4D ODBC

リファレンス
Mac™OS and Windows® 版

4D ODBC
by
Olivier Boulland

4D ODBC リファレンス
Mac™OS and Windows® バージョン 1.5

Copyright© 1996 ACI SA/ACI US, Inc.

All rights reserved

このマニュアルで説明されるソフトウェアは、本製品の 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。このソフトウェアおよびマニュアルは著作権にて保護されており、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、ソフトウェアおよびマニュアルの一部または全部を複製することはできません。電子的媒体の複製や、本製品の License Agreement で認められた以外のいかなる方法でこのソフトウェアを保管、または使用することもできません。

4th Dimension、4D、4D ロゴ、4D Server、4D Runtime、4D Compiler、4D Write、4D Calc、ACI、ACI ロゴは、ACI SA の登録商標です。

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。

重要
このソフトウェアの使用には、本製品に同梱の License Agreement（使用許諾契約書）に同意することが必要です。ソフトウェアの使用の前に、License Agreement を注意深くお読みください。

目 次

第 1 章　序章

4D ODBC は、4th Dimension の外部ルーチンのひとつであり、Macintosh や Windows 上で動作する 4th Dimension データベースと ODBC データベースとの通信を可能にします。4D ODBC により、使用している 4th Dimension データベースから、ODBC データベースに保存されたデータの表示、操作、修正を行うことができます。

このマニュアルについて

このマニュアルでは、4th Dimension データベースを使い、ODBC を介してアクセス可能なデータソースを実行、操作、更新する方法について説明します。

このマニュアルは、4th Dimension 言語および ODBC の SQL 言語をすでに習得しているユーザを対象としています。初心者の方は、このマニュアルをお読みになる前に、これら 2 つの製品を習得されることをお勧めします。

このマニュアルは、主に 3 つの部分に分れています。

- パート I :「はじめに」では、4D ODBC の主な概念を説明し、フロントエンドアプリケーションの開発時に必要となる設計上の選択についていくつか説明します。
- パート II :「ユーザリファレンス」では、データ管理に関する手法について説明します。第 1 の方法として、コンテキストを使用し、4th Dimension のデータをデータソースのデータにリンクします。第 2 の方法では、low level コマンドを使用し、データソース側へ SQL 文を送信し、それを実行します。
- パート III :「ランゲージリファレンス」では、4D ODBC 言語の各コマンドを説明します。

クロスプラットフォーム

このマニュアルでは、Macintosh および Windows 上で動作する 4D ODBC を対象として説明を行います。Macintosh 版および Windows 版 4D ODBC は、ほぼ同一の機能と概念を備えていますが、このマニュアルでは必要に応じて、それぞれの違いについても触れています。これらの相違点には、グラフィカルユーザインタフェースやキーボードコマンドなどが含まれます。

このマニュアルでは、Macintosh および Windows 環境を表わす図も示されます。

4th Dimension と 4D ODBC

4D ODBC は、4th Dimension および 4D Server で使用することができます。4D ODBC により、ODBC データソースのクライアントとなるデータベースを 4th Dimension から作成することができます。このクライアントデータベースの各ユーザは、同時に ODBC データベースへ接続し、使用することができます。

4D Server では、マルチ開発用のデータベースを作成できます。4D ODBC を使うことにより、4D Server で複数の開発者が ODBC データベースへの接続を行へます。

このマニュアルでは、動作に異る点がない限り、4th Dimension と 4D Server の 2 つの製品はともに 4th Dimension として表現されています

理解を深めるために

このマニュアルには、内容を一層深く理解できるように、一定の表記を使用しています。

次のような表記を使用しています。

注：*4th Dimension を効率よく使用できるように、このような強調文で注釈や近道を提供します。*

このような注意書きは、重要な情報に対する注意を促しています。

警告：*このような警告は、データが失われる可能性のある状況に対する注意を促しています。*

コマンド

このマニュアルでは、4D ODBC コマンドはすべて特定のフォントを使い、大文字で表記します（例：**OC DEBUG MESSAGE**）。4th Dimension に組み込まれた 4D ODBC コマンドはすべて、OC という文字で始まります。

関数

4D ODBC の関数は、最初の文字が大文字で表記されます（例：**OC Set SQL in cursor**）。

テーブル名

フィールド名、レイアウト名、および他の項目名と区別するため、本文中でテーブル名は角括弧で囲まれています。たとえば、会社テーブルは [会社] テーブルと表わされます。

第 2 章　はじめに

このマニュアルは、統合 4D ODBC のデザイナ、管理者、ユーザのためのリファレンスガイドです。このマニュアルでは、ODBC データベースの全体的なアーキテクチャや機能、および ODBC ドライバで利用可能な 4th Dimension のプログラミング言語と機能について、ユーザの方が理解していることを前提としています。

4th Dimension は、Macintosh および Windows 対応のパワフルなデータ管理用ツールです。4D ODBC を使用して開発されたアプリケーションは、使い易いグラフィカルインタフェースを備えた強力なリレーショナルデータベースです。

4D ODBC により、4th Dimension および ODBC データソースの利点を活かしたアプリケーションの開発が可能になります。4D ODBC を使い、SQL データベースに保存しているデータを 4th Dimension からアクセスすることができます。

ODBC アーキテクチャ

ODBC（Open Database Connectivity）は、各種関数の納められたライブラリで、これらの関数により、SQL 言語（Structured Query Language）を使用して 4th Dimension のようなアプリケーションからデータベース管理システム（DBMS）へのアクセスが可能になります。ODBC インタフェースは、異なるデータベース管理システムに対して、ベンダに制限されないアクセスを提供します。

ODBC アーキテクチャは 4 つの要素で構成されます。

- アプリケーション
- ドライバマネージャ
- ドライバ
- データソース

ODBC ドライバから提供される主要な機能は次の通りです。

- DBMS への接続と接続解除
- クエリの実行とクエリ結果の保管場所とデータ形式の提供
- オンライントランザクション処理の準備
- ODBC インタフェース外部の機能（DBMS 特定の機能）
- ドライバマネージャはドライバをロードする DLL（dynamic linked library）であり、各種ドライバ用に ODBC 関数への単一のエントリポイントを提供する

接続の選択

4D ODBC アプリケーションを設計する際、まずはじめに接続するデータベースを決定します。ODBC からは、利用できるデータベースの一覧や各データベースタイプの説明、それらのデータベースとの接続などの各種機能が提供されます。

目的とする特定のデータベースを対象にアプリケーションを設計することができます。たとえば、経理部は ORACLE データベースにレコードを保存しているものとしましょう。仕入注文システムを設計する場合、ORACLE ドライバが必要であることがわかります。また、データベースのタイプやこのデータベースへの接続に必要な属性もわかります。目的のデータソース用にアプリケーションを設計する際、DBMS およびドライバから提供される特定の機能を利用することが可能です。

一方で、あらゆるデータベースを使って作業できるようなアプリケーションを設計することが必要な場合もあります。使用するドライバや接続するデータベースはあらかじめわかっていません。この場合、開発者は全 ODBC データソースに共通の機能だけを利用するように配慮しなくてはなりません。

4D ODBC により、開発者はアプリケーションを開発する際に、どちらのケースにも対応することができます。

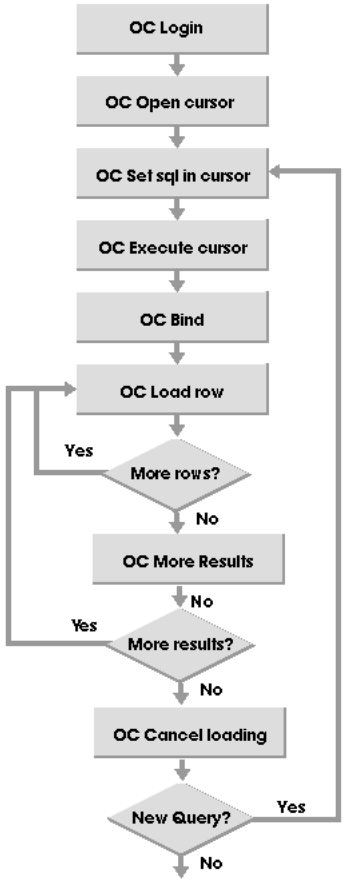
第 3 章　Low Level コマンドとコントロールコマンド

この節では、4D ODBC において low level コマンドやコントロールコマンドを使用し、データソースへアクセスする際に重要となる事柄について説明します。これらのコマンドは、名前や構文、機能の点でネイティブな ODBC の API コールと非常に似ています。ODBC の処理や機能性について詳細に分析することはこの節の目的ではありません。

ODBC の各関数は 7 種類に分類されます。これらのプロシージャグループにより、通信処理のあらゆる段階でデータソースとの対話が実現します。

- 接続
- カーソル
- SQL 文の実行
- 処理結果の受信
- 処理のキャンセル
- エラー処理
- トランザクション制御

次のチャートは、4D ODBC ルーチンを使用した際に、4th Dimension とデータソースの間でどのようなやり取りが行われるかを示したものです。



接続

ODBC データソースへの初期接続は、ログインコマンドにより処理されます。これらのコマンドは、データソースへの接続を開始し、渡されたユーザ名とパスワードを使ってデータソースへログインします。

4D ODBC には 2 種類の接続開始方法があります。

- OC Login dialog
- OC Login

OC Login dialog 関数は、ダイアログボックスを表示します。このコマンドを使用して、データソースを選択し、ログイン名およびパスワードを指定することができます。

OC Login 関数を使用すると、ユーザ名、パスワード、データソース名を送信することができます。このコマンドにより、必要な値をハードコードしたり、独自のインタフェースを作成してユーザに値の入力を要求することができます。

同じデータソース、または異なるデータソースへの接続を複数同時にオープンすることも可能です。

カーソル

いったん接続が確立されると、4th Dimension と ODBC データソースの間でダイアログを表示することが可能になります。これら 2 アプリケーション間の通信を制御する高度な仕組みがカーソルです。

カーソルを使うと、クエリーの結果として返された各ローをひとつずつ処理することができます。カーソルは、ODBC データソース上のプロセスであると考えられます。作成した各カーソルは、後で ODBC データソース上のデータの選択、挿入、更新、削除を行う SQL 文と組合わされます。同時に複数のカーソルを作成し、使用することができますが、カーソルを数多くオープンすると、パフォーマンスに悪影響を与えることがありますので注意してください。

プロセス確立の所要時間は、以下の状況に応じて変わってきます。

- 使用する OS
- データサーバ構成
- サーバへの接続ユーザ数

4D ODBC のルーチンは、異なる 2 つの方法でカーソルの使用を管理します。第 1 の方法ではカーソルを個別に作成し、第 2 の方法ではカーソルが既に存在することが前提となります。

SQL 文の実行

SQL 文の実行メソッドは、プリペアードとダイレクトの 2 種類です。プリペアード実行メソッドは、何度も SQL 文をサブミットするようなアプリケーションを作成する場合に使用します。プリペアード実行メソッドでは、引数の値を変更する可能性があります。この方法では、クエリーの実行時に使われる最適化情報が保存され、ルーチンを実行するたびに再解析を行う必要がありません。したがって、同じ SQL 文を何度も実行する場合には効果的な方法です。

SQL リクエストの完了前に、アプリケーションで結果セットの情報が必要ではない場合や、SQL リクエストのサブミットを一度しか行わないようなアプリケーションなら、ダイレクト実行メソッドがよいでしょう。

プリペアード SQL 文の使用

ODBC の使用時、SQL 文の引数に値を渡すことにより、SQL 文をダイナミックに組み立てることができます。ダイナミック SQL 文は、SQL 文の引数マーカーに値を埋め込むことにより作成されます。SQL 引数マーカーとは、クエスチョンマーク（？）のことであり、ホスト側変数として使用されます。

同じ SQL 文を何度も実行する必要がある場合には、ダイナミック SQL 引数と OC Set SQL、OC Execute cursor を組み合わせて使用します。

ダイナミック SQL の利点はさまざまですが、たとえば次のような事柄があげられます。

- 引数がわからない場合でも SQL コマンドを組み立てられる。
- 代替変数を使用している SQL コマンドで、その値を変更する場合に、OD Set SQL in cursor により再解析を行う必要がない。
- 実行するたびに SQL 最適化情報が保存される。

次のプロシージャでは、SQL の insert 文を作成するために、入力用引数として配列の変数を使用しています。

▼ この例題は、配列等の 4D インタフェースと、ODBC の SQL 機能とを組み合わせることによって最適化を図っています。

```
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
```

ダイレクト SQL 文の使用

OC Execute direct cursor は、OC Set SQL in cursor と OC Execute cursor の機能を組合せ、単一のルーチンにしたものです。特定の SQL 文を作成しても、それを一度しか呼び出さないような場合、OC Execute direct cursor が効果的です。DBMS では SQL 文の実行後、使用した最適化情報をすべて破棄してしまいます。

▼ 次の例題では、ダイレクト SQL 文を使用して SQL の delete コマンドを実行しています

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

結果の受信

OC Bind は、データソースからダイレクトにデータを取得します。4D ODBC ルーチンの大部分は、OC Bind をベースにしています。このコマンドは、4th Dimension オブジェクトにデータソースのオブジェクトをバインドします。この関数を使用し、テーブル X のカラム A を 4th Dimension データベースの変数やフィールドにバインドすることができます。

OC Bind は、ODBC 側の SQL カラムと 4th Dimension 側のフィールドや変数や配列のように、2 つのオブジェクト間のバインド定義を行います。オブジェクト間のバインドを設定した後は、OC Load row 関数を使い、ODBC データソースから選択した値を対応する 4th Dimension オブジェクトへ転送することができます。OC Load row 関数は、さらに結果が存在するかどうかを示す値を戻します。

データを取得するまでの処理は、次の 3 フェーズに分かれます。

- コマンド送信・実行
- 結果の送り先を割り当てる
- 結果の取得

OC Bind 関数を使い、4th Dimension の 3 種類のオブジェクトに SQL Server のカラムをバインドさせることができます。

- 変数
- 配列
- フィールド

変数へのバインド

変数へのバインドは、データローを一度にひとつずつ取り込むのに適した方法です。たとえば、ユーザが選択した特定のローの詳細情報を取得する場合などです。この方法は最も簡単ですが、結果が複数ローに渡る場合に、それを保存したり再使用することはできません。メッセージ作成や、ローの表示、レコードへのデータの保存時に一時的に保管する目的にも、この方法は適しています。

▼ 例題

```
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)
```

配列へのバインド

配列 へのバインドは、リストで使用するため複数行のデータローを取り込む際に適した方法です。4D コネクティビティを開発するディベロッパの多くは、4D の出力レイアウトに相当するものを作成する場合にこの方法を使っています。4D インタフェースツール（スクロールエリア、リスト、ポップアップメニューなど）に対応する場合にも、この方法は適しています。たとえば、より読み取りやすい方法で、ロジカルかつユニークなキーの要素を続けて取り込み、それを書き写すために使用することができます。

配列へのバインドを行う際、あらかじめ **ARRAY STRING** や **ARRAY INTEGER** などのコマンドを使い、その配列を定義しておく必要があります。最も適した 4th Dimension のデータタイプを用いて、配列を定義してください。

配列へのバインドを行う際、OC Load row を何度もコールすると、4D ODBC により新しい配列要素が自動的に追加され、追加された配列要素に値がロードされます。

可能なかぎり、テキスト配列 (**ARRAY TEXT**) ではなく固定長配列 (**ARRAY STRING**) を使用してください。

▼ 例題

```
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
```

配列に 1セットの結果を直接代入することだけを目的とするなら、OC Query exec を使用することができます。OC Query exec は、さらに優れた方法で SQL Server データを使用して配列をロードします。また、OC Query exec は “ ダイレクト ” メソッドにより SQL コマンドを実行します。

▼ 例題

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

フィールドへのバインド

4th Dimension のフィールドを直接利用すると、後でレポートやラベルを作成する時に使用するデータの中間的な保管場所となります。この場合、必ずプログラムを用いてレコードの登録管理を行ってください。

```
$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

結果を返す SQL コマンドを実行した後、4D ODBC ルーチンでその結果の詳細を分析することができます。

OC Number rows processed や OC Number of columns を使用し、あるコマンドによって処理されたローの数や、結果セット中の選択されたカラム数を調べることができます。

OC Describe column や OC Column attributes により、結果セットに取り込んだカラムの属性情報を知ることができます。

おそらくユーザにとって最も有益な結果は、OC Load row 関数から戻されたものでしょう。OC Load row は、さらに取り込むべきローがある場合に 1 を戻し、取り込むべきローがもう存在しない場合には 0 を戻します。この情報を利用し、結果セットからすべてのローを返すようなプロシージャを作成することも可能です。

▼ 次の例題は、OC Load row を使い 4D 配列へローをロードしています。while ループにより、ローがまだ存在するかチェックしている点に注目してください。

