3")
$res:=OC Execute cursor (cursor_id)
$res:=OC Load row (cursor_id)
While ($res=1)
   $res:=OC Load row (cursor_id)
End while
```

分析コマンド

SQL文は2つの異なるカテゴリに分類できます：

- たった1つの文を含むコマンド（1つの結果セット）
- 実際には複数のSQLコマンドからなるストアードプロシージャのような複合コマンド（複数の結果セット）

OC More resultsによりさらに結果セットがあるかどうか判定するためのSQLコマンドを分析することができます。

アプリケーションは次に移る前に1つのセットの全結果を検索するよう要求されません。**OC More results**が呼ばれる時、前の結果セットは破棄されます（**OC Cancel loading**と同様に）。したがって、現在の結果セットから希望の列すべてをロードした後でこの機能を使ってください。

アプリケーションによっては、ユーザ自身がSQL文を入力できるなら、それを呼び出す前にプロシージャ中のコマンドの数を知らなくても構いません。それゆえ、コマンドが完全に実行されたことを確認するために**OC More results**を用いてデータソースコマンドバッファを読まなければなりません。

次の式はSQLサーバストアードプロシージャを実行し、すべての結果セットから最初の3列の値を返し、返された結果の数を記述した「警告」ボックスを表示したものです。**OC Load row**及び**OC More results**からの結果を用いることで生成された列や結果セットに関係なく、いくらかのコマンドの結果を検索する包括的なルーチンを開発することができます。

```
$sql:="exec sp_help titles"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
   ` 配列のバインド
$res:=OC Bind (cursor_id;1;"ar1")
$res:=OC Bind (cursor_id;2;"ar2")
$res:=OC Bind (cursor_id;3;"ar3")
$res:=OC Execute cursor (cursor_id)
$count:=1
$more:=1
While ($more=1)
   $res:=1
   While ($res=1)
      $res:=OC Load row (cursor_id)
   End while
   $res:=OC Execute cursor (cursor_id)
   $more:=OC More results (cursor_id)
   $count:=1+$more
End while
ALERT (String ($count)+" set of results returned")
```

処理のキャンセルとトランザクションコントロール

処理のキャンセル

4D ODBCで実行されたあらゆる処理は、**OC CANCEL LOADING**コマンドを呼び出すことでキャンセルできます。**OC CANCEL LOADING**によって、最近実行されたクエリからあらゆる列をキャンセルすることが可能です。

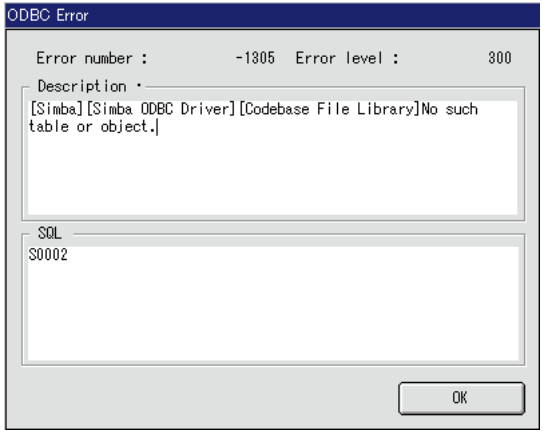
トランザクションコントロール

トランザクションコントロールによって処理をコミットあるいはロールバックすることができます。ODBCのデフォルトのモードは「auto-commit」モードで、すべてのSQL 文は完全処理です。

しかし、**OS Set login option**関数を使用することで「manual-commit」モードを許可する接続オプションを設定できます。「manual-commit」モードを使用しているならば手動で処理をコミットする**OC TRANSACT COMMAND**コマンドを使用する必要があります。

エラー処理

4D ODBCの「エラーマネジメント」モードにはバックグラウンドで動作し、次のダイアログボックスでエラーメッセージを表示する標準エラーマネージングプロシージャを使用しています。

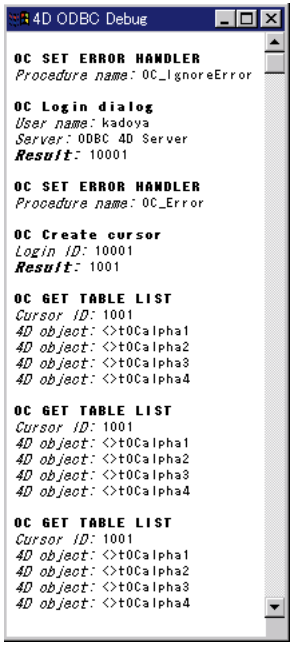


このダイアログボックスでエラー番号、エラーレベル、エラーメッセージを表示します。また、4th DimensionのプロセスIDとエラーのSQLステータスを表示します。4D ODBCエラー番号、エラーメッセージをリストアップすることについては「付録A」

「4D ODBCエラーコード」を参照してください。

4D ODBCコントロールコマンドの**OC SET ERROR HANDLER**コマンドは、エラーが起こる度に実行されるエラーハンドリングプロシージャをインストールしています。これによって、起こり得る実行エラーをコントロールし、デフォルトエラー処理を受け付けないように設定することができます。エラーハンドリングプロシージャの使用でカスタマイズされたエラー処理ルーチンを生成、または分析のためのローカル4Dデータベースに対するエラー発生を記録するメッセージを仕掛けることができます。

開発の間、4D ODBCルーチンの処理をトレースする4D ODBCデバッグウインドウを使用できます。



デバッグウインドウによってすでに呼び出された各ルーチンを記録し、パラメータや各ルーチンの結果を表示します。"trace on error"オプションによって4D ODBCエラーが生じた時はいつでも、コードを検査するために自動的に「4Dトレース」モードを呼び出します。

コンテキストの利用

コンテキストは4D ODBCオブジェクトとODBCデータソースオブジェクトです。コンテキストをデータ操作に用いる場合、サーバ上のエントリの選択、挿入、更新、削除のオペレーションはローカル4D ODBCのオペレーションと同じ方法で現されます。4D ODBCインタフェースはホストデータの管理の為に使用しています。

この章ではコンテキストの利点と実行、戦略の概要を紹介します。コンテキストコマンドで使用される手続き、フロー、及びコンテキスト作成法、データ検索のためのコンテキスト使用法について説明します。

コンテキストコマンドの完全な説明については、第8章「OCコンテキストの定義」及び第11章「OCコンテキストの利用」を参考にしてください。

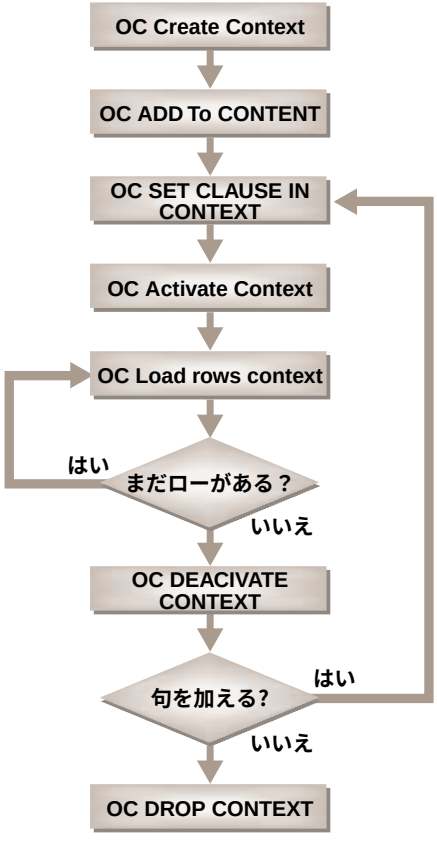
コンテキストの利点

下位レベルDB-Library呼び出しのみを用いて4D ODBCアプリケーションを開発することができますが、コンテキストはいくつかの利点を開発者に提供しています。

- **データ選択**：コンテキストによってSQLプログラミングを使わずにデータソースからデータを選択することができます。実際にグラフィカルインタフェースによってコンテキストコードは生成されます。
- **データ編集**：コンテキストが確立された後は冗長なSQLプログラミングを使用しなくてもシンプルなコマンドによって、データソース列を更新、挿入、削除できます。
- **SQL互換性**：コンテキストは標準のSQLとODBLルーチンの接続に使用できます。
- **複数のコンテキスト**：クライアント／サーバアプリケーションのデザインにおいて開発者に広い柔軟性を許し、多くのコンテキストを同時に開くことができます。
- **4th Dimension風プログラミング**：コンテキストは4D ODBCプログラミング接続で統合されたSQL開発環境を開発者に提供します。この理由のために、経験豊かな4D ODBC開発者は、Transacts SQLを使用したアプリケーションを実行することよりも、コンテキストを使用したアプリケーションを開発する方が容易だと分かることでしょう。

コンテキストフェーズ

次のダイアグラムでコンテキストマネージメントコマンドの使用を説明します。



大部分のコンテキストコマンドは次のフェーズ、すなわちコンテキスト定義、コンテキスト選択、コンテキスト編集と結び付けられます。

コンテキストの定義

コンテキスト定義フェーズはコンテキストを初期化するために使用します。初期設定の結果は、特定のコンテキストの選択及び編集を行うために、後で使用できるコンテキストIDです。

次のコマンドはコンテキスト定義に使用されます：

- **OC Create context**
- **OC Create context dialog**
- **OC Local context file**
- **OC ADD TO CONTEXT**

  コンテキストの定義のために設定されるだろうデータソーステーブルを指定しなければなりません。

  コンテキストの確立に**OC Create context**を使用するなら、**OC ADD TO CONTEXT**によって4D ODBCオブジェクトと選択されたデータソースコラムをバインドしなければなりません。**OC ADD TO CONTEXT**ではデータオブジェクトのソースを用途を指定しています。

  コンテキスト定義の詳細は、後述の「コンテキストの作成」を参照してください。

コンテキスト選択

コンテキスト選択フェーズコマンドはデータソーステーブルの列を指定、検索します。

次のコマンドはデータ選択に使用します：

- **OC SET CLAUSE IN CONTEXT**
- **OC EDIT CLAUSES IN CONTEXT**

  バインディングが生成された時、節を定義できます。節によって、データソースからどのデータを選択するかを指定できます。次の節はコンテキストでサポートされています。すなわち、Where、Group by、Having、Connect By、Start withです。これらの節はいつでも変更できます。

コンテキストのアクティブコントロール

コンテキストが定義され、select節が追加した後、データの検索または編集を行うに先立ち、コンテキストをアクティブにする必要があります。コンテキストを結合した節を変更するため、はじめにコンテキストを非アクティブ化しなければなりません。

次のコマンドはコンテキストのアクティブ化に使用します：

- **OC Activate context**
- **OC DEACTIVATE CONTEXT**

コンテキストへのデータ検索

一度、コンテキストの定義と節をアクティブ化したら、次のコマンドをデータ検索に使用できます：

- **OC Load rows context**
- **OC Previous in context**
- **OC First in Context**
- **OC Last in Context**
- **OC Goto in context**

OC Load rows contextはコンテキストを満たすように要求しただけの列を返します。

コンテキストのデータ操作

コンテキストを定義し、特定の列が検索された後、レコードを更新、削除、挿入するために次のシンプルなコマンドを使用できます。：

- **OC Update in context**
- **OC INsert in context**
- **OC Delete in context**

コンテキストの中のデータを削除または更新するために、データソーステーブルに少なくとも1つの論理的、ユニークキーを宣言しなければなりません。編集可能なものとして、コンテキストを宣言する度に、4D ODBCはユニークなインデックスを探します。キーが存在しない場合、編集は行えません。

コンテキストコマンドの完全な説明については、第8章「OCコンテキストの定義」と第11章「OCコンテキストの利用」を参照してください。

コンテキストの作成

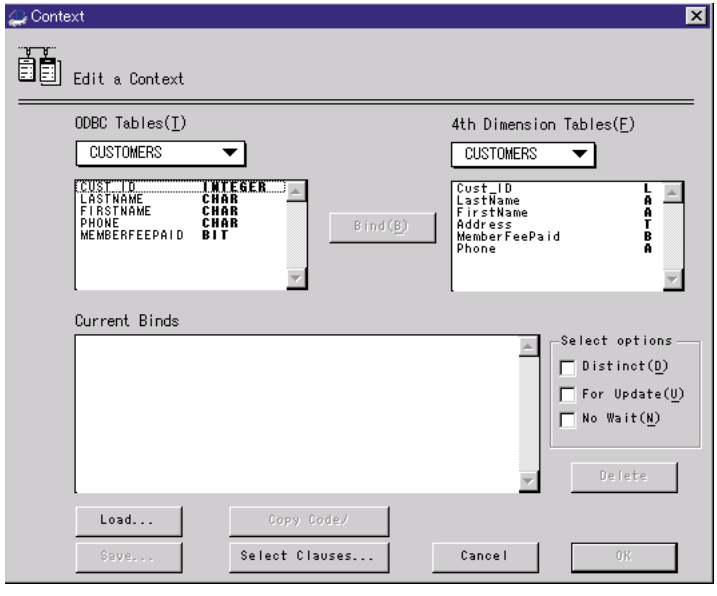
「コンテキスト定義」ダイアロボグックスの使用、または手続き文セットの実行でコンテキストを作成できます。ダイアログボックスを使用する場合、4D ODBCはコンテキスト定義文を生成します。

ダイアログボックスを使ってコンテキストを作成する

データソースサーバ上のデータの選択、及び4th Dimensionのデータ要素をそれと関連付けるコンテキストを定義するのに「Edit a Context」ダイアログボックスを使用できます。

「Edit a Context」ダイアログボックスを開くには：

1. **OC Create a Context Dialog Box機能を実行する。**



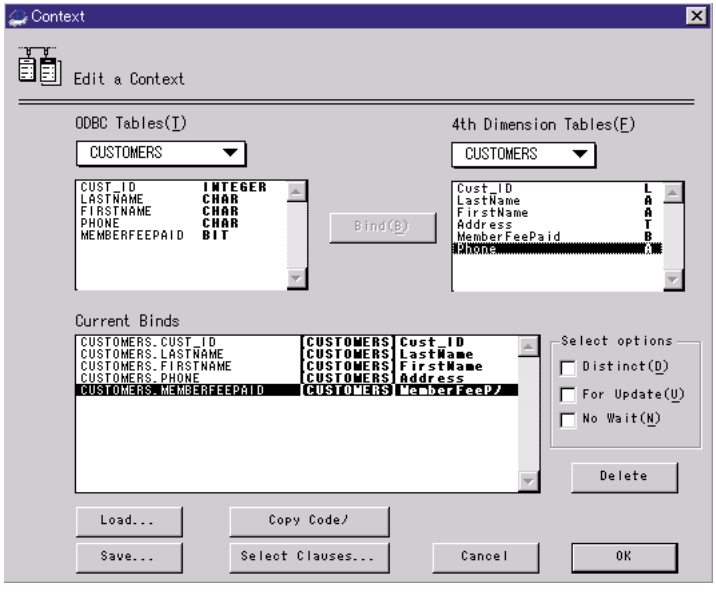
ダイアログボックスを使用して次のことができます：

- コンテキストのバインドの定義
- コンテキストへの節及びオプションの追加
- クリップボードへのコンテキスト定義コードのコピー
- ファイルへのコンテキスト定義のセーブ
- ファイルからのコンテキスト定義のロード

バインドの定義

「Edit a Context」ダイアログボックスによって、データソースの列または式、及び4th Dimensionフィールド、変数、配列を選択し、それらの2つを互いにバインドできます。

複数のデータソースオブジェクトを複数の4th Dimensionオブジェクトと関連付けるために複数のバインドを作成できます。

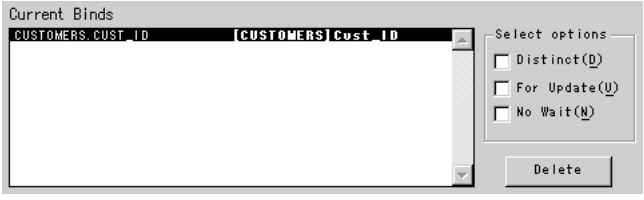


作成した各バインドは現在のバインドエリアにリストされます。「Edit a Context」ダイアログボックスを使用してデータソースオブジェクトと4th Dimensionオブジェクト間のバインドを定義するためには、次のステップに従います。

1. **ODBCTableのアップメニューの下のリストか列や式を選択する。**
   別のテーブルから列を表示するためにはODBC Tableポップアップメニューからテーブル名を選択します。
2. **4th Dimension Fileのポップアップメニューの下のリストから4th Dimension fieldを選択する。**
   別のファイルからfieldを表示するには、4th Dimension Fileポップアップメニューからファイル名を選択します。

   バインドしたいODBCオブジェクトと4th Dimensionオブジェクトを現在選択にしています。コンテキストにバインドを追加する前に、バインドの3つのオプションの間から選択できます。
3. **バインドボタンをクリックする。**
   コンテキストに現在バインドを定義しています。コンテキストをアクティブ化するまで、バインドはアクティブ化されません。

4. **コンテキストへ追加するバインドを定義するには1-3のステップを繰り返す。**
すべてのバインドはバインドを設定するオプションのリマインダと共に、「Current Binds」エリアにリストされます（更新、ソート、プライマリキー）。次の図ではCUSTOMERS.CUST_IDコラムが[CUSTOMERS]CUSTにリンクされています。



いつでも、「Current Binds」エリアから削除するものを選択し、deleteボタンをクリックすることによってbindを削除することができます。

コンテキストオプションの選択

コンテキストをアクティブ化する前にコンテキストのふるまいに作用するオプション、または節を指定しようとしてもかまいません。

コンテキストのオプションを指定するには：

1. **「Select Options」エリアからのオプションを選択する。**

■ **Distinctオプション**：このオプションは返されるべきdistinct値のみ指定します。

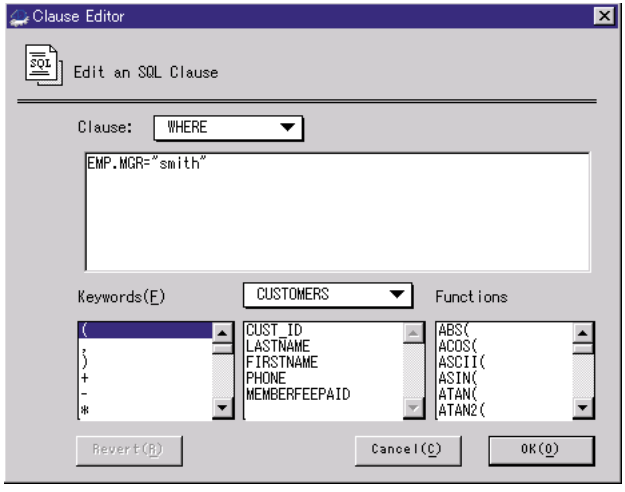
■ **For Updateオプション**：このオプションは他のすべてのユーザに対しロックすることによって更新を行うよう選択された列を準備します。

■ **No Waitオプション**：このオプションはアンロックになるようSELECTステートメントが選択した列を4D ODBCが持つべきではないと指定します。その列がロックされればクエリはキャンセルされます。No Waitオプションを選択できるようにするにはFor Updateオプションを選択しなければなりません。

コンテキストへの節の追加

コンテキストに節を追加するために：

1. **Select Clausesボタンをクリックする。**
「Edit a SQL Clause」ダイアログボックスが表示され、WHERE、GROVP BYまたはCONNECT BYといった節を作成できます。



節ポップアップメニューから作成する 節のタイプを選択できます。

節を構築するためには、テキストエリアにそれを入力します。節の作成を簡素化するにはキーワード、列名、及び機能をダイアログボックス下部のリストから選択します。

中央のリストの他のテーブルから列を表示するにはポップアップを利用し、タイトルバーの上下いずれかの矢印でテーブルのリストをスクロールし、表示したポップアップメニューからテーブル名を選択します。

コンテキストの作成ステートメントのコピー

典型的に、コンテキストは開発ツールとしてグラフィカルに定義されます。迅速なアプリケーションの開発促進のため、4D ODBCよるコンテキスト定義に相当する4th Dimensionステートメント作成を要求できます。使用する必要があるごとにコンテキストを定義するかわりに4D ODBC文を4th Dimensionメソッドに配置できます。

コンテキストをアクティブ化する前にクリップボードにこれらのステートメントをコピーするために：

1. **Copy Codeボタンをクリックする。**
ボタンをクリックすると、4D ODBCは、コンテキストIDのストアに必要な4th Dimension変数に名前を付けるように要求します。

2. 変数名を記入し、OKボタンをクリックする。
ステートメントはクリップボードにコピーされます。その場合、これらのステートメントは4th Dimensionメソッドにペースト可能です。

ファイルへのコンテキスト定義のセーブ

4D ODBCによって、ファイルにコンテキストをセーブできます。ファイルにセーブされたコンテキストは、後で「Edit a Context」ダイアログボックスにロードされます。ファイルにセーブされたコンテキストは**OC Open context file**関数を用いてメモリにロード可能です。

生成したコンテキストをファイルにセーブするために：

1. **「Save As」ボタンをクリックする。**

Save Fileダイアログボックスが表示され、ファイル名と場所を指定できます。

2. **ボタンをクリックする。**
ファイルはセーブされます。

ファイルからコンテキストをロードする

ファイルにあらかじめセーブされたコンテキストをロードするために：

1. **「Edit a Context」ダイアログボックスのOpenボタンをクリックする。**

「OpenFile」ダイアログボックスが表示されます。

2. **ファイルを選択し、Open ボタンをクリックする。**
コンテキストはロードされ、「Edit a Context」ダイアログボックスに表示されます。**OC Open context**関数を用いて、手続き的にコンテキストをロードすることも可能です。

手続き文を使用したコンテキストの作成

「Edit a Context」ダイアログボックスを使用したくない場合、4D ODBCコマンドと関数を使って手続き的にコンテキストを定義できます。

手続き的にコンテキストを作成する基本ステップは次のようなものです：

1. **OC Create context関数の実行により、コンテキストを初期化し、コンテキスト識別を検索する。**
コンテキスト識別は他のすべてのコンテキストコマンドのコンテキストを識別するのに使われます。

2. **OC ADD TO CONTEXT文シリーズ実行でコンテキストを定義する。**

OC ADD TO CONTEXTコマンドによって4th Dimensionオブジェクトとデータソースオブジェクトをバインドし、相当するUPDATE、SORT及びPRIMARYKEYオプションを追加することができます。

ここで4D ODBCが作るSELECT文は次のようなものです。**OC SET CLAUSE IN CONTEXT**文のシリーズの実行で節をコンテキスト定義に追加してください。

20課の従業員だけを確認したいのなら次の文を使用します：

OC SET CLAUSE IN CONTEXT (Context_ID;2;"DEPTNO=20")

このラインの追加は次のようにクエリを修正します：

SELECT empno, ename,… FROM emp WHERE DEPTNO=20

キーワード「WHERE」は 4D ODBCで自動的に追加されます。

OC Get clause in context文シリーズの実行でコンテキストの節セットを後で検索することができます。

コンテキストを通した結果の検索

OC ADD to contextコマンドを使用することで、データソースコラムと変数、配列、フィールドのような4D ODBCデータオブジェクトの2つのデータコンテナ間でバインドが作成されます。

注：OC Create context dialogコマンドにより、フィールドとバインドするコンテキストを作成する近道を提供しますが、コンテキストは4D ODBC**に一致したファイルが存在する必要はありません。**

変数へのコンテキスト

変数へのコンテキストはデータをストアせず、また局所的に構造を作成せずにデータを検索する最も単純な方法です。変数へのコンテキストは一列を検証、編集するレイアウトの作成に最適です。この方法はメモリ固有変数を使用し、ディスクアクセスを必要としません。したがって、変数へのコンテキストは優れた性能を発揮します。

コンテキストを必要とするすべての4D ODBC変数はあらかじめ宣言されねばなりません。次のコンパイラ指令を使用して、変数を宣言します：C_BOOLEAN、C_DATE、C_INTEGER、C_LONGINT、C_REAL、C_STRING、C_TEXT、またはC_TIME

コンテキストが変数に対するバインドを利用するなら、サーバから1回に1行以上を読み込めません。制限コマンドが1に設定されるなら、**OC Load rows context**は変数と共にのみ動作します。

▼ 次の例はテーブルタイトルと4D ODBC変数セット間のコンテキストを定義します。

```
C_TEXT (vTitle;vType;vPub_id)
C_REAL (vTitleId;vPrice;vYtd_sales)
If (login#0)
   myContext:=OC Create context ("pubs")
   If (myContext#0)
      OC ADD TO CONTEXT (myContext;"title_id";"vTitleId")
      OC ADD TO CONTEXT (myContext;"title";"vTitle")
      OC ADD TO CONTEXT (myContext;"type";"vType")
      OC ADD TO CONTEXT (myContext;"pub_id";"vPub_id")
      OC ADD TO CONTEXT (myContext;"price";"vPrice")
      OC ADD TO CONTEXT (myContext;"ytd_sales";"vYtd_sales")
      $res:=OC Load rows context (myContext)
   End if
End if
```

配列へのコンテキスト

コンテキストで使われたすべての4D ODBC配列はあらかじめ宣言されねばなりません。配列は、次の配列コマンドを用いて宣言されます：**ARRAY BOOLEAN**、**ARRAY DATE**、**ARRAY INTEGER**、**ARRAY LONGINT**、**ARRAY REAL**、**ARRAY TEXT**、**ARRAY STRING**

▼ 次の例では4D ODBC配列のarTitleにタイトルコラムをロードしてテーブルタイトルにコンテキストを定義しています。

```
ARRAY STRING (30;arTitle;0)
ARRAY STRING (30;arType;0)
ARRAY STRING (30;arPrice;0)

myContext:=OC Create context ("titles")
OC ADD TO CONTEXT (myContext;"title";"arTitle")
OC ADD TO CONTEXT (myContext;"type";"arType")
OC ADD TO CONTEXT (myContext;"price";"arPrice")
   ` データオブジェクトのバインド
$res:=OC Load rows context (myContext)
   ` ローの検索
```

フィールドへのコンテキスト

4D ODBCフィールドの使用で、次のリポートやラベルのプロダクションに中間データ記憶を提供します。

コンテキストからフィールドへデータを検索する場合、1に制限して**OC Load rows**を使用するなら手続き的にレコードを作成し、セーブしなければなりません。このプロセスが実行される時、4D ODBCの現在のレコードを認識します。

コンテキストからフィールドへデータ検索する場合、1より大きい値に制限して**OC Load rows context**を使用するなら、4D ODBCですべてのレコードは自動的に作製され、セーブされます。

1つのコンテキスト内のバインドは、ただ1つの4D ODBCファイルで確立します。このようにフィールド1[ファイル1]とコラムcol1、およびフィールド1[ファイル2]とコラムcol2をバインドできません。フィールドであるすべての4D ODBCオブジェクトは同じファイルに属さなければなりません。

▼ 次の例ではテーブルタイトルと4D ODBCファイル[title]間のコンテキストを作成しています。

