Fachbereich **Automatisierung und Informatik**	HOCHSCHULE harz Hochschule für angewandte Wissenschaften (FH) Wernigerode

„Datenbank – Grundlagen“

Entwurfsarbeit

Entwicklung einer Datenbank
zur Reservierung
von Sitzplätzen in einer Eisenbahn

WS 2004 / 05

Dipl. Inf., Dipl.-Ing. (FH) Michael Wilhelm
Friedrichstraße 57 - 59
38855 Wernigerode

Raum: 2.202
Tel.: 03943/659-338
Fax: 03943/659-399
Email: mwilhelm@hs-harz.de

Inhaltsverzeichnis

1 Abbildungsverzeichnis

2 Pflichtenheft / Aufgabenstellung

Entwicklung einer Datenbank zur Verwaltung von Reservierungen einer Eisenbahn.

Die vielfach diffusen Anforderungen an eine Datenbankapplikation müssen in Worte gefasst werden. Dabei sind wichtige Zusammenhänge und Vorgaben klar zu definieren.

Einschränkungen / Eigenschaften:

- Zugverkehr nur im Inland.
- Jeder Zug hat nur ein Waggon mit n-Sitzplätzen
- Strecken sind über einzelne Bahnhöfe definiert (Segmente).
- Der Kunden kann auf jedem einzelnen Bahnhof aussteigen.
- Jede Strecke besteht aus mehreren Segmenten.
- Der Preis wird pro Segment festgelegt.
- Jeder Kunde kann mehrere Reservierungen vornehmen. Es gilt nur die Anzahl. Aber es gibt nur eine Ansprechperson.
- Züge können mehrfach pro Tag pro Strecke fahren.
- Es gibt nur eine Klasse.
- Es gibt keine Sitzplatz-Verwaltung.
- Der Sitzplatz kann aber mehrfach reserviert werden (keine Überschneidung).
- Alle Tabellen müssen sich in der dritten Normalform befinden.

3 Konzeptionelles Modell

3.1 Entities:

Aus der Aufgabenstellung sind Entitätsmengen zu bilden, um so eine Gruppierung nach gewissen Eigenschaften herbeizuführen. bei überlappenden Entitätsmengen sind die entsprechenden umfassenden Entitätsmengen zu bilden. In diesem Kapitel werden alle Entities mit Primärschlüssel beschrieben.

Verwendete Entities:

- Kunde (#Knr)
- Bahnhof (#BNr)
- DB_Fahrt (#FNr)
- Waggon (#WgNr)
- Lok (#LokNr)
- Segmente (#SegNr)
- Strecken (#StreckenNr)

Die Entities „Lok“ und „Waggon“ sind leicht zu verstehen. Sie werden für eine einzelne Fahrt benötigt. Dabei hat ein Zug nur einen Waggon mit einer Klasse. Es werden jedem Waggon n-Sitzplätze zugeordnet. Eine DB-Fahrt besteht aus einem Start- und Zielbahnhof (z. B. HH nach FFM) und kann z. B. um 08:00 und 09:00 starten. Dementsprechend hat eine DB-Fahrt eine Strecke, eine Startzeit und ein Datum. Ein Kunde kann nun jede einzelne Fahrt mehrfach buchen (n×m-Beziehung).

Eine Strecke ist aus mindestens ein Segment aufgebaut. Ein Segment ist die kleinste Strecken-Einheit. Sie besteht aus zwei Bahnhöfen.

In diesem Kapitel werden alle möglichen Beziehungen zwischen den Entitäten festgehalten, wobei auch „nicht-hierarchische“ Beziehungen zulässig sind. Unklare Beziehungen sind weiter zu verfeinern.

Die Abbildung 1 zeigt die Entities mit ihren Beziehungen.

3.2 Beziehungen

Die Beziehungen „LOK“, „WAGGON“ wurden schon im letzten Kapitel erläutert. Da eine Strecke mehrmals am Tag abgefahren werden kann, muss die Strecke in einem separatem Entity verwaltet werden. Als Schlüssel wird die Streckennummer eingeführt. Ersatzweise wären auch zwei Bahnhöfe (B1, B2) möglich.

Die Entities „STRECKE“, „DB_SEGMENT“ und Bahnhöfe haben eine sehr komplexe Beziehung. Eine Strecke hat eine Start- und Zielbahnhof und besteht aus mehreren Segmenten. Diese wiederum bestehen aus zwei Bahnhöfen. Für das logische Modell existieren mehrere Lösungen dieses Problems.

Der Kunde und die einzelne Fahrt haben eine M:N-Beziehung. Diese wird dann im logischen Modell durch eine weitere Beziehung, „Reservierung“ ersetzt.

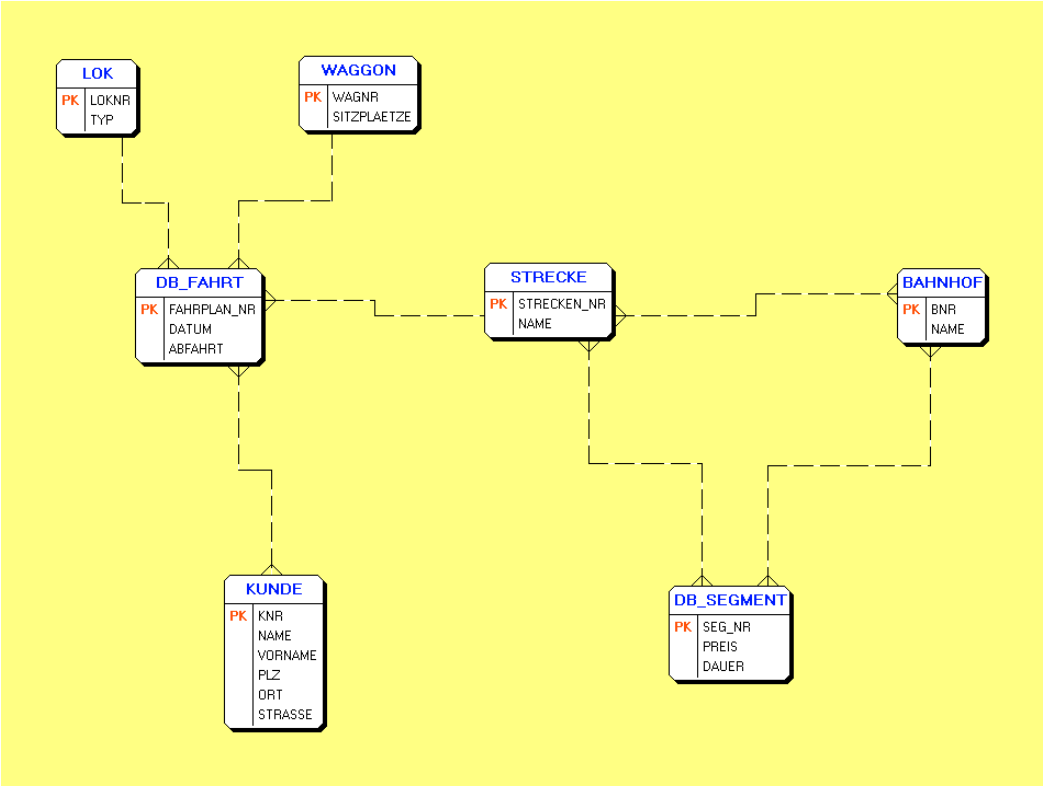

Abbildung 1 Beziehungen zwischen den Entities

4 Logisches Modell

Das logische Modell wird aus dem konzeptionellen Modell entwickelt. Alle hierarchischen Beziehungen bleiben erhalten, alle anderen werden durch Zwischenbeziehungen erweitert. Damit ergibt sich folgende Abbildung 2. De Bezeichnungen wurden vom Programm vorgegeben und können verändert werden.

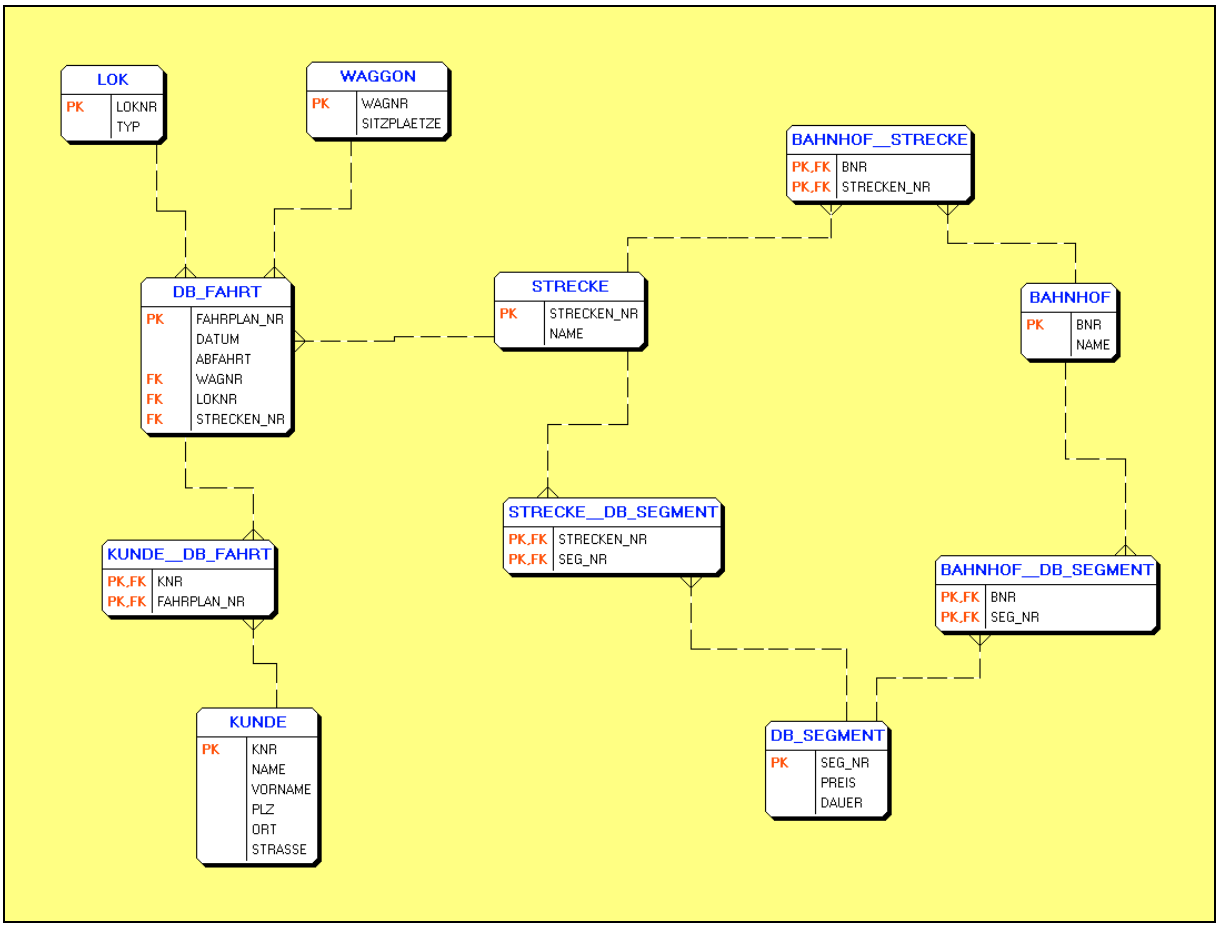

Abbildung 2 Logisches Modell

Die Namen der neuen Beziehungen werden aus den Kurznamen abgeleitet. Dabei sind die Beziehungen zwischen den Entities „Strecke“, „Segment“ und Bahnhof nicht korrekt abgebildet. Die nächsten Kapitel zeigen die verschiedenen Möglichkeiten. Diese Änderung der automatischen Generierung ist bei diesem komplexen Beispiel normal. Bei einfachen Beispielen müssen keine Änderungen vorgenommen werden. Um die Transformation zu vereinfachen, kann man auch die kritischen Beziehungen im konzeptionellen Modell löschen und diese Beziehungen manuell im logischen Modell einbauen.

Die Abbildung 3 zeigt die eingefügten Kurznamen. Die neue Beziehung, das neue Entity wird aus den Namen der beiden Entities gebildet. In der unteren Abbildung sind alle Relationen agezeigt. Der erste Eintrag „STRECKE__STRK_SEGMT“ ist die Beziehung zwischen dem Entity „Srecke“ und dem neuem Entity „STRK_SEGMT“. Für die Namensgebung werden die Kurznamen verwendet.

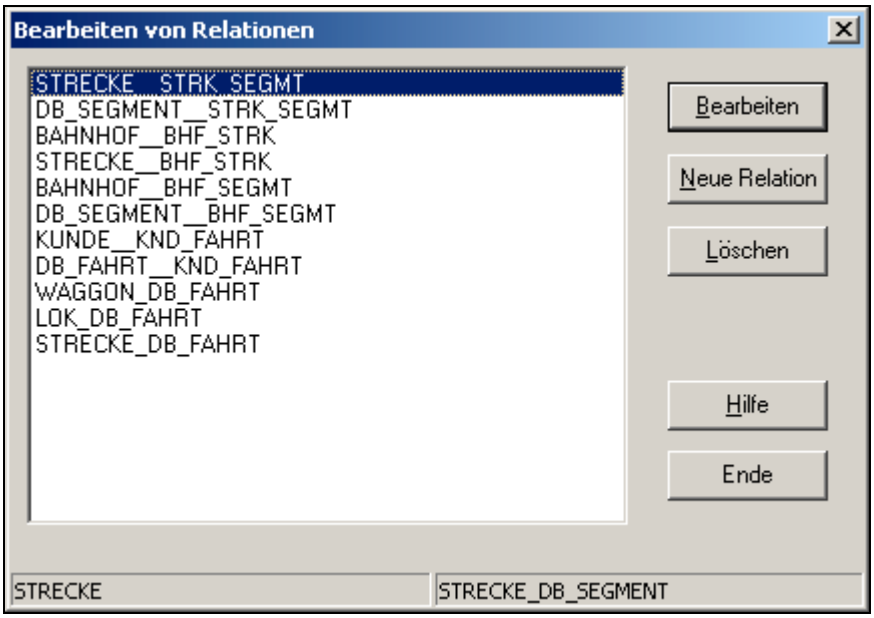

Abbildung 3 Kurznamen

Die folgenden Kapitel beschreiben Lösungen, um die Beziehungen zwischen den Entities „Strecke“, „Segment“ und Bahnhof zu ermölichen. Dabei ist entscheidend, dass sowohl das Entity Strecke, als auch das Entity Segment jeweils eine Beziehung zu zwei Bahnhöfen haben.

