



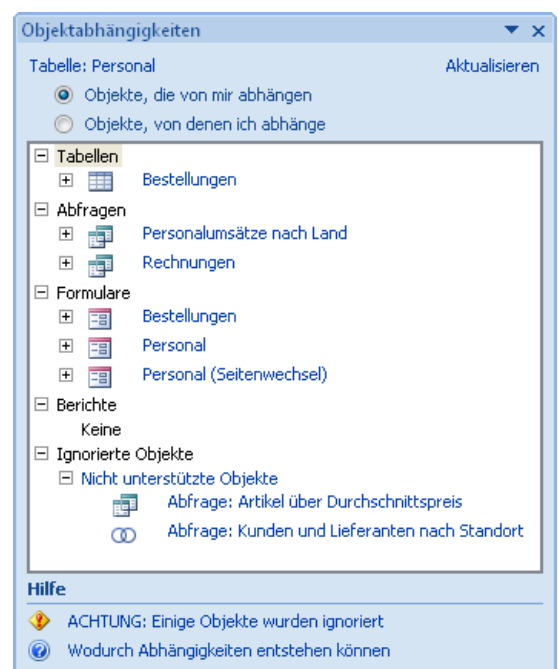
In Access-Datenbanken verliert man relativ schnell den Überblick, welches Objekt von welchem anderen abhängig ist. Für fremde Datenbanken, in denen "nur mal schnell" eben was geändert werden soll, gilt das noch viel mehr. Das hier gezeigte AddIn *relAcc* zeigt, welche Probleme bei der Analyse von Datenbanken auftreten und wie diese sich lösen lassen. Es kann allerdings auch nicht wirklich alle Abhängigkeiten ermitteln, weil diese sich vor allem im VBA-Code beispielsweise in Variablen verstecken können.

### Vorhandene Möglichkeiten

Access selber kennt seit der Version 2003 die so genannten "Objektabhängigkeiten", welche einen Teil der hier gewünschten Analyse schon bieten. Sie können die Vorgänger und Nachfolger zu einem markierten Objekt (auch über mehrere Ebenen hinweg) anzeigen.

Allerdings muss dazu nicht nur die (fehlerbehaftete und langsame) Objektnamen-Autokorrektur aktiviert sein, sondern es fehlen sogar wesentliche Objekte: berücksichtigt werden bis Access 2003 keine Aktionsabfragen und bis einschließlich Access 2010 auch keine Union-Abfragen, keine Unterabfragen, keine Makros und kein VBA-Code. Das ist für eine ernsthafte Analyse ungeeignet.

Alternativ gibt es viele Tools, die diese Lücke schließen wollen und dabei meistens umfangreiche TreeViews oder Tabellen erzeugen. Mit zunehmender Anzahl der analysierten Objekte wird jedoch eine grafische Darstellung immer wichtiger, um noch den Überblick zu behalten oder gezielt einzelne Objekte betrachten zu können.



Der Vorgänger zu *relAcc*, das ebenfalls von mir entwickelte AddIn *Accplorer*, konnte zwar bereits eine grafische Ausgabe in einem interaktiven Formular oder einem statischen Bericht erzeugen. Bei vielen Objekten in der analysierten Datenbank kam auch dieses AddIn aber recht schnell an die Grenzen der Bedienbarkeit und Übersichtlichkeit oder schlicht der zulässigen Kontrollelemente auf einem Bericht.

Das *relAcc*-AddIn trennt daher die Analyse von der Darstellung und erlaubt damit die Nutzung der jeweils optimalen Grafik-Engine. Bereits realisiert ist die grafische Ausgabe in Access (als Bericht), Excel, Visio und MSProject, wobei diese drei externen Dateiformate alle eine interaktive Nachbearbeitung der Grafik erlauben. Weitere Ausgabemöglichkeiten lassen sich dank der einheitlichen Schnittstelle leicht ergänzen.

### Vier Phasen

Die Analyse einer Datenbank lässt sich in 4 Phasen gliedern:

- alle Eigenschaften auslesen (auch solche, die Access nicht so gerne herausrückt)
- Objekte mit ihrem Typ sicher erkennen (auch solche ohne Namenskonventionen)

- alle Relationen ermitteln (auch solche, die Access selber nicht findet)
- Ergebnisse als Grafik ausgeben (auch, wenn keine Spezial-Software installiert ist)

Auch wenn zur einfacheren Bedienung die ersten drei Phasen in der Oberfläche zu einem Schritt zusammengefasst werden, haben sie doch inhaltlich unterschiedliche Anforderungen.

## Eigenschaften auslesen

Im ersten Schritt ermittelt *relAcc* die in der Datenbank vorhandenen Objekte und trägt sie mit eindeutiger ID in der internen Tabelle *tblObjekte* ein. Dabei werden auch Unterobjekte ermittelt wie Steuerelemente (in Formularen und Berichten) und vor allem Untermakros und einzelne Prozedurnamen im VBA-Code. Das ist notwendig, weil sonst vor allem in VBA-Modulen scheinbare Zirkelbezüge auftreten, obwohl in Wirklichkeit nur eine erste Prozedur eine zweite im gleichen Modul aufruft.

objID	objItemID	objTypeRef	objName	objDetail	objEbene	objProID	objVorgaenger
64	54	5	Form_frmStart	imgLaunchAddr_Click	1		
65	54	5	Form_frmStart	btnAbschritt_Click	1		
66	54	5	Form_frmStart	btnAbschritt2_Click	1		
67	54	5	Form_frmStart	lstReferenzen_Click	2		
68	54	5	Form_frmStart	PlatzPruefenUndMelden	2		
69	54	5	Form_frmStart	tblVielere_Aktualisieren	2		
70	54	2	zsmfEigenschaften		3		
71		2	zsmfVorNach		1		
72	71	5	Form_zsmfVorNach	lstNachfolger_ObjClick	2		
73	71	5	Form_zsmfVorNach	lstObjekte_Click	2		
74	71	5	Form_zsmfVorNach	lstVorgaenger_ObjClick	2		
75		4	macAllesDrin		5		
76	75	4	macAllesDrin	HierFolgtNeues	1		
77		4	macAllesFrueh		1		
78		4	macMehrere		2		
79	78		macMehrere	UntermakroName	1		
80		3	Bericht		1		
81		3	Bericht		1		