```
myContext:=OC Define context ("titles")
OC ADD TO CONTEXT (myContext;"title";"[titles]title")
OC ADD TO CONTEXT (myContext;"type";"[titles]type")
OC ADD TO CONTEXT (myContext;"price";"[titles]price")
$res:=OC Load rows context (myContext)
```

設計選択

4D ODBCによって、ODBCデータソースのフロントエンドとして、4th Dimensionの使用が可能です。このように、ユーザにあつらえの、使いやすいインタフェースを提供する一方で、データソースのデータ記憶とデータ操作能力を利用できます。4th Dimensionによって、仮想的にあらゆるユーザインタフェースを作成できます。

この環境での4D ODBCの利用効率部分は、システムがクライアント／サーバアーキテクチャをベースとしていることです。4D ODBCの利用で、クライアント4th Dimensionアプリケーションにより、データ操作機能が実現すべきデータソースについてわかります。サーバはこれらの機能を実現し、あらゆる必要なデータを返します。クライアント／サーバをベースとしたアプリケーションでは、データはいつも、サーバにあります。

実行選択

4th Dimensionアプリケーションを作成する場合、データベース設計において、クライアント／サーバアーキテクチャを効率的に利用したいものです。データベース設計の選択する方法によって、データ転送速度に十分な衝撃を与えられます。

次の節では、4D ODBCデータベースをプログラムする異なる方法について説明します。

コンテキスト使用による実行

コンテキストを使用する場合、4th Dimensionデータベースのフィールド、変数、配列と、データソースのコラムとの間の関係を記述します。

例えば、データソーステーブルのFirstNameコラムと、4th Dimension変数のｖFirstNameを関連付けることができます。一旦コンテキストを作成したら、4th Dimensionフィールド、変数、配列の点で、データ操作を行うために、4D ODBCのコンテキストコマンドを使用できます。

コンテキストによって、データソースデータベースからデータを検索、表示、さらに4th Dimensionアプリケーションで修正し、データソースに返すシステムを、素早く容易に起動できます。4th Dimensionデータベース上に存在させ、データソースデータベースのフロントエンドとして走らせるのに、もっとも良く役立ちます。これは、SQL経験をほとんど持たない開発者には特に役立ちます。コンテキストはまた、プロトタイプの作成および迅速なアプリケーション開発に理想的です。アプリケーションを開発し、SQLを熟知するにつれて、下位レベルコマンドをデータベースにまとめる方向を選ぶでしょう。これにより、APIによる完全な機能提供と同様に、性能の利点を提供できます。

データベース実行でコンテキストの使用を選択する場合は、重要な性能問題について述べられている次の節を読んでください。

4th Dimensionフィールドを用いたバインド

一般に、4th Dimentionフィールドとデータソースコラム間でのバインドの作成は避けるべきです。4D Passportフィールドをともなうバインドの作成は、データソースデータベース構造を反映する4th Dimensionデータベース構造の保守を要求します。

しかし、バインドを使用する二重のデータベース構造の作成は必要ではありません。コンテキストは変数や配列の使用で確立可能で、それはクライアント／サーバ アーキテクチャには、より効率的な方法です。

フィールドとのバインドが妥当な理由のひとつは、バッチ処理のためです。バッチ処理の実行のために、データを検索し、局所的にストアするフィールドとのバインドを使用できます。それから、データソースとの通信を終え、クライアントマシン上のデータを動作させることができます。後で訂正データを返すために、データソースとの

再接続も可能です。

4th Dimension 変数や配列の使用によるバインド

バインドにおいて4th Dimension変数や配列を用いる場合、ホストデータベースを二重にする4th Dimensionデータベースの保守を回避できます。バインドは4th Dimensionフィールドに依存しないため、4th Dimensionデータベースは、単一ファイルとフィールドのみを含むことが可能です。データを転送し、操作するには、あらゆる組み合わせや形式に変数と配列を表示するレイアウトを作成します。

▼ 次の例では、変数を用いてコンテキストはどのように定義されるかを説明しています。各変数は最初に宣言され、ホストデータベースでコラムと関連付けられています。

```
C_INTEGER (vCust_num;vCust_rep)
C_STRING (50;vCust_name;vCust_city)
C_STRING (255;vCust_addr1;vCust_addr2)
C_STRING (2;vCust_state)
C_STRING (10;vCust_zip)
C_REAL (vCredit;vReceivable)

context :=OC Create context ("customers")
If (context#0)
   OC ADD TO CONTEXT (context; "cust_num"; "vCust_num")
   OC ADD TO CONTEXT (context; "cust_name"; "vCust_name")
   OC ADD TO CONTEXT (context; "cust_rep"; "vCust_rep")
   OC ADD TO CONTEXT (context; "addr1"; "vCust_addr1")
   OC ADD TO CONTEXT (context; "addr2"; "vCust_addr2")
   OC ADD TO CONTEXT (context; "cust_city"; "vCust_city")
   OC ADD TO CONTEXT (context; "cust_state"; "vCust_state")
   OC ADD TO CONTEXT (context; "cust_zip"; "vCust_zip")
   OC ADD TO CONTEXT (context; "credit_line"; "vCredit")
   OC ADD TO CONTEXT (context; "receivable"; "vReceivable")
End if
```

下位レベルコマンドを用いた実行

下位レベルコマンドを用いてデータベースを設計する場合、実行すべきデータ操作をデータソースに知らせるために、SQL文を使用できます。あらゆるタイプのデータ操作を実行するために、SQL文を作成できます。サーバ上の列を選択、挿入、更新、または削除できます。データの選択は、ある状態をベースとしていて、関連データをともなう状態を含んでいます。

一般に、下位レベルコマンドの使用は、コンテキストコマンド使用より効率的です。コンテキストコマンドで、4D ODBCはサーバの要求を実行する前に、コマンドをSQLクエリに訳さねばなりません。下位レベルコマンドで、要求はすでにSQL項に指定されます。4D ODBCは下位レベル文を、データソースが理解可能な文に翻訳するために

データソースのAPIを使用します。

変数や配列を用いたコンテキストのように、下位レベルコマンドは4th Dimensionフィールドに依存せず、データベース構造の維持を不必要にします。

あらゆる形式でデータを表示させる複数のレイアウトを作成する単一のファイル、またはフィールドだけを持つアプリケーションを得られます。

▼ 次の例では、ホストデータベースの一連の行を選択するために、下位レベルコマンドを用いる方法を説明し、最初の行のデータを検索しています。その方法では、はじめに送信し、選択文を実行しています。コラムごとで、最初の行にデータをロードしています。

```
` Declare variables used to display data on the layout
C_INTEGER (vCust_num;vCust_rep)
C_STRING (50;vCust_name;vCust_city)
C_STRING (255;vCust_addr1;vCust_addr2)
C_STRING (2;vCust_state)
C_STRING (10;vCust_zip)
C_REAL (vCredit;vReceivable)
C_TEXT (vSQL)

` construct our SQL query
vSQL := "select cust_num,cust_name,cust_rep,addr1,addr2,cust_city,"
vSQL := vSQL+"cust_state,cust_zip,credit_line,receivable from customers"

$result:=OC Set SQL in Cursor (cursor_id;vSQL)
$result:=OC Execute cursor (cursor_id)
$result:=OC Bind (cursor_id;1;"vCust_num")
$result:=OC Bind (cursor_id;2;"vCust_name")
$result:=OC Bind (cursor_id;3;"vCust_rep")
$result:=OC Bind (cursor_id;4;"vCust_addr1")
$result:=OC Bind (cursor_id;5;"vCust_addr2")
$result:=OC Bind (cursor_id;6;"vCust_city")
$result:=OC Bind (cursor_id;7;"vCust_state")
$result:=OC Bind (cursor_id;8;"vCust_zip")
$result:=OC Bind (cursor_id;9;"vCredit")
$result:=OC Bind (cursor_id;10;"vReceivable")
n:=OC Load row (cursor_id)
```

クライアント／サーバアーキテクチャ選択

4th Dimensionアプリケーションを作成する場合、データベース選択において、クライアント／サーバアーキテクチャを効率的に利用したいものです。データベース設計のために選択した方法によって、データ転送速度に十分な衝撃を与えられます。次の節では、4D ODBCデータベースを実行する異なる方法について説明します。

純粋なフロントエンドとしての4D ODBC

4th Dimensionがフロントエンドとして使われる場合、ODBCデータソースは、単一のデータ記憶として使用され、4Dはクライアントサーバアプリケーションのインタフェースビルダとしてのみ使用されます。

このアーキテクチャでは、4th Dimensionに局所的にストアされるアプリケーションデータはありません。この方法は実行のためだけに最適化され、データソース自身の制限にだけ強制されます。

4th Dimension手続き言語は、多くのデータ確認と操作機能に富んでいるため、4Dはフロントエンドアプリケーションビルダとして、よく適しています。

さらに、4D Compilerと4th Insiderは、強力な開発環境を特別に提供します。

4D ODBCでのバッチ処理

バッチ処理は、周期的に4th Dimensionまたは4D Serverデータエンジンにダウンロードされるデータソースへと中心的にデータストアする方法です。

データソースのリソース制約、または効率的リアルタイムアクセスを禁じたネットワークが存在する状況で、このは使用可能です。この方法は、データの4th Dimensionへのダウンロードによる特徴の報告、検索、および4D本来の特徴を用いる時、4th Dimension構築の利用にも使用されます。

さらに、このアーキテクチャはポータブルコンピュータ上に、4Dデータソースを載せるためにメカニズムを供給します。それで、ユーザーがオフィスから外出している時、データアクセスが可能です。

4D ODBCの分散処理

分散アーキテクチャによって、ODBCデータソースと4th Dimension（または4D Server）両方の間で、データを共有することにより、両世界の最良のものを結び付けることができます。これらの2つの環境の間にデータ構造を分散することにより、多数のソースからデータを結合することができます。

適切な設計で、このデータソース混合は完全にユーザに明白でありえます。

分散アーキテクチャの例では、企業データを、OracleあるいはSybaseのような大量の

データソースへストアする大きい組織であることが考えられます。このデータは妥当で、すべての組織にアクセス可能です。部門ワークグループからのデータは、4D Serverデータベースにストアされることが考えられます。4D Serverと4D ODBCの力を使うことによって、これらのデータソース両方からデータを表示するレイアウトの設計が可能です。2つのシステム間に「仮想的な関係」を開発することも可能です。

カタログコマンド、はじめに

カタログコマンドは、例えばデータソース一覧にストアされたテーブルのリストや指定されたテーブルの列の名前の一覧、テーブルに関連付けられた索引などから情報を検索することができます。

カタログコマンドを使って次のようなことが可能となります：

- インストールされたドライバのリストの検索（**OC GET DSN LIST**）
- データソースのテーブル、列、索引等のスキーマに関する情報を得る（**OC GET TABLE LIST**、**OC GET COLUMN LIST**、**OC GET SPECIAL COLUMN LIST**、**OC GET PRIMARY KEY LIST**、**OC GET FOREIGN KEY LIST**）
- サーバプロシージャのリストとその列のリストの利用（**OC GET PROCEDURE LIST**、**OC GET PROCEDURE COLUMN LIST**）
- テーブルと列の特権的な情報を得る（**OC GET TABLE PRIVILEGE LIST**、**OC GET COLUMN PRIVILEGE LIST**、**OC GET TABLE STAT**）
- 接続に関する情報を検索する

OC GET COLUMN LIST

OC GET COLUMN LIST (カーソルID;テーブル;列配列;タイプ配列;列長配列;ヌル配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
テーブル	文字列	→	テーブル名
列配列	文字列配列	←	列名のリスト
タイプ配列	文字列配列	←	列のタイプのリスト
列長配列	文字列配列	←	列長のリスト
ヌル配列	文字列配列	←	列のヌルフィールドの説明

説明

OC GET COLUMN LISTコマンドはODBCデータソース上のテーブルの中から列のリストを得ることができます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜テーブル＞は列情報を得る対象となるテーブルの名前です。

＜列配列＞は列名のリストの値を格納する文字列配列の名前です。

＜タイプ配列＞は列タイプのリストの値を格納する文字列配列名です。

＜列長配列＞は列長のリストの値を格納する文字列配列名です。

＜ヌル配列＞は列のヌルフィールドの説明を格納する文字列配列名です

例題

次の式は列のリストをロードして配列に説明を付加します。

```
ARRAY STRING (30;arCol;0)
ARRAY STRING (30;arType;0)
ARRAY STRING (30;arLen;0)
ARRAY STRING (30;arNull;0)
If (arTab#0)
   OC GET COLUMN LIST (cur_id;arTab{arTab};"arCol";"arType";"arLen";"arNull")
Else
   ALERT (select a table in the table name array !")
End if
```

参照

なし

OC GET COLUMN PRIVILEGE LIST

OC GET COLUMN PRIVILEGE LIST (カーソルID;テーブル;所有者配列;権限受け渡し配列;権限受取り配列;権限配列;テーブルタイプ配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
テーブル	文字列	→	テーブル名
所有者配列	文字列配列	←	所有者名のリスト
権限受け渡し配列	文字列配列	←	権限受け渡しリスト
権限受取り配列	文字列配列	←	権限受け取り名
権限配列	文字列配列	←	権限のリスト
テーブルタイプ配列	文字列配列	←	テーブルタイプのリスト

説明

OC GET COLUMN PRIVILEGE LISTコマンドはテーブルのパーミッションを伝達します。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜テーブル＞はその列情報を得るテーブルの名前です。

＜所有者配列＞は列の所有者の名前のリストの値を格納する文字列配列名です。

＜権限受け渡し配列＞は権限配列で指定された権限を受け渡すリストを格納する文字列配列名です。

＜権限受取り配列＞は権限の受け取り名が権限配列で指定したリストの値を格納する文字列配列名です。

＜権限配列＞は権限のリストの値を格納する文字列配列名です。

＜テーブルタイプ配列＞はテーブルタイプのリストの値を格納する配列名です。

参照

なし

OC GET DSN LIST

OC GET DSN LIST (カーソルID;dsn配列;desc配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
dsn配列	文字列	→	4D配列名
desc配列	文字列	→	4D配列名

説明

OC GET DSN LISTコマンドは、どのODBCデータソースがクライアントマシン上で利用可能であるか問い合わせます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

注：＜カーソルID＞は接続が必要ないなら必須ではありません。その場合、パラメータを0とすることができます。しかし、既存のアプリケーションとの互換性のため、＜カーソルID＞を保持します。

＜dsn配列＞は利用可能なデータソースのリストの値を格納する文字列配列名です。

＜desc配列＞は利用可能なデータソースの説明のリストを取る文字列配列名です。

例題

次の文は利用可能なデータソースとその説明のリストをロードし、配列に格納します。

```
ARRAY STRING (30;arDSN;0)
ARRAY STRING (30;arDescrip;0)
OC GET DSN LIST (cursor_id;"arDSN";"arDescrip")
```

参照

なし

OC GET FOREIGN KEY LIST

OC GET FOREIGN KEY LIST (カーソルID;pkテーブルターゲット;fkテーブルターゲット;pkテーブルクオリファイア;pkテーブルオーナ;pkテーブル名;pk列名;fkテーブルクオリファイア;fkテーブルオーナ;fkテーブル名;fk列名;キーシーケンス;更新規則;削除規則;fk名;pk名)

引数	タイプ		説明
カーソルID	倍長整数	→	接続確認
pkテーブルターゲット	文字列	→	プライマリキーテーブルターゲット名
fkテーブルターゲット	文字列	→	外部キーテーブルターゲット名
pkテーブルクオリファイア	文字列	←	4D配列名
pkテーブル所有者	文字列	←	4D配列名
pkテーブル名	文字列	←	4D配列名
pk列名	文字列	←	4D配列名
fkテーブルクオリファイア	文字列	←	4D配列名
fkテーブル所有者	文字列	←	4D配列名
fkテーブル名	文字列	←	4D配列名
fk列名	文字列	←	4D配列名
キーシーケンス	文字列	←	4D配列名
更新規則	文字列	←	4D配列名
削除規則	文字列	←	4D配列名
fk名	文字列	←	4D配列名
pk名	文字列	←	4D配列名

説明

OC GET FOREIGN KEY LISTコマンドは他のテーブルのプライマリキーに関係する指定されたテーブルでの外部のキーのリストあるいは指定されたテーブルでプライマリキーに関係する他のテーブルでの外部のキーのリストを返します。

- ＜pkテーブルターゲット＞がテーブル名を含んでいる場合、**OC GET FOREIGN KEY LIST**は指定されたテーブルのプライマリキーとそれに関連するすべての外部キーを返します。
- ＜fkテーブルターゲット＞がテーブル名を含んでいる場合、**OC GET FOREIGN KEY LIST**は指定されたテーブルのすべての外部キーとそれに関連するプライマリキーを返します。
- ＜pkテーブルターゲット＞と＜fkテーブルターゲット＞の両方がテーブル名を含んでいる場合、**OC GET FOREIGN KEY LIST**は＜pkテーブルターゲット＞で指定されたプライマリキーに関連する、＜fkテーブルターゲット＞で指定されたテーブルの外部キーを返します。

＜カーソルID＞は有効なカーソルでなければなりません。

＜pkテーブルターゲット＞はプライマリキーを含んだテーブルです。

＜fkテーブルターゲット＞は外部キーを含んだテーブルです。

＜pkテーブルクオリファイア＞はプライマリキーテーブルのクオリファイアを確認します。データソースが適切でない場合は空文字列を返します。

＜pkテーブル所有者＞はプライマリキーテーブルの所有者を確認します。データソースが適切でない場合は空文字列を返します。

＜pkテーブル名＞はプライマリキーテーブルを確認します。

＜pk列名＞はプライマリキー列の確認をします。

＜fkテーブルクオリファイア＞はプライマリキーテーブルのクオリファイアを確認します。データソースが適切でない場合は空文字列を返します。

＜fkテーブル所有者＞はプライマリキーテーブルの所有者を確認します。データソースが適切でない場合は空文字列を返します。

＜fkテーブル名＞は外部キーテーブルを確認します。

＜fk列名＞は外部キー列の確認をします。

＜キーシーケンス＞は（1から始まる）キーの列シーケンス番号です。

更新規則はSQL処理がUPDATEの場合、外部キーに適用されるアクションです：
0 -> SQL_CASCADE
1 -> SQL_RESTRICT
2 -> SQL_SET_NULL
データソースが適切でない場合は空文字列を返します。

削除規則はSQL処理がDELETEの場合、外部キーに適用されるアクションです：
0 -> SQL_CASCADE
1 -> SQL_RESTRICT
2 -> SQL_SET_NULL
データソースが適切でない場合は空文字列を返します。

＜fk名＞は外部キーを確認します。データソースが適切でない場合は空文字列を返します。

＜pk名プライマリキー＞を確認します。データソースが適切でない場合は空文字列を返します。

例題

```
    ` 配列を宣言します
ARRAY STRING (15;apkTableQualifier;0)
ARRAY STRING (15;apkTableOwner;0)
ARRAY STRING (15;apkTableName;0)
ARRAY STRING (15;apkColumnName;0)
ARRAY STRING (15;afkTableQualifier;0)
ARRAY STRING (15;afkTableOwner;0)
ARRAY STRING (15;afkTableName;0)
ARRAY STRING (15;afkColumnName;0)
ARRAY STRING (15;aKeySeq;0)
ARRAY STRING (15;aUpdateRule;0)
ARRAY STRING (15;aDeleteRule;0)
ARRAY STRING (15;afkName;0)
ARRAY STRING (15;apkName;0)
loginID:=OC Login ("sa";"";"MSSQL65")
cursor:=OC Create cursor (loginID)
   OC GET FOREIGN KEY LIST (cursor;"titres";"titreauteur";"apkTableQualifier";
      "apkTableOwner";"apkTableName";"apkColumnName";"afkTableQualifier";
            "afkTableOwner";"afkTableName";"afkColumnName";"aKeySeq";
                  "aUpdateRule";"aDeleteRule";"afkName";"apkName")
OC DROP CURSOR (cursor)
OC LOGOUT (loginID)
```

参照

OC GET PRIMARY KEY LIST

OC GET PRIMARY KEY LIST

OC GET PRIMARY KEY LIST (カーソルID;テーブル;テーブルカタログ配列;テーブルスキーマ配列;テーブル名配列;列配列;シーケンス配列;キー名配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
テーブル	文字列	→	テーブル名
テーブルカタログ配列	文字列配列	←	カタログリスト
テーブルスキーマ配列	文字列配列	←	スキーマリスト
テーブル名配列	文字列配列	←	テーブルリスト
列配列	文字列配列	←	索引のベースとなる列のリスト
シーケンス配列	文字列配列	←	索引の連続オーダー
キー名配列	文字列配列	←	索引名

説明

OC GET PRIMARY KEY LISTコマンドはODBCデータソースのテーブルからプライマリキーのリストを得ることができます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜テーブル＞は列情報を得る対象となるテーブル名です。空文字列を渡すことによって、すべての関連のないテーブルをチェックすることができます。

＜テーブルカタログ配列＞はテーブルがベースとなるカタログの名前です。

＜テーブルスキーマ配列＞はテーブルがベースとなるスキーマの名前です。

テーブルのパラメータに空文字列を渡した場合、＜テーブル名配列＞は関連のないテーブルの名前を含んでいます。

＜列配列＞は索引のベースとなる列のリストの値を格納する文字列配列の名前です。

＜シーケンス配列＞は索引の連続オーダーです。

＜キー名配列索引＞の名前です。

例題
次の式はプライマリキーのリストをロードし、その記述を配列に格納しています。

```
ARRAY STRING (30;arCat0)
ARRAY STRING (30;arSch;0)
ARRAY STRING (30;arNam;0)
ARRAY STRING (30;arCol;0)
ARRAY STRING (30;arSeq;0)
ARRAY STRING (30;arKey;0)
   OC GET PRIMARY KEY LIST (cur_id;table;"arCat";"arSch";"arNam";"arCol";
                                                                "arSeq";"arKey")
```

参照
なし

OC GET PROCEDURE COLUMN LIST

OC GET PROCEDURE COLUMN LIST (カーソルID;プロシージャ;列配列;タイプ配列;精密な配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
プロシージャ	文字列	→	プロシージャ名
列配列	文字列配列	←	列名のリスト
タイプ配列	文字列配列	←	列データタイプのリスト
精密な配列	文字列配列	←	精度値

説明
OC GET PROCEDURE COLUMN LISTコマンドは手続きプロシージャによって使用された列のリストを得ることができます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜プロシージャ＞は列情報を得る対象となるプロシージャ名です。

＜列配列＞はプロシージャによって使用された列名のリストを格納する文字列配列名です。

＜タイプ配列＞は列データタイプのリストの値を格納する文字列配列名です。

＜精密な配列＞は列の精度値をを格納する文字列配列名です。

参照
なし

OC GET PROCEDURE LIST

OC GET PROCEDURE LIST (カーソルID;DB配列;所有者配列;プロシージャ配列;タイプ配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルＩＤ
DB配列	文字列配列	←	データベース名のリスト
所有者配列	文字列配列	←	所有者名のリスト
プロシージャ配列	文字列配列	←	プロシージャ名
タイプ配列	文字列配列	←	プロシージャタイプのリスト

説明

OC GET PROCEDURE LISTコマンドはODBCデータソース上のストアードプロシージャのリストを得ることができます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜DB配列＞はプロシージャが存在するデータベース名のリストを格納する文字列配列名です。

＜所有者配列＞は所有者名のリストを格納する文字列配列名です。

＜プロシージャ配列＞はプロシージャ名を格納する文字列配列名です。

＜タイプ配列＞プロシージャタイプのリストの値を格納する文字列配列名です。

例題

次の式はストアードプロシージャのリストをロードし、配列に格納します。

```
ARRAY STRING (30;arDB;0)
ARRAY STRING (30;arOwn;0)
ARRAY STRING (30;arProc;0)
ARRAY STRING (30;arType;0)
If (arTab#0)
   OC GET PROCEDURE LIST (cur_id;"arDB";"arOwn";"arProc";"arType")
Else
   ALERT ("select a table in the table name array !")
End if
```

参照

なし

OC GET SPECIAL COLUMN LIST

OC GET SPECIAL COLUMN LIST (カーソルID;モード;テーブルターゲット;スコープ;列名;データタイプ;タイプ名;精度;長さ;スケール;擬似列)

引数	タイプ		説明
カーソルID	倍長整数	→	接続確認
モード	整数	→	1または2
テーブルターゲット	文字列	→	ターゲットテーブルの名前
スコープ	文字列	←	4D配列名
列名	文字列	←	4D配列名
データタイプ	文字列	←	4D配列名
タイプ名	文字列	←	4D配列名
精度	文字列	←	4D配列名
長さ	文字列	←	4D配列名
スケール	文字列	←	4D配列名
擬似列	文字列	←	4D配列名

説明

OC GET SPECIAL COLUMN LISTコマンドは、特定のテーブルで一意的に横列を識別する列の最適なセットを得ることができます。横列でどんな値でもトランザクションによって更新される時、それは同じく自動的に更新される列を返します。

＜カーソルID＞は有効なカーソルでなければなりません。

＜モード＞はモードのセットを許可します：
1->SQL_BEST_ROWID、一意的に定義される横列を返します。
2->SQL_ROWER、横列のどんな値でもトランザクションによって更新される時、同じく自動的に列を更新します。

＜テーブルターゲット＞はデータソースのターゲットテーブル名です。

＜スコープ＞はROWIDの実際の範囲です。次の値を1つ含んでいます：
0->SQL_SCOPE_CURROW
ROWIDはその横列に置かれている間だけ有効であることを保証します。もし横列がもう1つのトランザクションによってアップデートされたか、あるいは削除されたなら、ROWIDを使っている後のreselectが横列を返さないかもしれません。
1->SQL_SCOPE_TRANSACTION
ROWIDは現在のトランザクションの期間中、有効であることが保証されます。
2->SQL_SCOPE_SESSION
ROWIDは（トランザクション処理限界前の）セッションの期間中、有効であることを保証します。

＜列名＞はデータソースのターゲットテーブル名です。

＜データタイプ＞はSQLデータタイプです。これはODBC SQLデータタイプまたはドライバ特定のSQLデータタイプであり得ます。

＜タイプ名＞はデータソースに依存するタイプの名前です。

＜精度＞はデータソースの列の精度です。

＜長さ＞はSQL GETDATAあるいはSQL FETCH処理の転送されたデータのバイトの長さです。数値データのために、サイズがデータソースにストアされたデータのサイズと異なっているかもしれません。

＜スケール＞はデータソースの列のスケールです。空文字列がスケールが適用可能ではないデータのために返されます。

＜ダミー列＞は列がオラクルROWIDとして疑似列であるかどうか示します。

例題

```
ARRAY STRING (15;aScope;0)
ARRAY STRING (15;aColumnName;0)
ARRAY STRING (15;aDataType;0)
ARRAY STRING (15;aTypeName;0)
ARRAY STRING (15;aPrecision;0)
ARRAY STRING (15;aLength;0)
ARRAY STRING (15;aScale;0)
ARRAY STRING (15;aPseudoColumn;0)

loginID:=OC Login ("sa";"";"MSSQL65")
cursor:=OC Create cursor (loginID)
   OC GET SPECIAL COLUMN LIST (cursor;1;"employee";"aScope";"aColumnName";
         "aDataType";"aTypeName";"aPrecision";"aLength";"aScale";"aPseudoColumn")
OC DROP CURSOR (cursor)
OC LOGOUT (loginID)
```

参照

OC GET COLUMN LIST

OC GET TABLE LIST

OC GET TABLE LIST (カーソルID;DB配列;所有者配列;テーブル配列;タイプ配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
DB配列	文字列配列	←	データベース名のリスト
所有者配列	文字列配列	←	所有者名のリスト
テーブル配列	文字列配列	←	テーブル名のリスト
タイプ配列	文字列配列	←	テーブルタイプのリスト

説明

OC GET TABLE LISTコマンドはODBCデータソース上の利用可能なテーブルのリストを得ることができます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜DB配列＞はデータベース名のリストの値を格納する文字列配列名です。

＜所有者配列＞は所有者名のリストの値を格納する文字列配列名です。

＜テーブル配列＞はテーブル名のリストの値を格納する文字列配列名です。

＜タイプ配列＞はテーブルタイプのリストの値を格納する文字列配列名です。

例題

次の式は利用可能なテーブルのリストをロードし、それらの説明を配列に格納します。

```
ARRAY STRING (30;arDB;0)
ARRAY STRING (30;arOwn;0)
ARRAY STRING (30;arTab;0)
ARRAY STRING (30;arType;0)
  OC GET TABLE LIST (cursor_id;"arDB";"arOwn";"arTab";"arType")
```

参照

なし

OC GET TABLE PRIVILEGE LIST

OC GET TABLE PRIVILEGE LIST (カーソルID;テーブル;所有者配列;権限受け渡し配列;権限受け取り配列;権限配列;許可配列)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
テーブル	文字列	→	テーブル名
所有者配列	文字列	←	所有者名のリスト
権限受け渡し配列	文字列	←	権限受け渡しリスト
権限受け取り配列	文字列	←	権限受け取り名
権限配列	文字列	←	権限のリスト
許可配列	文字列	←	認められる属性のリスト

説明

OC GET TABLE PRIVILEGE LISTコマンドはテーブルのパーミッションを伝達します。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜テーブル＞は列情報を得る対象となるテーブルの名前です。

＜所有者配列＞は列の所有者名のリストを格納する文字列配列名です。

＜権限受け渡し配列＞は権限配列で指定された権限を受け渡すリストを格納する文字列配列名です。

＜権限受取り配列＞は権限の受け取り名が権限配列で指定したリストの値を格納する文字列配列名です。

＜権限配列＞は権限のリストの値を格納する文字列配列名です。

＜許可配列＞は認められている属性のリストの値を格納する文字列配列名です。

例題

次の式はプライマリキーのリストをロードしてその説明を配列に格納します。