4.1 Zwei externe Tabellen, zwei Einträge

Die Bahnhöfe werden nach theoretischer Grundlage in ein Extra-Entity gespeichert. Dabei muss aber bedacht werden, dass eine Strecke aus zwei Bahnhöfen besteht. Dementsprechend existieren zwei Einträge pro Strecke im Entity „Strecke_Bahnhof“. Unterschieden werden diese, indem ein zusätzliches Attribut „Abfahrsbahnhof“ eingefügt wurde. Die Abfrage ist bei dieser Lösung komplizierter, drei Entities betroffen sind. Dieselbe Technik wird nun auch bei der Beziehung „DB_Segment“, „Segmente“ und „Bahnhof“.

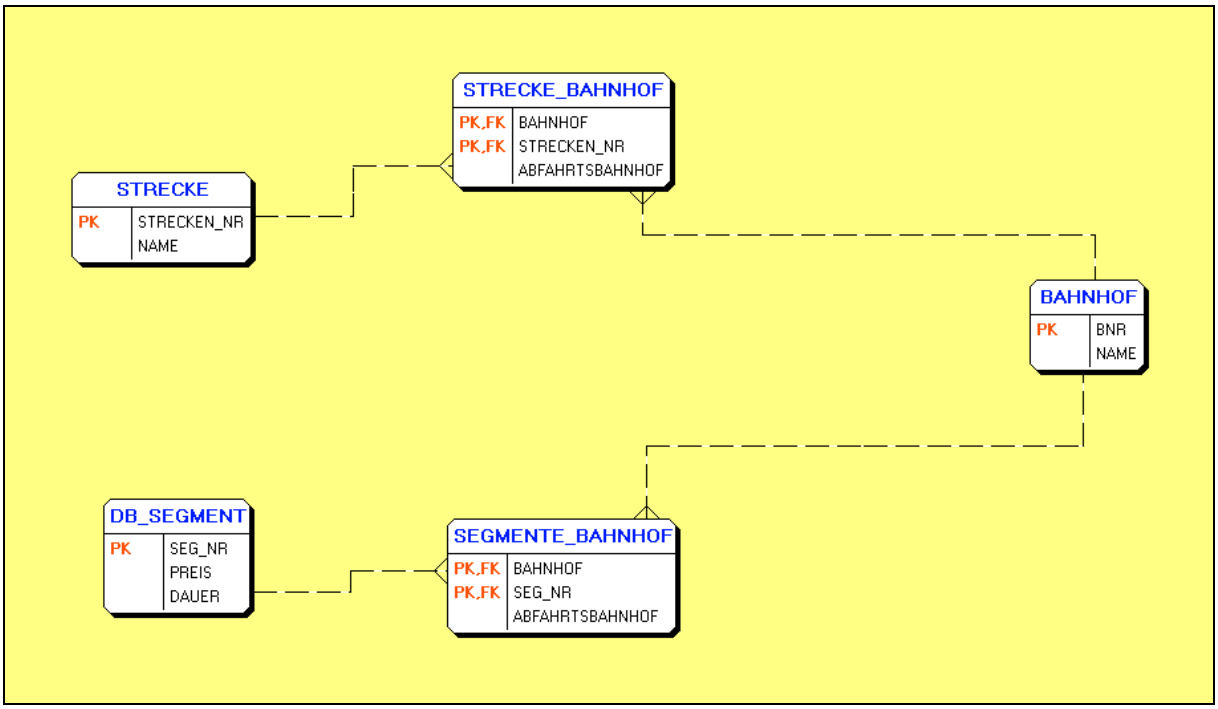

Abbildung 4 Beziehung Strecke, Segmente, Bahnhof

Ein weiterer Nachteil ist das Eingeben der Daten in die Datenbank. Die zusätzliche Tabelle birgt die Gefahr, das nur eine Beziehung eingegeben wird.

DDL-Code:

```
CREATE TABLE STRECKE (
  STRECKEN_NR  INTEGER  NOT NULL,
  NAME  VARCHAR(40),

  PRIMARY KEY (STRECKEN_NR)
);


CREATE TABLE BAHNHOF (
  BNR  INTEGER  NOT NULL,
  NAME  VARCHAR(40),

  PRIMARY KEY (BNR)
);


CREATE TABLE DB_SEGMENT (
  SEG_NR  INTEGER  NOT NULL,
  PREIS  NUMERIC(7,2)  NOT NULL,
  DAUER  NUMERIC(7,2),

  PRIMARY KEY (SEG_NR)
);


CREATE TABLE STRECKE_BAHNHOF (
  BAHNHOF  INTEGER  NOT NULL,
  STRECKEN_NR  INTEGER  NOT NULL,
  ABFAHRTSBAHNHOF  BOOL  NOT NULL,

  PRIMARY KEY (BAHNHOF, STRECKEN_NR )
);

CREATE TABLE SEGMENTE_BAHNHOF (
  BAHNHOF  INTEGER  NOT NULL,
  SEG_NR  INTEGER  NOT NULL,
  ABFAHRTSBAHNHOF  BOOL  NOT NULL,

  PRIMARY KEY (BAHNHOF, SEG_NR )
);
```

Beispiel für Daten in der Datenbank:

BNr	**Name**
1	Hamburg
2	Hannover
3	Braunschweig
4	Magdeburg
5	Berlin
6	Frankfurt / Main
7	München

Tabelle Bahnhof

StreckenNr	Name
10	Hamburg-Berlin
20	Hamburg-Frankfurt
30	Frankfurt-München

Tabelle Strecke

BahnhofsNr	StreckenNr	Abfahrtsbahnhof
1	10	TRUE
5	10	FALSE
6	20	FALSE
1	20	TRUE
6	30	TRUE
7	30	FALSE

Tabelle Strecke_Bahnhof

In Tabelle „Strecke_Bahnhof“ sieht man die Gefahr, dass man einen Bahnhof beim Eintragen „vergessen“ kann. Über ein Dialogfenster mit zwei gleichzeitigen Eingaben kann man diese Gefahr eliminieren.

4.2 Zwei externe Tabellen, ein Eintrag

Die Bahnhöfe werden wiederum in ein Extra-Entity gespeichert. Hier werden aber die beiden Einträge in einen Datensatz zusammen gefasst. Die Abfragen sind nun etwas einfacher. Die Primärschlüssel sind jeweils die Streckennummer und die Segmentnummer.

DDL-Code:

```
CREATE TABLE STRECKE_BAHNHOF (
  STRECKEN_NR  INTEGER  NOT NULL,
  BAHNHOF1  INTEGER  NOT NULL,
  BAHNHOF2  INTEGER  NOT NULL,

  PRIMARY KEY (STRECKEN_NR )
);



CREATE TABLE SEGMENTE_BAHNHOF (
  SEG_NR  INTEGER  NOT NULL,
  BAHNHOF1  INTEGER  NOT NULL,
  BAHNHOF2  INTEGER  NOT NULL,

  PRIMARY KEY (SEG_NR )
);
```

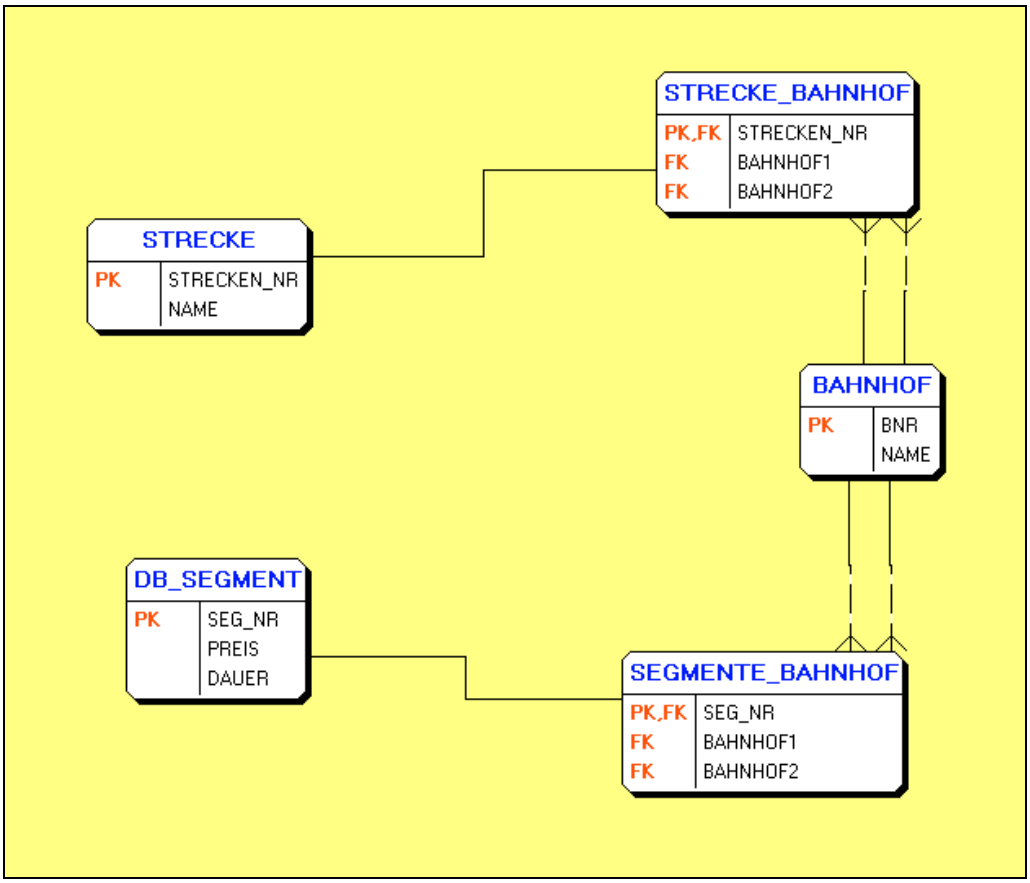

Abbildung 5 Beziehung Strecke, Segment, Bahnhof

Abbildung 5 zeigt die Beziehungen für die gewählte Lösung. Hervorzuheben ist hierbei, dass die Beziehungen der Entities „Strecke:Bahnhof“ und „Bahnhof“ aus **zwei** Beziehungen bestehen. Diese neuen Fremdbeziehungen sind aber notwendig, um Sicherheiten gegenüber dem Eintragen und Löschen zu haben. Das Programm wurde angepasst, damit diese beiden Linien darstellt werden können.

Beispiel für Daten in der Datenbank:

BNr	Name
1	Hamburg
2	Hannover
3	Braunschweig
4	Magdeburg
5	Berlin
6	Frankfurt / Main
7	München

Tabelle Bahnhof

StreckenNr	Name
10	Hamburg-Berlin
20	Hamburg-Frankfurt
30	Frankfurt-München

Tabelle Strecke

StreckenNr	Bahnhof1	Bahnhof2
10	1	5
20	1	6
30	6	7

Tabelle Strecke_Bahnhof

In Tabelle „Strecke_Bahnhof“ sind nun pro Tupel beide Bahnhöfe eingetragen.

Abbildung 5 zeigt die Beziehungen für die gewählte Lösung. Dabei ist hervorzuheben, dass beide Beziehungen (Strecke zu Strecke_Bahnhof, Segment zu Segment_Bahnhof) jeweils eine 1:1 Beziehung ist. Dementsprechend kann man diese Beziehung ersatzlos in die Entites „Strecken“, „Segmente“ einfügen.

Damit ergibt sich folgende endgültige Lösung:

4.3 Keine Zwischentabelle

Die Bahnhöfe, Strecken und Segmente werden in jeweils ein Entity gespeichert. Die Beziehungen werden in den Entities „Strecken“ und „DB_Segment“ eingefügt. Die Abfragen sind nun noch etwas einfacher. Die Primärschlüssel sind jeweils die Streckennummer und die Segmentnummer. Die doppelten Beziehungen zwischen „Strecke“ und „Bahnhof“ bzw. „DB_Segment“ und Bahnhof bleiben.

Um die Beziehungen mittels Designer zu definieren, ruft man den Eintrag „Einfügen,Beziehungen“ auf. Das Quell-Entity ist der „Bahnhof.“ Das Ziel-Entity ist die „Strecke“.

Die Abbildung 6 zeigt die Einstellungen. Die automatische Verknüpfung der Attribute zwischen den Entities kann hier nicht funktionieren, da zwei Attribute den Fremdschlüssel übernehmen müssen. Deshalb wechseltman auf das Register „Link-Attribute“ und verneint die Abfrage (siehe Abbildung 7).

In Register werden alle Attribute der beiden Attribute angezeigt. Nun muss man diese manuell verbinden.

