CHAPTER 1 フォームとレポート

CHAPTER 2 コントロール

CHAPTER 3 SQLとエラー処理

CHAPTER 4 レコードの取得と検索

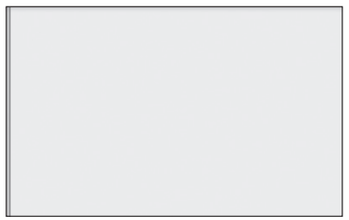
CHAPTER 5 データの編集

CHAPTER 6 文字列／数値／日付操作

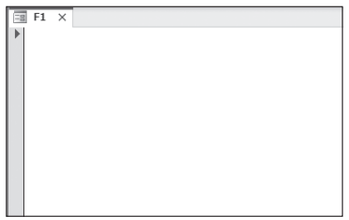
001 フォームを開く

➔ AccessはGUI（グラフィカルユーザインターフェース）機能が優れているので、フォームを活用すると使いやすいアプリケーションが実装できます。そんなフォームを開く、基本のコードです。

Before



After



サンプルコード（標準モジュール）

```
01 Sub sample()
02   DoCmd.OpenForm "F1" ←"F1"フォームを既定のビューで開く
03 End Sub
```

フォーム名とビューを指定する

フォームを開くにはDoCmd.OpenFormメソッドという命令のあとに、半角スペースで区切ってから、対象のフォーム名を記述します。フォーム名は文字列で指定するため、「"」（ダブルクォーテーション）で囲みます。

フォーム名に続いて、カンマで区切ってビューの指定も可能です。サンプルコードのように省略すると「DoCmd.OpenForm "F1", acNormal」と同じ意味となり、既定のビュー、つまりフォームビューで開きます。「DoCmd.OpenForm "F1", acDesign」と書けばデザインビューで、「DoCmd.OpenForm "F1", acLayout」と書けばレイアウトビューで、それぞれ開きます。

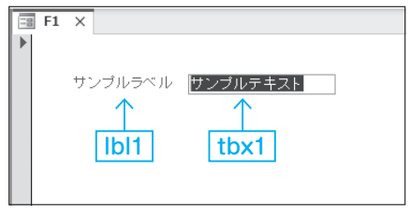
```
DoCmd.OpenForm "F1", acNormal
```

- DoCmd.OpenForm：フォームを開く命令
- "F1"：フォーム名
- 半角スペース
- acNormal：ビュー（acNormalの場合は省略可）

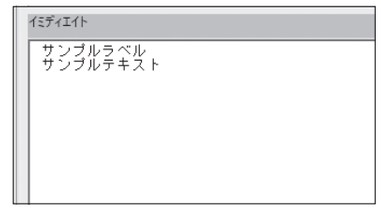
002 フォーム上の情報を参照する

➔ フォームビューで開いているフォームからは、さまざまな情報を読み取ることができます。たとえば、配置してあるラベルの標題（表示されているテキスト）や、テキストボックスの値などです。

Before



After



サンプルコード（標準モジュール）

```
01 Sub sample()
02   Debug.Print Form_F1.lbl1.Caption ←ラベルの標題を取得
03   Debug.Print Form_F1.txb1.Value ←テキストボックスの値を取得
04 End Sub
```

オブジェクト名.コントロール名.プロパティで取得できる

コントロールの名前は、プロパティシートの「その他」タブの「名前」から設定できます。フォームビューで表示されるテキストを取得するプロパティは、ラベルは「Caption」（標題）、テキストボックスやコンボボックスは「Value」（値）と、コントロールによって異なります。

Debug.Printは、VBEのイミディエイトウィンドウに結果を出力する命令で、現在のプロパティ値の確認などに便利です。イミディエイトウィンドウは、VBEのツールバーの、「表示」→「イミディエイトウィンドウ」から表示できます。

サンプルコードは、「F1」フォームがフォームビューで開いている状態で実行してください。

```
Debug.Print Form_F1.lbl1.Caption
```

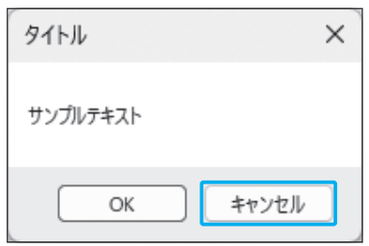
- Debug.Print：イミディエイトウィンドウへ出力
- Form_F1：オブジェクト名
- lbl1：オブジェクト名
- Caption：プロパティ
- Form_F1.lbl1.Caption：コントロール名