```
ARRAY STRING (30;arOwn;0)
ARRAY STRING (30;arGrantor;0)
ARRAY STRING (30;arGrantee;0)
ARRAY STRING (30;arPriv;0)
ARRAY STRING (30;arGrant;0)
   OC GET TABLE PRIVILEGE LIST (cur_id;arTab{arTab};"arOwn";"arGrantor";
                                                    "arGrantee";"arPriv";"arGrant")
```

参照

なし

OC GET TABLE STAT

OC GET TABLE STAT (カーソルID;テーブルターゲット;テーブルクオリファイア;テーブル所有者;テーブル名;非一意性;索引クオリファイア;索引名;タイプ;索引シーケンス;列名;対照;重要性;ページ;フィルタ状態)

引数	タイプ		説明
カーソルID	倍長整数	→	接続確認
テーブルターゲット	文字列	→	ターゲットテーブル名
テーブルクオリファイア	文字列	←	4D配列名
テーブル所有者	文字列	←	4D配列名
テーブル名	文字列	←	4D配列名
非一意性	文字列	←	4D配列名
索引クオリファイア	文字列	←	4D配列名
索引名	文字列	←	4D配列名
タイプ	文字列	←	4D配列名
索引シーケンス	文字列	←	4D配列名
列名	文字列	←	4D配列名
対照	文字列	←	4D配列名
重要性	文字列	←	4D配列名
ページ	文字列	←	4D配列名
フィルタ状態	文字列	←	4D配列名

説明

OC GET TABLE STATコマンドは1つのテーブルについての統計値のリストとテーブルと関連付けられた索引を返します。

＜カーソルID＞は有効なカーソルでなければなりません。

＜テーブルターゲット＞はデータソースのターゲットテーブルの名前です。

＜テーブルクオリファイア＞は統計値あるいは索引が適用されるテーブルのテーブルクオリファイアです。データソースが適切でないなら空文字列を受け取ることができます。

＜テーブル所有者＞は統計値あるいは索引が適用されるテーブルのテーブル所有認証者です。データソースが適切でないなら空文字列を受け取ることができます。

＜テーブル名＞は統計値あるいは索引が適用されるテーブルのテーブル認証者です。

＜非一意性＞は索引が複写された値を禁止するかどうか示します。

＜索引クオリファイア＞は**DROP INDEX**を利用して索引名を制限するために使われる確認証明です。索引がデータソースによってサポートされないなら、あるいはもしタイプが SQL_TABLE_STATであるなら、空文字列が返されます。

＜索引名＞は索引名を制限するために使われる認証者です。

＜タイプ＞は返される情報のタイプです：
0-> SQL_TABLE_STAT テーブルのための統計値を示す
1-> SQL_INDEX_CLUSTURED 索引群を示す
2-> SQL_INDEX_HASHED 再構成された索引を示す
3-> SQL_INDEX_OTHER 他のタイプの索引を示す

＜索引シーケンス＞は1から始まる索引の列シーケンス数です。タイプがSQL_TABLE_STATであるなら、空文字列が返されます。

＜列名＞は列の確認標識です。タイプがSQL_TABLE_STATであるなら、空文字列が返されます。

＜対照＞は列ソートの連続：昇順のためのA、降順のためのD。

＜タイプ＞がSQL_TABLE_STATであるなら、あるいはもし列ソートの連続がサポートされていないなら、空文字列が返されます。

＜タイプ＞がSQL_TABLE_STATであるなら、重要性はテーブルの中の横列の数です、さもなければそれは索引でユニークな値の数です。データソースが適切でないなら空文字列が返されます。

＜ページ＞は（＜タイプ＞がSQL_TABLE_STATなら）テーブル、または（＜タイプ＞がSQL_TABLE_STATでないのなら）索引のストアに使われたページ数です。データソースから値が利用できない、あるいはデータソースが利用できないならば空文字列を返します。

＜フィルタ状態＞：索引がフィルタされた索引ならば、これは給料>300のようなフィルタ状態です。フィルタ状態が決定されなかったら空文字列を返します。

例題

```
   ` 配列を宣言します
ARRAY STRING (15;aTableQualifier;0)
ARRAY STRING (15;aTableOwner;0)
ARRAY STRING (15;aTableName;0)
ARRAY STRING (15;aNonUnique;0)
ARRAY STRING (15;aIndexQualifier;0)
ARRAY STRING (15;aIndexName;0)
ARRAY STRING (15;aType;0)
ARRAY STRING (15;aSeqInIndex;0)
ARRAY STRING (15;aColName;0)
ARRAY STRING (15;aCollation;0)
ARRAY STRING (15;aCardinality;0)
ARRAY STRING (15;aPages;0)
ARRAY STRING (15;aFilterCondition;0)

loginID:=OC Login ("sa";"";"MSSQL65")
cursor:=OC Create cursor (loginID)
   OC GET TABLE STAT (cursor;"employee";"aTableQualifier";"aTableOwner";
         "aTableName";"aNonUnique";"aIndexQualifier";"aIndexName";
         "aType";"aSeqInIndex";"aColName";"aCollation";
         "aCardinality";"aPages";"aFilterCondition")
OC DROP CURSOR (cursor)
OC LOGOUT (loginID)
```

参照

なし

OC GET TYPE INFO

OC GET TYPE INFO (カーソルID;データタイプ;配列名)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
データタイプ	整数	→	データタイプ属性ID
配列名	文字列配列	←	すべてのデータタイプの属性

説明

OC GET TYPE INFOコマンドはすべての利用可能なデータタイプからアプリケーションのためにデータタイプ属性を得ることができます。

＜カーソルID＞は有効なカーソルでなければなりません。

＜データタイプ＞はリクエストされたデータタイプ属性のための参照番号です：

データタイプ属性	ID
名前	1
データタイプ	2
精度	3
リテラル接頭辞	4
リテラル接尾辞	5
作成されたパラメータ	6
ヌル	7
微妙なケース	8
探索可能	9
符号なし	10
金額のオートインクリメント	11
局所的なタイプ名	12
最小スケール	13
最大スケール	14

＜配列名＞はすべてのデータタイプの属性を格納する配列名です。

例題

データソース中にヌル、長さ、名前のデータタイプ属性があるか検索します。

```
ARRAY STRING (30;arName;0)
ARRAY STRING (30;arLength;0)
ARRAY STRING (30;arNull;0)
   $result:=OC GET TYPE INFO (cursor_id;1;"arName")
   $result:=OC GET TYPE INFO (cursor_id;3;"arLength")
   $result:=OC GET TYPE INFO (cursor_id;7;"arNull")
```

参照
OC GET TYPE INFO LIST

OC GET TYPE INFO LIST

OC GET TYPE INFO LIST (ログインID;配列名1;配列名2;配列名3;配列名4;配列名5;配列名6;配列名7;配列名8;配列名9;配列名10;配列名11;配列名12;配列名13;配列名14;配列名15)

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
配列名1	文字列配列	←	名前
配列名2	文字列配列	←	データタイプID
配列名3	文字列配列	←	精度
配列名4	文字列配列	←	リテラル接頭辞
配列名5	文字列配列	←	リテラル接頭辞
配列名6	文字列配列	←	作成されたパラメータ
配列名7	文字列配列	←	ヌル
配列名8	文字列配列	←	微妙なケース
配列名9	文字列配列	←	探索可能
配列名10	文字列配列	←	符号なし
配列名11	文字列配列	←	金額
配列名12	文字列配列	←	オートインクリメント
配列名13	文字列配列	←	局所的なタイプ名
配列名14	文字列配列	←	最小スケール
配列名15	文字列配列	←	最大スケール

説明
OC GET TYPE INFO LISTコマンドはアプリケーションですべての利用可能なデータタイプに対してすべてのデータ属性を得ることができます。

＜ログインID＞は有効なログインIDでなければなりません。

参照
OC GET TYPE INFO

コンフィグレーションコマンド、はじめに

この章のコマンド群はドライバ、ログイン、カーソル、データタイプに関連する様々な情報を検索し、セットすることができます。

コンフィグレーションコマンドを使って次のようなことが可能となります：

- ODBCドライバについての情報を検索し、ドライバがどのAPI機能をサポートしているかを決定します（**OC Get function**、**OC Get Info**、**OC GET DRIVER CAPABILITIES**、**OC GET DRIVER DESC**）。
- ODBCドライバによって翻訳されるSQLステートメント（**OC CONVERT TO NATIVE**）
- 接続オプションの検索と設定（**OC Get login option**、**OC Set login option**）
- カーソルオプションの接続と設定（**OC Get cursor option**、**OC Set cursor option**）

OC CONVERT TO NATIVE

OC CONVERT TO NATIVE (ログインID;コマンド)

引数	タイプ		説明
ログインID	倍長整数	→	接続認証者
コマンド	文字列｜テキスト	→	コマンドの実行

説明

OC CONVERT TO NATIVEコマンドはODBCドライバによって翻訳されたSQL文字列を返します。

＜ログインID＞は有効な接続の認証者でなければなりません。

＜コマンド＞は翻訳するSQLコマンドです。

これらの例の結果は問い合わされたデータベースに依存します（データソースのPiecesIDは整数型と仮定します)。

■ 4DからSQLコマンド：

SELECT {fn CONVERT (PiecesId , SQL_SMALLINT)} FROM Pieces

■ MS SQL Serverより：

SELECT convert (smallint , PiecesId) FROM Pieces

■ ORACLEより：

SELECT to_number (PiecesId) FROM Pieces

例題

この4Dコードの例はこの結果を得ることができるようにします。

```
C_TEXT (sql)
loginID:=OC Login ("sa";"";"MSSQL65")
sql:="SELECT {fn CONVERT (PiecesId, SQL_SMALLINT)} FROM Pieces"
   OC CONVERT TO NATIVE (loginID;sql)
OC LOGOUT (loginID)
```

参照

なし

OC Get sursor option

OC Get sursor option (カーソルID;オプションID)→文字列

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
オプションID	整数	→	関数の参照番号
戻り値	文字列	←	コンフィグレーション情報、 エラーならば空文字列

説明

OC Get login option関数はカーソルに関するコンフィグレーション情報を検索します。

＜カーソルID＞は前もって作成されたカーソルのIDでなければなりません。

＜オプションID＞はリクエストされたオプションの参照番号です。参照番号のリストはこのルーチンの説明の後に続きます。

OC Get cousor optionが成功するとリクエストした情報が返されます。失敗すると空文字列が返されます。

オプションIDの参照番号は次の通りです：

関数	ID
SQL_QUERY_TIMEOUT	0
SQL_MAX_ROWS	1
SQL_NOSCAN	2
SQL_MAX_LENGTH	3
SQL_ASYNC_ENABLE	4
SQL_BIND_TYPE	5
SQL_CURSOR_TYPE	6
SQL_CONCURRENCY	7
SQL_KEYSET_SIZE	8
SQL_ROWSET_SIZE	9
SQL_SIMULATE_CURSOR	10
SQL_RETRIEVE_DATA	11
SQL_USE_BOOKMARKS	12
SQL_GET_BOOKMARKS	13
SQL_ROW_NUMBER	14

例題

次の方法はSQL_AUTOCOMMIT option（ID=102）に関する情報を得て変数$resultに格納します。

```
$result:=OC Get cursor option (login;102)
```

参照

OC Set cursor option

OC GET DRIVER CAPABILITIES

OC GET DRIVER CAPABILITIES (ログインID;コマンドID)

引数	タイプ		説明
カログインID	倍長整数	→	ログインID
コマンドID	整数	→	4D ODBC 呼び出しのID

説明

OC GET DRIVER CAPABILITIESコマンドはODBC呼び出しがサポートしているドライバに問い合わせることでいずれの 4D ODBC コマンドが利用可能かを知ることができます。結果はデバッガウィンドウに表示されます。

別の方法を使ってコマンドIDを引数として使うことで、特定の4D ODBCコマンドを求めることができます。

4D ODBC**呼び出し**	ID
OC Login dialog	1
OC Login	2
OC LOGOUT	3
OC Query exec	4
OC Execute SQL	5
OC Clone 4D table	6
OC Create cursor	7
OC DROP CURSOR	8
OC SET CURSOR NAME	9
OC Get cursor name	10
OC Set SQL in Cursor	11
OC Execute cursor	12
OC Execute direct cursor	13
OC Bind	14
OC Load row	15
OC More results	16
OC Number of columns	17
OC Describe column	18
OC Column attributes	19
OC Number rows processed	20
OC Create context dialog	21
OC Create context	22
OC DROP CONTEXT	23
OC ADD TO CONTEXT	24
OC Load context file	25
OC Load context picture	26
OC SAVE CONTEXT FILE	27
OC Save context picture	28
OC EDIT CLAUSES IN CONTEXT	29

参照

なし

OC GET DRIVER DESC

OC GET DRIVER DESC (descタブ名;attタブ名;attValタブ名)

引数	タイプ		説明
descタブ名	文字列	←	ドライバの説明を格納する配列
attタブ名	文字列	←	属性名を格納する配列
attValタブ名	文字列	←	属性の値を格納する配列

説明

OC GET DRIVER DESCコマンドはインストールされた各ドライバについてドライバの説明、属性名、各属性の値を得るため、ドライバマネージャに問い合わせを行ないます。

＜descタブ名＞はドライバの説明を格納する配列名です。

＜attタブ名＞は属性名を格納する配列名です。

＜attValタブ名＞は各属性の値を格納する配列名です。

注：このコマンドはログインが有効である必要はありません。

例題

```
ARRAY STRING (15;aDescTabName;0)
ARRAY STRING (15;aAttTabName;0)
ARRAY STRING (15;aAttValTabName;0)
   OC GET DRIVER DESC ("aDescTabName";"aAttTabName";"aAttValTabName")
```

参照

なし

OC Get function

OC Get function (ログインID;関数ID;4Dオブジェクト)→整数

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
関数ID	整数	→	関数の参照番号
4Dオブジェクト	文字列	←	サポートされている関数名
戻り値	整数	←	1=関数は実装されています。 -1=関数は実装されていません。

説明

OC Get functionルーチンはドライバがどの特定のODBC関数をサポートしているかを決定することを可能にします。

＜ログインID＞は有効なログインIDでなければなりません。

＜関数ID＞はODBC関数の参照番号です。参照番号のリストはこのルーチンの説明の後に続きます。

＜4Dオブジェクト＞はサポートされている関数名を格納する4th Dimensionの変数名もしくはフィールド名です。

OC Get functionは関数が実装されていれば1を、そうでなければ-1を返します。

関数IDの参照番号は次の通りです：

ODBC関数各レベル	ID	4Dに実装されているか
コアレベル		
SQL_API_SQLALLOCCONNECT	1	はい
SQL_API_SQLALLOCENV	2	はい
SQL_API_SQLALLOCSTMT	3	はい
SQL_API_SQLBINDCOL	4	はい
SQL_API_SQLCANCEL	5	はい
SQL_API_SQLCOLATTRIBUTES	6	はい
SQL_API_SQLCONNECT	7	はい
SQL_API_SQLDESCRIBECOL	8	はい
SQL_API_SQLDISCONNECT	9	はい
SQL_API_SQLERROR	10	はい
SQL_API_SQLEXECDIRECT	11	はい
SQL_API_SQLEXECUTE	12	はい
SQL_API_SQLFETCH	13	はい
SQL_API_SQLFREECONNECT	14	はい

SQL_API_SQLFREEENV	15	はい
SQL_API_SQLFREESTMT	16	はい
SQL_API_SQLGETCURSORNAME	17	はい
SQL_API_SQLNUMRESULTCOLS	18	はい
SQL_API_SQLPREPARE	19	はい
SQL_API_SQLROWCOUNT	20	はい
SQL_API_SQLSETCURSORNAME	21	はい
SQL_API_SQLSETPARAM	22	はい
SQL_API_SQLTRANSACT	23	はい

レベル1

SQL_API_SQLCOLUMNS	40	はい
SQL_API_SQLDRIVERCONNECT	41	
SQL_API_SQLGETCONNECTOPTION	42	はい
SQL_API_SQLGETDATA	43	はい
SQL_API_SQLGETFUNCTIONS	44	はい
SQL_API_SQLGETINFO	45	はい
SQL_API_SQLGETSTMTOPTION	46	はい
SQL_API_SQLGETTYPEINFO	47	はい
SQL_API_SQLPARAMDATA	48	
SQL_API_SQLPUTDATA	49	
SQL_API_SQLSETCONNECTOPTION	50	はい
SQL_API_SQLSETSTMTOPTION	51	はい
SQL_API_SQLSPECIALCOLUMNS	52	
SQL_API_SQLSTATISTICS	53	
SQL_API_SQLTABLES	54	はい

レベル2

SQL_API_SQLBROWSECONNECT	55	
SQL_API_SQLCOLUMNPRIVILEGES	56	はい
SQL_API_SQLDATASOURCES	57	はい
SQL_API_SQLDESCRIBEPARAM	58	はい
SQL_API_SQLEXTENDEDFETCH	59	
SQL_API_SQLFOREIGNKEYS	60	
SQL_API_SQLMORERESULTS	61	はい
SQL_API_SQLNATIVESQL	62	
SQL_API_SQLNUMPARAMS	63	はい
SQL_API_SQLPARAMOPTIONS	64	
SQL_API_SQLPRIMARYKEYS	65	はい
SQL_API_SQLPROCEDURECOLUMNS	66	はい
SQL_API_SQLPROCEDURES	67	はい

SQL_API_SQLSETPOS	68	
SQL_API_SQLSETSCROLLOPTIONS	69	
SQL_API_SQLTABLEPRIVILEGES	70	はい
SQL_API_SQLDRIVERS	71	
SQL_API_SQLBINDPARAMETER	72	

例題

次の式は関数ID番号についてユーザに問い合わせ、実装されていれば関数名を返します。

```
C_STRING (30;var1)
$num:=Request ("Enter ID #")
   $res:=OC Get function (login;Num($num);var1)
If ($res=1)
   ALERT (var1+ "is Implemented ”)
Else
   ALERT ("Not Implemented")
End if
```

参照

なし

OC Get info

OC Get info (ログインID;情報ID)→文字列

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
情報ID	整数	→	関数の参照番号
戻り値	文字列	←	全般的な情報とデータソース、もしくはエラーの場合空文字列。

説明

OC Get info関数はドライバとHDBCに関連付けられたデータソースに関する全般的な情報を検索します。

＜ログインID＞は有効なログインIDでなければなりません。

情報IDはリクエストされた情報の参照番号です：

関数	ID
SQL_ACTIVE_CONNECTIONS	0
SQL_ACTIVE_STATEMENTS	1
SQL_DATA_SOURCE_NAME	2
SQL_DRIVER_HDBC	3
SQL_DRIVER_HENV	4
SQL_DRIVER_HSTMT	5
SQL_DRIVER_NAME	6
SQL_DRIVER_VER	7
SQL_FETCH_DIRECTION	8
SQL_ODBC_API_CONFORMANCE	9
SQL_ODBC_VER	10
SQL_ROW_UPDATES	11
SQL_ODBC_SAG_CLI_CONFORMANCE	12
SQL_SERVER_NAME	13
SQL_SEARCH_PATTERN_ESCAPE	14
SQL_ODBC_SQL_CONFORMANCE	15
SQL_DATABASE_NAME	16
SQL_RDBMS_NAME	17
SQL_RDBMS_VER	18
SQL_ACCESSIBLE_TABLES	19
SQL_ACCESSIBLE_PROCEDURES	20
SQL_PROCEDURES	21
SQL_CONCAT_NULL_BEHAVIOR	22

SQL_CURSOR_COMMIT_BEHAVIOR	23
SQL_CURSOR_ROLLBACK_BEHAVIOR	24
SQL_DATA_SOURCE_READ_ONLY	25
SQL_DEFAULT_TXN_ISOLATION	26
SQL_EXPRESSIONS_IN_ORDERBY	27
SQL_IDENTIFIER_CASE	28
SQL_IDENTIFIER_QUOTE_CHAR	29
SQL_MAX_COLUMN_NAME_LEN	30
SQL_MAX_CURSOR_NAME_LEN	31
SQL_MAX_OWNER_NAME_LEN	32
SQL_MAX_PROCEDURE_NAME_LEN	33
SQL_MAX_QUALIFIER_NAME_LEN	34
SQL_MAX_TABLE_NAME_LEN	35
SQL_MULT_RESULT_SETS	36
SQL_MULTIPLE_ACTIVE_TXN	37
SQL_OUTER_JOINS	38
SQL_OWNER_TERM	39
SQL_PROCEDURE_TERM	40
SQL_QUALIFIER_NAME_SEPARATOR	41
SQL_QUALIFIER_TERM	42
SQL_SCROLL_CONCURRENCY	43
SQL_SCROLL_OPTIONS	44
SQL_TABLE_TERM	45
SQL_TXN_CAPABLE	46
SQL_USER_NAME	47
SQL_CONVERT_FUNCTIONS	48
SQL_NUMERIC_FUNCTIONS	49
SQL_STRING_FUNCTIONS	50
SQL_SYSTEM_FUNCTIONS	51
SQL_TIMEDATE_FUNCTIONS	52
SQL_CONVERT_BIGINT	53
SQL_CONVERT_BINARY	54
SQL_CONVERT_BIT	55
SQL_CONVERT_CHAR	56
SQL_CONVERT_DATE	57
SQL_CONVERT_DECIMAL	58
SQL_CONVERT_DOUBLE	59
SQL_CONVERT_FLOAT	60
SQL_CONVERT_INTEGER	61
SQL_CONVERT_LONGVARCHAR	62
SQL_CONVERT_NUMERIC	63

SQL_MAX_STATEMENT_LEN	105
SQL_MAX_TABLES_IN_SELECT	106
SQL_MAX_USER_NAME_LEN	107
SQL_MAX_CHAR_LITERAL_LEN	108
SQL_TIMEDATE_ADD_INTERVALS	109
SQL_TIMEDATE_DIFF_INTERVALS	110
SQL_NEED_LONG_DATA_LEN	111
SQL_MAX_BINARY_LITERAL_LEN	112
SQL_LIKE_ESCAPE_CLAUSE	113
SQL_QUALIFIER_LOCATION	114

OC Get infoは成功したならリクエストされた情報を、失敗したなら空文字列を返します。

例題

この例はSQL_POSITIONED_STATEMENTS関数（ID=80）に関連する情報を得て、その結果を変数$resultに返します。

```
$result:=OC Get info (login;80)
```

参照

なし

OC Get login option

OC Get login option (ログインID;オプションID)→文字列

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
オプションID	整数	→	関数参照番号
戻り値	文字列	←	接続オプションの値、 もしくはエラーの場合は空文字列

説明

OC Get login option関数は接続オプションの値を検索します。

＜ログインID＞は有効なログインIDでなければなりません。

＜オプションID＞はリクエストされたオプションの参照番号です。参照番号のリストはこのルーチンの説明の後に続きます。

OC Get login optionは成功したならリクエストされた情報を返します。失敗したなら空文字列を返します。

オプションIDの参照番号は次の通りです：

関数	ID
SQL_ACCESS_MODE	101
SQL_AUTOCOMMIT	102
SQL_LOGIN_TIMEOUT	103
SQL_OPT_TRACE	104
SQL_OPT_TRACEFILE	105
SQL_TRANSLATE_DLL	106
SQL_TRANSLATE_OPTION	107
SQL_TXN_ISOLATION	108
SQL_CURRENT_QUALIFIER	109
SQL_ODBC_CURSORS	110
SQL_QUIET_MODE	111
SQL_PACKET_SIZE	112

例題

次の式SQL_AUTOCOMMIT option（ID=102）に関する情報を得て、その結果を変数$resultに返します。

```
$result:=OC Get login option (login;102)
```

参照
OC Set login option

OC Set cursor option

OC Set cursor option (ログインID;オプションID;オプション値)→倍長整数

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
オプションID	整数	→	関数参照番号
オプション値	文字列	→	オプションがセットされた値
戻り値	倍長整数	←	1=正常終了、0=エラー発生 -1=オプションIDが存在しない

説明

OC Set cursor option関数はカーソルに関連するコンフィグレーション情報をアプリケーションにセットすることを可能にします。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

＜オプションID＞はリクエストされたオプションの参照番号です。参照番号のリストはこのルーチンの説明の後に続きます。

オプション値はセットされるオプションの値です。

オプションIDの参照番号は次の通りです：

関数	ID
SQL_QUERY_TIMEOUT	0
SQL_MAX_ROWS	1
SQL_NOSCAN	2
SQL_MAX_LENGTH	3
SQL_ASYNC_ENABLE	4
SQL_BIND_TYPE	5
SQL_CURSOR_TYPE	6
SQL_CONCURRENCY	7
SQL_KEYSET_SIZE	8
SQL_ROWSET_SIZE	9
SQL_SIMULATE_CURSOR	10
SQL_RETRIEVE_DATA	11
SQL_USE_BOOKMARKS	12
SQL_GET_BOOKMARKS	13
SQL_ROW_NUMBER	14

OC Set cursor optionは処理が正常に終了した場合1を、失敗したならば0を、オプションIDが存在しない場合は-1を返します。

例題

次の手続きはSQL_AUTOCOMMIT option（ID=102）をマニュアル（値=0）にセットし、その結果を変数$resultに返します。

```
$result:=OC Set cursor option (login;102;"0")
```

参照

OC Get cursor option

OC Set login option

OC Set login option (ログインID;オプションID;オプション値)→倍長整数

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
オプションID	整数	→	関数参照番号
オプション値	文字列	→	オプションがセットされた値
戻り値	倍長整数	←	1=正常終了、0=エラー発生 -1 = オプションIDが存在しない

説明

OC Set login option関数はアプリケーションに接続属性をセットすることを可能にします。

＜ログインID＞は有効なログインIDでなければなりません。

＜オプションID＞はリクエストされたオプションの参照番号です。参照番号のリストはこのルーチンの説明の後に続きます。

＜オプション値＞はセットされるオプションの値です。

オプションIDの参照番号は次の通りです：

関数	ID
SQL_ACCESS_MODE	101
SQL_AUTOCOMMIT	102
SQL_LOGIN_TIMEOUT	103
SQL_OPT_TRACE	104
SQL_OPT_TRACEFILE	105

SQL_TRANSLATE_DLL	106
SQL_TRANSLATE_OPTION	107
SQL_TXN_ISOLATION	108
SQL_CURRENT_QUALIFIER	109
SQL_ODBC_CURSORS	110
SQL_QUIET_MODE	111
SQL_PACKET_SIZE	112

OC Set login optionは処理が正常に終了した場合1を、失敗したならば0を、オプションIDが存在しない場合は-1を返します。

例題

次の式はSQL_AUTOCOMMIT option（ID=102）をマニュアル（値=0）にセットし、その結果を変数$resultに返します。

```
$result:=OC Set login option (loginID;102;"0")
```

特殊なケース

オプション103（SQL_LOGIN_TIMEOUT）は将来にわたってすべての接続にタイムアウトをセットすることを可能にします。ログインIDは有効でなくてもかまいませんが、（引数の正しい番号を渡すために）ログインIDは倍長整数でなければなりません。例えば**OC Set Login option**（LoginID;103;"20"）はタイムアウトを20秒にセットします。デフォルトのドライバの値にするには第3引数を0にしなくてはなりません。

参照

OC Get login option

OC SET PACK OPTIONS

OC SET PACK OPTIONS (オプションID;オプション値)

引数	タイプ		説明
オプションID	整数	→	オプション番号
オプション値	文字列	→	オプション値

説明

OC SET PACK OPTIONSコマンドはグローバルオプションを4D ODBCパッケージレベルにセットすることを可能にします。

＜オプションID＞は修正するオプション番号です。オプション値引数で値をオプションとして認識します。

ただ1つのオプションが差し当たり利用可能です：

ID	値	説明
1	MDY、MYD、DMY DYM、YMD、YDM	サーバが4Dとは異なった日付フォーマットを使用している時4D ODBCのために特定のデータフォーマットを指定することを可能にします。

注：このコマンドには必要とされる新しいオプションが4D ODBCの将来のバージョンに追加されるでしょう。

例題

日付フォーマットをD（日）、M（月）、Y（年）と設定するには次のようにします。

```
OC SET PACK OPTIONS (1;"DMY")
```

参照

OC Get pack options

OC Get pack options

OC Get pack options (オプションID)→文字列

引数	タイプ		説明
オプションID	整数	→	オプション番号
戻り値	文字列	←	現在のオプション値

説明

OC Get pack optionsコマンドは4D ODBCパッケージグローバルオプションの現在のオプションIDの値を返します。

現バージョンで利用できるオプションは1つだけです：

ID	値	説明
1	MDY、MYD、DMY DYM、YMD、YDM	サーバが4Dとは異なった日付フォーマットを使用している時4D ODBCのために特定のデータフォーマットを指定することを可能にします。

参照

OC SET PACK OPTIONS

コントロールコマンド、はじめに

この章のコマンド群は4th Dimentionにデータをロードすることをキャンセルした場合、起こり得るエラーを管理できるようにするものです。コントロールコマンドを使って次のようなことが可能となります。

- エラー処理方法をインストール（**OC SET ERROR HANDLER**）
- デバッグウィンドウの開閉（**OC OPEN DEBUG WINDOW**、**OC CLOSE DEBUG WINDOW**）
- デバッグウィンドウへのメッセージ表示（**OC DEBUG MESSAGE**）
- クライアントコンフィグレーションのチェック（**OC Check configuration**）

OC Check configuration

OC Check configuration →整数

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	整数	←	エラーコード

説明

OC Check configurationはコマンドによってクライアントのコンフィグレーションをチェックし、プラグインが正常に働かない場合に、何が誤っているのかを知ることを可能にします。

次のリストは関数の戻り値です：

戻り値	説明
1	エラーなし。すべてのテストは成功しています。Macintosh、Windows
-1	ライブラリが見つかりませんでした。Macintosh
-2	メモリ不足のため、ライブラリをロードできませんでした。Macintosh
-3	ライブラリのバージョンが4D ODBCのこのバージョンでは サポートされていないか、ライブラリがロードされませんでした。Macintosh
-101	DLLが見つけられませんでした。Windows
-102	DLLがロードできませんでした。Windows

例題

この例はデータベースの使用を開始する式に使用されます。