## Objekttypen sicher erkennen

Zu jedem Objekttyp gibt es in *tblEigenschaften* eine frei editierbare Sammlung von auszulesenden Eigenschaften. Dort lässt sich nicht nur angeben, welche Eigenschaften überhaupt untersucht werden sollen, sondern auch, welcher Objekttyp sich daraus für die Fundstelle ergeben wird. Beispielsweise kann nach einem FROM-Schlüsselwort in SQL nur eine Tabelle oder Abfrage folgen. Durch die Suche des Namens inklusive Objekttyp wird die zugehörige ID auch bei mehreren gleichnamigen Objekten recht sicher ermittelt.

eigID	eigTypeRef	eigName	eigTypeRefArgument
1	0	SourceTableName	0
2	2	RecordSource	-10
3	3	RecordSource	-10
4	7	RowSource	-10
5	7	SourceObject	-10
6	5	OpenRecordset	-10
7	5	Execute	1
8	5	DoCmd.OpenTable	0
9	5	DoCmd.OpenQuery	1
10	5	DoCmd.OpenForm	2
11	5	DoCmd.OpenReport	3
12	5	DoCmd.OpenReport	3
13	4	RepaintObject	-1
14	4	RepaintObject	-1
15	4	RepaintObject	-1
16	4	RepaintObject	-1
17	4	RepaintObject	-1
18	4	RepaintObject	-1
19	4	RepaintObject	-1
20	4	RepaintObject	-1
21	4	RepaintObject	-1
22	4	RepaintObject	-1
23	4	RepaintObject	-1
24	4	RepaintObject	-1
25	4	RepaintObject	-1
26	4	RepaintObject	-1
27	4	RepaintObject	-1
28	4	RepaintObject	-1
29	4	RepaintObject	-1
30	4	RepaintObject	-1
31	4	RepaintObject	-1
32	4	RepaintObject	-1
33	4	RepaintObject	-1
34	4	RepaintObject	-1
35	4	RepaintObject	-1
36	4	RepaintObject	-1
37	4	RepaintObject	-1
38	4	RepaintObject	-1
39	4	RepaintObject	-1
40	4	RepaintObject	-1
41	4	RepaintObject	-1
42	4	RepaintObject	-1
43	4	RepaintObject	-1
44	4	RepaintObject	-1
45	4	RepaintObject	-1
46	4	RepaintObject	-1
47	4	RepaintObject	-1
48	4	RepaintObject	-1
49	4	RepaintObject	-1
50	4	RepaintObject	-1
51	4	RepaintObject	-1
52	4	RepaintObject	-1
53	4	RepaintObject	-1
54	4	RepaintObject	-1
55	4	RepaintObject	-1
56	4	RepaintObject	-1
57	4	RepaintObject	-1
58	4	RepaintObject	-1
59	4	RepaintObject	-1
60	4	RepaintObject	-1
61	4	RepaintObject	-1
62	4	RepaintObject	-1
63	4	RepaintObject	-1
64	4	RepaintObject	-1
65	4	RepaintObject	-1
66	4	RepaintObject	-1
67	4	RepaintObject	-1
68	4	RepaintObject	-1
69	4	RepaintObject	-1
70	4	RepaintObject	-1
71	4	RepaintObject	-1
72	4	RepaintObject	-1
73	4	RepaintObject	-1
74	4	RepaintObject	-1
75	4	RepaintObject	-1
76	4	RepaintObject	-1
77	4	RepaintObject	-1
78	4	RepaintObject	-1
79	4	RepaintObject	-1
80	4	RepaintObject	-1
81	4	RepaintObject	-1
82	4	RepaintObject	-1
83	4	RepaintObject	-1
84	4	RepaintObject	-1
85	4	RepaintObject	-1
86	4	RepaintObject	-1
87	4	RepaintObject	-1
88	4	RepaintObject	-1
89	4	RepaintObject	-1
90	4	RepaintObject	-1
91	4	RepaintObject	-1
92	4	RepaintObject	-1
93	4	RepaintObject	-1
94	4	RepaintObject	-1
95	4	RepaintObject	-1
96	4	RepaintObject	-1
97	4	RepaintObject	-1
98	4	RepaintObject	-1
99	4	RepaintObject	-1
100	4	RepaintObject	-1

## Relationen ermitteln

Für jedes gefundene Objekt in einer Eigenschaft lässt sich nun in *tblRelationen* als m:n-Verknüpfung der beiden IDs festhalten, dass hier eine Abhängigkeit besteht. Grundsätzlich wäre hier die Analyse beendet, wenn die grafische Darstellung (wie in Visio und MSPROJECT) die optische Anordnung selber organisieren würde.

Da aber weder der Access-Bericht noch die Excel-Shapes dies leisten, ist zusätzlich eine Ermittlung der jeweiligen Ebene realisiert. Dabei erhalten alle Objekte zuerst die Ebene 1 und erhöhen diesen Wert rekursiv auf jeweils +1 gegenüber ihren direkten Vorgängern. Die horizontale Anordnung bei der grafischen Ausgabe ist in diesem Fall direkt passend zu ihrer Ebene. Wegen des erheblichen Zeitaufwands bei der Ebenenzuordnung (und der Gefahr von Endlosschleifen bei rekursiven VBA-Funktionen) lässt sich das auf eine bestimmte Anzahl von Ebenen beschränken.

relID	relObjID_von	relObjID_nach	relName
1	2	4	Relations("tblRelationen") ForeignTable
2	5	1	Relations("tblEigenschaften") ForeignTable
3	5	2	Relations("tblObjekte") ForeignTable
4	5	8	Queries("qryEigenschaften") SQL
5	1	8	Queries("qryEigenschaften") SQL
6	5	9	Queries("qryProjekteExport") SQL
7	2	9	Queries("qryProjekteExport") SQL
8	11	10	Queries("qryObjekte") SQL
9	5	11	Queries("qryObjekteSortiert") SQL
10	2	11	Queries("qryObjekteSortiert") SQL
11	11	12	Queries("qryObjekteSortiert") SQL
12	11	14	Queries("qryRelationenEinzel") SQL
13	4	14	Queries("qryRelationenEinzel") SQL
14	11	15	Queries("qryRelationenEinzel") SQL
15	4	15	Queries("qryRelationenEinzel") SQL
16	2	16	Queries("qryRelationenMittel") SQL
17	4	16	Queries("qryRelationenMittel") SQL
18	11	17	Queries("qryRelationenMittel") SQL
19	4	17	Queries("qryRelationenMittel") SQL
20	2	19	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(97) ("tblObjekte", "dbOpenTable")
21	4	19	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
22	3	19	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
23	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
24	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
25	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
26	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
27	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
28	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
29	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
30	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
31	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
32	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
33	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
34	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
35	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
36	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
37	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
38	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
39	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
40	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
41	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
42	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
43	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
44	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
45	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
46	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
47	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
48	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
49	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
50	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
51	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
52	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
53	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
54	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
55	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
56	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
57	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
58	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
59	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
60	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
61	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
62	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
63	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
64	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
65	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
66	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
67	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
68	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
69	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
70	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
71	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
72	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
73	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
74	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
75	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
76	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
77	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
78	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
79	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
80	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
81	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
82	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
83	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
84	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
85	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
86	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
87	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
88	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
89	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
90	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
91	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
92	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
93	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
94	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
95	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
96	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
97	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
98	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
99	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")
100	19	15	Modules("Form_frmSchritt1") Proc("Aktualisieren") Lines(99) ("tblRelationen", "dbOpenTable")

