Entwurf und Implementierung einer räumlichen Datenbank für ATKIS-Daten mit Datenimport

Studienarbeit von
Stephan Falke

Aufgabenstellung und Betreuung:
Prof. Dr. Udo Lipeck
Dipl.-Math. Carsten Kleiner

Institut für Informatik B:
Datenbanken und Informationssysteme
Universität Hannover

Hannover, den 7. April 2000

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Vermessung der Erde und die Erfassung der gewonnenen Daten reicht sehr weit zurück. Verbunden damit ist das Problem, die gemessenen oder beobachteten Daten in geeigneter Weise aufzubewahren (zu speichern) und darzustellen, etwa durch Landkarten. Die enormen Speicherkapazitäten der Computer ermöglichen es, sehr viele Daten aufzunehmen. In Deutschland werden solche geodätische Daten in einzelnen Bundesländern durch deren Vermessungs- und Katasterämter aufgenommen und verwaltet. Um einheitliche Daten aufzunehmen und diese einheitlich zu speichern, wurde durch die Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland AdV bereits 1979 das Informationssystem Automatisierte Liegenschaftskarte ALK eingeführt.

Im Hinblick auf die Nutzung von Computern bei der Speicherung dieser Daten wurde 1989 von der AdV das Amtliche Topographisch-Kartographische Informationssystem ATKIS [Lan95] beschlossen. Die Landschaft wird erfaßt, indem einzelne Objekte gespeichert werden. Diese können punkt-, linien- oder flächenförmig sein und durch Attribute genauer beschrieben werden. Es gibt eine hierachische Klassenbildung der Objekte. Ziel des Systems ist es, die Landschaftsmodelle zu speichern. Es sollen drei verscheidene Digitale Landschaftsmodelle (DLM) aufgenommen werden, die in unterschiedlicher Genauigkeit die Landschaft wiedergeben. Beschlossen wurden zwei wesentliche Dinge: Zum einem gibt es Kataloge der Objekte und der Objektattribute und zum anderen eine Beschreibung dessen, welche Objekte in welches DLM aufgenommen werden. Um die Informationen einheitlich darzustellen, wurde ferner ein Signaturenkatalog verabschiedet, in dem zu jedem Objekt beschrieben wird, wie dieses darzustellen sei.

Um einen Datenaustausch zu ermöglichen ist ein Datenformat vereinbart worden, dieses ist die Einheitliche Datenbankschnittstelle (EDBS). In diesem Format werden die gesammelten Daten gespeichert. Jedoch ist dieses Format aus Gründen der Speicherverwaltung nicht zur direkten Bearbeitung geeignet. Um mit solch großen Datenmengen sinnvoll zu arbeiten, werden Datenbanken [RRZ99] benutzt; siehe dazu die Arbeiten von Koch [Koc94] [NK95] und Korte [Kor97].

Das Anliegen dieser Studienarbeit ist es, eine Datenbank für die Daten der

DLM zu entwerfen. Dabei soll die Neuerung der Oracle Version 8.1.5 (Oracle8*i*) benutzt werden, die es erlaubt, räumliche Daten in voreingestellten Typen zu speichern, und deshalb durch bessere Verwaltung der Daten einen sehr viel schnelleren Zugriff ermöglicht [Cor99a]. Ferner sollen alle Informationen der EDBS Dateien in die Datenbank übernommen werden, so daß alle Nutzer der DLM diese Datenbank nutzen können, um mit den geographischen Daten zu arbeiten. Insgesamt soll die Verwaltung geographischer Daten effizienter werden.

Sowohl die Datenbank als auch das Programm zum Erfassen der EDBS-Daten wurde so geschrieben, daß Erweiterungen und Aktualisierungen der ATKIS-Kataloge oder der DLM einfach einzubinden sind.

Im Kapitel 2.1 wird auf das ATKIS Datenmodell eingegangen. Wie die Daten dieses Modells gespeichert werden, wird im Kaptitel 2.2 durch die Beschreibung des EDBS Formats gezeigt. Die entworfene Datenbank wird im Kapitel 4 erläutert.

Das Einlesen der Daten geschieht in drei Schritten, nämlich dem Lesen der EDBS-Dateien, der Aufbereitung dieser Daten und der endgültigen Speicherung in der Datenbank. Die Programme für diese Teilschritte werden in den Kapiteln 3 und 5 beschrieben. Der prinzipielle Ablauf wird in Abbildung 1.1 schematisch dargestellt.

Erläuterungen zur Bedienung dieser Programme folgen im Kapitel 6.

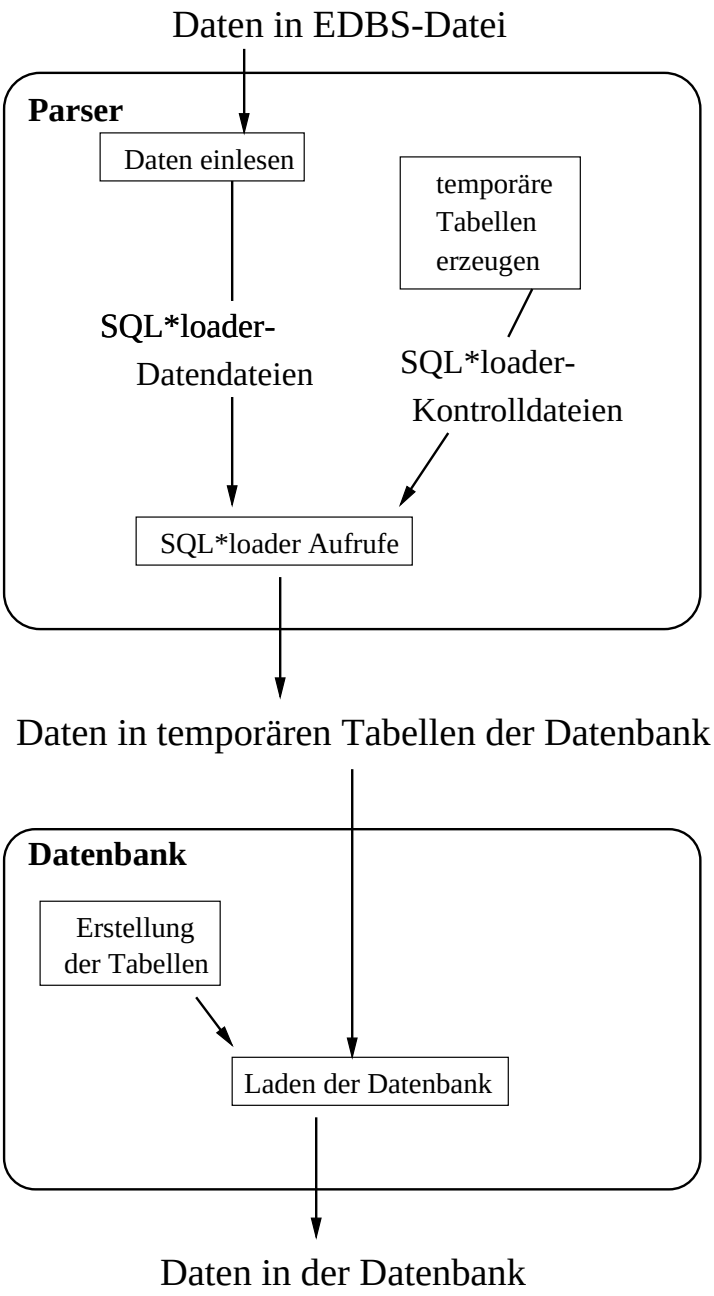

Abbildung 1.1: Ablaufdiagramm

Kapitel 2

ATKIS Datenmodell

Das ATKIS Datenmodell ist Teil eines Projekts, daß versucht, die verschiedenen Datenmodelle der Landesvermessungsämter zu einem gemeinsamen Modell zusammenzuführen. Es sollen die bereits erhobenen Daten in dieses Modell eingefügt werden können. Das Modell orientiert sich daher an schon vorhandenen Modellen.

2.1 Struktur der ATKIS-Daten

Um eine Landschaft in das Modell aufzunehmen, wird diese in einzelne Objekte eingeteilt. Die Einteilung ist so, daß jedes Objekt eine bestimmte Funktion besitzt. Jedes Objekt ist von einer bestimmten Objektart. Die Strukturierung der Daten wird somit der Strukturierung der Landschaft nachempfunden. Insbesondere werden keine Objekte gebildet, die ein bestimmtes Planquadrat einer Karte sind.

Zu jedem Objekt gibt es topographische und inhaltliche Informationen. Topographisch lassen sich die Objekte in punkt-, linien- und flächenförmige Objekte einteilen (Objekttyp). Ferner gibt es komplexe Objekte, die keine topographische Beschreibung haben, sondern aus mehreren anderen Objekten bestehen. Diese komplexen Objekte fassen Objekte zusammen, die eigentlich zusammengehören, aber aufgrund der Objektbildungsregeln als einzelne Objekte gespeichert werden.

Ein weiterer Objekttyp sind die rasterförmigen Objekte, die jedoch in den Beispieldaten nicht vorkommen. Daher wird sowohl hier als auch in den Programmen nicht auf diesen Objekttyp eingegangen.

Zu den unterschiedlichen Objekttypen werden folgende topographische Informationen angegeben:

- Bei punktförmigen Objekten werden die Koordinaten des Punktes angegeben,
- linienförmige Objekte setzen sich aus einer oder mehreren zusammenhän-

genden Linien zusammen, für die jeweils ein Anfangs- und ein Endpunkt sowie optional mehrere Stützpunkte (Lageparameter) angegeben werden,

- flächenförmige Objekte werden als geschlossener Linienzug beschrieben, der sich aus ein oder mehreren Einzellinien zusammensetzt.

Um diese topographischen Informationen der Objekte zu speichern, werden die Objekte nach festgelegten Regeln in Objektteile geteilt. Häufig besteht ein Objekt aus nur einem Objektteil; bei punktförmigen Objekten ist dies immer der Fall. Die Objektteile erhalten eine objektweit eindeutige Objektteilnummer.

Die inhaltlichen Informationen zu einem Objekt werden gespeichert, indem jedes Objekt zu einer Objektart zugeordnet wird. Für jedes DLM gibt es in der ATKIS Dokumentation einen Objektartenkatalog. In diesem wird festgelegt, welche Objektarten es gibt und welche Objekte für das Modell aufzunehmen sind. Alle Objektarten sind mit Beschreibung und einer eindeutigen Nummer aufgeführt. Die verschiedenen Objektarten sind in Objektgruppen zusammengefasst. Die Objektgruppen werden in neun Objektbereiche eingeteilt. Die stets vierstelligen Nummern der Objektarten geben diese hierachische Struktur wieder: An der ersten Ziffer erkennt man den Objektbereich, an den ersten beiden Ziffern die Objektgruppe.

Ein Beispiel: 3 steht für den Objektbereich *Verkehr*, 31 steht für die Objektgruppe *Straßenverkehr*. Die Objektarten in dieser Objektgruppe sind

3101 *Straße*,
3102 *Weg*,
3103 *Platz*,
3104 *Straße (komplex)*,
3105 *Straßenkörper*,
3106 *Fahrbahn*.

Weitere Informationen zu einem Objekt können über Objektattribute gespeichert werden. Die zu einer Objektart möglichen Attributtypen sind im Objektartenkatalog festgelegt und werden mit einer eindeutigen Nummer ausgezeichnet. Auch die zu dem Attribut möglichen Attributwerte sind dort festgelegt und mit einer Nummer ausgezeichnet. Es gibt Attributtypen, zu denen der Attributwert direkt angegeben werden kann, etwa die Breite einer Straße.

Die Objekteile können ebenfalls solche Attribute erhalten. Attribute eines Objekts gelten für alle Objektteile, umgekehrt gilt ein Attribut eines Objektteils jedoch nicht für das gesamte Objekt.

Jedes Objekt wird nicht nur einer Objektart, sondern auch einer Folie zugeordnet, in der die Objekte nach Ihrer Funktion eingeteilt sind [Wal97] [Har94]. Durch die Zuordnung zu einer Folie soll die fachliche Gruppierung der Daten erreicht werden. Beispiele für Folien sind *Schienenverkehr*, *Wasserflächen* und *politische Grenzen*.

2.2 EDBS-Format

Um Daten speichern oder übertragen zu können, wird in der ATKIS Dokumentation das EDBS-Format festgelegt. Die Daten werden in Textdateien gespeichert. Jede Zeile der Datei ist ein EDBS-Satz. Diese Sätze haben immer den gleichen Aufbau:

Parameter	**Länge**
Satzanfang	4
Satzlänge	4
Suchkriterium	4
0perationsschlüssel	4
Quittungs- und Editierschlüssel	12
Name der Information	8
Inhalt der Information	variabel

Ein EDBS-Satz, also jede Zeile der Datendatei, beginnt mit der Zeichenfolge `EDBS`. Die Länge eines Datensatzes (einer Zeile) ist auf 2000 Zeichen begrenzt. Die hier relevanten Zeilen der Datei sind die Grundrißdateien und die Attributdateien, für beide ist der Operationsschlüssel `BSPE`. Grundrißdateien enthalten Informationen zu Funktionen und Geometrien von ATKIS-Objekten, als Name der Information ist `ULOBNN␣␣` eingetragen. In Attributdateien können die Attribute zu Objekten oder Objektteilen zugeordnet werden, hier ist als Name der Information `ULTANN␣␣` eingetragen. Die anderen Angaben am Anfang eines EDBS-Satzes spielen keine Rolle.

Am Anfang einer EDBS-Datei werden zudem in EDBS-Sätzen Angaben über die Datei gemacht, die hier jedoch nicht von Interesse sind, da nur die geographische Information gelesen und bearbeitet wird.

Ein Beispiel zeigt, wie ein EDBS-Satz in Zeichengruppen einzuteilen ist:

$$\underbrace{\texttt{EDBS}}_{Satzanfang}\ \underbrace{\texttt{0049}}_{Satzlänge}\ \underbrace{\texttt{0000}}_{Suchkriterium}\ \underbrace{\texttt{BSPE}}_{Operationsschlüssel}$$

$$\underbrace{\texttt{0000620000}}_{Quittierungs-\ und\ Editierschlüssel}\ \underbrace{\texttt{ULTANN␣␣}}_{Name\ der\ Information}\ \underbrace{\texttt{00010001N016VGF␣␣␣7010000}}_{Inhalt\ der\ Information}$$

Dieses Dateiformat ist so aufgebaut, daß die Informationen zu einem Objekt über viele Sätze der Datei verteilt sind; die Attribute zu einem Objekt, deren topologische Informationen in einer Zeile beschrieben werden, muß in einer anderen Zeile stehen. Dies ist darin begründet, daß das Datenformat auf redundanzfreie Speicherung Wert legt. Da dennoch die EDBS-Dateien schnell einige Megabyte groß sind, ist es ein Problem, die Informationen zu einem Objekt zusammenzusuchen. Eine Verbesserung der Speicherverwaltung soll dieses Projekt aufzeigen. Dabei wird ein effizienterer und strukturierterer Speicherzugriff bei größerem Speicherbedarf erreicht.