```
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
```

コマンドの分析

SQL 文は 2 つのカテゴリーに分類されます。

- 単一のステートメントからなるコマンド。(1 つの結果セット)
- ストアドプロシージャのように、実際には複数の SQL 文で構成される複合コマンド。(複数の結果セット)

OC More results を使って SQL コマンドを分析し、さらに取り込むべき結果セットがあるかどうかの判断を行うことができます。

アプリケーションで次のセットに移る前に、1 つのセット内の結果をすべて取り込む必要はありません。しかし、OC More results がコールされると、前の結果セットは破棄されます(OC Cancel loading と同様)。したがって、この関数はカレント結果セットから必要なローをすべてロードした後に使用してください。

ユーザに独自の SQL 文を入力させるアプリケーションの場合、プロシージャをコールする前には、そのプロシージャにいくつコマンドが含まれているのかわからないことがあります。このような場合、必ず OC More results を使用してデータソースコマンドバッファを読み込み、実行の完了、およびコマンドが存在しないことを確認してください。

▼ 次のプロシージャでは、SQL Server のストアドプロシージャを実行して、すべての結果セットより最初の 3 カラムを返し、返された結果の数をアラートボックスに表示します。OC Load row と OC More results から返された結果を使用すると、取得したローや結果セットの数に関わらず、どのようなコマンドの結果でも取り込めるような汎用的なルーチンを作ることができる点に注目してください。

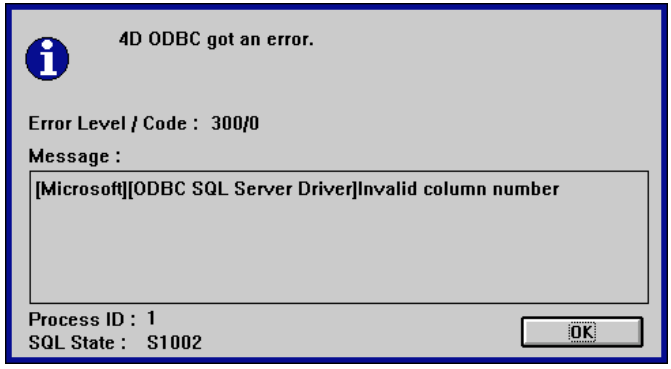
```
$sql:="exec sp_help titles"
$res:=OC Set SQL in Cursor (cursor_id;$sql)
  `配列へのバインド
$res:=OC Bind (cursor_id;1;"ar1")
$res:=OC Bind (cursor_id;2;"ar2")
$res:=OC Bind (cursor_id;3;"ar3")
$res:=OC Execute cursor (cursor_id)
$count:=1
$more:=1
While ($more=1)
  $res:=1
  While ($res=1)
    $res:=OC 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")
```

操作のキャンセル

4D ODBC で実行した操作はすべて、OC CANCEL LOADING 関数をコールすることによりキャンセルできます。OC CANCEL LOADING は、最後に実行されたクエリーの結果ローのうち、未処理のローをキャンセルします。

エラー処理

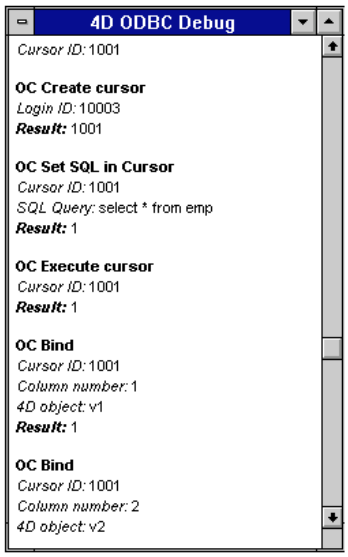
4D ODBC のエラー管理モードでは、標準のエラー管理プロシージャを使用しています。このプロシージャはバックグラウンドで稼働し、以下のようなダイアログボックス上にエラーメッセージを表示します。



このダイアログボックスには、エラー番号、エラーレベル、エラーメッセージが表示されます。また、4th Dimension のプロセス ID と、SQL エラーステータスも表示されます。4D ODBC のエラー番号およびエラーメッセージの一覧は、このマニュアルにある付録 A を参照してください。

4D ODBC の制御コマンドである OC SET ERROR HANDLER コマンドは、エラー処理用プロシージャを呼び出し、エラーが発生するたびこのプロシージャを実行します。したがって、自分で実行時エラーを制御し、デフォルトエラー処理を無効にすることが可能です。エラー処理用プロシージャを使用すると、独自のエラー処理ルーチンを作成したり、発生したエラーをローカル上の 4D データベースに記録できるようにするメッセージを設け、後でそれを分析することができます。

開発中、4D ODBC のデバッグウインドウを使い、作成した 4D ODBC ルーチンの処理をトレースすることができます。



デバッグウインドウを使い、コールされるたびに各ルーチンを記録したり、各ルーチンの引数や結果を表示することができます。「Trace on error」オプションを指定すると、4D ODBC のエラーが発生するたび自動的に 4D のトレースモードが起動され、コードの検証が行えます。

トランザクション制御

トランザクション制御により、トランザクションのコミットやロールバックを実行できます。ODBC のデフォルトモードは「自動コミット」モードであり、各 SQL 文は完結したひとつのトランザクションとなります。

しかし、OC Set login option 関数を使用すると、「手動コミット」モードが可能になるように接続オプションを設定することができます。「手動コミット」モードでは、OC TRANSACT COMMAND コマンドを使用し、自分でトランザクションをコミットする必要があります。

第 4 章　コンテキストの使用

コンテキストとは、4D ODBC オブジェクトと ODBC データソースオブジェクト間のダイナミックなリンクのことです。データを操作するためにコンテキストを使用する場合、サーバ上のエントリに対する選択、挿入、更新、削除の操作はローカルマシン上の 4D ODBC の処理と同じ方法で実行されます。ホストデータの管理には 4D ODBC のインタフェースが利用されます。

この節では、コンテキストを使用する利点および使用方法について簡単に説明します。また、コンテキストを使ったプロシージャの流れ図（フロー）を示し、次にコンテキストの作成方法、およびコンテキストを使用したデータの取得方法について説明します。

コンテキストコマンドの詳しい説明は、このマニュアルの第 8 章を参照してください。

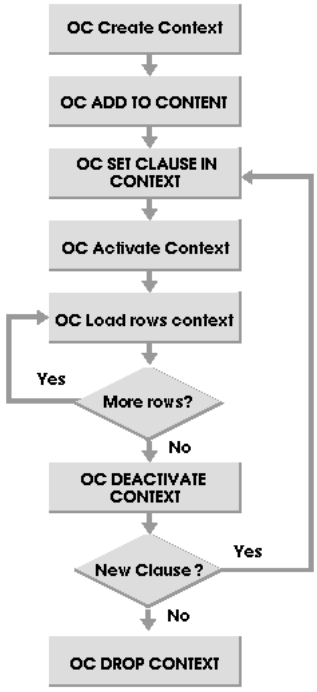
コンテキストの利点

ローレベルの DB-Library コールだけを使用しても 4D ODBC アプリケーションの開発は可能ですが、コンテキストを使用することにより、開発者にとって便利な点がいくつかあります。

- **データセレクション：**コンテキストにより、SQL プログラミングを使わなくてもデータソースからデータを選択することができます。コンテキストのコードは、グラフィカルなインタフェースを使用して作成することができます。
- **データ修正：**コンテキストを設定した後は、面倒な SQL プログラミングではなく、簡単なコマンドを使ってデータソースのローの更新、挿入、削除を行うことができます。
- **SQL 互換性:**コンテキストは通常の SQL や ODBC ルーチンとともに使用可能です。
- **複数のコンテキスト：**同時に複数のコンテキストをオープンした場合、非常に柔軟なクライアント／サーバアプリケーションを設計することができます。
- **4 th Dimension 風のプログラミング：**コンテキストを使用することにより、4D ODBC プログラミング環境に統合された SQL 開発環境が提供されます。この理由から、経験を積んだ 4D ODBC 開発者にとって、Transact SQL を使用したアプリケーションをインプリメントするより、コンテキストを使用する方がより簡単にアプリケーションを開発できるでしょう。

コンテキストフェーズ

次の図は、コンテキスト管理コマンドの使用を表わしています。



コンテキストコマンドの大部分は、次のようなフェーズと関連付けることができます。つまり、コンテキスト定義、コンテキスト選択、コンテキスト訂正という 3 つのフェーズです。

コンテキスト定義

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際に使用されます。この初期化の結果は、コンテキスト ID として特定コンテキストの選択、訂正を管理するために引き続き使用されます。

コンテキストの定義には次のコマンドを使用します。

- OC Create context
- OC Create context dialog
- OC Open context file
- OC ADD TO CONTEXT

コンテキストを定義する際に、コンテキストを設定するデータソースのテーブルを必ず指定してください。

コンテキストの設定に、OC Create context を利用する場合、OC ADD TO CONTEXT コマンドを使って、4D ODBC オブジェクトに選択したデータカラムを

バインドしておく必要があります。OC ADD TO CONTEXT コマンドは、データオブジェクトのソースと割り当て先を指定します。

コンテキストの定義に関する詳細は、この節の最後にある「コンテキストの作成」の項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のコマンドは、データソーステーブルのローを特定し、取り込みます。次のコマンドを使い、データセレクションを作成します。

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

バインドを設定した後で、句を定義することができます。句を使うことにより、データソースから選択するデータを指定できます。コンテキストによりサポートされる句は、Where 句、Group by 句、Having 句、Connect By 句、Start with 句です。これらの句はいつでも変更することができます。

コンテキストのアクティブ化

コンテキストを定義し、選択した句を追加したら、データの取得や修正を行う前に、コンテキストをアクティブにする必要があります。実際にコンテキストをアクティブにするには、データソースへ SELECT 文を送信します。コンテキストに関連する句を変更するには、まずコンテキストを非アクティブにしてください。

次のコマンドを使い、コンテキストをアクティブ／非アクティブにします。

- OC Activate context
- OC DEACTIVATE CONTEXT

コンテキストのデータ取得

コンテキスト定義および句をアクティブにしたら、次のコマンドを使ってデータを取り込みます。

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

OC Load rows context コマンドは、コンテキストで要求されただけの数のローを返します。

コンテキストのデータ修正

コンテキストを定義し、指定したローを取得したら、次のコマンドを使用してレコードの更新、削除、挿入を行うことができます。

- OC Update in context
- OC Insert in context
- OC Delete in context

コンテキスト内のデータを削除、または更新するには、データソーステーブル上の論理的かつユニークなキーを少なくともひとつは定義してください。4D ODBC では、コンテキストが「Modifiable」として宣言されている場合に、定義されたユ

ニークなキーを基に検索を行います。キーが指定されていないと修正は行えません。

これらのコマンドに関する詳細は、このマニュアルの第８章を参照してください。

コンテキストの作成

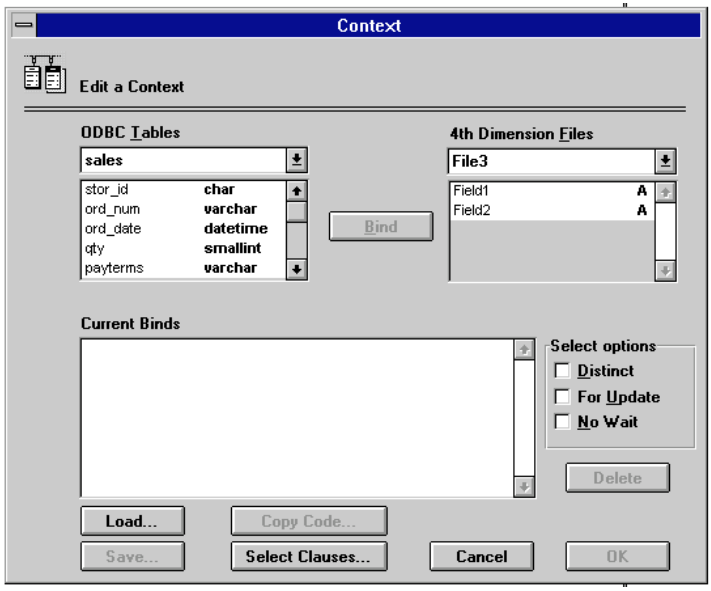
コンテキストの作成には、コンテキスト定義ダイアログボックスを利用する方法と、プロシージャ上で一連の文を実行する方法のどちらでも使用できます。ダイアログボックスを使用する場合、4D ODBC によりコンテキスト定義文が生成されます。

ダイアログボックスを使用したコンテキスト作成

「Edit a Context」ダイアログボックスを使い、データソースサーバ上のデータを選択するコンテキストを定義し、それを 4th Dimension のデータ要素にリレートさせることができます。

「Edit a Context」ダイアログボックスを開くには、次の手順に従ってください。

- **関数** OC Create context dialog **を実行します。**
  「Edit a Context」ダイアログボックスが表示されます。



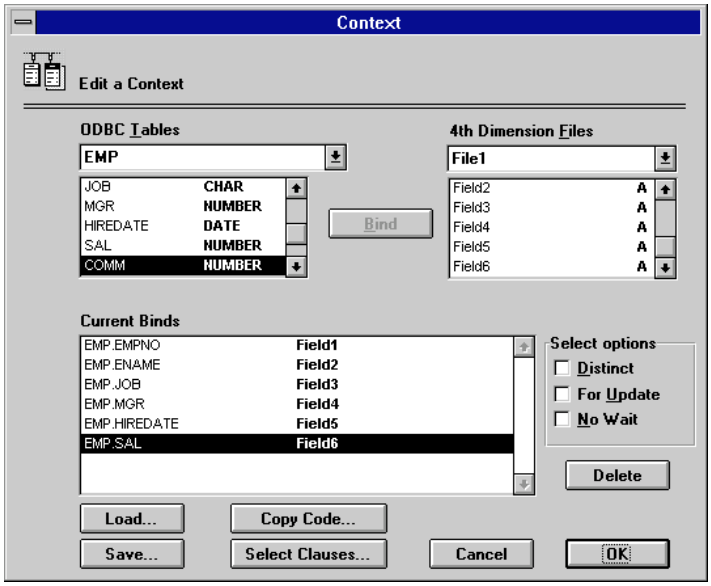
このダイアログボックスで実行できるのは次の事柄です。

- コンテキストへのバインド定義
- コンテキストへの句やオプションの追加
- コンテキスト定義用のコードをクリップボードへコピー
- コンテキスト定義をファイルに保存
- ファイルからコンテキスト定義をロード

バインド定義

「Edit a Context」ダイアログボックスでは、データソースのカラムや式を選択し、4th Dimension のフィールド、変数、配列を選択してこれらをバインドすることができます。

バインドを複数設定し、複数のデータソースオブジェクトと複数の 4th Dimension オブジェクトとを関連付けることができます。設定した各バインドは、「カレントバインド」エリアに一覧表示されます。



データソースオブジェクトと 4th Dimension オブジェクトとのバインドを設定するには、「Edit a Context」ダイアログボックスを使い、次の手順に従ってください。

1 **「ODBC Table」ポップアップメニューの下にあるリストから、カラムまたは式を選択します。**

別のテーブルのカラムを表示するには、「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 Files」ポップアップメニューの下にあるリストから、4th Dimension のフィールドを選択します。**

別のテーブルのフィールドを表示するには、「4th Dimension Tables」ポップアップメニューからテーブル名を選択します。

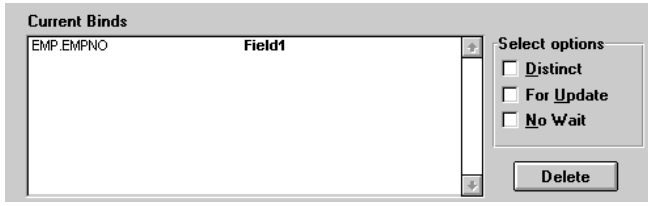
これでバインドしようとする ODBC オブジェクトと 4th Dimension オブジェクトの選択ができました。このバインドをコンテキストに加える前に、3 つのオプションを選択することができます。

3 **「Bind」ボタンをクリックします。**

コンテキストのバインド設定が完了しました。コンテキストをアクティブにするまで、このバインドはアクティブになりません。

4 コンテキストにバインドを更に定義するには、1 ～ 3 の手順を繰り返します。

すべてのバインドは、そのバインド（更新可、ソート済み、プライマリーキー）に対して選択したオプションとともに「Current Bind」エリアに表示されます。次の図は、カラム EMPNO がフィールド Field1 にリンクしていることを表わしています。

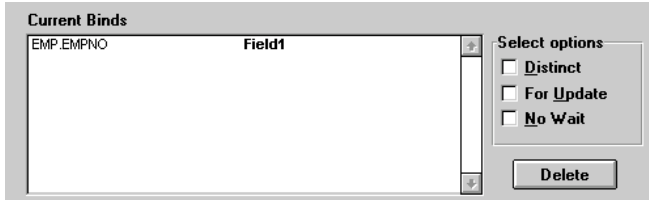


「Current Bind」エリアのバインドを選択し、「Delete」ボタンをクリックすることにより、いつでもバインドを削除することができます。

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にする前に、オプションや句を指定し、コンテキストの働きに変化をもたせることができます。コンテキストオプションを選択するには、次の手順に従ってください。

■ **「Select Options」エリアでコンテキストに対するオプション（複数可）を選択します。**



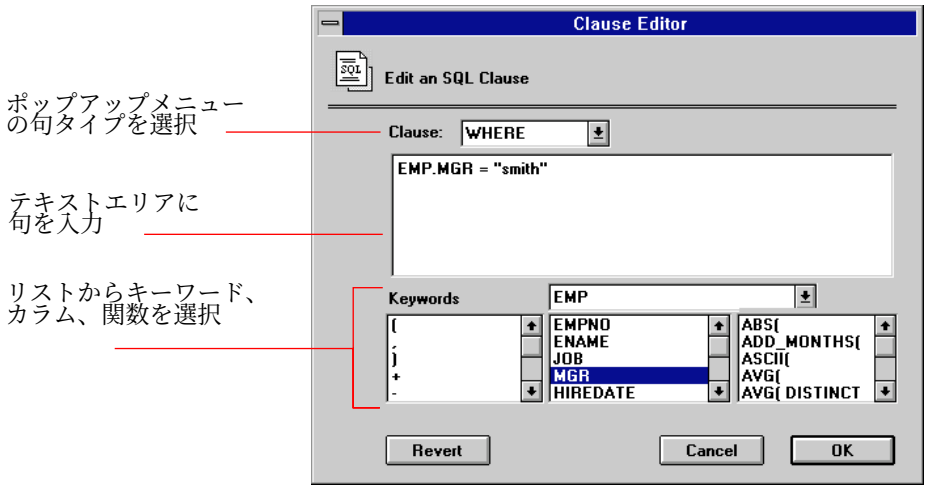
- **Distinct オプション：**このオプションは、個別の値が返されるようにします。これにより、重複した値が返されないようにすることができます。
- **For Update オプション：**このオプションは、他の全ユーザに対し選択したローをロックし、更新の準備を行います。
- **No Wait オプション：**このオプションを指定すると、4D ODBC は SELECT 文で選択されたローがロック解除されるまで待機しません。ローがロックされている場合、クエリーはキャンセルされます。「No Wait」オプションを指定する前に、必ず「For Update」オプションを選択しなければなりません。

コンテキストに句を追加する

コンテキストに句を追加するには、次の手順に従ってください。

1 **「Select Clauses...」ボタンをクリックします。**

「Edit a SQL Clause」ダイアログボックスが表示されます。このダイアログボックスを使い、WHERE、GROUP BY、CONNECT BY などの句を作成することができます。



作成する句のタイプを「Clause」ポップアップメニューから選択することができます。

句を作成するには、テキストエリアに句を入力します。ダイアログボックスの下部にあるリストから、キーワードやカラム名、関数を選択することにより、簡単に句を作成できます。

中央のリストに別のテーブルのカラムを表示したい場合、ポップアップリストを使い、表示されたポップアップメニューからテーブル名を選びます。または、タイトルバーの両側にある矢印を使い、テーブルリストをスクロールします。

コンテキスト作成文のコピー

通常、コンテキストは開発者用のツールとしてグラフィックを使い定義されます。アプリケーション開発速度を向上させるため、4D ODBC がコンテキスト定義に相当する 4th Dimension ステートメントを生成するよう要求することができます。コンテキストの使用が必要になる度、コンテキストの再定義を行う代わりに、4th Dimension のプロシージャ内に 4D ODBC ステートメントを置くことができます。

コンテキストをアクティブにする前に文をクリップボードへコピーするには、次の手順に従ってください。

1 **「Copy Code...」ボタンをクリックします。**

このボタンをクリックすると、コンテキスト ID を格納する 4th Dimension の変数名が 4D ODBC から要求されます。

2 **変数名を入力し、「OK」ボタンをクリックします。**

文がクリップボードにコピーされます。この後、4th Dimension のプロシージャにこれらの文をペーストします。

コンテキスト定義をファイルへ保存する

4D ODBC ではコンテキスト定義をファイルに保存することができます。ファイルに保存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可能です。また、関数 OC Open context file を使い、ファイルに保存されたコンテキストをメモリー上にロードすることもできます。

作成したコンテキストをファイルに保存するには、次の手順に従ってください。

1 **「Save As...」ボタンをクリックします。**
「Save File」ダイアログボックスが表示されますので、ファイル名と保存場所を指定します。

2 **「Save」ボタンをクリックします。**
ファイルが保存されます。

ファイルからコンテキストをロードする

ファイルへ保存する前にコンテキストをロードするには、次の手順に従ってください。

1 **「Edit a Context」ダイアログボックスから「Open...」ボタンをクリックします。**
「Open File」ダイアログボックスが現れます。

2 **ファイルを選択し、「Open」ボタンをクリックします。**
コンテキストがロードされ、「Open File」ダイアログボックス上に表示されます。

関数 OC Open context file を使用して、プロシージャからコンテキストをロードすることもできます。

プロシージャを使用したコンテキスト作成

「Edit a Context」ダイアログボックスを使いたくない場合、4D ODBC のコマンドや関数を利用してプロシージャからコンテキストを定義できます。

プロシージャからコンテキストを定義する基本手順は、次のとおりです。

1 **コンテキストを初期化した後に、関数** OC Create context **を実行してコンテキスト識別子を取得します。**
コンテキスト識別子は、その他すべてのコンテキストコマンドにおいてコンテキストを識別するために使われます。

2 **一連の** OC ADD TO CONTEXT **文を実行し、コンテキストを定義します。**
OC ADD TO CONTEXT コマンドは、データソースオブジェクトと 4th Dimension オブジェクトとをバインドし、UPDATABLE、SORT、PRIMARY KEY オプションに相当する機能を付加します。

この時点で、4D ODBC により作成された SELECT 文は、次のようになります。

```
SELECT empno, ename,… FROM emp
```

3 **一連の OC SET CLAUSE IN CONTEXT 文を実行し、コンテキスト定義に句を追加します。**
部門 20 の従業員だけを参照したい場合、次の文を利用することができます。

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

この行を追加すると、クエリーは次のようになります。

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

キーワード“ WHERE ”は、4D ODBC により自動的に追加されます。

OC Get clause in context 文を実行し、後でこの句を取り込むことができます。

コンテキストから結果を取得する

OC Add to context コマンドは、2 つのデータ格納場所である、データソースカラムと 4D ODBC データオブジェクト（変数、配列、フィールド等）とのバインドを設定します。

注：*OC Create context dialog コマンドを使うと、フィールドへのバインドを簡単に設定することができますが、コンテキストでは対応するファイルが 4D ODBC に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取得する最も簡単な方法であり、データを保存したり、ローカル上にストラクチャを作成する必要もありません。また、変数へのコンテキストは、単一のローを参照、編集するレイアウトを作成するには適しています。この方法では、メモリに常駐する変数を使用するため、ディスクアクセスの必要はありません。したがって、優れたパフォーマンスが得られます。

コンテキストで使用されるすべての 4D ODBC 変数は、あらかじめ宣言しておく必要があります。次のコンパイラ命令を使い、変数を定義します。

C_BOOLEAN、C_DATE、C_INTEGER、C_LONGINT、C_REAL、C_STRING、C_TEXT、C_TIME。

コンテキストにおいて変数へのバインドを使用している場合、サーバからは一度にひとつのローしか読み込めません。OC Load rows context は、コマンドの limit が 1 に設定されている場合のみ有効です。

▼ 次の例題は、テーブルのタイトルと一連の 4D ODBC 変数との間にコンテキストを定義しています。

```
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

コンテキストで使用される 4D ODBC の配列はすべて、あらかじめ宣言しておく必要があります。次の配列コマンドを使い、配列を定義します。

ARRAY BOOLEAN、ARRAY DATE、ARRAY INTEGER、ARRAY LONGINT、ARRAY REAL、ARRAY TEXT、ARRAY STRING。

▼ 次の例題は、テーブルのタイトルにコンテキストを定義し、4D ODBC の配列 arTitle にタイトルのカラムをロードします。

```
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をバインド
$res:=OC Load rows context(myContext)
  `ローを取り込む
```

フィールドへのコンテキスト

4D ODBC のフィールドを使用すると、データの一時的な保管場所となり、後でレポートやラベルを作成する際に利用できます。

コンテキストからフィールドへデータを取り込む時に、limit を 1 に設定して OC Load rows context コマンドを使用すると、プロシージャからレコードの作成や保存を行う必要が生じます。この処理を実行する際、必ず 4D ODBC のカレントレコードを認識しておいてください。

コンテキストからフィールドへデータを取り込む時に、limit を 1 より大きく設定して OC Load rows context コマンドを使用すると、4D ODBC によりすべてのレコードが自動的に作成、保存されます。

ひとつのコンテキスト内のバインド設定は、必ずひとつの 4D ODBC ファイルに対して行います。つまり、カラム Col1 を [File1]Field1 にバインドし、カラム Col2 を [File2]Field1 にバインドすることはできません。フィールドである 4D ODBC オブジェクトは、すべて同一ファイルのものでなくてはなりません。

▼ 次の例題は、テーブルのタイトルと 4D ODBC のファイル [titles] との間にコンテキストを作成します。

```
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)
```

第 5 章　設計時の選択

4D ODBC では、4th Dimension を ODBC データソースのフロントエンドとして使用できます。したがって、データソースが装備するデータの保管や操作に関する機能を利用しながらも、使いやすい独自のインタフェイスをユーザに提供することができます。4th Dimension により、あらゆるユーザインタフェイスを作成可能です。

この構成の下で 4D ODBC を使用する利点のひとつは、クライアント／サーバアーキテクチャをベースにしたシステム構築を行えるところです。4D ODBC を使うと、クライアントとなる 4th Dimension アプリケーションからデータソースに対し、実行するデータ操作を指示することができます。するとサーバ側ではこれらの処理が実行され、必要なデータが返されます。クライアント／サーバベースのアプリケーションでは、データは常にサーバ側に置かれます。

インプリメンテーションの選択

4th Dimension アプリケーションを作成する際に、効率的なクライアント／サーバアーキテクチャを利用したデータベースを設計したいと思われることでしょう。データベース設計における選択によっては、データ転送時の処理速度にかなりの影響を受ける場合があります。

次の節では、さまざまな 4D ODBC データベースプログラミング手法について説明します。

コンテキストの使用

コンテキストを使用する際には、4th Dimension データベースのフィールドや、変数、配列と、データソースのカラムとの関連性について記述します。たとえば、4th Dimension 変数である *vFirstName* と、データソースのテーブルのカラムである *FirstName* を関連付ける場合などです。一度コンテキストを作成すると、4D ODBC のコンテキストコマンドを使い、4th Dimension のフィールドや、変数、配列と関連付けたデータ操作を行うことができます。

コンテキストの目的は、データソースのデータベース上のデータをクライアントである 4D アプリケーションに取り込み、表示、あるいは修正し、それを再度データソースへ戻すというシステムを簡単に設定することです。コンテキストは、既存の 4th Dimension データベースをデータソースのフロントエンドとして運用するには最適な手法です。また、SQL の経験が少ない場合には、特に役立ちます。また、プロトタイプを作成したり、アプリケーション開発時間を短縮したい場合

には理想的な機能です。アプリケーションを開発し、SQL に関する理解が深まるにしたがい、ローレベルコマンドをデータベースで使用するようにしてもいいでしょう。ローレベルコマンドを使用すると、パフォーマンスが向上するだけでなく、ネィティブな API の機能をすべて活用できるようになります。

データベースでコンテキストを使用する場合、次の節にあるパフォーマンスに関する説明を参照してください。

4th Dimension フィールドを使用したバインド

通常、4D のフィールドとデータベースカラムの間にバインドを設定することはできる限り避けてください。4D Passport のフィールドにバインドを設定すると、データソースのデータベースのミラーとなる 4th Dimension データベースストラクチャを独自に管理する必要が生じます。

しかし、バインドの使用に当たり、必ずしもデータベースストラクチャの複製を作る必要はありません。変数や配列を利用してコンテキストを作成することも可能で、より効率良くクライアント / サーバアーキテクチャを作成することができます。

フィールドへのバインドが効果的なのは、バッチ処理の場合です。バッチ処理を行うにために、フィールドにバインドを設定してデータを取得し、それをローカル上に保存することができます。この後データソースへの接続を切り、クライアントマシン上で作業を行います。そして、再度データソースへ接続し、修正したデータを返します。

4th Dimension 変数、配列を使用したバインド

4th Dimension の変数や配列をバインドに使用する場合、ホストデータベースの複製となる 4th Dimension データベースを管理しなくてもよい方法があります。バインドは 4th Dimension のフィールドに依存していないので、4th Dimension データベースにテーブルを 1つ定義し、そのテーブルにフィールドを 1つ作成します。データの転送や処理を行うに際には、変数や配列を表示するレイアウトを作成します。これらの組合せや形式は自由です

▼ 次の例題は、変数を使用してコンテキストを定義する方法を示しています。はじめに各変数が宣言され、次にホストデータベースのカラムと関連付けられます。

```
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
```