## Bedienungsanleitung

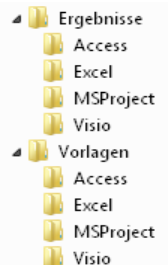
Das *relAcc*-AddIn ist eine \*.mdb-Datei der Version 2003 und bis einschließlich Access 2010 ohne Änderungen nutzbar. Damit es sowohl in der alten Menü-Oberfläche als auch den neuen Ribbons funktioniert, habe ich auf den Formularen selber eine Ribbon-Bedienung nachgebildet. Es gibt damit nur eine einzige Datenbank für alle Access-Versionen.

Wie jedes AddIn muss es über den AddIn-Manager installiert werden (Achtung: ab Access 2007 lässt sich ein AddIn nur mit Admin-Rechten installieren, die beim normalen Access-Start nicht aktiv sind!).

Je nach Access-Version rufst Du anschließend das Menü *Extras/Add-Ins/Add-In-Manager* oder im Ribbon das entsprechende Symbol wie im nebenstehenden Bild auf. Es sind keine weitergehenden Installationen, Registry-Einträge oder DLLs nötig.



Lediglich die Pfade müssen einmalig im Start-Formular eingestellt werden (und werden dann intern in *tblOptionen* gespeichert). Das AddIn erwartet an einer beliebigen Stelle eine Verzeichnis-Struktur nach dem nebenstehenden Muster oder erstellt diese per Schaltfläche. Auf dem Start-Formular kannst Du mit dem Info-Button nachsehen, ob alle diese Pfade korrekt eingestellt sind.



Alle Ribbons enthalten eine erste Gruppe für die Standard-Aktionen und (per Launchbutton aktivierbar) jeweils eine zweite Gruppe für erweiterte Aktionen. Dadurch bleibt die Bedienung übersichtlich.

## Start-Formular

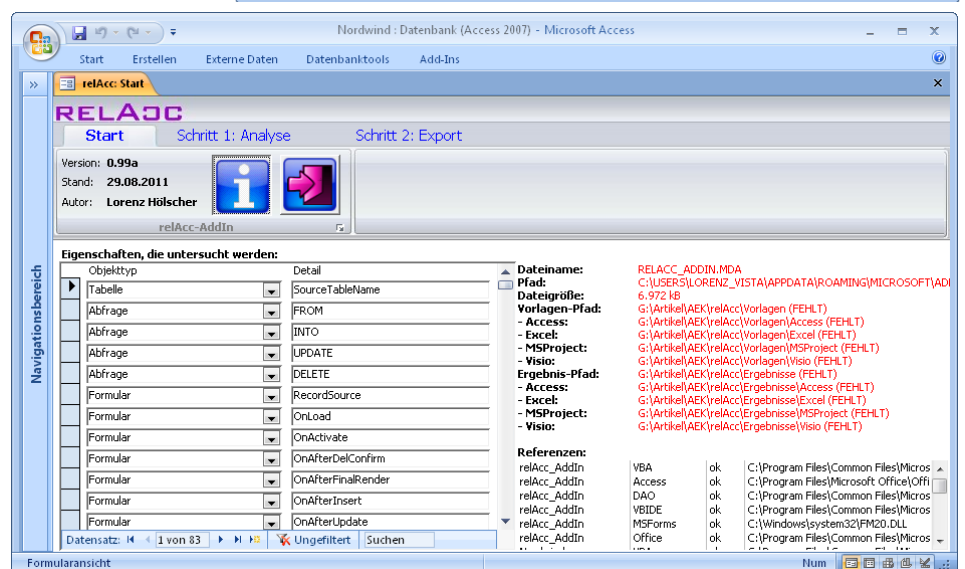
Beim Aufruf des Add-Ins über den entsprechenden Listeneintrag *relAcc [ri'læks]!*... erscheint automatisch im Vollbild das Start-Formular wie im nebenstehenden Bild. Es gibt insgesamt drei Formulare, zwischen denen mit den Registerkarten umgeschaltet wird.



Die wesentlichen Eigenschaften und Standardwerte siehst Du im Start-Formular beim

Klick auf den Info-Button. Dort lassen sich die Eigenschaften zu jedem Objekttyp angeben, die analysiert werden sollen.

Unabhängig von der eigentlichen Analyse sind die Verweise von AddIn und aktueller Datenbank zu sehen. Hier kannst Du bereits ableiten, ob alle erwarteten DLLs vorhanden sind.



Das AddIn zeigt (durch rote Färbung) automatisch an, wenn die Pfade für Vorlagen und Ergebnisse fehlen.

Um die Pfade zu ändern, musst Du den Launchbutton der *relAcc-AddIn*-Gruppe anklicken. Dann erscheint die zweite Gruppe *Vorlagen- und Ergebnis-Pfade* wie im nebenstehenden Bild. Gib hier die Pfade ein und speichere diese.



## Bedienung



Zeigt die Informationen über die Schlüsselwörter und die aktuelle Installation an.



Beendet das relAcc-AddIn.



Dieser Launchbutton zeigt die erweiterten Aktionen an.



Erzeugt die Standard-Verzeichnisse für Vorlagen und Ergebnisse (und speichert sie direkt in *tblOptionen*), kopiert aber keine Dateien hinein.



Ermöglicht die Auswahl des Vorlagen- oder Ergebnis-Pfades.

