

はじめに	003
サンプルファイルの使い方	010

CHAPTER

1

フォームとレポート

001	フォームを開く	012
002	フォーム上の情報を参照する	013
003	条件を付けてフォームを開く	014
004	フォームの移動ボタンを非表示にする	016
005	フォームのレコードセレクトを非表示にする	017
006	開くウィンドウだけを操作できるようにする	018
007	フォームを閉じる	020
008	保存する／しないを指定してフォームを閉じる	021
009	別フォームへ遷移する	022
010	情報を持たせてフォームを遷移する	024
011	複数の情報を持たせてフォームを遷移する	026
012	メッセージを表示する	028
013	メッセージボックスのタイトルを変更する	029
014	メッセージボックスのテキストを改行する	030
015	メッセージボックスに任意のアイコンを表示する	032
016	メッセージボックスに表示するボタンを指定する	034
017	メッセージボックスでクリックされたボタンを判定する	036
018	フォームを閉じる前に確認メッセージを出す	038
019	インプットボックスで値を取得する	040
020	インプットボックスに既定値を設定する	042
021	レポートを開く	044
022	印刷プレビューでレポートを開く	045
023	条件を付けてレポートを開く	046
024	レコードが存在する場合のみレポートを開く	048
025	レポートを閉じる	050
026	レポートを印刷する	051
027	プリンターを指定してレポートを印刷する	052
028	PDF形式でレポートを出力する	053
029	レポート印刷日時を表示する／表示しない	054
030	画像ファイル(印鑑/社章)を指定して表示する	056
031	担当者によって表示する画像を切り替える	058

CHAPTER

2

コントロール

032	タイトルを変更する～ラベル	062
033	表示/非表示を切り替える～ラベル	063
034	文字の書式を変更する～ラベル	064
035	値を入力する～テキストボックス	066
036	値をクリアする～テキストボックス	067
037	動的な内容を入力する～テキストボックス	068
038	テーブルの情報を入力する～テキストボックス	069
039	空欄かどうかチェックする～テキストボックス	070
040	全角の入力を禁止する～テキストボックス	071
041	数値以外の入力を禁止する～テキストボックス	072
042	フォーカスの離脱を制御する～テキストボックス	074
043	入力文字数を制限する～テキストボックス	076
044	入力数値の範囲を制限する～テキストボックス	078
045	正の整数だけ許可する～テキストボックス	079
046	未許可の間、背景色を変更する～テキストボックス	080
047	複数コントロールをクリアする～テキストボックス	082
048	値を入力する～コンボボックス	084
049	値をクリアする～コンボボックス	085
050	選択肢を設定する～コンボボックス	086
051	選択肢にテーブルの情報を読み込む～コンボボックス	087
052	選択肢をクリアする～コンボボックス	088
053	n番目の選択肢を入力する～コンボボックス	089
054	選択肢を複数列表示する～コンボボックス	090
055	関連情報を自動入力する～コンボボックス	092
056	連動して選択肢を絞り込む～コンボボックス	094
057	連動して画像を表示する～コンボボックス	096
058	使用可否を切り替える～チェックボックス	098
059	削除アイテムの表示を切り替える～チェックボックス	100
060	実行前に確認メッセージを表示する～ボタン	102
061	実行前に空欄をチェックする～ボタン	104
062	コントロールを初期化する～ボタン	106

CHAPTER

3

SQLとエラー処理

063	データベースを操作する	110
064	追加・更新・削除の命令を記述する	112
065	レコードセットを取得する	114
066	取得したレコードセットを出力する	116
067	長いSQL文を見栄えよく改行する	118
068	エラーを発生させる～エラー時の動きの確認	120
069	エラーを無視して進む～停止せずに実行を継続	121
070	エラー番号を取得する～原因の特定のため	122
071	エラー内容を取得する～日本語での説明文	123
072	エラー処理をリセットする～動きを元に戻す	124
073	エラー時に処理を分岐する～エラートラップ	126
074	特定のエラー時に処理を変更する	128
075	エラーの有無にかかわらず特定の処理を実行する	130