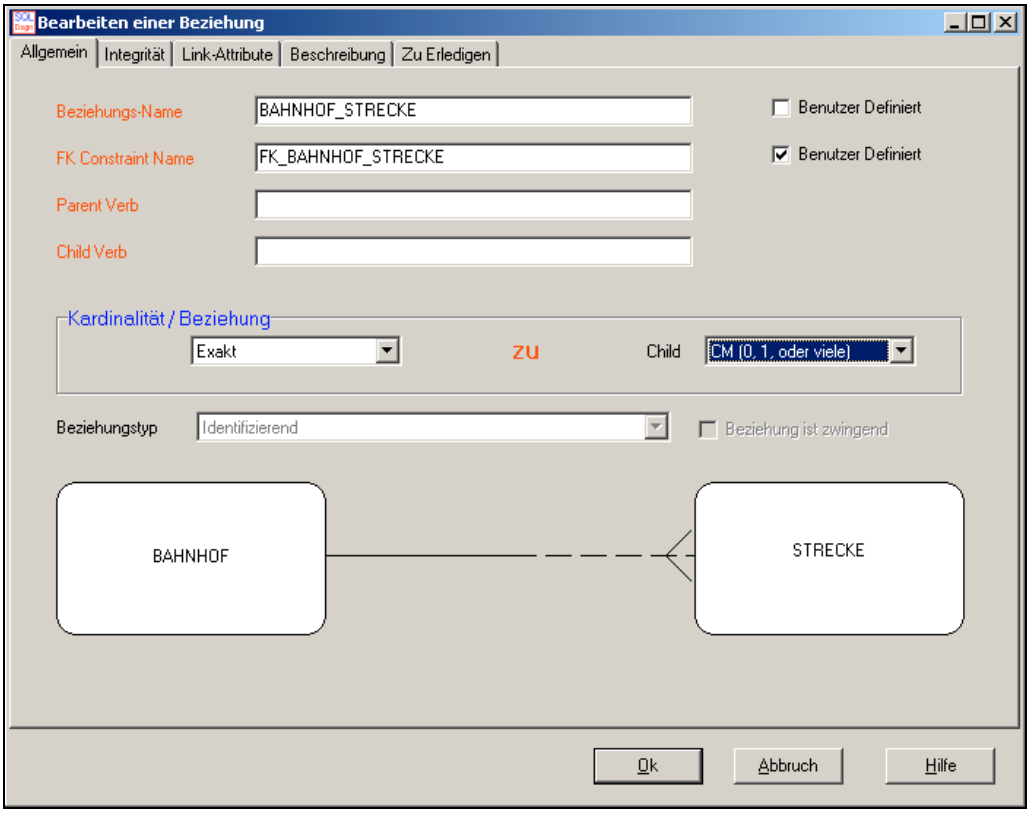

Abbildung 6 Beziehung definieren

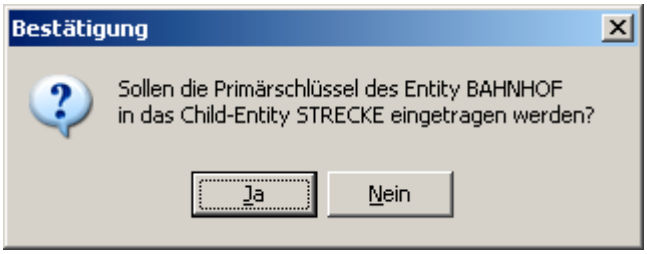

Abbildung 7 Automatisches Generieren des Fremdschlüssels

Mit dieser Abfrage (Abbildung 7) kann man die automatische Übertragung der Fremdschlüssel starten. Beim Betätigen des Schalters „Nein“ kann man die Fremdschlüssel manuell bestimmen. Dies ist dann wichtig, wenn man die Attribute schon im Ziel-Entity hat.

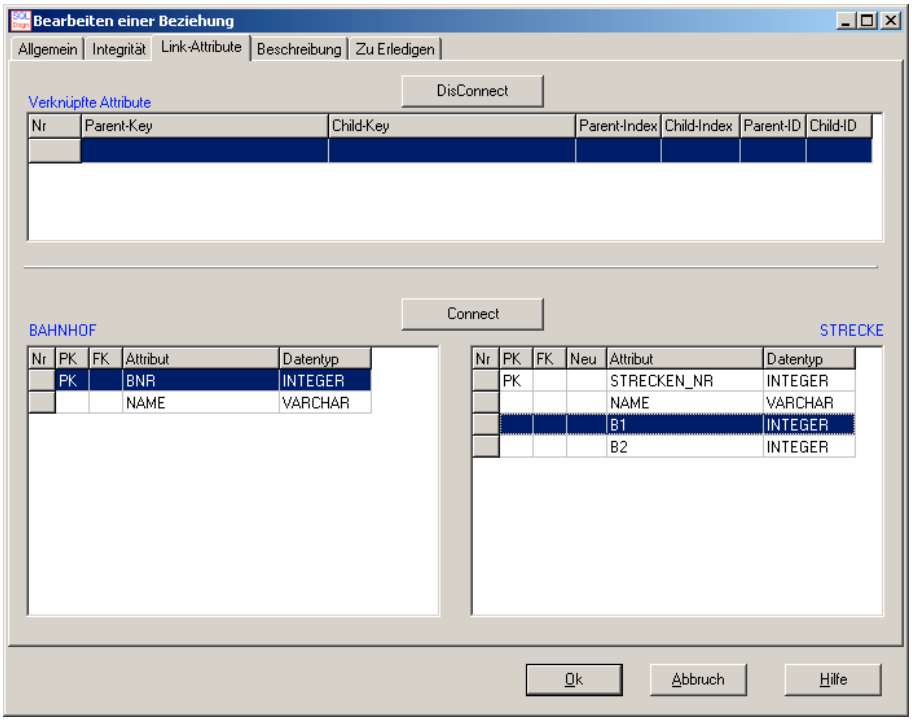

Abbildung 8 Manuelles Erstellen einer Beziehung

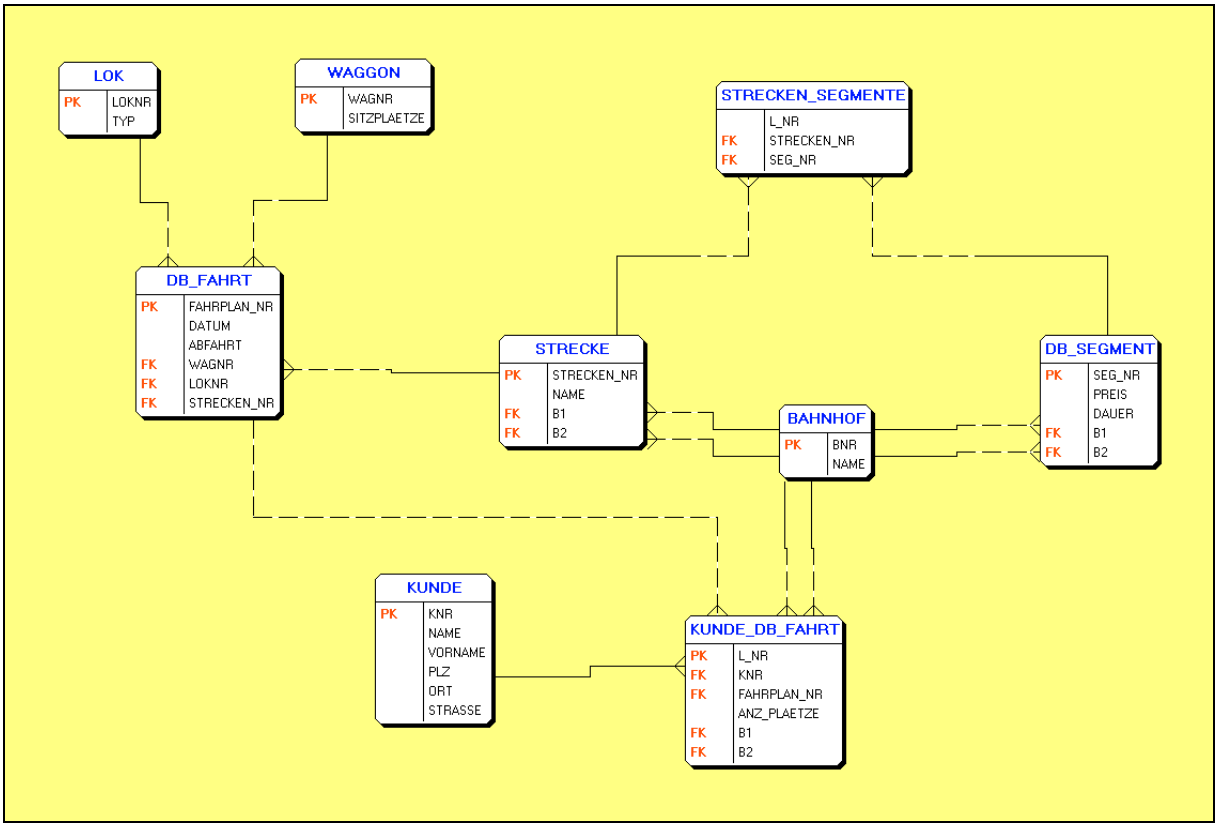

Abbildung 9 Logisches ERM-Modell

<u>Hinweis:</u>

Die Namen der Beziehungen werden aus den Name der Entities gebildet. Damit entstehen jeweils zwei gleichnamige Beziehungen für die Beziehungen:

- BAHNOF zu DB_FAHRT
- BAHNOF zu DB_SEGMENT
- BAHNOF zu STRECKE

Bei allen sechs Beziehungen muss man den Namen und den Fremdschlüsselnamen ändern. Am Besten in BAHNHOF_STRECKE1 und in BAHNHOF_STRECKE2 etc (siehe Abbildung 10).

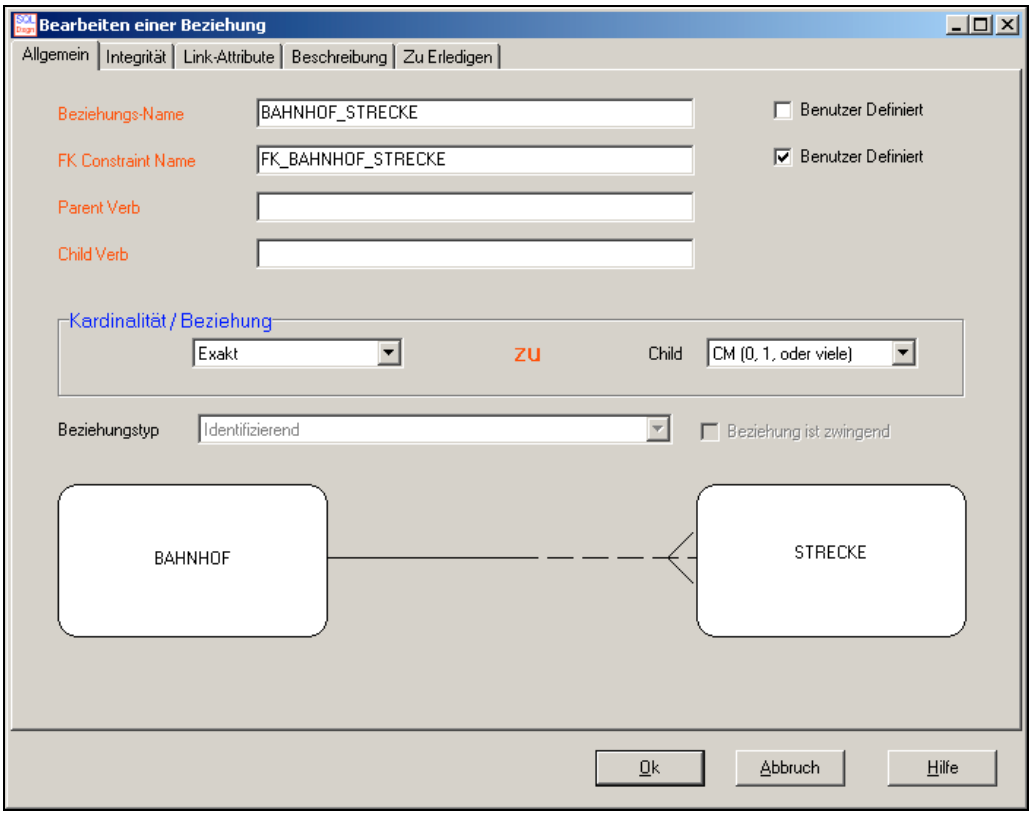

Abbildung 10 Beziehungsname ändern

Die obere Abbildung zeigt eine der sechs Beziehungen. Geändert werden muss der Beziehungsname in „BAHNHOF_STRECKE1“ und der Fremdschlüsselname in „FK_BAHNHOF_STRECKE1“. Welches Attribut die aktuelle Beziehung hat, erfährt man im Register „Link-Attribute“.

Als neue Beziehung wurde hier das Entity „Strecken_Segmente“ eingeführt. Damit werden Segmente auf die einzelnen Strecken verteilt. Da in einer Datenbank die einzelnen Tupel nicht sortiert sind, muss eine zusätzliche Laufende Nummer pro Strecke eingeführt werden.

Beispiel für Daten in der Datenbank:

BNr	Name
1	Hamburg
2	Hannover
3	Braunschweig
4	Magdeburg
5	Berlin
6	Göttingen
7	Fulda
8	Frankfurt / Main
9	Würzburg
10	Nürnberg
11	München

Tabelle Bahnhof

Strecken_Nr	Name	Bahnhof1	Bahnhof2
10	Hamburg-Berlin	1	5
20	Hamburg-Frankfurt	1	8
30	Frankfurt-München	8	11

Tabelle Strecke

Seg_Nr	Name	Bahnhof1	Bahnhof2
100	Hamburg-Hannover	1	2
101	Hannover-Braunschweig	2	3
102	Braunschweig-Magdeburg	3	4
103	Magdeburg-Berlin	4	5
104	Hannover-Göttingen	2	6
105	Göttingen-Fulda	6	7
106	Fulda-Frankfurt / Main	7	8
107	Frankfurt / Main-Würzburg	8	9
108	Würzburg-Nürnberg	9	10
109	Nürnberg-München	10	11

Tabelle DB_Segment

Durch die Fremdschlüsselbeziehung ist in dieser Tabelle gewährleistet, dass alle Bahnhöfe in der Tabelle „Strecke“ vorhanden sind. Das setzt aber voraus, das alle Bahnhöfe mindestens in einem Segment sind, also auch angefahren werden können.