016 メッセージボックスに表示するボタンを指定する

➔ **メッセージボックスのボタンは既定値がvbOKOnly（「OK」ボタンのみ）ですが、ほかにも「キャンセル」「はい」「いいえ」など、さまざまなボタンがあります。**

Before

After

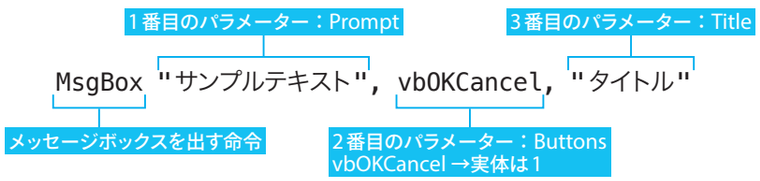


サンプルコード（標準モジュール）

```
01 Sub sample()
02   MsgBox "サンプルテキスト", vbOKCancel, "タイトル"
03 End Sub
```

2番目のパラメーターでボタンを指定できる

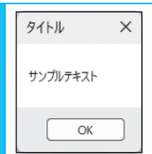
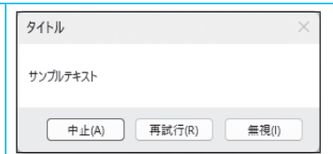
015と同じく、ボタン・アイコンの種類を設定できる、2番目のButtonsパラメーターを使います。既定値であるvbOKOnly（「OK」ボタンのみ）は省略できますが、それ以外のボタン配置のメッセージボックスを表示したい場合に、種類を指定します。

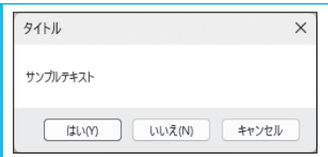
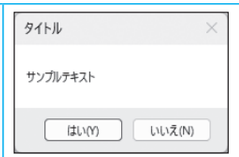
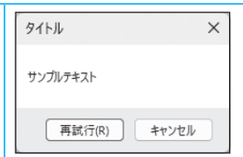


アイコンを同時に表示したい場合は、「MsgBox "サンプルテキスト", vbOKCancel + vbQuestion, "タイトル"」のように、アイコンの定数または値（P.33参照）を足し算の形で記述します。

ボタン配置の種類

メッセージボックスのボタンの種類は、以下のように6種類から選択できます。

イメージ			
定数	vbOKOnly	vbOKCancel	vbAbortRetryIgnore
値	0	1	2

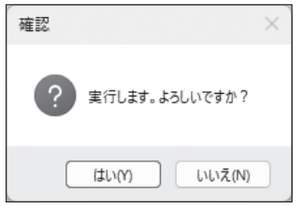
イメージ			
定数	vbYesNoCancel	vbYesNo	vbRetryCancel
値	3	4	5

「OK」「キャンセル」、「はい」「いいえ」などの判断に使えるボタンのほか、「中止」や「再試行」といったボタンも用意されています。

ONE POINT ボタン＋アイコンの合計値で判断される

「OK」「キャンセル」や「はい」「いいえ」のようなボタン配置のメッセージボックスを出す場合は、表示するテキストも疑問形になるはずです。そんな場合は「？」マークのアイコンも付いていると、ユーザーはわかりやすいですよね。

下図は「MsgBox "実行します。よろしいですか？", vbYesNo + vbQuestion, "確認"」というコードを実行した結果ですが、「MsgBox "実行します。よろしいですか？", 36, "確認"」と書かれているのと同じです。ボタンとアイコンの合計値はすべての組み合わせで異なるので、コンピューターは数値でかんたんに判断できるのです。



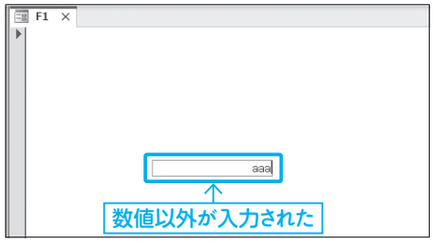
とはいえ、人間の目には数値の合計値では何を表現しているのかわかりにくいので、可読性の面から、コードの記述は定数をおすすめします。