2.2.1 Grundrißdatei

Eine Grundrißdatei ist ein EDBS-Satz, bei dem `ULOBNN␣␣` für den Namen der Information eingetragen ist. Der Inhalt der Information wird nach folgender Grammatik beschrieben. Die Einträge in den geschweiften Klammern kommen dabei genauso häufig vor, wie in dem Wiederholungsfaktor `<WHF>` vor ihnen angegeben ist. Jeder Wiederholungsfaktor ist genau vier Zeichen lang, führende Nullen werden aufgefüllt. Auch ein Wiederholungsfaktor von null (`0000`) ist möglich.

```
<GRUNDRISSKENNZEICHEN>  →  00010001 <Grundrisskennzeichen>
                           <WHF> { <LINIE> }
                           <WHF> { <OBJEKT> }
            <LINIE>     →  0001 <Endpunkt der Linie>
                           <WHF> { <LINIENFUNKTION> }
                           <WHF> { <Lageparameter> }
           <OBJEKT>     →  0001 <Funktion des Objekts>
                           <WHF> { <Besondere Information>
                                   <WHF> { <Geometrieangaben> } }
    <LINIENFUNKTION>    →  0001 <Funktion der Linie>
                           <WHF> { <Fachparameter> }
```

Neben der Datengruppe `<Grundrisskennzeichen>` kommt stets mindestens eine der Datengruppen `<LINIE>` oder `<OBJEKT>` vor. Bei einer `<LINIE>` wird mindestens ein `<Endpunkt der Linie>` angegeben.

Das EDBS-Format ist deshalb so unüberschaubar, weil die Informationen nicht objektorientiert gespeichert werden. Es soll vielmehr redundante Speicherung von Daten vermieden werden. So werden die topologischen Informationen zu einer Linie einmal gespeichert und dann zusätzlich alle Funktionen dieser Linie gespeichert. Jede dieser Funktionen ist wiederum einem Objekt zugeordnet. Eine Linie kann beispielsweise den Verlauf einer Straße beschreiben, ein Teil der Grenzlinie zweier Felder sein und zusätzlich eine politische Grenze beschreiben. Die Anzahl der Funktionen wird im Wiederholungsfaktor für die `<LINIENFUNKTION>` angegeben.

Der Bezugspunkt von Objekten wird im `<Grundrisskennzeichen>` in der Grundrißdatei, in der die `<Funktion des Objekts>` gespeichert ist, angegeben. Linien verlaufen von diesem Punkt des `<Grundrisskennzeichens>` zu dem `<Endpunkt der Linie>`. Ist die Liniengeometrie des entsprechenden `<Endpunktes der Linie>` keine Gerade, so könen in der Datengruppe `<Lageparameter>` weitere Punkte angegeben werden, die den Verlauf der Linie beschreiben; dadurch kann zum Beispiel ein Polygonzug beschrieben werden. Der Wiederholungsfaktor der `<Lageparameter>` gibt an, wieviele dieser Punkte angegeben werden.

In der Datengruppe `<Fachparameter>` können weitere Angaben zu der Linie gemacht werden, etwa Angaben zu Höhen von Linienpunkten. Dies kommt jedoch

in den Beispieldaten nicht vor und wird auch nicht in die Datenbank aufgenommen.

2.2.1.1 Grundrißkennzeichen

Im wesentlichen wird in der Datengruppe `<Grundrisskennzeichen>` eine Koordinate gespeichert. Dieser Punkt ist Bezugspunkt der Objekte und Anfangspunkt der Linien, die in der gleichen Grundrißdatei aufgeführt werden. Im Einzelnen sind die eingetragenen Daten die folgenden:

Datenelement	Länge
Numerierungsbezirk NBZ	8
Koordinaten im NBZ	12
Prüfzeichen	1

Die Punkte werden im Gauß-Krüger-Koordinatensystem angegeben. Der NBZ enspricht einer Fläche, die von einem Gitter mit fester Gitterlänge, die davon abhängt welches DLM beschrieben wird, gebildet wird. Die angegebenen Koordinaten bestimmen die Lage der Punkte in diesem NBZ in Bezug auf den südwestlichen Gitterschnittpunkt. (Auf einer genordeten Karte also unten links.)

Bei dem hier behandelten DLM 25, also für Karten im Maßstab 1:25000, ist der Gitterabstand 10 km. Für jede der beiden Dimensionen stehen sechs Dezimalen zur Angabe der Koordinaten in dem NBZ zur Verfügung. Die Lage eines Punktes kann also mit einer Genauigkeit von einem Zentimeter angegeben werden.

Die Stellen eins, zwei und fünf des NBZ bilden die x-Koordinate des Referenzpunktes, von dem aus die Koordinaten gemessen werden; die Stellen drei, vier und sieben bilden die y-Koordinate. Die ersten sechs Stellen der Koordinaten bilden die x-Koordinate im NBZ, die anderen sechs die y-Koordinate. Man kann also die absolute Koordinate eines Punktes im DLM aus den Ziffern zusammensetzen.

Ein Beispiel dazu:

$$\underbrace{35572\_9\_}_{NBZ}\underbrace{375649707253}_{Koordinate}$$

Hierdurch wird der Punkt $(352375649, 579707253)$ beschrieben.

Das Prüfzeichen wird nicht verarbeitet.

2.2.1.2 Funktion des Objekts

Neben Liniengeometrien können zu einem Referenzpunkt auch mehrere Objekte definiert werden. In der Datengruppe `<Funktion des Objekts>` wird ein Objekt mit seiner eindeutigen Objektnummer definiert und seine inhaltlichen Angaben gemacht. Der Referenzpunkt der Objekte ist der in der Datengruppe `<Grundrisskennzeichen>` der Grundrißdatei angegebene Punkt (siehe Abschnitt 2.2.1.1).

Die Informationen sind im einzelnen:

Datenelement	Länge
Folie	3
Objektart	4
Aktualität	2
Objekttyp	1
Objektnummer	7
Modelltyp	2
Entstehungsdatum	6
Veränderungskennung	1

Die Foliennummer und Objektartnummer sind im ATKIS-Objektartenkatalog nachgewiesen. Aus dem Objekttyp geht hervor, um was für ein Objekt es sich handelt. Dafür sind folgende Einträge möglich:

- P punktförmiges Objekt
- L linienförmiges Objekt
- F flächenförmiges Objekt
- K komplexes Objekt
- R rasterförmiges Objekt (nicht behandelt)

Das Entstehungsdatum wird bei einer Aktualisierung des Objekts auf das Veränderungsdatum gesetzt. Die sechs angegebenen Ziffern sind im einem Format, das -in dieser Reihenfolge- Jahreszahl, Monatszahl und Tag je zwei Ziffern einräumt.

Auf andere Daten wird nicht eingegangen. Sie dienen der Verwaltung von Daten in verschiedenen Modellen oder Datensammlungen.

2.2.1.3 Besondere Information

In der Datengruppe `<Besondere Information>` werden verschiedene Informationen zu einem Objekt nachgewiesen.

Die Datengruppe gliedert sich wie folgt:

Datenelement	Länge
Art der Information	2
Kartentyp	2
Signaturteilnummer	6
Text	33
Art der Geometrieangabe	2
Objektteilnummer	3

Abhängig von der Art der Information hat der Texteintrag eine bestimmte Bedeutung, die Möglichkeiten sind die folgenden:

Art der Information	Bedeutung des Textes
41	*hierarchische Referenz* auf ein übergeordnetes Objekt Stellen 1 - 7: Referenz (Objektnummer linksbündig)
42	*hierarchische Referenz* auf ein untergeordnetes Objekt Stellen 1 - 7: Referenz (Objektnummer linksbündig)
44	*Name* Stellen 1,2: Namenstyp GN: geographischer Name KN: Klassifizierungsname ZN: Zweitname
46	*Überführung* Stellen 1 bis 10: Objektnummer und Objektteilnummer des überführenden Objekts Stellen 11 bis 20: Objektnummer und Objektteilnummer des überführenden Objekts
47	*Unterführung* Stellen 1 bis 10: Objektnummer und Objektteilnummer des unterführenden Objekts Stellen 11 bis 20: Objektnummer und Objektteilnummer des unterführenten Objekts
48	*Objektteileintrag*

2.2.1.4 Geometrieangabe

Zu jeder `<Besondere Information>`, die einen Namen beschreibt, können einige Datengruppen `<Geometrieangabe>` auftreten. In dieser werden die Koordinaten angegeben, an denen der entsprechende Name in einer Karte stehen soll. Es ist möglich, für Objektnamen mehreren Stellen anzugeben. Dies ist etwa bei langen linienförmigen Objekten, wie zum Beispiel Autobahnen, sinnvoll.

Die Koordinatenangaben erfolgen wie beim `<Grundrisskennzeichen>` (siehe Abschnitt 2.2.1.1), daher sind auch die Daten wie folgt angegeben:

Datenelement	Länge
Numerierungsbezirk NBZ	8
Koordinaten im NBZ	12

2.2.1.5 Endpunkt der Linie

In der Datengruppe `<Endpunkt der Linie>` wird beschrieben, zu welchem Punkt eine Linie verläuft, die vom Punkt ausgeht, der in der Datengruppe `<Grundrißkennzeichen>` (siehe Abschnitt 2.2.1.1) der Grundrißdatei, in der sie steht, angegeben wurde.

Zu jedem Ausgangspunkt können so viele Endpunkte angegeben werden, wie es die maximale EDBS-Satzlänge erlaubt.

Im einzelnen sind die Daten wie folgt eingeteilt:

Datenelement	Länge
Numerierungsbezirk NBZ	8
Koordinaten im NBZ	12
Art der Liniengeometrie	2

Die Koordinaten des Endpunktes ergeben sich aus NBZ und Koordinaten ebenso wie in der Datengruppe `<Grundrißkennzeichen>` angegeben.

Die Art der Liniengeometrie bestimmt, ob die Datengruppe `<Lageparameter>` vorhanden ist, oder ob der entsprechende Wiederholungsfaktor null ist.

Ist der Wert der Liniengeometrie 15, so bedeutet dies, daß ein Polygonzug vom Linienanfangspunkt zum hier angegebenen Endpunkt gebildet wird. In der Datengruppe `<Lageparameter>` werden die Stützpunkte angegeben.

Der Wert 11 als Art der Liniengeometrie kennzeichnet eine Gerade, bei der keine `<Lageparameter>` angegeben werden.

Es sind auch weitere Arten von Liniengeometrien möglich, zum Beispiel Splines. In den Beispieldaten kommen diese jedoch nicht vor.

2.2.1.6 Funktion der Linie

Zu jeder topologischen Beschreibung einer Linie können einige -wieder beschränkt durch die maximale EDBS-Satzlänge- Linienfunktionen angegeben werden.

In der Datengruppe `<Funktion der Linie>` werden dazu die Objektteile angegeben, deren Topologie durch die Linie beschrieben wird. Die Folie und die Objektart des Objekts, dessen Objektteil beschrieben wird, muß mit der Folie und der Linienart übereinstimmen.

Die angegebenen Daten sind im einzelnen:

Datenelement	Länge
Folie	3
Linienart	4
Objektnummer rechts	7
Objektnummer links	7
Objektteilnummer rechts	3
Objektteilnummer links	3
Linienteilung rechts	1
Linienteilung links	1

Welche Bedeutung die Nummern von Folie und Linienart haben, kann im Objektartenkatalog nachgesehen werden. Die Unterteilung in rechts und links gilt für der Linie zugeordnete flächenförmige Objekte gleicher Objektart, wobei sich

die Lage der Objekte zur Linie auf den Ausgangspunkt der Linie bezieht. Wenn sich die Folie oder die Objektart des linken oder rechten Objektteils unterscheiden, werden sie auf zwei Datengruppen aufgeteilt. Die Datenelemente des linken bzw. des rechten Objektteils werden nicht belegt.

Bei linienförmigen Objekten hat diese links/rechts-Zuordnung keine Bedeutung. Es werden nur die Datenelemente eines Objektteils belegt.

2.2.1.7 Lageparameter

In der Datengruppe `<Lageparameter>` wird je ein Punkt eines Polygonzuges angegeben. Auch bei anderen komplizierteren Arten von Liniengeometrien werden bestimmte Lagaparameter angegeben. Diese komplizierten Geometrien sind jedoch nicht in den Beispielen vorhanden und werden deshalb nicht behandelt.

Die Daten teilen sich wie folgt auf:

Datenelement	Länge
Numerierungsbezirk NBZ	8
Koordinaten im NBZ	12

Die Koordinaten der Stützpunkte werden wie in der Datengruppe `<Grundrisskennzeichen>` angegeben (siehe Abschnitt 2.2.1.1). Die Stützpunkte werden nacheinander vom Anfangspunkt des Polygonzuges zum Endpunkt angegeben.

2.2.1.8 Fachparameter

Weitere Angaben zu der Topologie einer Linie können in der Datengruppe `<Fachparameter>` gemacht werden. In den vorhandenen Dateien wird aber kein Gebrauch davon gemacht. Die Daten sind im einzelnen:

Datenelement	Länge
Art	1
Kennung	1
Wert	7

2.2.1.9 Bespiel einer Grundrißdatei

Ein Beispiel für eine Grundrißdatei ist der folgende EDBS-Satz:

```
EDBS03740000BSPE000062␣␣0000ULOBNN␣␣0001000135572␣9␣385213718
66040002000135572␣9␣3875397251361100020001101211 1N016VK8␣␣␣␣␣
␣␣001␣␣␣0␣000000011012111␣␣␣␣␣␣␣N016VK3␣␣␣001␣000000000000135
572␣9␣3898697184071500030001101211 1␣␣␣␣␣␣␣N016VK8␣␣␣001␣00000
00011012111N016VK3␣␣␣␣␣␣␣001␣␣␣0␣000000011043101N019077␣␣␣␣␣␣
␣001␣␣␣0␣0000000335572␣9␣38705271862835572␣9␣3881427185813557
2␣9␣3887077185370000
```

In Tabelle 2.1 werden die Datengruppen dieses EDBS-Satzes erläutert. Die in der Zeile enthaltene Information ist in Abbildung 2.1 skizziert, ein genauer Maßstab ist nicht eingehalten.

Die Grundrißdatei beschreibt zwei Linien mit gemeinsamem Anfangspunkt. Eine der Linien ist ein Polygonzug, der drei Stützpunkte hat. Zu jeder Linie ist beschrieben, welche Objekte links und rechts von ihr liegen. Die Angabe erfolgt durch Objektnummern. Die zweite Linie ist zudem eine Straße.