```
$check:=OC Check configuration
Case of
   \ ($check=1)
      <>ServerAvailable:=True
   \ ($check=-1) ` API not present
      <>ServerAvailable:=False
   Else
      ALERT (" The ODBC API is partially missing or incorrect")
      <>ServerAvailable:=False
End case
If (<>ServerAvailable)
   DO_Synchro ([Customers])
End if
```

参照

なし

OC CLOSE DEBUG WINDOW

OC CLOSE DEBUG WINDOW

引数	タイプ	説明

このコマンドには、引数はありません。

説明

OC CLOSE DEBUG WINDOWコマンドは**OC OPEN DEBUG WINDOW**によって開かれたデバッグウィンドウを閉じます。

参照

OC DEBUG MESSAGE、OC OPEN DEBUG WINDOW

OC DEBUG MESSAGE

OC DEBUG MESSAGE (テキスト)

引数	タイプ		説明
テキスト	テキスト	→	メッセージテキスト

説明

OC DEBUG MESSAGEコマンドは**OC OPEN DEBUG WINDOW**によって開かれたデバッグウィンドウにテキストを表示します。

参照

OC CLOSE DEBUG WINDOW、OC OPEN DEBUG WINDOW

OC OPEN DEBUG WINDOW

OC OPEN DEBUG WINDOW

引数	タイプ	説明

このコマンドには、引数はありません。

説明

OC OPEN DEBUG WINDOWコマンドは4D ODBCコマンドへの呼び出しをリアルタイムにトレースするウィンドウを開きます。

このウィンドウは次のメニューをメニューバーに含んでいます：

- **ファイルメニュー**：このメニューはプリント、閉じる、ウィンドウの内容を記録するといったコマンドを含んでいます。
- **オプションメニュー**：このメニューはプロシージャパラメータ、結果、エラー、メッセージ、関数名が表示されているか否かにかかわらず表示されたデバッギング情報のレベルをウィンドウにセットします。

注：4D ODBCデバッガウィンドウは4D ODBCデバッグメッセージを表示するため、様々な4D処理を分離して管理するよう設計されています。4D ODBCバージョン6.0.2以降では、メッセージを表示しようと試す前にデバッガウィンドウでメッセージを表示する必要があるどんな4D ODBC呼び出しでもウィンドウ処理が始動し終えるのを待ちます。この変更はODBC OPEN DEBUG WINDOW呼び出しとウィンドウが実際に表示される間の段階でその情報が失われることを防止するためです。

参照

OC CLOSE DEBUG WINDOW、OC DEBUG MESSAGE

OC SET ERROR HANDLER

OC SET ERROR HANDLER (方式名)

引数	タイプ		説明
方式名	文字列	→	エラー処理方法の名前

説明

OC SET ERROR HANDLERコマンドはエラーが発生する毎に実行されるエラー処理方法をインストールします。これは起こり得る実行エラーをコントロールし、そしてデフォルトのエラー処理を無効にすることを可能にします。

＜方式名＞はインストールする方式の名前です。

デフォルトの動作に戻すには**OC SET ERROR HANDLER**("") のような空文字列を渡してください。

4D ODBCはその方式に5つのアーギュメントを渡します：

引数	タイプ	説明
$1	倍長整数	エラー番号
$2	倍長整数	エラーレベル
$3	テキスト	エラーの説明
$4	倍長整数	4D プロセスＩＤ
$5	文字列	エラーのSQLステータス

データベースを編集するには**C_LONGINT**、**C_TEXT**コマンドを使って$1から$5までの値を宣言しなくてはなりません。

デフォルトでは、エラー処理方法がインストールされないなら、すべてのエラーは4D ODBCによって受取られ、エラーの説明を表示して警告します。

参照

なし

OC TRANSACT COMMAND

OC TRANSACT COMMAND (ログインID,カーソルID,コミット／ロールバック)

引数	タイプ		説明
ログインID	倍長精通	→	ログインID
カーソルID	倍長整数	→	カーソルID
コミット/ロールバック	ブール	→	False＝コミット、True＝ロールバック

説明

OC TRANSACT COMMANDコマンドはトランザクションをコミットするかロールバックします。

トランザクションを使うために、始めに**OC Set login option**を使って接続をマニュアルコミットモードにセットしなくてはなりません。

若干のデータソースは、まったくトランザクションをサポートしません。どんな種類の文がトランザクションに含まれることができるかについて、他のものが制限うを確認します。データソースがどのような種類のトランザクション処理をサポートするかを見つけ出すために、SQL_TXN_CAPABLE値を得るために**OC Get info**を使用してください。

参照

OC Set login option

コンテキストコマンドの定義、はじめに

コンテキストは、テーブルのコラムと4th Dimensionのフィールド、変数、配列との間のバインドから構成されています。コンテキストは一度定義されれば、SQL文を使わずに4th DimensionにODBCデータをロードし、ODBCデータソースの情報を更新することができます。

この章のコンテキスト定義コマンドによって、次のことができます。

- コンテキストを作成する（**OC Create context dialog**、**OC Create context**）
- コンテキストにバインドを定義する（**OC ADD TO CONTEXT**）
- 節をコンテキストに追加する（**OC EDIT CLAUSES IN CONTEXT**、**OC SET CLAUSE IN CONTEXT**）
- コンテキスト定義をセーブ、またはロードする（**OC Save context picture**、**OC Save context file**、**OC Load context picture**、**OC Load context file**）
- コンテキストを閉じる（**OC DROP CONTEXT**）

ダイアログボックスの使用、または数行のSQLステートメントの実行により、コンテキストを作成できます。どちらかの方法に関する詳細は、第3章「コンテキストの使用」を参照してください。

OC ADD TO CONTEXT

OC ADD TO CONTEXT (コンテキストID;列名;4Dオブジェクト)

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
列名	文字列	→	ODBC 列または式
4Dオブジェクト	文字列	→	4th Dimension変数と配列名

説明

OC ADD TO CONTEXTコマンドによって、ODBCコラム、表現と4th Dimension オブジェクトをバインドします。**OC Create context**によって初期化されたコンテキスト定義にバインドは追加されます。

＜コンテキストID＞は、あらかじめ作成され、使われていないコンテキストIDでなければなりません。

＜列名＞によって、ODBCコラム名（テーブルコラム方式、またはコラム形式中）か、("sal*1.15"のような）SQL表現を指定します。選択したいコラムにデフォルトテーブル名を宣言しないなら、テーブル．コラム形式を用いてコラムがあるテーブルを指定しなければなりません。**OC Create context dialog**や**OC Create context**を用いて、デフォルトテーブルを指定できます。

＜4Dオブジェクト＞は4th Dimensionのフィールド名、変数名、または配列名です。

例題

次の方法では、コンテキスト定義を初期化し、コンテキスト内のバインドを定義しています。

```
Context_ID=OC Create context ( "EMP")     ` Initiate a context definition
OC ADD TO CONTEXT (Context_ID; "empno"; "[Employees]No")
   OC ADD TO CONTEXT (Context_ID; "ename"; "[Employees]Name")
   OC ADD TO CONTEXT (Context_ID; "sal"; "vSalary")
```

OC Activate contextを用いてコンテキストが動作するとき、次のSELECT文はODBCサーバに送信されます。

```
SELECT empno, ename, sal FROM emp
```

参照

OC Activate context、OC Create context

OC Create context

OC Create context ({テーブル名; {;明確な節{;更新{;ノーウェイト}}}})→倍長整数

引数	タイプ		説明
テーブル名	文字列	→	デフォルトODBCテーブル
明確な節	ブール	→	使われた別の節
更新	ブール	→	更新節
ノーウェイト	ブール	→	ノーウェイト節
戻り値	倍長整数	←	コンテキストID、 エラーが発生した場合1

説明

OC Create context機能は、コンテキストを作ることを望むことを示し、そのコンテキストのIDを返します。

この機能の実行によって、いかなる4th Dimensionのフィールド、変数、配列もODBCコラムと関連付けしません。バインドごとに**OC ADD TO CONTEXT**を呼び出すことで、これらのバインドを指定します。

明示的にテーブル名を付けないならば、テーブル名は**OC ADD TO CONTEXT**コマンドのためにデフォルトテーブルとして使用されるテーブルを指定します。テーブル名はユーザテーブル、もしくはテーブル形式の中に存在しなければなりません。

別のオプション選択で、SQL文が特有の行を返すことを明示します。この機能がデータソースでサポートされるならば、更新オプションの選択により、列をロックし始めるでしょう。

ノーウェイトオプション選択で、それがもう1人のユーザーによってロックされるなら、読み込み専用モードの列かどうかアクセスするでしょう。

デフォルトで、4D ODBCはロードする前に、列がフリーになるまで待ちます。エラー発生時、**OC Create context**は-1を返します。その他の状態では、0より大きい妥当な数を返します。

OC Create contextと**OC ADD TO CONTEXT**を用いてコンテキストを作成した後、**OC Activate context**の呼び出しによりコンテキストを作動させなければなりません。

例題

次の文で、「Scott.EMP」コンテキストを初期化します。

```
Context_ID=OC Create context ("Scott.EMP")
```

参照

OC Activate context、OC ADD TO CONTEXT、OC Create context dialog

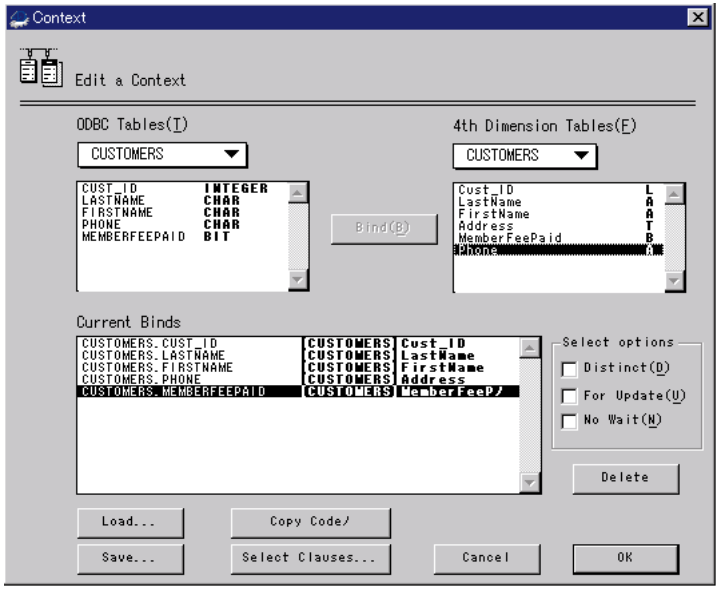
OC Create context dialog

OC Create context dialog (ログインID{;テーブル名{;4Dテーブル番号}})→倍長整数

引数	タイプ		説明
ログインID	倍長整数	→	接続ID
テーブル名	文字列	→	デフォルトのODBCテーブル名
4Dテーブル番号	整数	→	デフォルトの4th Dimensionテーブル番号
戻り値	倍長整数	←	コンテキストID、 エラーが発生した場合は1

説明

OC Create context dialog機能はEdit a Contextダイアログボックスを表示します。このダイアログボックスの使用で、4th Dimensionフィールドと ODBCデータソーステーブルのコラム間の動的リンクを作成できます。



コンテキストを定義するEdit a Contextダイアログボックスの使用に関する詳細は、第3章「コンテキストの利用」を参照してください。

＜ログインID＞は正しいログインIDでなければなりません。ログインIDによって、テーブルポップアップメニューのためのテーブルリストのデータソースを決定します。このリストは、ログインユーザがアクセスするそれらのテーブルをベースとしています。

＜テーブル名＞はデフォルトODBCテーブルポップアップメニューに現われるべきデータソーステーブルを特定します。

明白にコマンドのためのテーブル名をつけないなら、このテーブルは、**OC ADD TO CONTEXT**コマンドのデフォルトテーブルとして使用されるでしょう。＜テーブル名＞は、ユーザテーブル、もしくは テーブル形式の中になければなりません。

＜4Dテーブル名＞はデフォルトで4th Dimensionテーブルポップアップメニューに現われるべき4th Dimensionテーブル番号を指定します。

ユーザがダイアログボックスの「OK」をクリックすると、**OC Activate context**を呼び出して、コンテキストを発動する時、**OC Create context dialog**が使用するコンテキストIDを返します。エラー発生時、**OC Create context dialog**は-1を返します。

例題

次の文で、Edit a Contextダイアログボックスを表示します。

Context_ID=***OC Create context dialog*** (login_ID)

参照

OC Activate context、OC ADD TO CONTEXT、OC Create context

システム変数およびセット

ユーザがダイアログボックスで「OK」をクリックする時、システム変数OKは1に設定されます。ユーザがダイアログボックスで「Cancel」をクリックする時、**OC Create context dialog**は0を返します。さらに、システム変数OKは0に設定されます。

OC DROP CONTEXT

OC DROP CONTEXT (コンテキストID)

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID

説明

OC DROP CONTEXTコマンドは、コンテキストIDで識別されるコンテキストの定義によって使用されるメモリを解放します。

OC DROP CONTEXTコマンドの呼び出しで、コンテキストを閉じた後、コンテキストはもはや使用できず、そのIDは不正になります。**OC DROP CONTEXT**が呼び出された時、コンテキストが有効になるなら、それは最初に無効化されます。

例題

次の方法で、コンテキストを作成し、閉じられるダイアログボックス表示します。

```
Context_ID=OC Create context dialog (login_ID)
OC DROP CONTEXT (Context_ID)
```

参照

OC Create context、OC Create context dialog、OC DEACTIVATE CONTEXT

OC EDIT CLAUSES IN CONTEXT

OC EDIT CLAUSES IN CONTEXT (ログインID;コンテキストID)

引数	タイプ		説明
ログインID	倍長整数	→	接続ID
コンテキストID	倍長整数	→	コンテキストID

説明

OC EDIT CLAUSES IN CONTEXTコマンドは、Edit a SQL節ダイアログボックスを表示して、コンテキストを指定する節を編集することができます。

＜ログインID＞はアクセス可能なテーブル、コラムのリストをロードするために使用する正当な通信IDに違いありません。

＜コンテキストID＞はあらかじめ作成され、無効化されたコンテキストIDでなければなりません。

ダイアログボックスの使用で、次の節を編集できます：

- WHERE
- GROUP BY
- HAVING
- CONNECT BY
- START WITH

例題

次の方法で、コンテキストを作成し、コンテキストのための節を指定できるダイアログボックスを表示します。

```
Context_ID=OC Create context ("EMP")
   ` コンテキストの作成
OC ADD TO CONTEXT (Context_ID;"empno";"[Employees]No")
OC ADD TO CONTEXT (Context_ID;"ename";"[Employees]Name")
   ` バインドの定義
OC EDIT CLAUSES IN CONTEXT (Login_ID;Context_ID)
   ` パーソナライズクエリ
```

参照

OC Get clause in context、OC SET CLAUSE IN CONTEXT

システム変数およびセット

ユーザがダイアログボックスで「OK」をクリックする時、システム変数OKは1に設定されます。ユーザがダイアログボックスで「Cancel」をクリックする時、システム変数OKは0に設定されます。

OC Get clause in context

OC Get clause in context (コンテキストID;節番号)→テキスト

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
節番号	整数	→	節番号
戻り値	テキスト	←	コンテキスト節のテキストまたは エラーが発生したなら空文字列

説明

OC Get clause in context機能は、コンテキスト節のテキストを検索します。エラー発生時、**OC Get clause in context**は空文字列を返します。

＜コンテキストID＞はあらかじめ作成され、有効化、または無効化されたコンテキストIDでなければなりません。

節番号は次の節番号のひとつであり得ます：

1 FROM
2 WHERE
3 GROUP BY
4 HAVING
5 CONNECT BY
6 START WITH
7 ORDER BY

例題

次の方法で、ユーザはEdit a Contextダイアログボックスを使用して、自身のコンテキストを定義することができます。

ダイアログボックスの使用で、ユーザはWHERE節のような節を追加することができます。WHERE節を検索した後、ユーザが重要な情報を得られないように、WHERE節に追加制限を加えます。

```
Context_ID=OC Create context dialog (Login_ID)      ` コンテキストの作成
If (Context_ID>0)                                   ` If user did not click Cancel
  clause:=OC Get clause in context (Context_ID;2)   ` WHERE節の検索
  If (clause#"")                                    ` If WHERE clause exists
    clause:="("+clause+") AND "                     ` AND 追加
  End if                                            ` 連結の準備
  clause:=clause+"EMP.job <> 'President'"
                                                    `WHERE節の定義とコンディションの追加
  OC SET CLAUSE IN CONTEXT (Context_ID;2;clause)
  ` ....
End if
```

参照

OC EDIT CLAUSES IN CONTEXT、OC SET CLAUSE IN CONTEXT

OC Load context file

OC Load context file (ファイルネーム) → 倍長整数

引数	タイプ		説明
ファイルネーム	テキスト	→	コンテキストの説明を含むファイルの名前
戻り値	倍長整数	←	コンテキストID、 エラーが発生したなら-1、 キャンセルボタンをクリックするなら0

説明

OC Load context file機能によって、**OC SAVE CONTEXT FILE**を使用して、ディスク上のファイルにあらかじめセーブされたコンテキスト定義を含んでいるドキュメントを開きます。

OC Load context fileは新規に開かれたコンテキストのIDを返します。ファイルネームが空文字列ならば、Open Fileダイアログボックスが表示されます。ファイルネームが完全

なパス名でないなら、**OC Load context file**はデータベース構造ファイルを含むフォルダ内のドキュメントを探します。

OC Load context fileはエラー時には-1を返し、ユーザがOpen fileダイアログボックスの「Cancel」ボタンをクリックしたときには0を返します。

OC Activate contextを呼び出して、コンテキストを有効化できます。

参照

OC Create context、OC Create context dialog、OC SAVE CONTEXT FILE

OC Load context picture

OC Load context picture (コンテキストピクチャ) → 倍長整数

引数	タイプ		説明
コンテキストピクチャ	ピクチャ	→	コンテキストの説明を含むピクチャ変数
戻り値	倍長整数	←	コンテキストID、エラーの場合は-1

説明

OC Load context picture機能では、ピクチャフィールド、変数からコンテキスト定義をロードします。この機能は新規作成されたコンテキストのIDを返します。**OC Save context picture**を使用して、このコンテキストはあらかじめ作成されています。**OC Load context picture**はエラー時に-1を返します。

OC Activate contextの呼び出しで、コンテキストを有効化してもよいです。

データタイプはリクエストされたデータタイプ属性のための参照番号です：

データタイプ属性	ID
名前	1
データタイプ	2

参照

OC Create context、OC Create context dialog、OC Save context picture

OC SAVE CONTEXT FILE

OC SAVE CONTEXT FILE (コンテキストID;ファイルネーム)

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストIDのセーブ
ファイルネーム	テキスト	→	コンテキスト定義を格納するファイル名

説明

OC SAVE CONTEXT FILEコマンドは、ディスク上のファイルにコンテキスト定義をセーブします。

＜コンテキストID＞はあらかじめ作成され、有効あるいは無効なコンテキストIDでなければなりません。

＜ファイルネーム＞は作成したドキュメント名を指定します。ファイルネームが空文字列ならば、ユーザに対し、Save Fileダイアログボックスが表示されます。ファイルネームが完全なパス名でないならば、**OC SAVE CONTEXT FILE**はデータベース構造ファイルと同じフォルダにファイルをセーブします。

同名のドキュメントがすでに存在するなら、上書きされます。

参照

OC Load context file

OC Save context picture

OC Save context picture (コンテキストID) → ピクチャ

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
戻り値	ピクチャ	←	コンテキスト定義のピクチャ

説明

OC Save context picture機能は、ピクチャタイプの変数の形でコンテキスト定義が返されます。変数は後の検索のため、ピクチャフィールドにセーブされます。

＜コンテキストID＞はあらかじめ作成され、有効あるいは無効なコンテキストIDでなければなりません。

参照

OC Load context picture

OC SET CLAUSE IN CONTEXT

OC SET CLAUSE IN CONTEXT (コンテキストID;節番号;節)

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
節番号	整数	→	セットされた節の番号
節	テキスト	→	節のテキスト

説明

OC SET CLAUSE IN CONTEXTコマンドは**OC EDIT CLAUSES IN CONTEXT**によってグラフィックにすることを、手続き的に達成します。

ユーザに独自の節を構築して欲しくない場合、**OC SET CLAUSE IN CONTEXT**を使用するべきです。ユーザに節を制御してもらうが、節を入れるために別のインタフェースを作成してもらいたいなら、節引数に節を含む変数を指定できます。

OC SET CLAUSE IN CONTEXTでコンテキストに節を定義できます。

＜コンテキストID＞はあらかじめ作成され、無効なコンテキストIDでなければなりません。

節番号は定義したい節をあらわす番号でなければなりません：

2 WHERE
3 GROUP BY
4 HAVING
5 CONNECT BY
6 START WITH
7 ORDER BY

節は、関連付けられたキーワードのない節（すなわち、"WHERE"や他のキーワードがない節）のテキストです。

例題

次の文では、第30課の従業員の検索に使用できるWHERE節を作成します。

OC SET CLAUSE IN CONTEXT (Context_ID;2;"DEPTNO = 30")

参照

OC EDIT CLAUSES IN CONTEXT、OC Get clause in context

ログインコマンド、はじめに

この章のコマンドによって、ODBCデータソースへのログインとログアウト、通信における複数の上位レベル処理が実行可能です。

ログイン時、4D ODBCはコマンドを適用すべき通信を識別する多くの4D ODBCコマンドで使用される通信IDを返します。通信IDは、サーバからログアウトする場合にも使用されます。

ログインおよびログアウトコマンド

これらのコマンドで、ODBCサーバとの通信を制御できます。

ログインコマンドの使用で、次のことができます。

- ダイアログボックスを用いたログイン（**OC Login Dialog**）
- ログインパラメータの指定によるログイン（**OC Login**）
- ログアウト（**OC LOGOUT**）

上位コマンド

いくつかの上位コマンドによって、通信レベルでの制御を実現できます。これらのコマンドは、下位レベルAPIコマンドを連結し、単純なコマンドで複雑な機能を実現できます。

これらのコマンドを用いて、次のようなことが可能です。

- SQL文の実行（**OC Query exec**または**OC Execute**）
- ODBCデータソース上の4th Dimensionファイルの複製（**OC Clone 4D table**）
- 4th Dimensionファイル作成のために、ODBCデータソースからのテーブルを複製（**OC Clone ODBC table**）

OC Clone 4D table

OC Clone 4D table (接続ID{;4Dテーブル番号{;オプション}}) R Integer

引数	タイプ		説明
接続ID	倍長整数	→	Login_ID
4Dテーブル番号	整数	→	4Dテーブル番号
オプション	倍長整数	→	オプション
戻り値	整数	←	1=the operation is successful、 0=the user clicks Cancel -1=an error occurs

説明

OC Clone 4D tableコマンドは、4th Dimensionテーブルと等しいコラム定義でデータソース上にテーブルを作成します。

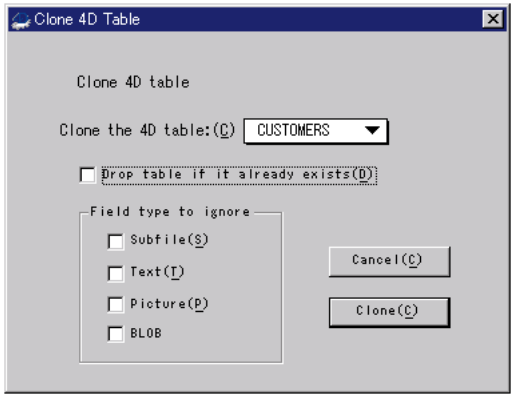
＜接続ID＞は正しいログインIDもしくはカーソルIDでなければなりません。正しいログインIDを使用するなら、4D ODBCは、この機能の使用により、自動的にカーソルの開閉ができるでしょう。正しいカーソルIDをパスすれば、この機能で特有のカーソルを使用できるでしょう。

＜4Dテーブル番号＞は4th Dimensionテーブルの番号で、その構造は複写されます。

＜オプション＞は、オプションの組み合わせです。複数のオプションは値の追加で選ばれます。オプションとは次のようなものです。

Replace=1	**OC Clone 4D table**は新しいテーブルを作成する前にDROP TABLE文を作成します。
WithoutSubfile=2	**OC Clone 4D table**はサブファイルを無視します。 サブテーブルの複製は現在サポートされていません。 このオプションはサブテーブルを含むいくつかのテーブルに要求されねばなりません。
WithoutText=4	**OC Clone 4D table**はTextタイプのフィールドを無視します。
WithoutPicture=8	**OC Clone 4D table**はPictureタイプのフィールドを無視します。
kWithout=16	**OC Clone 4D Table**はBLOBタイプのフィールドを無視します。

4D TableNbrが指定されないなら、**OC Clone 4D table**は、Clone 4D Tableダイアログボックスを表示します。



ポップアップメニューによって、複製される4th Dimensionテーブル、利用できるオプションのチェックボックスリストを選択できます。

新しいテーブルは、4th Dimensionテーブル名を持ちます。またそのコラムは、4th Dimensionテーブル上のフィールド名を持ちます。フィールド名がスペースを含むなら、そのスペースは下線に置き換わります。

タイプは次のルールで変換されます：

4th Dimension	ODBC
文字 (length)	SQL_CHAR
テキスト	SQL_LONGVARCHAR
実数	SQL_DECIMAL
整数	SQL_INTEGER
倍長整数	SQL_BIGINT
日付	SQL_DATE
時間	SQL_TIME
ブール	SQL_BIT
ピクチャ	SQL_BINARY
BLOB	SQL_BINARY

データソース上でコラムを作成する方法を確かめるために、特定のODBCドライバを調べましょう。4th Dimensionフィールドが義務的属性を持つなら、対応するコラムはNOT NULLオプションで作成されます。

インデックスは、各インデックスフィールドごとに作成されます。その名称は、4th Dimensionテーブル名、"$ ” サイン、フィールド名、"I_"によって前に置かれたテーブルの接尾辞、連続番号の鎖です。

例えば、EMPテーブルのSalフィールドのインデックスは、"I_EMP$SAL"と名づけられます。

4D フィールドがUnique属性を持つ場合、インデックスはUNIQUEオプションとともに作成されます。

OC Clone 4D tableは、制御が成功したら1を返し、ユーザがダイアログボックスの「Cancel」をクリックしたら0を返し、エラーが発生したら-1を返します。

例題

次の方法では、4Dデータベース上のテーブルを、ODBCサーバにコピーします。テーブルがサーバ上に既に存在するなら、自動的に置き換えます。

```
Login_ID:=OC Login dialog
If (Login_ID>0)
   For ($i;1;Count tables)
      rc:=OC Clone 4D table (Login_ID;$i;1)
   End for
End if
```

参照

OC Login

システム変数およびセット

ユーザが「OK」をクリックしたら、システム変数は1に設定されます。ユーザが「Cancel」をクリックしたら、システム変数OKは0に設定されます。

OC Clone ODBC table

OC Clone ODBC table (ログインID{;テーブル名}) → 整数

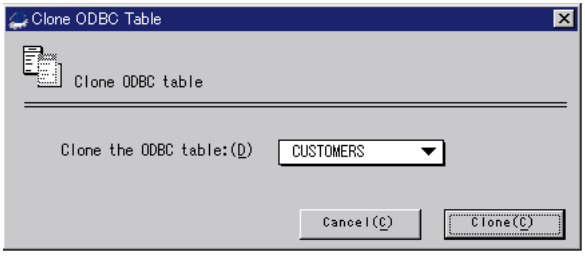
引数	タイプ		説明
ログインID	倍長整数	→	Login ID
テーブル名	文字列	→	Name of the 4D table
戻り値	整数	←	1=The operation is successful、 0=The user clicks Cancel、 -1=An error occurs

説明

OC Clone ODBC table関数は、ODBCテーブルと同等のフィールド定義で4th Dimensionテーブルを作成します。

＜ログインID＞は正しいログインでなければなりません。

■ 第2パラメータを承認しなかったら、**OC Clone ODBC table**は、Clone ODBC Tableダイアログボックスを表示します。



■ テーブル名を第2パラメータとして承認することで、ユーザに選択させないで承認された名前のテーブルを複製するコマンドを強要するでしょう。4Dテーブルは同じ名前を持っています。

新しいテーブルは、FileNという名称を持ち、Nはテーブル番号です。フィールド名は、ODBCデータソースのコラム名です。

次の法則で、タイプは変換されます：

4th Dimension	ODBC
文字 (length)	SQL_CHAR
テキスト	SQL_LONGVARCHAR
実数	SQL_DECIMAL
整数	SQL_INTEGER
倍長整数	SQL_BIGINT
日付	SQL_DATE
時間	SQL_TIME
ブール	SQL_BIT
ピクチャ	SQL_BINARY

データソースでコラムが作成される方法を確かめるため、特定のODBCドライバのドキュメントを調べましょう。

OC Clone ODBC tableは、制御が成功した時1を返し、ユーザがダイアログボックスの「Cancel」をクリックしたら0を返し、エラーが発生したら-1を返します。

ユーザが「Clone ODBC Table」ダイアログボックスで「Cancel」をクリックしたら、システム変数OKは、0に設定されます。

例題

次の方法で、データソースが見つかるすべてのテーブルを、局地的に複製します。

```
` OC CloneDB (CuursorID)
` CursorID : Longint, from OC Create cursor.