Strecken_Nr	Segment_Nr	Laufende_Nr
10	100	1
10	101	2
10	102	3
10	103	4
20	100	1
20	105	3
20	104	2
20	106	4
30	107	1
30	108	2
30	109	3

Tabelle Strecken_Segmente

Die Nummern sind so gewählt, dass immer eindeutig ist, welche Tabelle gemeint ist. Des Weiteren wurden in der Tabelle nicht alle Datensätze in der richtigen Reihenfolge eingetragen. Dies ist beabsichtigt, da es in einer Datenbank keine Reihenfolge gibt.

Im nächsten Kapitel werden alle Primär- und Fremdschlüssel bestimmt.

4.4 Definition der Identifikationsschlüssel und Attribute

Für jede Entitätsmenge ist ein natürlicher oder künstlicher ID-Schlüssel festzulegen. Bei künstlichen ID-Schlüsseln wird ein neues Attribut eingeführt. Bei natürlichen ID-Schlüsseln wird ein bestehendes Attribut verwendet. Des Weiteren werden Primär- und Fremdschlüssel bestimmt. In dieser Notation wird der Primärschlüssel mit dem Zeichen (#) versehen. Ein Fremdschlüssel wird unterstrichen.

4.4.1 KUNDE

#KNr
Name
Vorname
PLZ
Ort
Straße

4.4.2 BAHNHOF
#BNr
Name

4.4.3 WAGGON
#WagNr
Sitzplaetze

4.4.4 LOK
#LokNr
Typ

4.4.5 DB_FAHRT
#FNr,
LokNr
WagNr
StreckenNr
Datum (Ein Jahr)
Abfahrts(zeit)

4.4.6 DB_SEGMENT
#SegNr
Bahnhof1
Bahnhof2
Preis
Dauer

4.4.7 STRECKE
#StreckenNr
Name
Bahnhof1
Bahnhof2

4.4.8 STRECKEN_SEGMENTE
#StreckenNr
#SegNr
LaufendeNr

4.4.9 Kunden_DB_Fahrt
#L_Nr
KNr
FNr
Bahnhof1
Bahnhof2
Anzahl_Plaetze

Bemerkungen:

In der Tabelle Kunden_DB_Fahrt bzw. Reservierungen wurde eine laufende Nummer eingefügt. Dieses könnte auf den Bescheinigungen gedruckt werden. Damit ist es auch möglich, dass ein Kunde mehrere Reservierungen vornehmen kann.

5 Datentypen der Attribute

Diese Kapitel beschreibt für jedes Entity die Datentypen der Attribute.

5.1 Datentypen Interbase

Folgende Datentypen können in einer FireBird-Datenbank verwendet.

Datentyp	**Beschreibung**
SMALLINT	Ganzzahlig im Bereich –32768 bis +32767
INTEGER	Ganzzahlig im Bereich -2.147.483.648 .. 2.147.483.647
NUMERIC(P,S)	P-Stellen Gesamt. s-Stellen nach dem Komma. Berechnung ohne Komma und Vorzeichen
FLOAT	Nachkommazahlen im Bereich $1.5*10^{-45}$ .. $3.4*10^{38}$. Die Mantisse hat 7-8 Stellen
DOUBLE PRECISION	Nachkommazahlen im Bereich $5.0*10^{-324}$ .. $1.7*10^{308}$. Die Mantisse hat 15-16 Stellen
VARCHAR(s)	Zeichenwert variabler Länge mit einer maximalen Größe von s
CHAR(s)	Zeichenwert fester Länge mit einer Größe von s
DATE	Datumswert zwischen dem 1. Januar 4712 v. Chr. und dem 31. Dezember 9999 n. Chr.
TIME	Zeitwert zwischen dem 1. Januar 4712 v. Chr. und dem 31. Dezember 9999 n. Chr.
TIMESTAMP	Datum- und Zeitwert zwischen dem 1. Januar 4712 v. Chr. Und dem 31. Dezember 9999 n. Chr.
CURRENCY	Währung. Zweistelliger Numerikwert
BLOB	Binary Large Object

5.1.1 Verwendungszweck

Für Primärschlüssel sind alle numerische Datentypen geeignet. Sinnvoll ist aber nur der Datentyp „Integer“. Der Typ „Smallint“ hat einen zu kleinen Datenbereich. 30000 Werte sind sehr schnell erreicht.

Für Attribute des Geschäftsbereichs ist dagegen der Datentyp Numerik bzw. Number ideal, da dieser automatisch auf- bzw. abgerundet wird.

Bei zeichenorientierten Attribute kann man beide Varianten Char oder VarChar wählen.

Für Währungen wäre der Datentyp Currency ideal. Die Praxis zeigt aber, dass nicht immer sichergestellt ist, dass der Euro oder Dollar definiert ist. Was macht man, wenn man den Dollar <u>und</u> den Euro haben möchte. Dann ist es sinnvoller gleich den Numerictyp zu wählen. Dann kann man auch ein Datenbankprogramm der deutschen Börse anbieten (Dollarkurs ist vierstellig).

5.2 Festlegung der Datentypen der Attribute

KUNDEN

#KNr	INTEGER	NOT NULL
Name	CHAR(40)	NOT NULL
Vorname	CHAR(40)	
PLZ	NUMERIC(5)	
Ort	CHAR(30)	
Straße	CHAR(40)	

BAHNHOF

#BNr	INTEGER	NOT NULL
Name	CHAR(40)	

WAGGON

#WagNr	INTEGER	NOT NULL
Sitzplaetze	NUMERIC(3)	DEFAULT 1

LOK

#LokNr	INTEGER	NOT NULL
Typ	CHR(40)	

DB_FAHRT

#FNR	INTEGER	NOT NULL
LokNr	INTEGER	NOT NULL
WagNr	INTEGER	NOT NULL
StreckenNr	INTEGER	NOT NULL
Datum (Ein Jahr)	DATE	
Abfahrt	TIME	

STRECKE

#StreckenNr	INTEGER	
Name	CHAR(50)	
Bahnhof1	INTEGER	NOT NULL
Bahnhof2	INTEGER	NOT NULL

DB_SEGMENT

#SegNr	INTEGER	NOT NULL	
Preis	NUMERIC(7,2)		
Dauer	NUMERIC(5,2) oder TIME		Format HH,MM
Bahnhof1	INTEGER	NOT NULL	
Bahnhof2	INTEGER	NOT NULL	

STRECKEN_SEGMENTE

#StreckenNr	INTEGER	NOT NULL
#SegNr	INTEGER	NOT NULL
LaufendeNr	INTEGER	NOT NULL

KUNDE_DB_FAHRT (RESERVIERUNGEN)

#L_Nr	INTEGER	NOT NULL	
KNr	INTEGER		
FNr	INTEGER		
Bahnhof1	INTEGER		
Bahnhof2	INTEGER		
Anzahl_Plaetze	NUMERIC(3)	SMALLINT,	DEFAULT = 1

6 Normalisierungen durchführen

In diesem Kapitel werden alle Tabellen in die dritte Normalform überführt.

6.1 Erste Normal-Form

Definition der ersten Normalform:

Eine Relation ist in erster Normalform, wenn die den Tupeltyp bildenden Attribute Datenprimitiva einfachen Datentyps sind.

Da nur Datentypen der Datenbank verwendet wurden, sind alle Tabellen in der ersten Normalform.

6.2 Zweite Normal-Form

Die zweite Normalform betrifft nur Tabellen mit einer Kombination von Attributen für den ID-Schlüssel. Als Kriterium für zweite Normalform gilt, dass alle nicht zum ID-Schlüssel gehörigen Attribute einer Tabelle vom ganzen ID-Schlüssel und nicht nur von einzelnen Attributen davon funktional abhängig sein müssen.

Definition der zweiten Normalform:

Eine Relation liegt in zweiten Normalform vor, wenn sie sich in ersten Normalform befindet und wenn für die Menge A der Attribute der Identifikationsschlüsselattribute gilt:

$R.A \rightarrow R.B$

wobei die Menge B alle Nichtschlüsselattribute umfasst.

Nach unserer Definition wären folgende Tabellen betroffen und müssten überprüft werden:

- „STRECKEN_SEGMENTE“

6.2.1 Strecken_Segmente

#StreckenNr
#SegNr
LaufendeNr

Beispieldaten für die Tabelle_Segmente:

Strecken_Nr	Segment_Nr	Laufende_Nr
10	100	1
10	101	2
10	102	3
10	103	4
20	100	1
20	105	3
20	104	2
20	106	4
30	107	1
30	108	2
30	109	3

Tabelle Strecken_Segmente

Die zweite Normalform ist dann nicht gegeben, wenn für das Nicht-Attribut – hier Laufende_Nr – eine Abhängigkeit bezüglich der Teilattribute des Primärschlüssels existiert. Das heißt, wenn es eine funkttinale Abhängigkeit f(L_Nr) = Segement_Nr bzw. f(L_Nr) = Strecken_Nr gibt. Dabei muss nur eine Teilabhängigkeit existieren.

Prüfen der Abhänggkeit: f(L_Nr) = Strecken_Nr

Eine Teilabhängigkeit besteht nicht, wenn es bei zwei gleichen L_Nr unterschiedliche Strecken_Nr existieren. Dies ist er Fall:

f(L_Nr)	=	Strecken_Nr
f(1)	=	10
f(1)	=	20

Damit ist die Behauptung widerlegt.

Prüfen der Abhänggkeit: f(L_Nr) = Segment_Nr

Eine Teilabhängigkeit besteht nicht, wenn es bei zwei gleichen L_Nr unterschiedliche Segment_Nr existieren. Dies ist er Fall:

f(L_Nr)	=	Segment_Nr
f(1)	=	100
f(1)	=	100
f(1)	=	107

Damit ist die Behauptung widerlegt.

Da die Tabelle nur ein Nichtschlüsselattribut hat, ist die Tabelle in der zweiten Normalform.

6.3 Dritte Normal-Form

Eine Relation ist in dritten Normalform, wenn sie sich in ersten oder zweiten Normalform befindet und kein Nichtschlüsselattribut transitiv abhängig ist vom Identifikationsschlüssel.

Beispiel aus dem Script:

#KNr	#ANr	#VNr	Verkäufer	Datum
1	1	1	Schmid	23.02. 2002
1	2	2	Peter	07.08. 2002
2	3	3	Frey	17.06. 2002
3	4	1	Schmid	15.07. 2002
4	5	3	Frey	13.11. 2002

Tabelle Verkäufe

Für die Tabelle „Verkäufe“ gilt das nicht. Hier kann man aus der Verkäufernummer auf den Verkäufernamen schließen. Die Verkäufernummer ihrerseits ist aber vom ID-Schlüssel „KNR, ANr“ funktional abhängig. Somit besteht eine transitive Abhängigkeit zwischen dem ID-Schlüssel KNR, ANr und dem Attribut „Verkäufer“. Die Tabelle muss weiter aufgespalten werden.

#KNr	#ANr	Datum	#VNr
1	1	23.02. 2002	1
1	2	07.08. 2002	2
2	3	17.06. 2002	3
3	4	15.07. 2002	1
4	5	13.11. 2002	3

Tabelle Verkäufe

VNr	Verkäufer
1	Schmid
2	Peter
3	Frey
4	Schenk

Tabelle Verkäufer

6.3.1 Tabelle KUNDEN

Tabellendefinition:

#KNr
Name
Vorname
PLZ
Ort
Straße

Mit keinem Attribut kann man mit normalen Daten auf andere Attribute schließen. Wären jeder Nach- und Vorname jeweils nur einmal vergeben, dann wäre das ein Verstoß gegen die dritte Normalform.

6.3.2 BAHNHOF

#BNr
Name

Da diese Tabelle nur ein Nichtschlüsselattribut hat, ist es in der dritten Normalform.

6.3.3 WAGGON

#WagNr
Sitzplaetze

Da diese Tabelle nur ein Nichtschlüsselattribut hat, ist es in der dritten Normalform.

6.3.4 LOK

#LokNr
Typ

Da diese Tabelle nur ein Nichtschlüsselattribut hat, ist es in der dritten Normalform.

6.3.5 DB_FAHRT

#FNR
<u>LokNr</u>
<u>WagNr</u>
<u>StreckenNr</u>
Datum
Abfahrt

Diese Tabelle hat zwei Nichtschlüsselattribute (Datum und Abfahrt). Da die Einheit des Attributs Abfahrt in Stunden bzw. Minuten gerechnet wird, besteht keine Abhängigkeit zum Attribut Datum. Es sei denn, dass jeder Zeit nur eine eineindeutige Startzeit hat. Dieses entspricht aber nicht den Vorausetzungen.

6.3.6 STRECKE

#StreckenNr
Name
Bahnhof1
Bahnhof2

Es besteht zwar eine funktionale Abhängigkeit Name zur Streckennummer, aber es existiert keine Beziehung zwischen den Bahnhöfen und dem Streckennamen.

StreckenNr	**Name**	**Bahnhof1**	**Bahnhof2**
10	Hamburg-Berlin	1	5
20	Hamburg-Frankfurt	1	8
30	Frankfurt-München	8	11

Tabelle Strecke

Aus den obige Beispieldaten ergeben sich aus f(Bahnhof1) = StreckenNr
f(1) = 10
f(1) = 20

Aus den obige Beispieldaten ergeben sich aus f(Bahnhof2) = StreckenNr
f(8) = 20
f(8) = 30

Beide Beispiele widersprechen einer Abhängigkeit.

6.3.7 DB_SEGMENT

#SegNr
Preis
Dauer
Bahnhof1
Bahnhof2

Keines der Nichtschlüsselattribute haben eine Teilabhängigkeit zu einem anderen Nichtschlüsselattribut. Es kann jedesmal ein Beispiel à la Kapitel 6.3.6 konstruiert werden. Es kann Segmente mit gleichem Preis geben, ebenso kann es Segmente mit gleicher Dauer geben. Die Bahnhöfe nur dann eine funktionale Beziehung, wenn jede Dauer bzw. jeder Preis sich nicht wiederholt. Dieses entspricht aber nicht dem Pflichtenheft. Also ist die Tabelle in der drittenNormalform.