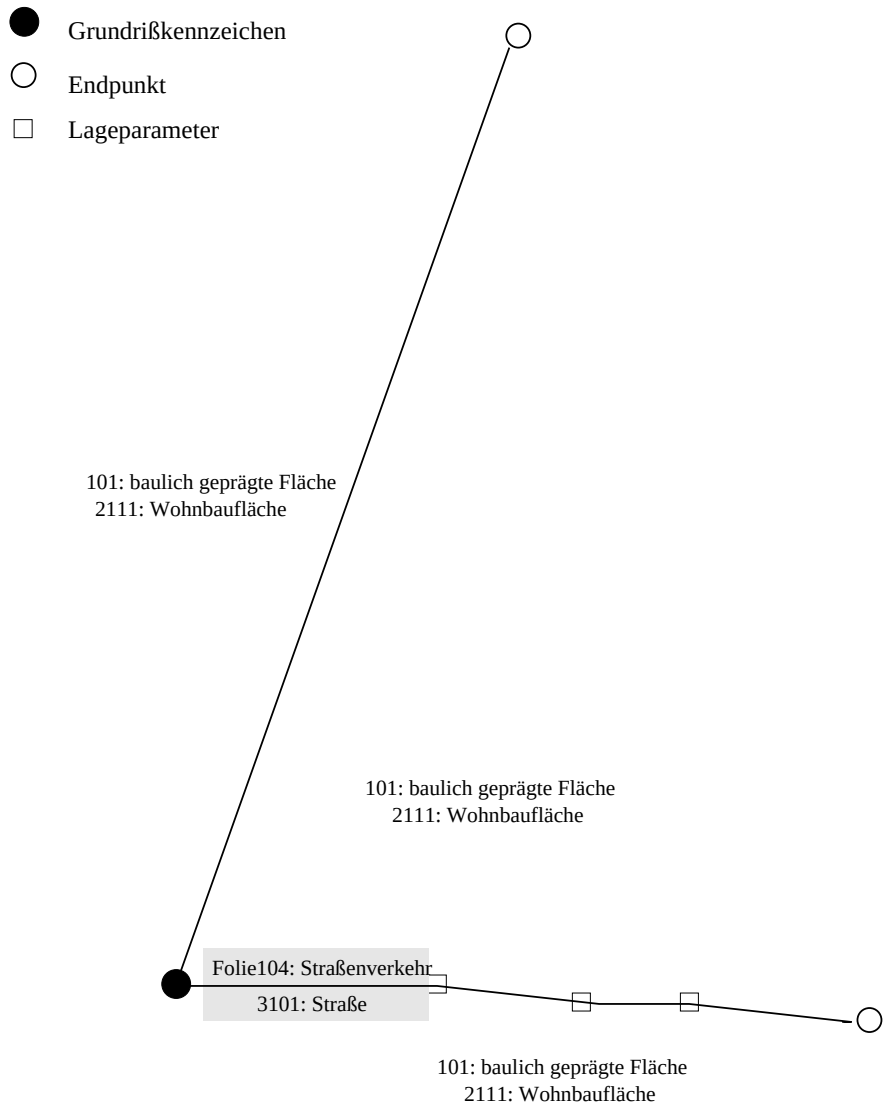

Abbildung 2.1: Beispiel einer Grundrißdatei

EDBS03740000BSPE000062␣␣0000	
	Zeilenkopf
ULOBNN␣␣	Grundrißdatei
0001	immer 0001
0001	immer 0001
35572␣9␣385213718660	Numerierungsbezirk, Koordinaten
4	Prüfzeichen
0002	WHF Enpunkt der Linie
0001	immer 0001, Beginn Linie 1
35572␣9␣387539725136	Koordinaten Endpunkt 1
11	Art der Geometrie (11: Gerade)
0002	WHF Funktion der Linie
0001	immer 0001, Beginn Funktion 1
101	Folie (101: Baulich geprägte Fläche)
2111	Objektart (2111: Wohnbaufläche
N016VK8	Objektnummer links
␣␣␣␣␣␣␣	Objektnummer rechts (kein Objekt)
001	Objektteilnummer links
␣␣␣	Objektteilnummer rechts
0	Linienteilung links
␣	Linienteilung rechts
0000	WHF Fachparameter
0001	immer 0001, Beginn Funktion 2
1012111␣␣␣␣␣␣␣N016VK3␣␣␣001␣00000	
	Funktion 2
0000	Anzahl der Lageparameter
0001	Beginn Endpunkt 2
35572␣9␣389869718407	Koordinaten Endpunkt 2
15	Art der Geometrie (15: Polygonzug)
0003	Anzahl der Funktionen
00011012111␣␣␣␣␣␣␣N016VK8␣␣␣001␣00000	
	Funktion 1
00011012111N016VK3␣␣␣␣␣␣␣001␣␣␣0␣0000	
	Funktion 2
00011043101N019077␣␣␣␣␣␣␣001␣␣␣0␣0000	
	Funktion 3
0003	Anzahl der Lageparameter
35572␣9␣387052718628	Lageparameter 1
35572␣9␣388142718581	Lageparameter 2
35572␣9␣388707718537	Lageparameter 3
0000	WDF Funktion des Objekts

Tabelle 2.1: Beispiel einer Grundrißdatei

2.2.2 Attributdatei

Ein EDBS-Satz, der eine Attributdatei ist, hat, wie bereits einführend erwähnt, als Name der Information `ULTANN␣␣` eingetragen. Eine Attributdatei ist jedoch sehr viel einfacher als eine Grundrißdatei, was schon die einfache Grammatik für den Inhalt der Information zeigt. Die Attributdateien werden gemäß folgender Grammatik gebildet:

```
<ATTRIBUTEINHEIT>  →  00010001 <Attributkennzeichen>
                      <WHF> { <Attribut> }
```

Durch den Wiederholungsfaktor `<WHF>` wird angegeben, wieviele Attribute dem Objekt oder Objektteil, das im Attributkennzeichen spezifiziert wurde, zugeordnet werden.

2.2.2.1 Attributkennzeichen

Die Datengruppe `<Attributkennzeichen>` setzt sich wie folgt zusammen:

Datenelement	Länge
Objektnummer	7
Objektteilnummer	3
Prüfzeichen	1
Aktualität	2

Durch die Angabe der Objektnummer und einer Objektteilnummer erfolgt die Zuordnung der Attribute zu den Objektteilen. Wird die Objektteilnummer leer gelassen, sind also drei Leerzeichen eingetragen, so werden die Attribute dem Objekt mit der angegebenen Nummer zugeordnet. Prüfzeichen und Aktualität werden nicht ausgewertet.

2.2.2.2 Attribut

In der Datengruppe Attribut werden Attributtyp und der Wert dieses Attributs angegeben. Die Datengruppe enthält nur diese beiden Informationen:

Datenelement	Länge
Attributtyp	4
Attributwert	7

Welche Attribute sich auf welche Objektarten beziehen können, ist ebenso im ATKIS-Objektartenkatalog aufgeführt, wie die möglichen Attributwerte von bestimmten Attributen.

In Tabelle 2.2 sind exemplarisch alle Attribute der Objektart *Straße* aufgeführt.

```
BDI Verkehrsbedeutung innerörtlich
   1000 Durchgangsverkehr
   2000 Ortsverkehr
   2001 Sammelverkehr
   2002 Anliegerverkehr
   9997 Attribut trifft nicht zu

BDU Verkehrsbedeutung überörtlich
   1000 überörtlicher Verkehr
   1001 Fernverkehr
   1002 Regionalverkehr
   1003 Nahverkehr, zwischenörtlicher Verkehr
   9997 Attribut trifft nicht zu

BFS Besondere Fahrstreifen
   1000 mit Radweg
   2000 mit Fußweg
   3000 mit Rad- und Fußweg
   9997 Attribut trifft nicht zu

BRF Breite der Fahrbahn
---- tatsächlicher Wert, Angabe auf 0,5 m in dm

BRV Breite des Verkehrsweges
---- Klassenangabe entsprechend Seite 3.E, Nr. 2.1, 3.1

FKT Funktion
   1808 Fußgängerzone
   2301 Straßenverkehr

FSZ Anzahl der Fahrstreifen
   ---- tatsächliche Anzahl

IBD Internationale Bedeutung
   2001 Europastraße
   9997 Attribut trifft nicht zu

NTZ Nutzung
   1100 privat
   1200 öffentlich
   1400 militärisch

OFM Oberflächenmaterial (Fahrbahnbefestigung)
   2201 Pflaster
   2700 Beton
   4110 Bitumen, Asphalt
   9999 sonstiges

WDM Widmung
   1301 Bundesautobahn
   1303 Bundesstraße
   1305 Landesstraße, Staatsstraße
   1306 Kreisstraße
   1307 Gemeindestraße
   1308 Forststraße
   9997 Attribut trifft nicht zu
   9999 sonstige

ZUS Zustand
   1100 in Betrieb
   1200 außer Betrieb, stillgelegt
   1300 im Bau
```

Tabelle 2.2: Mögliche Attribute zur Objektart Straße

2.2.2.3 Beispiel einer Attributdatei

Ein Beispiel für eine Attributdatei ist der folgende EDBS-Satz:

```
EDBS00710000BSPE000062␣␣0000ULTANN␣␣00010001N019077␣␣␣901
0002WDM␣␣␣␣1307IBD␣␣␣␣9997
```

In Tabelle 2.3 werden die einzelnen Daten erklärt.

EDBS00710000BSPE000062␣␣0000	
	Zeilenkopf
ULTANN␣␣	Attributdatei
0001	immer 0001
0001	immer 000
N019077	Objektnummer
␣␣␣	Objektteilnummer, hier ganzes Objekt
9	Prüfzeichen
01	Aktualität
0002	WDH Attribute
WDM␣	Attributtyp: Widmung
␣␣␣1307	Attributwert: Gemeindestraße
IBD␣	Attributtyp: Internationale Bedeutung
␣␣␣9997	Attributwert: trifft nicht zu

Tabelle 2.3: Beispiel einer Attributdatei

Die im Beispiel der Grundrißdatei beschriebene Straße wird durch die Attributdatei als Gemeindestraße ohne internationale Bedeutung klassifiziert.

Kapitel 3

Erfassen der Dateien

Die im Kapitel 2.2 beschriebene Grammatik ist Grundlage für das Erfassen der Datei. Es werden zunächst die in der EDBS-Datei stehenden Informationen in eine temporäre Datenbank übertragen. Dies wird erreicht, indem ein Parserprogramm die EDBS-Datei liest und parallel dazu Dateien erstellt, die in einem weiteren Schritt von dem SQL*loader gelesen werden. Dadurch wird die gesamte Information in die temporäre Datenbank geladen.

Zunächst wird im Abschnitt 3.1 auf die temporäre Datenbank eingegangen. In Abschnitt 3.2 wird das Parserprogamm erklärt und anschließend werden einige Bemerkungen zur Benutzung des SQL*loaders angeführt.

3.1 Temporäre Datenbank

Die temporäre Datenbank soll alle Informationen der Attribut- und Grundrißdateien enthalten. Die Struktur orientiert sich daher an deren Grammatik. Daten werden nun auch redundant gespeichert. So werden beispielsweise die Grundrißkoordinaten, die in einer Grundrißdatei einmal gespeichert wurden, nun mit jedem Objekt und mit jeder Linie in deren Datensätzen gespeichert. Prüfzeichen und Aktualitätsflaggen werden nicht in die temporäre Datenbank übernommen, es wäre jedoch ein Leichtes, diese als zusätzliche Spalten in die Tabellen einzufügen.

Die Tabellen und deren Fremd- und Primärschlüssel werden durch ein dem Namen vorangestelltes „`t_`“ gekennzeichnet. Sollten mehrere temporäre Datenbanken gleichzeitig im System vorhanden sein, zum Beispiel, wenn mehrere EDBS-Dateien eingelesen werden sollen, so werden die Tabellen- und Schlüsselnamen durch einen angehängten Index modifiziert.

Zusammen mit den Tabellen der Datenbank werden Primär- und Fremdschlüssel definiert. Dieses sind für die Datenbank Integritätsbedingungen, die in EDBS-Dateien nur schwierig zu überprüfen sind. Insbesondere ist in den EDBS-Dateien nicht garantiert, daß Objekte, deren Geometrien oder Attribute definiert werden, schon bekannt sind. Hier werden die Daten einer EDBS-Datei tabellen-

weise in die temporäre Datenbank geladen, so daß durch die richtige Reihenfolge der Tabellen im Ladevorgang auch die Integritätsbedingungen überprüft werden können. Die Tabellen werden in der Reihenfolge geladen, in der die temporären Tabellen hier aufgeführt werden.

Damit es in allen Tabellen Primärschlüssel geben kann, werden einige laufende Nummern gespeichert, die nicht aus den EDBS-Dateien zu lesen sind, sondern die das Parserprogramm liefert, zum Beispiel die Liniennummern.

Zunächst werden die einzelnen Tabellen einer temporären Datenbank beschrieben. Im Abschnitt 3.1.9 wird ein Modell der gesamten Datenbank aufgezeigt.

3.1.1 t_object

In der Tabelle `t_object` werden die Informationen gespeichert, die in der Datengruppe `<Funktion des Objekts>` gespeichert werden. Außerdem werden die Koordinaten des Objekts gespeichert, die aus der Datengruppe `<Grundrisskennzeichen>` der Grundrißdatei zu entnehmen sind.

Die Tabelle hat folgende Spalten:

Spaltenname	Datenformat	Bedingung	Quelle
ObjNo	VARCHAR2(7)	NOT NULL	Objektnummer
ObjCoordX	NUMBER(9)	NOT NULL	`<Grundrisskennzeichen>`
ObjCoordY	NUMBER(9)	NOT NULL	`<Grundrisskennzeichen>`
Layer	VARCHAR2(3)	NOT NULL	Folie
ObjRace	VARCHAR2(4)	NOT NULL	Objektart
Actual	VARCHAR2(2)	NOT NULL	Aktualität
ObjType	VARCHAR2(1)	NOT NULL	Objekttyp
ModType	VARCHAR2(2)	NOT NULL	Modelltyp
BuildDate	VARCHAR2(6)	NOT NULL	Erstellungsdatum
Different	VARCHAR2(1)	NOT NULL	Veränderungskennung

Die Koordinaten werden als Zahlen gespeichert. Die notwendige Umwandlung aus der Angabe in Normierungsbezirk und Relativkoordinate in absolute Koordinaten leistet bereits das Parserprogrammm (siehe Kapitel 3.2). Die Objektnummer wird als Primärschlüssel definiert. Dies ist möglich, da die Objektnummern eindeutig sein sollen.

3.1.2 t_object_data

Die Informationen der Datengruppe `<besondere Information>` werden in der Tabelle `t_object_data` gespeichert. Die Objektnummer des Objekts, auf das sich die Informationen beziehen, ist über die Objektnummer der Datengruppe `<Funktion des Objekts>`, der dieser Datensatz zugeordnet ist, zu beziehen.

Spaltenname	Datenformat	Bedingung	Quelle
ObjNo	VARCHAR2(7)	NOT NULL	<Funktion des Objekts>
ObjDataNo	NUMBER	NOT NULL	laufende Nummer
InfoRace	VARCHAR2(2)	NOT NULL	Informationsart
MapType	VARCHAR2(2)	NOT NULL	Kartentyp
SigNo	VARCHAR2(6)	NOT NULL	Signaturnummer
Text	VARCHAR2(33)	NOT NULL	Text
GeoRace	VARCHAR2(2)	NOT NULL	Geometrieart
ObjPart	VARCHAR2(3)	NOT NULL	Objektteilnummer

Die Objektnummer ist Fremdschlüssel. Sie muß in der Tabelle `t_object` als Objektnummer vorhanden sein. Die `ObjDataNo` zählt die Datengruppen die zu einer Datengruppe `<Funktion des Objekts>` gehören. Da es zu einem Objekt genau eine Funktion gibt, kann die Kombination `ObjNo` und `ObjDataNo` als Primärschlüssel definiert werden.

3.1.3 t_object_geo

Die zusätzlichen Angaben zu einer Datengruppe `<besondere Information>`, also die Datengruppen `<Geometrieangabe>`, werden in der Tabelle `t_object_geo` gespeichert. Jede dieser Datengruppen ist genau einer Datengruppe `<besondere Information>` zugeordnet, so daß deren Primärschlüssel in dieser Tabelle als Fremdschlüssel definiert wird.

Spaltenname	Datenformat	Bedingung	Quelle
ObjNo	VARCHAR2(7)	NOT NULL	<Funktion des Objekts>
ObjDataNo	NUMBER	NOT NULL	laufende Nummer
ObjParaNo	NUMBER	NOT NULL	laufende Nummer
CoorX	NUMBER(9)	NOT NULL	NBZ und Koordinate
CoorY	NUMBER(9)	NOT NULL	NBZ und Koordinate

Die einzelnen Einträge werden numeriert, wobei die Numerierung nur für eine `<besondere Information>` eindeutig ist. Der Primärschlüssel ist die Kombination aus Fremdschlüssel und Numerierung.