ローレベルコマンドの使用

ローレベルコマンドを利用するデータベースを設計する場合、SQL 文を使って実行するデータ処理をデータソースに知らせます。SQL 文を作成することにより、あらゆるデータ操作を実行できるようになります。たとえば、サーバ上のローの選択、挿入、更新、削除を行えます。一定の条件に基づいてデータを選択することができ、しかもリレート先データを条件に含めることができます。

一般的に、コンテキストコマンドよりローレベルコマンドを使用する方が効率的です。コンテキストコマンドの場合、サーバ上でリクエストを実行する前に、4D ODBC 側でコマンドを SQL のクエリーに翻訳する必要があります。しかし、ローレベルコマンドの場合には、リクエストは既に SQL 言語で指定されています。4D ODBC ではデータソースの API を利用して、データソースが理解できる形式にローレベル文を翻訳します。

変数や配列を使ったコンテキストと同様に、ローレベルコマンドも 4th Dimension フィールドに依存していません。したがって、データベースストラクチャの管理は不要です。フィールドを１つだけ含むテーブルを１つ定義したアプリケーションを作成して、そこに複数のレイアウトを作り、自由な形式でデータを表示することもできます。

▼ 次の例題では、ローレベルコマンドを使って、ホストデータベース上にある一連のローを選択し、一行目のローにデータを取り込む方法を示します。このプロシージャでは、まず select 文を送信し、実行します。この後、1 カラムずつ最初のローにそのデータをロードします。

```
` レイアウト上でデータ表示に使用する変数の定義
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)

 ` 独自の SQL クエリー作成
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 アプリケーションの作成時、効率的なクライアント／サーバアーキテクチャを利用したデータベースを設計したいと思われることでしょう。データベース設計段階における選択によっては、データ転送時の処理速度にかなりの影響を受ける場合があります。

次の節では、4D ODBC データベースを構築する上でのさまざまな手法について説明します。

4D ODBC をフロントエンドとして使用する

4th Dimension をフロントエンドとして使用すると、ODBC データソースが唯一のデータレポジトリ（保管場所）となり、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を構築する目的にのみ使われます。このアーキテクチャにおいて、アプリケーションのデータは、ローカルの 4th Dimension 上に保存されません。この手法はパフォーマンス向上のために最適化され、データソース側の制約を受けるだけです。

4th Dimension のプロシージャ言語には、データの認証や操作を行う関数が豊富に用意されています。これが、フロントエンドアプリケーション構築のために 4D が最も適している理由です。さらに、4D Compiler や 4D Insider を利用することにより、非常に機能性豊かな開発環境が提供されます。

4D ODBC によるバッチ処理

バッチ処理は、集中的に ODBC データソースに保存されたデータを、定期的に 4th Dimension や 4D Server データベースエンジンへダウンロードするための手法です。このアーキテクチャは、データソースやネットワークに関するリソースの制約があり、効率的なリアルタイムアクセスが禁止されているような状況において使われます。また、この手法により、データを 4th Dimension へダウンロードした後で、4D 独自の機能を使用して、ビルトイン機能であるレポートや検索を行う際にこのデータを利用することができます。さらに、このアーキテクチャでは、モバイルコンピュータ上に 4D データソースを導入し、オフィス外でユーザがデータへアクセスできるような仕組みを提供できます。

4D ODBC を使用した分散処理

分散アーキテクチャを使い、ODBC データソースと 4th Dimension（または 4D Server）との間でデータを共有して、両方の利点を組み合わせることができます。これら 2 つの環境にデータストラクチャを分散することにより、複数のソースのデータを組み合わすことができます。適切に設計されていれば、このデータソースのブレンドはユーザから意識されることはまったくありません。

分散アーキテクチャの例として、大企業などで企業データを Oracle や Sybase といった大規模なデータソースに保管するシステムがあります。このデータは整合性が保たれ、企業全体からアクセス可能です。部門別ワークグループのデータなら、4D Server データベースに保存してもよいでしょう。4D Server と 4D ODBC のパワーを活用すれば、データを表示する側のデータソースに合わせたレイアウトを設計することができます。また、これら 2 システム間で “ 仮想リレート ” を開発することも可能です。

第 6 章　ログインコマンド

この章では、ODBC データソースへのログインやログアウトを行うコマンドや、接続時に高度な操作を実行するためのコマンドを説明します。

ログイン時、4D ODBC から接続 ID が返されます。この ID は、コマンドの対象となる接続を識別するため、ほとんどの 4D ODBC コマンドで使用されます。また、接続 ID はサーバーからのログアウトの際にも使用されます。

ログインコマンドとログアウトコマンド

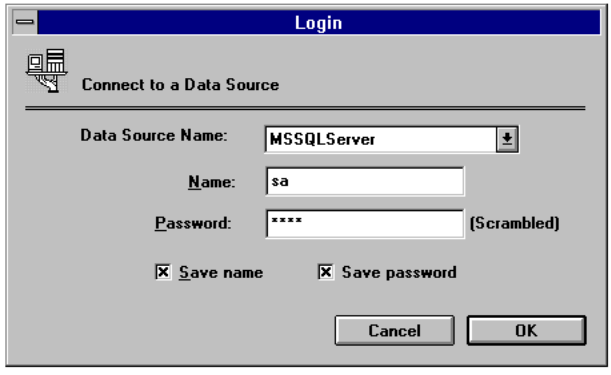
この節にあるコマンドを使い、ODBC サーバへの接続を制御することができます。ログインコマンドにより、次のような事柄を行えます。

- ダイアログボックスを使用したログイン (OC Login Dialog),
- ログイン引数を指定したログイン (OC Login),
- ログアウト (OC LOGOUT)

OC Login dialog

OC Login dialog → 倍長整数

OC Login dialog 関数は、「Connect to a Data Source」ダイアログボックスを表示します。ユーザはダイアログボックスを使用して、データソースを選択し、ログイン名やパスワードを指定します。



「OK」をクリックすると、OC Login dialog によりログインが行われ、接続 ID が返されます。「Cancel」をクリックすると、OC Login dialog からは 0 が返されます。エラーの場合には -1 が返されます。

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。ポップアップメニューには、特定のデータソースドライバのインストール時に、ユーザマシンにインストールされたデータソースが表示されます。ユーザが選択したサーバ名は 4D ODBC のリソースに保存されます。「Save Name」チェックボックスをチェックするとユーザ名が保存され、「Save Password」チェックボックスをチェックすると、パスワードが保存されます。

警告：*機密保護に重点をおく場合、パスワードは保存しないでください。パスワードを保存すると、誰でも 4D ODBC データベースにアクセスしたり、サーバへ接続できるようになります。また、パスワードは保存時に暗号化されません。*

注：*ダイアログボックス上で「OK」をクリックすると、システム変数 OK が 1 にセットされます。「キャンセル」をクリックすると、システム変数 OK は 0 にセットされます。*

参照：OC Login, OC LOGOUT

OC Login

OC Login ({*User***} {***; Password* **{;** *Data Source***}})** → 倍長整数

引数	タイプ	説明
User	文字列	ユーザ名
Password	文字列	パスワード
Data Source	文字列	データソース名

OC Login 関数は、ユーザが指定した引数を使用して、ODBC サーバへログインし、接続 ID（識別子）を返します。

User は使用するユーザ名です。*User* を指定しないと、4D ODBC では前に「Connect to a Data Source」ダイアログボックス上で指定したユーザ名が使用されます。このダイアログボックスで「Save Name」チェックボックスをチェックした場合にだけ、ユーザ名が認識されます。

Password は *User* のパスワードです。*Password* を指定しないと、4D ODBC では前に「Connect to a Data Source」ダイアログボックス上で指定したパスワードが使用されます。このダイアログボックスで「Save Password」チェックボックスがチェックされた場合にだけ、*Password* が使用されます。

Data Source には接続しようとするデータソースを指定します。使用するデータソースドライバのために「ODBC Setup」で定義した名前を指定してください。*Data Source* を指定しないと、4D ODBC では前に「Connect to a Data Source」ダイアログボックス上で指定したデータソース名が使用されます。

▼ たとえば、次の文はユーザ名 “ mary ”、パスワード “ toto ” を使い、Microsoft SQL Server へログインします。

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

▼ 次の文は、「Connect to an ODBC Server」ダイアログボックス上で以前指定された サーバ、ユーザ名、パスワードを使ってログインします。

Login_ID := ***OC Login***

OC Login の操作が正常に終了すると接続 ID が返され、エラーの場合には -1 が返されます。前の例題では、変数 Login_ID に接続 ID が保存されます。この変数は、文の実行やデータの取得を行う他の 4D ODBC コマンドで使用したり、サーバからログアウトするために、OC LOGOUT コマンドで使用します。

参照： OC Login dialog、OC LOGOUT

OC LOGOUT

OC LOGOUT (*Login_ID***)**

引数	タイプ	**説明**
Login_ID	倍長整数	接続 ID

OC LOGOUT コマンドは特定の接続を終了します。

Login_ID には有効な接続 ID を指定してください。これは OC Login や OC Login dialog 関数でログインした時に取得したものです。

このコマンドにより、接続中にオープンされたカーソルとコンテキストがクローズされ、アクティブではなくなります。また、*Login_ID* で指定した ID は無効になります。しかし、*Login_ID* は自動的にゼロにリセットされないため、*Login_ID* の値を利用して、接続がオープンであるかどうか判断することはできません。

ログアウト時、4D ODBC はカレントトランザクションを終了します。

4th Dimension を終了すると、4D ODBC は自動的にサーバからログアウトします。

参照： OC Login、OC Login dialog

ハイレベルコマンド

ハイレベルコマンドのなかには、接続時の操作を実行できるものがあります。これらのコマンドはローレベルの API コマンドと組み合わされ、1 つの簡単なコマンドを使って複雑な処理を行えます。これらのコマンドを使用し、次のような事柄を行えます。

- SQL 文の実行（OC Query exec SQL または OC Execute SQL）
- ODBC データソース上に 4th Dimension テーブルのコピー（クローン）を作成（OC Clone 4D table）
- 4th Dimension テーブルを作るため、ODBC データソースのテーブルのクローンを作成 （OC Clone ODBC table）

OC Query exec

OC Query exec(*Cursor_ID*; *SQLCommand*; *Limit* {*; Array 1*{;...;*Arrayr22*}})
→ 整数

引数	タイプ	説明
Connection_ID	倍長整数	ログイン ID またはカーソル ID)
SQLCommand	テキスト	実行する SQL コマンド
Limit	整数	返す最大行数
ArrayN	文字列	4D 変数名、配列名

OC Query exec 関数は、SQL クエリーを送信し、4th Dimension 配列にその結果を格納します。

Connection_ID には、有効なログイン ID やカーソル ID を指定してください。有効なログイン ID を使用すると、4D ODBC によりこの関数で使うカーソルが自動的にオープン／クローズされます。有効なカーソル ID を渡した場合、関数では指定したカーソルが使用されます。

SQLCommand には、実行しようとする SQL クエリーのテキストを指定します。SQL コマンドで有効な構文が使用されている限り、すべての SQL コマンドは受け入れられます。ODBC でサポートされる SQL に関する詳細は、使用している ODBC ドライバのドキュメントを参照してください。

Limit には 4th Dimension に返す結果の最大行数を指定します。*Limit* に -1 を指定すると、OC Query exec により、結果がすべてロードされます。-1 より大きな値を指定した場合、残りの行はロードされません。行の制限を設け、後で残りの行をロードするには、第 7 章の「ローレベルコマンド」で説明されているカーソルコマンドを使用してください。

ArrayN;...; Array22 には、クエリーの結果を受け取る 4th Dimension 配列の名前を指定します。*ArrayN* という名称の配列には、カラム *N* の結果が納められます（結果がある場合）。

OC Query exec はデータソースから取得したカラム数を返し、エラーの場合には -1 を返します。

OC Query exec では、格納先の配列のデータタイプに合わせて、結果が変換されます。引数 *ArrayN* の数は、返される結果のカラム数に合わせる必要はありません。余分な引数やカラムは無視されます。

OC Query exec は SQL クエリーを実行する一番の近道です。この関数は次のような処理に相当します。

1. **OC Create cursor でカーソルを作成**
2. **OC Set SQL in cursor で SQL クエリーを送信**
3. **OC Execute cursor でクエリーを実行**
4. **結果を取り込むため、OC BIND でバインドを設定**
5. **OC Load row で結果をロード**
6. **OC DROP CURSOR でカーソルをドロップ**

▼ 次のプロシージャは、ODBC データソースへログインし、テーブル DEPT に行を挿入した後、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
```

▼ 次のプロシージャは、テーブル EMP のデータを 4D 配列の *arempno*、*arename*、*arsal* にロードします。

```
$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 Execute SQL

OC Execute SQL(*Cursor_ID*; *SQLCommand*; *ArrayList; Limit*) → **整数**

引数	タイプ	説明
Connection_ID	倍長整数	ログイン ID またはカーソル ID)
SQLCommand	テキスト	実行する SQL コマンド
ArrayList	配列	4D の文字列配列名
Limit	整数	返す最大行数

OC Execute SQL 関数は、SQL クエリーを送信し、結果を 4th Dimension 配列に納めます。

OC Execute SQL 関数は、前に説明した OC Query exec 関数に似ています。この関数により、データソースへ SQL コマンドを送信、実行し、リクエストの結果を 4D 配列に格納することができます。OC Query exec 関数では、引数として結果の格納場所となる配列を渡す必要がありますが、OC Execute SQL 関数では結果を納める配列の名前を指定したテキスト配列、または文字列配列が必要です。したがって、この関数を使用することにより、22 という配列数の制限が無くなります。

Connection_ID には、有効なログイン ID やカーソル ID を必ず指定してください。有効なログイン ID を使用すると、4D ODBC によりこの関数で使用するカーソルが自動的にオープン／クローズされます。有効なカーソル ID を渡した場合、関数では指定したカーソルが使用されます。

SQLCommand には、実行しようとする SQL クエリーのテキストを指定します。SQL コマンドで有効な構文が使用されている限り、すべての SQL コマンドは受け入れられます。ODBC でサポートされる SQL に関する詳細は、使用している ODBC ドライバのドキュメントを参照してください。

Limit には 4th Dimension に返す結果の最大行数を指定します。*Limit* に -1 を指定すると、OC Execute SQL により、結果がすべてロードされます。-1 より大きな値を指定した場合、残りの行はロードされません。行の制限を設け、後で残りの行

をロードするには、第 7 章の「ローレベルコマンド」で説明したカーソルコマンドを使用してください。

ArrayList には、クエリーの結果を受け取る配列の名前を格納した 4th Dimension の文字列配列を指定します。*ArrayList{n}* という名称のオブジェクトには、カラム *N* の結果が納められます（結果がある場合）。

関数の実行中にエラーが発生すると、OC Execute SQL 関数から -1 が返されます。処理が正常に終了すると 0 以上の値が返され、結果が納められた配列の数を示します。

▼ 次の例題では、OC Execute SQL 関数を使用して“ Sales ”テーブルから 3 カラムを取り込み、その結果を 3 つの 4D 配列に納めます。格納場所となる配列の名称は、配列 arArrays 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;arArrays;-1)
```

OC Clone 4D table

OC Clone 4D table(*Login_ID* **{**; *4DTableNumber* **{**; *Options***}}) → 整数**

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
4DTableNumber	整数	4D のテーブル番号
Options	倍長整数	オプション

OC Clone 4D table は、データソース上に 4th Dimension テーブルと同じカラム定義を持つテーブルを作成します。

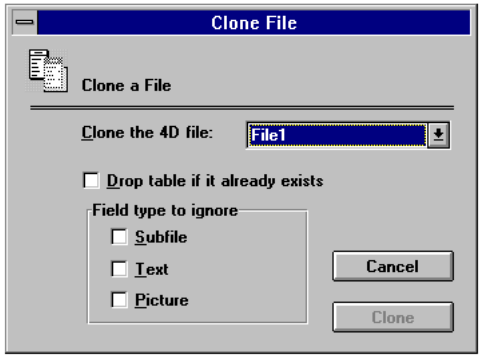
Connection_ID には、有効なログイン ID やカーソル ID を必ず指定してください。有効なログイン ID を使用すると、4D ODBC によりこの関数で使用するカーソルが自動的にオープン／クローズされます。有効なカーソル ID を渡した場合、関数では指定したカーソルが使用されます。

4DTableNumber には、4th Dimension のテーブル番号を指定します。つまり、複製しようとするストラクチャのテーブル番号です。

Options には、オプションを組合せて指定します。値を追加することにより、複数のオプションを指定できます。指定可能なオプションを次に示します。

Replace=1	OC Clone 4D table で新しくテーブルを作成する前に、DROP TABLE 文を生成します。
WithoutSubtable=2	OC Clone 4D table ではサブテーブルが無視されます。サブテーブルのクローン作成は現在サポートされていません。サブテーブルがあるテーブルには必ずこのオプションを指定してください。
WithoutText=4	OC Clone 4D table ではテキストタイプのフィールドが無視されます。
WithoutPicture=8	OC Clone 4D table ではピクチャタイプのフィールドが無視されます。

*4DTableNumber*を指定しないと、OC Clone 4D table により「Clone Table」ダイアログボックスが表示されます。



ポップアップメニューから、クローン対象となる 4th Dimension テーブルを選択できます。また、チェックボックスでは選択可能なオプションが示されます。

処理が正常に終了すると、OC Clone 4D table 関数から 1 が返され、エラーが発生すると -1 が返されます。また、ユーザが OK をクリックすると 0 が返さ、キャンセルをクリックすると、システム変数 OK に 0 がセットされます。

新しいテーブルには 4th Dimension のテーブル名が付けられ、カラムには 4th Dimension テーブルのフィールド名が付けられます。フィールド名にスペースが含まれている場合には、アンダースコアに置き換えられます。

タイプは次の規則に準じて変換されます。

4th Dimension	ODBC
文字 (*長さ*)	SQL_CHAR
テキスト	SQL_LONGVARCHAR
実数	SQL_DECIMAL
整数	SQL_INTEGER
倍長整数	SQL_BIGINT

4th Dimension	ODBC
日付	SQL_DATE
時間	SQL_TIME
ブール	SQL_BIT
ピクチャ	SQL_BINARY

データソースのカラムの作成方法は、お使いの ODBC ドライバのドキュメントを参照してください。

4th Dimension フィールドに「必須入力」属性が指定されている場合、対応するカラムは NOT NULL オプション付で作成されます。

インデックス付のフィールドの場合、各々に対するインデックスが作成されます。インデックス 名は、4th Dimension のテーブル名、“ $ ” 記号、フィールド名を組み合わせた名称の頭に接頭辞 “ I_ ” とテーブルのシーケンス番号が付けられます。たとえば、EMP テーブルのフィールド Sal のインデックスには、“ I_EMP$SAL ” という名前が付けられます。

