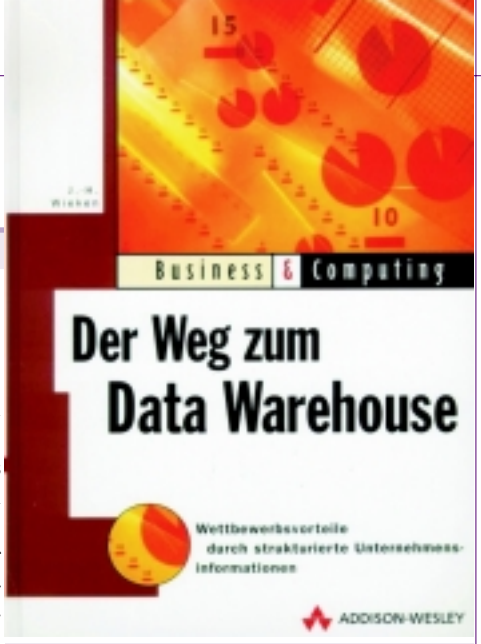
Der Weg zum Data Warehouse

Karel Štípek



J. H. Wieken, Der Weg zum Data Warehouse, Verlag Addison-Wesley, 1999, 262 Seiten, ISBN 3-8273-1560-3

Zuerst etwas Theorie

"Data Warehouse wird in allgemeinem verstanden als die Bereitstellung von Informationen für die Kontroll- und Entscheidungsprozesse in einem Unternehmen".

Soweit der erste Satz des Buches.

Ins Deutsche übersetzt bedeutet das etwa folgendes: Praktisch kein Unternehmen ist heute ohne den Einsatz der EDV-Technik denkbar. Verschiedene Computersysteme und Anwendungsprogramme erleichtern (manchmal auch erschweren, das wäre aber ein anderer Artikel) uns die Abwicklung der Arbeits- und Geschäftsvorgänge, wie Produktion, Ankauf, Verkauf, Lagerverwaltung, Personalwesen u.v.a.

Dabei werden große Mengen von Daten erfasst und gespeichert, die in der ersten Linie dem alltäglichen Geschäftsablauf dienen, sog. operative Systeme.

Die gespeicherten Informationen können aber noch einen anderen Zweck erfüllen. Sie spiegeln verschiedene Abhängigkeiten und Regeln ab, die für die Planungs- und Entscheidungsprozesse des Managements wichtig sind.

Komplexe Auswertungen aus vielen möglichen Blickwinkeln sind das eigentliche Ziel der Technologie, die als Data Warehouse bezeichnet wird.

Das Buch ist in folgende Hauptkapitel unterteilt:

1) Das Data-Warehouse-Konzept

Die vollkommene und analytisch optimale Lösung ist nicht immer realisierbar. Man beginnt nämlich bei dem Aufbau des Data Warehouses nicht "auf der grünen Wiese", sondern muß aus den schon vorhandenen operativen Systemen ausgehen und aus zeitlichen und finanziellen Gründen einige Kompromisse annehmen. Das Kapitel erklärt die Grundbegriffe, Ziele und die Unterschiede zwischen der 1-, 2- und 3-stufigen Data-Warehouse-Architektur.

2) Datenzugriff und Analyse (Front-End)

Die traditionelle Datenhaltung stellen die relationellen Datenbanken und SQL-Zugriffssprache dar. Es werden die Auswertungsmöglichkeiten mit Hilfe eines Managed Query Environments (MQE) diskutiert und anschließend die Grundideen eines OLAP - multidimensionalen Models vorgestellt - Dimensionen und Kennzahlen, Drill-down/Roll-up u.a. Ein neuer Begriff wird eingeführt - Data Mining. Damit wird die eigentliche Informationsgewinnung aus den gespeicherten Data-Warehouse-Daten bezeichnet.

3) Modellierung von OLAP-Lösungen

Ein OLAP-Modell sollte eigentlich vor dem Entwurf des Datenbankmodells erfolgen. Es enthält vor allem die Klassifikation von Dimensionen und die Auswahl von relevanten Kennzahlen.

4) Datenmodelle

Die Modellierung auf der Basis der relationellen Systeme wird extra für operative Systeme, Data-Warehouse und Data-Marts diskutiert. Den strengen Entity-Relationship Normalisierungsregeln wird die Erfordernis nach ausreichender Laufzeit-Schnelligkeit (Performance) gegenübergestellt, was manchmal zu einigen Konzepten der Denormalisierung, besonders bei Data-Marts führen kann.

5) Datenbereitstellung und Bereinigung

Es ist offensichtlich das größte Problem, Daten aus verschiedenen schon vorhandenen Systemen unter einen Hut zu bringen. Diese Aufgaben können bis zu 70 % des gesamten Aufwands für die Bereitstellung einer Data-Warehouse-Umgebung ausmachen. Der Vorgang wird als ETL bezeichnet, das stellt die drei Stufen dar: Extraktion, Transformation und Laden.