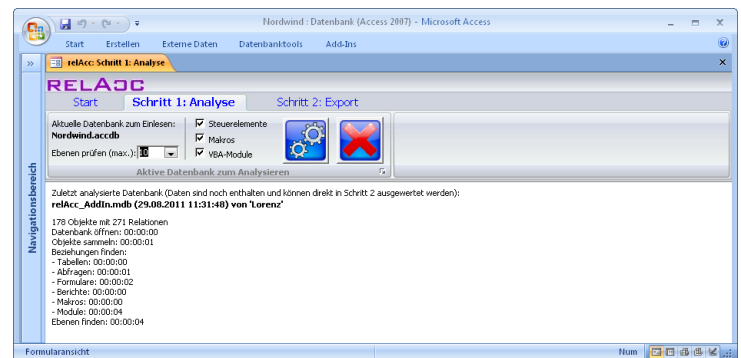


Speichert die Auswahl der Vorlagen- und Ergebnis-Pfade in der AddIn-Tabelle *tblOptionen*.

## Schritt 1: Analyse

Die eigentliche Analyse startet mit der Schaltfläche *Aktive Datenbank analysieren* auf dem Formular zum Schritt 2. Die Analyse-Daten sind immer im AddIn selber gespeichert und ersetzen dabei die vorherigen Werte.

Du kannst die Analyse-Daten aber in beliebig viele externe Speicher-Datenbanken exportieren und bei Bedarf wieder importieren. Diese Speicher-Datenbanken müssen aus der Vorlage *relAcc\_Leer.mdb* erzeugt werden, die im *\Vorlagen\Access-Verzeichnis* liegt.



Nach Klick auf den Launchbutton der Gruppe *Aktive Datenbank zum Analysieren* wird die Gruppe *Andere Analyse-Datenbanken verwalten* angezeigt.

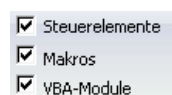
In dieser Gruppe kannst Du mit dem Button ganz rechts eine neue Speicher-Datenbank erzeugen und anschließend (wenn sie in der Combobox ausgewählt ist) die aktuell analysierten Daten mit dem entsprechenden Button dorthinein exportieren oder wieder importieren.



## Bedienung

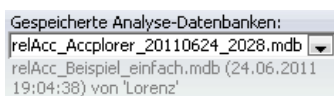


Zeigt den Namen der geöffneten Datenbank an.



Gibt an, wie viele Durchläufe zum Ermitteln der Ebene stattfinden. Die tatsächlich gefundene Ebene eines Objekts kann deutlich höher liegen, aber nur so lassen sich durch rekursive Code-Aufrufe ausgelöste Endlosschleifen verhindern. Die Ebenen-Ermittlung ist nur nötig für die Ausgabe als Access-Bericht oder Excel-Datei, sie kann sonst mit 0 ausgeschaltet werden.

Diese drei Teilbereiche können einzeln deaktiviert werden, da sie relativ zeitaufwändig sind, vor allem die Analyse der VBA-Module (Module in Formularen und Berichten werden hier ebenfalls deaktiviert).



Startet die eigentliche Analyse, bei der die Tabellen *tblObjekte* und *tblRelationen* mit den Werten der aktuellen Datenbank gefüllt werden.

Löscht (nach Rückfrage) die Inhalte der Tabellen *tblObjekte* und *tblRelationen*.

Dieser Launchbutton zeigt die erweiterten Aktionen an.

Dieser DatenImport-Button ist erst aktiv, wenn in der Combobox eine Datei ausgewählt ist. Deren Daten lassen sich damit in das AddIn importieren, sie ersetzen die bisherigen Werte.

Dieser DatenExport-Button ist ebenfalls erst aktiv, wenn in der Combobox eine Datei ausgewählt ist. Die Daten des AddIns werden damit in die markierte Datei exportiert. Meistens wird es sinnvoll sein, vorher eine leere Datei zu erzeugen.

Diese Combobox zeigt alle Dateien (aus dem Verzeichnis *\Ergebnisse\Access*) mit hellgrauer Zusatzinformation der ausgewählten Datei.

Mit dieser Schaltfläche wird aus *relAcc\_Leer.mdb* eine Kopie beliebigen Namens in das *\Ergebnisse\Access*-Verzeichnis geschrieben. Diese Kopie kann anschließend ausgewählt werden und mit dem Export-Button die aktuellen Daten erhalten.

## Schritt 2: Export

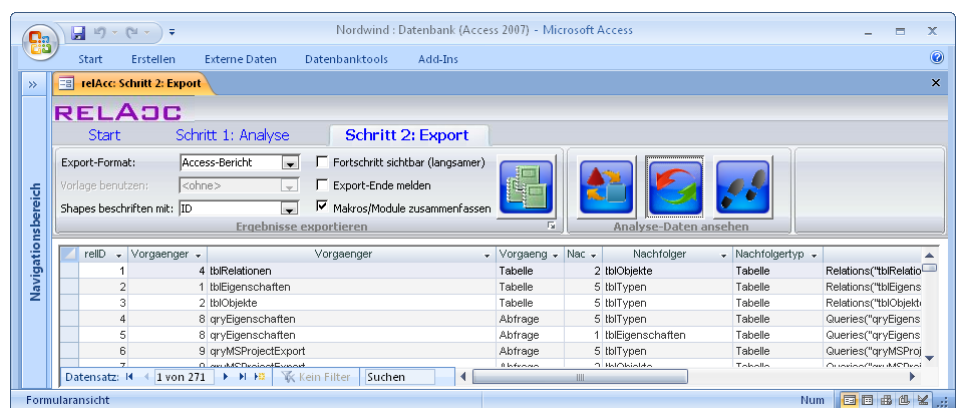
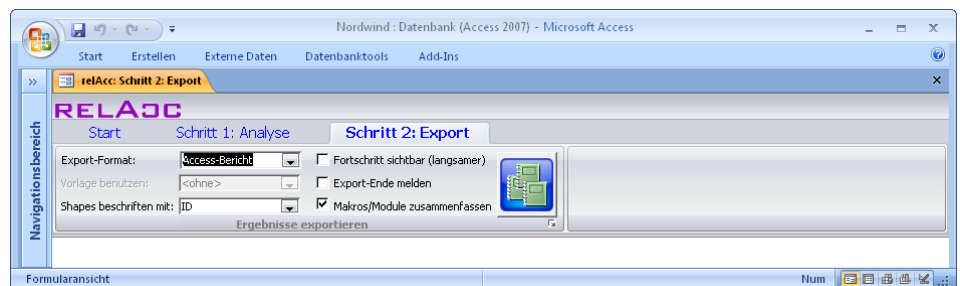
Die im AddIn gespeicherten Analyse-Daten lassen sich in verschiedene Grafikformate exportieren.