4th Dimension フィールドに「重複不可」属性が指定されている場合、インデックスは UNIQUE オプション付で作成されます。

▼ 次のプロシージャは 4D データベースのテーブルを ODBC サーバにコピーします。サーバ上にテーブルが既に存在する場合は、自動的に置き換えられます。

```
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

OC Clone ODBC table

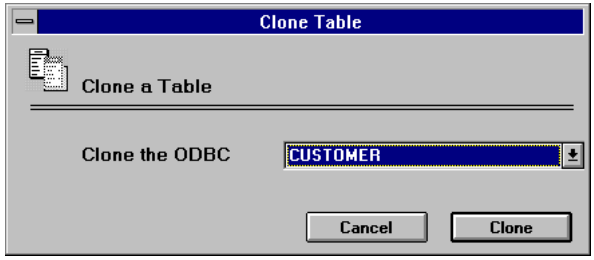
OC Clone ODBC table (*Table_ID*) → **整数**

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID

OC Clone ODBC table 関数は、ODBC テーブルに相当する 4th Dimension テーブルを作成し、フィールドを定義します。

Login_ID には、有効なログイン ID を必ず指定 してください。

OC Clone ODBC table を実行すると、「Clone Table」ダイアログボックスが表示されます。



処理が正常に終了すると、OC Clone ODBC table から 1 が返され、エラーが発生すると -1 が返されます。ユーザがキャンセルをクリックすると、システム変数 OK に 0 がセットされます。

新しいファイルには FileN という名前が付けられます。N はファイル番号です。フィールド名には ODBC データソースのカラム名が付けられます。

タイプは次の規則に準じて変換されます。

4th Dimension	ODBC
文字 (長さ)	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

データソースのカラムの作成方法は、使用している ODBC ドライバドキュメントを参照してください。

第 7 章　コンテキストコマンド

コンテキストは、テーブルのカラムと、4th Dimension のフィールド、変数、配列とのバインドで構成されます。コンテキストを定義すると、これを使って ODBC データを 4th Dimension にロードしたり、SQL 文を使わずに ODBC データソースの情報を更新することができます。

この章のコマンドを使い、次のような事柄を実行できます。

- コンテキストの作成 (OC Create context dialog、OC Create context)
- コンテキストのバインド定義 (OC ADD TO CONTEXT)
- コンテキストへの句の追加 (OC EDIT CLAUSES IN CONTEXT、OC SET CLAUSE IN CONTEXT)
- コンテキスト定義の保存、ロード (OC Save context picture、OC Save context file、OC Load context picture、OC Load context file)
- コンテキストのアクティブ化 (OC Activate context)
- ODBC のローを 4th Dimension へロード (OC Previous in context、OC Goto in context、OC Load rows context)
- データの更新、挿入、削除 (OC Update in context、OC Insert in context、OC Delete in context)
- コンテキスト情報の取得 (OC Get clause in context)
- コンテキストの非アクティブ化、またはクローズ (OC DEACTIVATE CONTEXT、OC DROP CONTEXT)

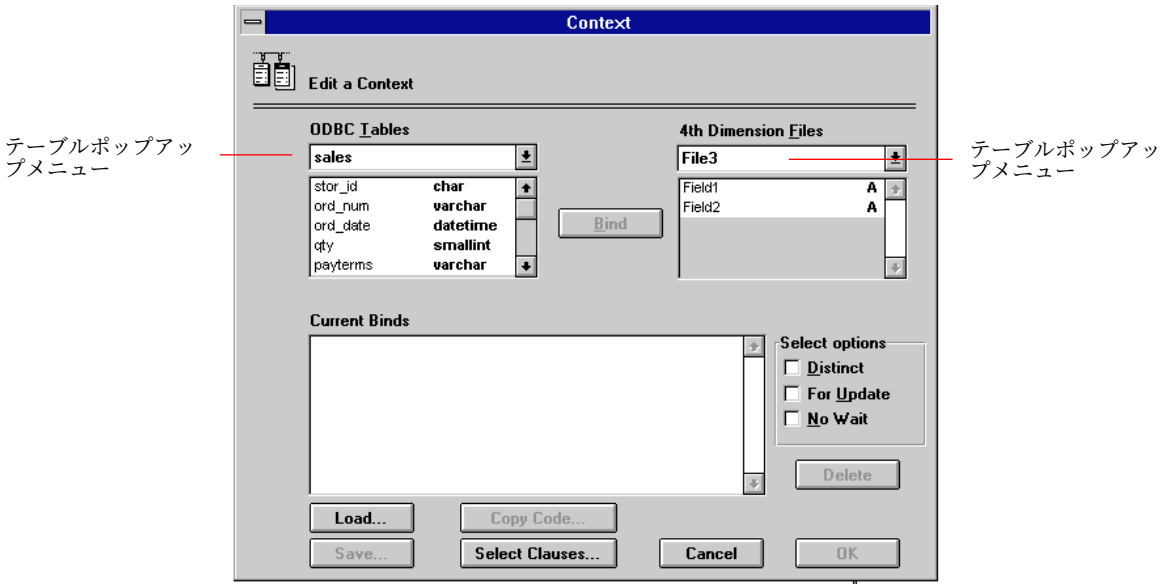
コンテキストは、ダイアログボックスを使用したり、プロシージャから一連の文を実行することにより作成することができます。それぞれの方法に関する詳細は、第 3 章「コンテキストの使用」を参照してください。

OC Create context dialog

OC Create context dialog (*Login_ID* {; *TableName* {; *4DTableNum*}}) → 倍長整数

引数	タイプ	説明
Login_ID	倍長整数	接続 ID
TableName	文字列	デフォルトの ODBC テーブル名
4DTableNum	整数	デフォルトの 4th Dimension テーブルのテーブル番号

関数は、「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を使い、4th Dimension のフィールドと ODBC データソーステーブルのカラムのダイナミックリンクを作成することができます。



「Edit a Context」ダイアログボックスを使ったコンテキスト定義については、第 3 章の「ダイアログボックスを使用したコンテキスト作成」を参照してください。

Login_ID には必ず、有効なログイン ID を指定します。ログイン ID により、テーブルポップアップメニューに表示するデータソースのテーブルリストが決定されます。このリストは、ログインユーザが今までアクセスしたテーブルを元に作成されます。

TableName には、デフォルトとして「ODBC Tables」ポップアップメニューに表示するデータソーステーブルを指定します。OC ADD TO CONTEXT コマンドでテーブル名を特に指定しない場合、ここで指定したテーブルがデフォルトとして使用されます。必ず *TableName* には、*user.table* または *table* フォームにあるテーブル名を指定してください。

4DTableNum には、デフォルトとして「4th Dimension Files」ポップアップメニューに表示する 4th Dimension テーブルのテーブル番号を指定します。

ダイアログボックスで「OK」をクリックすると、OC Create context dialog からコンテキスト IDが返されます。このコンテキスト IDは、OC Activate context でコンテキストをアクティブな状態にする時に使います。また、システム変数 OK は 1にセットされます。

ダイアログボックスで「Cancel」をクリックすると、OC Create context dialog から 0が返されます。また、システム変数 OK は 0 にセットされます。

エラーの場合、OC Create context dialog から -1が返されます。

▼ 次の文は、「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 Create context、OC ADD TO CONTEXT、OC Activate context

OC Create context

OC Create context ({*TableName*};{Dinstinct;ForUpdate;NoWait}) → 倍長整数

引数	タイプ	説明
TableName	文字列	デフォルトの ODBC テーブル名
Distinct	ブール	Distinct 句を使用
ForUpdate	ブール	For Update 句を使用
NoWait	ブール	No Wait 句を使用

OC Create context 関数は、コンテキストを作成することを示し、作成するコンテキストの IDを返します。

この関数を実行しても、4th Dimension のフィールドや変数と ODBC のカラムは関連付けられません。各種バインドを設定するには、OC ADD TO CONTEXT をコールします。

TableName には、デフォルトテーブルを指定します。ここで指定したテーブルが OC ADD TO CONTEXT コマンドでデフォルトとして使用されます。*TableName* には必ず、*user.table* または *table* フォームにあるテーブル名を指定してください。

Distinct オプションを指定すると、SQL 文からはユニークなローだけが返されます。

ForUpdate オプションを指定すると、ローのロックが初期化されます（使用するデータソースでロック機能がサポートされている場合）。

NoWait オプションを指定すると、他のユーザによりローがロックされており、リードオンリーモードになっている場合でも、そのローへアクセスすることができます。デフォルトでは、ロードする前にそのローが開放されるまで待機します。

エラーが発生した場合、OC Create context から -1 が返され、それ以外の場合は、0 以上の有効なコンテキスト番号が返されます。

OC Create context と OC ADD TO CONTEXT コマンドでコンテキストを作成した後は、必ず OC Activate context 関数をコールしてコンテキストをアクティブな状態にしてください。

▼ 次の文は “ SCOTT.EMP ” コンテキストを開始します。

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

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

OC ADD TO CONTEXT

OC ADD TO CONTEXT (*Context_ID*; *ColName*; *4DObject*})

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
ColName	文字列	ODBC カラムまたは式
4DObject	文字列	4th Dimension 変数、配列、フィールド名

OC ADD TO CONTEXT コマンドは、4th Dimension のオブジェクトを ODBC のカラムや式にバインドします。このバインド設定は、OC Create context 関数で開始されたコンテキスト定義に追加されます。

Context_ID には必ず、あらかじめ作成したアクティブな状態ではないコンテキスト ID を指定します。

ColName には、ODBC のカラム名（“ *table.column* ” または “ *table* ” という形式で）または SQL の式（例 :“ sal*1.15 ”）を指定します。選択しようとするカラムに対し、デフォルトテーブル名を宣言していない場合、“ *table.column* ” という形式を使い、カラムが存在するテーブルを必ず特定してください。デフォルトテーブル名は、OC Create context dialog、または OC Create context により指定することができます。

4DObject には、4th Dimension のフィールド、変数、配列をポインタで指定します。

▼ 次のプロシージャはコンテキスト定義を開始した後、コンテキストにバインドを設定します。

```
Context_ID = OC Create context( "EMP")` コンテキスト定義の開始
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 関数を使い、コンテキストをアクティブな状態にすると、次の SELECT 文が ODBC サーバへ送られます。

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

参照： OC Create context、OC Activate context

OC EDIT CLAUSES IN CONTEXT

OC EDIT CLAUSES IN CONTEXT (*Login_ID*; *Context_ID*)

引数	タイプ	説明
Login_ID	倍長整数	接続 ID
Context_ID	倍長整数	コンテキスト ID

OC EDIT CLAUSES IN CONTEXT コマンドにより、「Edit a SQL Clause」ダイアログボックスが表示され、特定のコンテキストの句を編集することができます。

Login_ID には必ず、アクセス可能なテーブルとカラム一覧をロードする際に使用した有効な接続 ID を指定します。

Context_ID には必ず、あらかじめ作成した、アクティブな状態ではないコンテキスト ID を 指定します。

このダイアログボックスを使用して、次の句を修正できます。

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

ユーザがダイアログボックスで「OK」をクリックすると、システム変数 OK は 1 にセットされます。ダイアログボックスで「Cancel」をクリックすると、システム変数 OK は 0 にセットされます。

▼ 次のプロシージャでは、コンテキストを作成した後、ユーザがコンテキストに句を指定できるようにダイアログボックスを表示します。

```
Context_ID = OC Create a context("EMP")
  `コンテキスト作成
OC ADD TO CONTEXT(Context_ID;"empno";»[Employees]No)
OC ADD TO CONTEXT(Context_ID;"ename";»[Employees]Name)
  `バインド定義
OC EDIT CLAUSES IN CONTEXT(Login_ID;Context_ID)
  `独自のクエリを作成
```

参照： OC SET CLAUSE IN CONTEXT、OC Get clause in context

OC SET CLAUSE IN CONTEXT

OC SET CLAUSE IN CONTEXT (*Context_ID*; *ClauseNumber*; *Clause*)

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
ClauseNumber	整数	セットする句の番号
Clause	テキスト	句のテキスト

OC SET CLAUSE IN CONTEXT コマンドは、OC EDIT CLAUSES IN CONTEXT コマンドでグラフィックを使用して行なわれる作業を、プロシージャから行います。

ユーザに句を作成させたくない場合、OC SET CLAUSE IN CONTEXT コマンドを使用してください。句に関して、ユーザにある程度の制御を許したいが、句を入力するインタフェースは別途作成したい場合、引数 *Clause* に句を格納した変数を指定することができます。

OC SET CLAUSE IN CONTEXT コマンドは、コンテキストに句を定義します。*Context_ID* には必ず、あらかじめ作成した、アクティブな状態ではないコンテキスト ID を使用します。

ClauseNumber には、定義しようとする句を表わす番号を指定します。

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

Clause には、関連するキーワード（例：" WHERE " や他のキーワード）を使わずに、句のテキストを指定します。

▼ 次の文は、WHERE 句を作成します。この句により、部門番号が 30 である従業員を検索することができます。

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

参照： OC EDIT CLAUSES IN CONTEXT、OC Get clause in context

OC Get clause in context

OC Get clause in context (*Context_ID*; *ClauseNumber*) → **テキスト**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
ClauseNumber	整数	取得する句の番号

OC Get clause in context 関数は、コンテキストの句のテキストを取得します。

Context_ID には必ず、あらかじめ作成した、アクティブな、またはアクティブな状態ではないコンテキスト ID を指定します。

ClauseNumber には、句を表わす次の番号を指定します。

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

エラーの場合、OC Get clause in context から空の文字列が返されます。

▼ 次のプロシージャでは、「Edit a Context」ダイアログボックスを表示して、ユーザに独自のコンテキストを定義させます。このダイアログボックスを使い、ユーザは WHERE 句などの句を追加します。その後、プロシージャで WHERE 句を取得して、その WHERE 句に制限を付加し、ユーザが社長に関する情報を取り出せないようにします。

```
Context_ID = OC Create context dialog(Login_ID)` コンテキスト作成
If(Context_ID>0)                          ` ユーザがキャンセルをクリックしなかった場合
  clause:=OC Get clause in context(Context_ID;2) `WHERE 句を取得
  If (clause#"")`WHERE 句が存在した場合
     clause:="("+clause+") AND "`AND を付加
  End if`                                 ` 組合せの準備
  clause:=clause+"EMP.job <> 'President'"` 条件を追加
  OC SET CLAUSE IN CONTEXT(Context_ID;2;clause) `WHERE 句の定義 ...
End if
```

参照： OC EDIT CLAUSES IN CONTEXT、OC SET CLAUSE IN CONTEXT

OC Save context picture

OC Save context picture (*Context_ID*) → ピクチャ

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID

The OC Save context picture 関数は、ピクチャタイプの変数形式でコンテキスト定義を返します。この変数を取り込んだ後、ピクチャフィールドに保存することができます。

Context_ID には必ず、あらかじめ作成したアクティブな、またはアクティブな状態ではないコンテキスト ID を指定します。

参照： OC Load context picture

OC Load context picture

OC Load context picture (*ContextPicture*) → 倍長整数

引数	タイプ	説明
ContextPicture	ピクチャ	コンテキスト定義を含むピクチャ変数

OC Load context picture 関数は、ピクチャフィールドや変数からコンテキスト定義をロードします。また、この関数からは新しく作成したコンテキストの ID が返されます。このコンテキストは、OC Save context picture 関数であらかじめ作成しておく必要があります。

エラーの場合、OC Load context picture から -1 が返されます。

OC Activate context をコールすると、コンテキストをアクティブにすることができます。

参照： OC Save context picture、OC Create context dialog、OC Create context

OC SAVE CONTEXT FILE

OC SAVE CONTEXT FILE (*Context_ID; FileName*)

引数	タイプ	説明
Context_ID	倍長整数	保存するコンテキスト ID
FileName	テキスト	コンテキスト定義を受け取るファイル名

OC SAVE CONTEXT FILE コマンドは、コンテキスト定義をディスク上のファイルに保存します。

Context_ID には必ず、あらかじめ作成したアクティブな、またはアクティブな状態ではないコンテキスト ID を指定します。

FileName には、作成する文書ファイルの名前を指定します。*FileName* に空の文字列を指定すると、「Save File」ダイアログボックスが表示されます。また、*FileName* に指定したパス名が不完全であった場合は、OC SAVE CONTEXT FILE によりデータベースストラクチャファイルと同じフォルダにファイルが保存されます。

同じ名前の文書ファイルが既に存在する場合には、上書きされます。

参照： OC Load context file

OC Load context file

OC Load context file (*FileName*) → 倍長整数

引数	タイプ	説明
FileName	テキスト	コンテキスト定義を含むファイル名

OC Load context file 関数は、コンテキスト定義が納められた文書ファイルを開きます。これは、前に OC SAVE CONTEXT FILE コマンドでディスクに保存した文書ファイルです。

OC Load context file 関数は、新しく開かれたコンテキストの ID を返します。

FileName に空の文字列を指定すると、「Open File」ダイアログボックスが表示されます。また、*FileName* に指定したパス名が不完全であった場合には、OC Load context file により、データベースストラクチャファイルと同じフォルダが検索されます。

エラーの場合、OC Load context picture から -1 が返され、「Open File」ダイアログボックスで「Cancel」をクリックした場合には、0 が返されます。

OC Activate context をコールすると、コンテキストをアクティブにすることができます。

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

OC Activate context

OC Activate context (*Login_ID*; *Context_ID*) → 整数

引数	タイプ	説明
Login_ID	倍長整数	接続 ID
Context_ID	倍長整数	コンテキスト ID

OC Activate context 関数は、コンテキスト定義が格納されたピクチャやファイルをロードして作成したコンテキストや、OC Create context や OC Create context dialog を使って作成したコンテキストをアクティブな状態にします。

Login_ID には必ず、有効な接続 ID を指定します。*Context_ID* には必ず、あらかじめ作成した、アクティブな状態ではないコンテキスト ID を使用してください。

エラーが発生した場合、OC Activate context から -1 が返され、それ以外の場合は 1 が返されます。

OC Activate context によりカーソルが作成され、クエリー結果に応じて一連のローが定義されます。しかし、結果ローは取得しません。結果ローを表示するには、OC Load rows context と OC Previous in context を使用します。OC Load rows context を使い、結果ローの最初の行を取り込むことができます。

コンテキストがアクティブな状態である間は、OC ADD TO CONTEXT や OC EDIT CLAUSES IN CONTEXT、OC SET CLAUSE IN CONTEXT で定義を修正することはできません。

OC DEACTIVATE CONTEXT をコールして、非アクティブにするまで、コンテキストはアクティブな状態のままです。

参照： OC DEACTIVATE CONTEXT、OC Create context、OC Load rows context、OC Previous in context

OC Previous in context

OC Previous in context (*Context_ID* **{;** *Index***}) → 整数**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID

OC Previous in context 関数は、カレントローのひとつ前の結果ローをロードします。OC Activate context の後のように、カレントローが定義されていない場合、OC Previous in context では何も実行されません。

Context_ID には必ず、あらかじめ作成したアクティブなコンテキスト ID を指定してください。

OC Previous in context 関数により、カレントローに関連する 4th Dimension オブジェクトが更新されます。フィールドへのバインドについては、カレントレコードは更新されますが、保存されません。カレントレコードが存在しない場合、OC Previous in context 関数では対応するバインドが無視されます。

OC Previous in context が正常に終了すると 1 が返され、前のローが存在しない場合には 0 が、エラーの場合には -1 が返されます。

参照： OC Activate context、OC Load rows context

OC Goto in context

OC Goto in context (*Context_ID***;** *RowNumber* **{;** *Index***}) → 整数**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
RowNumber	倍長整数	コンテキストのロー番号

OC Goto in context 関数は、*RowNumber* に指定したローをロードします。*RowNumber* のローがカレントローとなります。

Context_ID には必ず、あらかじめ作成したアクティブなコンテキスト ID を指定してください。

RowNumber に、コンテキスト内の行数を超える数や、1 より小さい数を指定した場合、OC Goto in context 関数からは 0 が返され、何も実行されません。

OC Goto in Context では、新しいカレントローを使って 4th Dimension オブジェクトが更新されます。フィールドへのバインドについては、カレントレコードは更新されますが、保存されません。カレントレコードが存在しない場合、OC Goto in context 関数ではバインドが無視されます。

OC Goto in context が正常に終了すると 1 が返され、*RowNumber* のローが存在しない場合には 0 が、エラーの場合には -1 が返されます。

OC Load rows context

OC Load rows context (*Context_ID* {*; Limit*}) → 倍長整数

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
Limit	倍長整数	ロードする最大レコード数

OC Load rows context 関数は、コンテキストによって定義されたすべてのローを、コンテキストでバインドされた 4th Dimension オブジェクトへロードします。

Context_ID には必ず、あらかじめ作成したアクティブなコンテキスト ID を指定してください。

Limit により、ロードされる最大行数を指定することができます。

OC Load rows context では、必要に応じて 4th Dimension のレコードや配列要素が作成されます。Limit に 1 を指定した場合や、コンテキストにフィールドへのバインドが定義されている場合、OC Load rows context では結果の行をロードするために、メモリ上にレコードが作成されますが、保存されません（SAVE RECORD コマンドがコールされない）。

結果が 4th Dimension レコードにロードされると、4D ODBC で作成されたレコードでカレントセレクションが作られます。

OC Load rows context からはロードされた行数が返され、エラーの場合には -1 が返されます。

参照： OC Activate context

OC Update in context

OC Update in context (*Context_ID* {*; Index*}) → 整数

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
Index	整数	配列のインデックス

OC Update in context 関数は、コンテキストのバインドで指定された 4th Dimension オブジェクトをもとにして、ODBC サーバ上のカレントローを更新します。

Context_ID には必ず、あらかじめ作成した、アクティブなコンテキスト ID を指定してください。

Index は、配列へのバインドに対してのみ適用されます。*Index* を指定しないと、OC Update in context 関数により、カレントロー番号に対応する配列の行でカレントローが更新されます。カレントロー番号が配列サイズよりも大きい場合には、OC Update in context 関数から -1 が返されます。

Index を指定すると、*Index* で特定されたインデックスの位置にある配列の行を使い、カレントローが更新されます。*Index* が配列サイズよりも大きい場合には、OC Update in context 関数から -1 が返されます。

OC Update in context により“ UPDATE...WHERE ”タイプの SQL クエリーが送信され、コンテキストの一部であり、かつ“ ForUpdate ” オプション（OC ADD TO CONTEXT 参照）が設定されているカラムだけが特定されます。

OC Update in context が正常に終了すると 1 が返され、エラーの場合には -1 が返されます。

参照： OC Insert in context、OC Delete in context

OC Insert in context

OC Insert in context (*Context_ID* **{;** *Index***}) → 整数**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
Index	整数	テーブルのインデックス

OC Insert in context 関数は、コンテキストのバインドで指定された 4th Dimension オブジェクトを使用して、ODBC サーバ上に新しいローを作成します。

Context_ID には、あらかじめ作成したアクティブなコンテキスト ID を必ず指定してください。

Index は、配列へのバインドに対してのみ適用されます。*Index* を指定しないと、OC Insert in context 関数により、カレントロー番号に対応する配列要素を使って新しいローが作成されます。ロー番号が配列サイズよりも大きい場合には、OC Insert in context 関数から -1 が返されます。

Index を指定すると、*Index* で特定したインデックスの位置にある配列行を使い、新しいローが作成されます。*Index* が配列サイズよりも大きい場合には、*OC Insert in context* 関数から -1 が返されます。

OC Insert in context により、" INSERT... " タイプの SQL クエリーが送信されます。

OC Insert in context が正常に終了すると 1 が返され、エラーの場合には -1 が返されます。

OC Insert in context を実行してもカレントローは修正されず、新しいローはコンテキストに配置されません

参照： OC Update in context、OC Delete in context

OC Delete in context

OC Delete in context (*Context_ID***) → 整数**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID
Index	整数	テーブルのインデックス

OC Delete in context 関数は、ODBC サーバ上にあるコンテキストのカレントローを削除します。

Context_ID には必ず、あらかじめ作成したアクティブなコンテキスト ID を指定してください。

Index を指定しないと、OC Delete in context 関数により、カレントロー番号に対応する配列要素を使って、新しいローが削除されます。ロー番号が配列サイズよりも大きい場合には、OC Delete in context 関数から -1 が返されます。

カレントローが削除されると、次のローが存在する場合にはそれがカレントローになります。次のローが存在しなければ、前のローがカレントローになります。削除した後にローが無くなった場合、カレントローは定義されません。

OC Delete in context 関数をコールした後、ロー番号やカレントロー番号は変更されませんが、ODBC サーバ上のローは物理的に削除されます。OC Load rows context や OC Previous in context、または OC Goto in context を使い、前に削除されたローをロードしようとすると、4D ODBC からはエラーが返されます。

OC Delete in context により“ DELETE...WHERE ”タイプのSQL クエリーが送信されます。

OC Delete in context が正常に終了すると 1 が返され、*RowNumber* で指定したローが存在しない場合には 0 が、エラーの場合には -1 が返されます。

参照： OC Update in context、OC Insert in context

OC DEACTIVATE CONTEXT

OC DEACTIVATE CONTEXT (*Context_ID***)**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID

OC DEACTIVATE CONTEXT コマンドは、OC Activate context を使って前にアクティブな状態にしたコンテキストをアクティブではない状態にします。

Context_ID には、あらかじめ作成したアクティブなコンテキスト ID を必ず指定してください。

コンテキストは、OC SET CLAUSE IN CONTEXT コマンドを使って修正することができます。

参照： OC Activate context

OC DROP CONTEXT

OC DROP CONTEXT (*Context_ID***)**

引数	タイプ	説明
Context_ID	倍長整数	コンテキスト ID

OC DROP CONTEXT コマンドは、*Context_ID* で指定したコンテキスト定義により使用されているメモリ領域を開放します。

OC DROP CONTEXT コマンドでコンテキストをクローズした後、コンテキストは使用不可となり、コンテキスト ID は無効になります。OC DROP CONTEXT コマンドのコール時に、コンテキストがアクティブな状態であれば、まずアクティブではない状態に設定されます。

▼ 次のプロシージャでは、ダイアログボックスを表示して、ユーザにコンテキストを作成させた後、そのコンテキストをクローズします。

Context_ID = ***OC Create graphical context***(Login_ID)
OC DROP CONTEXT(Context_ID)

参照：OC Create context、OC Create context dialog、
OC DEACTIVATE CONTEXT

第 8 章　ローレベルコマンド

この章では、4th Dimension と ODBC データソース間で通信を管理するためのコマンドを説明します。この通信の管理にローレベルコマンドを使用する場合は、SQL 文を必ず指定し、ODBC データソースのデータの選択、追加、更新、削除を行います。

この節にあるコマンドを使い、次のような事柄を行えます。

- カーソルの作成 (OC Create cursor)
- カーソルへの SQL 文のセット (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間のバインド定義 (OC Bind)
- カーソルの実行 (OC Execute cursor, OC Execute direct cursor)
- 結果ローのロード (OC Load row, OC More results、OC CANCEL LOADING)
- カーソル情報の取得 (OC Number rows processed、OC Number of columns, OC Describe column, OC Column attributes)
- 引数マーカーの使用 (OC Bind parameter、OC Describe Parameter、OC Number of parameters)
- カーソル名の設定と取得 (OC SET CURSOR NAME、OC Get cursor name)
- カーソルのクローズ (OC DROP CURSOR)

OC Create cursor

OC Create cursor (*Login_ID*) → 倍長整数

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID

OC Create cursor 関数は、特定の接続のためにカーソルを作成してオープンします。

カーソルとは、クエリーの結果として取得したローが、一度に一つずつ処理されるようにする仕組のことです。ODBC データソースのプロセスとしてカーソルを捉えることもできます。作成したカーソルは、SQL 文と組み合わされ、ODBC データソースのデータの検索、追加、更新、削除を行います。複数のカーソルを同時に作成できますが、カーソルを多く作成すると、パフォーマンスに影響を与えますので注意してください。