3.1.4 t_line

In der Tabelle `t_line` werden die Liniengeometrien gespeichert. Um diese identifizieren zu können, werden die Datensätze numeriert. Die Nummern müssen demnach innerhalb der ganzen EDBS-Datei eindeutig sein.

Spaltenname	Datenformat	Bedingung	Quelle
LineNo	NUMBER	NOT NULL	laufende Nummer
CoordX	NUMBER(9)	NOT NULL	<Grundrisskennzeichen>
CoordY	NUMBER(9)	NOT NULL	<Grundrisskennzeichen>
EndPointX	NUMBER(9)	NOT NULL	NBZ und Koordinate
EndPointY	NUMBER(9)	NOT NULL	NBZ und Koordinate
Geometry	VARCHAR2(2)	NOT NULL	Art der Liniengeometrie

Die Koordinaten werden aus der Datengruppe <Grundrisskennzeichen> der eigenen Grundrißdatei entnommen.

3.1.5 t_line_function

Die Funktionen der Linien, also die Zuordnung der Linien zu den Objekten, werden in der Tabelle t_line_function aufgelistet. Diese Informationen stehen in der Datengruppe <Funktion der Linie>, die einem bestimmten <Endpunkt der Linie> zugeordnet ist. Die Liniennummer wird als Fremdschlüssel in die Tabelle aufgenommen. Eine laufende Nummer (LinefunctionNo) ist zusammen mit dem Fremdschlüssel der primäre Schlüssel.

Spaltenname	Datenformat	Bedingung	Quelle
LineNo	NUMBER	NOT NULL	laufende Nummer
LineFunctionNo	NUMBER	NOT NULL	laufende Nummer
Layer	VARCHAR2(3)	NOT NULL	Folie
LineRace	VARCHAR2(4)	NOT NULL	Linienart
ObjNoR	VARCHAR2(7)		Objektnummer rechts
ObjNoL	VARCHAR2(7)		Objektnummer links
ObjNoPartR	VARCHAR2(3)		Objektteilnummer rechts
ObjNoPartL	VARCHAR2(3)		Objektteilnummer links
LineDifR	VARCHAR2(1)		Linienteilung rechts
LineDifL	VARCHAR2(1)		Linienteilung links

Die beiden Objektnummern ObjNoL und ObjNoR sind eine Referenz auf die Objektnummer der Tabelle t_object. Es ist möglich, daß nur ein Objekt referenziert wird. Die Referenz und die Einträge zur Objektteilnummer und Linienteilung werden dann mit NULL belegt.

3.1.6 t_line_data

Die Lageparameter einer Linie, beim Polygonzug die Stützpunkte, werden in der Tabelle t_line_data gespeichert. Die einzelnen Datensätze sind genau einer Liniengeometrie zugeordnet. Dies wird mit der Liniennummer als Fremdschlüssel realisiert. Die laufende Nummer der Lageparameter ist der Reihenfolge in der Grundrißdatei zu entnehmen, da die Reihenfolge der Lageparameter wichtig ist.

Spaltenname	Datenformat	Bedingung	Quelle
LineNo	NUMBER	NOT NULL	laufende Nummer
LineDataNo	NUMBER	NOT NULL	laufende Nummer
CoordX	NUMBER(9)	NOT NULL	NBZ und Koordninate
CoordY	NUMBER(9)	NOT NULL	NBZ und Koordninate

Die Datensätze sind eindeutig durch Liniennummer und Lageparameternummer (`LineDataNo`) zu identifizieren. Diese Kombination bildet den primären Schlüssel.

3.1.7 t_line_param

Die zusätzlichen Informationen zu Linien werden in der Datengruppe `<Fachparameter>` angegeben und in der Tabelle `t_line_param` gespeichert.

Spaltenname	Datenformat	Bedingung	Quelle
LineNo	NUMBER	NOT NULL	laufende Nummer
LineFunctionNo	NUMBER	NOT NULL	laufende Nummer
LineParamNo	NUMBER	NOT NULL	laufende Nummer
Race	VARCHAR2(1)	NOT NULL	Fachparameterart
Type	VARCHAR2(1)	NOT NULL	Fachparametertyp
Value	VARCHAR2(7)	NOT NULL	Wert

Ein Datensatz ist durch die Kombination von Liniennummer und Linienfunktionsnummer Fremdschlüssel genau einem Eintrag der Tabelle `t_line_function` zugeordnet. Zusammen mit einer laufenden Nummer ergibt sich der Primärschlüssel.

3.1.8 t_attribute

Die Angaben in den Attributdateien werden in die Tabelle `t_attribute` eingefügt. Objektnummer und Objektteilnummer werden aus der Datengruppe `<Attributkennzeichen>` entnommen, Attributtyp und -wert aus `<Attribut>`. Für jede Attributdatei werden soviele Einträge erstellt, wie es der Wiederholungsfaktor der Datengruppe `<Attribut>` anzeigt.

Spaltenname	Datenformat	Bedingung	Quelle
ObjNo	VARCHAR2(7)	NOT NULL	Objektnummer
ObjPartNo	VARCHAR2(7)	NOT NULL	Objektteilnummer
Type	VARCHAR2(4)	NOT NULL	Attributtyp
Value	VARCHAR2(7)	NOT NULL	Attributwert

Das Objekt, auf das sich das Attribut bezieht, wird über die Objektnummer als Fremdschlüssel identifiziert. Es wird nicht untersucht, ob ein Attribut für die Objektart des referenzierten Objekts zugelassen ist, ebenso wird der Attributwert nicht überprüft.

Der Primärschlüssel der Tabelle ist die Kombination der Spalten `ObjNo`, `ObjPartNo` und `Type`. Einem Objektteil dürfen beliebig viele Attribute zugeordnet werden, deren Werte sollen aber eindeutig sein.

3.1.9 Modell der temporären Datenbank

Das Modell in der Abbildung 3.1 zeigt auf, welche Fremdschlüssel definiert sind. An den Kanten stehen die Constraint-Namen der Fremdschlßselattribute.

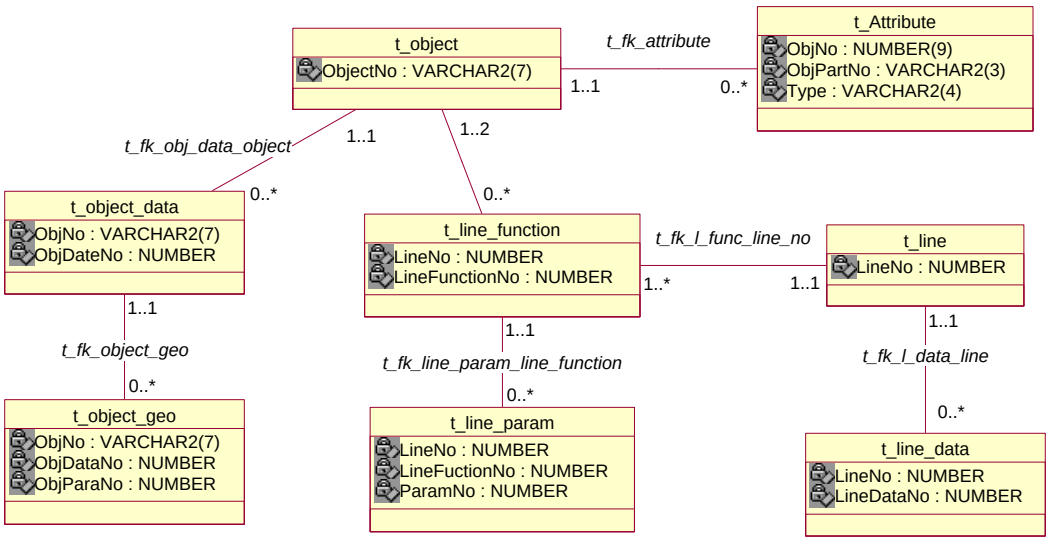

Abbildung 3.1: Modell der temporären Datenbank; eingetragenen Daten sind Schlüsselelemente

3.2 Parserprogramm

Das Parserprogramm hat die Aufgaben

1. eine EDBS-Datei einzulesen,
2. Daten- und Kontrolldateien für den SQL*loader zu erstellen,
3. das Schema der temporäre Datenbank zu erstellen und
4. den Ladevorgang durchzuführen.

Das Programm erstellt zu einer EDBS-Datei eine temporäre Datenbank.

Um für Erweiterungen und Veränderungen offen zu sein, ist das Parserprogramm objektorientiert in Java geschrieben [RRZ97]. Die Klassen des Packages `parser` sind im Folgenden aufgeführt und erläutert.

Zum Aufruf des Programms wird die Methode `main()` der Klasse `ReadIt` aufgerufen, näheres im Abschnitt 6.2.

Weitere Informationen sind in der javadoc-Dokumentation gegeben. Diese kann bei der Installation angefertigt werden.

3.2.1 Konstanteninterface

In der abstrakten Klasse `ConstantsBase` werden alle benötigten Konstanten erklärt. Bereits definiert und als unveränderbar `final` deklariert sind die Längen der einzelnen Daten im EDBS-Format, SQL-Kommandos zur Erstellung der temporären Datenbank sowie Kommandos der SQL*loader Dateien. Die Fehlermeldungen und Dateinamenendungen sind definiert, können aber überschrieben werden. Somit können verschiedene Konstantenklassen für unterschiedliche Sprachen oder Benutzer unterschiedlichen Vorwissens implementiert werden. Die in dieser Interface-Klasse erklärten Fehlermeldungen und Eingabeaufforderungen sind auf deutsch.

3.2.2 Konstantendatei

Die Klasse `Constants` implementiert das Konstanteninterface. Es ist möglich, eine andere Konstantenklasse zu definieren, die Fehlermeldungen in einer anderen Sprache bietet oder die für Benutzer mit mehr oder weniger Computer- oder Datenbankenwissen zugeschnitten ist. Hier wird die Belegung der Fehlermeldungen etc. der Basisklasse übernommen.

3.2.3 Dateifilter

Um die Datei in einem Dialog auszuwählen, wird ein Dateifilter benötigt. Die Klasse `MyFileFilter` ist ein solcher. Es werden alle Dateien akzeptiert.

3.2.4 Parserausnahme

Die `ParsingException` ist eine `Exception`, die um eine zusätzliche Zeile erweitert ist. Darin wird die Zeile, in der ein Fehler aufgetreten ist, zitiert.

3.2.5 Parser

Die Prozeduren für das eigentliche Einlesen und Parsen der EDBS-Datei werden durch die Klasse `ReadIt` zur Verfügung gestellt. Es können Datendateien aus einer EDBS-Datei für den SQL*loader erstellt werden. Treten beim Parsen Fehler auf, wird der Einlesevorgang in der nächsten Zeile fortgesetzt, bei erkannten Syntaxfehlern, etwa einem fehlerhaften Wiederholungsfaktor, werde nur die betroffenen Daten verarbeitet. Ist in der Konstantenklasse des Packages die Variable

LOG gesetzt ist, werden zusätzliche Informationen, zum Beispiel zu fehlerhaften Zeilen, ausgegeben.

Die EDBS-Dateien werden zeilenweise eingelesen und bearbeitet. Um direkt der Grammatik der Grundriß- und Attributdateien nachzugehen, gibt es für jede Datengruppe eine Funktion, die die ihr zugeordneten Datengruppen bearbeitet. In dem von all diesen Methoden verfügbaren `Vector` werden die eingelesenen Daten gespeichert. Zum einem können ausgelesene Informationen an andere Methoden übergeben werden, zum Beispiel die Koordinaten des Grundrißkennzeichens. Andererseits kann die Methode `writeDataTupel()` diesen `Vector` zu einer Zeile zusammenfassen, die direkt in eine der Datendateien geschrieben werden kann. Nachteilig ist, daß der `Vector` sehr genau verwaltet werden muß.

Das Flußdiagramm in Abbildung 3.2 zeigt, wie die Grammtik nachvollzogen wird, und welche Methoden für das Bearbeiten welcher Daten verantworlich sind.

Nach Abschluß des Parsevorgangs liegen acht Datendateien vor, die vom SQL*loader geladen werden können. Die Methode `main()` erstellt mit der Methode `createTempTables()` der Datenbankerstellungs-Klasse eine temporäre Datenbank. Anschliessend werden die Datendateien in die Datenbank eingefügt indem die Methode `startLoader()` der Datenbankeinfügungs-Klasse aufgerufen wird.

3.2.6 Dateischreiber

Um den Dateinamen einer zum Schreiben geöffneten Datei auch nach dem Öffnungsvorgang noch zur Verfügng zu haben, ist die Klasse `MyFileOutputStream` vorhanden. Sie ist eine Erweiterung eines einfachen `MyFileOutputStream`, jedoch wird auch der Dateiname gespeichert.

3.2.7 Datenbankerstellung

Um die Datenbanken zu erstellen, wird die Datenbankanbindung JDBC genutzt. Die Klasse `CreateTempTables` bietet die Möglichkeit, eine Datenbank mit allen temporären Tabellen zu erstellen. Dabei werden eventuell alle Namen um einen Index erweitert, damit auch mehrere temporäre Datenbanken gleichzeitig ohne Überschneidungen im Datenbanksystem vorhanden sein können. Es gibt eine in der Konstantenklasse festgelegte Obergrenze für den Index, die nicht überschritten wird. Der genaue Ablauf zur Erstellung einer temporären Datenbank wird im Abbildung 3.3 gezeigt.

Alle Datei- und Spaltennamen sowie Tabellen- und Spaltendefinitionen sind im Konstanteninterface festgelegt.

Ferner können die Kontrolldateien für den SQL*loader geschrieben werden, wobei der Index der erstellten temporären Datenbank berücksichtigt werden muß.