本書のサンプルでは、記述を短くするためメッセージボックスのアイコンを省略して書いているものが多いですが、実務では積極的に使ってみてください。

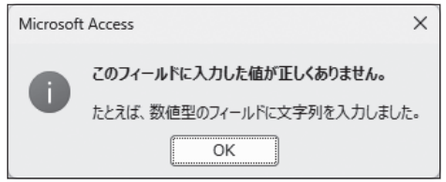
041 数値以外の入力を禁止する～テキストボックス

➔ **テキストボックスは、特に指定しなければ、どんな形式の値でも入力できます。用途によって数値や日付など、属性で制限すると入力ミスが少なくなります。**

Before



After

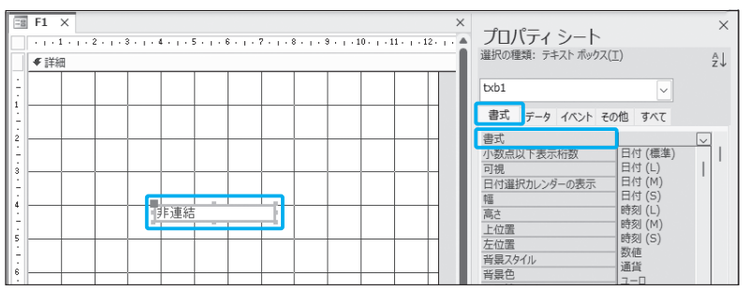


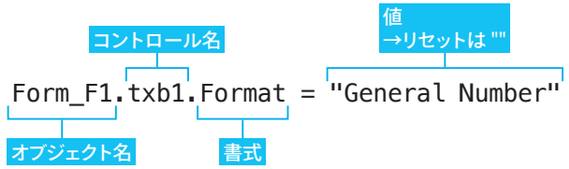
サンプルコード（標準モジュール）

```
01 Sub sample()
02    Form_F1.txb1.Format = "General Number" ←テキストボックスの書式を"数値"にする
03 End Sub
```

テキストボックスの「書式」プロパティを変更する

このコードは、プロパティシートで行うテキストボックスの「書式」の値の変更を、VBAで実行するものです。このサンプルは「F1」をフォームビューで開いた状態で実行してください。また、サンプル実行後、値を入力したら Enter キーを押してください。





テキストボックスのFormatプロパティを、変更したい書式にして実行します。書式を元のテキスト形式へ戻したい場合は、「""」と設定してリセットします。

テキストボックスに値が入った状態では、適応できない形式へ変更しようとするとエラーが発生するので注意してください。

「書式」プロパティの設定値

設定できる値は以下のようになっています。

内容	設定値
日付（標準）	General Date
日付（L）	Long Date
日付（M）	Medium Date
日付（S）	Short Date
時刻（L）	Long Time
時刻（M）	Medium Time
時刻（S）	Short Time

内容	設定値
数値	General Number
通貨	Currency
ユーロ	Euro
固定	Fixed
標準	Standard
パーセント	Percent
指数	Scientific

TrueとFalse、YesとNoはそのまま、"True/False"、"Yes/No"と記述します。

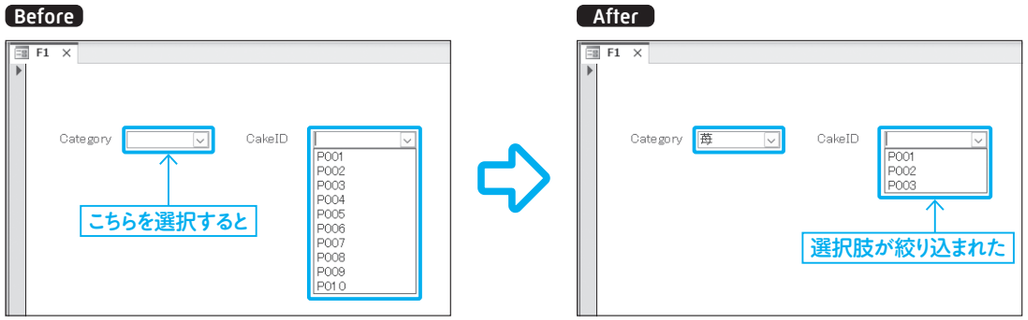
ONE POINT VBAだけでなくプロパティシートも活用しよう

VBAでの設定は一時的なもので、フォームを閉じると元に戻ってしまいます。用途があらかじめ決まっているテキストボックスはプロパティシートで設定したほうが、間違いがありません。

状況によって書式を変更したい場合に、VBAを活用するのがおすすめです。

056 連動して選択肢を絞り込む ～コンボボックス

➔ **コンボボックスの選択肢は、多すぎると選びにくくなってしまいます。1つ目のコンボボックスでカテゴリーを選択すると、連動して2つ目のコンボボックスの選択肢が絞り込まれるしくみを作ってみましょう。**