6) Metadatenmanagement

Metadaten sind "Daten über Daten", also alle Angaben die nicht den wirklichen fachlichen Inhalt tragen, sondern die Verwaltung des gesamten Systems ermöglichen. Dazu gehören z.B. Beschreibungen von Datenstrukturen, künstlich generierte Schlüssel, u.a. Ein Metadatenmodell wird vorgestellt, das auf einer zentralen Speicherung der Metadaten in einem sog. Repository aufgebaut wird.

7) Verfahren zum Aufbau eines Data Warehouse

Im letzten Kapitel werden die vorher vorgestellten Techniken zu konkreten Vorgangsvorschlägen zusammengefasst. Der Kernpunkt dabei sind die Phasen des Aufbaus eines Data-Marts.

Und jetzt etwas Praxis - Import von Excel-Blättern ins Access

Wenn man den Auftrag bekommt eine Lösung für komplexe Auswertungen zu entwickeln, ist man damit konfrontiert, dass es schon viele Teillösungen bei der Firma gibt. Praktisch jeder Anwender kennt sich mit Excel gut aus. Eine schöne Tabelle eventuell mit Graphen und Bildern zusammenzustellen ist die Frage von ein paar Stunden. Der Chef bekommt die wichtigen Zahlen, der Mitarbeiter das Geld und beide scheinen restlos zufrieden zu sein.

Der Wermutstropfen dabei ist, dass nicht einmal die Mitarbeiter, geschweige die Abteilungen so weit koordiniert vorgehen, dass wirklich ein systematisches Werk entsteht. Die Produkte werden zu Recht auch als "Insellösungen" bezeichnet.

Für die Bearbeitung von größeren Datenmengen und die Entwicklung einer komfortablen Bedienungsoberfläche ist Access viel besser geeignet als Excel. So kommt man einmal auf die Idee, die Excel-Blätter ins Acces zu importieren und weiter verarbeiten. Der Vorgang ist automatisiert und so kann man das Ergebnis bald sehen. Die Probleme fangen aber dann an, wenn man die Tabellen zu einem relationellen Datenmodell zusammenfassen will.

1 Es werden viele redundante Daten importiert. Zum Beispiel Namen- und Adressenangaben (lange Textfelder) müssen in jeder Teillösung, aber nur einmal in einer Access-Datenbank gespeichert werden.

2 Verschiedene Autoren verwenden verschiedene Kodierungssysteme - bei einem wird Zugang mit "Z" und Abgang mit "A" bezeichnet, bei anderem z.B. mit Zahlen 1 und 2.

3 Es ist sinnvoll, alle möglichen Datums- und Zeitformate in das richtige Date/Time Access-Format zu konvertieren.

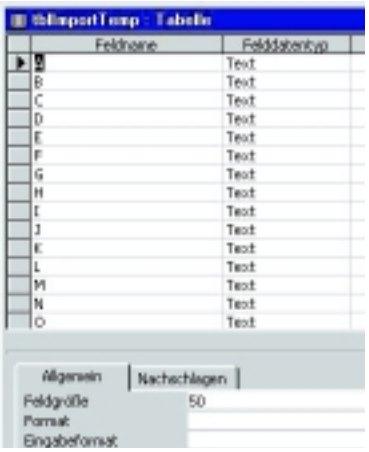
4 Eine Überprüfung der zulässigen Werte schon beim Import trägt positiv zu der Erhaltung der Datenkonsistenz bei.

5 Der Typ der Datenfelder sollte nicht automatisch sondern explizit definierbar sein.

6 Manchmal ist es sinnvoll statt mehreren Feldern ein berechnetes Ergebnis direkt zu speichern und so die Performance auf der Access-Seite zu verbessern.

Ich verwende beim Import vom Excel ins Access ein zweistufiges Verfahren, womit ich die o.g. Schwierigkeiten umgehen kann.

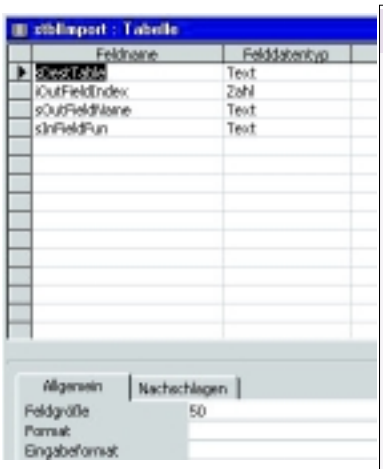
1 Das Excel-Blatt wird in eine temporäre Access-Tabelle **"tblImportTemp"** importiert, die alle Felder vom Typ Text hat. Die Namen der Felder sind ident mit den Bezeichnungen der Excel-Spalten, das heißt A, B, C, ... bis die maximal erwartete Spalte.



2 Die Access-Tabelle wird immer neu erstellt. Der dafür notwendige SQL-Ausdruck wird mit Hilfe der Definitionstabelle **"stblImport"** generiert, die für jedes Zielfeld einen Datensatz enthält.