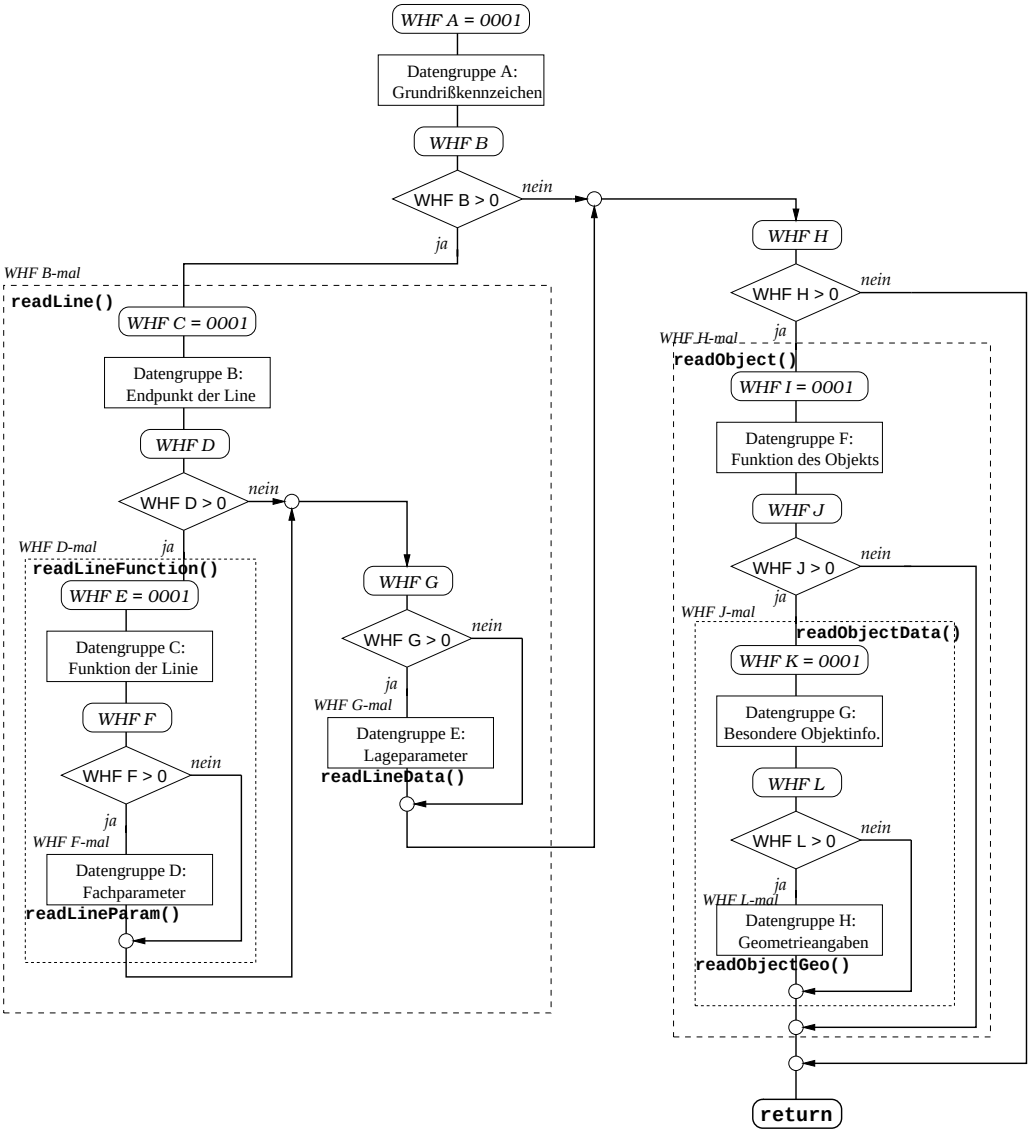

Abbildung 3.2: Flußdiagramm zum Einlesen einer Grundrßdateien

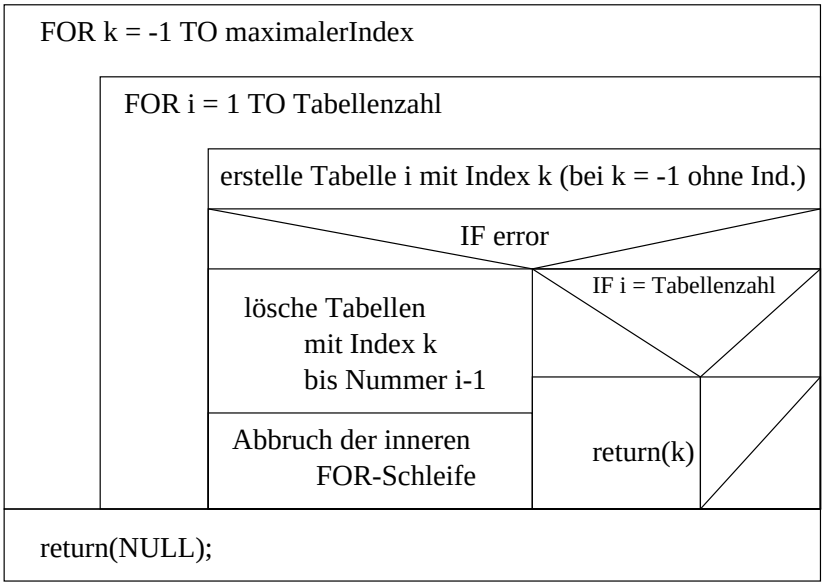

Abbildung 3.3: Ablaufdiagramm zur Erstellung eines Tabellenschemas

3.2.8 Shell

Die Klasse `Shell` stellt eine einfache Shell zur Verfügung, mit deren Hilfe der SQL*loader gestartet werden kann.

3.2.9 Datenbankeinfügung

Die Klasse `Loader` stellt die Methode `startLoader()` zur Verfügung. Diese erstellt die benötigten Kontrolldateien für den SQL*loader und eine Datei, die ausgeführt werden muß, um die Daten in die temporäre Datenbank einzufügen. Mit der Shell wird diese Datei gestartet.

3.3 SQL*loader

Das Programm SQL*loader ist Teil der Oracle8*i* Distribution. Es ist ein Hilfmittel, das es ermöglicht, eine Datendatei direkt in eine Tabelle der Datenbank zu importieren.

Der Aufruf des SQL*loader zum Import von Objektdaten lautet:

```
sqlldr USERID=<SQLUSER>/<PASSWD> CONTROL=<CONTROL FILE>
DATA=<DATA FILE> LOG=<LOG FILE> BAD=<BAD FILE> ERRORS=<MAX ERRORS>
SILENT=HEADER,FEEDBACK
```

Die Parameter des Aufrufs sind:

<SQLUSER>	Benutzername bei dem Datenbanksystem
<PASSWD>	Benutzerpaßwort bei dem Datenbanksystem
<CONTROL FILE>	Pfad und Name der Kontrolldatei
<DATA FILE>	Pfad und Name der Datendatei
<BAD FILE	Pfad und Name der Fehlerdatei
<LOG FILE	Pfad und Name der Log-Datei
<MAX ERRORS>	Anzahl der erlaubten Fehler

Fehlerhafte Datensätze, also solche, die die Integritätsbedingungen der Tabelle verletzen, werden nicht in die Tabelle eingefügt. Gleichzeitig wird ein Fehlerzähler inkrementiert. Überschreitet dieser die angebene Höchstgrenze, so wird der Vorgang abgebrochen.

Die Angabe von `SILENT=HEADER,FEEDBACK` bewirkt die Unterdrückung der Bildschirmausgabe von Fortschrittmeldungen. Ist im Konstanteninterface die Variable `LOG` gesetzt, wird diese Angabe weggelassen.

Alle Pfade zu den Dateien werden relativ zu dem Verzeichnis angegeben, in dem dieser Aufruf erfolgt. Die Aufgaben der einzelnen Dateien sind im Folgenden aufgeführt:

3.3.1 Kontrolldatei

Die Kontrolldateien steuern den SQL*loader. In diesen Dateien wird der Name der Tabelle, in die die Daten eingefügt werden, und die betroffenen Spaltennamen angegeben. Zu jeder vom Parser erstellten Datendatei wird eine eigene Kontrolldatei erstellt. Der Aufbau ist der folgende:

```
LOAD DATA
INFILE <DATA FILE>
INTO <Tabellenname>
FIELD TERMINATED BY ','
(<Spaltenname 1>, ... , <Spaltenname n>)
```

Alle Kontrolldateien sind im Anhang A.1 zitiert.

3.3.2 Datendatei

Die Datendateien werden vom SQL*loader erstellt. In jeder Zeile wird ein Datensatz angegeben. Die einzelen Einträge werden durch ein Trennzeichen, daß auch in der Kontrolldatei angegeben ist, getrennt.

3.3.3 Fehlerdatei

Die Fehlerdateien werden vom SQL*loader erstellt. In diese werden alle Zeilen der Datendateien geschrieben, deren Datensätze nicht in die Tabelle aufgenom-

men werden konnten, weil sie fehlerhafte Daten enthielten. Diese Fehler können verletzte Integritätsbedingungen oder nicht eingehaltene Spaltenformate sein.

3.3.4 Log-Datei

Auch diese Datei wird vom SQL*loader erstellt. In dieser wird der Einfügevorgang dokumentiert. Angegeben wird, wieviele Datensätze in welcher Zeit eingelesen werden konnten, Fehler und deren Spezifikation und Ähnliches.

Kapitel 4

Modell der Datenbank

Die hier vorgestellte Datenbank speichert die topographischen Angaben mittels des `MDSYS.SDO_GEOMETRY`-Typs des Datenbanksystems Oracle8*i*, der räumliche Daten speichern kann. Ein Vorteil ist, daß das Datenbankmodell sehr einfach wird. Ein anderer ist es, daß mit dem Datenbanksystem Indexe, Methoden etc. zur Verfügung gestellt werden, mit deren Hilfe die Arbeit mit den geographischen Objekten sehr erleichtert und verbessert wird.

4.1 Externe Sicht

Die externe Sicht auf die Datenbank ist an geographischen Objekten orientiert. Alle Objekte der Datenbank haben eine direkte oder eine indirekte Referenz auf ein geographisches Objekt. Die externe Sicht auf die Datenbank wird in Abbildung 4.1 dargestellt.

Im Unterschied zu den Arbeiten von Koch [Koc94] [NK95] und Korte [Kor97] kann mit dieser Arbeit die geographische Beschreibung der Objekte zu einem einzigen Objekt zusammengefaßt werden, wärend für diese Informationen in den früheren Datenbanken mehrere Objekte und Relationen benutzt werden mußten.

Die Formatdaten, also die Objektarten, Folien und Objektattribute, werden nur einmal in das Datenbanksystem eingefügt. Die dort eingetragenen Daten werden direkt aus dem entsprechenden Objektartenkatalog des jeweiligen Modells entnommen. Sind mehrere Datenbanken im Datenbanksystem, so benutzen alle die gleichen Formatdaten.

4.2 Interne Struktur der Datenbank

Die interne Struktur der Datenbank ist neben der üblichen Abbildung von $n..n$-Relationen auf Tabellen und $1..n$-Relationen auf Fremdschlüsselmit mit einer kleinen Änderung wie die externe Sicht der Datenbank. Jedes Objekt besteht aus Objekteilen, die durch je eine Objektgeometrie beschrieben werden. Objekte

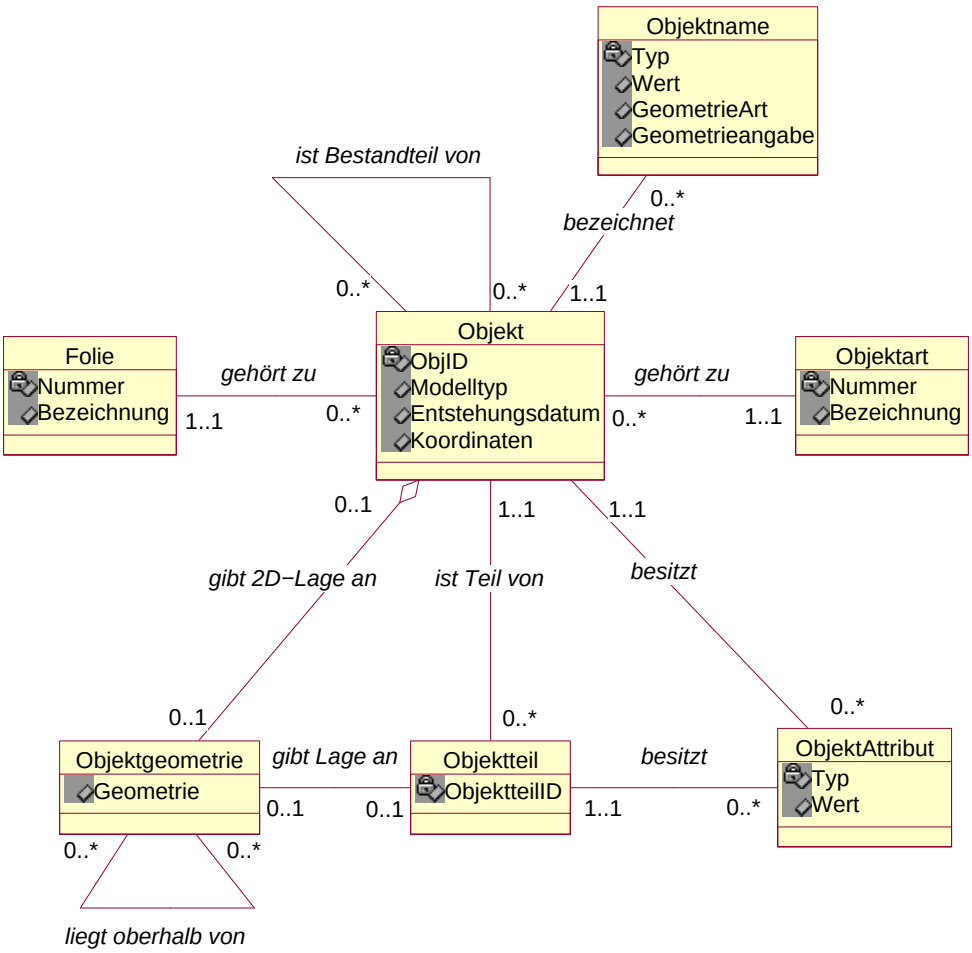

Abbildung 4.1: Externe Sicht der Datenbank. Die Schloßsymbole kennzeichnen Schlüsselattribute.

können nicht direkt durch eine Geometrie beschrieben werden, sondern nur über die Objektteile. Die einzelnen Tabellen werden im Folgenden vorgestellt.

4.2.1 objects

In der Tabelle `objects` werden die Informationen gespeichert, die zu jedem Objekt vorhanden sind.

Spaltenname	Datenformat	Bedingung	Information
ObjNo	VARCHAR2(7)	NOT NULL	Objektnummer
LayerNo	VARCHAR2(3)	NOT NULL	Folie
ObjectRaceNo	VARCHAR2(4)	NOT NULL	Objektart
CoordinateX	NUMBER(9)	NOT NULL	Koordinate des Objekts
CoordinateY	NUMBER(9)	NOT NULL	Koordinate des Objekts
Actual	VARCHAR2(2)	NOT NULL	Aktualität
ModType	VARCHAR2(2)	NOT NULL	Modelltyp
BuildDate	VARCHAR2(6)		Erstellungsdatum

Sowohl die Objektart als auch die Folie sind Fremdschlüssel. Die Objektnummer ist der Primärschlüssel der Tabelle und somit eindeutig.

4.2.2 object_geometries

Die einzelnen Objektteile und deren Geometrien werden in der Tabelle `object_geometries` gespeichert. Die gesamte topographische Information wird in dem Datentyp `MDSYS.SDO_GEOMETRY` der `Spatial`-Erweiterung von Oracle8*i* gespeichert. Um eine einfache Verwaltung zu ermöglichen werden topographische Informationen ausschließlich zu Objektteilen gespeichert. Kann ein Objekt durch eine einzige Geometrie beschrieben werden, so wird in dieser Tabelle eine Zeile eingefügt, die diese Geometrie als ein Objektteil verwaltet. Bei punktförmigen Objekten ist dies immer der Fall.

Spaltenname	Datenformat	Bedingung	Information
ObjectNo	VARCHAR2(7)	NOT NULL	Objektnummer
ObjectPartNo	VARCHAR2(3)	NOT NULL	Objektteilnummer
ObjectGeometry	MDSYS.SDO_GEOMETRY		geometrische Beschreibung

Die Objektnummer ist ein Fremdschlüssel mit Referenz auf auf ein Eintrag der Tabelle `objects`. Als Primärschlüssel ist die Kombination aus Objektnummer und -teilnummer definiert.

4.2.3 above

Unter- bzw. Überführungen werden in der Tabelle `above` gespeichert. In der externen Sicht ist dies durch die Relation `liegt über` realisiert.

Spaltenname	Datenformat	Bedingung	Information
TopNo	VARCHAR2(7)	NOT NULL	Objektnummer oben
TopPartNo	VARCHAR2(3)		Objektteilnummer oben
DownNo	VARCHAR2(7)	NOT NULL	Objektnummer unten
DownPartNo	VARCHAR2(3)		Objektteilnummer unten

Ist einer der Einträge der Objektteilnummer `NULL`, so wird ein gesamtes Objekt über- bzw. unterführt. Die Objektnummern müssen in der Tabelle `objects` vorkommen.

4.2.4 complex_objects

Objekte können einem anderen Objekt zugeordnet werden. Dies ist dann ein komplexes Objekts. Diese Zuordnungen werden in der Tabelle `complex_objects` gespeichert. In der externen Sicht ist dies durch die Relation `liegt oberhalb von` realisiert.