サンプルコード（Form_F1モジュール）

```
01 Private Sub cmb1_AfterUpdate() ←更新後処理
02   Me.cmb2.RowSource = "SELECT fCakeID FROM T1 WHERE fCategory='" & Me.cmb1.Value & "';"
03 End Sub
                                  ↑cmb2の選択肢を絞り込む
```

テーブルに「カテゴリー」となるフィールドを作成しておく

この動きは**055**と同じくコンボボックスの「更新後処理」（AfterUpdate）イベントプロシージャを使って、1つ目のコンボボックスの選択が変更されたら、2つ目のコンボボックスの選択肢を設定し直す、というしくみにします。

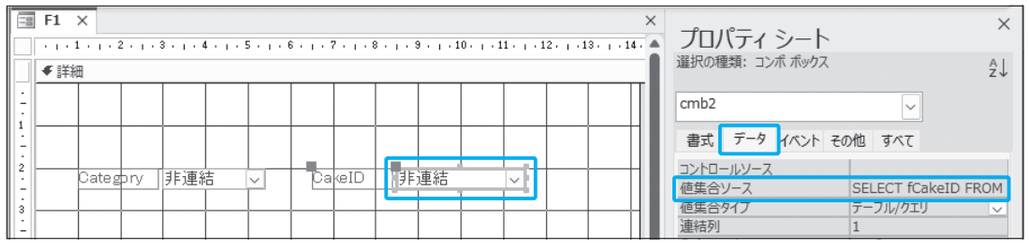
そのためには、テーブルにカテゴリーの情報が必要になります。このサンプルのT1テーブルは、図のように「fCategory」というフィールドで3つのカテゴリーに分かれています。

T1

fCakeID	fCakeName	fListPrice	fCostPrice	fCategory
P001	苺ショート ピース	¥350	¥150	苺
P002	苺ショート 5号	¥2,500	¥900	苺
P003	苺ショート 6号	¥3,200	¥1,000	苺
P004	チョコレート ピース	¥320	¥130	チョコレート
P005	チョコレート 4号	¥1,600	¥600	チョコレート
P006	チョコレート 5号	¥2,000	¥700	チョコレート
P007	プレーンマフィン	¥250	¥90	マフィン
P008	チョコレートマフィン	¥250	¥100	マフィン
P009	ブルーベリーマフィン	¥250	¥120	マフィン
P010	抹茶マフィン	¥250	¥120	マフィン

フォームが表示されたときの選択肢の設定を行っておく

また、このサンプルの「F1」フォーム上にあるcmb2は、フォームが表示された時点ではT1テーブルのすべてのレコードのfCakeIDフィールドが選択肢として設定されていてほしいので、プロパティシートの「データ」タブの「値集合ソース」を「SELECT fCakeID FROM T1;」としておきます。



cmb1は、カテゴリーであるfCategoryフィールドが選択肢として設定されていてほしいので、プロパティシートの「データ」タブの「値集合ソース」を「SELECT fCategory FROM T1 GROUP BY fCategory;」としておきます。「GROUP BY フィールド名」と記述すると、重複した値を1つにまとめて表示してくれます。

SQLのSELECT構文に条件を設定する

ここまで、SQLのSELECT構文を使ってテーブルの情報を取り出すために「SELECT フィールド名 FROM テーブル名;」と書いてきました。これは、テーブルのすべてのレコードが対象です。

レコードを絞り込むには、SELECT フィールド名 FROM テーブル名 **WHERE 条件**;」と付け足します。

```
"SELECT fCakeID FROM T1 WHERE fCategory='" & Me.cmb1.Value & "';"
```

- `fCakeID`：取り出したいフィールド名
- `T1`：対象のテーブル名
- `fCategory='" & Me.cmb1.Value & "'`：取り出すレコードの条件

SQLではレコードに条件を付ける部分は**WHERE句**と呼ばれ、**038**や**055**で紹介したDLookup関数の条件を記述する部分にも使われています。

063 データベースを操作する

➔ **VBAでテーブルのデータを操作するには、SQLを組み込むのが一番の近道です。ここでは、もっともシンプルなコードでテーブルのデータを変更する書き方を紹介します。**

Before

T1

fCakeID	fCakeName	fListPrice	fCostPrice
P001	苺ショート ピース	¥350	¥150
P002	苺ショート 5号	¥2,500	¥900
P003	苺ショート 6号	¥3,200	¥1,000
P004	チョコレート ピース	¥320	¥130
P005	チョコレート 4号	¥1,600	¥600
P006	チョコレート 5号	¥2,000	¥700
P007	プレーンマフィン	¥250	¥90
P008	チョコレートマフィン	¥250	¥100
P009	ブルーベリーマフィン	¥250	¥120
P010	抹茶マフィン	¥250	¥120