CHAPTER

4

レコードの取得と検索

076	テーブルを開く	134
077	選択クエリを開く	135
078	選択クエリを書き換える	136
079	テーブル／選択クエリをサブフォームに表示する	138
080	サブフォームの編集を不可にする	139
081	レコードセットをリストボックスへ出力する	140
082	リストボックスの列数を指定する	142
083	リストボックスの列幅を指定する	143
084	リストボックスの全幅を指定する	144
085	リストボックスにフィールド名を表示する	145
086	オプショングループで動的に処理を変更する	146
087	最初／最後のレコードを選択する	148
088	前のレコードへ移動する	150
089	次のレコードへ移動する	152
090	特定のレコードに移動する	154
091	レコードの選択を解除する	156
092	レコードが選択されているか判定する	158
093	選択されているレコードの情報を取得する	160

094	レコードを並び替える	162
095	レコードの初期状態を設定する	164
096	AND条件でレコードを絞り込む	166
097	OR条件でレコードを絞り込む	168
098	チェックボックスの値で情報を切り替える	170
099	コンボボックスの値でレコードを絞り込む	172
100	テキストボックスの値で範囲を指定して絞り込む	174
101	入力された文字を含むレコードを検索する	176
102	レコード数を取得する	178
103	フィールド数を取得する	180
104	計算結果をフィールドに表示する～演算フィールド	182
105	複数テーブルを組み合わせる～テーブル結合	184

CHAPTER

5

データの編集

106	アクションクエリを実行する	188
107	警告メッセージの表示／非表示を制御する	190
108	任意の警告メッセージを表示する	191
109	パラメーター付きのアクションクエリを実行する	192
110	ユーザーが入力した値でINSERT構文を実行する	194
111	ユーザーが入力した値でUPDATE構文を実行する	196
112	ユーザーが入力した値でDELETE構文を実行する	198
113	「最大値+1」の新規IDを算出する	200
114	「文字列+最大値+1」の新規IDを算出する	201
115	連結フォームの自動保存を制御する	202
116	トランザクション制御を実行する	208

CHAPTER

6

文字列／数値／日付操作

117	半角／全角の変換を行う	214
118	小文字／大文字の変換を行う	215
119	ひらがな／カタカナの変換を行う	216
120	文字を置換する	217
121	スペースを取り除く	218
122	文字数を取得する	219
123	左／右からn文字取り出す	220

124	x文字目からn文字取り出す	221
125	指定文字の位置を調べる	222
126	指定文字があるか調べる	223
127	パスからフォルダー名とファイル名を分離する	224
128	数値かどうか判定する	226
129	数値を四捨五入する	227
130	数値を3桁区切りにする	228
131	数値をゼロ埋めして表示する	229
132	現在の日付/時刻を取得する	230
133	日付から曜日を取得する	231
134	日付の書式を変更する	232
135	時間の書式を変更する	234
136	日付を加算/減算する	236
137	日付の間隔を計算する	237
138	データの型を変換する	238
139	日付かどうか判定する	239
140	日付の年/月/日をそれぞれ取得する	240
141	年/月/日から日付を合成する	241
142	指定月の初日を取得する	242
143	指定月の末日を取得する	243
144	その日が週末かどうか判定する	244

CHAPTER
7

ユーザーによる処理の分岐

145	指定のIDの有無をチェックする	246
146	パスワードを照合する	248
147	指定IDが管理者かどうかチェックする	250
148	メニューフォームでログインする	252
149	閲覧できるテーブルを制限する～ボタン	254
150	閲覧できるテーブルを制限する～オプションボタン	256
151	表示フィールドを制限する～リストボックス	258

CHAPTER
8

VBAの文法と関数

152	Withで記述を省略する	262
153	オブジェクト変数を利用する	263

154	繰り返し処理(For～Next)で合計数を算出する	264
155	繰り返し処理(Do～Loop)で指定日以前だけ処理する	266
156	繰り返し処理(For Each～Next)で情報を取得する	268
157	条件分岐(If Else)でコントロール別にクリアする	270
158	条件分岐(Select Case)でコントロール別にクリアする	272
159	プロシージャを部品化して再利用する	274
160	引数を渡してプロシージャを呼び出す	276
161	関数を自作する	278
162	テキストボックスの数値チェックを関数化する	280
163	DLookup関数と同じ働き関数をDAOで作る	282
164	DMax関数と同じ働き関数をDAOで作る	284
165	その日が休日かどうか判定する	286
166	生年月日から年齢を取得する	288
167	トランザクション制御のSQL実行を関数化する	290