6.3.8 STRECKEN_SEGMENTE

#StreckenNr
#SegNr
LaufendeNr

Strecken_Nr	Segment_Nr	Laufende_Nr
10	100	1
10	101	2
10	102	3
10	103	4
20	100	1
20	105	3
20	104	2
20	106	4
30	107	1
30	108	2
30	109	3

Tabelle Strecken_Segmente

Aus den obige Beispieldaten ergeben sich aus f(Laufende_Nr) = StreckenNr
f(1) = 10
f(1) = 20
f(1) = 30

Aus den obige Beispieldaten ergeben sich aus f(Laufende_Nr) = StreckenNr
f(1) = 100
f(1) = 100 // bis hierher noch abhängig
f(1) = 107 // jetzt verloren

Beide Beispiele widersprechen einer Abhängigkeit.

6.3.9 KUNDE_DB_FAHRT (RESERVIERUNGEN)

#L_Nr
KNr
FNr
Bahnhof1
Bahnhof2
Anzahl_Plaetze

Da keine funktionale Beziehungen zwischen den Tabellen Kunde, DB_Fahrt und Bahnhof bestehen muss nur das Nichtschlüsselattribut „Anzahl der Plaetze“ untersucht werden. Ein Kunde kann aber für eine beliebige Fahrt zwei oder mehrere Reservierungen vornehmen. Damit kann es dann keine funktionale Teilabhängigkeit geben.

Damit sind alle Tabellen dieses Entwurfs in der dritten Normalform

7 Konsistenzbedingungen formulieren

Bei diesem Schritt geht es darum, Bedingungen zu formulieren, welche von den gespeicherten Daten eingehalten werden müssen. Damit ist sichergestellt, dass die Datenkonsistenz jederzeit erhalten bleibt.

Beispiel:

Tabelle	**Attribut**	**Wertebereich**
Personen	PNr	Ganze Zahl mit 6 Ziffern zwischen 100000 und 9999999
	Name	Zeichenkette mit 30 Zeichen
	Vorname	Zeichenkette mit 20 Zeichen
	Lohnstufe	1 bis 9
Geburtstag	Datum	Datumsfeld im Format TT.MM.JJJJ
Student	Studiengang	Gültig sind nur die Einträge II KI KT WI AT AA

7.1 Allgemeine Konsistenz-Bedingungen

7.1.1 KUNDEN

- Name nur in Buchstaben
- PLZ fünfstellig, numerisch, Speicherung aber Alphanumerisch, da die linksführende Null nicht verschwinden darf.

7.1.2 BAHNHOF

keine

7.1.3 WAGGON

Sitzplaetze größer Null
Sitzplaetze kleiner einem maximalen Wert

7.1.4 LOK

keine

7.1.5 DB_FAHRT

- Datum (Ein Jahr): Bereich zwischen 01.10.YYYY und 30.09.YYYY+1
- Abfahrt 0.00 bis 24.00

7.1.6 DB_SEGMENT

- Preis Preis zwischen 10.00 bis 1000,00 €
- Dauer Dauer zwischen 30 min und 12 Stunden

7.1.7 STRECKE

keine

7.1.8 STRECKEN_SEGMENTE

- Pro Strecke darf nur ein Segment einmal verwendet werden.
- Die Laufender Nummer darf nur einmal verwendet werden.
- Die laufende Numerierung muss ohne Lücken sein (1 bis n).

7.1.9 KUNDE_DB_FAHRT (RESERVIERUNGEN)

- Anzahl_Plaetze größer Null
- Anzahl_Plaetze ≤ Anzahl freier Plätze für alle Segmente der Route !!!

7.2 Konsistenz-Bedingungen in SQL

Constraints erzwingen Regeln auf Tabellenebene. Mit Hilfe von Constraints kann man Folgendes erreichen:
- für Daten geltende Regeln erzwingen, wenn Tabellenzeilen eingefügt, aktualisiert oder gelöscht werden. Das Constraint muss beachtet werden, damit der Vorgang Erfolg hat.
- das Löschen einer Tabelle verhindern, wenn Abhängigkeiten von anderen Tabellen vorhanden sind.
- Regeln für andere Wergzeuge wie Oracle-Developer bereitstellen

Constraint	**Beschreibung**
NOT NULL	Gibt a, dass die Spalte keinen NULL-Wert enthalten darf.
UNIQUE	Gibt eine Spalte oder Spaltenkombination an, deren Werte in allen Zeilen der Tabelle eindeutig sein müssen.
PRIMARY KEY	Identifiziert jede Zeile der Tabelle eindeutig
FOREIGN KEY	Richtet eine Fremdschlüsselbeziehung zwischen der Spalte und einer Spalte der referenzierten Tabelle ein und setzt diese durch.
CHECK	Gibt eine Bedingung an, die erfüllt sein muss.

Einige Bedingungen lassen sich nicht mit einer SQL-Anweisungen lösen. Diese braucht dann nur in „Prosa“ definiert werden“

7.2.1 KUNDEN

Name nur in Buchstaben

Nur mit PL-SQL lösbar.

PLZ fünfstellig, numerisch

Bei dieser Abfrage müssen die Besonderheiten der fünfstelligen Postleitzahlen(PLZ) berücksichtigt werden. Die Postleitzahlen fangen auch mit Null an. Dementsprechend sind folgende PLZ gültig

12345
23456
99999
00999
10000

Damit ergibt sich folgende SQL-Anweisung:

```
CHECK (
        ( PLZ>=999) AND (PLZ<=99999)
)
```

Diese Bedingung funktioniert nur, wenn PLZ ein numerischer Typ ist. Für unseren Fall müssen wir den String PLZ umwandeln in einen numerischen Wert. Gibt es dann eine Exception, so muss dieser Datensatz verworfen werden.

7.2.2 FAHRPLAN

Das Attribut Datum muss innerhalb eines Jahres liegen. Die Interbase liefert mit der EXTRACT-Funktion die gewünschte Abfrage:

Datum:

SQL-Anweisung für das Attribut Datum:

```
CHECK (
        (
                20041001
                >=
                EXTRACT(day FROM Datum)+ EXTRACT(month FROM Datum)*100+
                EXTRACT(year FROM Datum)*10000

        )
        AND
        (
                EXTRACT(day FROM Datum)+ EXTRACT(month FROM Datum)*100+
                EXTRACT(year FROM Datum)*10000
                <
                20051001

        )


)
```

Dieser Check ist gültig für den Bereich 01.10.2004 bis 30.09.2005

Abfahrt:

Überprüfe Zeit im Intervall 0.00 bis 24.00

Wird automatisch von der Datenbank geprüft.

7.2.3 SEGMENTE

- Preis — Preis zwischen 10.00 bis 1000,00 €
- Dauer — Dauer zwischen 30 min und 12 Stunden

```
CHECK (
        (
                ( 10.00 <= PREIS ) AND (PREIS<=1000.00)
        )
        AND
        (
                ( 0.5 <= DAUER ) AND (DAUER<=12.00)
        )

)
```

7.2.4 STRECKEN_SEGMENTE

Pro Strecke darf nur ein Segment verwendet werden.

Nur mit PL-SQL lösbar.

Die Laufender Nummer darf nur einmal verwendet werden.

Nur mit PL-SQL lösbar.

Die laufende Nummerierung muss ohne Lücken sein (1 bis n).

Nur mit PL-SQL lösbar.

7.2.5 WAGGON

Sitzplätze größer Null

SQL-Anweisung für das Attribut Sitzplaetze:

```
CHECK (
        Sitzplaetze >0
)
```

7.2.6 KUNDE_DB_FAHRT (RESERVIERUNGEN)

Anzahl_Plaetze größer Null

SQL-Anweisung für das Attribut Abfahrt:

```
CHECK (
        Anz_Plaetze>0
)
```

Anzahl_Plaetze ≥ Anzahl freier Plätze

SQL-Anweisung:

```
CHECK
  (
    Anz_Plaetze   >=     (    [
                                  SELECT Sitzplaetze
                                   FROM Waggon  w
                                   WHERE w.WagNr = (
                                        SELECT WagNr
                                        FROM DB_FAHRT f
                                        WHERE f.FAHRPLAN_NR = FNR
                                      )
                                    )
                            ]  -
                            (
                                 SELECT sum(ANZ_PLAETZE)
                                 FROM KUNDE_DB_FAHRT r
                                 WHERE r.FAHRPLAN_NR = FNR

                            )

                    )
  )
)
```

Diese Überprüfung ist zu restriktiv, da überprüft wird, ob auf der gesamten Strecke genügend Plätze frei sind. Belegt ein Kunde auf dem ersten Segment einer Strecke alle Plätze und will ein zweiter Kunde ab dem zweiten Segment diese Strecke buchen, so wird dieses verweigert!

8 Transaktionen formulieren

Beim späteren Datenbankbetrieb muss der Datenbestand manipuliert werden. Es ist notwendig, alle zulässigen Manipulationsarten (Transaktionen) und deren Ablauf klar zu definieren, wobei die Datenkonsistenz erhalten bleiben muss.

8.1 CreateTables

In diesem Kapitel werden alle Tabellen mittels einer SQL-Anweisung erzeugt.

```
CREATE TABLE DB_FAHRT (
  FAHRPLAN_NR  INTEGER  NOT NULL,
  DATUM  DATE  NOT NULL,
  ABFAHRT  TIME  NOT NULL,
  WAGNR  INTEGER  NOT NULL,
  LOKNR  INTEGER  NOT NULL,
  STRECKEN_NR  INTEGER  NOT NULL,

  PRIMARY KEY (FAHRPLAN_NR)
);


CREATE TABLE STRECKE (
  STRECKEN_NR  INTEGER  NOT NULL,
  NAME  VARCHAR(40),
  B1  INTEGER  NOT NULL,
  B2  INTEGER  NOT NULL,

  PRIMARY KEY (STRECKEN_NR)
);


CREATE TABLE BAHNHOF (
  BNR  INTEGER  NOT NULL,
  NAME  VARCHAR(40),

  PRIMARY KEY (BNR)
);


CREATE TABLE LOK (
  LOKNR  INTEGER  NOT NULL,
  TYP  VARCHAR(40),

  PRIMARY KEY (LOKNR)
);


CREATE TABLE WAGGON (
  WAGNR  INTEGER  NOT NULL,
  SITZPLAETZE  NUMERIC(7,2)  NOT NULL  DEFAULT 1,
```

```
  PRIMARY KEY (WAGNR)
);
```

```
CREATE TABLE KUNDE (
  KNR  INTEGER  NOT NULL,
  NAME  VARCHAR(40)  NOT NULL,
  VORNAME  VARCHAR(40),
  PLZ  VARCHAR(40),
  ORT  VARCHAR(30),
  STRASSE  VARCHAR(40),

  PRIMARY KEY (KNR)
);
```

```
CREATE TABLE DB_SEGMENT (
  SEG_NR  INTEGER  NOT NULL,
  PREIS  NUMERIC(7,2)  NOT NULL,
  DAUER  NUMERIC(7,2),
  B1  INTEGER  NOT NULL,
  B2  INTEGER  NOT NULL,

  PRIMARY KEY (SEG_NR)
);
```

```
CREATE TABLE KUNDE_DB_FAHRT (
  L_NR  INTEGER  NOT NULL,
  KNR  INTEGER  NOT NULL,
  FAHRPLAN_NR  INTEGER  NOT NULL,
  ANZ_PLAETZE  NUMERIC(3)  NOT NULL  DEFAULT 1,
  B1  INTEGER  NOT NULL,
  B2  INTEGER  NOT NULL,

  PRIMARY KEY (L_NR)
);
```

```
CREATE TABLE STRECKEN_SEGMENTE (
  L_NR  INTEGER  NOT NULL,
  STRECKEN_NR  INTEGER  NOT NULL,
  SEG_NR  INTEGER  NOT NULL
);
```

8.2 Fremdschlüssel

Die folgenden Anweisungen fügen die Fremdschlüssel hinzu. Dieses Reihenfolge ist besser, da sonst eine genau definierte Reihenfolge zur Erzeugung der Tabellen notwendig wäre.

```
ALTER TABLE KUNDE_DB_FAHRT
   ADD CONSTRAINT FK_KUNDE__KND_FAHRT FOREIGN KEY (KNR) REFERENCES KUNDE(KNR);
```

ALTER TABLE **KUNDE_DB_FAHRT**
ADD CONSTRAINT FK_DB_FAHRT__KND_FAHRT FOREIGN KEY (FAHRPLAN_NR) REFERENCES DB_FAHRT(FAHRPLAN_NR);

ALTER TABLE **DB_FAHRT**
ADD CONSTRAINT FK_WAGGON_DB_FAHRT FOREIGN KEY (WAGNR) REFERENCES WAGGON(WAGNR);

ALTER TABLE **DB_FAHRT**
ADD CONSTRAINT FK_LOK_DB_FAHRT FOREIGN KEY (LOKNR) REFERENCES LOK(LOKNR);

ALTER TABLE **DB_FAHRT**
ADD CONSTRAINT FK_STRECKE_DB_FAHRT FOREIGN KEY (STRECKEN_NR) REFERENCES STRECKE(STRECKEN_NR);

ALTER TABLE **STRECKE**
ADD CONSTRAINT FK_BAHNHOF_STRECKE FOREIGN KEY (B1) REFERENCES BAHNHOF(BNR);

ALTER TABLE **STRECKE**
ADD CONSTRAINT FK_BAHNHOF_STRECKE FOREIGN KEY (B2) REFERENCES BAHNHOF(BNR);

ALTER TABLE **DB_SEGMENT**
ADD CONSTRAINT FK_BAHNHOF_DB_SEGMENT FOREIGN KEY (B1) REFERENCES BAHNHOF(BNR);

ALTER TABLE **DB_SEGMENT**
ADD CONSTRAINT FK_BAHNHOF_DB_SEGMENT FOREIGN KEY (B2) REFERENCES BAHNHOF(BNR);

ALTER TABLE **STRECKEN_SEGMENTE**
ADD CONSTRAINT FK_STRECKE_STRECKEN_SEGMENTE FOREIGN KEY (STRECKEN_NR) REFERENCES STRECKE(STRECKEN_NR);

ALTER TABLE **STRECKEN_SEGMENTE**
ADD CONSTRAINT FK_DB_SEGMENT_STRECKEN_SEGMENTE FOREIGN KEY (SEG_NR) REFERENCES DB_SEGMENT(SEG_NR);

ALTER TABLE **KUNDE_DB_FAHRT**
ADD CONSTRAINT FK_BAHNHOF_KUNDE_DB_FAHRT FOREIGN KEY (B1) REFERENCES BAHNHOF(BNR);

ALTER TABLE **KUNDE_DB_FAHRT**
ADD CONSTRAINT FK_BAHNHOF_KUNDE_DB_FAHRT FOREIGN KEY (B2) REFERENCES BAHNHOF(BNR);

8.3 Insert-Data

In diesem Kapitel werden Beispiel-SQL-Anweisungen für das Eintragen der Daten vorgenommen.