After

T1

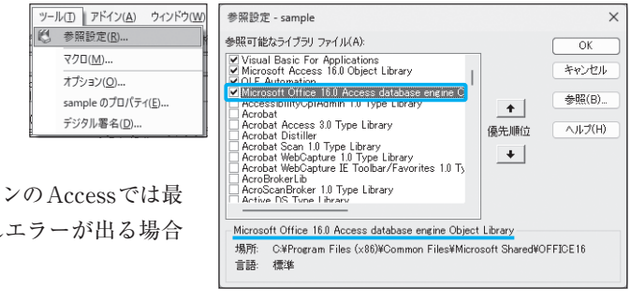
fCakeID	fCakeName	fListPrice	fCostPrice
P001	苺ショート ピース	¥350	¥150
P002	苺ショート 5号	¥2,500	¥900
P003	苺ショート 6号	¥3,200	¥1,000
P004	チョコレート ピース	¥320	¥130
P005	チョコレート 4号	¥1,600	¥600
P006	チョコレート 5号	¥2,000	¥700
P007	プレーンマフィン	¥250	¥90
P008	チョコレートマフィン	¥300	¥100
P009	ブルーベリーマフィン	¥250	¥120
P010	抹茶マフィン	¥250	¥120

サンプルコード（標準モジュール）

```
01 Sub sample()
02   Dim db As Database ←データベースオブジェクト用の変数を宣言
03   Set db = CurrentDb ←現在操作しているデータベースに接続
04   Dim sql As String ←SQLを格納する文字列型の変数
05   sql = "UPDATE T1 SET fListPrice=300 WHERE fCakeID='P008';" ←SQLを格納
06   db.Execute sql, dbFailOnError ←レコードを更新するSQLを実行
07   db.Close ←データベースを閉じる
08   Set db = Nothing ←データベースオブジェクトの破棄
09 End Sub
```

データベースへ接続・操作する「DAO」を利用する

DAO（Data Access Objects）と呼ばれるVBAとデータベースの仲介をしてくれる技術がAccessに標準で備わっています。これは「ツール」→「参照設定」で右図のようにチェックが入っている状態で利用できます。最新バージョンのAccessでは最初からチェックが入るはずですが、もしエラーが出る場合はここを確かめてみてください。



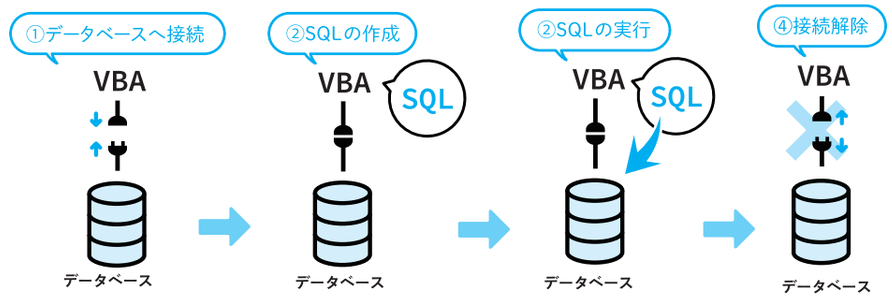
このサンプルで利用している変数dbはDAOを利用したDatabase型のオブジェクトで、このオブジェクトを使ってデータベース操作を行います。

4つの段階を踏む

データベースを操作するには、次の4つの段階を踏みます。

①データベースオブジェクトにAccessデータベースをセット（接続）して、VBAから操作できる状態にします。②**SQL**（Structured Query Language）というデータベース用の言語で、命令文を作成します。VBAの中では、SQLは「"」（ダブルクォーテーション）で囲んで文字列として扱います。③SQLを実行します。④セットしたデータベースの接続を解除して、データベースオブジェクトを破棄します。

実行後はテーブルを開き直すかリボンの「すべて更新」ボタンで結果が確認できます。



この4つの段階と照らし合わせると、サンプルコードは次のような役割で動いています。

```
Dim db As Database
Set db = CurrentDb          ←①データベースへ接続
Dim sql As String
sql = "任意のSQL文"          ←②SQLの作成
db.Execute sql, dbFailOnError ←③SQLの実行
db.Close
Set db = Nothing            ←④接続解除
```