Für den Fall, dass keines der externen Dateiformate auf dem Rechner installiert ist, ist auch ein Export in einen neuen Access-Bericht möglich. Dieser ist dann noch ungespeichert und kann nur in der aktuellen Datenbank gespeichert werden.

Nach Klick auf den Launchbutton der Gruppe *Ergebnisse exportieren* lassen sich die Daten bereits hier in Listenform ansehen.

Die drei Buttons der Gruppe *Analyse-Daten ansehen* zeigen die (jeweils erweiterte) Tabelle *tblObjekte*, die Tabelle *tblRelationen* sowie eine Ansicht mit drei Listboxen. In dieser Ansicht werden für das in der Mitte ausgewählte Objekt oben seine direkten Vorgänger und unten seine direkten Nachfolger angezeigt.

Für den Export in ein externes Grafikformat (Excel, Visio oder MSPProject) wählst Du oben links den entsprechenden Eintrag aus, dabei ändert sich das Bild des Export-Buttons. Aus dem jeweiligen





\Vorlagen\<Programm>-Verzeichnis werden in der Combobox *Vorlage benutzen* alle vorhandenen Dateien mit der passenden Endung angezeigt.

Je nach Dateiformat können in der Vorlagendatei Makros oder Formatvorlagen hinterlegt werden. Am besten schaust Du Dir die beiliegenden Vorlagen an, wenn Du das zugehörige Programm überhaupt installiert hast.

## Bedienung

Diese Combobox erlaubt die Auswahl verschiedener Export-Formate. Der nebenstehende Export-Button ändert entsprechend sein Aussehen und zeigt das Icon des zugehörigen Programms an.

Abhängig vom benutzten Export-Format (Access-Berichte können keine Vorlagen nutzen) werden hier die Dateien des passenden Verzeichnisses in \Vorlagen angezeigt. Die Namen der Vorlagen sind beliebig.

Die entstehenden Kästchen können kurz mit ihrer *ID*, etwas ausführlicher mit *Name* (bei untergeordneten Objekten nur des Elternobjekts, bei Prozeduren also beispielsweise nur der Modulname *modTest*) oder komplett mit *GanzerName* (beispielsweise *modTest.LiesDaten*) beschriftet werden. Je nach Namensgebung in der analysierten Datenbank werden sich die Kästchen dann überlappen.

Normalerweise wird die Exportdatei nicht laufend aktualisiert, damit es etwa 1,5-2fach schneller geht. Wer zugucken willt, setzt hier ein Häkchen.

Der Access-Fortschrittsbalken beim Export ist eher unauffällig, mit einem Häkchen hier erscheint am Ende eine Meldung.

Oft ist es gar nicht wichtig, welche Prozedur eines Moduls eine Verknüpfung zu einem anderen Modul hat, es reicht, die Abhängigkeit der beiden Module zu sehen. Dafür können diese als Module (bzw. Makros statt benannter Untermakros) zusammengefasst werden. Das reduziert die exportierten Kästchen erheblich und macht es dadurch deutlich übersichtlicher.

Dieser Export-Button zeigt jeweils das Icon des Export-Formats an und ruft die zugehörige Export-Klasse auf.



Dieser Launchbutton zeigt die erweiterten Aktionen an.

Dieser ToggleButton zeigt die Objekte (nämlich aus *qryObjekteSortiertGruppirt* oder *qryObjekteSortiertEinzel*, abhängig von *Makros/Module zusammenfassen*) der aktuellen Analyse an.

Dieser ToggleButton zeigt die Relationen (nämlich aus *qryRelationenGruppirt* oder *qryRelationenEinzel*, abhängig von *Makros/Module zusammenfassen*) der aktuellen Analyse an

Dieser ToggleButton zeigt drei Listboxen an, deren mittlere alle Objekte auflistet. Zum jeweils markierten Objekt sind in der oberen Listbox deren direkte Vorgänger und in der unteren Listbox deren direkte Nachfolger zu sehen. Ein Doppelklick auf einen Vorgänger oder Nachfolger macht diesen zum mittig markierten Objekt.

## Technisches Konzept

Für das Verständnis oder die Erweiterung um andere Export-Formate ist es sicherlich hilfreich, die Konzepte beim Erzeugen der Grafiken zu kennen.

## Export Access/Excel

Diese beiden Export-Formate sind sich sehr ähnlich, denn die Position der Bericht-Controls (Access) bzw. Shapes (Excel) muss schon im VBA-Code durch Berechnung festgelegt werden.

Der Top-Wert ergibt sich aus einer Variablen, in der die Höhe (plus einer Abstand-Konstanten) der schon gezeichneten Objekte aufsummiert wird. Der Left-Wert entspricht der (mit einem Faktor multiplizierten) Ebene, so dass sich insgesamt ein Spaltenlayout mit Ausrichtung nach oben ergibt.

Da die Objekte standardisierte Namen ("Objekt " & ID) erhalten, kann in einer zweiten Schleife anschließend deren Position ermittelt werden, um die Verbindungslinien zu erzeugen.

Aus Platzgründen sind als Beschriftungen meistens die IDs sinnvoll, daher werden die Zusatz-Informationen in vorhandenen Objekt-Eigenschaften gespeichert.

Bei den Access-Controls ist es die Tag-(Marke-)Eigenschaft, bei Excel-Shapes bleibt als einzig brauchbare Eigenschaft nur der Alternativ-Text für das Web. Beide werden nicht automatisch in Quick-Infos o.ä. angezeigt, sondern müssen manuell im Entwurfsmodus (Access) oder im FormatZelle-Dialog sichtbar gemacht werden.

Der Vorteil dieser beiden Export-Formate besteht darin, dass sie immer (Access) oder fast immer (Excel) installiert sind. Ein Access-Bericht bleibt jedoch auch im Entwurfsmodus immer statisch, weil bei Verschiebungen von Rechtecken die Verbindungslinien (für die nicht einmal Pfeilspitzen möglich sind) manuell korrigiert werden müssen.