C_LONGINT ($curs)
$curs := $1
ARRAY STRING (15;t1;0)
ARRAY STRING (15;t2;0)
ARRAY STRING (60;tab_name;0)
ARRAY STRING (15;tab_type;0)
ARRAY STRING (15;t5;0)

OC GET TABLE LIST ($curs;"t1";"t2";"tab_name";"tab_type";"t5")
For ($i;1;Size of array(tab_type))
   If (tab_type{$i}="table")
      MESSAGE ("Clone table:"+tab_name{$i})
    $r:=OC Clone ODBC table (l;tab_name{$i})
   End if
End for
```

参照

なし

OC Excuse SQL

OC Excuse SQL (接続ID;SQLコマンド{;リミット{;配列リスト}})→ 整数

引数	タイプ		説明
接続ID	倍長整数	→	ログインIDもしくはカーソルID
SQLコマンド	テキスト	→	SQLコマンド
リミット	整数	→	結果ラインの最大番号
配列リスト	文字列配列	→	配列名配列
戻り値	整数	←	満たされた配列の数 エラー発生なら-1

説明

OC Excute SQL関数によって、SQLクエリを送信し、4th Dimension配列に結果をストアすることができます。

OC Excute SQL関数は、**OC Query exe**関数と似ています。データソースのSQLコマンドの送信、実行、および4D配列での要求結果の検索が可能です。目的配列にパラメータとして渡されるよう要求する**OC Query exe**と異なり、**OC Excute SQL**は目的配列名を含むテキストあるいは文字列配列を期待します。そのために、このコマンドで、22配列の制限は排除されます

＜接続ID＞は正しいログインIDもしくはカーソルIDでなければなりません。正しいログインIDを使用するなら、4D ODBCは、この機能の使用により、自動的にカーソルの開閉ができるでしょう。正しいカーソルIDを渡せば、この機能で特有のカーソルを使用できるでしょう。

＜SQLコマンド＞は実行するSQLクエリのテキストです。すべてのSQLコマンドは、その構文が正しい限り、受け入れられます。ODBCでサポートされたSQLの完全な説明に関しては、ODBCドライバのドキュメントを参照してください。

コマンドがデータを返しさえすれば、オプションのパラメータのlimitやarrayListは渡されねばなりません。リミットは、4th Dimensionに返す結果ラインの最大番号を指定します。limit=-1なら、**OC Excute SQL**はすべての結果をロードします。-1以外の数が指定されるなら、残りの列はロードされません。制限を設けて、後に残りの列をロードするには、第10章「OC下位レベル」で説明されるカーソルコマンドを使用しましょう。

＜配列リスト＞は、クエリの結果を検索するための配列名を含む、4th Dimension文字列配列です。

arrayList{n}と名づけられたオブジェクトは、存在するなら、コラムNからの結果を受け取ります。

実行中、エラーが発生したら、**OC Excute SQL**は-1を返し、それ以外の場合は、満たされる配列番号をあらわす0以上の値を返します。

例題

次の例では、"Sales"テーブルから3つのコラムを検索し、**OC Excute SQL**関数を使用して結果を3つの4D配列に配置します。目的の配列名はarArrays配列の要素として、関数に渡されます。

```
ARRAY STRING (30;arRep;0)
ARRAY STRING (30;arCust;0)
ARRAY STRING (30;arProduct;0)
ARRAY STRING (30;arArrays;3)
cur:=OC Create cursor (login)

arArrays{1}:="arRep"
arArrays{2}:="arCust"
arArrays{3}:="arProduct"

OC OPEN DEBUG WINDOW
$sql:="select REPID, CUSTID, PRODNAME from sales"
   $res:=OC Execute SQL (cur;$sql;-1;arArrays)
```

参照

なし

OC Login

OC Login ({ユーザ;{;パスワード{;データソース}}}) → 倍長整数

引数	タイプ		説明
ユーザ	文字列	→	ユーザ名
パスワード	文字列	→	パスワード
データソース	文字列	→	データソース名
戻り値	倍長整数	←	接続ID、エラーなら-1

説明

OC Login関数は、指定したログインパラメータを利用してODBCサーバに記録を出し、通信IDを返します（確認）。

＜ユーザ＞は使用するユーザの（任意の）名前です。＜ユーザ＞を指定しないなら、4D ODBCが前にデータソースダイアログボックスに接続に登録された名前を使います。セーブ名前チェックボックスがダイアログボックスで選ばれた場合に限り、ユーザ名は知られています。

＜パスワード＞はユーザのためにのパスワードです。パスワードを指定しないなら、4D ODBCが前にデータソースダイアログボックスに接続に登録されたパスワードを使います。セーブパスワードチェックボックスがダイアログボックスで選ばれても、パスワードがただ使われるだけです。

接続先のデータソース名です。特定のデータソースドライバのためにODBCセットアップで定義されるようなデータソース名を指定する必要があります。サーバを指定しないなら、4D ODBCがデータソースダイアログボックスに接続で選ばれた最後のサーバを使います。**OC Login**は正常終了なら接続IDを、エラーなら-1を返します。

例題

1. 次のステートメントは、マイクロソフトSQLサーバの中に、ログファイルに書きます。使用するユーザ名は「Mary」、パスワードは「toto」です。

```
Login_ID := OC Login ("mary";"toto";"MSSQLServer")
```

2. 次のステートメントはサーバを使ってログインします、ユーザ名とパスワードが「ODBCサーバー」ダイアログボックスに接続で選ばれて続きます。

```
Login_ID := OC Login
```

前の陳述で、接続IDはLogin_ID変数にしまっておかれます。この変数はその時ステートメントを実行して、そしてデータを検索する他の4D ODBCコマンドで、そしてサーバからのログアウトにOCログアウトで使われることができます。

参照

OC Login dialog、OC LOGOUT

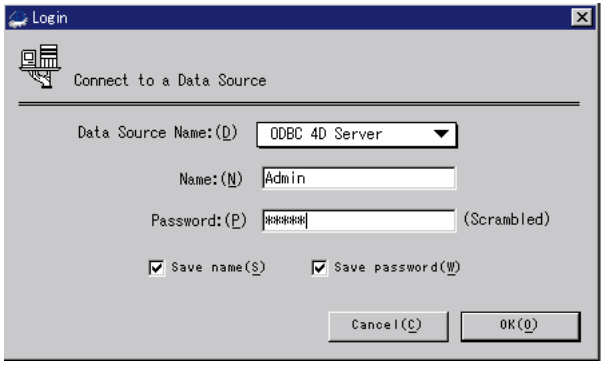
OC Login dialog

OC Login dialog → 倍長整数

引数	タイプ		説明
このコマンドには、引数はありません。			
戻り値	倍長整数	←	接続ID、 ユーザがダイアログのキャンセルを押したなら0、エラーが発生したのなら-1

説明

OC Login dialog関数は「データソース」ダイアログボックスに接続を見せて、そしてユーザにデータ源を選択して、さらにログイン名とパスワードを指定することができます。



ユーザがOKをクリックするなら、**OC Login dialog**がログインして、そして接続IDを返します。もしユーザがCancelをクリックするなら、**OC Login dialog**が0を返します。エラーが起きたなら**OC Login dialog**は-1を返します。

ユーザは「データソース」ポップアップメニューからデータソースを選択します。そしてそれは、データソースのために特定のドライバをインストールすることによって、ユーザのマシンの上にインストールされたデータソースをリストします。4D ODBCがそのリソースでユーザによって選択されたサーバの名前をセーブします。セーブ名チェックボックスが選ばれて、ユーザがセーブパスワードチェックボックスを選んだなら、パスワードであるなら、それは同じくユーザの名前をセーブします。

警告：セキュリティの重要性から、ユーザがパスワードをセーブしないことをお勧めします。ユーザがパスワードをセーブする場合は、ユーザの4D ODBCデータベースへのアクセスを持っていれば誰でもサーバに接続することができます。さらに、それがセーブされている間に、パスワードは暗号化されません。

参照

OC Login、OC LOGOUT

システム変数とセット

ユーザがダイアログボックスのOKをクリックしたなら、システム変数OKは1にセットされます。Cancelをクリックするなら0がセットされます。

エラー処理

エラーが起きたなら、**OC Login dialog**は-1を返します。

OC LOGOUT

OC LOGOUT (ログインID)

引数	タイプ		説明
ログインID	倍長整数	→	接続ID

説明

OC LOGOUTコマンドは指定された接続を終わらせます。

OC LOGOUTは**OC Login**あるいは**OC Login dialog**関数を使ったログイン時に検索される正当な接続IDであるに違いありません。

カーソルと接続の間に開かれたコンテキストはそれぞれ閉じられて、効力をなくされます。そして＜ログインID＞は無効になります。しかしながら、＜ログインID＞の値を使うことができないように、＜ログインID＞が接続を開いているかどうか決定するためにゼロに自動的にリセットされません。

ログアウト時、4D ODBCは現在のトランザクションを終了します。

4th Dimension終了時、4D ODBCサーバから自動的にログアウトします。

参照

OC Login、OC Login dialog

OC Query exec

OC Query exec (接続ID;SQLコマンド;リミット{;配列}{;配列2;...;配列N}) → 整数

引数	タイプ		説明
接続ID	倍長整数	→	ログインIDもしくはカーソルID
SQLコマンド	テキスト	→	SQLコマンド
リミット	整数	→	結果ラインの最大番号
配列	文字列	→	4D変数もしくは配列名（N <= 22）
戻り値	整数	←	返された列番号、 もしくはエラーが起きたなら-1

説明

OC Execute SQL関数はSQLのクエリとストアを送ることを許可します。そして4th Dimension配列を結果として生じます。

＜接続ID＞は有効なログイン、あるいはカーソルIDであるに違いありません。

有効なログインIDを使うなら、4DD ODBCがこの機能での使用のために自動的にカーソルを開いて、そして閉じるでしょう。有効なカーソルIDを通過させるなら、この機能は指定されたカーソルを使うでしょう。

＜SQLコマンド＞は実行するべきSQLの問合せのテキストです。文法が有効である限り、すべてのSQLコマンドは受け入れられます。ODBCによってサポートされるSQLを完全に記述するためにODBCドライバのドキュメントを参照してください。

＜リミット＞は4th Dimensionに戻るための結果ラインの最大の数を指定します。＜リミット＞が-1に等しいなら、**OC query exec**がすべての結果をロードします。-1以外の数が指定されるなら、残っていているローはロードされているはずはありません。限界を達成して、後に残っていているローをロードするために第10章「OC下位レベル」に記述されているカーソルのコマンドを使用してください。

配列1から配列22はクエリの結果を格納する4th Dimension配列名です。配列Nによって名前をつけられた配列は、もしそれが存在するなら、列Nからの結果を受けます。

OC Query execは結果を変換し、最終的に目的の配列のデータタイプに対応します。列が返した結果の番号に対応することは配列Nパラメータの数にとって必要ではありません。余分のパラメータあるいは列は無視されます。

データソースから返ってくる列の番号あるいはエラーが発生した場合は-1を返します。

OC Query execはSQLクエリを実行する最も単純な方法です。この機能は次の手順と等しいです：

1. **OC Create**カーソルでカーソルを作ってください。
2. **OC Set SQL in cursor**と一緒にSQLクエリを送ってください。
3. **OC Execute cursor**でクエリを実行してください。
4. **OC BIND**で結果を返すためにバインドを作ってください。
5. **OC Load row**で結果をロードしてください。
6. **OC DROP CURSOR**と一緒にカーソルを解放します。

例題

1. 次の方法はODBCデータソースにログインして、ローをDEPTテーブルに挿入して、そしてログアウトします：

```
Login_ID:=OC Login dialog
If (Login_ID>0)
   $sql:="INSERT INTO DEPT VALUES  (50,'PRODUCTS','PARIS')"
   $col:=OC Query exec (Login_ID;$sql;-1)
   OC LOGOUT (Login_ID)
End if
```

2. 次の方法はEMPテーブルから4D配列arempno、arenameとarsalにデータをロードします：

```
   $col:=OC Query exec (Login_ID;"SELECT empno, ename, sal FROM EMP";
                                                    -1;"arEmpno";"arEName";"arSal")
```

参照

OC Create Cursor、OC Execute SQL

OC下位レベル　10

この章のコマンド群は4th DimensionとODBCデータソースの間のコミュニケーションを管理することができます。このコミュニケーションを管理する下位レベルのコマンドを使う場合、ODBCデータソース上でデータを選択、挿入、更新、及び削除するために用いるSQLステートメントを指定しなければなりません。

この章の下位レベルのコマンドを使って、次のようなことが可能となります。

- カーソルの作成（**OC Create cursor**）
- SQLステートメントをカーソルに置く（**OC Set SQL in cursor**）
- 4th Dimensionオブジェクトとデータソースオブジェクトの間をバインドする（**OC Bind**）
- カーソルの実行（**OC Execute cursor**、**OC Execute direct cursor**）
- ローの結果のロード（**OC Load row**、**OC More results**、**OC CANCEL LOADING**）
- カーソルについての情報の検索（**OC Number rows processed**、**OC Number of columns**、**OC Describe column**、**OC Column attributes**）
- 使用パラメータマーカー（**OC Bind parameter**、**OC Describe parameter**、**OC Number of parameters**）
- カーソル名の割り当てと検索（**OC SET CURSOR NAME**、**OC Get cursor name**）
- カーソルのクローズ（**OC DROP CURSOR**）

OC Bind

OC Bind (カーソルID;参照番号;4Dオブジェクト) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
参照番号	整数	→	参照番号
4Dオブジェクト	文字列	→	4D変数名またはフィールド名
戻り値	整数	←	1=正常終了 -1=エラー発生

説明

OC BindコマンドはSQL選択ステートメントからの結果列と4th Dimensionオブジェクト（フィールド、変数、配列）の間のリファレンスを定義します。参照されたSQLステートメントは**OC SET SQL in cursor**または**OC Execute direct cursor**のいずれかが使用されます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜参照番号＞はSQLステートメントで参照されるSQL列のオーダー番号です。

＜4Dオブジェクト＞は参照番号によって参照された列の値を格納するフィールド、変数、配列の名前です。

注：4Dオブジェクトの名前は文字列（引用文の中で）、例えば"arArray1"として渡されます。さらに変数や配列を使う場合は適切なコンパイラ指令、例えばARRAY STRING (30;arArray1;0)を使って前もって定義しておく必要があります。

OC Bindは正常終了ならば1、エラーが発生したならば-1を返します。

例題

1. 次の方法は、著者テーブルから4th Dimension変数までコラムをバインドし、サーバから1つのローをロードします。

```
C_STRING (30;var1)
C_STRING (30;var2)
C_STRING (30;var3)
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind (cursor_id;1;"var1")
$res:=OC Bind (cursor_id;2;"var2")
$res:=OC Bind (cursor_id;3;"var3")

$res:=OC Execute cursor (cursor_id)
$res:=OC Load row (cursor_id)
```

2. 次の方法は、著者テーブルからサーバから多数のコラムをロードするため4th Dimension 配列までコラムをバインドします。

```
ARRAY STRING (30;ar1;0)
ARRAY STRING (30;ar2;0)
ARRAY INTEGER (ar3;0)
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
$res:=OC Bind (cursor_id;1;"ar1")
$res:=OC Bind (cursor_id;2;"ar2")
$res:=OC Bind (cursor_id;3;"ar3")

$res:=OC Execute cursor (cursor_id)
$res:=OC Load row (cursor_id)
While ($res=1)
   $res:=OC Load row (cursor_id)
End while
```

参照

OC Execute cursor、OC Execute direct cursor、OC SET SQL in cursor

OC Bind parameter

OC Bind parameter (カーソルID;参照番号;4Dオブジェクト;IOタイプ{;転換}) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
参照番号	整数	→	参照番号
4Dオブジェクト	文字列	→	4D変数および4Dフィールド名
IOタイプ	整数	→	I／Oタイプ
転換	整数	→	「見通しがきかない」変換のための1
戻り値	整数	←	1=正常終了、-1=エラー発生

説明

OC Bindパラメータ機能は動的なSQLステートメントに4th Dimensionオブジェクトをバインドします。SQLステートメントの中にパラメータマーカーを埋め込むことによって、動的なSQLステートメントが組み立てられます。SQLパラメータマーカーはホスト変数として使われていたクエスチョンマーク（?）です。

＜カーソルID＞は前もって作成されたカーソルのIDでなければなりません。

＜参照番号＞はSQLステートメントで参照されるパラメータのオーダー番号です。

＜4Dオブジェクト＞は参照番号によって参照されたコラムの値を格納するフィールドまたは変数の名前です。

＜IOタイプ＞は次のテーブルをベースにするI／Oタイプの参照コードです：

I／Oタイプ	コード
入力	1
入力・出力	2
コラム結果	3
出力	4
戻り値	5

「自動」変換選択をアクティブ化するなら、変換で1をパスするべきです。ODBCにパラメータを渡す場合、4D ODBCが4D内部のデータ表現からODBCによって格納できるデータ表現まで、それを変換しなければなりません。

そうするために、SQL_API_SQLDESCRIBEPARAMとして知られているAPI呼び出しを発行することによって、4D ODBCがODBCドライバにパラメータの性質（あるいはデータタイプ）を「求めよう」とします。若干のドライバが、4D ODBCが何の変換を操作するべきか見つけだすことを不可能にして、この呼び出しをサポートしません。これらのドライバの欠点を克服するために、4D ODBCは盲目的な変換を実装しています。必要とさ

れるなら、4D ODBCが4Dオブジェクトのデータタイプに基づいてパラメータ値を最も近いODBCデータタイプに変換するでしょう。例（2）変換パラメータの使用に関する詳細な情報を参照してください。

自動変換は次のテーブルに基づいています：

4D**オブジェクト**	ODBC **データタイプ**
C_STRING	SQL_CHAR
C_TEXT	SQL_CHAR
C_BOOLEA	SQL_CHAR
C_DATE	SQL_TIMESTAMP
C_TIME	SQL_TIMESTAMP
C_REAL	SQL_DOUBLE
C_LONGINT	SQL_INTEGER
C_INTEGER	SQL_SMALLINT
C_PICTURE	SQL_BINARY

OC Bind parameterは正常終了ならば1、エラーが発生したならば-1を返します。

例題

1. 次の方法はSQLを実行するためのインプットパラメータが文を挿入する時、配列からの値を使います。

```
C_STRING (5;vstate)
ARRAY STRING (30;arState;10)
arState{1}:="CA"
arState{2}:="NY"
arState{3}:="MO"
arState{4}:="CT"
arState{5}:="FL"
arState{6}:="AZ"
arState{7}:="AR"
arState{8}:="NV"
arState{9}:="WA"
arState{10}:="TX"
$sql:="insert into states values (?)"
$res:=OC Set SQL in cursor (process;$sql)
$res:=OC Bind parameter (process;1;"vState";1)
For ($i;1;10)
   vState:=arState{$i}
   $res:=OC Execute cursor (process)
End for
```

次のテーブルはどこにパラメータマーカーを使うことができないか説明します：

位置	**例**
選択リスト	SELECT ? from TABLE
両方とも比較での表現	? >= ?
両方を2進オペレーターのオペランドとして	? * ?
両方のBETWEEN節の最初と2番目の演算数として	? BETWEEN ? and COL3
両方のBETWEEN節の最初と3番目の演算数として	? BETWEEN Col4 and ?
IN節で式と最初の値両方として	? IN (?, 10, 15, 20)
プラスあるいはマイナスオペランドとして	Col = -?
セット機能のアーギュメントとして	SELECT CUSTOMER_NAME、 AVG(?) FROM CUSTOMER GROUP BY CUSTOMER _NAME

2. ストアードプロシージャ、sp_getBookTitleが2つのパラメータを受け入れます。author_idは整数、book_idは文字列です。

```
C_STRING (30;$Author_id)
C_STRING (30;$Book_id)
C_STRING (30;$unused)
$Author_id:="1245"
$Book_id:="GMN9422"

$DescribeSupported: = OC Get function (Login_ID; 58; "$unused")

If ($DescribeSupported=1)
      ` デフォルトのデータタイプ変換メカニズムを使用
   $r:=OC Bind parameter (CursID;1;"$author_id";1;0)
      ` $Author_idでのC_STRING値はパラメータの記述に従って整数に変換されます。
      ` この呼び出しについて働くために、ODBCドライバは
      ` SQL_API_SQLDESCRIBEPARAMコールをサポートしなくてはなりません。
      ` 大部分のドライバはこの呼び出しをサポートしていません。
Else
   $r:=OC Bind parameter (CursID;1;"$Author_id ";1;1)
      ` 最も近くのSQL datatypeに変換されて$Author_idでの
      ` C_STRING値がそうであるデータタイプ変換メカニズムを無効に
      ` してください：これがもっとずっと多くのドライバで
      ` 働かせるでしょうが、VARCHARはそれほど柔軟ではありません。
End if
```

ストアードプロシージャがその定義で安定していることが既知の場合、**OC Bind**パラメータにそれを渡す前に、データを前もって変換するために4D変換メカニズムを使うことは賢明です。

前の例は書き直すことができます：

```
C_STRING (30;$Author_id)
C_INTEGER ($NumAuthor_id)
C_STRING (30;$Book_id)
C_STRING (30;$unused)

$Author_id:="1245"
$Book_id:="GMN9422"

$DescribeSupported: = OC Get function (Login_ID; 58; "$unused")

If ($DescribeSupported=1)
      ` デフォルトのデータタイプ変換メカニズムを使用
   $r:=OC Bind parameter (CursID;1;"$author_id";1;0)
      ` $Author_idでのC_STRING値はパラメータの記述に従って整数に変換されます。
      ` この呼び出しについて働くために、ODBCドライバは
      ` SQL_API_SQLDESCRIBEPARAMコールをサポートしなくてはなりません。
      ` 大部分のドライバはこの呼び出しをサポートしていません。
Else
   $NumAuthor_id := Num ($Author_id)
      ` データタイプ変換メカニズムを無効にします
   $r:=OC Bind parameter (CursID;1;"$NumAuthor_id ";1;1)
      ` $NumAuthor_idでのC_INTEGER値は最も近くのSQL datatypeに
      ` 変換されます。INTEGERはもっとずっと多くのドライバと一緒に
      ` 動作しますが、それほど柔軟ではありません。
End if
```

参照

OC Execute cursor、OC Set SQL in cursorn

OC CANCEL LOADING

OC CANCEL LOADING (カーソルID)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID

説明

OC CANCEL LOADINGコマンドは現在のクエリと関連付けられたどんな結果セットでも破棄します。このコマンドは新しい問合せによる使用のためにカーソルをリセットすることができます。

参照

OC Load Row、OC More Results

OC Column attributes

OC Column attributes (カーソルID；列番号；属性) → 文字列

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
列番号	整数	→	列の番号
属性	整数	→	属性番号
戻り値	文字列	←	属性の値、またはエラー発生の場合空文字列

説明

OC Column attributes関数は**OD Set SQL in cursor**コマンドを実行した後に列属性の情報を検索することができます。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

＜列番号＞は特性がリクエストされる列のSQLステートメントでオーダー番号を指定します。

属性は求められた属性のID番号を指定します。次の値は属性のためのものです：

属性	ID
SQL_COLUMN_COUNT	0
SQL_COLUMN_NAME	1
SQL_COLUMN_TYPE	2
SQL_COLUMN_LENGTH	3
SQL_COLUMN_PRECISION	4
SQL_COLUMN_SCALE	5
SQL_COLUMN_DISPLAY_SIZE	6
SQL_COLUMN_NULLABLE	7
SQL_COLUMN_UNSIGNED	8
SQL_COLUMN_MONEY	9
SQL_COLUMN_UPDATABLE	10
SQL_COLUMN_AUTO_INCREMENT	11
SQL_COLUMN_CASE_SENSITIVE	12
SQL_COLUMN_SEARCHABLE	13
SQL_COLUMN_TYPE_NAME	14
SQL_COLUMN_TABLE_NAME	15
SQL_COLUMN_OWNER_NAME	16
SQL_COLUMN_QUALIFIER_NAME	17
SQL_COLUMN_LABEL	18

OC Column attributeはリクエストされた情報か、エラーの場合は空文字列を返します。

例題

次の方法は警告がクエリで最初の列の大きさを記述しているのを表示します。

```
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in cursor (cursor_id;$sql)
$info:=OC Column attributes (cursor_id,1; 3)
ALERT ("This column has a length of "+String ($info))
```

参照

OC EXECUTE CURSOR、OC Set SQL in cursor

OC Create cursor

OC Create cursor (ログインID) → 倍長整数

引数	タイプ		説明
ログインID	倍長整数	→	ログインID
戻り値	倍長整数	←	カーソルID、エラーが発生した場合は-1

説明

OC Create cursor関数は指定された通信のカーソルを作成し、開くことができます。

カーソルは、クエリから生じた各列が、一度にひとつ処理されるメカニズムです。カーソルは、ODBCデータソース上のプロセスと見なせます。作成した各カーソルは、あとでODBCデータソース上のデータを選択、挿入、更新、および削除するSQLステートメントと通信します。複数のカーソルを同時に作成、利用できますが、過多のカーソル利用はパフォーマンスに影響を与えます。

＜ログインID＞は有効なログインIDでなければなりません。

OC Create cursorは正常終了ならカーソルIDを、エラーが発生したのなら-1を返します。

参照

OC DROP CURSOR、OC Execute cursor、OC Execute direct cursor、OC Login、OC Login dialog

OC Describe column

OC Describe column (カーソルID;列番号;列名;列タイプ)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
列番号	整数	→	列の番号
列名	文字列	←	4D文字列変数
列タイプ	整数	←	4D整数変数
戻り値	整数	←	1=正常終了、-1=エラー発生

説明

OC Describe columnコマンドはSQLクエリの前にカーソルと関連付けられた列の結果の名前とタイプを検索します。**OC Set SQL in cursor**を用いて、SQLステートメントをカーソルに設定した後、このコマンドを実行できます。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

＜列番号＞は特性がリクエストされる列のSQLステートメントでオーダー番号を指定します。

＜列名＞は関数が呼び出された後、列名を含んでいる4D変数です。

＜列タイプ＞は関数が呼び出された後、列タイプの値を含んでいる4D変数（整数）です。それぞれのデータタイプのコードです：

SQL_CHAR	1	SQL_TIMESTAMP	11
SQL_NUMERIC	2	SQL_VARCHAR	12
SQL_DECIMAL	3	SQL_LONGVARCHAR	-1
SQL_INTEGER	4	SQL_BINARY	-2
SQL_SMALLINT	5	SQL_VARBINARY	-3
SQL_FLOAT	6	SQL_LONGVARBINARY	-4
SQL_REAL	7	SQL_BIGINT	5
SQL_DOUBLE	8	SQL_TINYINT	-6
SQL_DATE	9	SQL_BIT	-7
SQL_TIME	10		

参照

OC EXECUTE CURSOR、OC Set SQL in cursor

OC Describe parameter

OC Describe parameter (カーソルID,参照番号,4Dオブジェクト) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
参照番号	整数	→	参照番号
4Dオブジェクト	整数	←	パラメータのデータタイプ説明
戻り値	整数	←	1=正常終了、-1=エラー発生

説明

OC Describe parameter関数はカーソルと関連付けられた指定されたパラメータのデータタイプの記述を得ます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜参照番号＞はSQLステートメントで参照されるパラメータのオーダー番号です。

＜4Dオブジェクト＞は参照番号によって参照されたパラメータのデータタイプ記述を含む4th Dimensionの整数変数名です。テーブルは**OC Describe column**コマンドのためのそれぞれのデータタイプのコードを提供します。

OC Describe parameterは正常終了なら1を、エラーが発生したのなら-1を返します。

参照

OC Bind parameter、**OC Number of parameters**

OC DROP CURSOR

OC DROP CURSOR (カーソルID)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID

説明

OC DROP CURSORコマンドは**OC Create cursor**が以前に作成したカーソルによって使われたメモリを解放します。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

例題

次の例は前に開かれたカーソルをドロップします。