Login_ID には、必ず有効なログイン ID を使用します。

処理が正常に終了すると、OC Create cursor からカーソル ID が返され、エラーの場合には -1 が返されます。

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

OC Set SQL in cursor

OC Set SQL in cursor (*Cursor_ID; SQLCommand*) → 整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
SQLCommand	テキスト	実行する SQL コマンド

OC Set SQL in cursor 関数は、SQL 文とカーソルとを関連付けます。この関数により、ODBC データソース側に SQL 文が送られます。OD Execute cursor をコールすると、SQL 文が実行されます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を指定します。

SQLCommand には、実行する SQL クエリーのテキストを指定します。SQL コマンドで有効な構文が使用されている限り、SQL コマンドはすべて受け付けれられます。ODBC でサポートされる SQL に関する詳細は、使用している ODBC ドライバのドキュメントを参照してください。

処理が正常に終了すると、OC Set SQL in cursor から 1 が返され、エラーの場合には -1 が返されます。

SQL 文で引数を使用することにより、SQL 文をダイナミックに作成することができます。SQL 文での引数の使用については、OC Bind parameter の項を参照してください。

▼ 次のプロシージャでは、フィールド [Employees]Name と変数 vFirstName の値を使用して、ODBC データソースにローを作成します。

```
$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

OC Bind

OC Bind(*Cursor_ID*; *ReferenceNum*; *4DObject*) → 整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
ReferenceNum	整数	参照番号
4DObject	文字列	4D 変数またはフィールドの名前

OC Bind コマンドを使用し、SQL の select 文で得た結果のカラムと 4th Dimension オブジェクト（フィールド、変数、配列）とを関連づけることができます。OC SET SQL in cursor または OC Execute direct cursor 関数により、関連づけられた SQL 文が送られます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

ReferenceNum には、SQL 文で参照した SQL カラムの順序番号を指定します。

4DObject には、*ReferenceNum* で参照したカラムの値を受け取るフィールドや変数、配列の名前を指定します。

注：*4DObject という名前は、引用符で囲まれ、文字列として渡されます（例："arArray1"）。さらに、変数や配列を使う場合は、適切なコンパイル命令を使用し、あらかじめ定義しておく必要があります（例：Array String(30;arArray1;0)）。*

処理が正常に終了すると、OC Bind から 1 が返され、エラーの場合には -1 が返されます。

▼ 次のプロシージャでは、テーブル “ authors ” のカラムを 4th Dimension の変数にバインドし、サーバよりローを 1 行ロードします。

```
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)
```

▼ 次のプロシージャでは、テーブル " authors " のカラムを 4th Dimension の配列にバインドし、サーバより複数カラムをロードします。

```
ARRAY STRING(30;ar1;0)
ARRAY STRING(30;ar2;0)
ARRAY INTEGER(ar3;0)
q:="select * from authors where city = 'Oakland'"
$res:=OC Set SQL in Cursor (cursor_id;q)
$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 SET SQL in cursor, OC Execute cursor, OC Execute direct cursor

OC Execute cursor

OC Execute cursor (*Cursor_ID*) → 整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

The OC Execute cursor コマンドは、ODBC データソースのカーソルに関連付けられている SQL 文を実行します。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

実行しようとする SQL 文が " SELECT... " タイプのクエリーであれば、OC Load rows cursor を使って結果行をすべてロードしていない場合や、OC CANCEL LOADING コマンドをコールしていない場合には、OC Execute cursor によりカーソルがアクティブな状態になります。

処理が正常に終了すると、OC Execute cursor から 1 が返され、エラーの場合には -1 が返されます。

参照： OC Set SQL in cursor, OC Execute direct cursor, OC CANCEL LOADING

OC Execute direct cursor

OC Execute direct cursor (*Cursor_ID*) → 整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
SQLCommand	テキスト	実行する SQL コマンド

OC Execute direct cursor コマンドは、SQL 文を ODBC データソースへ送信し、それを実行します。OC Execute direct cursor コマンドは、OC Set SQL in cursor と OC Execute cursor をコールした結果に相当します。

OC Execute direct cursor は、特定の SQL 文を作成し、それを 1 度だけ実行したい場合に便利なコマンドです。処理終了後、その SQL 文の実行時に使われた最適化情報は、すべて廃棄されます。SQL 文を繰り返し実行する必要がある場合、OC Set SQL in cursor と OC Execute cursor を組み合わせて使用してください。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

SQLCommand には、実行する SQL クエリーのテキストを指定します。SQL コマンドで有効な構文が使用されている限り、SQL コマンドはすべて受け付けれられます。ODBC でサポートされる SQL に関する詳細は、使用している 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 処理が正常に終了すると、OC Execute direct cursor から 1 が返され、エラーの場合には -1 が返されます。

参照： OC Set SQL in cursor, OC Execute cursor, OC CANCEL LOADING

OC Load row

OC Load row (*Cursor_ID*) → 倍長整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC Load row 関数は、OC Execute cursor や OC Execute direct cursor を使用した SQL クエリーの実行結果であるローをロードします。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態のカーソル ID を使用します。

OC Load row により、ODBC データソースの次のロー（カーソルのある場所を基準とする）が取り込まれ、その値は 4th Dimension オブジェクト（OC Bind 関数で指定）に割り当てられます。

4th Dimension フィールドへのバインドの場合、OC Load row によってメモリ上のレコードに値が割り当てられますが、保存されません。つまり、SAVE RECORD コマンドのコールは行われません。随時、レコードの作成と保存を必ず行ってください。

配列へのバインドの場合、OC Load row により配列に要素が追加され、その要素に値が割り当てられます。

取り込むべきローがまだある場合、OC Load row からは 1 が返され、ローがすべて取り込まれた場合には 0 が、エラーの場合には -1 が返されます。

変数や配列のロードに関する例題は、OC Bind の項を参照してください。

参照： OC Set SQL in cursor, OC Execute cursor, OC Execute direct cursor, OC Number rows processed

OC Number rows processed

OC Number rows processed (*Cursor_ID*) → 倍長整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC Number rows 関数は、INSERT、UPDATE、DELETE コマンドを使う SQL 文を実行したカーソルに対して適用されると、追加、更新、削除されたローの数を返します。クエリーが正常終了した後に、この関数を使用してください。

Cursor_ID には必ず、あらかじめ作成してあるカーソル ID を指定してください。

エラーの場合には、OC Execute direct cursor から -1 が返されます。

▼ 次のプロシージャは、テーブル " sales " から店 ID が " 6380 " のカラムをすべて削除し、処理されたローの数をアラートとして表示します。

```
$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 Set SQL in cursor, OC EXECUTE CURSOR

OC Number of columns

OC Number of columns (*Cursor_ID*) → 倍長整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC Number of columns 関数は、クエリーの結果に取得したカラム数を返します。OC Set SQL in cursor コマンドが正常終了した後であれば、この関数はいつでも使用できます。

Cursor_ID には必ず、あらかじめ作成してあるカーソル ID を指定してください。

エラーの場合、OC Number of columns から -1 が返されます。

▼ 次のプロシージャは、サーバへ SQL 文を送り、クエリーの結果として得られたカラム数をアラート表示します。

```
$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.")
```

OC Describe column

OC Describe column(*Cursor_ID*; *ColumnNum; ColumnName*; *ColumnType*) → 整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
ColumnNum	整数	カラム番号
ColumnName	変数	4D の文字列タイプ変数
ColumnType	変数	4D の整数タイプ変数

OC Describe column コマンドを使用すると、前にカーソルと関連付けた SQL クエリーで、結果として返されたカラムの*名前*と*タイプ*を取得できます。OC Set SQL in cursor を使い、カーソルに SQL 文をセットした後で、このコマンドを実行します。

Cursor_ID には必ず、あらかじめ作成してあるカーソル ID を指定してください。

ColumnNum には、各特性を取得しようとするカラムに関する SQL 文の順序番号を指定します。

ColumnName には 4D 変数を指定し、この関数がコールされた後、カラム名を格納します。

ColumnType is には、4D 変数（整数タイプ）を指定し、この関数がコールされた後、カラムタイプを示す値を格納します。各データタイプを表わす値は次の通りです。

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

参照： OC Set SQL in cursor, OC EXECUTE CURSOR

OC Column attributes

OC Column attributes (*Cursor_ID; ColumnNum; Attribute***) → 文字列**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
ColumnNum	整数	カラム番号
Attribute	整数	属性番号

OC Column attributes 関数を使用すると、OD Set SQL in cursor コマンドを実行した後で、カラムの属性情報を取得することができます。

Cursor_ID には必ず、あらかじめ作成してあるカーソル ID を指定してください。

ColumnNum には、各特性を取得しようとするカラムに関する SQL 文の順序番号を指定します。

Attribute には、求める属性の ID 番号を指定します。

属性として指定できる値は次の通りです。

属性	ID
SQL_COLUMN_COUNT	0
SQL_COLUMN_NAME	1
SQL_COLUMN_TYPE	2

属性	ID
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 attributes からは求める情報が返されます。エラーの場合には空の文字列が返されます。

▼ 次のプロシージャは、クエリーの最初のカラムサイズをアラートとして表示します。

```
$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 Set SQL in cursor, OC EXECUTE CURSOR

OC More results

OC More results(*Cursor_ID*) → 倍長整数

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC More results 関数により、クエリーにまだ取り込むべき結果があるかどうかを判断できます。まだ取得する結果がある場合、それらの結果が処理されます。この関数は、バッチ型 SQL 文をサポートするために使います。たとえば、複数の結果セットが返されるストアドプロシージャや、複数の結果セットが返されるその他の DBMS コマンドなどで使用します。

アプリケーションで、あるセットから別のセットへと移動する前に、結果をすべて取得する必要はありません。OC More results がコールされると、前の結果セットは破棄されます。したがって、カレント結果セットから必要なローをすべてロードした後にこの関数を使用してください。

Cursor_ID には必ず、あらかじめ作成してあるカーソル ID を指定してください。

処理が正常に終了すると、OC More results から 1 が返され、エラーの場合には -1 が返されます。

▼ 次のプロシージャは、SQL Server のストア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 カラムの値を返し、返された結果の数をアラートとして表示します。

```
$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 rows cursor (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 Set SQL in cursor, OC Execute cursor

OC CANCEL LOADING

OC CANCEL LOADING(*Cursor_ID***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC CANCEL LOADING コマンドは、カレントクエリーに関連する結果セットをすべて破棄します。このコマンドを使い、新しいクエリーを使用する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 Bind parameter

OC Bind parameter(*Cursor_ID;ReferenceNm;4DObject;IO Type***) → 整数**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
ReferenceNum	整数	参照番号
4DObject	文字列	4D 変数またはフィールド
IO Type	整数	IO タイプ

OC Bind parameter 関数は、4th Dimension オブジェクトをダイナミック SQL 文にバインドします。ダイナミック SQL 文は、SQL 文中の引数マーカーを置き換えることにより作成されます。SQL の引数マーカーはクエスチョンマーク（?）で示され、ホスト変数として使用されます。

Cursor_ID には必ず、あらかじめ作成してあるカーソル ID を指定してください。

ReferenceNum には、SQL 文で参照される引数の順序番号を指定します。

4DObject には、*ReferenceNum* で参照されるカラムの値を受け取るフィールドや変数、または配列の名前を指定します。

IO Type には、IO タイプを示す参照コードを指定します。指定できるコードは次の通りです。

I/O タイプ	コード
INPUT	1
INPUT_OUTPUT	2
RESULT_COL	3
OUTPUT	4
RETURN_VALUE	5

処理が正常に終了すると、OC Bind parameter から 1 が返され、エラーの場合には -1 が返されます。

▼ 次のプロシージャでは、SQL の insert 文を実行するために、配列の値を入力用の引数として使用します。

```
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
```

引数マーカーを使用できない箇所を次に示します。

場所	例
選択リスト中	SELECT ? from TABLE
比較文の両辺として	? >= ?
バイナリ命令 *operands の両辺として	? * ?
BETWEEN 句の第 1、第 2 オペランドとして	? BETWEEN ? and COL3
BETWEEN 句の第 1、第 3 オペランドとして	? BETWEEN Col4 and ?
IN 句の式および最初の値として	? IN (?, 10, 15, 20)
ユーナリープラスまたはマイナス オペランドとして	Col = -?
1 セットの関数の引数として	SELECT CUSTOMER_NAME, AVG(?) FROM CUSTOMER GROUP BY CUSTOMER _NAME

参照： OC Set SQL in cursor, OC Execute cursor

OC Describe parameter

OC Describe parameter(*Cursor_ID;ReferenceNum;4DObject***) → 整数**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
ReferenceNum	整数	参照番号
4DObject	文字列	4D 変数またはフィールドの名前

OC Describe parameter 関数は、カーソルに関連する、指定された引数のデータタイプの説明を取得します。

Cursor_ID には必ず、あらかじめ作成しアクティブな状態ではないカーソル ID を指定してください。

ReferenceNum には、SQL 文で参照される引数の順序番号を指定します。

4DObject には、整数タイプの 4th Dimension 変数の名前を指定します。この変数には、*ReferenceNum* で参照される引数のデータタイプの説明を格納します。各データタイプに対するコードは 68 ページの表を参照してください。

処理が正常に終了すると、OC Describe parameter から 1 が返され、エラーの場合には -1 が返されます。

参照： OC bind parameter, OC Number of parameters

OC Number of parameters

OC Number of parameters(*Cursor_ID***) → 整数**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC Number of parameters 関数は、カーソルに関連する引数の番号を返します。

Cursor_ID には必ず、あらかじめ作成したカーソル ID を指定してください。

エラーの場合には、OC Number of parameters から -1 が返されます。

参照： OC bind parameter, OC Describe parameter

OC SET CURSOR NAME

OC SET CURSOR NAME(*Cursor_ID; CursorName***)**

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

OC SET CURSOR NAME コマンドは、カーソルの名前を設定します。カーソル名は位置指定した UPDATES/DELETES 命令を実行する際に必要になります。位置指定した UPDATES/DELETES 命令では、結果セット内のカーソルの現在位置を使用して、現在処理されているローがどれであるかを判断します。SELECT FOR UPDATE 文が実行されると、ODBC では常にカーソル名は内部的に自動設定されます。しかし、このコマンドを使用すると、アプリケーションで意味を持つカーソル名を設定することができます。

Cursor_ID には必ず、あらかじめ作成したカーソル ID を指定してください。

CursorName には、*Cursor_ID* に割り当てる名前を指定します。

参照： OC Create cursor, OC Get cursor name

OC Get cursor name

OC Get cursor name(*Cursor_ID***) → 文字列**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC Get cursor name 関数はカーソル名を返します。

Cursor_ID には必ず、あらかじめ作成したカーソル ID を指定してください。

参照： OC Create cursor, OC SET CURSOR NAME

OC DROP CURSOR

OC DROP CURSOR (*Cursor_ID***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID

OC DROP CURSOR コマンドは、OC Create cursor コマンドで作成したカーソルにより使用されたメモリ領域を解放します。

Cursor_ID には必ず、あらかじめ作成したカーソル ID を指定してください。

▼ 次の例題はすでにオープンされていたカーソルをドロップします。

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

第 9 章　カタログコマンド

カタログコマンドを使い、データソースのカタログに保存されている情報を取得することができます。たとえば、テーブル一覧や、特定テーブルのカラム名の一覧、テーブルに関連するインデックスなどの情報です。カタログコマンドを使い、行える事柄は次の通りです。

- インストールしている ODBC ドライバの一覧を取得 (OC GET DSN LIST)
- データソーススキーマに関する情報（テーブル、カラム、インデックス等）を取得 (OC GET TABLE LIST, OC GET COLUMN LIST, OC GET PRIMARY KEY LIST)
- サーバプロシージャおよびそのカラムリストの一覧にアクセス (OC GET PROCEDURE LIST, OC GET PROCEDURE COLUMN LIST)
- テーブルとカラムに関する権限情報を取得 (OC GET TABLE PRIVILEGE LIST, OC GET COLUMN PRIVILEGE LIST)

OC GET DSN LIST

OC GET DSN LIST(*Cursor_ID; DB_Array; Mode*)

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
DSN_Array	文字列	4D 配列の名前
Desc_Array	文字列	4D 配列の名前

OC GET DSN LIST コマンドを使い、クライアントマシンで利用できる ODBC データソースを問い合わせることができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

DSN_Array には、文字列配列の名前を指定します。この配列に利用可能なデータソース一覧情報を受け取ります。

Desc_Array には、文字列配列の名前を指定します。この配列に利用可能なデータソースに関する詳細情報の一覧を受け取ります。

▼ 次の文では、利用可能なデータソース一覧とその詳細を配列にロードします。

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

OC GET TABLE LIST

OC GET TABLE LIST(*Cursor_ID; DB_Array; Owner_Array; Table_Array; Type_Array*)

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
DB_Array	文字列	4D 配列の名前
Owner_Array	文字列	4D 配列の名前
Table_Array	文字列	4D 配列の名前
Type_Array	文字列	4D 配列の名前

OC GET TABLE LIST コマンドを使い、ODBC データソースで利用可能なテーブル一覧を取得することができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

DB_Array には文字列配列の名前を指定します。この配列にデータベース名の一覧を受け取ります。

Owner_Array には文字列配列の名前を指定します。この配列に所有者名の一覧を受け取ります。

Table_Array には文字列配列の名前を指定します。この配列にテーブル名の一覧を受け取ります。

Type_Array には文字列配列の名前を指定します。この配列にテーブルタイプの一覧を受け取ります。

▼ 次のプロシージャでは、利用可能なテーブル一覧とその詳細を配列にロードします。

```
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")
```

OC GET COLUMN LIST

OC GET COLUMN LIST(*Cursor_ID; Table; Col_Array; Type_Array; Length_Array; Null_Array***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Table	文字列	テーブル名
Col_Array	文字列	4D 配列の名前
Type_Array	文字列	4D 配列の名前
Length_Array	文字列	4D 配列の名前
Null_Array	文字列	4D 配列の名前

OC GET COLUMN LIST コマンドを使い、ODBC データソースのテーブルのカラム一覧を取得することができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

Table には、取得しようとするカラム情報を持つテーブル名を指定します。

Col_Array には、文字列配列の名前を指定します。この配列にカラム名の一覧を受け取ります。

Type_Array には、文字列配列の名前を指定します。この配列にカラムタイプの一覧を受け取ります。

Length_Array には、文字列配列の名前を指定します。この配列にカラム長の一覧を受け取ります。

Null_Array には文字列配列の名前を指定します。この配列にカラムのヌルフィールドの詳細情報を受け取ります。

▼ 次のプロシージャでは、カラム一覧とその詳細を配列にロードします。

```
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(" テーブル名配列からテーブルを選択してください！ ")
End if
```

OC GET PRIMARY KEY LIST

OC GET PRIMARY KEY LIST(*Cursor_ID; Table; Col_Array; Sequence_Array; KeyName_Array***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Table	文字列	テーブル名
Col_Array	文字列	4D 配列の名前
Sequence_Array	文字列	4D 配列の名前
KeyName_Array	文字列	4D 配列の名前

OC GET PRIMARY KEY LIST を使い、ODBC データソースのテーブルのプライマリーキー一覧を取得することができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

Table には、取得しようとするカラム情報を持つテーブル名を指定します。

Col_Array には、文字列配列の名前を指定します。この配列にインデックスのベースとなるカラム一覧を受け取ります。

Sequence_Array には、インデックスの順序を指定します。

KeyName_Array にはインデックス名を指定します。

▼ 次のプロシージャでは、プライマリーキーの一覧とその詳細を配列にロードします。

```
ARRAY STRING(30;arCol;0)
ARRAY STRING(30;arSeq;0)
ARRAY STRING(30;arKey;0)
If(arTab#0)
   OC GET COLUMN LIST(cur_id;arTab{arTab};"arCol";"arSeq";"arKey")
Else
  ALERT(" テーブル名配列からテーブルを選択してください！ ")
End if
```

OC GET PROCEDURE LIST

OC GET PROCEDURE LIST(*Cursor_ID; DB_Array ; Owner_Array; Proc_Array; Type_Array***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
DB_Array	文字列	4D 配列の名前
Owner_Array	文字列	4D 配列の名前
Proc_Array	文字列	4D 配列の名前
Type_Array	文字列	4D 配列の名前

OC PROCEDURE LIST コマンドを使い、ODBCデータソースのストアドプロシージャ一覧を取得することができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

DB_Array には、文字列配列の名前を指定します。この配列にプロシージャが属すデータベース名の一覧を受け取ります。

Owner_Array には、文字列配列の名前を指定します。この配列にそのプロシージャの所有者名の一覧を受け取ります。

Proc_Array には、文字列配列の名前を指定します。この配列にプロシージャ名の一覧を受け取ります。

Type_Array には、文字列配列の名前を指定します。この配列にプロシージャタイプの一覧を受け取ります。

▼ 例題

```
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(" テーブル名配列からテーブルを選択してください！ ")
End if
```

OC GET PROCEDURE COLUMN LIST

OC GET PROCEDURE COLUMN LIST(*Cursor_ID; Procedure;Column_Array; Type_Array; Precision_Array***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Procedure	文字列	テーブル名
Column_Array	文字列	プロシージャ名
Type_Array	文字列	4D 配列の名前
Precision_Array	文字列	4D 配列の名前

The OC GET PROCEDURE COLUMN LIST コマンドを使い、*Procedure* に指定したプロシージャで使用されるカラムの一覧を取得することができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

Procedure には、取得したいカラム情報を含むプロシージャ名を指定します。

Column_Array には、文字列配列の名前を指定します。この配列にプロシージャで使用されるカラム名一覧を受け取ります。

Type_Array には文字列配列の名前を指定します。この配列にカラムのデータタイプ一覧を受け取ります。

Precision_Array には、文字列配列の名前を指定します。この配列にカラムの精度の値を受け取ります。

OC GET TABLE PRIVILEGE LIST

OC GET TABLE PRIVILEGE LIST(*Cursor_ID; Table ; Owner_Array; Grantor_Array;Grantee_Array;Priv_Array;Grant_Array***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Table	文字列	テーブル名
Owner_Array	文字列	4D 配列の名前
Grantor_Array	文字列	4D 配列の名前
Grantee_Array	文字列	4D 配列の名前
Priv_Array	文字列	4D 配列の名前
Grant_Array	文字列	4D 配列の名前

OC GET TABLE PRIVILEGE LIST コマンドにより、テーブルに関するパーミッションを知ることができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

Table には、取得したいカラム情報を含むテーブル名を指定します。

Owner_Array には、文字列配列の名前を指定します。この配列にカラムの所有者名一覧を受け取ります。

Grantor_Array には、文字列配列の名前を指定します。*Priv_Array* で指定した権限を与える者の一覧をこの配列に受け取ります。

Grantee_Array には文字列配列の名前を指定します。*Priv_Array* で指定した権限を与えられた者の一覧をこの配列に受け取ります。

Priv_Array には文字列配列の名前を指定します。この配列に権限の一覧を受け取ります。

Grant_Array には文字列配列の名前を指定します。許可できる属性の一覧をこの配列に受け取ります。

▼ 次のプロシージャでは、プライマリーキーの一覧とその詳細を配列にロードします。