Spaltenname	Datenformat	Bedingung	Information
BigNo	VARCHAR2(7)	NOT NULL	Nr. des komplexen Objekts
SmallNo	VARCHAR2(7)	NOT NULL	Nr. des zugeordneten Objekts

Beide Objektnummern sind Fremschlüssel, die auf einen Eintrag in der Tabelle `objects` verweisen.

4.2.5 object_names

Die Namen einzelner Objekte können in der Tabelle `object_names` abgelegt werden. In der geschachtelten Tabelle (nested table) werden die Positionen der Schriftzüge angegeben.

Spaltenname	Datenformat	Bedingung	Information
ObjectNo	VARCHAR2(7)	NOT NULL	Objektnummer
TypeNo	VARCHAR2(2)	NOT NULL	Art des Namen
Value	VARCHAR2(31)		der Name
Geo	Name_Geo_Table_Type		Tabelle mit Positionen

4.2.6 attributes

Die Attribute der einzelnen Objekte oder der Objektteile werden in der Tabelle `attributes` gespeichert. Ein Attribut kann einem Objektteil zugeordnet werden. Ist kein Objektteil angegeben, so wird das Attribut dem Objekt zugeordnet. In einem solchen Fall gilt das Attribut für alle Teile des Objekts.

Spaltenname	Datenformat	Bedingung	Information
ObjectNo	VARCHAR2(7)	NOT NULL	Objektnummer
ObjectPartNo	VARCHAR2(3)		Objektteilnummer
Type	VARCHAR2(4)	NOT NULL	Attributnummer
Value	VARCHAR2(7)	NOT NULL	Attributwert

Die Objektnummer wird als Fremdschlüssel mit Referenz auf die Tabelle `objects` definiert. Wenn ein Attribut einem Objektteil zugeordnet ist, wird nicht überprüft, ob dieses Objektteil auch in der Tabelle `object_geometries` aufgeführt wird.

4.3 Formattabellen

Mit den im Folgenden erläuterten Tabellen werden die Definitionen eines Objektartenkataloges im Datenbanksystem einmalig gespeichert. Es werden dabei die Nummern und deren Bedeutung gespeichert. Im Gegensatz zu den Objektartenkatalogen wird hier nicht definiert, welche Attribute zu welcher Objektart zugelassen werden. Es wäre ja beispielsweise sinnlos, einem Acker ein solches Attribut zuzuschreiben, daß ihn als Autobahn auszeichnet. Da jedoch eine Überprüfung dieser Integritätsbedingung daten- und rechenaufwendig wäre, wird auf diese Informationen in der Datenbank verzichtet.

4.3.1 object_race

In der Tabelle `object_race` werden die möglichen Objektarten gespeichert. Es gibt zu jedem ATKIS-Modell ein Objektartenkatalog, der entsprechend in eine Tabelle umgesetzt werden muß. Die Spalten sind die folgenden:

Spaltenname	Datenformat	Bedingung	Information
ObjectRaceNo	VARCHAR2(4)	NOT NULL	Objektartnummer
Name	VARCHAR2(100)		Bezeichnung der Objektart

4.3.2 layer

Jede Objektart ist einer Folie zugeteilt, die etwas über die Funktion der Objektart aussagt. Diese Gliederung ist eine andere als die hierachische Gliederung, die sich in den Objektartennummern ausdrückt. Dies ist eine redundant gespeicherte Information, da aus der Objektartennummer eines Objekts bereits hervorgeht, welcher Folie das entsprechende Objekt zuzuordnen ist. In der Literatur wird auf diese Information nicht eingegangen. Der Vollständigkeit halber sei hier die Tabelle `layer` aufgeführt, auf die sich die Foliennummern der Objekte beziehen.

Spaltenname	Datenformat	Bedingung	Information
LayerNo	VARCHAR2(4)	NOT NULL	Foliennummer
Name	VARCHAR2(100)		Folienbezeichung

4.3.3 attribute_types

In der Tabelle `attribute_types` werden die Attributnummern und -bezeichnungen gespeichert. Mit Hilfe dieser Tabelle kann zu einer zugeteilten Attributnummer festgestellt werden, wie ein Attribut bezeichnet wird.

Spaltenname	Datenformat	Bedingung	Information
AttTypeNo	VARCHAR2(4)	NOT NULL	Attributnummer
Name	VARCHAR2(100)		Bezeichung

4.3.4 attribute_values

Die Werte der Attribute werden in der Tabelle `attribute_values` abgelegt. Eine bestimmte Nummer eines Attributes hat eine festgelegte Bedeutung. Ist es möglich, zu einem Attribut den Wert in einer Zahl anzugeben, beispielsweise die Straßenbreite in Metern, so ist in der Attributwertetabelle „␣␣␣----“ eingetragen.

Spaltenname	Datenformat	Bedingung	Information
Type	VARCHAR2(4)	NOT NULL	Attributnummer
Value	VARCHAR2(4)	NOT NULL	Attributwert
Name	VARCHAR2(100)		Bezeichung

Kapitel 5

Aufbereitung der Daten

Um die Daten von den tempoären Tabellen in die endgültige Datenbank zu laden, sind zwei Schritte notwendig: Tabellenerstellung und Ladevorgang. Die Methode `create()` des `PL/SQL-packages` [Cor99b] führt beide Schritte durch.

5.1 Tabellenerstellung

Die Tabellenerstellung erfolgt mit eventuellen Indizes in Tabellen und Schlüsselnamen nach dem gleichen Muster, wie die Erstellung der temporären Datenbank (siehe Abbildung 3.3). Die Obergrenze des Index ist im Quellcode festgelegt und kann beliebig verändert werden. Bei einer erfolgreichen Erstellung der Datenbank und abgeschlossenem Ladevorgang, wird der Index als `VARCHAR2` von der Methode `create()` zurückgeliefert. Bei Fehlern ist das Ergebnis `NULL`.

5.2 Ladevorgang

Der Ladevorgang der Daten ist auf das Laden der Daten der einzelnen Tabellen eingeteilt. Die Tabellen werden in der Reihenfolge der Beschreibung der Ladevorgänge durchgeführt, damit es nicht zur Verletzung von Fremdschlüsselbedingungen kommt.

Damit die Daten aus der richtigen temporären Datenbank entnommen werden, muß der Index der temporären Datenbank im Aufruf von `create()` angegeben werden.

Die im Kapitel 4.3 erklärten Tabellen, also die Tabellen, die das Datenformat beschreiben, müssen zuvor mit dem SQL*loader geladen werden.

5.2.1 objects

Die Daten, die den Objekten direkt zugeordnet werden, werden aus der temporären Tabelle `t_objects` übernommen. Da es unterschiedliche Erweiterungen

der Objektartenkataloge der verschiedenen Landesvermessungsämter gibt, werden unbekannt Objektarten und Folien, die als Fremdschlüssel auftreten, in die entsprechenden Tabellen eingefügt, und zwar mit einer Beschreibung, die darauf hinweist, daß dieser Eintrag automatisch erstellt wurde. Unbekannt bedeutet hier, daß es keinen passenden Eintrag in der Formattabelle gibt.

Wichtig ist das Laden der Objektdaten zu Beginn, da alle anderen Tabellen die Objektnummer als direkten oder indirekten Fremdschlüssel haben.

5.2.2 object_names

Für jede Objektbezeichnung der Tabelle `t_object_name` wird ein Eintrag in die Tabelle `object_names` vorgenommen. Die Spalte mit der Tabelle der Geometrieangaben zu den Namenszügen bleibt zunächst unbelegt. Dieser Vorgang kann durch das Ausführen einer einzigen Einfügeanweisung, die eine geschicke Auswahlanweisung benutzt, durchgeführt werden. Wegen der Fremdschlüsseldeklaration der Tabelle `object_names` wird gewährleistet, daß keine Namen zu unbekannten Objektnummern gespeichert werden.

In einem weiteren Schritt, wird die auf diese Art gefüllte Tabelle vervollständigt. Alle Geometrieangaben der Tabelle `t_object_geo` werden so in die eingebaute Tabelle eingefügt, daß sie dem passenden Objekt zugeordnet werden. Mittels eines Cursors der Tabelle `object_geo` werden die Geometrieangaben durchgegangen. Mittels eines zweiten Cursors wird aus der Tabelle `object_names` die Zeile ausgewählt, in die die Geometrieangabe eingefügt werden soll. Dabei werden sowohl Objektnummer als auch die Bezeichnerart auf Gleichheit geprüft.

5.2.3 object_geometries

Das Einfügen der topografischen Daten in die Datenbank ist die wesentliche Aufgabe dieser Arbeit. Es werden verschiedene Prozeduren und Funktionen benutzt, deren Aufgabe hier kurz aufgezeigt wird. Weiter unten folgt eine genauere Beschreibung der Arbeitsweise.

- `load_object_geometry()`: Mit dieser Prozedur werden die topologischen Informationen zu den Objekten in die Tabelle `object_geometries` geladen. Zu allen linien- oder flächenförmigen Objekten werden in die temporäre Tabelle `t_geo` Daten geladen, die von den folgenden Methoden bearbeitet und ausgewertet werden.

- `update_t_geo_line()`: Diese Prozedur bearbeitet die topologischen Daten linienförmiger Objekte, die in der Tabelle `t_geo` stehen, so daß eine Beschreibung der Linie einfach wird. Es wird dabei ein Linienverfolgungsalgorithmus benutzt.

- `update_t_geo_area()`: Unter Benutzung der Prozedur für linienförmige Objekte werden die topologischen Angaben zu flächenförmigen Objekten sortiert.

- `describe_line()`: Diese Funktion liefert eine Zeichenkette, die die topologischen Angaben eines linienförmigen Objekts auf Basis der aufbereiteten Daten der Tabelle `t_geo`.

- `describe_area()`: Diese Funktion liefert eine solche Zeichenkette für die flächenförmigen Objekte

Bei punktförmigen Objekten wird in der Tabelle `objects_geometries` eine einfache Angabe der Punktkoordinaten angegeben. In der `Spatial`-Erweiterung von Oracle8*i* ist im Datentyp `MDSYS.SDO_GEOMETRY` die Spalte `SDO_POINT` für punktförmige Objekte vorgesehen. Die Einträge der Spalten `SDO_ELEM_INFO` und `SDO_ORDINATES` sind `NULL`. Genaue Angaben und Hinweise werden in der Dokumention von Oracle8*i* gegeben.

Umgekehrt bleibt bei linien- und flächenförmigen Objekten die Spalte für die Punktkoordinaten frei. Um die Spalten der geometrischen Angabe zu füllen, sind kompliziertere Berechunungen durchzuführen. Diese werden objektweise durchgeführt: Zunächst werden die geometrischen Angaben aller Linien in die temporäre Tabelle `t_geo` geladen, die als Linienfunktion die Nummer des Objekts eingetragen haben. Die Prozeduren `update_t_geo_line` bzw. `update_t_geo_area` sortieren diese Tabellen, indem in dafür vorgesehenen Spalten Sortierungsnummern gesetzt werden. Wie dies genau geschieht, wird in den Abschnitten 5.2.3.1 und 5.2.3.2 aufgezeigt.

Zu jedem Objektteil wird eine Zeichenkette erstellt, die eine Einfügeanweisung darstellt, durch deren Ausführung die topologischen Angaben des Objektteils in die Datenbank geladen werden. Man beachte, daß die Erstellung und Bearbeitung der temporären Tabellen objektweise, die Einfüngung in die Datenbank jedoch objektteilweise geschieht. Die Funktionen `describe_line` und `describe_area` liefern den Teil der Zeichenkette, die Daten für die Spalten `SDO_ELEM_INFO` und `SDO_ORDINATES` enthält. Die Funktionsweise der beiden Funktionen wird in den Abschnitten 5.2.3.3 und 5.2.3.4 behandelt.

5.2.3.1 update_t_geo_line

Aus der temporären Tabelle wird die Linie ausgewählt, die den Anfangs- oder Endpunkt hat, der am weitesten links liegt. Diese Linie wird mit einer Nummer versehen. Aus der Tabelle wird nun eine unbenutzte Linie gesucht, deren Anfangs- oder Endpunkt mit dem Endpunkt der Startlinie übereinstimmt. Diese Linie wird mit der nächsthöheren Nummer markiert und die Richtung, in der die Linie in den Linienzug eingefügt werden muß, gespeichert. Dieser Anstückelungsvorgang wird wiederholt, bis keine nächste unbenutzte Linie mehr gefunden werden kann.

Anschließend wird dieser Linienverfolgungsalgorithmus vom Startpunkt der Anfangslinie startend angewendet.

Anzumerken bleibt, daß laut ATKIS-Objektartenkatalog nur einfache Linien beschrieben werden dürfen. Die Linien dürfen nicht aufspalten, durch Verbindungslinien aufgespalten werden, die eigene Linie kreuzen oder berühren.

5.2.3.2 update_t_geo_area

Der Algorithmus für die Flächen benutzt den Linienalgorithmus. Die Linien, die dabei erkannt werden, sind geschlossene Polygone. Diese dürfen sich nicht selbst durchdringen. Um Inselpolygone, also Polygone, bei denen ein Stück ausgeschnitten ist, zu verarbeiten, werden nacheineinander Linienzüge, also einfache Polygone, erkannt. Nach dem oben angeführten Auswahlkriterium für die Startlinie ist das erste Polygon das äußere.

5.2.3.3 describe_line

Der Eintrag der Spalte `SDO_ORDINATES` wird als eine Liste von durch Komma getrennte Koordinaten angegeben, deren Bedeutung durch den Eintrag in der Spalte `SDO_ELEM_INFO` angegeben wird. Ist dieser Eintrag `(1, 2, 1)`, so wird ein Linienzug beschrieben, bei dem die Beschreibung des Startpunktes an der ersten Stelle der Koordinatenliste beginnt.

Um die Koordinatenangaben als String zu schreiben, wird die bearbeitete Tabelle `t_geo` nach der eingetragenen Sortiernummer sortiert. Die Anfangskoordinaten jedes Linienstücks werden an die beschreibende Zeichenkette angehängt. Ist die Linie ein Polygonzug, so werden die Lageparameter ebenfallls angefügt. Diese sind in der Tabelle `t_line_data` gespeichert. Durch die Liniennummer wird festgestellt, welches die einzutragenden Daten sind. Die Reihenfolge ergibt sich aus der Nummer `LineDataNo`. Dabei muß beachtet werden, in welcher Richtung die Linie durchlaufen wird.

Abschliessend werden die Endkoordinaten des Linienzuges angehängt. Als Rückgabewert der Funktion ergibt sich dann diese zusammengesetzte Zeichenkette, vor die noch der oben zitierte Eintrag gesetzt wird.

5.2.3.4 describe_area

Der Eintrag der Spalte `SDO_ELEM_INFO` ist nun eventuell etwas komplizierter: Eine Fläche wird eventuell nicht nur durch ein umschließendes Polygon, sondern zusätzlich durch Polygone beschrieben, die Teilflächen aus dem umschließenden Polygon herausschneiden. Für jedes Polygon wird ein Tripel `(offset, 3, 1)` eingetragen. Dabei steht `offset` für die Position in der Koordinatenliste, ab der das Polygon beschrieben wird.

Die Koordinaten werden ebenso wie bei der Linienbeschreibung aufgenommen, jedoch wird das umschließende Polygon und alle Inselpoygone nacheinander eingetragen und zugleich die `offset`-Werte gesetzt.