```
$res:=OC DROP CURSOR (cursor_id)
```

参照

なし

OC Execute cursor

OC Execute cursor (カーソルID) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	整数	←	1=正常終了、-1=エラー発生

説明

OC Execute cursorコマンドはODBCデータソース上でカーソルに関連付けられたSQLステートメントを実行します。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

実行されているSQLステートメントがSELECT... typeのクエリであるなら、長期間すべての結果としてローが**OC Load row**によってロードされなかった場合、あるいは**OC CANCEL LOADING**が呼び出されなかった場合、**OC Execute cursor**がカーソルをアクティブ化します。

OC Execute cursorは正常終了なら1を、エラーが発生したのなら-1を返します。

参照
OC CANCEL LOADING、**OC Execute direct cursor**、**OC Set SQL in cursor**

OC Execute direct cursor

OC Execute direct cursor (カーソルID,SQLコマンド) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
SQLコマンド	テキスト	→	SQLコマンドの実行
戻り値	整数	←	1=正常終了、-1=エラー発生

説明
OC Execute direct cursorコマンドが送信して、ODBCデータソースにSQLステートメントを実行します。

OC Execute direct cursorは**OC Set SQL in cursor**と**OC Execute cursor**の両方を呼び出すことについては同等です。

間に合わせのSQLステートメントを作って、ただ一度呼ばれるであろうそのステートメントを実行することに対して、**OC Execute direct cursor**は有用です。実行成功後、DBMSはSQLステートメントを実行するために使ったどんな最適化情報でも捨てます。同じSQLステートメントの繰り返し実行が必要なとき、**OC Execute cursor OC**と**OC Set SQL in cursor**のコンビネーションを使ってください。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜SQLコマンド＞は実行するSQLクエリのテキストです。それらの文法が有効である限り、すべてのSQLコマンドは受け入れられます。ODBCによってサポートされるように、SQLの完全な解説のためにODBCドライバのドキュメントを参照してください。

OC Execute direct cursorは正常終了なら1を、エラーが発生したのなら-1を返します。

参照
OC CANCEL LOADING、**OC Execute cursor**、**OC Set SQL in cursor**

OC Get cursor name

OC Get cursor name (カーソルID) → 文字列

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	文字列	←	カーソル名

説明

OC Get cursor nameコマンドはカーソルの名前を返します。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

参照

OC Create cursor、OC SET CURSOR NAME

OC Load row

OC Load row (カーソルID) → 倍長整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	倍長整数	←	1=追加のローが差し迫っています。 0=これ以上のローはありません。 -1=エラーの発生

説明

OC Load row関数はSQLクエリが**OC Execute cursor**と**OC Execute direct cursor**を使って実行し、結果として生じたローをロードします。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

OC Load rowが（カーソルの場所に基づいた）ODBCデータソース上に次のローをフェッチして、そしてその値を**OC Bind**関数で指定した4th Dimensiomオブジェクトに割り当てます。

4th Dimensionフィールドをバインドしたのなら、**OC Load row**はメモリ上のレコードに値を割り当てますがそれをセーブしません。**SAVE RECORD**コマンドは呼び出されません。手続きによってレコードを作って、そしてセーブしなくてはなりません。

4th Dimension配列をバインドしたのなら、**OC Load row**が配列に追加の要素を加えて、その要素に値をロードします。

OC Load rowは差し迫っている追加のローがあるなら1を、それ以上のローがないなら0、エラーが起こるなら-1を返します。

例題

変数と配列にロードすることについての例のために**OC Bind**の項でリストされている例を見てください。

参照

なし

OC More results

OC More results (カーソルID) → 倍長整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	倍長整数	←	1=正常終了、-1=エラー発生

説明

OC More results関数がクエリで利用可能なもっと多くの結果があるか否かにかかわらず決定して、もしそうであるなら結果を処理し始めます。

この機能は結果あるいは多数の結果を返す他のいかなるDBMS関数のその戻り値が多数のセットでも達成する、ストアードプロシージャのような、バッチSQLステートメントの使用をサポートするために使われます。

アプリケーションは、次のものに動く前に、1つのセットですべての結果を検索するように要求されません。**OC More results**が呼び出されたとき、達成された前の結果は破棄されます。したがって、達成された現在の結果からすべての望ましいローをロードした後で、この機能を使ってください。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

OC More resultsは正常終了なら1を、エラーが発生したのなら-1を返します。

例題

次の方法はSQLサーバのストアードプロシージャを実行して、すべての結果セットからの最初の3つの縦列の値を返して、そして警告ボックスを表示します。

```
$sql:="exec sp_help titles"
$res:=OC Set SQL in cursor (cursor_id;$sql) `bind to arrays
$res:=OC Bind (cursor_id;1;"ar1")
$res:=OC Bind (cursor_id;2;"ar2")
$res:=OC Bind (cursor_id;3;"ar3")
$res:=OC Execute cursor (cursor_id)
$count:=1
$more:=1
While ($more=1)
   $res:=1
   While ($res=1)
      $res:=OC Load row (cursor_id)
   End while
   $res:=OC Execute cursor (cursor_id)
   $more:=OC More results (cursor_id)
   $count:=1+$more
End while
ALERT (String ($count)+" set of results returned")
```

参照

OC Execute cursor、OC Set SQL in cursor

OC Number of columns

OC Number of columns (カーソルID) → 倍長整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	倍長整数	←	結果列の番号、エラーが発生したなら-1

説明

OC Number of columns関数はクエリによって結果列の番号を返します。**OC Set SQL in cursor**コマンドが正常に終了した後いつでも、この機能を使うことができます。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。**OC Number of columns**がエラーを発生した場合、-1を返します。

例題

次の方法はサーバにSQLステートメントを送って、問合せでコラムの数を示して警告を提供します。

```
$sql:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in cursor (cursor_id;$sql)
$numcols:=OC Number of columns (cursor_id)
ALERT ("This query has "+String($numcols)+" columns.")
```

参照

なし

OC Number of parameters

OC Number of parameters (カーソルID) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	整数	←	パラメータの番号、 エラーが発生したなら-1

説明

OC Number of parameters関数はカーソルと関連付けられたパラメータの番号を返します。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

OC Number of parametersがエラーを発生した場合、-1を返します。

参照

OC Bind parameter、OC Describe parameter

OC Number rows processed

OC Number rows processed (カーソルID) → 倍長整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
戻り値	倍長整数	←	処理されたローの番号、 エラーが発生したなら-1

説明

挿入、更新、あるいは削除コマンドを使ってSQLステートメントを実行したカーソルに適用される時、機能が番号を返す**OC Number rows processed**はローを差し込んで、修正するか、あるいは削除しました。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

OC Number rows processedがエラーを発生した場合、-1を返します。

例題

次の方法は店ID6380と結び付けられるテーブル販売からすべてのローを削除して、いくつのローがもたらされたか記述している警告を提供します。

```
$sql:="delete from sales where stor_id = '6380'"
$result:=OC Set SQL in Cursor (cursor_id;$sql)
$result:=OC Execute cursor (cursor_id)
$result:=OC Load row (cursor_id)
$numrows:=OC Number rows processed (cursor_id)
ALERT (String ($numrows)+" sales records deleted.")
```

参照

OC EXECUTE CURSOR、OC Set SQL in cursor

OC SET CURSOR NAME

OC SET CURSOR NAME (カーソルID;カーソル名)

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
カーソル名	文字列	→	カーソル名

説明

OC SET CURSOR NAMEコマンドはカーソルの名前をつけます。カーソル名が置かれた更新／削除を行うために要求されます。更新／削除がいずれのローが行動されるはずであるか決定する予定になっている結果におけるカーソルの現在のポジションを使います。SELECT FOR UPDATEステートメントが実行されるときはいつでも、カーソル名がODBCに自動的に内部であるように創造されますが、アプリケーションにそれを内部で有意義にするためにそれ自身カーソルに名前をつけることができます。

＜カーソルID＞は前もって作成されたカーソルのIDでなくてはなりません。

＜カーソル名＞はカーソルIDに割り当てられる名前です。

参照

OC Create cursor、OC Get cursor name

OC Set SQL in cursor

OC Set SQL in cursor (カーソルID;SQLコマンド) → 整数

引数	タイプ		説明
カーソルID	倍長整数	→	カーソルID
SQLコマンド	テキスト	→	SQLコマンドの実行
戻り値	整数	←	1=正常終了、-1=エラー発生

説明

OC Set SQL in cursor関数がSQLステートメントをカーソルと関連付けます。コマンドはODBCデータソースにステートメントを送ります。**OC EXECUTE CURSOR**を呼び出す時、ステートメントは実行されます。

＜カーソルID＞は使われていないカーソル、且つ前もって作成されているIDでなければなりません。

＜SQLコマンド＞は実行するべきSQLの問合せのテキストです。文法が有効である限り、すべてのSQLコマンドは受け入れられます。ODBCによってサポートされるように、SQLの完全な解説のためにODBCドライバのドキュメントを参照してください。

SQLステートメントの中でパラメータを埋め込むことによって、ダイナミックなSQLステートメントを組み立てることができます。より多くのSQLステートメントでパラメータを使うことについての情報のために、**OC Bind parameter**を見てください。

例題

次の方法は［従業員］名前フィールドとvFirstName変数で値を使っているODBCデータソースの上にローを作ります。