```
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")
```

OC GET COLUMN PRIVILEGE LIST

OC GET COLUMN PRIVILEGE LIST(*Cursor_ID;Table; Owner_Array; Grantor_Array; Grantee_Array;Priv_Array;Grant_Array***)**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Table	文字列	テーブル名
Column	文字列	カラム名
Owner_Array	文字列	4D 配列の名前
Grantor_Array	文字列	4D 配列の名前
Grant_Array	文字列	4D 配列の名前
Priv_Array	文字列	4D 配列の名前
Grant_Array	文字列	4D 配列の名前

The OC GET COLUMN PRIVILEGE LIST コマンドにより、カラムに関するパーミッションを知ることができます。

Cursor_ID には必ず、あらかじめ作成した、アクティブな状態ではないカーソル ID を使用します。

Table には、取得したいカラム情報を含むカラム名を指定します。

Owner_Array には、文字列配列の名前を指定します。この配列にカラムの所有者名の一覧を受け取ります。

Grantor_Array には、文字列配列の名前を指定します。*Priv_Array* で指定した権限を与える者の一覧をこの配列に受け取ります。

Grantee_Array には、文字列配列の名前を指定します。*Priv_Array* で指定した権限を与えられた者の一覧をこの配列に受け取ります。

Priv_Array には、文字列配列の名前を指定します。この配列に権限の一覧を受け取ります。

Grant_Array には文字列配列の名前を指定します。この配列にテーブルタイプの一覧を受け取ります。

第 10 章　コントロールコマンド

この章では、4th Dimension へのデータのロードをキャンセルするためのコマンドや、エラー管理を行うためのコマンドを説明します。コントロールコマンドを使い、実行できる事柄は次の通りです。

- エラー処理用のプロシージャのインストール (OC SET ERROR HANDLER),
- デバッグウインドウのオープンとクローズ (OC OPEN DEBUG WINDOW、OC CLOSE DEBUG WINDOW),
- デバッグウインドウへのメッセージ表示 (OC DEBUG MESSAGE).

OC OPEN DEBUG WINDOW

OC OPEN DEBUG WINDOW

OC OPEN DEBUG WINDOW コマンドは、ウインドウをオープンして 4D ODBC コマンドをコールし、リアルタイムでのトレースを行います。

このウインドウにはメニューバーがあり、次のメニューを使用できます。

- **File メニュー：**このメニューには、ウインドウの内容をプリント、クローズ、保管するためのコマンドがあります。
- **Options メニュー：**このメニューを使用し、ウインドウに表示するデバッグ情報レベルを設定します。たとえば、プロシージャの引数、結果、エラー、メッセージを表示するか、またはファンクション名だけにするかというような設定が行えます。

参照：OC CLOSE DEBUG WINDOW, OC DEBUG MESSAGE

OC SET ERROR HANDLER

OC SET ERROR HANDLER (*ProcedureName*)

引数	タイプ	説明
ProcName	文字列	エラー処理プロシージャの名前

OC SET ERROR HANDLER コマンドにより、エラー処理プロシージャをインストールし、エラーが起きる度にこのプロシージャを実行することができます。したがって、実行時エラーを制御し、デフォルトのエラー処理を無効にすることが可能です。

*ProNam*e には、インストールするプロシージャ名を指定します。

デフォルトのエラー処理に戻すには、空の文字列を渡します（例：OC SET ERROR HANDLER("")）。

4D ODBC からは次の 5 種類の引数がプロシージャへ渡されます。

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	エラー内容
$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

データベースをコンパイルする場合には、C_LONGINT や C_TEXT 等のコマンドを使い、$1 から $5 までの変数を必ず宣言してください。

デフォルトでは、エラー処理プロシージャはインストールされず、4D ODBC によりすべてのエラーが制御され、エラーメッセージが表示されます。

OC CLOSE DEBUG WINDOW

OC CLOSE DEBUG WINDOW

OC CLOSE DEBUG WINDOW コマンドは、OC OPEN DEBUG WINDOW コマンドでオープンされたデバッグウインドウをクローズします。

参照： OC OPEN DEBUG WINDOW, OC DEBUG MESSAGE

OC DEBUG MESSAGE

OC DEBUG MESSAGE (*Text*)

引数	タイプ	説明
Text	テキスト	メッセージテキスト

OC DEBUG MESSAGE コマンドは、OC OPEN DEBUG WINDOW コマンドでオープンされたデバッグウインドウ上にテキストを表示します。

参照： OC OPEN DEBUG WINDOW, OC CLOSE DEBUG WINDOW

OC TRANSACT COMMAND

OC TRANSACT COMMAND(Login_ID;Cursor_ID;Commit/Rollback)

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
Cursor_ID	倍長整数	カーソル ID
Commit/Rollback	倍長整数	0 = コミット
		1 = ロールバック

OC TRANSACT COMMAND コマンドは、トランザクションのコミットやロールバックを行います。

トランザクションを使用するには、まず OC Set login オプションを使い、接続をマニュアル－コミットモードに設定する必要があります。

データソースの中にはトランザクションをまったくサポートしていないものがあり、また、トランザクションに組み込める文を制限しているものもあります。どのような種類のトランザクション処理がデータソースでサポートされているかを調べるには、OC Get information を使用して SQL_TXN_CAPABLE の値を取得します。

参照： OC Set login option

第 11 章 コンフィグレーションコマンド

この章では、ドライバやログイン、カーソル、データタイプに関する各種情報の取得や設定を行うためのコマンドについて説明します。コンフィグレーションコマンドを使い、実行できる事柄は次の通りです。

- ODBC ドライバに関する情報を取得し、ドライバでサポートされている API 機能を調べる (OC Get function、OC Get Info),
- 接続オプションを取得し、設定する (OC Get login option、OC Set login option),
- カーソルオプションを取得し、設定する (OC Get cursor option、OC Set cursor option),
- 接続に関する情報を取得する (OC GET TYPE INFO LIST).

OC Get function

OC Get function(*Login_ID***;** *Function_ID***;** *4DObject***) → 整数**

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
Function_ID	倍長整数	関数の参照番号
4DObject	文字列	4D 変数名、またはフィールド名

OC Get function 関数を使い、特定の ODBC 機能がドライバでサポートされるかどうかを判断することができます。

Login_ID には有効なログイン ID を指定します。

Function_ID には ODBC 関数の参照番号を指定します。参照番号の一覧は、このルーチンの説明の後に記載します。

4DObject には 4th Dimension 変数の名前を指定します。関数がサポートされている場合、その関数名がこの変数に格納されます。

関数がサポートされている場合、OC Get function からは 1 が返され、それ以外の場合には -1 が返されます。

▼ 次のプロシージャでは、ユーザに関数の ID 番号の入力を求め、その関数がサポートされている場合に関数名を返します。