ONE POINT SQLの作成と実行は1行でもできる

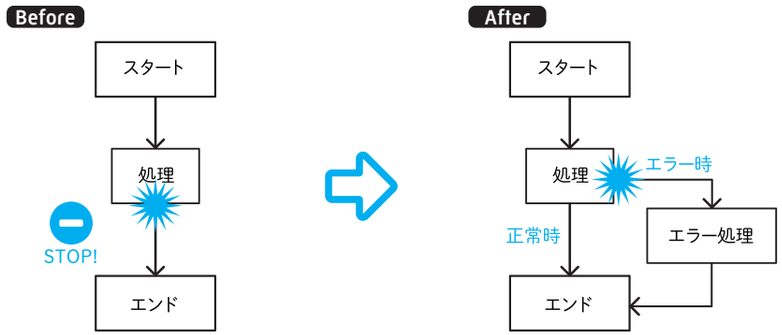
サンプルではSQLを文字列型の変数に格納してから実行していますが、実行時のパラメーターに直接SQLを指定することも可能です。

```
db.Execute "UPDATE T1 SET fListPrice=300 WHERE fCakeID='P008';", dbFailOnError
```

SQLの実行 / SQL / 実行のオプション

073 エラー時に処理を分岐する～エラートラップ

➔ **エラー処理には、「エラーが発生したときだけ指定の行へジャンプ」という指示があります。この指示により、エラーが発生したとき、発生しなかったときの処理を分岐させることができます。**



サンプルコード（標準モジュール）

```
01 Sub sample()                          ←エラーが発生したらERR_HANDLER行にジャンプする指示
02   On Error GoTo ERR_HANDLER
03   Dim n As Long                       ←整数型の変数を宣言
04   n = "abc"                           ←値の代入（数値か否かで結果が変わる）
05   Debug.Print "正常終了しました"      ←エラーが起こらなかったときの処理
06   Exit Sub                            ←正常時はここで終了
07
08 ERR_HANDLER:                          ←エラー時にジャンプするラベル行
09   Debug.Print Err.Number & ": " & Err.Description   ←エラー番号と内容を出力
10 End Sub
```

エラー発生時に指定行へジャンプする指示

このサンプルでは、3行目で整数型の変数nを宣言していて、4行目で変数nに代入する値が「数値」と判断できるかどうかで結果が変わります。これは、2行目の「On Error GoTo ○」の指示によるものです。

記述された次の行から、「エラー発生時にはプログラムを停止せずに○行へジャンプする」という指示になります。

```
On Error GoTo ERR_HANDLER
```
（On Error：エラーが起きたときの指示／GoTo：指定の位置へジャンプする／ERR_HANDLER：行ラベルを指定）

プログラムの行を識別するために名前を付ける

8行目に「ERR_HANDLER:」とありますが、これは行に名前を付ける記述です。

プログラム上で特定の行にジャンプさせるためには、行が識別できなくてはいけません。そのために付けるのが**行ラベル**です。任意の文字列に「:」を付けることで行ラベルとして認識されます。何を表す行なのか目的を端的に記述するとよいでしょう。

特にこのサンプルのように、エラーを検知し、分岐させて適切に処理するしくみのことを**エラートラップ**と呼びます。

エラーが発生しなかった場合

4行目で数値を代入した場合、エラーは発生しません。そのまま5行目に進み、イミディエイトウィンドウに「正常終了しました」と出力したあと、6行目でプロシージャを終了します。

ここで終了しておかないと、さらに先に進んで「ERR_HANDLER:」後のコードも実行してしまうことになるので、「On Error GoTo ○」の指示をしたら、エラー処理のあるブロックの直前にはExit Subを忘れないようにしましょう。

```
Sub sample()
  On Error GoTo ERR_HANDLER
  Dim n As Long
  n = 100
  Debug.Print "正常終了しました"
  Exit Sub                          ←ここで終了するのでERR_HANDLER:より先を通らない

ERR_HANDLER:
  Debug.Print Err.Number & ": " & Err.Description
End Sub
```

エラーが発生した場合

4行目で数値以外のものを代入した場合、5行目が実行不可能のためエラーが発生しますが、2行目の「On Error GoTo」の指示により、プログラムを停止せずに8行目の「ERR_HANDLER:」へジャンプします。