5.2.4 complex_objects

Die Tabelle `complex_objects` speichert hierachische Strukturen der Objekte. Diese Daten werden aus der temporären Tabelle `t_object_data` gewonnen. Die durch eine Auswahlanweisung aus der temporären Tabelle herausgefilterten Daten werden direkt in die Datenbank eingefügt.

5.2.5 above

Unter- und Überführungen werden in die Tabelle `above` eingefügt, indem alle Einträge der Tabelle `t_object_data`, die eine solche Information enthalten, übertragen werden. Mit einer Auswahlanweisung werden die betreffenden Zeilen der temporären Tabelle ausgewählt. Mit einem Cursor-Durchlauf werden die Daten dann in die Tabelle `above` zeilenweise übertragen.

Auch hier ist durch die Fremdschlüsseldeklaration bei der Tabellendefinition sichergestellt, daß keine Unter- bzw. Überführungen von unbekannten Objektnummern gespeichert werden.

5.2.6 attributes

Um die Objektattribute aus der tempoären in die endgültige Datenbank zu überführen, werden die Einträge der temporären Tabelle `t_attribute` zeilenweise in die Tabelle `attributes` überführt. Es werden nur Tupel übernommen, bei denen die Attribute in den Formatdatentabellen eingetragen sind. Ist eine Attributnummer unbekannt, so wird dieses Tupel nicht in die Datenbank übernommen.

Kapitel 6

Bedienung der Programme

6.1 Installation

Die Installation erfolgt unter Benutzung von `gmake`. Zur Installation wechseln Sie bitte in das Verzeichnis der Installationsdateien. Dort befindet sich auch die Datei `Makefile`. Durch den Aufruf

`make` *Installationsart* `U=`*Login* `D=`*Zielverzeichnis* `C=`*Classpath* `S=`*Source*

wird ein durch den Parameter bestimmter Teil des Programms installiert. Unter den Parameter *Login* können Sie Ihren Benutzernamen bei der Datenbank angeben. Ihr Paßwort wird bei den Installationsarten `fad`, `classes` oder `all` abgefragt.

Mit dem Parameter *Zielverzeichnis* wird das Verzeichnis angegeben, in das der Verzeichnisbaum installiert wird.

Unter `C=`*Classpath* können Sie angeben, im welchem Verzeichnis der Treiber für die Datenbankanbindeung zu finden ist.

Wenn Sie dieses Makefile aus einem anderen Ordner als dem Quellverzeichnis starten wollen, so können Sie den Pfad zum Quellverzeichnis relativ zum Pfad des aktuellen Verzeichnisses mit dem Parameter *Source* angeben.

Aus der folgenden Liste entnehmen Sie bitte, welche *Installationsart* Sie auswählen wollen:

- `doc`: Dieses Handbuch wird kopiert.
- `javadoc`: Die Quellcodes der Javaklassen des Parserpakets werden kopiert und die Dokumentation mittels `javadoc` erstellt.
- `doc-all`: Handbuch und `javadoc`-Dokumentation werden erstellt.
- `parser`: Die Quellcodes der Javaklassen des Parserpakets werden kopiert und die Javaklassen werden erstellt.

- `fad`: Die Quellcodes des Datenbankpakets werden kopiert und die Datenbank im Datenbanksystem installiert.
- `classes`: Datenbank und Parser werden installiert.
- `all`: Dokumentation und Programme werden installiert.

6.2 Parser

Um das Parserprogramm auszuführen, starten Sie die Javaklasse `ReadIt` im Verzeichnis `/classes/parser/`. Als Parameter können sie folgendes angeben:

1. Benutzername beim Datenbanksystem,
2. Paßwort beim Datenbanksystem,
3. EDBS-Datendateiname.

Parameter die nicht angegeben werden, werden vom Programm abgefragt. Wenn die zu bearbeitenden Daten in verschiedene Dateien aufgeteilt sind, haben Sie zwei Möglichkeiten, diese als ein Datenpaket in das Datenbanksystem aufzunehmen: Entweder erstellen Sie eine Datei, die aus den aneinandergehängten Dateien besteht, und lesen diese ein, oder Sie geben als Dateinamen eine entsprechende Anweisung an.

Die eventuell notwendigen Abfragen von Benutzername und Paßwort erfolgen in einem einfachen Fensterdialog; siehe auch Abbildung 6.1 und 6.2. Ist keine Datei angegeben, so erscheint ein Dateiauswahldialog, wie in Abbildung 6.3 gezeigt.

Abbildung 6.1: Fenster zur Eingabe des Benutzernamens

6.3 Datenbank

Wenn Sie die Konvertierung der Daten von den temporären Tabellen in die Zieldatenbank oder eine andere Prozedur bzw. Funktion des Packages, die `public`

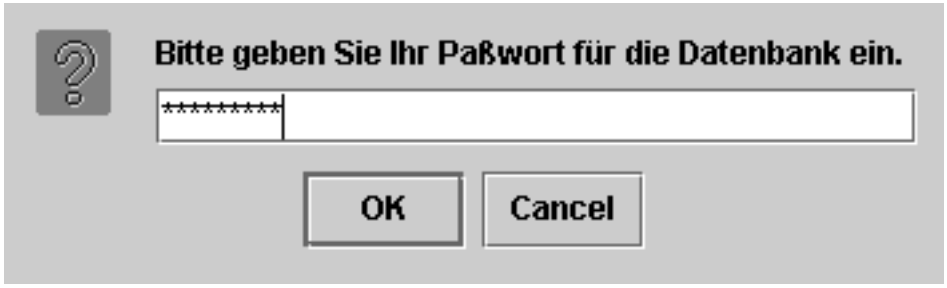

Abbildung 6.2: Fenster zur Eingabe des Paßwortes

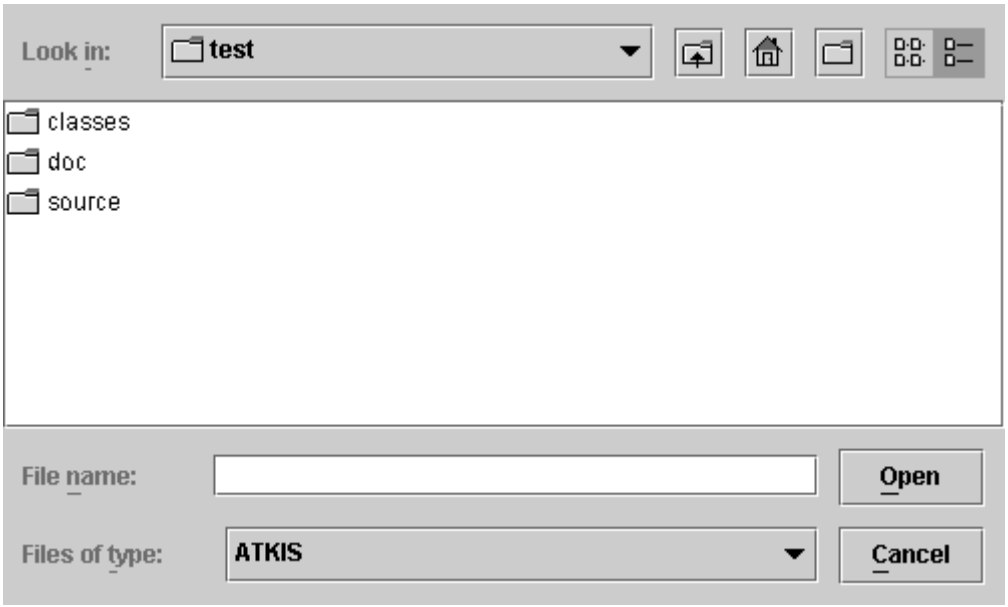

Abbildung 6.3: Fenster zur Auswahl der Datei

ist, aufrufen wollen, können Sie dies tun, indem Sie im SQLplus-Programm (start mit `sqlplus`)

`fad.`*Methode*(*Parameterliste*)

eingeben.

Wollen Sie etwa eine gesamte Datenbank löschen, bei der die Tabellennamen den Index `2` haben, so können Sie dies durch den Befehl

```
fad.drop_tables('2',10)
```

erledigen. Die `10` steht dabei für den internen Tabellennummer der Tabelle, bis zu der Sie die Tabellen löschen wollen. Haben die Tabellennamen keinen zusätzlichen Index, so geben Sie den Befehl

```
fad.drop_tables('',10)
```

ein.

Anhang A

Listings

A.1 Temporäre Datenbank

Hier werden die SQL-Kommandos aufgelistet, mit der die temporären Tabellen erstellt werden. Im Beispiel werden die Tabellen mit dem Index 1 erstellt. Die Dateinamen haben den leeren Index.

A.1.1 t_object

Tabellendefinition:

```
CREATE TABLE t_object1
( ObjNo     VARCHAR2(7) NOT NULL CONSTRAINT
                                    t_pk1_object PRIMARY KEY,
  ObjCoorX  NUMBER(9)   NOT NULL,
  ObjCoorY  NUMBER(9)   NOT NULL,
  Layer     VARCHAR2(3) NOT NULL,
  ObjRace   VARCHAR2(4) NOT NULL,
  Actual    VARCHAR2(2) NOT NULL,
  ObjType   VARCHAR2(1) NOT NULL,
  ModType   VARCHAR2(2) NOT NULL,
  BuildDate VARCHAR2(6) NOT NULL,
  Differnce VARCHAR2(1) NOT NULL  )
```

Kontrolldatei:

```
load data
infile 't_object.load'
into table t_object1
fields terminated by ','
(ObjNo,ObjCoorX,ObjCoorY,Layer,ObjRace,Actual,ObjType,
 ModType,BuildDate,Differnt)
```

A.1.2 t_object_data

Tabellendefinition:

```
CREATE TABLE t_object_data1
( ObjNo     VARCHAR2(7) NOT NULL  CONSTRAINT t_fk_1obj_data_object
                                    REFERENCES t_object1(ObjNo),
  ObjDataNo NUMBER      NOT NULL,
  InfoRace  VARCHAR(2)  NOT NULL,
  MapType   VARCHAR(2)  NOT NULL,
  SigNo     VARCHAR(6)  NOT NULL,
  Text      VARCHAR(33) NOT NULL,
  GeoRace   VARCHAR(2)  NOT NULL,
  ObjPartNo VARCHAR(3)  NOT NULL,
  CONSTRAINT t_pk_1object_data PRIMARY KEY (ObjNo, ObjDataNo) )
```

Kontrolldatei:

```
load data
infile 't_object_data.load'
into table t_object_data1
fields terminated by ','
(ObjNo,ObjDataNo,InfoRace,MapType,SigNo,Text,GeoRace,ObjPartNo)
```

A.1.3 t_object_geo

Tabellendefinition:

```
CREATE TABLE t_object_geo1
( ObjNo     VARCHAR2(7) NOT NULL,
  ObjDataNo NUMBER      NOT NULL,
  ObjParaNo NUMBER      NOT NULL,
  CoorX     NUMBER(9)   NOT NULL,
  CoorY     NUMBER(9)   NOT NULL,
  CONSTRAINT t_fk_1obj_geo_obj_data
                    FOREIGN KEY (ObjNo, ObjDataNo)
                    REFERENCES t_object_data1(ObjNo, ObjDataNo),
  CONSTRAINT t_pk_1object_geo
                    PRIMARY KEY (ObjNo, ObjDataNo, ObjParaNo) )
```

Kontrolldatei:

```
load data
infile 't_object_geo.load'
into table t_object_geo1
fields terminated by ','
(ObjNo,ObjDataNo,ObjParaNo,CoorX,CoorY)
```

A.1.4 t_line

Tabellendefinition:

```
CREATE TABLE t_line1
( LineNo    NUMBER      NOT NULL CONSTRAINT
                                  t_pk_1line PRIMARY KEY,
  CoordX    NUMBER(9)   NOT NULL,
  CoordY    NUMBER(9)   NOT NULL,
  EndPointX NUMBER(9)   NOT NULL,
  EndPointY NUMBER(9)   NOT NULL,
  Geometry  VARCHAR(2)  NOT NULL  )
```

Kontrolldatei:

```
load data
infile 't_line.load'
into table t_line1
fields terminated by ','
(LineNo,CoordX,CoordY,EndPointX,EndPointY,Geometry)
```

A.1.5 t_line_function

Tabellendefinition:

```
CREATE TABLE t_line_function1
( LineNo          NUMBER      NOT NULL CONSTRAINT
                                        t_fk_1l_func_line_no
                                  REFERENCES t_line1(LineNo),
  LineFunctionNo  NUMBER      NOT NULL,
  Layer           VARCHAR2(3) NOT NULL,
  LineRace        VARCHAR2(4) NOT NULL,
  ObjNoR          VARCHAR2(7) CONSTRAINT t_fk_1l_func_obj_no_r
                                  REFERENCES t_object1(ObjNo),
  ObjNoL          VARCHAR2(7) CONSTRAINT t_fk_1l_func_obj_no_l
                                  REFERENCES t_object1(ObjNo),
  ObjPartNoR      VARCHAR2(3),
  ObjPartNoL      VARCHAR2(3),
  LineDifR        VARCHAR2(1),
  LineDifL        VARCHAR2(1),
  CONSTRAINT t_pk_1line_function
                           PRIMARY KEY (LineNo, LineFunctionNo) )
```

Kontrolldatei:

```
load data
infile 't_line_function.load'
into table t_line_function1
fields terminated by ','
(LineNo,LineFunctionNo,Layer,LineRace,ObjNoR,ObjNoL,
 ObjPartNoR,ObjPartNoL,LineDifR,LineDifL)
```

A.1.6 t_line_param

Tabellendefinition:

```
CREATE TABLE t_line_param1
( LineNo          NUMBER      NOT NULL,
  LineFunctionNo  NUMBER      NOT NULL,
  LineParamNo     NUMBER      NOT NULL,
  Race            VARCHAR2(1) NOT NULL,
  Type            VARCHAR2(1) NOT NULL,
  Value           VARCHAR2(7) NOT NULL,
  CONSTRAINT t_fk_1line_param_line_function
            FOREIGN KEY (LineNo, LineFunctionNo)
            REFERENCES t_line_function1(LineNo, LineFunctionNo),
  CONSTRAINT t_pk_1line_param
            PRIMARY KEY (LineNo, LineFunctionNo, LineParamNo) )
```

Kontrolldatei:

```
load data
infile 't_line_param.load'
into table t_line_param1
fields terminated by ','
(LineNo,LineFunctionNo,LineParamNo,Race,Type,Value)
```

A.1.7 t_line_data

Tabellendefinition:

```
CREATE TABLE t_line_data1
( LineNo      NUMBER    NOT NULL CONSTRAINT t_fk_1l_data_line_no
                                     REFERENCES t_line1(LineNo),
  LineDataNo  NUMBER    NOT NULL,
  CoordX      NUMBER(9) NOT NULL,
  CoordY      NUMBER(9) NOT NULL,
  CONSTRAINT t_pk_1line_data PRIMARY KEY (LineNo, LineDataNo) )
```

Kontrolldatei:

```
load data
infile 't_line_data.load'
into table t_line_data1
fields terminated by ','
(LineNo,LineDataNo,CoordX,CoordY)
```

A.1.8 t_attribute

Tabellendefinition:

```
CREATE TABLE t_attribute1
( ObjNo       VARCHAR2(7) NOT NULL
                  CONSTRAINT t_fk_1attribute_object
                  REFERENCES t_object1(ObjNo),
  ObjPartNo   VARCHAR2(3) NOT NULL,
  Type        VARCHAR2(4) NOT NULL,
  Value       VARCHAR2(7) NOT NULL,
  CONSTRAINT t_pk_1attribute1
                  PRIMARY KEY (ObjNo, ObjPartNo, Type)
)
```

Kontrolldatei:

```
load data
infile 't_attribute.load'
into table t_attribute
fields terminated by ','
(ObjNo,ObjPartNo,Type,Value)
```

A.2 Formatdaten

A.2.1 object_race

Tabellendefinition:

```
CREATE TABLE layer
(  LayerNo VARCHAR2(3)   NOT NULL
                              CONSTRAINT pk_layer PRIMARY KEY,
   Name VARCHAR2(100)
);
```

Kontrolldatei:

```
load data
infile 'objectRace.load'
into table object_race
fields terminated by ','
(ObjRaceNo,Name)
```

A.2.2 layer

Tabellendefinition:

```
CREATE TABLE object_race
( ObjRaceNo VARCHAR2(4) NOT NULL    CONSTRAINT
                                    pk_object_race PRIMARY KEY,
  Name VARCHAR2(100)
);
```

Kontrolldatei:

```
load data
infile 'layer.load'
into table layer
fields terminated by ','
(LayerNo,Name)
```

A.2.3 attribute_types

Tabellendefinition:

```
CREATE TABLE attribute_types
( AttTypeNo VARCHAR2(4) NOT NULL    CONSTRAINT
                                    pk_attribute_type PRIMARY KEY,
  Name      VARCHAR(100)
);
```

Kontrolldatei:

```
load data
infile 'attType.load'
into table attribute_types
fields terminated by ','
(AttTypeNo,Name)
```

A.2.4 attribute_values

Tabellendefinition:

```
CREATE TABLE attribute_values
( Type      VARCHAR2(4) NOT NULL    CONSTRAINT fk_attribuet_type
                            REFERENCES attribute_types(AttTypeNo),
  Value     VARCHAR2(7) NOT NULL,
  Name      VARCHAR2(100),
  CONSTRAINT pk_attribute_value PRIMARY KEY (Type, Value)
);
```

Kontrolldatei:

```
load data
infile 'attTypeValue.load'
into table attribute_values
fields terminated by ','
(Type,Value,Name)
```

A.3 Datenbank

Die Kommandos zur Erstellung der Tabellen der Datenbank sind hier aufgeführt. Die Tabellennamen und Schlüsselbezeichner wurden dabei mit dem Index 1 versehen.