CHAPTER
9

データの連携

168	外部データをインポートする	294
169	保存済みのインポート操作を呼び出す	296
170	外部データへエクスポートする	298
171	保存済みのエクスポート操作を呼び出す	300
172	特殊フォルダーを利用する	302
173	フォルダー・ファイルの存在を確認する	303
174	選択ダイアログを利用する	304
175	テーブルからテーブルヘレコードを追加する	308
176	別ファイルにテーブルのバックアップを作成する	310

APPENDIX
付録

AccessおよびVBAの概要

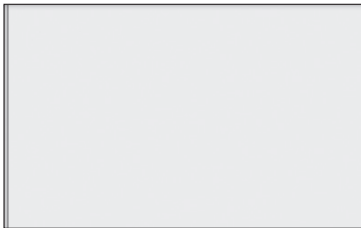
A-1	プログラミングの知識	314
A-2	プログラミングの工夫	320

索引	325
----	-----

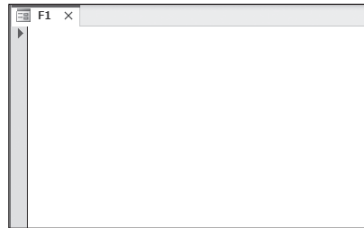
001 フォームを開く

→ AccessはGUI(グラフィカルユーザーインターフェース)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。

半角スペース ビュー(acNormalの場合は省略可)

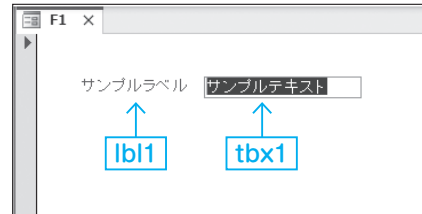
```
DoCmd.OpenForm "F1", acNormal
```

フォームを開く命令 フォーム名

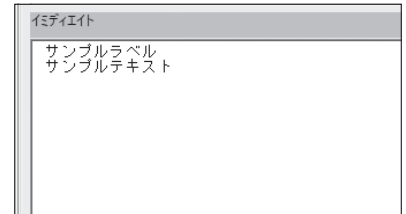
002 フォーム上の情報を参照する

→ 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.tbx1.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「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
```

イミディエイトウィンドウへ出力 オブジェクト名 プロパティ 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。

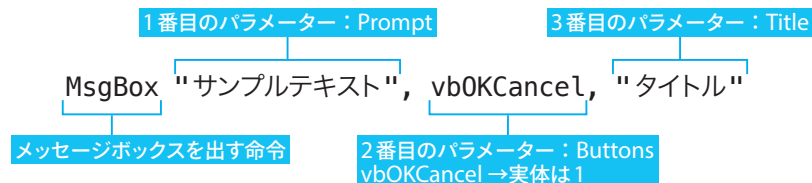


サンプルコード (標準モジュール)

```
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。

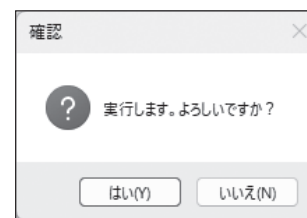
イメージ			
定数	vbOKOnly	vbOKCancel	vbAbortRetryIgnore
値	0	1	2
イメージ			
定数	vbYesNoCancel	vbYesNo	vbRetryCancel
値	3	4	5

「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。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。