```
$sql:="insert into authors (lname, fname) values ('smith','john')"
$res:=OC Set SQL in cursor (Cursor_ID; $sql)
OC EXECUTE CURSOR (Cursor_ID))
```

参照

OC EXECUTE CURSOR

コンテキストコマンドの利用

コンテキストは一旦定義されれば、SQLステートメントを使用しなくても、ODBCデータを4th Dimensionにロードし、ODBCデータソース上の情報を情報を更新することができます。

この章のコンテキストコマンドによって、次のことができます。

- コンテキストの有効化（**OC Activate context**）
- 4th DimensionへのODBC列のロード（**OC Previous in context**、**OC Next in context**、**OC Goto in context**、**OC Load rows context**、**OC First in context**、**OC Last in context**）
- データの更新、挿入、削除（**OC Update in context**、**OC Insert in context**、**OC Delete in context**）
- コンテキストについての情報検索（**OC Get clause in context**）
- コンテキストの無効化（**OC DEACTIVATE CONTEXT**）

ダイアログボックスを使用するか、手続き文のセットを実行することによって、コンテキストを作成できます。どちらかの方法に関する詳細は、第3章「コンテキストの利用」を参照してください。

OC Activate context

OC Activate context (ログインID;コンテキストID) → 整数

引数	タイプ		説明
ログインID	倍長整数	→	接続ID
コンテキストID	倍長整数	→	コンテキストID
戻り値	整数	←	1=正常終了しました -1=エラーが発生しました

説明

OC Activate context機能によって、**OC Create context**や**OC Create context dialog**で作成されたコンテキスト、またはコンテキスト定義をともなう画像やファイルのロードによるコンテキストを有効化します。

＜ログインID＞は正しい通信IDでなければなりません。コンテキストIDはあらかじめ作成され、無効なコンテキストIDでなければなりません。

OC Activate contextは、エラー時には-1を返し、それ以外のときは1を返します。

OC Activate contextはカーソルを作成し、クエリの結果と一致する列のセットを定義します。それはいかなる結果列も検索しません。結果列を表示するには、**OC Load rows context**や**OC Previous in context**を使用します。

最初の結果列を検索するために、**OC Load rows context**を使用できます。

コンテキストが有効な限り、**OC ADD TO CONTEXT**、**OC EDIT CLAUSES IN CONTEXT**、または**OC SET CLAUSE IN CONTEXT**を用いて、定義を修正することはできません。

OC DEACTIVATE CONTEXTを呼び出して、コンテキストを無効化するまでは、コンテキストは有効な状態です。

参照

OC Create context、**OC DEACTIVATE CONTEXT**、**OC Load rows context**、**OC Previous in context**

OC DEACTIVATE CONTEXT

OC DEACTIVATE CONTEXT (context_ID)

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID

説明

OC DEACTIVATE CONTEXTコマンドは**OC Activate context**によってあらかじめ有効化されたコンテキストを無効化します。

＜コンテキストID＞はあらかじめ作成され、有効にされたコンテキストIDでなければなりません。

OC SET CLAUSE IN CONTEXTコマンドの使用によって、コンテキストは修正されます。

参照

OC Activate context

OC Delete in context

OC Delete in context (コンテキストID{;インデックス}) → 整数

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
インデックス	インデックス	→	テーブルインデックス
戻り値	整数	←	1=正常終了 0=ローは存在しません -1=エラーが発生しました

説明

OC Delete in context機能は、ODBCサーバ上のコンテキストの現在の列を削除します。＜コンテキストID＞はあらかじめ作成され、有効なコンテキストIDでなければなりません。

＜インデックス＞が指定されていないなら、**OC Delete in context**は現在の列番号と一致する配列要素を使用して、新しい列を削除します。列番号が配列のサイズより大きいなら、**OC Delete in context**は-1を返します。

現在の列が削除された時、次の列が存在するならそれは現在の列になり、次の列が存在しないなら、前の列が現在の列になります。削除後に列が存在しないなら、現在の列はもはや定義されません。

列番号と現在の列番号は、**OC Delete in context**が呼び出された後、変更されないで、ODBCサーバ上の列は物理的に削除されます。

OC Load rows context、**OC Previous in context**、**OC Goto in context**、**OC Last in context**または**OC First in context**で、前に削除された列をロードしようとするなら、4D ODBCはエラーを返します。

OC Delete in contextは"DELETE ...WHERE"のようなSQLクエリを送信します。

OC Delete in contextは正常動作したら1、RowNumber列が存在しないなら0、エラーが発生したら-1を返します。

参照

OC Insert in context、**OC Update in context**

OC First in context

OC First in context (context_ID) → Integer

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
戻り値	整数	←	1=正常終了 0=始めのローを削除しました -1=エラーが発生しました

説明

OC First in context機能は、コンテキストの最初の列をロードします。

＜コンテキストID＞はあらかじめ作成され、有効なコンテキストIDでなければなりません。

OC First in contextは、現在の列と関連のある4th Dimensionオブジェクトを更新します。フィールドをともなうバインドのために、現在のレコードはセーブされずに更新されます。現在のレコードが存在しないなら、**OC First in context**は一致するバインドを無視します。

OC First in contextは、エラーが発生しない時は1、最初の列が削除されたときは0、エラーが発生したときは-1を返します。

参照

OC Last in context、OC Previous in context

OC Goto in context

OC Goto in context (context_ID; rowNumber) → Integer

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
ロー番号	倍長整数	→	コンテキスト中のローの数
戻り値	整数	←	1=正常終了 0=ローは存在しません -1=エラーが発生しました

説明

OC Goto in context機能は、ロー番号列をロードします。ロー番号列は現在の列になります。

＜コンテキストID＞はあらかじめ作成され、有効なコンテキストIDでなければなりません。

＜ロー番号＞がコンテキストの列番号より大きいか、1より小さいなら、**OC Goto in context**は何もせずに、0を返します。

OC Goto in contextは、新規現在列で4th Dimensionオブジェクトを更新します。

フィールドをともなうバインドのために、現在のレコードはセーブされずに更新されます。現在のレコードが存在しないなら、**OC Goto in context**は一致するバインドを無視します。

OC Goto in contextは、正常動作なら1、ロー番号列が存在しないなら0、エラーが発生したら-1を返します。

参照

OC Next in context、OC Previous in context

OC Insert in context

OC Insert in context (コンテキストID{;インデックス}) → 整数

引数	タイプ		説明
コンテキストID	倍長整数	→	カーソルID
インデックス	整数	→	インデックス配列
戻り値	整数	←	1=正常終了、-1=エラーが発生しました

説明

OC Insert in contextは、コンテキストバインドに指定された4th Dimensionオブジェクトを使用して、ODBCサーバ上に新規列を作成します。

＜コンテキストID＞はあらかじめ作成され、有効なコンテキストIDでなければなりません。

インデックスは配列をともなうバインドのみ適用されます。インデックスが指定されていないならば、**OC Insert in context**は、現在の列番号と一致する配列要素を用いて、新しい列を作成します。列番号が配列のサイズより大きいならば、**OC Insert in context**は-1を返します。

インデックスが指定されたなら、**OC Insert in context**はインデックスindexの配列の行を用いて、新しい列を作成します。インデックスが配列のサイズより大きいならば、**OC Insert in context**は-1を返します。**OC Insert in context**は、INSERTタイプのSQLクエリを送信します。

OC Insert in contextの実行は、現在の列を修正せず、コンテキストに新しい列を配置しません。

OC Insert in contextは、正常に実行されたなら1を返し、エラーが発生したら-1を返します。

参照

OC Delete in context、OC Update in context

OC Last in context

OC Last in context (コンテキストID) → 整数

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
戻り値	整数	←	1=正常終了しました 0=最後のローを削除しました -1=エラーが発生しました

説明

OC Last in context機能は、コンテキストの最終列をロードします。

＜コンテキストID＞はあらかじめ作成され、有効なコンテキストIDでなければなりません。

OC Last in contextは現在の列と関連付けられた4th Dimensionオブジェクトを更新します。フィールドをともなうバインドのため、現在のレコードはセーブされずに更新されます。現在のレコードが存在しないなら、**OC Last in context**は一致したバインドを無視します。

OC Last in contextは、エラーが発生しないなら1、最終列が削除されたら 0 、エラーが発生したら-1を返します。

参照

OC First in context、OC Previous in context

OC Load rows context

OC Load rows context (コンテキストID{;限界}) → 倍長整数

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
限界	倍長整数	→	最大ロードレコード数
戻り値	倍長整数	←	ロードされたローの数、 エラーが発生したなら-1

説明

OC Load rows context機能は、コンテキストによって定義されたすべての列を、コンテキストバインドで指定された4th Dimensionオブジェクトにロードします。

＜コンテキストID＞はあらかじめ作成され、有効なIDでなければなりません。

＜限界＞によって、ロードすべき列の最大値を示すことができます。

OC Load rows contextは必要なだけの4th Dimensionレコード、あるいは配列の要素を作成します。限界=1で、コンテキストがフィールドとのバインドを含むなら、**OC Load rows context**は、セーブせずに、結果行をロードするためにメモリにレコードを作成します（**SAVE RECORD**コマンドは呼び出されません）。

結果が4th Dimensionレコードにロードされるとき、現在のレコード選択は、4D ODBCで作成されたレコードに設定されます。

OC Load rows contextは、ロードされた列番号、またはエラー発生時に-1を返します。

参照

OC Activate context

OC Next in context

OC Next in context (コンテキストID) → 整数

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
戻り値	整数	←	1=正常終了 0=次のローが存在しません -1=エラーが発生しました

説明

OC Next in context機能は、現在列に先立つ結果列をロードします。現在列が未定義なら、**OC Activate context**の後のように、**OC Next in context**は何もしません。この機能は、**OC load rows context**(Context_ID;1) による実行のために使用されます。

＜コンテキストID＞は、あらかじめ作成され有効であるコンテキストIDでなければなりません。

OC Next in contextは現在の列と関連付けられた4th Dimensionオブジェクトを更新します。フィールドをともなうバインドのため、現在のレコードはセーブされずに更新されます。現在のレコードが存在しないなら、**OC Next in context**は一致するバインドを無視します。

OC Next in contextは、エラーが発生していない時は1、次の列が存在しない時は0、エラー発生時には-1を返します。

参照

OC Previous in context

OC Previous in context

OC Previous in context (context_ID) → Integer

引数	タイプ		説明
コンテキストID	倍長整数	→	Context ID
戻り値	整数	←	1=正常終了 0=前のローが存在しません -1=エラーが発生しました

説明

OC Previous in context機能は、現在列に先立つ結果列をロードします。現在列が未定義なら、**OC Activate context**の後のように、**OC Previous in context**は何もしません。

＜コンテキストID＞は、あらかじめ作成され有効であるコンテキストIDでなければなりません。

OC Previous in contextは現在の列と関連付けられた4th Dimensionオブジェクトを更新します。フィールドをともなうバインドのため、現在のレコードはセーブされずに更新されます。現在のレコードが存在しないなら、**OC Previous in context**は一致するバインドを無視します。

OC Previous in contextは、エラーが発生していない時は1、前の列が存在しない時は0、エラー発生時には-1を返します。

参照

OC Activate context、OC Load rows context

OC Update in context

OC Update in context (context_ID{;index}) → Integer

引数	タイプ		説明
コンテキストID	倍長整数	→	コンテキストID
索引	整数	→	配列インデックス
戻り値	整数	←	1=正常終了 -1=エラー発生しました

説明

OC Update in context機能は、コンテキストバインドで指定された4th Dimensionオブジェクトをベースとした ODBCサーバ上の現在列を更新する。

＜コンテキストID＞は、あらかじめ作成され有効であるコンテキストIDでなければなりません。

＜索引＞は、配列をともなうバインドのみに適用されます。＜索引＞が指定されないなら、**OC Update in context**は、現在の列番号と一致する配列の行を用いて、**OC Update in context**は、現在の列を更新します。

現在の列番号が配列のサイズより大きいならば、**OC Update in context**は-1を返します。

OC Update in contextは“ UPDATE ... WHERE ”タイプのSQLクエリを送信し、コンテキストの部分で、kForUpdateオプションを持つコラムのみを指定します（**OC ADD TO CONTEXT**を参照）。

OC Update in contextは、正常に実行された時は1、エラー発生時には-1を返します。

参照

OC Delete in context、OC Insert in context

4D ODBCエラーコード　A

次のリストは4D ODBCによって返されるエラーコードです。

エラー	エラーメッセージ
1	コマンドなしで呼び出しが実行されました。
2	割り当てられたプロセスなしで呼び出しが実行されました。
3	これらのバインドのためにマッキントッシュの上にメモリを割り当てることが不可能です。
4	この4Dタイプはこのリリースでサポートされません。
5	ピクチャ配列はこのリリースでサポートされません。
6	メモリ不足のためプロシージャのコピーができません。
7	2つの異なった4Dファイルではフィールドをバインドすることができません。
8	このコンテキストは何もバインドしませんでした。
9	4Dポインタタイプはサポートされていません。
10	4Dポインタ配列はサポートされていません。
11	4D 2次元配列はサポートされていません。
12	配列が4Dに割り当てられていません。
13	新しい配列を割り当てている間にメモリに問題が発生しました。
14	ログインパラメータが良くありません。ログインしないか、あるいは変数を変更してください。
15	パラメータとして渡されたカーソルは無効であるか、あるいは閉じられました。
16	コンテキストパラメータが無効です。
17	パラメータの1つが良くありません。プログラムをチェックしてください。
18	このタイプはサポートされていません。
19	このファイル／フィールドは存在しません。
20	この変数は存在しないか、あるいは宣言されなければなりません。
21	このプロシージャは存在しません。
22	「デバッグ」ウインドウを開くのにメモリが不足しています。
23	このコンテキストはアクティブ化されていませんでした。このコマンドを使う前にアクティブ化されてなくてはなりません。
24	このコンテキストは既にアクティブではありません。

エラー	エラーメッセージ
25	このコンテキストは既にアクティブです。
26	このコンテキストはこの処理の前に閉じられていなければなりません。
27	このコンテキストは更新モードを許可されていません。
28	この限界は無効です。それはイコールであるか、あるいはゼロより大きいに違いありません。
29	無効な条項のIDです。
30	4D ODBCのこのバージョンは4th Dimensionバージョン3.0.2あるいは4D Server 1.0.2あるいはそれ以降の版でのみ実行が可能です。
31	無効なコンテキストファイルです。
32	コンテキストファイルが存在しません。
33	コンテキストの読み出しエラーです。
34	このピクチャはコンテキストではありません。
35	関数IDの範囲がなくなっています。
36	情報タイプは4D ODBCによって無効あるいはサポートされません。
37	このオブジェクトは大きすぎます。ローは挿入あるいは更新されないでしょう。
38	レコードを4Dの中に作成できないエラーが起こりました。
39	レコードを4Dの中にセーブできないエラーが起こりました。
40	パラメータとして渡された配列は宣言されていないか、正しくないタイプです。

SQL GetInfo機能は、ODBC1.0拡張レベル1で紹介されています。この機能は、ドライバとhdbcで関連付けられるデータソースについての一般情報を返します。

構文

RETCODE SQL GetInfo（hdbc, fInfoType, rgbInfoValue, cbInfoValueMax, pcbInfoValue)

SQL GetInfo機能は、次のアーギュメントを受け取ります：

Type	アーギュメント	使用	説明
HDBC	hdbc	Input	通信操作
UWORD	InfoType	Input	情報の型
PTR	rgbInfoValue	Output	情報記憶のためのポインタ
SWORD	cbInfoValueMax	Input	rgbInfoValueバッファの最大長
SWORD FAR* p	cbInfoValue	Output	全体のバイト数（文字データのためのヌル終了バイトを含む）

HDBCは通信操作です。

UWORDは情報のタイプです。fInfoTypeはinterstのタイプを表す値でなければなりません。詳細は、後述の「コメント」の節を参照してください。

PTRは情報記憶のためのポインタです。要求されたfInfoTypeによって、返される情報は次のようなものです。

ヌル終了文字列、16bit整数値、32bitフラグ、32bitバイナリ値。

SWORDはrgbInfoValueバッファの最大長です。SWORD FAR*は全体のバイト数（文字データのためのヌル終了バイトを含む）で、rgbInfoValueに返すために利用可能です。文字データのために、cbInfoValueMax以上のバイト数が利用可能な場合は、rgbInfoValueの情報は（cbInfoValueMax -1）バイトに縮められ、ドライバによりヌル終了されます。他のすべてのデータタイプのために、cdValueMaxの値は無視され、ドライバはrgbValueは32bitと想定されます。

戻り値

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

SQLGetInfoがSQL_ERRORまたはSQL_SUCCESS_WITH_INFOを返す時、SQLErrorを呼び出すことによって、関連づけられたSQLSTATE値が得られるかもしれません。

次の表は、SQLGetInfoによって共通に返されるSQLSTATE値をリストし、この機能のコンテキスト内で各々を説明したものです。すなわち、表記 "(DM)"は、Driver Managerで返されるSQLSTATEの記述に先立ちます。別な方法で注記されないならば、各SQLSTATE値で関連付けられた戻りコードはSQL_ERRORです。

SQLSTATE	エラー	説明
01000	General warning	ドライバ特有の情報についてのメッセージ（Func-tion returns SQL_SUCCESS_WITH_INFO）
01004	Data truncated	バッファrgbInfoValueは求められた情報のすべてを返すのに十分大きくないので、情報を縮めます。（Function returns SQL_SUCCESS_WITH_INFO）
08003	Connection not open	（DM）fInfoTypeに要求された情報のタイプは通信のオープンを要求します。ODBCでとっておかれた情報タイプについて、SQL_ODBC_VERだけが通信を開かずに返せます。
22003	Numeric value out of range	要求された情報の戻り値が数値、あるいはバイナリの意味の損失をおこします。
IM001	Driver does not support this function	（DM）hdbcと一致するドライバが関数をサポートしていません。
S1000	General error	特定されないSQLSTATEエラーと実行が特定されたSQLSTATEエラーが明記されています。アーギュメントszErrorMsgのSQLErrorで返されたエラーメッセージでは、エラーとその原因を説明しています。
S1001	Memory allocation failure	ドライバは、機能の完成、実行のサポートを要求するメモリの割り当てをできません。
S1009	Invalid argument value	（DM）The fInfoTypeはSQL_DRIVER_HSTMTで、rgbInfoValueで指示された値は正しい文の操作ではありません。

S1090	Invalid string or buffer length	（DM）cbInfoValueMaxで指定された値が0より小です。
S1096	Information type out of range	（DM）fOptionで指定された値はODBCの情報形式で予約されたブロック番号ですが、このバージョンはドライバが正しくサポートしていません。
S1C00	Driver not capable	fOptionで指定された値はドライバ特有の情報形式で予約された番号の範囲内ですが、ドライバがサポートされていません。
S1T00	Timeout expired	要求されたデータソースが返される前にタイムアウトしました。SQLSetStmtOption、SQL_QUERY_TIMEOUTでタイムアウト時間は設定されます。

コメント

現在定義されている情報タイプについては、この節で説明されています。異なるデータソース利用のために、更に定義されるでしょう。0～999の情報タイプはODBCで予約されます。ドライバ開発者は、ドライバ特有の利用のためにSQL_INFO_DRIVER_START異常の値を予約しなければなりません。

SQLGetInfoは5つの異なる形式から1つの形式で情報を返します：

- ヌル終了文字列
- 16bit整数列
- 32bitビットマスク
- 32bit整数値
- 32bitバイナリ値

次の情報タイプの各形式は、タイプ記述で書かれています。よって、アプリケーションはrgbInfoValueの戻り値をキャストしなければなりません。アプリケーションが32bitビットマスクからデータ検索する方法の例については、この節のコード例を参照してください。

ドライバは、次のテーブルで定義された各情報タイプの値を返します。情報タイプがドライバやデータソースに適用されないなら、ドライバは次の値の1つを返します。

rgbInfoValue形式	**戻り値**
文字列("Y"or "N")	"N"
文字列(not "Y"or "N")	空文字列
16-bit整数	0
32-bitビットマスク or 32-bitバイナリ値	0L

データソースがプロシージャをサポートしていない場合、SQLGetInfoはプロシージャに関連するfInfoTypeの値に次の値を返します。

fInfoType	戻り値
SQL_PROCEDURES	"N"
SQL_ACCESSIBLE_PROCEDURES	"N"
SQL_MAX_PROCEDURE_NAME_LEN	0
SQL_PROCEDURE_TERM	空文字列

SQLGetInfoは ODBCの情報タイプ範囲内だがドライバで未定義なODBCのバージョンであるfInfoType値のSQLSTATE S1096（不正引数値）を返します。ドライバが従うODBCのバージョンを決定するため、アプリケーションは、SQL_DRIVER_ODBC_VERとともに、SQLGetInfoを呼び出します。

注：アプリケーション開発者は、ODBC1.0ドライバで定義されているがドライバやデータソースに適用されないfInfoTypeの値によって、ODBC1.0ドライバがSQL_ERRORやSQLSTATE S1C00（ドライバが利用不能）を返すことに気付いておくべきです。

情報のタイプ

この節では、SQLGetInfoでサポートされる情報のタイプをリストしています。情報タイプは絶対的に分類され、アルファベット順にリストされています。

ドライバ情報

次のfInfoTypeは、アクティブな文番号、データソース名、API実行レベルといった、ODBCドライバについての情報を返します。

SQL_ACTIVE_CONNECTIONS
SQL_ACTIVE_STATEMENTS
SQL_DATA_SOURCE_NAME
SQL_DRIVER_HDBC
SQL_DRIVER_HENV
SQL_DRIVER_HLIB
SQL_DRIVER_HSTMT
SQL_DRIVER_NAME
SQL_DRIVER_ODBC_VER
SQL_DRIVER_VER
SQL_FETCH_DIRECTION
SQL_FILE_USAGE
SQL_GETDATA_EXTENSIONS
SQL_LOCK_TYPES
SQL_ODBC_API_CONFORMANCE
SQL_ODBC_SAG_CLI_CONFORMANCE
SQL_ODBC_VER
SQL_POS_OPERATIONS
SQL_ROW_UPDATES
SQL_SEARCH_PATTERN_ESCAPE
SQL_SERVER_NAME

DBMS製品情報

次のfInfoType値は、DBMSの名称やバージョンなどのDBMS製品情報を返します。

SQL_DATABASE_NAME
SQL_DBMS_NAME
SQL_DBMS_VER

データソース情報

次のfInfoType値は、カーソルの性質や処理能力のような、データソースの情報を返します。

SQL_ACCESSIBLE_PROCEDURES
SQL_ACCESSIBLE_TABLES
SQL_BOOKMARK_PERSISTENCE
SQL_CONCAT_NULL_BEHAVIOR
SQL_CURSOR_COMMIT_BEHAVIOR
SQL_CURSOR_ROLLBACK_BEHAVIOR
SQL_DATA_SOURCE_READ_ONLY
SQL_DEFAULT_TXN_ISOLATION
SQL_MULT_RESULT_SETS
SQL_MULTIPLE_ACTIVE_TXN
SQL_NEED_LONG_DATA_LEN
SQL_NULL_COLLATION
SQL_OWNER_TERM
SQL_PROCEDURE_TERM
SQL_QUALIFIER_TERM
SQL_SCROLL_CONCURRENCY
SQL_SCROLL_OPTIONS
SQL_STATIC_SENSITIVITY
SQL_TABLE_TERM
SQL_TXN_CAPABLE
SQL_TXN_ISOLATION_OPTION
SQL_USER_NAME

サポートされたSQL

次のfInfoType値は、データソースでサポートされたSQLステートメントの情報を返します。これらの情報タイプはODBC SQL文法を記述し尽くしません。代わりに、データソースが共通に異なるレベルをサポートする、文法の一部分を記述します。

アプリケーションは、SQL_ODBC_SQL_CONFORMANCE情報タイプからサポートされた文法の標準を定めるべきで、また明記された適合レベルから相違を定めるために他の情報タイプを使用すべきです。

SQL_ALTER_TABLE
SQL_COLUMN_ALIAS
SQL_CORRELATION_NAME
SQL_EXPRESSIONS_IN_ORDERBY
SQL_GROUP_BY
SQL_IDENTIFIER_CASE
SQL_IDENTIFIER_QUOTE_CHAR
SQL_KEYWORDS
SQL_LIKE_ESCAPE_CLAUSE
SQL_NON_NULLABLE_COLUMNS
SQL_ODBC_SQL_CONFORMANCE
SQL_ODBC_SQL_OPT_IEF
SQL_ORDER_BY_COLUMNS_IN_SELECT
SQL_OUTER_JOINS
SQL_OWNER_USAGE
SQL_POSITIONED_STATEMENTS
SQL_PROCEDURES
SQL_QUALIFIER_LOCATION
SQL_QUALIFIER_NAME_SEPARATOR
SQL_QUALIFIER_USAGE
SQL_QUOTED_IDENTIFIER_CASE
SQL_SPECIAL_CHARACTERS
SQL_SUBQUERIES
SQL_UNION

SQLの制限

次のfInfoType値は、確認の最大長と選択リスト中のコラムの最大番号のような、SQLステートメントの確認、節に適用される制限情報を返します。ドライバか、データソースによって制限が課せられるかもしれません。

SQL_MAX_BINARY_LITERAL_LEN　SQL_MAX_CHAR_LITERAL_LEN
SQL_MAX_COLUMN_NAME_LEN　SQL_MAX_COLUMNS_IN_GROUP_BY
SQL_MAX_COLUMNS_IN_ORDER_BY　SQL_MAX_COLUMNS_IN_INDEX
SQL_MAX_COLUMNS_IN_SELECT　SQL_MAX_COLUMNS_IN_TABLE
SQL_MAX_CURSOR_NAME_LEN　SQL_MAX_INDEX_SIZE
SQL_MAX_OWNER_NAME_LEN　SQL_MAX_PROCEDURE_NAME_LEN
SQL_MAX_QUALIFIER_NAME_LEN　SQL_MAX_ROW_SIZE
SQL_MAX_ROW_SIZE_INCLUDES_LONG　SQL_MAX_STATEMENT_LEN
SQL_MAX_TABLE_NAME_LEN　SQL_MAX_TABLES_IN_SELECT
SQL_MAX_USER_NAME_LEN

スカラ関数情報

次のfInfoType値は、データソースやドライバでサポートされたスカラ関数の情報を返します。

SQL_CONVERT_FUNCTIONS
SQL_NUMERIC_FUNCTIONS
SQL_STRING_FUNCTIONS
SQL_SYSTEM_FUNCTIONS
SQL_TIMEDATE_ADD_INTERVALS
SQL_TIMEDATE_DIFF_INTERVALS
SQL_TIMEDATE_FUNCTIONS

変換情報

CONVERTスカラ関数で、次のfInfoType値は、データソースが指定されたSQLデータタイプを変換できるSQLデータタイプのリストを返します。

SQL_CONVERT_BIGINT　SQL_CONVERT_BINARY
SQL_CONVERT_BIT　SQL_CONVERT_CHAR
SQL_CONVERT_DATE　SQL_CONVERT_DECIMAL
SQL_CONVERT_DOUBLE　SQL_CONVERT_FLOAT
SQL_CONVERT_INTEGER　SQL_CONVERT_LONGVARBINARY
SQL_CONVERT_LONGVARCHAR　SQL_CONVERT_NUMERIC
SQL_CONVERT_REAL　SQL_CONVERT_SMALLINT
SQL_CONVERT_TIME　SQL_CONVERT_TIMESTAMP
SQL_CONVERT_TINYINT　SQL_CONVERT_VARBINARY
SQL_CONVERT_VARCHAR

情報タイプ定義

この節では、アルファベット順に各情報タイプ、紹介されるODBCのバージョン、その記述をアルファベット順にリストします。

- SQL_ACCESSIBLE_PROCEDURES (ODBC 1.0)

  文字列、すなわちSQLプロシージャで返されるすべてのプロシージャをユーザが実行可能ならば"Y"を、実行できないならば"N"を返します。

- SQL_ACCESSIBLE_TABLES (ODBC 1.0)

  文字列、すなわちSQLtablesで返されるすべてのテーブルに、ユーザにSELECT特権が保証されているならば"Y"を、ユーザがアクセス不能ならば"N"を返します。

- SQL_ACTIVE_CONNECTIONS (ODBC 1.0)

  ドライバがサポートできるアクティブなhdbcの最大数を指定している16bitの整数値。この値は、ドライバかデータソースで課せられた制限を反映できます。

- SQL_ACTIVE_STATEMENTS (ODBC 1.0)

  ドライバがhdbcのためにサポートできるアクティブなhstmtの最大数を指定している16bitの整数値。

  この値はドライバかデータソースで課せられた制限を反映できます。指定される制限がないか、わからない制限ならば、この値は0に設定されます。

- SQL_ALTER_TABLE (ODBC 2.0)

  データ情報源によって支援されてALTER TABLE文で節を列挙している32bitのビットマスク。

  次のビットマスクはいずれの節がサポートされるか決定するのに使われます：

  SQL_AT_ADD_COLUMNSQL_AT_DROP_COLUMN

- SQL_BOOKMARK_PERSISTENCE (ODBC 2.0)

  ブックマークが存続するオペレーションを列挙している32bitのビットマスク。次のビットマスクは、ブックマークが持続する引数で決定すべきフラグと関連して使われます：

  SQL_BP_CLOSE = Bookmarksはアプリケーションがhstmtと関連したカーソルを閉じるために、SQL_CLOSEオプションで、SQLFreeStmtを呼び出した後で、正しくなります。

  SQL_BP_DELETE=The bookmark for a rowは列が削除された後で、正しくなります。

  SQL_BP_DROP=Bookmarksは、アプリケーションがhstmtをドロップするため、SQL_DROPオプションで、SQLFreeStmtを呼び出した後で、正しくなります。

  SQL_BP_SCROLL = Bookmarksは、（SQLExtendedFetchを使用して）オペレーションをスクロールした後、正しくなります。SQLExtendedFetchが呼び出された後、すべての

ブックマークは正しいままでなければならないので、この値は、アプリケーションによって、ブックマークがサポートされているか決めるために使うことができます。

SQL_BP_TRANSACTION=Bookmarksは、アプリケーションが処理を引き渡し、ロールバックした後で、正しくなります。

SQL_BP_UPDATE = The bookmark for a rowは、列のコラムが更新され、キーコラムを含んだ後で、正しくなります。

SQL_BP_OTHER_HSTMT=A bookmark associated with one hstmtは、他のhstmtとともに使用されます。

- SQL_COLUMN_ALIAS (ODBC 2.0)

  文字列、すなわちデータソースがコラムエイリアスをサポートしているなら"Y"を、それ以外なら"N"を返します。

- SQL_CONCAT_NULL_BEHAVIOR (ODBC 1.0)

  どのようにデータソースが、NULLでない値の文字データタイプコラムと、NULL値の文字データタイプコラムを結合するかを示している16bitの整数値、すなわち、SQL_CB_NULL=Resultはヌル値で、SQL_CB_NON_NULL=Resultはヌル値でないコラムとの結合です。

- SQL_CONVERT_BIGINT　　SQL_CONVERT_LONGVARCHAR

  SQL_CONVERT_BINARY　　SQL_CONVERT_NUMERIC

  SQL_CONVERT_BIT　　SQL_CONVERT_REAL

  SQL_CONVERT_CHAR　　SQL_CONVERT_SMALLINT

  SQL_CONVERT_DATE　　SQL_CONVERT_TIME

  SQL_CONVERT_DECIMAL　　SQL_CONVERT_TIMESTAMP

  SQL_CONVERT_DOUBLE　　SQL_CONVERT_TINYINT

  SQL_CONVERT_FLOAT　　SQL_CONVERT_VARBINARY

  SQL_CONVERT_INTEGER　　SQL_CONVERT_VARCHAR

  SQL_CONVERT_LONGVARBINARY

  (ODBC 1.0)

  32bitビットマスク。ビットマスクは、CONVERTスカラー関数でデータソースによってサポートされる変換がfInfoTypeで名づけられたタイプのデータのために作用することを示します。ビットマスクが0なら、データソースは名づけられたタイプのデータのために、同データタイプへの変換を含め、変換をサポートしません。

例えば、データソースがSQL_BIGINTデータタイプへのSQL_INTEGERデータの変換をサポートしているかを理解するため、アプリケーションはSQL_CONVERT_INTEGERのfInfoTypeでSQLGetInfoを呼び出します。アプリケーションはSQL_CVT_BIGINTで返されたビットマスクのANDを取ります。結果として生じた値が0以外ならば、変換はサポートされています。

次のビットマスクは変換がサポートを測定するのに使用されます。

SQL_CVT_BIGINT	SQL_CVT_LONGVARCHAR
SQL_CVT_BINARY	SQL_CVT_NUMERICS
SQL_CVT_BIT	SQL_CVT_REAL
SQL_CVT_CHAR	SQL_CVT_SMALLINT
SQL_CVT_DAT	SQL_CVT_TIME
SQL_CVT_DECIMAL	SQL_CVT_TIMESTAMP
SQL_CVT_DOUBLE	SQL_CVT_TINYINT
SQL_CVT_FLOAT	SQL_CVT_VARBINARY
SQL_CVT_INTEGER	SQL_CVT_VARCHAR
SQL_CVT_LONGVARBINARY	

■ SQL_CONVERT_FUNCTIONS (ODBC 1.0)

ドライバによってサポートされ、関連づけられたスカラー変換関数を列挙する32bitのビットマスク。次のビットマスクはいずれの変換機能がサポートされるか、測定するために使われます。

SQL_FN_CVT_CONVERT

■ SQL_CORRELATION_NAME (ODBC 1.0)

テーブル相互関係名がサポートされるかどうかを示す16bitの整数。SQL_CN_DIFFERENT=相関関係名がサポートされますが、それらが表すテーブル名とは違わなくてはなりません。

SQL_CN_ANY=相関関係名がサポートされ、そしてどんな正当なユーザ定義名でもあり得ます。

■ SQL_CURSOR_COMMIT_BEHAVIOR (ODBC 1.0)

COMMITオペレーションがデータソースでどのようにカーソルと準備文に影響を与えるか示す16bitの整数値。

SQL_CB_DELETE=カーソルを閉じ、準備文を削除します。カーソルの再利用のために、アプリケーションは再準備し、hstmtを実行しなければなりません。

SQL_CB_CLOSE=カーソルを閉じます。準備文のために、アプリケーションは再びSQLPrepareを呼び出さないで、hstmtにSQLExecuteを呼び出します。

SQL_CB_PRESERVE=COMMITオペレーション前と同じ位置にカーソルを維持します。アプリケーションはデータを取り続けられ、再準備なしに、カーソルを閉じhstmt

を再実行できます。

■ SQL_CURSOR_ROLLBACK_BEHAVIOR (ODBC 1.0)

どのようにROLLBACKオペレーションがカーソルに作用し、データソースに文を準備するかを示す16bit整数値。

SQL_CB_DELETE=カーソルを閉じ、準備文を削除します。カーソルを再利用するためには、アプリケーションを再準備し、hstmtを実行しなければなりません。

SQL_CB_CLOSE=カーソルを閉じます。準備文のために、アプリケーションはSQLPrepareを再び呼び出さずに、SQLExecuteをhstmtに呼び出すことができます。SQL_CB_PRESERVE=ROLLBACKオペレーション前と同じ位置にカーソルを維持します。アプリケーションは、データを取り続け、または再準備なしに、カーソルを閉じ、hstmtを再実行できます。

■ SQL_DATA_SOURCE_NAME (ODBC 1.0)

通信の間に使われたデータソース名を持っている文字列。アプリケーションがSQLConnectを呼び出したら、szDSN引数値です。アプリケーションがSQLDriverConnectまたはSQLBrowseConnectを呼び出したら、これはドライバに手渡される通信文字列のDSNキーワード値です。通信文字列が（ドライバキーワードを含んでいる時のように）DSNキーワード値を含んでいなかったなら、これは空文字列です。

■ SQL_DATA_SOURCE_READ_ONLY (ODBC 1.0)

文字列、すなわち、データソースが「読込み専用」モードに設定されている場合は"Y"、そうでない場合は、"N"です。この特徴はデータソース自身にのみ関係があり、データソースにアクセスを可能にするドライバの特徴ではありません。

■ SQL_DATABASE_NAME (ODBC 1.0)

使用中の現在のデータベース名を持つ文字列。データソースが名づけられたオブジェクトを定義するなら「データベース」と呼ばれます。

注：ODBC2.0で、このfInfoType値はSQL_CURRENT_QUALIFIER通信引数で置き換えられました。ODBC2.0ドライバがSQL_DATABASE_NAME情報タイプをサポートし続けるべきで、ODBC2.0アプリケーションは、ODBC1.0ドライバでそれを使うだけであるべきです。

■ SQL_DBMS_NAME (ODBC 1.0)

ドライバでアクセスされるDBMS製品名を持つ文字列です。バージョンは##.##.####の形式で、最初の2桁はメジャーバージョンで、次の2桁がマイナーバージョン、最後の4桁はリリースバージョンを示します。ドライバは、この形式で製品バージョンを与えなければなりませんが、同様にDBMS製品固有のバージョンを加えなければなりません。例えば、"04.01.000 Rdb 4.1"のようにです。

■ SQL_DEFAULT_TXN_ISOLATION (ODBC 1.0)

ドライバあるいはデータソースでサポートされる、デフォルト分散処理レベルを示す

32bitの整数。データソースが処理をサポートしない場合は0となります。次の項では、分散処理レベルの定義を行っています。すなわち、次のようなものです。

不正な読取り

処理1が列を変更します。処理1が変更を行う前に、処理2は変更された列を読み込みます。処理1が変更を戻すなら、処理2は存在しないと思われる列を読んでしまっています。

一度限りの読取り

処理1は列を読みます。処理2は列の変更あるいは削除を行います。処理1がその列を読み直そうとしたら、違う列の値を読むか、または列の削除に気付くでしょう。

幽霊

処理1が検索基準を満たす列の組を読みます。処理2は検索基準に当てはまる列を挿入します。処理1が列を読んだ文を再び実行するなら、異なった列の組を検索します。データソースが処理をサポートしていたら、ドライバは次のビットマスクの1つを返します。

SQL_TXN_READ_UNCOMMITTED=不正な読取り、一度限りの読取り、および幽霊がありえます。

SQL_TXN_READ_COMMITTED=不正な読取りはありえませんが、一度限りの読取り、および幽霊がありえます。

SQL_TXN_REPEATABLE_READ=不正な読取り、および一度限りの読取りはありえませんが、幽霊はありえます。

SQL_TXN_SERIALIZABLE=処理は連続的に行われます。不正な読取り、一度限りの読取り、および幽霊はありえません。

SQL_TXN_VERSIONING=処理は連続的に行われますが、SQL_TXN_SERIALIZABLEと共により高く同時発生がありえます。不正な読取りはありえません。典型的に、SQL_TXN_SERIALIZABLEは同時発生を減らすロックされたプロトコルを利用することで、実行されます。SQL_TXN_VERSIONINGはレコードバージョニングのようなロックされないプロトコルを使用することで、実行されます。

ちなみにOracleの読取り調和の分散レベルは、SQL_TXN_VERSIONINGです。

- SQL_DRIVER_HDBCSQL_DRIVER_HENV (ODBC 1.0)

  32bit値。ドライバ環境が引数hdbcによって決定され、操作、通信操作を行います。これらの情報タイプはドライバマネージャだけで実行されます。

- SQL_DRIVER_HLIB (ODBC 2.0)

  32bit値。ドライバDLLがロードされる時、ドライバマネージャに返されるライブラリハンドルです。ハンドルはただSQLGetInfoへの呼び出しで指定されたhdbcに有効なだ

けです。この情報タイプはハンドルマネージャーのみで実行されます。

- SQL_DRIVER_HSTMT (ODBC 1.0)

  32bit値。ドライバマネージャ文操作により決定されたドライバ文の操作です。それはアプリケーションからrgbInfoValueで入力に渡されます。rgbInfoValueが入出力両方の文である場合は注意してください。

  rgbInfoValue値で渡される入力hstmtはhdbc文に割り当てられたhstmtに違いありません。

  この情報タイプはドライバマネージャだけが実行します。

- SQL_DRIVER_NAME (ODBC 1.0)

  データソースへのアクセスに使用されるドライバのファイル名を持つ文字列。

- SQL_DRIVER_ODBC_VER(ODBC 2.0)

  ドライバがサポートするODBCバージョンを持つ文字列。

  バージョンは##.##の形式で、最初の2桁はメジャーバージョンで、次の2桁がマイナーバージョンを示します。

  SQL_SPEC_MAJORとSQL_SPEC_MINORは、メジャーバージョン番号、マイナーバージョン番号を定義します。

  このマニュアルに書かれているODBCのバージョンとして、2と0があるので、ドライバは"02.00"を返すべきです。ドライバがSQLGetInfoをサポートしても、fInfoType文のこの値をサポートしていないなら、ドライバマネージャは"01.00"を返します。

- SQL_DRIVER_VER (ODBC 1.0)

  ドライバのバージョンを持つ文字列で、オプションでドライバを記述します。最小では、バージョンは##.##.####の形式で、最初の2桁はメジャーバージョンで、次の2桁がマイナーバージョン、最後の4桁はリリースバージョンを示します。

- SQL_EXPRESSIONS_IN_ORDERBY (ODBC 1.0)

  文字列。ORDER BY listでデータソースが表現をサポートしている場合は"Y"、サポートしていない場合は"N"です。

- SQL_FETCH_DIRECTION (ODBC 1.0)

  情報タイプはODBC1.0で紹介されています。各ビットマスクは紹介されたバージョンで分類されています。32bitビットマスクはサポートされた取出し指示オプションを列挙しています。次のビットマスクはいずれのオプションがサポートされているか測定するためにフラグと関連して使われます。

ビットマスク	バージョン
SQL_FD_FETCH_NEXT	ODBC 1.0
SQL_FD_FETCH_FIRST	ODBC 1.0
SQL_FD_FETCH_LAST	ODBC 1.0
SQL_FD_FETCH_PRIOR	ODBC 1.0
SQL_FD_FETCH_ABSOLUTE	ODBC 1.0
SQL_FD_FETCH_RELATIVE	ODBC 1.0
SQL_FD_FETCH_RESUME	ODBC 1.0
SQL_FD_FETCH_BOOKMARK	ODBC 2.0

■ SQL_FILE_USAGE (ODBC 2.0)

データソースでどのように単一層ドライバが直接ファイルを扱うか示す16bitの整数値。

SQL_FILE_NOT_SUPPORTED=ドライバが単一層ではありません。例えば、ORACLEドライバは二層ドライバです。

SQL_FILE_TABLE=単一層ドライバでは、データソースのファイルをテーブルとして扱います。例えばXbaseドライバは各Xbaseファイルをテーブルとして扱っています。

SQL_FILE_QUALIFIER=単一層ドライバは、データソースのファイルを適格として扱います。例えば、Microsoft AccessドライバはMicrosoft Accessファイルを完全なデータベースとして扱っています。アプリケーションは、ユーザがどのようにデータを選択するか定めるために使用できます。例えば、ORACLEユーザやMicrosoft Accessユーザが、一般的にデータをテーブルにストアされたものと見なす一方で、Xbaseユーザはデータをファイルにストアされたものと見なしています。ユーザがXbaseデータソースを選択する時、アプリケーションは「Windows File Open common」ダイアログボックスを表示できます。また、ユーザがMicrosoft AccessやORACLEデータソースを選択する時、アプリケーションは「custom Select Table」ダイアログボックスを表示できます。

■ SQL_GETDATA_EXTENSIONS (ODBC 2.0)

SQLGetDataへの拡張を列挙する32bitのビットマスク。次のビットマスクは、ドライバがSQLGetDataのため、どんな共通拡張を支援するか定めるためにフラグと関連して使用します。

すなわち、SQL_GD_ANY_COLUMN=SQLGetDataはバインドされた列の前のものを含め、バインドされていない列にでも呼び出されます。

SQL_GD_ANY_ORDERも返されないのなら、列番号を上げるためには、列が呼び出されなければならないことに注意しましょう。

SQL_GD_ANY_ORDER=バインドされてない列にも、秩序立ててSQLGetDataは呼び出すことができます。SQLGetDataはSQL_GD_ANY_COLUMNも返されないならば、最後にバインドされた列の後で、列に呼び出されることに注意しましょう。

SQL_GD_BLOCK=SQLGetDataはSQLSetPosで列を配置した後で、データのブロック（2

列以上）のいくらかの横列のバインドされていない縦列で、呼び出すことができます。

SQL_GD_BOUND=SQLGetDataは、バインドされていない列と同様に、バインドされた列でも呼び出せます。SQL_GD_ANY_COLUMNも返されないならば、ドライバはこの値を返せません。

SQLGetDataは最後にバインドされた列の後に生じ、列番号を増やすために呼び出されるバインドされていない列は、データを返すようにのみ要求されています。

■ SQL_GROUP_BY (ODBC 2.0)

選択されたリストでGROUP BY節の列と集合できない列の間の関係を指定する16bitの整数値。

SQL_GB_NOT_SUPPORTED=GROUP BY節がサポートされていません。

SQL_GB_GROUP_BY_EQUALS_SELECT=GROUP BY節は、選択されたリストですべての集合できない列を含まなければなりません。他の列は含めることができません。例えば、SELECT DEPT、MAX(SALARY) FROM EMPLOYEE GROUP BY DEPTなどです。

SQL_GB_GROUP_BY_CONTAINS_SELECT=GROUP BY節は、選択されたリストにすべての集合できない列を含まねばなりません。選択されていないリストの列も含まれます。例えば、SELECT DEPT、MAX(SALARY)FROM EMPLOYEE GROUP BY DEPT、AGEなどです。

SQL_GB_NO_RELATION=GROUP BY節の列と選択されたリストは関連付けられません。集合できない意味、すなわち選択されたリストの集合できない列はデータソースへの依存しています。例えば、SELECT DEPT、SALARY FROM EMPLOYEE GROUP BY DEPT、AGEなどです。

■ SQL_IDENTIFIER_CASE (ODBC 1.0)

次のような16bit 整数値。

SQL_IC_UPPER=SQLの認証者は大文字小文字の違いを無視して、大文字でシステムカタログにストアされます。

SQL_IC_LOWER =

認証者は大文字小文字の違いを無視して、小文字でシステムカタログにストアされます。

SQL_IC_SENSITIVE=

SQLの認証者は大文字小文字の違いを識別して、入り混ざった場合でシステムカタログにストアされます。

SQL_IC_MIXED=

SQLの認証者は大文字小文字の違いを無視して、入り混ざった場合でもシステムカタログにストアされます。

■ SQL_IDENTIFIER_QUOTE_CHAR (ODBC 1.0)

SQLステートメントの引用された認証者の始まりと終わりの範囲限界として使用される文字列（引数としてODBC関数に渡された認証者が引用される必要がありません)。

データソースが引用された認証者をサポートしないなら、空白が返されます。

■ SQL_KEYWORDS (ODBC 2.0)

すべてのデータソース固有のキーワードをコンマで分けたリストを含む文字列。

このリストはODBCに特有なキーワード、あるいはデータソースとODBC両方で使用されるキーワードを含みません。

■ SQL_LIKE_ESCAPE_CLAUSE (ODBC 2.0)

文字列。すなわち、データソースがパーセント文字（%）やアンダースコア文字（_）をエスケープ文字としてサポートしていて、ドライバがエスケープ文字を定義するODBC構文をサポートしているなら "Y"、それ以外なら"N"です。

■ SQL_LOCK_TYPES (ODBC 2.0)

SQLSetPosでfLock文のサポートされたロックタイプを列挙した32bitビットマスク。次のビットマスクは、サポートされたロックタイプを定めるフラグで、使用されます。

SQL_LCK_NO_CHANGE
SQL_LCK_EXCLUSIVE
SQL_LCK_UNLOCK

■ SQL_MAX_BINARY_LITERAL_LEN (ODBC 2.0)

SQLステートメントでバイナリ文字の最大長を指定する32bit整数値（SQLGetTypeInfoで返される接頭辞と接尾辞を除く16進数文字)。例えば、バイナリ文字0xFFAAは4の長さを持っています。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_CHAR_LITERAL_LEN (ODBC 2.0)

SQLステートメントで文字の最大長を指定する32bit整数値（SQLGetTypeInfoで返される接頭辞と接尾辞を除く数字)。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_COLUMN_NAME_LEN (ODBC 1.0)

データソースの列名の最大長を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_COLUMNS_IN_GROUP_BY (ODBC 2.0)

GROUP BY節で許される最大の列番号を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_COLUMNS_IN_INDEX (ODBC 2.0)

インデックスで許される最大の列番号を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_COLUMNS_IN_ORDER_BY (ODBC 2.0)

ORDER BY節で許される最大の列番号を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_COLUMNS_IN_SELECT (ODBC 2.0)

選択リストで許される最大の列番号を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_COLUMNS_IN_TABLE (ODBC 2.0)

テーブルで許される最大の列番号を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_CURSOR_NAME_LEN (ODBC 1.0)

データソースでのカーソル名の最大長を指定する16bit整数値。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_INDEX_SIZE (ODBC 2.0)

インデックスの結合されたフィールドで許される最大のバイト数を指定する32bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_OWNER_NAME_LEN (ODBC 1.0)

データソースの所有者名の最大長を指定する16bit整数値。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_PROCEDURE_NAME_LEN (ODBC 1.0)

データソースのプロシージャ名の最大長を指定する16bit整数値。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_QUALIFIER_NAME_LEN (ODBC 1.0)

データソースの認証者名の最大長を指定する16bit整数値。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_ROW_SIZE (ODBC 2.0)

テーブルの一列の最大長を指定する32bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_ROW_SIZE_INCLUDES_LONG (ODBC 2.0)

文字列。すなわち、最大列サイズが、列ですべてのSQL_LONGVARCHARとSQL_LONGVARBINARYコラムの長さを含むSQL_MAX_ROW_SIZE情報タイプを返すなら"Y"、それ以外なら"N"です。

■ SQL_MAX_STATEMENT_LEN (ODBC 2.0)

データソースのテーブル名の最大長を指定する16bit整数値。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MAX_TABLES_IN_SELECT (ODBC 2.0)

SELECT文のFROM節で、許されたテーブルの最大番号を指定する16bit整数値。制限がないか、制限がわからない場合、この値は0に設定されます。

■ SQL_MAX_USER_NAME_LEN (ODBC 2.0)

データソースのユーザ名の最大長を指定する16bit整数値。最大長がないか、長さがわからない場合、この値は0に設定されます。

■ SQL_MULT_RESULT_SETS (ODBC 1.0)

文字列。すなわち、データソースが複合結果セットをサポートしている場合は"Y"、サポートしてない場合は"N"です。

■ SQL_MULTIPLE_ACTIVE_TXN (ODBC 1.0)

文字列。すなわち、多重通信セットでアクティブな処理が許されるある場合"Y"、一度にひとつの通信がアクティブ処理を持つことができるなら"N"。

■ SQL_NEED_LONG_DATA_LEN (ODBC 2.0)

文字列。すなわち、データソースに送られる値の前に、データソースが倍長データ値（データタイプがSQL_LONGVARCHAR、SQL_LONGVARBINARY、またはロング、データソース特有データタイプ）の長さをを必要とする場合"Y"、そうでないなら"N"です。詳細については、SQLBind ParameterとSQLSetPosを参照してください。

■ SQL_NON_NULLABLE_COLUMNS (ODBC 1.0)

データソースでヌルにならない列をサポートしているかを指定した16ビット整数値。SQL_NNC_NULL=すべての列がヌル可能ににならねばなりません。SQL_NNC_NON_NULL=列はヌルにならないかもしれません（CREATE TABLE文でデータソースはNOT NULL 列をサポートします)。

■ SQL_NULL_COLLATION (ODBC 2.0)

リストでソートされたヌルが指定された16ビット整数値。
SQL_NC_END=ヌルはソートオーダに関係なく、リストの終わりにソートされます。SQL_NC_HIGH=ヌルはリストのハイエンドにソートされます。SQL_NC_LOW=ヌルはリストのローエンドにソートされます。SQL_NC_START=ヌルはソートオーダに関係なく、リストのはじめにソートされます。

■ SQL_NUMERIC_FUNCTIONS (ODBC 1.0)

情報タイプはODBC1.0で紹介されました。すなわち、それぞれのビットマスクがそれが紹介されたバージョンで分類されます。

ドライバと関連付けられたデータソースにサポートされたスカラー関数を列挙する32bitのビットマスクです。次のビットマスクは、どの数の関数がサポートされるか決定するために使われます。

ビットマスク	バージョン
SQL_FN_NUM_ABS	ODBC 1.0
SQL_FN_NUM_ACOS	ODBC 1.0
SQL_FN_NUM_ASIN	ODBC 1.0
SQL_FN_NUM_ATAN	ODBC 1.0
SQL_FN_NUM_ATAN2	ODBC 1.0
SQL_FN_NUM_CEILING	ODBC 1.0
SQL_FN_NUM_COS	ODBC 1.0
SQL_FN_NUM_COT	ODBC 1.0
SQL_FN_NUM_DEGREES	ODBC 2.0
SQL_FN_NUM_EXP	ODBC 1.0
SQL_FN_NUM_FLOOR	ODBC 1.0
SQL_FN_NUM_LOG	ODBC 1.0
SQL_FN_NUM_LOG10	ODBC 2.0
SQL_FN_NUM_MOD	ODBC 1.0
SQL_FN_NUM_PI	ODBC 1.0
SQL_FN_NUM_POWER	ODBC 2.0
SQL_FN_NUM_RADIANS	ODBC 2.0
SQL_FN_NUM_RAND	ODBC 1.0
SQL_FN_NUM_ROUND	ODBC 2.0
SQL_FN_NUM_SIGN	ODBC 1.0
SQL_FN_NUM_SIN	ODBC 1.0
SQL_FN_NUM_SQRT	ODBC 1.0
SQL_FN_NUM_TAN	ODBC 1.0
SQL_FN_NUM_TRUNCATE	ODBC 2.0

■ SQL_ODBC_API_CONFORMANCE (ODBC 1.0)

ODBC適合レベルを示す16bitの整数値。SQL_OAC_NONE=なし、SQL_OAC_LEVEL1 =Level 1がサポートされています。SQL_OAC_LEVEL2 =Level 2がサポートされています。

■ SQL_ODBC_SAG_CLI_CONFORMANCE (ODBC 1.0)

SAG仕様書の関数の遵守を示す16ビットの整数値。SQL_OSCC_NOT_COMPLIANT= SAGに迎合的ではありません。すなわち、1つ以上のコア関数はサポートされていません。

SQL_OSCC_COMPLIANT=SAGへ迎合します。

■ SQL_ODBC_SQL_CONFORMANCE (ODBC 1.0)

ドライバによってサポートされているSQL文法を示す16bit整数値。SQL_OSC_MINIMUM=最小限の文法がサポートされています。SQL_OSC_CORE=コアの文法がサポートされています。SQL_OSC_EXTENDED=拡張された文法がサポートされています。

■ SQL_ODBC_SQL_OPT_IEF (ODBC 1.0)

文字列。すなわち、データソースがオプションの完全性拡張ファシリティをサポートしているなら"Y"、そうでないなら、"N"です。

■ SQL_ODBC_VER (ODBC 1.0)

ドライバマネージャが従うODBCのバージョンを持つ文字列。

バージョンは##.##の形式で、最初の2桁はメジャーバージョンで、次の2桁がマイナーバージョンを示します。これはドライバマネージャに単に実行されます。

■ SQL_ORDER_BY_COLUMNS_IN_SELECT (ODBC 2.0)

文字列。すなわち、ORDER BY節の列が選択リストになければならない場合は"Y"、そうでない場合は"N"です。

■ SQL_OUTER_JOINS (ODBC 1.0)

情報タイプはODBC1.0で紹介されました。それぞれの戻り値が紹介されたバージョンで分類されます。

文字列。すなわち、"N"=No.データソースは外部の結合をサポートしていません(ODBC 1.0)。"Y"=Yes.データソースは2テーブルの外部結合をサポートします、そしてドライバは重ねられた外部結合以外のODBC 外部結合構文をサポートします。

■ SQL_OWNER_TERM (ODBC 1.0)

所有者のデータソースのベンダー名を持っている文字列。例えば、「所有者」、「権限ID」、「概要」。

■ SQL_OWNER_USAGE (ODBC 2.0)

所有者が使われることができる文を列挙する32bitのビットマスク。

SQL_OU_DML_STATEMENTS=所有者がすべてのデータ操作言語文でサポートされます。すなわち、選択、挿入、更新、削除、そしてもしサポートされるなら、更新のための選択と、文を更新、削除した箇所。

SQL_OU_PROCEDURE_INVOCATION=所有者がODBC手続き文でサポートされます。

SQL_OU_TABLE_DEFINITION=

所有者がすべてのテーブル定義文でサポートされます。テーブルを作成、視野作成、テーブル変更、テーブルドロップ、ドロップビュー。

SQL_OU_INDEX_DEFINITION＝

所有者がすべてのインデックス定義文でサポートされます：インデックス作成、ドロップインデックス。

SQL_OU_PRIVILEGE_DEFINITION＝

所有者がすべての特典定義文でサポートされます：承認と取り消し。

■ SQL_POS_OPERATIONS (ODBC 2.0)

SQLSetPosでサポートされたオペレーションを列挙している32bitのビットマスク。

次のビットマスクは、どの文がサポートされるかを決定するために使われています。

SQL_POS_POSITION
SQL_POS_REFRESH
SQL_POS_UPDATE
SQL_POS_DELETE
SQL_POS_ADD

■ SQL_PROCEDURE_TERM (ODBC 1.0)

プロシージャのデータソースのベンダー名を持っている文字列。例えば、「データベースプロシージャ」、「ストアドプロシージャ」、「プロシージャ」など。

■ SQL_PROCEDURES (ODBC 1.0)

文字列。すなわち、データソースがプロシージャをサポートし、ドライバがODBCプロシージャ構文をサポートしているなら"Y"で、そうでないなら"N"です。

■ SQL_QUALIFIER_LOCATION (ODBC 2.0)

限定されたテーブル名におけるクオリファイアのポジションを示している16ビットの整数値：

SQL_QL_START
SQL_QL_END

例えば、ディレクトリ（有資格者）名が、\EMPDATA\EMP.DBFのように、テーブル名の始めにあるので、XbaseライバがSQL_QL_STARTを返します。オラクルサーバドライバはSQL_QL_ENDを返します。なぜならクオリファイアは、 ADMIN.EMP@EMPDATAのように、テーブル名の終わりにいるからです。

■ SQL_QUALIFIER_NAME_SEPARATOR (ODBC 1.0)

文字列：データソースがクオリファイア名とそれに従うクオリファイア名の文字要素の間にセパレータとして定義する一文字、あるいは複数文字です。

■ SQL_QUALIFIER_TERM (ODBC 1.0)

クオリファイアのデータソースのベンダーの名前を持っている文字列、例えば、データベースあるいはフォルダです。

■ SQL_QUALIFIER_USAGE (ODBC 2.0)

クオリファイアが使うことのできるステートメントを列挙している32ビットのビットマスク。次のbitmasksはどこにクオリファイアを使うことができるか決定するために使われます：

SQL_QU_DML_STATEMENTS=クオリファイアはすべてのデータ取り扱い言語ステートメントでサポートされます：SELECT、INSERT、UPDATE, DELETE、サポートされるならSELECT FOR UPDATE、配置された更新、削除ステートメント

SQL_QU_PROCEDURE_INVOCATION=クオリファイアはODBC手続きを呼び出すステートメントでサポートされます。

SQL_QU_TABLE_DEFINITION=クオリファイアはすべてのテーブル定義ステートメントでサポートされます：CREATE TABLE、CREATE VIEW、ALTER TABLE、DROP TABLE、DROP VIEW

SQL_QU_INDEX_DEFINITION=クオリファイアはすべてのインデックス定義ステートメントでサポートされます：CREATE INDEX、DROP INDEX SQL_QU_PRIVILEGE_DEFINITION=クオリファイアはすべての権利定義ステートメントでサポートされます：GRANT、REVOKE

■ SQL_QUOTED_IDENTIFIER_CASE (ODBC 2.0)

次の通りの16ビットの整数値：

SQL_IC_UPPER=SQLでの引用された認証標識は大文字小文字の違いを無視し、大文字でシステムカタログにストアされます。

SQL_IC_LOWER=SQLでの引用された認証標識は大文字小文字の違いを無視し、小文字でシステムカタログにストアされます。

SQL_IC_SENSITIVE=SQLでの引用された認証標識は大文字小文字の違いを識別して、大小文字入り混ざった状態でシステムカタログにストアされます。

SQL_IC_MIXED=SQLでの引用された認証標識は大文字小文字の違いを無視して、大小文字入り混ざった状態でシステムカタログにストアされます。

■ SQL_ROW_UPDATES (ODBC 1.0)

文字列："Y"は、キーセットによって引き起こされているか、カーソルが入り混ざっているために行なわれるローのバージョン保守、あるいは全てのフェッチされたローのための値などならば、それに従って、ローが最後にフェッチされた時から、様々なユーザーによってローにされたどんな変更でも検出することができます。それ以外"N"です。

■ SQL_SCROLL_CONCURRENCY (ODBC 1.0)

コントロールオプションがスクロール可能なカーソルのためにサポートした同時発生を列挙している32ビットのビットマスク。次のビットマスクはいずれのオプションが

サポートされるか決定するために使われます：SQL_SCCO_READ_ONLY=カーソルは読み取り専用です。更新は許可されていません。SQL_SCCO_LOCK=カーソルが更新できることを保証するのに十分なロックをすることについて、最も低いレベルを使います。SQL_SCCO_OPT_ROWVER=カーソルがSQLBaseAE ROWIDのようなローのバージョンやSybase TIMESTAMPを比較して、効果的な同時発生制御を行ないます。SQL_SCCO_OPT_VALUES = カーソルは値を比較して、効果的な同時発生制御を行ないます。

- SQL_SCROLL_OPTIONS (ODBC 1.0) **情報タイプはODBC1.0で紹介されました；**

  それぞれのビットマスクはそれが紹介されたバージョンで分類されます。スクロール可能なカーソルのためにサポートされているスクロールオプションを列挙している32ビットのビットマスク。次のビットマスクはいずれのオプションがサポートされるかを決定するために使われます：

  SQL_SO_FORWARD_ONLY=カーソルはただ前方へスクロールするだけです（ODBC 1.0）。

  SQL_SO_STATIC=達成された結果のデータは静的です（ODBC 2.0）。

  SQL_SO_KEYSET_DRIVEN=ドライバはセーブして、そして達成された結果ですべてのローのためにキーを使います（ODBC 1.0）。

  SQL_SO_DYNAMIC=ドライバはrowset（keysetのサイズとrowsetのサイズはほぼ同じである）ですべてのローのためにキーを保持します（ODBC1.0）。

  SQL_SO_MIXED=ドライバはkeysetですべてのローのためにキーを保持します。keysetのサイズはrowsetのサイズより大きいです。カーソルは内部キーセットと動的な外部キーセットによって動かされています（ODBC 1.0）。

- SQL_SEARCH_PATTERN_ESCAPE (ODBC 1.0)

  ドライバがサポートするものとしてエスケープ文字の使用が許可されていて、パターンマッチのメタキャラクタであるアンダースコア（_）とサーチパターンとして有効なパーセント（%）という指定された文字列があります。このエスケープ文字は探索文字列をサポートするそれらのカタログ機能アーギュメントだけ適用します。この文字列が空であるなら、ドライバはサーチパターンとしてエスケープ文字をサポートしません。このfInfoTypeはカタログ機能に制限されます。エスケープ文字の使用の記述のためにサーチパターン文字列は、この章の以前のSearch Pattern Argumentsを見てください。

- SQL_SERVER_NAME (ODBC 1.0)

  実際のデータソース特定のサーバ名を持っている文字列。データソース名がSQLConnect、SQLDriverConnect、SQLBrowseConnect の間に使われるとき有用です。

■ SQL_SPECIAL_CHARACTERS (ODBC 2.0)

データソース上で、テーブル、カラム、あるいはインデックス名のような、オブジェクト名で使われることができるすべての特別な文字（a-z,A-Z,0-9,_を除く全ての文字）を含んでいる文字列。例えば、"#$^"。

■ SQL_STATIC_SENSITIVITY (ODBC 2.0)

32ビットのビットマスクを列挙することが静的であるか、あるいはkeysetによって引き起こされたカーソルへのアプリケーションによる変更であるか否かにかかわらずSQLSetPosあるいは配置された更新あるいは削除ステートメントを通してそのアプリケーションによって検出されることができます：SQL_SS_ADDITIONS=加えられたローはカーソルに見えます；カーソルはこれらのローにスクロールすることができます。これらのローがカーソルに加えられる場はドライバに依存します。SQL_SS_DELETIONS=削除されたローは、カーソルは既に利用可能ではなく、そして結果セットは残しておきません。カーソルがデリートされたローからスクロールした後、それはそのローに戻ることができません。

SQL_SS_UPDATES=ローへの更新はカーソルに見えます；もしカーソルがアップデートされたローからスクロールしてそれに戻るなら、カーソルによって返されたデータは、オリジナルのデータではなく、アップデートされたデータです。keysetによって引き起こされたカーソルでキー値を最新のものにすることが既存のローをデリートして、新しいローを加えていると考えられるから、この値は常にkeysetによって引き起こされたカーソルのために返されます。アプリケーションが他のユーザによって達成された結果によって行われた変更を検出することができるかどうかは、他のカーソルを含めて同じアプリケーションで、カーソルタイプに依存します。詳細な情報は、第7章Retrieving ResultsのScrollable Cursorsをご覧ください。

■ SQL_STRING_FUNCTIONS (ODBC 1.0) **情報タイプは ODBC1.0で紹介されました；**

それぞれのビットマスクがそれを紹介したバージョンで分類されます。スカラーの文字列を列挙している32ビットのビットマスクがドライバによってサポートされて機能して、そしてデータソースを関連づけました。次のビットマスクはいずれの文字列機能がサポートされるか決定するために使われます：

SQL_FN_STR_ASCII	ODBC 1.0
SQL_FN_STR_CHAR	ODBC 1.0
SQL_FN_STR_CONCAT	ODBC 1.0
SQL_FN_STR_DIFFERENCE	ODBC 2.0
SQL_FN_STR_INSERT	ODBC 1.0
SQL_FN_STR_LCASE	ODBC 1.0
SQL_FN_STR_LEFT	ODBC 1.0
SQL_FN_STR_LENGTH	ODBC 1.0
SQL_FN_STR_LOCATE	ODBC 1.0
SQL_FN_STR_LOCATE_2	ODBC 2.0
SQL_FN_STR_LTRIM	ODBC 1.0

SQL_FN_STR_REPEAT	ODBC 1.0
SQL_FN_STR_REPLACE	ODBC 1.0
SQL_FN_STR_RIGHT	ODBC 1.0
SQL_FN_STR_RTRIM	ODBC 1.0
SQL_FN_STR_SOUNDEX	ODBC 2.0
SQL_FN_STR_SPACE	ODBC 2.0
SQL_FN_STR_SUBSTRING	ODBC 1.0
SQL_FN_STR_UCASE	ODBC 1.0

アプリケーションがstring_exp1、string_exp2、でLOCATEをスカラーの機能と呼んで、そして議論を始めることができるなら、ドライバはSQL_FN_STR_LOCATEビットマスクを返します。アプリケーションがただstring_exp1とstring_exp2アーギュメントだけでLOCATEをスカラーの機能と呼ぶことができるなら、ドライバはSQL_FN_STR_LOCATE_2ビットマスクを返します。ドライバが完全にLOCATEのスカラーの機能をサポートするなら両方のビットマスクを返します。

■ SQL_SUBQUERIES (ODBC 2.0)

サブクエリをサポートする述語を列挙している32ビットのビットマスク：

SQL_SQ_CORRELATED_SUBQUERIES
SQL_SQ_COMPARISON
SQL_SQ_EXISTSSQL_SQ_IN
SQL_SQ_QUANTIFIED

■ SQL_SQ_CORRELATED_SUBQUERIES

ビットマスクがサブクエリをサポートするすべての述語がサポートするサブクエリを関連付けたことを示します。

■ SQL_SYSTEM_FUNCTIONS (ODBC 1.0)

スカラーのシステム関数を列挙している32ビットのbitmaskがドライバによってサポートされて機能して、データソースを関連づけられました。次のビットマスクはいずれのシステム機能がサポートされるか決定するために使われます：

SQL_FN_SYS_DBNAME
SQL_FN_SYS_IFNULL
SQL_FN_SYS_USERNAME

■ SQL_TABLE_TERM (ODBC 1.0)

テーブルのデータソースのベンダーの名前を持っている文字列、例えば、テーブルあるいはファイル。

■ SQL_TIMEDATE_ADD_INTERVALS (ODBC 2.0)

ドライバによってサポートされてタイムスタンプ間隔を列挙している32ビットのビットマスクとTIMESTAMPADDスカラー関数のための関連づけられたデータソースです。次のビットマスクはいずれの間隔がサポートされるか決定するために使われます：

SQL_FN_TSI_FRAC_SECOND
SQL_FN_TSI_SECOND
SQL_FN_TSI_MINUTE
SQL_FN_TSI_HOUR
SQL_FN_TSI_DAY
SQL_FN_TSI_WEEK
SQL_FN_TSI_MONTH
SQL_FN_TSI_QUARTER
SQL_FN_TSI_YEAR

■ SQL_TIMEDATE_DIFF_INTERVALS (ODBC 2.0)

ドライバによってサポートされてタイムスタンプ間隔を列挙している32ビットのビットマスクとTIMESTAMPDIFFスカラー関数のために関連づけられたデータソースです。次のビットマスクはいずれの間隔がサポートされるか決定するために使われます：

SQL_FN_TSI_FRAC_SECOND
SQL_FN_TSI_SECOND
SQL_FN_TSI_MINUTE
SQL_FN_TSI_HOUR
SQL_FN_TSI_DAY
SQL_FN_TSI_WEEK
SQL_FN_TSI_MONTH
SQL_FN_TSI_QUARTER
SQL_FN_TSI_YEAR

■ SQL_TIMEDATE_FUNCTIONS (ODBC 1.0) **情報タイプはODBC1.0で紹介されました；**

それぞれのビットマスクはそれが紹介されたバージョンで分類されます。スカラーの日付と時を列挙している32ビットのビットマスクがドライバによってサポートされて機能して、そしてデータソースを関連づけました。次のbitmasksはいずれの日付を決定するために使われ、時間関数がサポートされます：

ビットマスク	バージョン
SQL_FN_TD_CURDATE	ODBC 1.0
SQL_FN_TD_CURTIME	ODBC 1.0
SQL_FN_TD_DAYNAME	ODBC 2.0
SQL_FN_TD_DAYOFMONTH	ODBC 1.0
SQL_FN_TD_DAYOFWEEK	ODBC 1.0
SQL_FN_TD_DAYOFYEAR	ODBC 1.0
SQL_FN_TD_HOUR	ODBC 1.0
SQL_FN_TD_MINUTE	ODBC 1.0
SQL_FN_TD_MONTH	ODBC 1.0
SQL_FN_TD_MONTHNAME	ODBC 2.0
SQL_FN_TD_NOW	ODBC 1.0

SQL_FN_TD_QUARTER ODBC 1.0
SQL_FN_TD_SECOND ODBC 1.0
SQL_FN_TD_TIMESTAMPADD ODBC 2.0
SQL_FN_TD_TIMESTAMPDIFF ODBC 2.0
SQL_FN_TD_WEEK ODBC 1.0
SQL_FN_TD_YEAR ODBC 1.0

- SQL_TXN_CAPABLE (ODBC 1.0) **情報タイプはODBC1.0で紹介されました；**

  それぞれの戻り値はそれが紹介されたバージョンで分類されます。ドライバあるいはデータソースでトランザクションサポートを記述している16ビットの整数値：

  SQL_TC_NONE=トランザクションはサポートしていません（ODBC 1.0）。

  SQL_TC_DML=トランザクションがただ Data Manipulation Language（DML）ステートメント（SELECT、INSERT、UPDATE、DELETE）を含むことができるだけです。

  Data Definition Language（DDL）ステートメントがトランザクションで遭遇したデータがエラーを引き起こします（ODBC 1.0）。

  SQL_TC_DDL_COMMIT=トランザクションがただ DMLステートメントを含むことができるだけです。DDLステートメント（CREATE TABLE, DROP INDEX, an so on）が遭遇したトランザクションが原因でトランザクションがゆだねられました（ODBC 2.0）。

  SQL_TC_DDL_IGNORE=トランザクションがただDMLステートメントを含んでいるだけです。トランザクションで遭遇したDDLステートメントが無視されます（ODBC 2.0）。

  SQL_TC_ALL=トランザクションがDDLステートメントとDMLステートメントのどんなオーダーでも含むことができます（ODBC 1.0）。

- SQL_TXN_ISOLATION_OPTION(ODBC 1.0)

  ドライバあるいはデータソースから利用可能なトランザクション隔離レベルを列挙している32ビットのビットマスク。これらの隔離レベルの記述のために、SQL_DEFAULT_TXN_ISOLATIONの記述をご覧ください。次のビットマスクはいずれのオプションがサポートされるか決定するためにフラグと関連して使われます：

  SQL_TXN_READ_UNCOMMITTED
  SQL_TXN_READ_COMMITTED
  SQL_TXN_REPEATABLE_READ
  SQL_TXN_SERIALIZABLE
  SQL_TXN_VERSIONING

- SQL_UNION (ODBC 2.0)

  UNION節に対するサポートを列挙している32ビットのビットマスク：

  SQL_U_UNION=データソースはUNION節をサポートしています。

SQL_U_UNION_ALL=データソースはUNION節ですべてのキーワードをサポートします（この場合SQLGetInfoはSQL_U_UNIONとSQL_U_UNION_ALLの両方を返します）。

■ SQL_USER_NAME (ODBC 1.0)

ログイン名と異なり得る特定のデータベースで使われた、名前を持っている文字列。

コード例

SQLGetInfoはrgbInfoValueで32ビットのビットマスクとしてサポートされるオプションのリストを返します。それぞれのオプションのためのビットマスクはオプションがサポートされるかどうか決定するためにフラグと関連して使われます。

例えば、アプリケーションがサブストリングのスカラー関数がhdbcと結び付けられたドライバによってサポートされるかどうか決定するために次のコードを使うことができます：