```
C_STRING(30;var1)
$num:=Request("Enter ID #")
$res:=OC Get Function (login;Num($num);var1)
If ($res=1)
   ALERT(var1+" はサポートされています ")
Else
   ALERT (" サポートされていません ")
End If
```

FunctionID に指定する参照番号は次の通りです。

各レベルの ODBC 関数	ID	4D ODBC でのサポート
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
Level 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

各レベルの ODBC 関数	ID	4D ODBC でのサポート
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
Level 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
SDK 2.0 Additions		
SQL_API_SQLDRIVERS	71	
SQL_API_SQLBINDPARAMETER	72	

OC Get info

OC Get info(*Login_ID*; *Info_ID*) → 文字列

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
Info_ID	倍長整数	関数の参照番号

OC Get info 関数は、HDBC に関連するドライバやデータソースについての一般情報を取得します。

Login_ID には有効なログイン ID を指定します。

Info_ID には求める情報の参照番号を指定します。この参照番号を次に示します。

類別	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_DBMS_NAME	16
SQL_DBMS_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

類別	ID
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_CONVERT_REAL	64
SQL_CONVERT_SMALLINT	65
SQL_CONVERT_TIME	66
SQL_CONVERT_TIMESTAMP	67
SQL_CONVERT_TINYINT	68
SQL_CONVERT_VARBINARY	69
SQL_CONVERT_VARCHAR	70
SQL_CONVERT_LONGVARBINARY	71
SQL_TXN_ISOLATION_OPTION	72
SQL_ODBC_SQL_OPT_IEF	73
SQL_CORRELATION_NAME	74
SQL_NON_NULLABLE_COLUMNS	75
SQL_DRIVER_HLIB	76
SQL_DRIVER_ODBC_VER	77
SQL_LOCK_TYPES	78
SQL_POS_OPERATIONS	79
SQL_POSITIONED_STATEMENTS	80
SQL_GETDATA_EXTENSIONS	81
SQL_BOOKMARK_PERSISTENCE	82

類別	ID
SQL_STATIC_SENSITIVITY	83
SQL_FILE_USAGE	84
SQL_NULL_COLLATION	85
SQL_ALTER_TABLE	86
SQL_COLUMN_ALIAS	87
SQL_GROUP_BY	88
SQL_KEYWORDS	89
SQL_ORDER_BY_COLUMNS_IN_SELECT	90
SQL_OWNER_USAGE	91
SQL_QUALIFIER_USAGE	92
SQL_QUOTED_IDENTIFIER_CASE	93
SQL_SPECIAL_CHARACTERS	94
SQL_SUBQUERIES	95
SQL_UNION	96
SQL_MAX_COLUMNS_IN_GROUP_BY	97
SQL_MAX_COLUMNS_IN_INDEX	98
SQL_MAX_COLUMNS_IN_ORDER_BY	99
SQL_MAX_COLUMNS_IN_SELECT	100
SQL_MAX_COLUMNS_IN_TABLE	101
SQL_MAX_INDEX_SIZE	102
SQL_MAX_ROW_SIZE_INCLUDES_LONG	103
SQL_MAX_ROW_SIZE	104
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

▼ 次のプロシージャは、SQL_POSITIONED_STATEMENTS 関数（ID = 80）の情報を取得し、その結果を変数 $result に格納します。

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

処理が正常に終了すると、OC Get info より求める情報が返されます。それ以外の場合には、空の文字列が返されます。

OC Get login option

OC Get login option(*Login_ID*; *Option_ID*) → 文字列

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
Option_ID	倍長整数	関数の参照番号

OC Get login option 関数は、接続オプションの値を取得します。

Login_ID には有効なログイン ID を指定します。

Option_ID には求めるオプションの参照番号を指定します。参照番号の一覧は、このルーチンの説明の後に記載します。

▼ 次のプロシージャは、SQL_AUTOCOMMIT オプション（ID = 102）の情報を取得し、その結果を変数 $result に格納します。

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

処理が正常に終了すると、OC Get login option より求める情報が返されます。それ以外の場合には、空の文字列が返されます。

引数 *Option_ID* に指定する参照番号を次に示します。

類別	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

OC Set login option(*Login_ID*; *Option_ID;Option_Value*) → **文字列**

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
Option_ID	倍長整数	関数の参照番号
Option_Value	文字列	オプションに設定する値

The OC Set login option 関数を使用すると、アプリケーションから接続属性を設定することができます。

Login_ID には有効なログイン ID を指定します。

Option_ID には目的とするオプションの参照番号を指定します。参照番号の一覧は、このルーチンの説明の後に記載します。

Option_Value にはオプションに設定する値を指定します。

▼ 次のプロシージャは、SQL_AUTOCOMMIT オプション（ID = 102）を手動（値 = 0）に設定し、その結果を変数 $result に格納します。

$result:=***OC Set login option***(login;102;0)

処理が正常に終了すると、OC Set login option から 1 が返されます。エラーの場合には 0 が、*Option_ID* が有効範囲外の値であれば -1 が返されます。

Option_ID の参照番号は次の通りです。

類別	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 Get cursor option

OC Get cursor option(*Login_ID*; *Option_ID*) → 文字列

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Option_ID	倍長整数	関数の参照番号

OC Get login option 関数は、カーソルの設定情報を取得します。

Cursor_ID には必ず、あらかじめ作成したカーソル ID を指定します。

Option_ID には求めるオプションの参照番号を指定します。参照番号の一覧は、このルーチンの説明の後に記載します。

▼ 次のプロシージャは、SQL_AUTOCOMMIT オプション（ID = 102）の情報を取得し、その結果を変数 $result に格納します。

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

処理が正常に終了すると、OC Get cursor option より求める情報が返されます。それ以外の場合には、空の文字列が返されます。

Option_ID の参照番号は次の通りです。

類別	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

OC Set cursor option(*Login_ID*; *Option_ID;Option_Value*) → **文字列**

引数	タイプ	説明
Cursor_ID	倍長整数	カーソル ID
Option_ID	倍長整数	関数の参照番号
Option_Value	文字列	オプションに設定する値

OC Set login option 関数を使用すると、アプリケーションからカーソルに関する設定情報をセットすることができます。

Cursor_ID には必ず、あらかじめ作成したカーソル ID を指定します。

Option_ID. には求めるオプションの参照番号を指定します。参照番号の一覧は、このルーチンの説明の後に記載します。

Option_Value には、オプションに設定する値を指定します。

▼ 次のプロシージャは、SQL_AUTOCOMMIT オプション（ID = 102）を手動（値 = 0）に設定し、その結果を変数 $result に格納します。

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

処理が正常に終了すると、OC Set cursor option から 1 が返されます。エラーの場合には 0 が、*Option_ID* が有効範囲外の値であれば -1 が返されます。

Option_ID の参照番号は次の通りです。

類別	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 GET TYPE INFO LIST

OC GET TYPE INFO LIST(*Login_ID*; *Option_ID*)

引数	タイプ	説明
Login_ID	倍長整数	ログイン ID
Attribute_ID	整数	関数の参照番号
4DObject	文字列	4D 配列名

OC Get login option 関数を使用すると、アプリケーションで接続属性を取得することができます。

Login_ID には、有効なログイン ID を指定します。

Attribute_ID には、求めるデータタイプ属性の参照番号を指定します。参照番号の一覧は、このルーチンの説明の後に記載します。

4DObject には、4th Dimension の文字列配列を指定し、この配列に結果が格納されます。

▼ 次の例題では、データソースのデータタイプの名前、長さ、ヌル属性を取得します。

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

Attribute_ID の参照番号は次の通りです。

データタイプ属性	ID
Name	1
Data Type	2
Precision	3
Literal Prefix	4
Literal Suffix	5
Create Params	6
Nullable	7
Case Sensitive	8
Searchable	9
Unsigned	10
Money auto increment	11
Local Type Name	12
Min scale	13
Max scale	14

付録 A　エラーコード

4D ODBC. から返されるエラーコードの一覧を次に示します。

エラー	エラーメッセージ
1	コマンドを指定せずに実行呼び出しが行われました。
2	プロセスを割り当てずに実行呼び出しが行われました。
3	このバインドに Macintosh のメモリを割り当てられません。
4	本バージョンでは、この 4D タイプはサポートされていません。
5	本バージョンではピクチャ配列はサポートされていません。
6	プロシージャをコピーできません。メモリ不足です。
7	異なる 2 つの 4D ファイルのフィールドをバインドすることはできません。
8	このコンテキストにバインドを設定することはできません。
9	4D ポインタ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 の二次元配列はサポートされていません。
12	配列は 4D に割り当てられませんでした。
13	新しい配列の割り当て中にメモリの問題が発生しました。
14	指定したログイン引数が正しくありません。ログインしなかったか、あるいは変数を変更しました。
15	引数で渡されたカーソルは無効であるか、あるいはクローズされています。
16	指定したコンテキスト引数は無効です。
17	引数の 1 つが正しくありません。プログラムをチェックしてください。
18	サポートされていないタイプです。
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、破棄されています。
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ための十分なメモリがありません。
23	コンテキストがアクティブではありません。このコマンドの使用の前に、コンテキストをアクティブにしてください。
24	このコンテキストは既にアクティブではありません。
25	このコンテキストは既にアクティブです。
26	この処理の前に、コンテキストがクローズされています。

エラー	エラーメッセージ
27	このコンテキストでは update モードを使用できません。
28	この制限は無効です。ゼロ以上の値を指定してください。
29	句の ID が無効です。
30	このバージョンの 4D ODBC は、4th Dimension バージョン 3.02 または 4D Server1.0.2 以上でのみ使用できます。
31	コンテキストファイルが無効です。
32	コンテキストファイルが見つかりません。
33	コンテキスト読み込み中にエラーが発生しました。
34	このピクチャはコンテキストではありません。
35	この関数の ID は範囲外です。
36	情報のタイプが無効であるか、または 4D ODBC でサポートされていません。
37	オブジェクトが大きすぎます。ローは追加されず、更新されません。
38	エラーのため 4D にレコードは作成されません。
39	エラーのため 4D にレコードは保存されません。
40	引数として渡された配列が定義されていないか、タイプが正しくありません。

付録 B　SQL GetInfo 関数

ODBC 1.0 では、SQLGetInfo 関数が導入されました。Extension Level は 1です。この関数は、*hdbc* に関連するドライバやデータソースの一般情報を返します。

構文

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

SQLGetInfo 関数で使用できる引数を次に示します。

タイプ	引数	用法	説明
HDBC	hdbc	入力	接続ハンドル
UWORD	fInfoType	入力	情報タイプ
PTR	rgbInfoValue	出力	情報の保管場所へのポインタ
SWORD	cbInfoValueMax	入力	rgbInfoValue バッファの最大長
*SWORD FAR**	pcbInfoValue	出力	rgbInfoValue へ返すため使用できる合計バイト数（文字データのヌルの最終バイトを除く）

HDBC は接続ハンドルです。

UWORD は情報タイプです。*fInfoType* には、求めるタイプを表わす値を必ず指定します。詳細は、「コメント」の節を参照してください。

PTR は情報の保管場所を示すポインタです。要求した *fInfoType* に応じて、返される情報は、ヌルで終わる文字列、16 ビットの整数値、32 ビットのフラグ、32 ビットのバイナリ値のいずれかです。

SWORD は、*rgbInfoValue* バッファの最大長です。

*SWORD FAR** は *rgbInfoValue* に返すために利用できる合計バイト数（文字データのヌルの最終バイトを除く）です。文字データの場合、返される情報の長さが *cbInfoValueMax* と同じか、それよりも長ければ、ドライバによってこの情報は *cbInfoValueMax*-1 バイトの長さになるように切り捨てられ、最終バイトはヌルになります。その他のデータタイプの場合、*cbValueMax* の値は無視され、ドライバでは *rgbValue* のサイズが 32 ビットであると見なされます。

返り値

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

SQLGetInfo から SQL_ERROR、または SQL_SUCCESS_WITH_INFO が返された場合に、SQLError をコールすると、関連する SQLSTATE の値を取得できることがあります。次の表は、SQLGetInfo から返される一般的な SQLSTATE の値を示します。また、この関数のコンテキストを説明しています。ドライバマネージャから返される SQLSTATE の場合、説明の前に " (DM) " という注釈が付いています。特に記載がないかぎり、各 SQLSTATE の値に関連する返り値は SQL_ERROR です。

SQLSTATE	エラー	説明
01000	一般的な警告	Driver 特定の情報メッセージ（関数は SQL_SUCCESS_WITH_INFO を返す ）
01004	データ切り捨て	バッファ rgbInfoValue の長さが充分ではないため、要求した情報をすべて返せない。したがって、情報は切り捨てられる。引数 pcbInfoValue には、切り捨てられない状態での要求した情報の長さが格納される。（関数は SQL_SUCCESS_WITH_INFO を返す ）
08003	接続がオープンされていない	（DM）fInfoType で指定した情報タイプを求めるには接続がオープンされている必要がある。ODBC で予約されている情報タイプについては、接続がオープンされていない場合でも SQL_ODBC_VER だけを返すことができる。
22003	範囲外の数値	要求した情報を返すと、数値やバイナリの有意性が失われる可能性がある。
IM001	この機能はドライバでサポートされない	（DM）hdbc に対応するドライバでは、この機能はサポートされない
S1000	一般的なエラー	特定の SQLSTATE が存在しないために起きるエラー、または実現する特定の SQLSTATE が定義されていないため起きるエラー。エラーの説明とその原因は、SQLError から返されたエラーメッセージに含まれ、これは引数 szErrorMsg に格納されている。
S1001	メモリ割り当て失敗	ドライバは関数の実行や完了のために必要なメモリを割り当てることができない。.

SQLSTATE	エラー	説明
S1009	引数の値が無効	（DM）fInfoType が SQL_DRIVER_HSTMT であった。また、rgbInfoValue で指された値が有効なステートメントハンドルではなかった。
S1090	文字列、またはバッファ長が無効	（DM）引数 cbInfoValueMax に指定した値が 0 より小さい。
S1096	情報タイプが範囲外	（DM）引数 fOption に指定した値は、ODBC の情報タイプとして予約されている数値であったが、ドライバでサポートされる ODBC のバージョンでは無効。
S1C00	ドライバ動作不可	引数 fOption に指定した値は、ドライバ特定の情報タイプとして予約されている数値であったが、このドライバではサポートされていない。
S1T00	タイムアウト	要求した情報がデータソースから返される前に、タイムアウト時間が終了した。タイムアウト時間の設定は、SQL-SetStmtOption と SQL_QUERY_TIMEOUT で行う。

コメント

この節では、現時点で定義されている情報タイプについて説明します。異なるデータソースを活用するため、これらの定義はさらに増える予定です。0 から 999 までのデータタイプは、ODBC で予約されています。したがって、ドライバを特定して使用するには、ドライバ開発者が SQL_INFO_DRIVER_START より大きな値を予約する必要があります。詳細は、第 11 章の「ODBC 関数を実行するためのガイドライン」に記載されているドライバ特定のデータタイプや類別タイプ、情報タイプ、オプションを参照してください。

rgbInfoValue に返される情報の形式は、要求した *fInfoType* に応じて変わります。SQLGetInfo から返される情報は、次の 5 種類の形式のうちのひとつです。

- ヌルで終わる文字ストリング
- 16 ビットの整数値
- 32 ビットのビットマスク
- 32 ビットの整数値
- 32 ビットのバイナリ値

次の情報タイプの各形式は、タイプ説明のところに記載されています。アプリケーションでは場合に応じ、*rgbInfoValue* に返された値をキャストする必要があります。アプリケーションで 32 ビットのビットマスクからデータを取得する方法の例は、この節にあるコード例題を参照してください。

次の表に示す情報タイプに対し、ドライバから必ず各々の値が返されます。ある情報タイプがドライバやデータソースで適用されない場合、次の値のうちひとつがドライバから返されます。

rgbInfoValue の形式	返り値
文字列（“Y”または“N”）	“N”
文字列（“Y”または“N”）	空の文字列
16 ビットの整数	0
32 ビットのビットマスク、または 32 ビットのバイナリ値	0L

データソースでプロシージャがサポートされていない場合、SQLGetInfo からはプロシージャに関係する *fInfoType* の値として次の値が返されます。

fInfoType	返り値
SQL_PROCEDURES	ﾒNﾓ
SQL_ACCESSIBLE_PROCEDURES	ﾒNﾓ
SQL_MAX_PROCEDURE_NAME_LEN	0
SQL_PROCEDURE_TERM	空の文字列

fInfoType の値として、SQLGetInfo から SQLSTATE の S1096（引数の値が無効）が返されます。この値は、ODBC で使用するために予約された情報タイプの範囲内にありますが、ドライバでサポートされないバージョンの ODBC によって定義されています。ドライバに適合する ODBC のバージョンを判断するには、アプリケーションから SQLGetInfo をコールし、情報タイプに SQL_DRIVER_ODBC_VER を指定します。*fInfoType* の値として、SQLGetInfo から SQLSTATE の S1C00（ドライバ動作不可）が返されます。この値は、ドライバ特定として予約された情報タイプの範囲内にありますが、このドライバではサポートされていません。

注：*アプリケーション開発者は、ODBC 1.0 のドライバから fInfoType の値として、SQL_ERROR や SQLSTATE S1C00（ドライバ動作不可）という値が返される可能性があることに注意してください。この値は、ODBC 1.0 で定義されていますが、ドライバやデータソースには適用されません。*

情報タイプ

この節では、SQLGetInfo でサポートされる情報タイプについて説明します。情報タイプは種類別に分類され、アルファベット順に記載されています。

ドライバ情報

次に示す *fInfoType* の値は、ODBC ドライバに関する情報を返します。たとえば、アクティブなステートメントの番号や、データソース名、API 準拠レベルなどの情報です。

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 プロダクト情報

次に示す *fInfoType* の値は、DBMS プロダクトに関する情報を返します。たとえば、DBMS 名やバージョンなどの情報です。

SQL_DATABASE_NAME
SQL_DBMS_NAME
SQL_DBMS_VER

データソース情報

次に示す *fInfoType* の値は、データソースに関する情報を返します。たとえば、カーソル特性やトランザクション機能の情報などです。

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

サポートされる SQL

次に示す *fInfoType* の値は、データソースでサポートされる SQL 文の情報を返します。これらの情報タイプは、ODBC の SQL 構文規約をすべて網羅するわけでは

ありません。しかし、データソースで異なるレベルのサポートを受けるために、通常必要となる文法が説明されています。

アプリケーション内で情報タイプ SQL_ODBC_SQL_CONFORMANCE を使用し、サポートされる構文の総体的なレベルを判断し、さらに別の情報タイプを使って提示された準拠レベルの種類を判断してください。

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

次に示す *fInfoType* の値は、SQL 文の識別子や句に適用される制限についての情報を返します。たとえば、識別子の最大長や、選択リスト内のカラム数上限などの制限です。ドライバまたはデータソースにより制限されます。

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	

スカラー関数情報

次に示す *fInfoType* の値は、データソースとドライバでサポートされるスカラー関数についての情報を返します。

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

次に示す *fInfoType* の値は、SQL のデータタイプの一覧を返します。データソースで CONVERT スカラー関数を使用して、特定の SQL データタイプをこれらのデータタイプに変換できます。

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	

情報タイプについて

この節では、それぞれの情報タイプをアルファベット順に説明します。また、その情報タイプが導入された ODBC のバージョンも示します。

- SQL_ACCESSIBLE_PROCEDURES (ODBC 1.0)
  文字列。ユーザが SQLProcedures から返されるすべてのプロシージャを実行できる場合には “ Y ”。ユーザが実行できないプロシージャがある場合には “ N ” になります。
- SQL_ACCESSIBLE_TABLES (ODBC 1.0)
  文字列。ユーザが SQLtables から返されるすべてのテーブルに対し SELECT 権限を得られる場合には “ Y ” になります。ユーザがアクセスできないテーブルがある場合には “ N ” になります。
- SQL_ACTIVE_CONNECTIONS (ODBC 1.0)
  16 ビットの整数値。ドライバでサポートされるアクティブな *hdbc* の最大数を特定します。この値は、ドライバまたはデータソースで規定された限界値を表わしている場合があります。制限が指定されていない場合や、わからない場合、この値はゼロになります。
- SQL_ACTIVE_STATEMENTS (ODBC 1.0)
  16 ビットの整数値。*hdbc* 用としてドライバでサポートされるアクティブな *hstmts* の最大数を特定します。この値は、ドライバまたはデータソースで規定された限界値を表わしている場合があります。制限が指定されていない場合や、わからない場合、この値はゼロになります。
- SQL_ALTER_TABLE (ODBC 2.0)
  32 ビットのビットマスク。ALTER TABLE 文において、データソースでサポートされる句を一覧します。次のビットマスクは、サポートされる句を調べるために使用されています。
  SQL_AT_ADD_COLUMNSQL_AT_DROP_COLUMN

- SQL_BOOKMARK_PERSISTENCE (ODBC 2.0)
  32 ビットのビットマスク。ブックマークが維持される操作を一覧します。次のビットマスクは、ブックマークが維持される操作を判断するために、フラグとともに使用されます。
  SQL_BP_CLOSE = ブックマークが有効になるのは、アプリケーションで SQL_CLOSE オプションを指定し、*hstmt* に関連するカーソルをクローズした後である。
  SQL_BP_DELETE = ローのブックマークは、ローが削除された後に有効になる。
  SQL_BP_DROP = ブックマークが有効になるのは、アプリケーションで SQL_DROP オプションを指定して、SQL-FreeStmt をコールし、*hstmt* をドロップした後である。
  SQL_BP_SCROLL = ブックマークはスクロール操作（SQLExtendedFetch をコールする）を行った後に有効になる。SQLExtendedFetch をコールした後、すべてのブックマークは必ず有効な状態であることが必要なので、この値をアプリケーションで利用し、ブックマークがサポートされているかどうかの判断に使用できる。
  SQL_BP_TRANSACTION = ブックマークが有効になるのは、アプリケーションでトランザクションのコミットやロールバックを行った後である。
  SQL_BP_UPDATE = ローのブックマークが有効になるのは、そのローのカラム（キーカラムを含む）が更新された後である。
  SQL_BP_OTHER_HSTMT = ある *hstmt* に関連するブックマークを別の *hstmt* と一緒に使用することができる。
- SQL_COLUMN_ALIAS (ODBC 2.0)
  文字列。データソースでカラムエイリアスがサポートされている場合は “ Y ”、サポートされていない場合は “ N ” になります。
- SQL_CONCAT_NULL_BEHAVIOR (ODBC 1.0)
  16 ビットの整数値。データソースにおいて、ヌル値の文字タイプカラムとノンヌル値の文字タイプカラムの結合の取り扱い方を示します。
  SQL_CB_NULL = 結果はヌル値になる。
  SQL_CB_NON_NULL = 結果はノンヌル値のカラムになる。
- 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

  (ODBC 1.0)
  32 ビットのビットマスク。*fInfoType* に指定したタイプ名のデータを求めるため、CONVERT スカラー関数を使用した際に、データソースでサポートされる変換を示します。ビットマスクがゼロの場合、そのタイプ名のデータの変換はデータソースでサポートされません。同じデータタイプでの変換もサポートされません。

たとえば、データソースで SQL_INTEGER データの SQL_BIGINT への変換がサポートされているかどうかを調べるには、アプリケーションから SQLGetInfo をコールし、*fInfoType* に SQL_CONVERT_INTEGER を指定します。アプリケーションでは、返されたビットマスクと SQL_CVT_BIGINT との間で AND 演算が行われます。この結果がゼロ以外の値であれば、その変換はサポートされています。

サポートされる変換を調べるため、次のビットマスクが使用されます。

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)
  32 ビットのビットマスク。ドライバと関連するデータソースにおいてサポートされるスカラー変換関数を一覧します。サポートされる変換関数を調べるために、次のビットマスクが使用されます。
  SQL_FN_CVT_CONVERT
- SQL_CORRELATION_NAME (ODBC 1.0)
  16 ビットの整数値。テーブル相関名がサポートされるかどうかを示します。
  SQL_CN_NONE = 相関名はサポートされない。
  SQL_CN_DIFFERENT = 相関名はサポートされるが、それが示すテーブル名と異なる名称を指定しなくてはならない。
  SQL_CN_ANY = 相関名はサポートされ、ユーザは有効な名称を何でも定義できる。
- SQL_CURSOR_COMMIT_BEHAVIOR (ODBC 1.0)
  16 ビットの整数値。データソースのカーソルや準備された文に対し、COMMIT 操作が与える影響を表わします。
  SQL_CB_DELETE = カーソルはクローズされ、準備された文は削除されます。このカーソルをまた使用するには、再度アプリケーションで *hstmt* の準備と実行を行わねばなりません。
  SQL_CB_CLOSE = カーソルはクローズされます。準備された文に対し、*hstmt* 上でアプリケーションから SQLExecute をコールすることが可能であり、再度 SQLPrepare をコールする必要はありません。
  SQL_CB_PRESERVE = カーソルはCOMMIT操作の実行前と同じ位置に保たれます。再度 *hstmt* を準備しなくても、アプリケーションでは引き続きデータを取り込むことができ、またはカーソルをクローズして再び *hstmt* を実行することができます。

- SQL_CURSOR_ROLLBACK_BEHAVIOR (ODBC 1.0)
  16 ビットの整数値。データソースのカーソルや準備された文に対して ROLLBACK 操作が与える影響を示します。
  SQL_CB_DELETE = カーソルはクローズされ、準備された文は削除されます。このカーソルを再び使用するには、アプリケーションで再度 *hstmt* の準備と実行を行わねばなりません。
  SQL_CB_CLOSE = カーソルはクローズされます。準備された文に対し、*hstmt* 上でアプリケーションから SQLExecute をコールすることが可能であり、再度 SQLPrepare をコールする必要はありません。
  SQL_CB_PRESERVE = カーソルは ROLLBACK 操作の実行前と同じ位置に保たれます。再度 *hstmt* を準備しなくても、アプリケーションでは引き続きデータを取り込むことができ、またはカーソルをクローズして再び *hstmt* を実行することができます。
- SQL_DATA_SOURCE_NAME (ODBC 1.0)
  文字列。接続中に使用されるデータソース名を示します。アプリケーションから SQLConnect をコールした場合には、引数 *szDSN* の値となります。アプリケーションから SQLDriverConnect や SQLBrowseConnect をコールした場合には、ドライバに渡された接続文字列である DSN キーワードの値となります。接続文字列に DSN キーワードが含まれていない場合（例：DRIVER キーワードを含む場合等）には、空の文字列になります。
- SQL_DATA_SOURCE_READ_ONLY (ODBC 1.0)
  文字列。データソースが READ ONLY モードに設定されている場合には “ Y ”、それ以外は N ” になります。この値は、データソースにのみ関連し、データソースへのアクセスを左右するドライバには影響を与えません。
- SQL_DATABASE_NAME (ODBC 1.0)
  文字列。データソースで “ database ” という名前のオブジェクトが定義された場合に、現在使用中のデータベース名を示します。
  注：ODBC 2.0 において、この *fInfoType* の値は、接続オプション SQL_CURRENT_QUALIFIER の値に置き換えられています。ODBC 2.0 では、情報タイプ SQL_DATABASE_NAME が引き続きサポートされるはずです。ODBC 2.0 アプリケーションで ODBC 1.0 ドライバを使用する場合には、こちらの値だけを使用してください。
- SQL_DBMS_NAME (ODBC 1.0)
  文字列。ドライバからアクセスされる DBMS プロダクト名を示します。
- SQL_DBMS_VER (ODBC 1.0)
  文字列。ドライバからアクセスされる DBMS プロダクトのバージョンを示します。このバージョンは ##.##.#### という形式で示され、最初の 2 桁はメジャーバージョン、その次の 2 桁がマイナーバージョン、最後の 4 桁がリリースバージョンを表わします。ドライバは、この形式に従って DBMS プロダクトのバージョンを返さなくてはなりません。しかし、その DBMS プロダクト固有のバージョンを付け加えることもできます（例：“ 04.01.0000 Rdb 4.1 ”）。

- SQL_DEFAULT_TXN_ISOLATION (ODBC 1.0)
  :32 ビットの整数値。ドライバやデータソースでサポートされるデフォルトトランザクションのアイソレーションレベルを示し、データソースでトランザクションがサポートされていない場合にはゼロになります。トランザクションのアイソレーションレベルを定義するため、次の用語が使用されます。
  Dirty Read - トランザクション 1 がローの変更を行った。トランザクション 1 がこの変更をコミットする前に、トランザクション 2 は変更されたローを読み込んだ。トランザクション 1 がこの変更をロールバックした場合、トランザクション 2 は存在しないと思われるローを読み込むことになる。
  Nonrepeatable Read - トランザクション 1 がローを読み込んだ。トランザクション 2 はそのローの更新、または削除を行い、この変更をコミットした。トランザクション 1 が再びそのローを読み込むと、ローの内容が異なっているか、またはそのローが削除されている。
  Phantom - トランザクション 1 がある検索条件を満たす一連のローを読み込んだ。トランザクション 2 がその検索条件に合致するローを追加した。再度トランザクション 1 がそれらのローを読み込む文を実行すると、異なるセットのローを受け取ることになる。データソースでトランザクションがサポートされている場合、次のビットマスクのうちひとつがドライバから返される。
  SQL_TXN_READ_UNCOMMITTED = Dirty Read、Nonrepeatable Read、Phantom が可。
  SQL_TXN_READ_COMMITTED = Dirty Read は不可。Nonrepeatable Read、Phantom は可。
  SQL_TXN_REPEATABLE_READ = Dirty Read、Nonrepeatable Read は不可。Phantom は可。
  SQL_TXN_SERIALIZABLE = トランザクションの順序付けが可能。Dirty Read、Nonrepeatable Read、Phantom は不可。
  SQL_TXN_VERSIONING = トランザクションの順序付けは可能だが、SQL_TXN_SERIALIZABLE を使用した時よりも衝突が多発する可能性がある。Dirty Read は不可。特に SQL_TXN_SERIALIZABLE では、衝突を少なくするロックプロトコルを使用することにより、これを実現しているが、SQL_TXN_VERSIONING では、レコードバージョン付けなどのノンロックプロトコルを使用している。SQL_TXN_VERSIONING の例としては、Oracle の Read Consistency アイソレーションレベルがある。
- SQL_DRIVER_HDBCSQL_DRIVER_HENV (ODBC 1.0)
  32 ビットの値。ドライバの環境ハンドルや接続ハンドルを示し、引数 *hdbc* によって判断されます。これらの情報タイプはドライバマネージャでのみ実現されます。
- SQL_DRIVER_HLIB (ODBC 2.0)
  32 ビットの値。ドライバ DLL がロードされた際に、ドライバマネージャに返されるライブラリハンドルを示します。このハンドルは、SQLGetInfo をコールした時に指定した *hdbc* にのみ有効です。この情報タイプは、ドライバマネージャでしか実現されません。