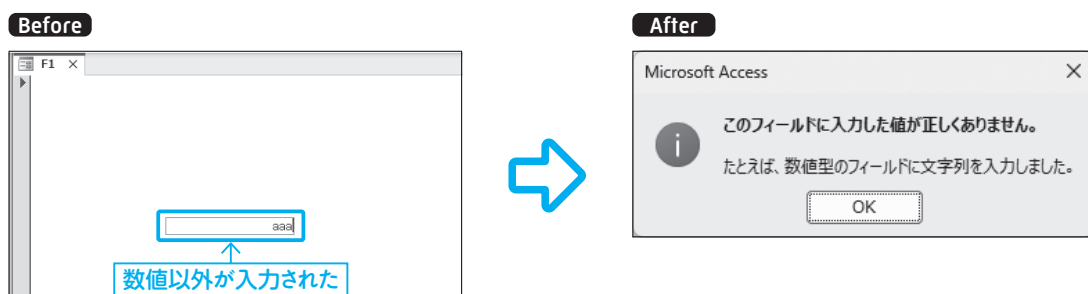
とはいえ、人間の目には数値の合計値では何を表現しているのかわかりにくいので、可読性の面から、コードの記述は定数をおすすめします。

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。

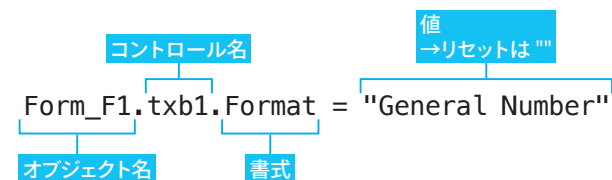
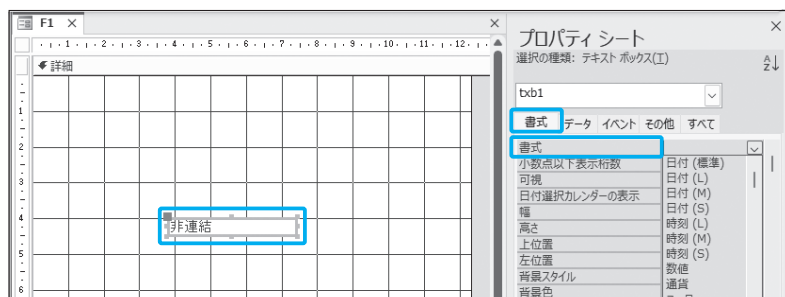