In Excel folgen die Pfeile immerhin dynamisch, wenn auch die Kästchen oft zwischen den vielen Linien schwierig zu markieren sind. Beide Formate kennen für die Zeichenobjekte keine Formatvorlagen, so dass die Farbgebung per VBA-Code erfolgen muss.

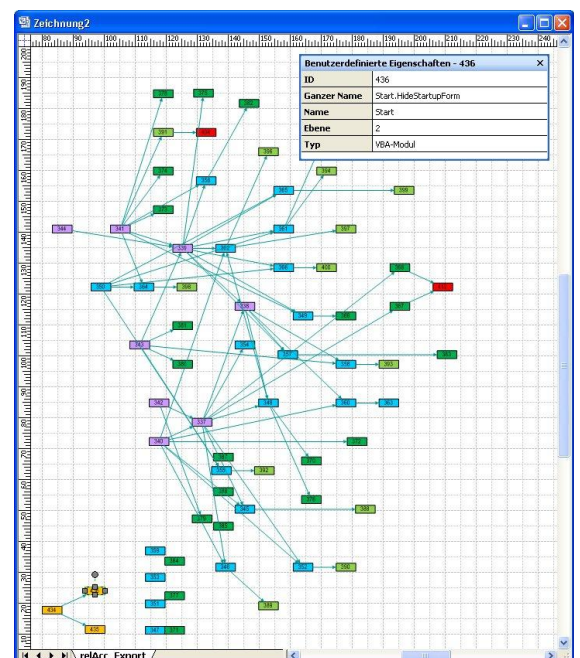
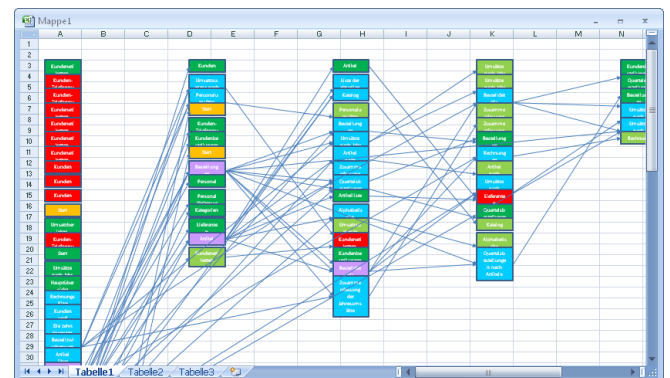
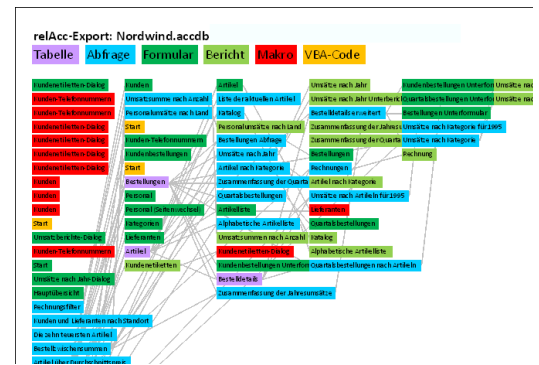
## Export Visio

Zwar gehört Visio zu den eher selten installierten Programmen, es bietet aber die optimale Grafik-Engine mit dynamischen Verbindern in verschiedenen Stilen, Formatvorlagen für alle Objekte und vor allem vollautomatischer Layout-Optimierung mit selbstständiger Vorgänger-/Nachfolger-Analyse.

Der Export beschränkt sich aus Sicht des VBA-Codes darauf, die Objekte und ihre Verbinder zu zeichnen. Eine Kenntnis von deren tatsächlicher Position ist dabei nicht nötig. Anschließend bietet Visio alle Möglichkeiten, um die Struktur vollautomatisch nach verschiedenen Kriterien neu auszurichten. Das bietet kein anderes Grafikprogramm so perfekt wie Visio.

Die als Beispiel enthaltene Vorlagendatei zeigt, wie durch eigene Makros zusätzliche Funktionalitäten leicht angefügt werden können.

Obwohl Visio ganz klar der Favorit für den grafischen Export ist, soll ein Nachteil nicht verschwiegen werden: das Ob-



jektmodell ist ziemlich eigenartig und die zugehörige VBA-Dokumentation dünn bis lückenhaft.

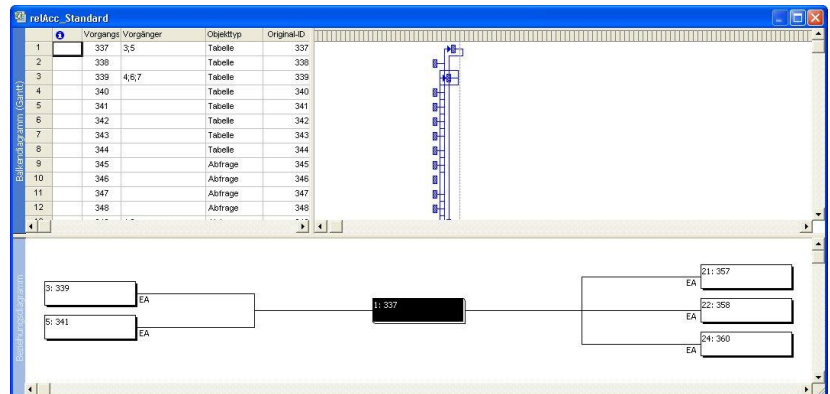
## Export MSProject

MSProject ist ein Kompromiss in der Nutzung als Grafik-Engine: nicht unbedingt viel häufiger installiert als Visio, aber mit Vorteilen im Umgang von Vorgängern/Nachfolgern.

Als scheinbare Projektvorgänge werden die Objekte hierhin exportiert, wobei deren "Dauer" egal ist, aber deren Relationen programmtypisch gut dargestellt werden können. Insbesondere die spezialisierten Ansichten

*Gantt/Beziehungsdiagramm* und *Netzplan* erleichtern die Analyse von Zusammenhängen deutlich.

Drei Besonderheiten müssen beim Export nach MSProject berücksichtigt werden: unabhängig von der tatsächlichen ID müssen alle Objekte immer lückenlos und beginnend mit der ID 1 nummeriert werden, die Vorgänger müssen als Semikolon-getrennte Liste der (auf 1 korrigierten!) IDs vorhanden sein und es sind keine rekursiven Abhängigkeiten darstellbar.