8.3.1 KUNDE

INSERT INTO "KUNDE"
VALUES (1, 'Müller', 'Andrea', 39343, 'Wernigerode', 'Breiter Weg 77');

INSERT INTO "KUNDE"
VALUES (2, 'Meyer', 'Hans', 39114, 'Magdeburg', 'Kurze Gasse 4a');

INSERT INTO "KUNDE"
VALUES (3, 'Schulz', 'Ralf', 39113, 'Magdeburg', 'Langer Weg 12');

INSERT INTO "KUNDE"
VALUES (4, 'Young', 'Angus', 12343, 'Weiterstadt', 'Gleichstrom 75');

INSERT INTO "KUNDE"
VALUES (5, 'Mctell', 'Ralf', 23434, 'Cochstedt', 'Franweinstraße 4');

INSERT INTO "KUNDE"
VALUES (6, 'Blackmore', 'Ritchie', 56433, 'Essen', 'Regenbogenweg 33');

8.3.2 BAHNHOF

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES (1, 'Hamburg HBF');

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES ('2', 'Hannover HBF');

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES ('3', 'Braunschweig HBF');

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES ('4', 'Magdeburg HBF');

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES ('5', 'Berlin HBF');

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES ('6', 'Göttingen HBF');

INSERT INTO "BAHNHOF"
("BNR", "NAME")
VALUES ('7', 'Fulda HBF');

```
INSERT INTO "BAHNHOF"
 ("BNR", "NAME")
 VALUES ( '8',  'Frankfurt HBF');

INSERT INTO "BAHNHOF"
 ("BNR", "NAME")
 VALUES ( '9',  ' Würzburg HBF');

INSERT INTO "BAHNHOF"
 ("BNR", "NAME")
 VALUES ( '10',  'Nürnberg HBF');

INSERT INTO "BAHNHOF"
 ("BNR", "NAME")
 VALUES ( '11',  'München HBF');
```

8.3.3 WAGGON

```
INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 1, 5);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 2, 6);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 3, 12);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 4, 8);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 5, 8);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 6, 12);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 7, 15);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 8, 20);
```

```
INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 9, 25);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 10, 10);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 11, 11);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 12, 8);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 13, 15);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 14, 15);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 15, 15);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 16, 15);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 17, 10);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 18, 11);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 19, 12);

INSERT INTO "WAGGON"
 ("WAGNR", "SITZPLAETZE")
 VALUES ( 20, 12);
```

8.3.4 LOK

```
INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2001, 'DAMPF 300');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2002, 'Starlight Express');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2003, 'ICE Magdeburg');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2004, 'ICE Nano-Train');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2005, 'ICE WR');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2006, 'DZug Speedie');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2007, 'Bummelbahn Hanswurst');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2008, 'ICE Essen');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2009, 'ICE Hamburg');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2010, 'ICE Darmstadt');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2011, 'ICE Berlin');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2012, 'ICE Stendal');
```

```
INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2013, 'ICE Hannover');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2014, 'ICE Braunschweig');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2015, 'ICE Nürnberg');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2016, 'ICE Goethe');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2017, 'ICE Schiller');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2018, 'ICE Seneca');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2019, 'ICE Donnervogel');

INSERT INTO "LOK"
 ("LOKNR", "TYP")
 VALUES ( 2020, 'ICE Transrapid');
```

8.3.5 DB_SEGMENT

```
INSERT INTO "DB_SEGMENT"
 ("SEG_NR", "PREIS", "DAUER", "B1", "B2")
 VALUES ( 100, 12.33, 0.5,  1,  2  );

INSERT INTO "DB_SEGMENT"
 ("SEG_NR", "PREIS", "DAUER", "B1", "B2")
 VALUES ( 101, 22.33, 1,  2,  3  );

INSERT INTO "DB_SEGMENT"
 ("SEG_NR", "PREIS", "DAUER", "B1", "B2")
 VALUES ( 102, 12.5,  0.5,  3,  4  );

INSERT INTO "DB_SEGMENT"
 ("SEG_NR", "PREIS", "DAUER", "B1", "B2")
```

VALUES (103, 17.0, 1, 4, 5);

INSERT INTO "DB_SEGMENT"
("SEG_NR", "PREIS", "DAUER", "B1", "B2")
VALUES (104, 22.0, 1.5, 2, 5);

INSERT INTO "DB_SEGMENT"
("SEG_NR", "PREIS", "DAUER", "B1", "B2")
VALUES (105, 17.4, 1.4, 6, 7);

INSERT INTO "DB_SEGMENT"
("SEG_NR", "PREIS", "DAUER", "B1", "B2")
VALUES (106, 25.0, 2, 7, 8);

INSERT INTO "DB_SEGMENT"
("SEG_NR", "PREIS", "DAUER", "B1", "B2")
VALUES (107, 12.0, 0.7, 8, 9);

INSERT INTO "DB_SEGMENT"
("SEG_NR", "PREIS", "DAUER", "B1", "B2")
VALUES (108, 15.0, 0.5, 9, 10);

INSERT INTO "DB_SEGMENT"
("SEG_NR", "PREIS", "DAUER", "B1", "B2")
VALUES (109, 22.0, 1.5, 10, 11);

8.3.6 STRECKE

INSERT INTO "STRECKE"
("STRECKEN_NR", "NAME", "B1", "B2")
VALUES (10, 'Hamburg nach Berlin', 1, 2);

INSERT INTO "STRECKE"
("STRECKEN_NR", "NAME", "B1", "B2")
VALUES (20, 'Hamburg nach Frankfurt', 1, 8);

INSERT INTO "STRECKE"
("STRECKEN_NR", "NAME", "B1", "B2")
VALUES (30, 'Frankfurt nach München', 8, 11);

8.3.7 STRECKEN_SEGMENTE

INSERT INTO "STRECKEN_SEGMENTE"
("STRECKEN_NR", "SEG_NR", "L_NR")
VALUES (10, 100,1);

INSERT INTO "STRECKEN_SEGMENTE"
("StreckenNr", " SegNr ", "LaufendeNr")

VALUES (1, 101,2);

8.3.8 DB_FAHRT

Strecke 10:

```
INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 1,    2001,    1,    10,    '02.01.2005', '08:00');

INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 2,    2002,    2,    10,    '02.01.2005', '09:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 3,    2003,    3,    10,    '02.01.2005', '10:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 4,    2004,    4,    10,    '02.01.2005', '11:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 5,    2005,    5,    10,    '02.01.2005', '12:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 6,    2001,    1,    10,    '03.01.2005', '08:00');

INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 7,    2002,    2,    10,    '03.01.2005', '09:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 8,    2003,    3,    10,    '03.01.2005', '10:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 9,    2004,    4,    10,    '03.01.2005', '11:00');


INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 10,    2005,    5,    10,    '03.01.2005', '12:00');
```

Strecke 20:

```
INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
```

```
 VALUES ( 11,          2006,    6,      20,     '02.01.2005', '08:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 12,          2007,    7,      20,     '02.01.2005', '09:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 13,          2008,    8,      20,     '02.01.2005', '09:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 14,          2009,    9,      20,     '02.01.2005', '10:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 15,          2010,    10,      20,     '02.01.2005', '11:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 16,          2011,    11,      20,     '02.01.2005', '12:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 17,          2006,    6,      20,     '03.01.2005', '08:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 18,          2007,    7,      20,     '03.01.2005', '09:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 19,          2008,    8,      20,     '03.01.2005', '09:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 20,          2009,    9,      20,     '03.01.2005', '10:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 21,          2010,    10,      20,     '03.01.2005', '11:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 22,          2011,    11,      20,     '03.01.2005', '12:00');
```

Strecke 30:

```
INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 23,          2012,    12,      30,     '02.01.2005', '08:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 24,          2013,    13,      30,     '02.01.2005', '10:00');

INSERT INTO "DB_FAHRT"
     ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 25,          2014,   14,      30,     '02.01.2005', '12:00');

INSERT INTO "DB_FAHRT"
```

```
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 26,        2015,   15,     30,     '02.01.2005', '14:00');

INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 27,        2016,   12,     30,     '03.01.2005', '08:00');

INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 28,        2017,   13,     30,     '03.01.2005', '10:00');

INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 29,        2018,   14,     30,     '03.01.2005', '12:00');

INSERT INTO "DB_FAHRT"
    ("FAHRPLAN_NR", "LOKNR","WAGNR","STRECKEN_NR", "DATUM", "ABFAHRT" )
 VALUES ( 30,        2019,   15,     30,     '03.01.2005', '14:00');
```

8.3.9 KUNDE_DB_FAHRT (Reservierungen)

Hamburg nach Berlin, 8:00, 2 Plätze, Kunde 4

```
INSERT INTO "KUNDE_DB_FAHRT"
        ("L_NR", "KNR", "FAHRPLAN_NR","ANZ_PLAETZE", "B1", "B2" )
 VALUES ( 1,        4,        1,        2,              1,     5);
```

Hamburg nach Hannover, 8:00, 1 Plätze, Kunde 1

```
INSERT INTO "KUNDE_DB_FAHRT"
        ("L_NR", "KNR", "FAHRPLAN_NR","ANZ_PLAETZE", "B1", "B2" )
 VALUES ( 2,        1,        1,        1,              1,     2);
```

Hamburg nach Göttingen, 9:00, 2 Plätze, Kunde 2

```
INSERT INTO "KUNDE_DB_FAHRT"
        ("L_NR", "KNR", "FAHRPLAN_NR","ANZ_PLAETZE", "B1", "B2" )
 VALUES ( 3,        2,        7,        2,              1,     6);
```

Göttingen nach Frankfurt-Main, 9:00, 3 Plätze, Kunde 3

```
INSERT INTO "KUNDE_DB_FAHRT"
        ("L_NR", "KNR", "FAHRPLAN_NR","ANZ_PLAETZE", "B1", "B2" )
 VALUES ( 4,        3,        7,        3,              6,     8);
```

Tabelle:

L_NR	KNR	FAHRPLAN_NR	ANZ_PLAETZE	B1	B2
1	4	1	2	1	5
2	1	1	1	1	2
3	2	7	2	1	6
4	3	7	3	6	8

8.4 Delete-Data

Diese Kapitel zeigt alle wichtigen Lösch-Operationen.

```
DELETE
FROM BAHNHOF
WHERE BNr=22;
```

```
DELETE
FROM KUNDE
WHERE KNr=2;
```

8.4.1 Alle Daten bzgl. Segment 12 löschen

- Löschen aller Strecken

```
DELETE
FROM STRECKE
WHERE StreckenNr= (
        Select StreckenNr
        From Strecken_Segmente
        Where SegNr = 12
        )
    ;
```

- Löschen aller Segmente in Strecken_Segmente

```
DELETE
FROM STRECKEN_SEGMENTE
WHERE SegNr = 12;
```

- Löschen aller Segmente in Segmente

```
DELETE
FROM SEGMENTE
WHERE SegNr = 12;
```

8.5 Abfragen

Diese Kapitel zeigt einige wichtigen Abfragen.