サンプルコード (標準モジュール)

```
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	内容	設定値
日付 (標準)	General Date	数値	General Number
日付 (L)	Long Date	通貨	Currency
日付 (M)	Medium Date	ユーロ	Euro
日付 (S)	Short Date	固定	Fixed
時刻 (L)	Long Time	標準	Standard
時刻 (M)	Medium Time	パーセント	Percent
時刻 (S)	Short Time	指数	Scientific

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

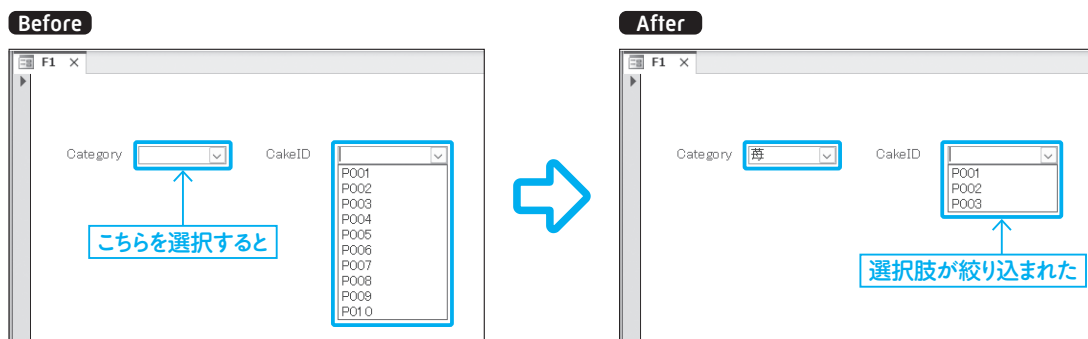
ONE POINT VBA だけでなくプロパティシートも活用しよう

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

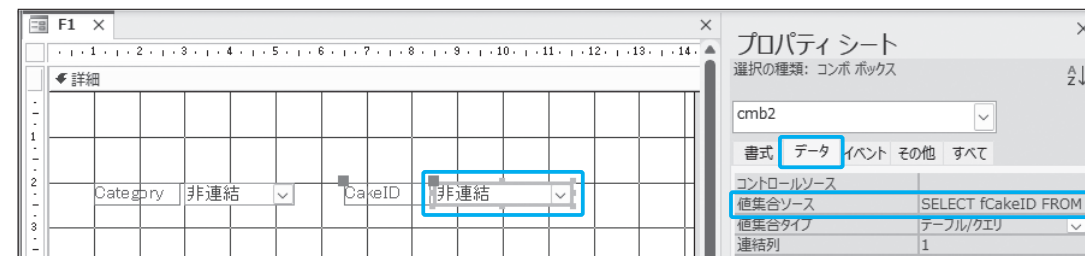
この動きは055と同じくコンボボックスの「更新後処理」(AfterUpdate) イベントプロシージャを使って、1つ目のコンボボックスの選択が変更されたら、2つ目のコンボボックスの選択肢を設定し直す、というしくみにします。

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。

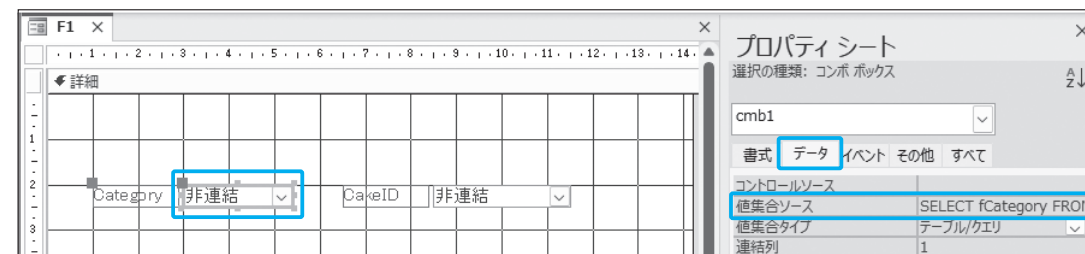
fCakeID	fCakeName	fListPrice	fCostPrice	fCategory
P001	苺ショート ビース	¥350	¥150	苺
P002	苺ショート 5号	¥2,500	¥900	苺
P003	苺ショート 6号	¥3,200	¥1,000	苺
P004	チョコレート ビース	¥320	¥130	チョコレート
P005	チョコレート 4号	¥1,600	¥600	チョコレート
P006	チョコレート 5号	¥2,000	¥700	チョコレート
P007	プレーンマフィン	¥250	¥90	マフィン
P008	チョコレートマフィン	¥250	¥100	マフィン
P009	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 & "';"
```

取り出したい
フィールド名

対象の
テーブル名

取り出すレコードの条件

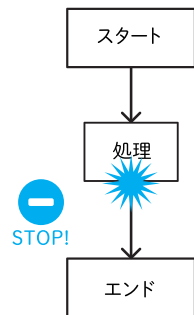
SQLではレコードに条件を付ける部分は**WHERE句**と呼ばれ、038や055で紹介したDLookup関数の条件を記述する部分にも使われています。

073

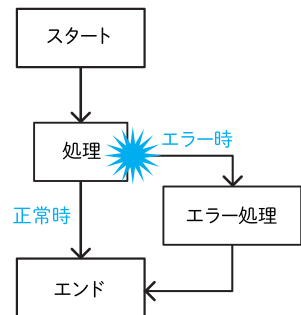
エラー時に処理を分岐する ～エラートラップ

➔ 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。

Before



After



サンプルコード (標準モジュール)

```

01 Sub sample()
02   On Error GoTo ERR_HANDLER
03   Dim n As Long
04   n = "abc"
05   Debug.Print "正常終了しました"
06   Exit Sub
07
08 ERR_HANDLER:
09   Debug.Print Err.Number & ": " & Err.Description
10 End Sub

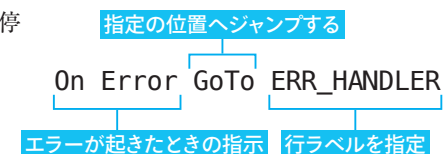
```

02 On Error GoTo ERR_HANDLER ← エラーが発生したらERR_HANDLER行にジャンプする指示
 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" ← エラーが起らなかったときの処理
 06 Exit Sub ← 正常時はここで終了
 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