## Schwierigkeiten

Natürlich läuft bei so einem Projekt nicht alles glatt, es gibt Schwierigkeiten und ärgerliche Entdeckungen. Einige davon möchte ich nennen, weil sie Auswirkungen auf das AddIn und dessen Einsatz haben.

### Fehlende Eigenschaften

Wenn ich Eigenschaften von Objekten suche, dann kann ich entweder eine ganz detaillierte Liste erstellen für jeden Objekttyp oder ich suche beispielsweise grundsätzlich die ControlSource-Eigenschaft eines Controls. Im letzteren Fall gibt es natürlich viele Controls (wie Label oder Rectangle), welche diese Eigenschaft gar nicht besitzen.

```

Sub FindeEigenschaften(objDieses As Object, typWelches As enumTypen)
    Dim rcsEigenschaften As Recordset
    Dim strPropInhalt As String
    Dim strEigName As String

    Set rcsEigenschaften = CodeDb.OpenRecordset( _
        "SELECT * FROM tblEigenschaften WHERE eigTypNrRef=" & typWelches, dbOpenDynaset)
    With rcsEigenschaften
        Do Until .EOF
            strEigName = .Fields("eigName").Value
            On Error Resume Next
            strPropInhalt = objDieses.Properties(strEigName)
            If Err.Number = 0 And strPropInhalt <> "" And strPropInhalt <> "[Event Procedure]" Then
                Debug.Print "Eigenschaft '" & strEigName & "' von '" & objDieses.Name & "'"
            End If
            On Error GoTo 0
            .MoveNext
        Loop
    End With
End Sub

```

Es hat sich aber herausgestellt, dass es überhaupt nicht zeitkritisch ist, alle Controls nach allen gewünschten Eigenschaften zu durchsuchen und den auftretenden Fehler im Falle des Scheiterns einfach nur zu unterdrücken.

### VBA-Codezeilen

VBA-Code unterscheidet zwischen Zeilen im Editor, so wie ein Benutzer sie sieht, und Code-Zeilen, so wie der Interpreter sie ausführt. Codezeilen können dabei mit " \_" über mehrere Editorzeilen hinweg reichen, so dass das AddIn diese erst aus Folgezeilen zusammensetzen muss, damit folgende Parameter erkannt werden können.

```

Sub SchwierigeZeilen()
    'das ist kommentiert:
    'DoCmd.OpenForm "frmBeispiel"

    DoCmd.OpenForm _
        "frmNochEinBeispiel"
End Sub

```

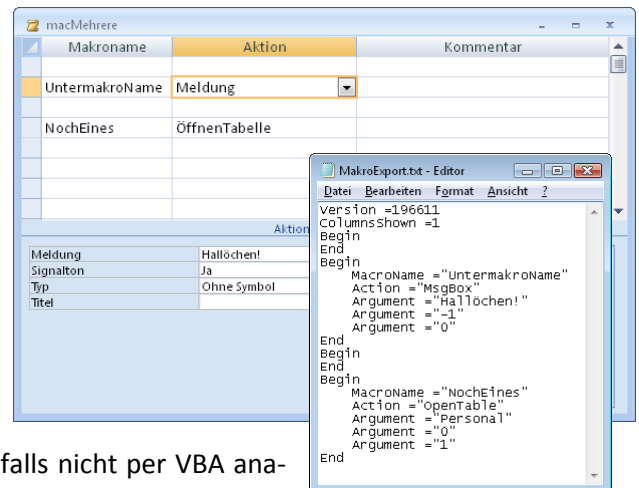
Außerdem ist es notwendig, Kommentarzeilen zu ignorieren, damit dort nicht fälschlich ungültige Relationen gefunden werden.



## Makros

Für Makros gibt es keinen Zugriff per VBA, sie lassen sich also nicht direkt analysieren. Auch darin enthaltene Untermakros ("Benannte Makros", "Gruppenmakros") sind keine eigenen Objekte, die per Auflistung gefunden werden können.

Daher lässt es sich nicht vermeiden, alle Makros jeweils in eine Textdatei (in das relAcc-Unterverzeichnis \Ergebnisse\Access\) exportieren. Von dort aus werden sie wiederum zeilenweise eingelesen, um die darin enthaltenen Namen zu ermitteln.



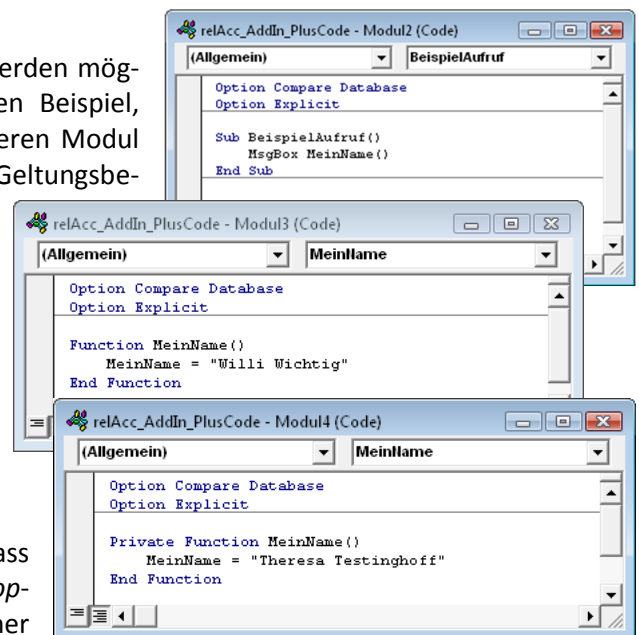
## Datenmakros

Datenmakros sind noch schlimmer. Sie lassen sich ebenfalls nicht per VBA analysieren und besitzen keine Namensauflistung, aber es gibt noch nicht einmal einen VBA-geeigneten Hinweis, ob überhaupt welche in der Tabelle enthalten sind. Außerdem entsteht bei deren Export mit dem gleichen *SaveAsText*-Befehl wie bei "normalen" Makros gar keine Text-Datei, sondern ein XML-Dokument. Dafür wäre wieder ein neuer eigener Parser notwendig. Wegen dieses erheblichen Aufwands prüft das AddIn keine Datenmakros.

## Geltungsbereich von Prozeduren

Gleichnamige Prozeduren in verschiedenen Modulen werden möglicherweise verwechselt. Ruft, wie im nebenstehenden Beispiel, eine Prozedur die Funktion *MeinName* aus einem anderen Modul auf, so ist die tatsächlich benutzte Funktion von deren Geltungsbereich abhängig.