```
UDWORD fFuncs;
SQLGetInfo(hdbc, SQL_STRING_FUNCTIONS,
(PTR)&fFuncs, sizeof(fFuncs), NULL);
if (fFuncs & SQL_FN_STR_SUBSTRING) /* サブストリングをサポートしています
*/...;else
/* サブストリングはサポートしていません */ ...;
```

関連機能

概要

接続オプションの設定を返す	SQLGetConnectOption (extension)
ドライバが機能をサポートするかどうか決定する	SQLGetFunctions (extension)
ステートメントオプションの設定を返す	SQLGetStmtOption (extension)
データソースについての情報を返す	SQLGetTypeInfo (extension)
データタイプ	

コンフィギュレーション　C

この章は各マシンにインストールを行うために必要とする、そしてシステムがサポートするすべてを記述します。

Windowsクライアント必要条件

Windows上で4D ODBCを使うための最小ソフトウェア必要条件は次の通りです：

- Windows95、Windows NT3.5、あるいはそれ以降
- 4th Dimension3.5、あるいはそれ以降、4D Engine3.5、あるいはそれ以降4D Runtime Classic3.5.3、あるいはそれ以降、4D Serverと4D Client1.5、あるいはそれ以降
- 少なくとも12MB RAM（Windows NTは16MB）を搭載しているx86 PCコンパチブルコンピュータ
- データソースの種類に対応した32ビットのODBCドライバ
  ドライバは通常このデータソースあるいはサーバを手に入れる会社によって供給されるか、あるいはこのタイプのミドルウェアを専門に扱っているベンダーによって供給されます。

Macintoshクライアント必要条件

Macintosh上で4D ODBCを使うための最小ソフトウェア必要条件は次の通りです：

- システム6.0.7
- 4th Dimension3.0.2 またはそれ以降、または4D Serverと4D Clientバージョン1.0.2またはそれ以降
- データソースの種類に対応したODBCドライバ
  ドライバは通常このデータソースあるいはサーバを手に入れる会社によって供給されるか、あるいはこのタイプのミドルウェアを専門に扱っているベンダーによって供給されます。
- （Visigenicあるいはアップルによって提供された）VisigenicからのODBCドライバマネージャ

コマンド索引

E

F

G

I

L

M

N

O

P

Q

S

T

U