- SQL_DRIVER_HSTMT (ODBC 1.0)
  A 32-bit value, the driver's statement handle determined by the Driver Manager statement handle, which is passed on input in *rgbInfoValue* from the application. Note that in this case, *rgbInfoValue* is both an input and an output argument. The input hstmt passed in *rgbInfoValue* must have been an *hstmt* allocated on the argument *hdbc*.This information type is implemented by the Driver Manager alone.32 ビットの値。ドラ

イバのステートメントハンドルで、ドライバマネージャステートメントハンドルにより判断されます。これは、入力時にアプリケーションから *rgbInfoValue* へ渡されます。この場合、*rgbInfoValue* が入力・出力両方の引数となることに注意してください。*rgbInfoValue* に渡される入力用 *hstmt* は、引数 *hdbc* に割り当てた *hstmt* でなくてはいけません。この情報タイプは、ドライバマネージャでしか実現されません。

- SQL_DRIVER_NAME (ODBC 1.0)
  文字列。データソースへアクセスする際に使用するドライバ名を示します。
- SQL_DRIVER_ODBC_VER (ODBC 2.0)
  文字列。ドライバでサポートされる ODBC バージョンを示します。このバージョンは ##.## という形式で示され、最初の 2 桁はメジャーバージョン、後の 2 桁がマイナーバージョンを表わします。SQL_SPEC_MAJOR と SQL_SPEC_MINOR を使い、メジャーバージョン、マイナーバージョンを定義します。このマニュアルで説明する ODBC バージョンの場合は 2 と 0 であり、ドライバから " 02.00 " が返されます。ドライバで SQLGetInfo がサポートされていても、引数 *fInfoType* のこの値がサポートされていない場合には、ドライバマネージャから " 01.00 " が返されます。
- SQL_DRIVER_VER (ODBC 1.0)
  文字列。ドライバのバージョンと、オプションとしてドライバの説明を示します。少なくとも、##.##.#### という形式でバージョンが示されます。最初の 2 桁はメジャーバージョン、次の 2 桁がマイナーバージョン、最後の 4 桁がリリースバージョンを表わします。
- SQL_EXPRESSIONS_IN_ORDERBY (ODBC 1.0)
  文字列。データソースで " ORDER BY " リスト内の式がサポートされる場合には " Y "、サポートされない場合には " N " になります。
- SQL_FETCH_DIRECTION (ODBC 1.0)
  この情報タイプは ODBC 1.0 で導入されました。各ビットマスクには、それが導入されたバージョンが付属します。32 ビットのビットマスク。サポートされる *fetch direction オプションを一覧します。サポートされるオプションを判断するために、次のビットマスクがフラグとともに使用されます。

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)
  16 ビットの整数値。単一層ドライバがデータソースのファイルをダイレクトに取り扱う方法を示します。
  SQL_FILE_NOT_SUPPORTED = ドライバは単一層ドライバではない。たとえば、ORACLE のドライバは二層構造のドライバである。
  SQL_FILE_TABLE = 単一層ドライバはデータソースのファイルをテーブルと

して取り扱う。たとえば、Xbase ドライバは Xbase ファイルをテーブルとして扱う。
SQL_FILE_QUALIFIER = 単一層ドライバは、データソースのファイルを修飾子として取り扱う。たとえば、Microsoft Access ドライバは Microsoft Access の各ファイルを完全なデータベースとして扱う。アプリケーションでは、ユーザがデータを選択する方法を決定するためにこれを使用することができる。たとえば、Xbase のユーザにとって、データはファイルに保存されているが、ORACLE や Microsoft Access ユーザにとって、データはテーブルに保存されている。ユーザが Xbase データソースを選択した場合に、アプリケーションから Windows の「ファイルを開く」ダイアログボックスを表示することができる。また、ユーザが Microsoft Access や ORACLE データソースを選択した場合には、アプリケーションで独自の「Select Table」ダイアログボックスを表示することができる。

- SQL_GETDATA_EXTENSIONS (ODBC 2.0)
  32 ビットのビットマスク。SQLGetData に関するエクステンションを一覧します。SQLGetData に関し、ドライバでサポートされる一般的なエクステンションを判断するために、次のビットマスクがフラグとともに使用されます。
  SQL_GD_ANY_COLUMN = バインドしていないカラムに対し、SQLGetData をコールすることができる。これには最後にバインドしたカラムより前のカラムも含まれる。SQL_GD_ANY_ORDER が同時に返された場合を除き、カラム番号の昇順にカラムをコールしなくてはならない点に注意する。
  SQL_GD_ANY_ORDER = カラム順を問わず、バインドしていないカラムに対し、SQLGetData をコールすることができる。SQL_GD_ANY_COLUMN が同時に返された場合を除き、最後にバインドしたカラムより後のカラムを呼び出さなくてはならない点に注意する。
  SQL_GD_BLOCK = バインドしていないカラムに対し、SQLGetData をコールすることができる。SQLSetPos により、ローの位置を指定した後であれば、データブロック（複数のロー）のどのローにあるカラムでもよい。
  SQL_GD_BOUND = バインドしていないカラムだけではなく、バインドしたカラムに対しても SQLGetData をコールすることができる。
  SQL_GD_ANY_COLUMN が同時に返された場合にのみ、この値がドライバから返される。
  SQLGetData は、バインドしていないカラムのデータを返すためにだけ使用されます。このカラムは最後にバインドしたカラムより後に現れ、カラム番号の昇順にコールされます。また、ローブロック内のローには属しません。

- SQL_GROUP_BY (ODBC 2.0)
  16 ビットの整数値。選択リスト内の GROUP BY 句のカラムと非集合カラムとの関係を示します。
  SQL_GB_NOT_SUPPORTED = GROUP BY 句はサポートされていない。
  SQL_GB_GROUP_BY_EQUALS_SELECT = 選択リストにおいて、GROUP BY 句に含まれるのはすべて非集合カラムでなくてはいけない。その他のカラムは含まれてはならない。（例：SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT）
  SQL_GB_GROUP_BY_CONTAINS_SELECT = 選択リストにおいて、GROUP BY 句に含まれるのはすべて非集合カラムでなくてはいけない。選択リストに属さないカラムを含めてもよい。（例：SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT, AGE）
  SQL_GB_NO_RELATION = GROUP BY 句のカラムと選択リストのカラムには

関連性がない。選択リストにおいて、グループ化されていないカラムと非集合カラムの意味はデータソースによって変わる。（例：SELECT DEPT, SALARY FROM EMPLOYEE GROUP BY DEPT, AGE）

- SQL_IDENTIFIER_CASE (ODBC 1.0)
  16 ビットの整数値。示される値は次の通りです。
  SQL_IC_UPPER = SQL の識別子は大文字と小文字の区別を行わず、システムカタログには大文字で保管される。
  SQL_IC_LOWER = SQL の識別子は大文字と小文字の区別を行わず、システムカタログには小文字で保管される。
  SQL_IC_SENSITIVE = SQL の識別子は大文字と小文字の区別を行い、システムカタログには大文字、小文字混在で保管される。
  SQL_IC_MIXED = SQL の識別子は大文字と小文字の区別を行わないが、システムカタログには大文字、小文字混在で保管される。
- SQL_IDENTIFIER_QUOTE_CHAR (ODBC 1.0)
  文字列。SQL 文において、引用された（括られた）識別子の区切り文字の始まりと終りとして使用されます（ODBC コマンドに対する引数として渡された識別子は括る必要がない）。データソースで引用された識別子がサポートされていない場合、ブランクが返されます。
- SQL_KEYWORDS (ODBC 2.0)
  文字列。データソース特定の全キーワードをカンマ区切りで一覧します。ODBC 特定のキーワードや、データソースと ODBC の両方で使用されているキーワードはこのリストに含まれません。ODBC のキーワードについての詳細は、付録 C の「予約キーワード表」の SQL 構文規約の項を参照してください。SQL_ODBC_KEYWORDS の#define値にはカンマ区切りでODBCキーワードの一覧が格納されます。
- SQL_LIKE_ESCAPE_CLAUSE (ODBC 2.0)
  文字列。データソースにおいて、LIKE キーワードのパーセント（%）記号やアンダースコア（_）記号の代わりに、エスケープ記号を使用可能な場合、または、ドライバにおいて、LIKE キーワードのエスケープ記号を定義している ODBC 構文がサポートされている場合には “ Y ”、その他の場合には “ N ” となります。
- SQL_LOCK_TYPES (ODBC 2.0)
  32 ビットのビットマスク。SQLSetPos の引数 fLock に指定できるロックタイプを一覧します。サポートされるロックタイプを判断するために、次のビットマスクがフラグとともに使用されます。

SQL_LCK_NO_CHANGE
SQL_LCK_EXCLUSIVE
SQL_LCK_UNLOCK

- SQL_MAX_BINARY_LITERAL_LEN (ODBC 2.0)
  32 ビットの整数値。SQL 文におけるバイナリ・リテラルの最大長を指定します（ヘキサデシマル文字の数。SQLGetTypeInfo から返された接頭辞と接尾字を除く）。たとえば、“ 0xFFAA ” というバイナリ・リテラルの長さは 4 です。最大長が特定されていない場合や、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)
  32 ビットの整数値。SQL 文におけるキャラクタ・リテラルの最大長を指定し

ます（文字数。SQLGetTypeInfo から返された接頭辞と接尾字をのぞく）。最大長が特定されていない場合や、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)
  16 ビットの整数値。データソースにおけるカラム名の最大長を指定します。最大長が特定されていない場合や、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)
  16 ビットの整数値。GROUP BY 句で使用できる最大カラム数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。インデックスで使用できる最大カラム数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。ORDER BY 句で使用できる最大カラム数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。選択リストで使用できる最大カラム数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。テーブルで使用できる最大カラム数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。データソースで使用できるカーソル名の最大長を指定します。最大長が特定されていない場合や、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)
  32 ビットの整数値。インデックスの連結フィールドで使用できる最大バイト数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。データソースの所有者名の最大長を指定します。最大長が特定されていない場合や、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)
  16 ビットの整数値。データソースで使用できるプロシージャ名の最大長を指定します。最大長が特定されていない場合や、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)
  16 ビットの整数値。データソースで使用できる修飾子名の最大長を指定します。最大長が特定されていない場合や、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)
  32 ビットの整数値。テーブル中のひとつのローの最大長を指定します。特に制限がない場合や、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。SQL_MAX_ROW_SIZE 情報タイプに返されたローの最大サイズに、そのローの SQL_LONGVARCHAR カラムや SQL_LONGVARBINARY カラムの長さがすべて含まれている場合には “ Y ”、それ以外の場合は “ N ” になります。
- SQL_MAX_STATEMENT_LEN (ODBC 2.0)
  32 ビットの整数値。SQL 文の最大長を指定します（バイト数。スペースを含む）。最大長が特定されていない場合や、わからない場合、この値は 0 に設定されます。
- SQL_MAX_TABLE_NAME_LEN (ODBC 1.0)
  16 ビットの整数値。データソースのテーブル名の最大長を指定します。最大長が特定されていない場合や、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)
  16 ビットの整数値。SELECT 文の FROM 句において使用できるテーブルの最大数を指定します。特に制限がない場合や、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)
  16 ビットの整数値。データソースのユーザ名の最大長を指定します。最大長が特定されていない場合や、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。データソースで複数の結果セットがサポートされている場合には “ Y ”、それ以外の場合は “ N ” になります。
- SQL_MULTIPLE_ACTIVE_TXN (ODBC 1.0)
  文字列。複数の接続においてアクティブトランザクションを使用可能な場合には “ Y ”、一度にひとつの接続においてのみアクティブトランザクションを使用できる場合は “ N ” となります。
- SQL_NEED_LONG_DATA_LEN (ODBC 2.0)
  文字列。データソースに値を送る前に、長いデータ値の長さ（SQL_LONGVARCHAR や SQL_LONGVARBINARY データタイプ、または、データソース特定のデータタイプ）をデータソース側に知らせる必要がある場合には “ Y ”、それ以外の場合は “ N ” となります。詳細については、SQLBindParameter と SQLSetPos の項を参照してください。
- SQL_NON_NULLABLE_COLUMNS (ODBC 1.0)
  16 ビットの整数値。データソースでノンヌルラベルカラムがサポートされているかどうかを特定します。
  SQL_NNC_NULL = すべてのカラムがヌル指定可能でなくてはいけない。
  SQL_NNC_NON_NULL = カラムにヌル指定できる可能性がある（CREATE TABLE 文で NOT NULL カラム制限がサポートされているデータソース）

- SQL_NULL_COLLATION (ODBC 2.0)
  16 ビットの整数値。ソート後のリストにおける NULL の位置を指定します。
  SQL_NC_END = ソート後、ソート順に関係なく NULL はリストの最後に置かれる。
  SQL_NC_HIGH = ソート後、NULL はリストのハイエンドに置かれる。
  SQL_NC_LOW = ソート後、NULL はリストのローエンドに置かれる。
  SQL_NC_START = ソート後、ソート順に関係なく NULL はリストの最初に置かれる。

- SQL_NUMERIC_FUNCTIONS (ODBC 1.0)
  ODBC 1.0 において、この情報タイプが導入されました。各ビットマスクには、それが導入された ODBC バージョンが付属します。
  32 ビットのビットマスク。データソースに関連するドライバにおいて、サポートされるスカラー数値関数を一覧します。サポートされる数値関数を判断するために、次のビットマスクが使用されます。

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)
  16ビットの整数値。ODBC準拠レベルを示します。
  SQL_OAC_NONE = な
  SQL_OAC_LEVEL1 = レベル1サポート
  SQL_OAC_LEVEL2 = レベル1サポート

- SQL_ODBC_SAG_CLI_CONFORMANCE (ODBC 1.0)
  16 ビットの整数値。SAG 仕様の関数に対する準拠状況を示します。
  SQL_OSCC_NOT_COMPLIANT = SAG 非準拠。1 つまたは複数の関数がサポートされない。
  SQL_OSCC_COMPLIANT = SAG 準拠。
- SQL_ODBC_SQL_CONFORMANCE (ODBC 1.0)
  16 ビットの整数値。ドライバでサポートされる SQL 文法を示します。
  SQL_OSC_MINIMUM = 最低限の文法をサポート
  SQL_OSC_CORE = コアとなる文法をサポート
  SQL_OSC_EXTENDED = 拡張文法をサポート
- SQL_ODBC_SQL_OPT_IEF (ODBC 1.0)
  文字列。データソースでオプションの Integrity Enhancement Facility がサポートされる場合には " Y "、サポートされない場合は " N " になります。
- SQL_ODBC_VER (ODBC 1.0)
  文字列。ドライバマネージャが準拠している ODBC のバージョンを示します。このバージョンは、" ##.## " という形式で示され、最初の 2 桁はメジャーバージョンを、後の 2 桁はマイナーバージョンを示します。この値はドライバマネージャでのみ有効です。
- SQL_ORDER_BY_COLUMNS_IN_SELECT (ODBC 2.0)
  文字列。ORDER BY 句のカラムが必ず選択リスト中に存在する必要がある場合には " Y "、それ以外の場合は " N " になります。
- SQL_OUTER_JOINS (ODBC 1.0)
  ODBC 1.0 において、この情報タイプが導入されました。各返り値の説明の後に、導入された ODBC バージョンが記載されています。
  文字列。" N " = 不可。データソースで外部結合はサポートされない。(ODBC 1.0)
  " Y " = 可。データソースでは、2 つのテーブル間の外部結合がサポートされ、ネスティングした外部結合を除き、ドライバでは ODBC の外部結合構文がサポートされる。しかし、ON 句において比較演算子の左側のカラムは、外部結合における左側のテーブルのカラムでなくてはならない。また、比較演算子の右側のカラムは、外部結合における右側のテーブルのカラムでなくてはならない。(ODBC 1.0)
  " P " = 一部分のみ可。データソースでは、外部結合が部分的にサポートされ、ドライバでは ODBC の外部結合構文がサポートされる。しかし、ON 句において比較演算子の左側のカラムは、外部結合における左側のテーブルのカラムでなくてはならない。また、比較演算子の右側のカラムは、外部結合における右側のテーブルのカラムでなくてはならない。さらに、外部結合における右側のテーブルを自己結合に含むことはできない。(ODBC 2.0)
  " F " = すべて可。データソースでは、ネスティングした外部結合が完全にサポートされ、ドライバでは ODBC の外部結合構文がサポートされる。(ODBC 2.0)
- SQL_OWNER_TERM (ODBC 1.0)
  文字列。所有者に対するデータソースベンダ名を示します。
  例 :" owner "、" Authorization ID "、" Schema "

- SQL_OWNER_USAGE (ODBC 2.0)
  32 ビットのビットマスク。所有者が使用できる文を一覧します。
  SQL_OU_DML_STATEMENTS = 所有者はすべての Data Manipulation Language 文を使用できる。つまり、SELECT、INSERT、UPDATE, DELETE、SELECT FOR UPDATE（サポートされている場合）、位置指定した更新や削除文を使用可能。
  SQL_OU_PROCEDURE_INVOCATION = 所有者は ODBC のプロシージャ呼び出し文を使用できる。
  SQL_OU_TABLE_DEFINITION = 所有者はすべてのテーブル定義文を使用できる。つまり、CREATE TABLE、CREATE VIEW、ALTER TABLE, DROP TABLE、DROP VIEW 文を使用可能。
  SQL_OU_INDEX_DEFINITION = 所有者はすべてのインデックス定義文を使用できる。つまり、CREATE INDEX、DROP INDEX 文を使用可能。
  SQL_OU_PRIVILEGE_DEFINITION = 所有者はすべての権限定義文を使用できる。つまり、GRANT、REVOKE 文を使用可能。
- SQL_POS_OPERATIONS (ODBC 2.0)
  32 ビットのビットマスク。SQLSetPos でサポートされる操作を一覧します。サポートされる操作を判断するために、次のビットマスクがフラグとともに使用されます。

SQL_POS_POSITION
SQL_POS_REFRESH
SQL_POS_UPDATE
SQL_POS_DELETE
SQL_POS_ADD

- SQL_POSITIONED_STATEMENTS (ODBC 2.0)
  32 ビットのビットマスク。位置指定した SQL 文でサポートされる操作を一覧します。サポートされる SQL 文を判断するために、次のビットマスクがフラグとともに使用されます。

SQL_PS_POSITIONED_DELETE
SQL_PS_POSITIONED_UPDATE
SQL_PS_SELECT_FOR_UPDATE

- SQL_PROCEDURE_TERM (ODBC 1.0)
  文字列。プロシージャに対するデータソースベンダ名を示します。
  例：“ database procedure ”、“ stored procedure ”、“ procedure ”
- SQL_PROCEDURES (ODBC 1.0)
  文字列。データソースでプロシージャがサポートされ、ドライバでは ODBC プロシージャ呼び出し構文がサポートされる場合には “ Y ”、それ以外の場合は “ N ” になります。
- SQL_QUALIFIER_LOCATION (ODBC 2.0)
  16 ビットの整数値。修飾されたテーブル名の修飾子の位置を示します。

SQL_QL_START
SQL_QL_END

- たとえば、Xbase ドライバからは SQL_QL_START が返されます。これは、\EMP-DATA\ EMP.DBF のように、ディレクトリ（修飾）名がテーブル名の最初にあるためです。また、ORACLE Server ドライバからは SQL_QL_END が返されます。これは、ADMIN.EMP@EMPDATA のように、修飾名がテーブル名の最後にあるためです。
- SQL_QUALIFIER_NAME_SEPARATOR (ODBC 1.0)
  文字列。データソースで区切り文字として定義された文字を示し、修飾名およびそれに続く修飾要素を区切るために使用されます。
- SQL_QUALIFIER_TERM (ODBC 1.0)
  文字列。修飾子に対するデータソースベンダ名を示します。
  例：“ database ”、“ directory ”
- SQL_QUALIFIER_USAGE (ODBC 2.0)
  32 ビットのビットマスク。修飾子が使用できる文を一覧します。使用できる修飾子を判断するために、次のビットマスクが使用されます。
  SQL_QU_DML_STATEMENTS = 修飾子はすべての Data Manipulation Language 文で使用できる。つまり、SELECT、INSERT、UPDATE, DELETE、SELECT FOR UPDATE（サポートされている場合）、位置指定した更新や削除文で使用可能。
  SQL_QU_PROCEDURE_INVOCATION = 修飾子は ODBC のプロシージャ呼び出し文で使用できる。
  SQL_QU_TABLE_DEFINITION = 修飾子はすべてのテーブル定義文で使用できる。つまり、CREATE TABLE、CREATE VIEW、ALTER TABLE, DROP TABLE、DROP VIEW 文で使用可能。
  SQL_QU_INDEX_DEFINITION = 修飾子はすべてのインデックス定義文で使用できる。つまり、CREATE INDEX、DROP INDEX 文で使用可能。
  SQL_QU_PRIVILEGE_DEFINITION = 修飾子はすべての権限定義文で使用できる。つまり、GRANT、REVOKE 文で使用可能。
- SQL_QUOTED_IDENTIFIER_CASE (ODBC 2.0)
  16 ビットの整数値。この値は次の通りです。
  SQL_IC_UPPER = SQL の引用識別子は大文字と小文字の区別を行わず、システムカタログには大文字で保管される。
  SQL_IC_LOWER = SQL の引用識別子は大文字と小文字の区別を行わず、システムカタログには小文字で保管される。
  SQL_IC_SENSITIVE = SQL の引用識別子は大文字と小文字の区別を行い、システムカタログには大文字、小文字混在で保管される。
  SQL_IC_MIXED = SQL の引用識別子は大文字と小文字の区別を行わないが、システムカタログには大文字、小文字混在で保管される。
- SQL_ROW_UPDATES (ODBC 1.0)
  文字列。キーセット駆動型のカーソルや混合カーソルに、取り込んだ全ローのバージョンや値が保持され、そのローを最後に取り込んだ後にユーザが行った変更を追跡することができない場合には “ Y ”、それ以外の場合は “ N ” となります。

- SQL_SCROLL_CONCURRENCY (ODBC 1.0)
  32 ビットのビットマスク。スクローラブルカーソルでサポートされる同時制御オプションを一覧します。サポートされるオプションを判断するために、次のビットマスクが使用されます。
  SQL_SCCO_READ_ONLY = カーソルはリードオンリー。更新不可。
  SQL_SCCO_LOCK = カーソルでは最低限のロックレベルが使用され、ローへの更新が保証される。
  SQL_SCCO_OPT_ROWVER = カーソルでは、ローバージョンと比べると緩やかな同時制御が使用される。これには、SQLBaseAE の ROWID や Sybase の TIMESTAMP などがある。
  SQL_SCCO_OPT_VALUES = カーソルでは、値に比べると緩やかな同時制御が使用される。
- SQL_SCROLL_OPTIONS (ODBC 1.0)
  ODBC 1.0 において、この情報タイプが導入されました。ビットマスクにはそれぞれ、導入された ODBC バージョンが付属します。
  32 ビットのビットマスク。スクローラブルカーソルでサポートされるスクロールオプションを一覧します。サポートされるオプションを判断するために、次のビットマスクが使用されます。
  SQL_SO_FORWARD_ONLY = カーソルは前方向へのスクロールのみ。(ODBC 1.0)
  SQL_SO_STATIC = 結果セットのデータはスタティック。(ODBC 2.0)
  SQL_SO_KEYSET_DRIVEN = ドライバでは、結果セット内の各ローのキーが保存、使用される。(ODBC 1.0)
  SQL_SO_DYNAMIC = ドライバでは、結果セットにおける各ローのキーが保持される（キーセットサイズはローセットサイズと同じ）。(ODBC 1.0)
  SQL_SO_MIXED = ドライバでは、結果セットにおける各ローのキーが保持される。また、キーセットサイズはローセットサイズよりも大きい。キーセット内でカーソルはキーセット駆動型であり、キーセット外ではダイナミックである。(ODBC 1.0)

- SQL_SEARCH_PATTERN_ESCAPE (ODBC 1.0)
  文字列。ドライバでエスケープ記号として使用する文字を指定します。この文字により、アンダースコア記号（_）やパーセント記号（%）などのメタ・キャラクタが、検索パターンの有効文字として使用できるようになります。このエスケープ記号は、カタログコマンドのうち、検索文字列を指定できる引数にのみ有効です。この文字列が空であれば検索パターンのエスケープ記号はサポートされません。この *fInfoType* はカタログコマンドだけを対象とします。検索パターンのエスケープ記号の使用方法は、この章の前半にある「検索パターン引数」を参照してください。
- SQL_SERVER_NAME (ODBC 1.0)
  文字列。実際に使用されているデータソース特定のサーバ名を示します。SQLConnect や SQLDriverConnect、SQLBrowseConnect において、データソース名が必要な場合に便利です。
- SQL_SPECIAL_CHARACTERS (ODBC 2.0)
  文字列。すべての特種文字が含まれます（つまり、a から z、0 から 9、アンダースコアを除くすべての文字）。データソース上で、テーブル名やカラム名、インデックス名などのオブジェクト名に使用されます（例：“ #$^ ”）。

- SQL_STATIC_SENSITIVITY (ODBC 2.0)
  32 ビットのビットマスク。アプリケーションから SQLSetPos や位置指定した更新や削除文を使用し、スタティックまたはキーセット駆動型のカーソルに変更を加えた場合、それをアプリケーションで認識できるかどうかを示します。
  SQL_SS_ADDITIONS = カーソルは追加されたローを認識できる。カーソルはこれらのローをスクロールすることができる。これらのローがカーソルに追加されるかどうかはドライバに依存する。
  SQL_SS_DELETIONS = カーソルでは、削除されたローを使用できず、結果セットに“穴”が空くこともない。削除されたローからカーソルをスクロールした後に、そのローへ戻ることはできない。
  SQL_SS_UPDATES = カーソルは更新されたローを認識できる。カーソルをスクロールして、更新されたローへ戻った場合、カーソルからは更新されたローが返され、元のデータではなくなっている。キーセット駆動型のカーソルでは、キーの値を更新すると、既存のローの削除、または新規ローの追加を行うものと見なされるため、常にこの値が返される。他のユーザ（同一アプリケーションの別カーソルを含む）が結果セットに変更を加えた場合に、それをアプリケーションから認識できるかどうかは、カーソルタイプに依存する。

- SQL_STRING_FUNCTIONS (ODBC 1.0)
  ODBC 1.0 において、この情報タイプが導入されました。ビットマスクにはそれぞれ、導入された ODBC バージョンが付属します。
  32 ビットのビットマスク。ドライバおよび関連データソースでサポートされるスカラー文字列関数を一覧します。サポートされる文字列関数を判断するために、次のビットマスクが使用されます。

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

アプリケーションで引数 string_exp1、string_exp2、start とともに LOCATE スカラー関数をコールできる場合、ドライバからはビットマスク SQL_FN_STR_LOCATE が返されます。しかし、アプリケーションで LOCATE スカラー関数をコールする際に、引数 string_exp1、string_exp2 しか指定できない場

合、ドライバからはビットマスク SQL_FN_STR_LOCATE_2 が返されます。ドライバで LOCATE スカラー関数が完全にサポートされる場合には、両方のビットマスクが返されます。

- SQL_SUBQUERIES (ODBC 2.0)
  32 ビットのビットマスク。副問合わせをサポートする属性を一覧します。

SQL_SQ_CORRELATED_SUBQUERIES
SQL_SQ_COMPARISON
SQL_SQ_EXISTSSQL_SQ_IN
SQL_SQ_QUANTIFIED

- The SQL_SQ_CORRELATED_SUBQUERIES
  ビットマスク。相関副問合わせをサポートする属性を一覧します。
- SQL_SYSTEM_FUNCTIONS (ODBC 1.0)
  32 ビットのビットマスク。ドライバおよび関連データソースでサポートされるスカラーシステム関数を一覧します。サポートされるシステム関数を判断するために、次のビットマスクが使用されます。

SQL_FN_SYS_DBNAME
SQL_FN_SYS_IFNULL
SQL_FN_SYS_USERNAME

- SQL_TABLE_TERM (ODBC 1.0)
  文字列。テーブルに対するデータソースベンダ名を示します。
  例 :“ table ”、“ file ”
- SQL_TIMEDATE_ADD_INTERVALS (ODBC 2.0)
  32 ビットのビットマスク。スカラー関数 TIMESTAMPADD のために、ドライバおよび関連データソースでサポートされるタイムスタンプの間隔を一覧します。サポートされる間隔を判断するために、次のビットマスクが使用されます。

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)
  32 ビットのビットマスク。スカラー関数 TIMESTAMPDIFF のために、ドライバおよび関連データソースでサポートされるタイムスタンプの間隔を一覧しま

す。サポートされる間隔を判断するために、次のビットマスクが使用されます。

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)
  ODBC 1.0 において、この情報タイプが導入されました。ビットマスクにはそれぞれ、導入された ODBC バージョンが付属します。
  32 ビットのビットマスク。ドライバおよび関連データソースでサポートされる日付と時間のスカラー関数を一覧します。サポートされる日付と時間の関数を判断するために、次のビットマスクが使用されます。

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)
  ODBC 1.0 において、この情報タイプが導入されました。返り値にはそれぞれ、導入された ODBC バージョンが付属します。
  16 ビットの整数値。ドライバおよびデータソースでサポートされるトランザクションを一覧します。
  SQL_TC_NONE = トランザクションはサポートされない。(ODBC 1.0)
  SQL_TC_DML = トランザクションでは Data Manipulation Language（DML）文だけ（SELECT、INSERT、UPDATE、DELETE）が使用できる。トランザクションで Data Definition Language（DDL）文を使用するとエラーになる。

(ODBC 1.0)
SQL_TC_DDL_COMMIT = トランザクションでは DML 文だけが使用できる。トランザクションで DDL 文（CREATE TABLE、DROP INDEX 等）を使用すると、トランザクションがコミットされる。(ODBC 2.0)
SQL_TC_DDL_IGNORE = トランザクションでは DML 文だけが使用できる。トランザクションで DDL 文を使用しても無視される。(ODBC 2.0)
SQL_TC_ALL = トランザクションでは DML 文と DDL 文を自由な順序で使用できる。(ODBC 1.0)

- SQL_TXN_ISOLATION_OPTION(ODBC 1.0)
  32 ビットのビットマスク。ドライバおよびデータソースでサポートされるトランザクションのアイソレーションレベルを一覧します。このアイソレーションレベルの説明は、SQL_DEFAULT_TXN_ISOLATION の項を参照してください。サポートされるオプションを判断するために、次のビットマスクがフラグとともに使用されます。

SQL_TXN_READ_UNCOMMITTED
SQL_TXN_READ_COMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE
SQL_TXN_VERSIONING

- SQL_UNION (ODBC 2.0)
  32 ビットのビットマスク。UNION 句がサポートされるかどうかを示します。
  SQL_U_UNION = データソースで UNION 句がサポートされる。
  SQL_U_UNION_ALL = データソースでUNION句のキーワードALLがサポートされる。

  （この場合、SQLGetInfo から SQL_U_UNION と SQL_U_UNION_ALL の両方が返されます）。
- SQL_USER_NAME (ODBC 1.0)
  文字列。特定のデータベースで使用する名前を示します。これは、ログイン名と同じ名前でなくてもかまいません。

コード例

SQLGetInfo からは、サポートされるオプションが *rgbInfoValue* に格納されて 32 ビットのビットマスク形式で返されます。サポートされるオプションを判断するために、各オプションのビットマスクがフラグとともに使用されます。

▼ 例題：hdbc に関連するドライバで SUBSTRING スカラー関数がサポートされるかどうかを判断するには、アプリケーションで次のコードを使用します。

```
UDWORD fFuncs;
SQLGetInfo(hdbc, SQL_STRING_FUNCTIONS, (PTR)&fFuncs,
sizeof(fFuncs), NULL);
if (fFuncs & SQL_FN_STR_SUBSTRING) /* SUBSTRING supported */...;else
/* SUBSTRING not supported */...;
```

関連する関数

情報の種類	参照
接続オプションの設定を返す	SQLGetConnectOption（拡張）
ドライバで関数がサポートされるか判断する	SQLGetFunctions（拡張）
文のオプション設定を返す	SQLGetStmtOption（拡張）
データソースのデータタイプ情報を返す	SQLGetTypeInfo（拡張）

索引

数字

C

D

E

F

G

H

N

O

S

W

え

お

か

く

け

テーマ別索引

ログインコマンド 37

コンテキストコマンド 47

ローレベルコマンド 61

アルファベット順索引