Da es im *Modul4* eine *Private*-Funktion ist, würde zur Laufzeit die gleichnamige Funktion aus *Modul3* aufgerufen. Es wäre jedoch ein erheblicher Aufwand, diesen Geltungsbereich einer Deklaration zu ermitteln, so dass einfach der erste Treffer mit dem Namen dieser Funktion berücksichtigt wird.

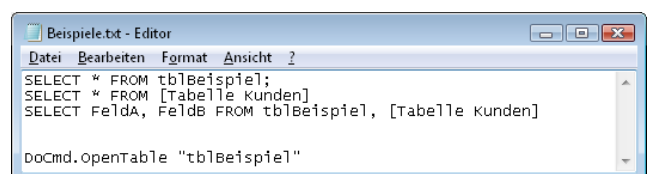


## Prozedur-Aufrufe erkennen

Prozeduren stehen nicht hinter Schlüsselwörtern, so dass hier die "Honigtopf"-Taktik versagt. Außer der *Do/Loop*-Schleife für die Suche nach Schlüsselwörtern gibt es daher eine anschließende Schleife, die gezielt nochmals alle Prozedurnamen untersucht.

## Objektnamen

Die gesuchten Objektnamen können ohne Begrenzer, in Gänsefüßchen oder in eckigen Klammern auftauchen, wie die nebenstehenden Beispiele zeigen. Eine solche abweichende Schreibweise würde aber deren zuverlässige Erkennung ausschließen. Alle gefundenen "Wörter" müssen daher mit einer Funktion gesäubert werden, so dass eventuelle Sonderzeichen entfernt sind.



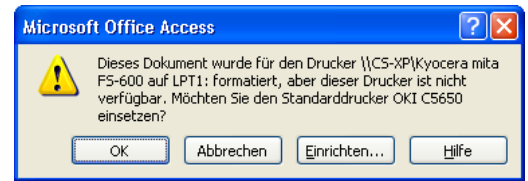
## Doppelte Relationen

Sowohl in SQL als auch im VBA-Code kann die gleiche Datenquelle (mit *Alias*) oder Prozedur mehrfach aufgerufen werden. Damit dadurch nicht überflüssige Relationen gespeichert werden, prüft die Prozedur zum Eintragen, dass diese Relation noch nicht vorhanden ist

## Fehlende Drucker

Ein auf diesem Rechner nicht installierter Drucker wird beim Öffnen des Berichts bemängelt. Das mag beim Drucken eines Berichts sinnvoll sein, gilt aber sogar in der Entwurfsansicht.

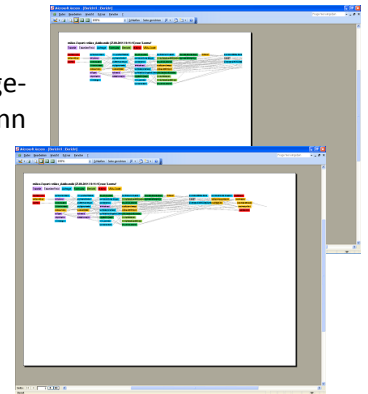
Ärgerlicherweise lässt sich diese Meldung nicht unterdrücken oder vorhersagen, sie erscheint immer. Daher gibt es hier im Code ein `On Error Resume Next`, damit wenigstens der `AddIn-Analysecode` weiterläuft, ansonsten würde ein Abbrechen den kompletten Code stoppen.



## Querformat

Das Ergebnis der Analyse wird von `relAcc` in einem querformatigen Bericht angezeigt, weil sehr viele Objekte nebeneinander gezeichnet werden. Offenbar kann Access aber einen solchen Bericht nicht korrekt darstellen, denn er wird zuerst fälschlich im Hochformat (oberes Bild) angezeigt.

Erst ein manueller Wechsel in die Entwurfsansicht und wieder zurück in die Vorschau (unteres Bild) stellt das Papier so querformatig dar, wie es in den Eigenschaften längst eingestellt ist. Ein per VBA-Code durchgeführter Wechsel zwischen Entwurfsansicht und Vorschau funktioniert nicht.



## Download

Ich stelle Euch natürlich alle gezeigten Dateien zum Download zur Verfügung. Der VBA-Code (auch der MSProject- oder Visio-Vorlagen-Makros) ist dabei offen und darf für eigene Anpassungen verändert werden.

## relAcc\_AddIn.mda

Diese Datei muss wie jedes AddIn in das AddIn-Verzeichnis kopiert werden. Sie enthält den kompletten Code und ist ab Access 2003 lauffähig.

## Vorlagen

Alle folgenden Dateien müssen jeweils in das `\Vorlagen\<Programm>`-Verzeichnis kopiert werden. Für deren Funktionsfähigkeit ist natürlich Voraussetzung, dass das zugehörige Programm auf dem Rechner installiert ist...

Datei	Bedeutung
<b>relAcc_Leer.mdb</b>	Diese leere Datei dient als Vorlage für eine Speicher-Datenbank der Analyse-Werte, die notwendigen Tabellen und Felder sind darin enthalten
<b>Visio-Vorlagen</b>	Diese Vorlagen enthalten Formatvorlagen (ohne Makros) und eine Symbolleiste zur Markierung (mit Makros) als Beispiel für die weitere Verarbeitung
<b>MSProject-Vorlage</b>	Diese Vorlage enthält zwei benutzerdefinierte Ansichten <i>Gantt/Beziehungsdiagramm</i> und <i>Netzplan</i>

## Fazit

Die Analyse von Access-Datenbanken ist mühsam, aber möglich. Es lassen sich nicht wirklich die allerletzten Zusammenhänge ermitteln, aber für den Überblick reicht es allemal. Eine grafische Ausgabe erleichtert das Lesen der Analyse-Daten erheblich, vor allem mit geeigneten Werkzeugen wie Visio.

Wer mag, darf den Code für eigene Zwecke verbessern. Die flexible Schnittstelle ermöglicht Euch dabei, eigene Grafik-Ausgaben zu verwirklichen. Manches (wie etwas Access-Projekte) ist in *relAcc* sicherlich arg stiefmütterlich behandelt worden, so dass hier ebenfalls ein weites Feld für Eure Anpassungen ist.

Lorenz Hölscher