8.5.1 Anzeige des Bahnhofs mit der Nummer?

a) Nummer 22
b) Nummer 3

```
SELECT NAME
FROM BAHNHOF
WHERE BNr=22;
```

8.5.2 Welcher Kunde hat die Kundennummer 5?

```
SELECT NAME, VORNAME
FROM KUNDEN
WHERE KNR=22;
```

8.5.3 Welcher Kunden wohnen in Magdeburg?

```
SELECT NAME, VORNAME
FROM KUNDEN
WHERE ORT = 'Magdeburg';
```

8.5.4 Welcher Kunden haben keine Reservierung vorgenommen?

a) Welche Kunden haben eine Reservierung vorgenommen:

```
SELECT NAME, VORNAME, k.KNR
FROM KUNDE k
WHERE   (KNR  IN (
                SELECT DISTINCT KNR
                FROM KUNDE_DB_FAHRT
                )
```

b) Welche Kunden haben <u>keine</u> Reservierung vorgenommen:

```
SELECT NAME, VORNAME, k.KNR
FROM KUNDE k
WHERE   NOT (KNR  IN (
                SELECT DISTINCT KNR
                FROM KUNDE_DB_FAHRT
                )
        )
```

8.5.5 Anzeige aller Strecken mit Bahnhofsnamen

a) Mit Bahnhofsnummer
b) Mit Bahnhofnamen

```
SELECT  s.STRECKEN_NR,
    (SELECT b.NAME FROM BAHNHOF b WHERE b.BNR=s.b1) AS Startbahnhof,
    (SELECT b.NAME FROM BAHNHOF b WHERE b.BNR=s.b2) AS Endbahnhof
FROM STRECKE s
ORDER BY s.STRECKEN_NR;
```

<u>Tabelle:</u>

STRECKEN_NR	Startbahnhof	Endbahnhof
10	Hamburg HBF	Berlin HBF
20	Hamburg HBF	Frankfurt / Main HBF
30	Frankfurt / Main HBF	München HBF

8.5.6 Anzeige aller Segmente mit Bahnhofsnamen

a) Mit Bahnhofsnummer
b) Mit Bahnhofnamen

```
SELECT  s.SEG_NR,
        s.B1,
       (SELECT b.NAME FROM BAHNHOF b WHERE b.BNR=s.b1) AS Startbahnhof,
       s.B2,
      (SELECT b.NAME FROM BAHNHOF b WHERE b.BNR=s.b2) AS Endbahnhof
FROM DB_SEGMENT s
ORDER BY s.SEG_NR;
```

Tabelle:

SEG_NR	B1	STARTBAHNHOF	B2	ENDBAHNHOF
100	1	Hamburg HBF	2	Hannover HBF
101	2	Hannover HBF	3	Braunschweig HBF
102	3	Braunschweig HBF	4	Madgeburg HBF
103	4	Madgeburg HBF	5	Berlin HBF
104	2	Hannover HBF	6	Göttingen HBF
105	6	Göttingen HBF	7	Fulda
106	7	Fulda	8	Frankfurt / Main HBF
107	8	Frankfurt / Main HBF	9	Würzburg HBF
108	9	Würzburg HBF	10	Nürnberg HBF
109	10	Nürnberg HBF	11	München HBF

8.5.7 Welche Segmente sind welchen Strecken zugeordnet?

```
SELECT ss.Strecken_nr, ss.seg_Nr, L_Nr,  Name
FROM STRECKEN_SEGMENTE ss, Strecke st
WHERE (ss.Strecken_Nr = st.Strecken_Nr) ;
```

Tabelle:

STRECKEN_NR	SEG_NR	L_NR	NAME
10	100	1	Hamburg nach Berlin
10	101	2	Hamburg nach Berlin
10	102	3	Hamburg nach Berlin
10	103	4	Hamburg nach Berlin
30	107	1	Frankfurt nach München
30	108	2	Frankfurt nach München
30	109	3	Frankfurt nach München
20	100	1	Hamburg nach Frankfurt
20	105	3	Hamburg nach Frankfurt
20	104	2	Hamburg nach Frankfurt
20	106	4	Hamburg nach Frankfurt

8.5.8 Welche Segmente sind welchen Strecken zugeordnet?

Zusätzlich sollen die Bahnhofsnummern mit angegeben werden

```
SELECT ss.Strecken_nr, ss.seg_Nr, L_Nr,  Name, B1, B2
FROM STRECKEN_SEGMENTE ss, Strecke st, db_segment d
WHERE (ss.Strecken_Nr = st.Strecken_Nr) AND (d.seg_nr = ss.seg_Nr);
```

Tabelle:

STRECKEN_NR	SEG_NR	L_NR	NAME	B1	B2
10	100	1	Hamburg nach Berlin	1	2
10	101	2	Hamburg nach Berlin	2	3
10	102	3	Hamburg nach Berlin	3	4
10	103	4	Hamburg nach Berlin	4	5
30	107	1	Frankfurt nach München	8	9
30	108	2	Frankfurt nach München	9	10
30	109	3	Frankfurt nach München	10	11
20	100	1	Hamburg nach Frankfurt	1	2
20	105	3	Hamburg nach Frankfurt	6	7
20	104	2	Hamburg nach Frankfurt	2	6
20	106	4	Hamburg nach Frankfurt	7	8

8.5.9 Anzeige des Fahrplans

Hier nur die Darstellung der Tabelle

SELECT *
FROM DB_FAHRT
ORDER BY FAHRPLAN_NR

Tabelle:

Siehe Seite 63, Kapitel 12.8.

8.5.10 Anzeige des Fahrplans mit Namen

Ersetzen der Nummern durch Namen (L**OKNR, STRECKEN_NR).**

1. Variante durch JOIN:

SELECT FAHRPLAN_NR, WAGNR, l.TYP "Lokomotive", s.NAME "Strecke", DATUM, ABFAHRT
FROM DB_FAHRT df , LOK l, STRECKE s
WHERE (df.LOKNR=l.LOKNR) AND (df.STRECKEN_NR=s.STRECKEN_NR)
ORDER BY FAHRPLAN_NR

Tabelle:

FAHR-PLAN-NR	WAGNR	Lokomotive	Strecke	DATUM	ABFAHRT
1	1	DAMPF 300	Hamburg nach Berlin	02.01.2005	08:00:00
2	2	Starlight Express	Hamburg nach Berlin	02.01.2005	09:00:00
3	3	Starlight Express	Hamburg nach Berlin	02.01.2005	10:00:00
4	4	ICE Magdeburg	Hamburg nach Berlin	02.01.2005	11:00:00
5	5	ICE Magdeburg	Hamburg nach Berlin	02.01.2005	12:00:00
6	1	DAMPF 300	Hamburg nach Berlin	03.01.2005	08:00:00
7	2	Starlight Express	Hamburg nach Berlin	03.01.2005	09:00:00
8	3	ICE Magdeburg	Hamburg nach Berlin	03.01.2005	10:00:00
9	4	ICE Nano-Train	Hamburg nach Berlin	03.01.2005	11:00:00
10	5	ICE WR	Hamburg nach Berlin	03.01.2005	12:00:00
11	6	DZug Speedie	Hamburg nach Frankfurt	02.01.2005	08:00:00

12	7	Bummelbahn Hanswurst	Hamburg nach Frankfurt	02.01.2005	09:00:00
13	8	ICE Essen	Hamburg nach Frankfurt	02.01.2005	09:00:00
14	9	ICE Hamburg	Hamburg nach Frankfurt	02.01.2005	10:00:00
15	10	ICE Darmstadt	Hamburg nach Frankfurt	02.01.2005	11:00:00
16	11	ICE Berlin	Hamburg nach Frankfurt	02.01.2005	12:00:00
17	6	DZug Speedie	Hamburg nach Frankfurt	03.01.2005	08:00:00
18	7	Bummelbahn Hanswurst	Hamburg nach Frankfurt	03.01.2005	09:00:00
19	8	ICE Essen	Hamburg nach Frankfurt	03.01.2005	09:00:00
20	9	ICE Hamburg	Hamburg nach Frankfurt	03.01.2005	10:00:00
21	10	ICE Darmstadt	Hamburg nach Frankfurt	03.01.2005	11:00:00
22	11	ICE Berlin	Hamburg nach Frankfurt	03.01.2005	12:00:00
23	12	ICE Stendal	Frankfurt nach München	02.01.2005	08:00:00
24	13	ICE Hannover	Frankfurt nach München	02.01.2005	10:00:00
25	14	ICE Braunschweig	Frankfurt nach München	02.01.2005	12:00:00
26	15	ICE Nürnberg	Frankfurt nach München	02.01.2005	14:00:00
27	12	ICE Goethe	Frankfurt nach München	03.01.2005	08:00:00
28	13	ICE Schiller	Frankfurt nach München	03.01.2005	10:00:00
29	14	ICE Seneca	Frankfurt nach München	03.01.2005	12:00:00
30	15	ICE Donnervogel	Frankfurt nach München	03.01.2005	14:00:00

Execution Time (hh:mm:ss.ssss)	00:00:00.0010
Prepare Time (hh:mm:ss.ssss)	00:00:00:0030
Starting Memory	775464
Current Memory	788756
Delta Memory	13292
Number of Buffers	2048
Reads	105
Writes	60
Plan	PLAN SORT (JOIN (S NATURAL,DF INDEX (FK_STRECKE_DB_FAHRT),L INDEX (RDB$PRIMARY4)))
Rows Affected	30

Statistik des SQL-Befehls

2. Variante durch Sub-Select:

```
SELECT   FAHRPLAN_NR,   WAGNR,
        (
            SELECT lk.TYP  FROM LOK lk WHERE (df.LOKNR=lk.LOKNR)
        )  "Lokomotive",
        (
           SELECT s.NAME FROM STRECKE s WHERE df.STRECKEN_NR=s.STRECKEN_NR
        )  "Strecke",
        DATUM, ABFAHRT
FROM DB_FAHRT df
ORDER BY FAHRPLAN_NR
```

Tabelle:

FAHR-PLAN-_NR	WAGNR	Lokomotive	Strecke	DATUM	ABFAHRT
1	1	DAMPF 300	Hamburg nach Berlin	02.01.2005	08:00:00
2	2	Starlight Express	Hamburg nach Berlin	02.01.2005	09:00:00
3	3	Starlight Express	Hamburg nach Berlin	02.01.2005	10:00:00
4	4	ICE Magdeburg	Hamburg nach Berlin	02.01.2005	11:00:00
5	5	ICE Magdeburg	Hamburg nach Berlin	02.01.2005	12:00:00
6	1	DAMPF 300	Hamburg nach Berlin	03.01.2005	08:00:00
7	2	Starlight Express	Hamburg nach Berlin	03.01.2005	09:00:00
8	3	ICE Magdeburg	Hamburg nach Berlin	03.01.2005	10:00:00
9	4	ICE Nano-Train	Hamburg nach Berlin	03.01.2005	11:00:00
10	5	ICE WR	Hamburg nach Berlin	03.01.2005	12:00:00

11	6	DZug Speedie	Hamburg nach Frankfurt	02.01.2005	08:00:00
12	7	Bummelbahn Hanswurst	Hamburg nach Frankfurt	02.01.2005	09:00:00
13	8	ICE Essen	Hamburg nach Frankfurt	02.01.2005	09:00:00
14	9	ICE Hamburg	Hamburg nach Frankfurt	02.01.2005	10:00:00
15	10	ICE Darmstadt	Hamburg nach Frankfurt	02.01.2005	11:00:00
16	11	ICE Berlin	Hamburg nach Frankfurt	02.01.2005	12:00:00
17	6	DZug Speedie	Hamburg nach Frankfurt	03.01.2005	08:00:00
18	7	Bummelbahn Hanswurst	Hamburg nach Frankfurt	03.01.2005	09:00:00
19	8	ICE Essen	Hamburg nach Frankfurt	03.01.2005	09:00:00
20	9	ICE Hamburg	Hamburg nach Frankfurt	03.01.2005	10:00:00
21	10	ICE Darmstadt	Hamburg nach Frankfurt	03.01.2005	11:00:00
22	11	ICE Berlin	Hamburg nach Frankfurt	03.01.2005	12:00:00
23	12	ICE Stendal	Frankfurt nach München	02.01.2005	08:00:00
24	13	ICE Hannover	Frankfurt nach München	02.01.2005	10:00:00
25	14	ICE Braunschweig	Frankfurt nach München	02.01.2005	12:00:00
26	15	ICE Nürnberg	Frankfurt nach München	02.01.2005	14:00:00
27	12	ICE Goethe	Frankfurt nach München	03.01.2005	08:00:00
28	13	ICE Schiller	Frankfurt nach München	03.01.2005	10:00:00
29	14	ICE Seneca	Frankfurt nach München	03.01.2005	12:00:00
30	15	ICE Donnervogel	Frankfurt nach München	03.01.2005	14:00:00

Execution Time (hh:mm:ss.ssss)	00:00:00.0070
Prepare Time (hh:mm:ss.ssss)	00:00:00:0040
Starting Memory	773696
Current Memory	783496
Delta Memory	9800
Number of Buffers	2048
Reads	106
Writes	74
Plan	PLAN (S INDEX (RDB$PRIMARY2))PLAN (LK INDEX (RDB$PRIMARY4))PLAN (DF ORDER RDB$PRIMARY1)
Rows Affected	30

Statistik des SQL-Befehls

Die zweite Variante ist deutlich langsamer als die erste Variante, obwohl oder gerade weil ein Index gesetzt ist.