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。



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なりません。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"
  Debug.Print "正常終了しました"
  Exit Sub

```

エラーが発生するので ERR_HANDLER:へジャンプ

```

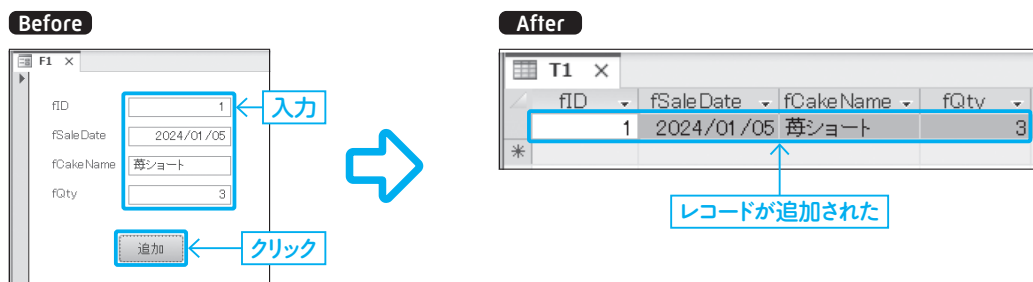
ERR_HANDLER:
  Debug.Print Err.Number & ": " & Err.Description
End Sub

```

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.txtID.Value & ", " & _ ← IDの値
13     "#" & Me.txtSaleDate.Value & "#, " & _ ← 販売日の値
14     "" & Me.txtCakeName.Value & "', " & _ ← ケーキ名の値
15     Me.txtQty.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

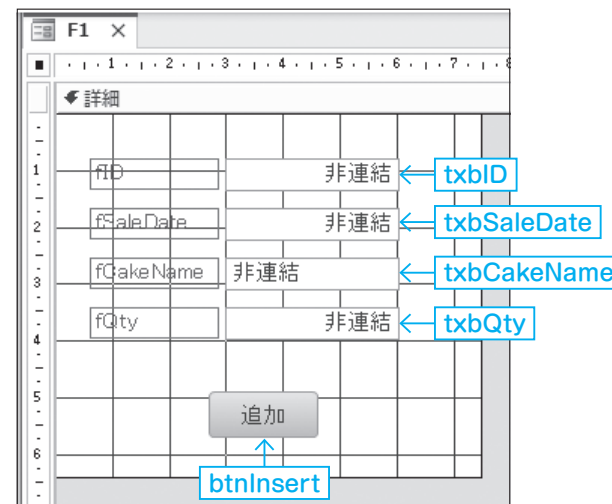
```

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.txtID.Value & ", " & _
  "#" & Me.txtSaleDate.Value & "#, " & _
  "" & Me.txtCakeName.Value & "', " & _
  Me.txtQty.Value & _
");"

```

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

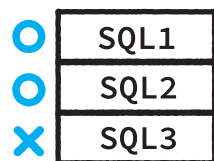
なお、このサンプルは必要最低限のコードになっているため、エラー処理が含まれていません。実際のアプリケーションに組み込む際は、075を参考にエラーラップも実装しましょう。プロシージャの先頭で「実行してもよろしいですか?」といった問いに、OKかキャンセルを選択させるメッセージボックスを入れるのもおすすめです。

116

トランザクション制御を実行する

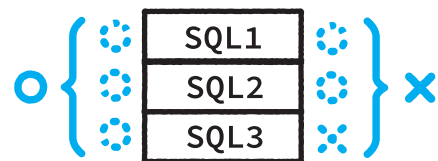
→ 複数のSQLがどこかで失敗してしまうと、処理は半端な状態になってしまいます。そんな事態が困る場面では、複数のSQLを1つのまとまりとして扱い、「すべて成功」または「すべて失敗」とする、トランザクション制御を利用しましょう。

Before



途中で失敗すると
半端な状態になってしまう

After



すべて成功または
すべて失敗のどちらかにする

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。

T1		T2			
fSaleID	fSaleDate	fDetailID	fSaleID	fCakeName	fQty
S001	2024/01/04	D001	S001	苺ショート	3
		D002	S001	チョコレート	2

このIDを同じにして管理する