Die Felder haben folgende Bedeutung:

sDestTable	Name der Zieltabelle
iOutFieldIndex	Ordnungsbegriff - bestimmt die Reihenfolge der Zielfelder
sOutFieldName	Name des Felds der Zieltabelle
sInFieldFun	Formel, wie aus den Feldern A, B, C, ... der Tabelle "tblImportTemp" das Zielfeld berechnet wird



Der Import eines Excel-Blattes wird mit der Funktion `ExcelToAccess` realisiert:

Mit der Möglichkeit, im Feld `sInFieldFun` einen Ausdruck einzugeben, kann man sich Vieles leisten.Einige Beispiele:

Einfache Übernahme der Textspalte B

```
B
```

Konversion der Spalte G in einen vordefinierten Access-Feldtyp (hier Zahl-Double)

```
Cdbl(G)
```

Einfügen einer führenden Null der Spalte P, wenn sie nur 5 Zeichen lang ist

```
Iif(Len(Trim(P))=5, '0'+P, P)
```

Konversion des Datums im Textformat JJMMTT auf Datums-Typ mit Berücksichtigung des Jahres 2000

```
DateSerial(CInt(Left$(R,2))+IIF(CInt(Left$(R,2)),2000,1900),Cint(Mid$(R,3,2)),Cint(Right$(R,2)))
```

Konversion des Datums im Excel-Format (Anzahl der Tage ab dem 1.1.1900) in Access-Datumstyp

```
DateSerial(1900,1,1)+CDbl(E)-2
```

Ein wichtiger Nebeneffekt des hier vorgestellten Konzepts ist auch die Tatsache, dass man automatisch ein Data-Repository zur Verfügung hat. Wenn Sie die Tabelle `stblImport` nach `sOutFieldName` sortieren, können Sie schnell finden, in welcher Tabelle sich das Feld befindet und wie es berechnet wurde.

```
Function ExcelToAccess(ImportSpec$, DestTable$, ExcelRange$, sWhere$)
'// IMPORTSPEC     Importspezifikation enspricht dem Feld sDestTable in der Tabelle stblImport
'// DESTTABLE   der Name der Zieltabelle (mehrere Tabellen können
'//             die gleiche Importspezifikation verwenden)
'// EXCELRANGE  der imporierte Bereich der Excel-Mappe
'//             (die Überschriften werden weggelassen)
'// SWHERE     ein möglicher Filter-Ausdruck

Dim res%
Dim xlsname$

ExcelToAccess = False
xlsname = Trim(GetFileName(desttable))  '// EXCEL-Datei suchen
If Len(xlsname) = 0 Then Exit Function  '// Abbrechen im Dialog gewählt

res = IntExcelImport(xlsname, importspec, desttable, ExcelRange, swhere)

MsgBox "Import der Tabelle " + UCase(desttable) + " beendet"
ExcelToAccess = res
End Function
```

Die Funktion `IntExcelImport()` besteht aus folgenden Schritten:

a) Löscht den Inhalt der temporären Tabelle und die Zieltabelle, falls vorhanden

```
Function IntExcelImport(xlsname$, importspec$, desttable$, ExcelRange$, swhere$)
Dim db As Database
Dim rec As Recordset
Dim qdf As QueryDef
Dim s$
IntExcelImport = False
Set db = CurrentDb()
DoCmd.RunSQL "DELETE * FROM tblImportTemp"
deltab db, desttable
```

b) Importiert den gewünschten Excel-Bereich in die temporäre Tabelle und überprüft, ob einige Datensätze übernommen wurden

```
DoCmd.TransferSpreadsheet A_IMPORT, 5, "tblImportTemp", xlsname, False, ExcelRange
Set rec = db.OpenRecordset("tblImportTemp", DB OPEN SNAPSHOT)
If rec.RecordCount = 0 Then
  MsgBox ("Keine Datensätze importiert")
  Exit Function
Else
  IntExcelImport = True
End If
rec.Close
```

c) Liest alle Datensätze für die gewünschte Importspezifikation aus der Tabelle `stblImport` und bildet den SQL-String.

```
Set rec = db.OpenRecordset("SELECT * FROM stblImport WHERE sDestTable = '" + importspec + "' ORDER BY iOutFieldIndex", DB_OPEN_SNAPSHOT)
If rec.EOF Then
  MsgBox "Keine Importspezifikation für die Tabelle " + desttable
  Exit Function
End If
s = "SELECT "
Do While Not rec.EOF
  s = s + rec!sInFieldFun + " AS [" + rec!sOutFieldName + "],"
  rec.MoveNext
Loop
rec.Close
s = Left$(s, Len(s) - 1)      '// letzte Komma weg
s = s + " INTO " + desttable
s = s + " FROM tblImportTemp"
If Len(swhere)  0 Then
  s = s + " WHERE " + swhere
End If
```

d) Weist den SQL-String einer temporären Abfrage zu und führt sie aus

```
Set qdf = db.QueryDefs("qryTemp")
qdf.SQL = s
qdf.Close
qdf.Execute
db.Close
End Function
```