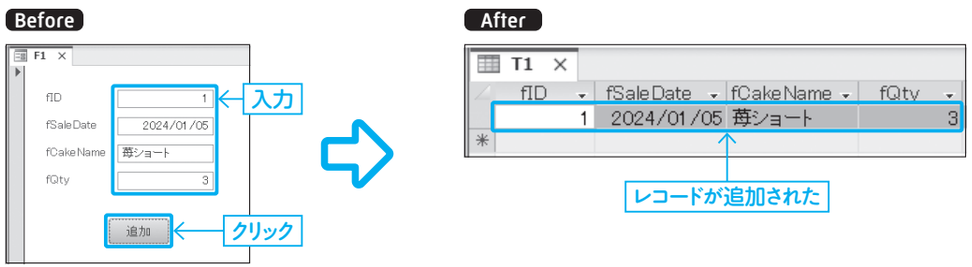
そこから先へ進み、9行目でエラー番号と内容をイミディエイトウィンドウへ出力したあと、10行目でプロシージャを終了します。

```
Sub sample()
  On Error GoTo ERR_HANDLER
  Dim n As Long
  n = "abc"                         ←エラーが発生するのでERR_HANDLER:へジャンプ
  Debug.Print "正常終了しました"
  Exit Sub

ERR_HANDLER:
  Debug.Print Err.Number & ": " & Err.Description
End Sub
```

110 ユーザーが入力した値でINSERT構文を実行する

➔ **Accessでアプリを作るならば、フォームとコントロールを使って、ユーザーに入力してもらった値でデータを変更するのもおすすめです。ここではINSERT（追加）のコードを紹介します。**



サンプルコード（Form_F1モジュール）

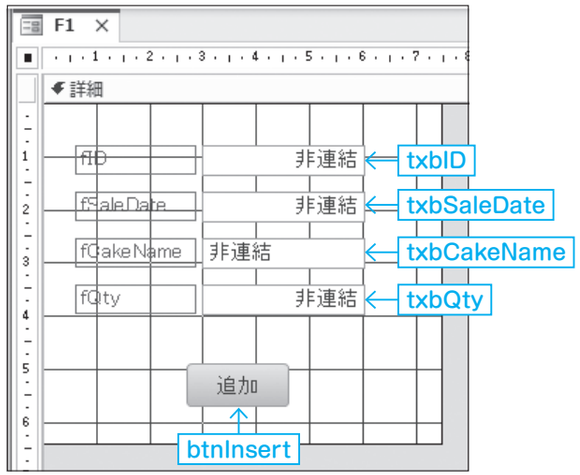
```
01 Private Sub btnInsert_Click() ←追加ボタンクリック時
02   Dim db As Database
03   Set db = CurrentDb ←データベース接続
04
05   Dim sql As String
06   sql = "INSERT INTO T1(" & _
07           "fID, " & _ ←ID
08           "fSaleDate, " & _ ←販売日
09           "fCakeName, " & _ ←ケーキ名
10           "fQty" & _ ←数量
11         ") VALUES(" & _
12           Me.txbID.Value & ", " & _ ←IDの値
13           "#" & Me.txbSaleDate.Value & "#, " & _ ←販売日の値
14           "'" & Me.txbCakeName.Value & "', " & _ ←ケーキ名の値
15           Me.txbQty.Value & _ ←数量の値
16         ");"
17   db.Execute sql, dbFailOnError ←SQL実行
18
19   db.Close ←接続解除
20   Set db = Nothing
21 End Sub
```


テキストボックスとボタンを配置しておく

このサンプルの「F1」フォームには、右図のような名前のテキストボックスが4つ、ボタンが1つ配置してあります。

テキストボックスは、データを収納する「T1」テーブルのフィールドに対応した名前になっています。

追加ボタンは、プロパティシートの「イベント」タブの「クリック時」からイベントプロシージャを作成しておきます。



ボタンのクリックイベントでINSERT構文のSQLを実行する

Form_F1モジュールのbtnInsert_Clickプロシージャにコードを記述することで、追加ボタンをクリックすると起動します。SQLを実行する基本の部分は**063**と同じです。

VALUES以降のブロックで、値の部分をコントロールに置き換えます。

SQLは文字列型で扱うため、改行するごとに1つの文字列となるように、「&」を使って結合しながら記述します。また、テキスト型の値は「' 」、日付型は「#」で囲み、最後の要素以外には「,」を含める必要があります。

```
") VALUES(" & _
  "1, " & _
  "#2024/01/05#, " & _
  "'苺ショート', " & _
  "3" & _
");"
```