A.3.1 objects

```
CREATE TABLE objects1
( ObjNo         VARCHAR2(7) NOT NULL CONSTRAINT
                                     pk_1object PRIMARY KEY,
  LayerNo       VARCHAR2(3) NOT NULL CONSTRAINT fk_1object_layer
                               REFERENCES layer_tn1(LayerNo),
  ObjectRaceNo  VARCHAR2(4) NOT NULL CONSTRAINT fk_1object_race
                               REFERENCES object_race(ObjRaceNo),
  CoordinateX   NUMBER(9)   NOT NULL,
  CoordinateY   NUMBER(9)   NOT NULL,
  ModType       VARCHAR2(2) NOT NULL,
  BuildDate DATE
);
```

A.3.2 object_names

Der Typ einer Zeile der Tabelle mit den Geometrieangaben wird mit dieser Anweisung erstellt:

```
CREATE TYPE name_geo_entry_typ1 AS OBJECT
( CoordinateX   NUMBER(9),
  CoordinateY   NUMBER(9)
);
```

Der Typ der Tabelle mit den Geometrieangaben wird anschließend mit dieser Anweisung erstellt:

```
CREATE TYPE name_geo_table_typ1 AS TABLE OF name_geo_entry_typ1;
```

Bei der Erstellung der Tabelle wird dieser Tabellentyp als Spaltenobjekt benutzt:

```
CREATE TABLE object_name1
( ObjectNo  VARCHAR2(7) NOT NULL CONSTRAINT fk_1object_name
                                        REFERENCES object1(ObjNo),
  Type    VARCHAR2(2) NOT NULL,
  Value   VARCHAR2(31),
  Geo     name_geo_table_typ1',
  CONSTRAINT pk_1object_name PRIMARY KEY (ObjectNo, Type)
)
NESTED TABLE Geo STORE AS name_geo_table_1;
```

A.3.3 object_geometries

```
CREATE TABLE object_geometries1
( ObjectNo          VARCHAR2(7)     NOT NULL
                                    CONSTRAINT fk_1geo_object
                                    REFERENCES object_tn1(ObjNo),
  ObjectPartNo      VARCHAR2(7)     NOT NULL,
  ObjectGeometry    MDSYS.SDO_GEOMETRY,
  CONSTRAINT pk_1object_geometry
             PRIMARY KEY (ObjectNo, ObjectPartNo)
);
```

A.3.4 complex_objects

```
CREATE TABLE complex_objects1
( BigNo VARCHAR2(7)   NOT NULL CONSTRAINT fk_1comlpex_object_big
                               REFERENCES object1(ObjNo),
  SmallNo VARCHAR2(7) NOT NULL CONSTRAINT fk_1comlpex_object_small
                               REFERENCES object1(ObjNo),
  CONSTRAINT pk_1complex_object PRIMARY KEY (BigNo, SmallNo)
);
```

A.3.5 above

```
CREATE TABLE above1
( TopNo         VARCHAR2(7)     NOT NULL,
  TopPartNo     VARCHAR2(3)     NOT NULL,
  DownNo        VARCHAR2(7)     NOT NULL,
  DownPartNo    VARCHAR2(3)     NOT NULL,
  CONSTRAINT fk_1above_top FOREIGN KEY (TopNo, TopPartNo)
             REFERENCES object_geometry1(ObjectNo, ObjectPartNo),
  CONSTRAINT fk_1above_down FOREIGN KEY (DownNo, DownPartNo)
             REFERENCES object_geometry1 (ObjectNo, ObjectPartNo),
  CONSTRAINT pk_1above
             PRIMARY KEY (TopNo, TopPartNo, DownNo, DownPartNo)
);
```

A.3.6 attributes

```
CREATE TABLE attributes1
( ObjectNo VARCHAR2(7)  NOT NULL CONSTRAINT fk_1attribute_obj
                                    REFERENCES object1(ObjNo),
  ObjectPartNo VARCHAR2(7)  NOT NULL,
  Type VARCHAR2(4)      NOT NULL CONSTRAINT fk_1attribute_types
                         REFERENCES attribute_types(AttTypeNo),
  Value VARCHAR2(7)     NOT NULL,
  CONSTRAINT pk_1attribute
                PRIMARY KEY (ObjectNo, ObjectPartNo, Type, Value)
);
```

A.3.7 t_geo

Um die Geometrien der einzelnen linien- oder flächenförmigen Objkektteile zu berechnen, wird eine zusätzliche temporäre Tabelle angelegt.

```
CREATE TABLE t_geo1
( LineNo        NUMBER      NOT NULL,
  ObjPartNo     VARCHAR2(3) NOT NULL,
  BegX          NUMBER(9)   NOT NULL,
  BegY          NUMBER(9)   NOT NULL,
  EndX          NUMBER(9)   NOT NULL,
  EndY          NUMBER(9)   NOT NULL,
  Race          VARCHAR2(2) NOT NULL,
  Direction     NUMBER(1)   NOT NULL,
  MyNo          NUMBER,
  PolygonNo     NUMBER,
```

```
  CONSTRAINT pk_'||my_no||'t_geo PRIMARY KEY (LineNo, ObjPartNo)
);
```

A.4 Makefile

Zur Installation werden zwei Make-Dateien benutzt. Dies ermöglicht es, das Paßwort des Users bei der Datenbank nur einmal und zwar ohne dies am Bildschirm auszugeben, abzufragen. Zunächst die vom benutzer aufgerufene Datei:

```
A := `echo "input password: \c" >&2; stty -echo; sed q;
            stty echo; echo >&2`
D=/tmp
S=.
parser-name=parser
PL_SQL-name=PL_SQL
classes-directory=/classes
source-directory=/source
PL-name=PL_SQL
parser-directory=/$(parser-name)
fad-directory=/fad
PL-directory=/$(PL-name)
doc-directory=/doc
html-directory=/html
install-directory=$(S)

classpath=.:/databases/dbis/db-java/classes:/
            databases/dbis/3rdparty:
            $(root)$(source-directory)$(parser-directory)$(C)

sql-user=$(U)
root=$(D)

all : doc-all classes

classes : parser fad
    cp $(install-directory)/Makefile $(root)
    cp $(install-directory)/Makefile_Format $(root)

fad : mkdir-fad
    cp $(install-directory)$(fad-directory)/*.sql
            $(root)$(source-directory)$(fad-directory)
    cp $(install-directory)$(fad-directory)/*.load
            $(root)$(source-directory)$(fad-directory)
    cp $(install-directory)$(fad-directory)/*.control
            $(root)$(source-directory)$(fad-directory)
```

```
    $(MAKE) -sf Makefile_Format U=$(sql-user)
        D=$(root)$(source-directory)$(fad-directory) p=$(A)

mkdir-fad : $(root)$(source-directory)$(fad-directory)
$(root)$(source-directory)$(fad-directory):
    mkdir -p $(root)$(source-directory)$(fad-directory)

parser : mkdir-parser
    cp $(install-directory)$(parser-directory)/*.java
            $(root)$(source-directory)$(parser-directory)
    cp $(install-directory)/ClearData
            $(root)$(classes-directory)
    CLASSPATH=$(classpath) javac -d $(root)$(classes-directory)
        $(root)$(source-directory)$(parser-directory)/*.java

mkdir-parser : $(root)$(source-directory)$(parser-directory)
                $(root)$(classes-directory)$(parser-directory)
                $(root)$(source-directory)$(parser-directory):
    mkdir -p $(root)$(source-directory)$(parser-directory)
$(root)$(classes-directory)$(parser-directory):
    mkdir -p $(root)$(classes-directory)$(parser-directory)

doc-all : javadoc doc

javadoc : mkdir-javadoc
    cp $(install-directory)$(parser-directory)/*.java
            $(root)$(source-directory)$(parser-directory)
    cp $(install-directory)$(PL-directory)/*.java
            $(root)$(source-directory)$(PL-directory)/.java
    cd $(root)$(source-directory); javadoc -author -d
            $(root)$(doc-directory)$(html-directory) -private
            -sourcepath ./ $(parser-name) $(PL-name)

mkdir-javadoc :    $(root)$(doc-directory)$(html-directory)
            $(root)$(source-directory)$(parser-directory)
            $(root)$(source-directory)$(PL-directory)
$(root)$(doc-directory)$(html-directory):
    mkdir -p $(root)$(doc-directory)$(html-directory)
$(root)$(source-directory)$(parser-directory):
    mkdir -p $(root)$(source-directory)$(parser-directory)
$(root)$(source-directory)$(PL-directory):
    mkdir -p $(root)$(source-directory)$(PL-directory)


doc : mkdir-doc
```

```
    cp $(install-directory)$(doc-directory)/*
            $(root)$(doc-directory)

mkdir-doc :    $(root)$(doc-directory)
$(root)$(doc-directory):
    mkdir -p $(root)$(doc-directory)

clear-source :
    rm -rf $(root)$(source-directory)

clear-doc :
    rm -rf $(root)$(doc)

clear :
    rm -rf $(root)
```

Diese zweite Datei wird von `gmake` aufgerufen. Hier wird nach dem Paßwort gefragt:

```
D=.

default:
    sqlplus $(U)/$(p) @$(D)/create
    sqlldr USERID=$(U)/$(p)
                    CONTROL=$(D)/objectRace.control
                    DATA=$(D)/objectRace.load
                    ERRORS=999 SILENT=HEADER,FEEDBACK
    sqlldr USERID=$(U)/$(p)
                    CONTROL=$(D)/layer.control
                    DATA=$(D)/layer.load
                    ERRORS=999 SILENT=HEADER,FEEDBACK
    sqlldr USERID=$(U)/$(p)
                    CONTROL=$(D)/attType.control
                    DATA=$(D)/attType.load
                    ERRORS=999 SILENT=HEADER,FEEDBACK
    sqlldr USERID=$(U)/$(p)
                    CONTROL=$(D)/attTypeValue.control
                    DATA=$(D)/attTypeValue.load
                    ERRORS=999 SILENT=HEADER,FEEDBACK
```

A.5 Sonstiges

Hier noch kurze Anmerkungen zu zwei zusätzliche Dateien.

A.5.1 ClearData

Durch das Ausführen der Datei `ClearData` werden alle Dateien tempoären Dateien gelöscht.

```
rm *.load *.control *.bad *.log make_db.sql -f;
```

A.5.2 readme

In der Datei `readme` stehen Bemerkungen zur Installation. Da es sich fast wörtlich um das Kapitel 6.1 handelt, ist hier auf ein Listing verzichtet worden.

Anhang B

Abkürzungsverzeichnis

AdV: Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland
im Internet: `http://www.adv-online.de`

ALK: Automatisierte Liegenschaftskarte
Informationssystem zu geographischen Daten; eingeführt 1979

ATKIS: Amtliches Topographisch-Kartographisches Informationssystem
1989 beschlossen von der AdV
im Internet: `http://www.atkis.de`

DLM: Digitales Landschaftmodell
aufgebaut für verschiedene Maßtäbe

EDBS: Einheitliche Datenbankschnittstelle
in diesem Format werden die ATKIS-Daten gespeichert

javadoc: Programm zur Erstellung einer Dokumentation aus kommentierten java-Code
im Internet: `http://java.sun.com/products/jdk/javadoc/`

JDBC: Java Database Connectivity
Java-Paket `java.sql`
bietet Schnittstelle zu Datenbanken
im Internet: `http://java.sun.com/products/jdbc/`

NBZ: Numerierungsbezirk

Oracle8*i*: Datenbankprogramm der Oracle Corporation
Versionsnummer: 8.1.5.1.0
im Internet: `http://www.oracle.com`

SQL*loader: ein Programm im Packet Oracle8*i*
erlaubt schnellen Datenimport von grossen Paketen in eine leere Tabelle

WDF: Wiederholungsfaktor

Literaturverzeichnis

[Cor99a] Oracle Corporation. *Dokumentation zu Oracle 8i: Oracle 8i Spatial User's Guide and Reference*, Februar 1999.

[Cor99b] Oracle Corporation. *Dokumentation zu Oracle 8i: PL/SQL User's Guide and Reference*, Februar 1999.

[Har94] Rolf Harbeck (Hg.). *Das Geoinformationssystem ATKIS und seine Nutzung in Wirtschaft und Verwaltung*. Landesvermessungsamt Nordrhein-Westfalen, 1994.

[Koc94] Hans Koch. *Entwurf und Implementierung eines Informationssystems für ATKIS–Daten*. Diplomarbeit, Inst. für Programmiersprachen und Informationssysteme Abt. DB, TU Braunschweig, Braunschweig, Dezember 1994.

[Kor97] Karsten Korte. *Transformation von ATKIS-Daten in eine relationale Datenbank zur Verkehrsplanung*, 1997.

[Lan95] *ATKIS-Gesamtdokumentation*. Technischer Bericht, November 1995.

[NK95] Karl Neumann, Hans Koch. *Ein experimentelles Informationssystem für ATKIS-Daten. Nachrichten aus dem Karten- und Vermessungswesen*, (113):Seiten 179–190, 1995.

[RRZ97] *Java*. RRZN-Klassifikation: SPR.JAV 2, 1997.

[RRZ99] RRZN. *Datenbanken und SQL*. RRZN - Universität Hannover, Hannover, 1. Auflage, Mai 1999.

[Wal97] Volker Walter. *Zuordnung von raumbezogenen Daten - am Beispiel der Datenmodelle ATKIS und GDF*. Dissertation, Universität Stuttgart, Januar 1997.