⇨

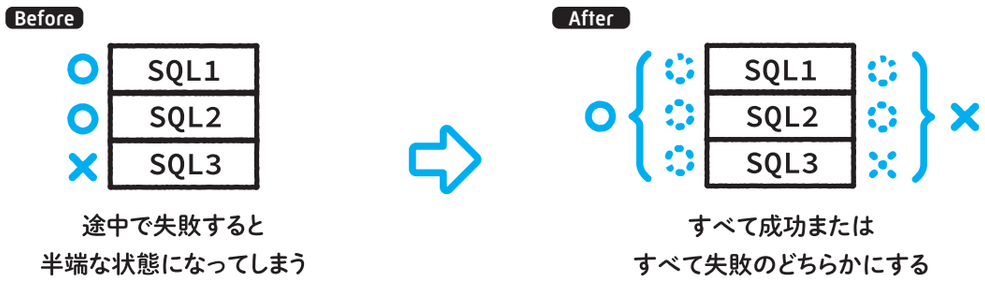
```
") VALUES(" & _
  Me.txbID.Value & ", " & _
  "#" & Me.txbSaleDate.Value & "#, " & _
  "'" & Me.txbCakeName.Value & "', " & _
  Me.txbQty.Value & _
");"
```

実行後はテーブルを開き直すかリボンの「すべて更新」ボタンで結果が確認できます。

なお、このサンプルは必要最低限のコードになっているため、エラー処理が含まれていません。実際のアプリケーションに組み込む際は、**075**を参考にエラートラップも実装しましょう。プロシージャの先頭で「実行してもよろしいですか？」といった問いに、OKかキャンセルを選択させるメッセージボックスを入れるのもおすすめです。

116 トランザクション制御を実行する

➔ 複数のSQLがどこかで失敗してしまうと、処理は半端な状態になってしまいます。そんな事態が困る場面では、複数のSQLを1つのまとまりとして扱い、「すべて成功」または「すべて失敗」とする、トランザクション制御を利用しましょう。



サンプルコード（標準モジュール）

```
01 Sub sample()
02   On Error GoTo ERR_HANDLER ←エラーが起きたらErrorHandler行にジャンプする指示
03
04   Dim ws As DAO.Workspace ←トランザクション用オブジェクト作成
05   Set ws = DBEngine(0)
06
07   Dim db As Database ←接続
08   Set db = CurrentDb
09
10   Dim sql1 As String                    ←1つ目のSQL
11   sql1 = "INSERT INTO T1(fSaleID, fSaleDate) VALUES('S001', #2024/01/04#);"
12   Dim sql2 As String
13   sql2 = "INSERT INTO T2(fDetailID, fSaleID, fCakeName, fQty) " & _
14          "VALUES('D001', 'S001', '苺ショート', 3);" ←2つ目のSQL
15   Dim sql3 As String
16   sql3 = "INSERT INTO T2(fDetailID, fSaleID, fCakeName, fQty) " & _
17          "VALUES('D002', 'S001', 'チョコレート', 2);" ←3つ目のSQL
18
19   ws.BeginTrans ←トランザクション開始
20
21   db.Execute sql1, dbFailOnError ←1つ目のSQLを実行
22   db.Execute sql2, dbFailOnError ←2つ目のSQLを実行
23   db.Execute sql3, dbFailOnError ←3つ目のSQLを実行
24
25   ws.CommitTrans ←確定
26
27   Debug.Print "正常終了しました" ←正常時のメッセージ
28   GoTo FINALLY
29
30 ERR_HANDLER: ←例外処理（エラーが起きたらここへジャンプ）
31   ws.Rollback ←元の状態へ戻す
32   Debug.Print Err.Number & ": " & Err.Description ←エラー時のメッセージ
33
34 FINALLY: ←最終処理
35   If Not db Is Nothing Then ←接続があったら解除
36     db.Close
37     Set db = Nothing
38   End If
39   If Not ws Is Nothing Then ←トランザクション用のオブジェクトを破棄
40     ws.Close
41     Set ws = Nothing
42   End If
43 End Sub
```

親子関係のあるテーブルで管理する

このサンプルには、販売の親情報の「T1」テーブルと、子情報の「T2」テーブルが用意されています。1回の販売で複数アイテムの販売があるときなどは、このような親子関係のあるテーブルで管理するのが一般的です。

サンプルコードでは、この2つのテーブルに、3つのSQLを実行します。1つ目のSQLは「T1」（親テーブル）へ「fSaleID」（販売ID）と「fSaleDate」（販売日）を追加します。2つ目と3つ目のSQLは「T2」（子テーブル）へ「fDetailID」（詳細ID）と「fSaleID」（販売ID）、「fCakeName」（ケーキ名称）、「fQty」（個数）を追加します。1つの販売に対し、アイテムが2つ売れた情報を登録します。