Die nächste Variante zwingt die Datenbank zur Benutzung von Indizes.

```
SELECT   FAHRPLAN_NR,   WAGNR,
        (
           SELECT lk.TYP  FROM LOK lk WHERE (df.LOKNR=lk.LOKNR)
           PLAN (LK INDEX (RDB$PRIMARY4))
         )  "Lokomotive",
         (
           SELECT s.NAME FROM STRECKE s WHERE df.STRECKEN_NR=s.STRECKEN_NR
           PLAN (S INDEX (RDB$PRIMARY2))
         )  "Strecke",
         DATUM, ABFAHRT
FROM DB_FAHRT df
PLAN (DF ORDER RDB$PRIMARY1)
ORDER BY FAHRPLAN_NR
```

8.5.11 Wie viele Plätze wurden für Strecke 1 (Fahrplan) verkauft?

```
SELECT SUM(ANZ_PLAETZE)
FROM KUNDE_DB_FAHRT
WHERE FAHRPLAN_NR=1;
```

8.5.12 Liste aller Fahrten mit Waggon 2?

```
SELECT SITZPLAETZE
FROM WAGGON w, DB_FAHRT df
WHERE (w.WAGNR=df.WAGNR) and (w.WAGNR=2)
```

8.5.13 Wie viele Plätze hat der Waggon 2 insgesamt reserviert?

```
SELECT SUM(ANZ_PLAETZE)
FROM WAGGON w, KUNDE_DB_FAHRT kdf, DB_FAHRT df
WHERE (w.WAGNR=df.WAGNR) and (w.WAGNR=2) and (df.FAHRPLAN_NR=kdf.FAHRPLAN_NR)
```

Antwort: 5 Plätze

8.5.14 Anzeige der Plätze des Zuges mit der Fahrplannummer FNR

FNR ist die eine feste Fahrplannummer (1, 2, oder 4). Realisiert werden soll die Abfrage mit einem Subselect.

```
SELECT Sitzplaetze
FROM Waggon  w
WHERE w.WagNr = (
        SELECT WagNr
        FROM DB_FAHRT f
        WHERE f.FAHRPLAN_NR = FNR
    )
```

FAHRPLAN_NR	Belegte Anz Plätze
1	3
7	5

8.5.15 Anzeige der belegten Plätze des Zuges mit der Fahrplannummer

```
SELECT FAHRPLAN_NR, sum(ANZ_PLAETZE) "Belegte Anz Plätze"
FROM KUNDE_DB_FAHRT
group by FAHRPLAN_NR
```

FAHRPLAN_NR	Belegte Anz Plätze
1	3
7	5

8.5.16 Anzeige der freien Plätze des Zuges mit der Fahrplannummer

```
SELECT   FAHRPLAN_NR,   WAGNR,
        (
          (
             SELECT Sitzplaetze
             FROM Waggon  w
             WHERE w.WagNr = (
                SELECT WagNr
                FROM DB_FAHRT f
                WHERE f.FAHRPLAN_NR = df.FAHRPLAN_NR
               )
          )
         -
         (
              SELECT COALESCE(sum(ANZ_PLAETZE),0)
              FROM KUNDE_DB_FAHRT r
              WHERE r.FAHRPLAN_NR = df.FAHRPLAN_NR
          )
        )  "Freie Plätze",
        DATUM, ABFAHRT
FROM DB_FAHRT df
ORDER BY FAHRPLAN_NR
```

Tabelle:

FAHRPLAN_NR	WAGNR	Freie Plätze	DATUM	ABFAHRT
1	1	2	02.01.2005	08:00:00
2	2	6	02.01.2005	09:00:00
3	3	12	02.01.2005	10:00:00
4	4	8	02.01.2005	11:00:00
5	5	8	02.01.2005	12:00:00
6	1	5	03.01.2005	08:00:00
7	2	1	03.01.2005	09:00:00
8	3	12	03.01.2005	10:00:00
9	4	8	03.01.2005	11:00:00
10	5	8	03.01.2005	12:00:00
11	6	12	02.01.2005	08:00:00
12	7	15	02.01.2005	09:00:00
13	8	20	02.01.2005	09:00:00
14	9	25	02.01.2005	10:00:00
15	10	10	02.01.2005	11:00:00
16	11	11	02.01.2005	12:00:00
17	6	12	03.01.2005	08:00:00
18	7	15	03.01.2005	09:00:00
19	8	20	03.01.2005	09:00:00
20	9	25	03.01.2005	10:00:00
21	10	10	03.01.2005	11:00:00
22	11	11	03.01.2005	12:00:00
23	12	8	02.01.2005	08:00:00
24	13	15	02.01.2005	10:00:00
25	14	15	02.01.2005	12:00:00
26	15	15	02.01.2005	14:00:00
27	12	8	03.01.2005	08:00:00
28	13	15	03.01.2005	10:00:00
29	14	15	03.01.2005	12:00:00
30	15	15	03.01.2005	14:00:00

8.5.17 Anzeige der freien Plätze des Zuges mit der Fahrplannummer FNR mit Loktyp

```
SELECT   FAHRPLAN_NR,   WAGNR,  (
                SELECT lk.TYP
                FROM LOK lk, DB_FAHRT dbf
                WHERE (lk.LOKNR= dbf.LOKNR) and (df.FAHRPLAN_NR=dbf.FAHRPLAN_NR)
                ) "LOKTYP",
        (
          (
            SELECT Sitzplaetze
            FROM Waggon  w
            WHERE w.WagNr = (
                SELECT WagNr
                FROM DB_FAHRT f
                WHERE f.FAHRPLAN_NR = df.FAHRPLAN_NR
              )
          )
          -
          (
              SELECT COALESCE(sum(ANZ_PLAETZE),0)
              FROM KUNDE_DB_FAHRT r
              WHERE r.FAHRPLAN_NR = df.FAHRPLAN_NR
          )
        )  "Freie Plätze",
        DATUM, ABFAHRT
FROM DB_FAHRT df
ORDER BY FAHRPLAN_NR
```

Tabelle:

FAHR-PLAN_NR	WAGNR	LOKTYP	Freie Plätze	DATUM	ABFAHRT
1	1	DAMPF 300	2	02.01.2005	08:00:00
2	2	Starlight Express	6	02.01.2005	09:00:00
3	3	Starlight Express	12	02.01.2005	10:00:00
4	4	ICE Magdeburg	8	02.01.2005	11:00:00
5	5	ICE Magdeburg	8	02.01.2005	12:00:00
6	1	DAMPF 300	5	03.01.2005	08:00:00
7	2	Starlight Express	1	03.01.2005	09:00:00
8	3	ICE Magdeburg	12	03.01.2005	10:00:00
9	4	ICE Nano-Train	8	03.01.2005	11:00:00
10	5	ICE WR	8	03.01.2005	12:00:00
11	6	DZug Speedie	12	02.01.2005	08:00:00
12	7	Bummelbahn Hanswurst	15	02.01.2005	09:00:00
13	8	ICE Essen	20	02.01.2005	09:00:00
14	9	ICE Hamburg	25	02.01.2005	10:00:00
15	10	ICE Darmstadt	10	02.01.2005	11:00:00
16	11	ICE Berlin	11	02.01.2005	12:00:00
17	6	DZug Speedie	12	03.01.2005	08:00:00
18	7	Bummelbahn Hanswurst	15	03.01.2005	09:00:00
19	8	ICE Essen	20	03.01.2005	09:00:00
20	9	ICE Hamburg	25	03.01.2005	10:00:00
21	10	ICE Darmstadt	10	03.01.2005	11:00:00
22	11	ICE Berlin	11	03.01.2005	12:00:00
23	12	ICE Stendal	8	02.01.2005	08:00:00
24	13	ICE Hannover	15	02.01.2005	10:00:00
25	14	ICE Braunschweig	15	02.01.2005	12:00:00

26	15	ICE Nürnberg	15	02.01.2005	14:00:00
27	12	ICE Goethe	8	03.01.2005	08:00:00
28	13	ICE Schiller	15	03.01.2005	10:00:00
29	14	ICE Seneca	15	03.01.2005	12:00:00
30	15	ICE Donnervogel	15	03.01.2005	14:00:00

9 Zusammenfassung

Der logische Entwurfsprozess beschreibt den schrittweisen Aufbau eines optimalen Datensystems, wobei die Datenkonsistenz eine zentrale Rolle einnimmt. Dabei sind folgende Sachverhalte zu beachten:

- Der logische Entwurfsprozess ist keineswegs eindeutig. Zwei verschiedene Datenbankspezialisten werden meistens auch unterschiedliche Datensysteme entwerfen.

- Der logische Entwurfsprozess ist iterativ. Eine anfänglich grobe Datenstruktur wird schrittweise verfeinert, wobei neue Entitätsmengen und Beziehungen entstehen können.

- Der Normalisierungsprozess alleine ist keine Gewähr für ein praxistaugliches Datensystem. Es ist durchaus möglich, dass zu Gunsten der Systemleistung und / oder der Benutzerfreundlichkeit vom theoretischen Datenmodell abgewichen werden muss.

Die Effizienz der Datenmodellierung hängt auch hier stark von der Erfahrung ab. Diese erlangt man nur durch Übung und Praxis. Dennoch können folgende Regeln beim Entwurfsprozess sehr hilfreich sein:

- Die Anforderungen an das Datenbanksystem sollten möglichst präzise in Form eines Pflichtenheftes formuliert werden. Unklare Vorgaben führen zwangsläufig zu mangelhaften Applikationen.

- Es sollten in der ersten Phase möglichst viele Entitätsmengen gebildet werden. Damit werden rekursive Beziehungen, welche das Problem unnötig komplizieren, weitgehend vermieden.

- Nicht-hierarchische Beziehungen sollten nicht im gleichen Schritt, sondern nacheinander transformiert werden. Falls neue Entitätsmengen entstehen, sind die entsprechenden Beziehungen umgehend zu formulieren. Dadurch können versteckte Redundanzen vermieden werden.

- Man sollte die Datenbasis zuerst global normalisieren und erst dann zur optimalen Normalform übergehen. Damit werden Redundanzen als solche erkannt und können programmtechnisch so verwaltet werden, dass die Datenkonsistenz jederzeit garantiert werden kann.

10 Literatur

Codd, E. F.: A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Vol 13, No. 6, p. 377-387, 1970

Chen, P. P.: The Entity-Relationship Model-Toward a Unified View of Data, ACM Transactions on Database Systems, Vol. 1, p. 9-36, 1976

Date, C. J.: An Introduction to Database System, Vol. 1, Reading Massachusetts, 1985

Jackson, G. A.: Entwurf relationaler Datenbanken, Hanser Verlag München, Wien, 1989

Vossen, G., Witt, K.-U.: Entwicklungstendenzen bei Datenbanksystemen, Oldenbourg Verlag München, Wien, 1991

Wedekind, H.: Datenbanksysteme I, Bibliographisches Institut Mannheim, 1981

Schicker, E.: Datenbanken und SQL, B. G. Teubner Stuttgart, 1996

Sauer, H.: Relationale Datenbanken, Addison-Wesley, 1995

Meier, A., Wüst, Th.: Objektorientierte Datenbanken, dpunkt Verlag, 1997

Abbey, Corey, Abramson: Oracle 8i für Einsteiger, Hanser Verlag 1999

Ponndorf, St., Matthäus, W.-G.: Oracle 8i und Java, Addison-Wesley, 1999

Kähler, W.-M.: Relationales und objektrelationales SQL, Vieweg & Sohn Verlagsgesellschaft mbH, 1999

McCullough-Dieter, C.: Oracle8i für Dummies, MITP-Verlag GmbH Bonn, 1999

Kofler, M.: MySQL, Addison-Wesley, 2001

Riccardi, G.: Datenbanksysteme mit Internet und Java-Applikationen, Addison-Wesley, 2001

Hohenstein, U., Pleßer, V.: Oracle 9i, Effiziente Anwendungsentwicklung mit objektrelationalen Konzepten, dpunkt.verlag, 2002

Kuhlmann, G., Müllmerstadt, F.: SQL, Der Schlüssel zu relationalen Datenbanken, Rowohlt Taschenbuch Verlag, 2001

11 Anhang

Kompletter SQL-Code, generiert vom Designer.

11.1 Header

```
/*==============================================================*/
/* Project Filename: D:\Daten\DP60\DBS-Dev\dbs\Eisenbahn.sql      */
/* Project Name:     Projekt HS-Harz                              */
/* Author:           HS-Harz                                      */
/* DBMS:             Firebird                                     */
/* Copyright:        Copyright by ???                             */
/* Generated on:     07.01.2005 01:51:14                          */
/*==============================================================*/
```

11.2 Tabellen

```
CREATE TABLE DB_FAHRT (
  FAHRPLAN_NR  INTEGER
            NOT NULL,
  DATUM  DATE
            NOT NULL,
  ABFAHRT  TIME
            NOT NULL,
  WAGNR  INTEGER
            NOT NULL,
  LOKNR  INTEGER
            NOT NULL,
  STRECKEN_NR  INTEGER
            NOT NULL,

  PRIMARY KEY (FAHRPLAN_NR)
);



CREATE TABLE STRECKE (
  STRECKEN_NR  INTEGER
            NOT NULL,
  NAME  VARCHAR(40),
  B1  INTEGER
            NOT NULL,
  B2  INTEGER
            NOT NULL,

  PRIMARY KEY (STRECKEN_NR)
);



CREATE TABLE BAHNHOF (
  BNR  INTEGER
            NOT NULL,
  NAME  VARCHAR(40),

  PRIMARY KEY (BNR)
```

```
);


CREATE TABLE LOK (
  LOKNR  INTEGER
            NOT NULL,
  TYP  VARCHAR(40),

  PRIMARY KEY (LOKNR)
);


CREATE TABLE WAGGON (
  WAGNR  INTEGER
            NOT NULL,
  SITZPLAETZE  NUMERIC(7,2)
            DEFAULT 1
            NOT NULL,

  PRIMARY KEY (WAGNR)
);


CREATE TABLE KUNDE (
  KNR  INTEGER
            NOT NULL,
  NAME  VARCHAR(40)
            NOT NULL,
  VORNAME  VARCHAR(40),
  PLZ  VARCHAR(40),
  ORT  VARCHAR(30),
  STRASSE  VARCHAR(40),

  PRIMARY KEY (KNR)
);


CREATE TABLE DB_SEGMENT (
  SEG_NR  INTEGER
            NOT NULL,
  PREIS  NUMERIC(7,2)
            NOT NULL,
  DAUER  NUMERIC(5,2),
  B1  INTEGER
            NOT NULL,
  B2  INTEGER
            NOT NULL,

  PRIMARY KEY (SEG_NR)
);


CREATE TABLE KUNDE_DB_FAHRT (
  L_NR  INTEGER
            NOT NULL,
  KNR  INTEGER
            NOT NULL,
```

```
  FAHRPLAN_NR  INTEGER
          NOT NULL,
  ANZ_PLAETZE  NUMERIC(3)
          DEFAULT 1
          NOT NULL,
  B1  INTEGER
          NOT NULL,
  B2  INTEGER
          NOT NULL,

  PRIMARY KEY (L_NR)
);



CREATE TABLE STRECKEN_SEGMENTE (
  L_NR  INTEGER
          NOT NULL,
  STRECKEN_NR  INTEGER
          NOT NULL,
  SEG_NR  INTEGER
          NOT NULL
);
```

11.3 Fremdschlüssel

```
ALTER TABLE KUNDE_DB_FAHRT
   ADD CONSTRAINT FK_KUNDE__KND_FAHRT FOREIGN KEY (KNR) REFERENCES KUNDE(KNR);


ALTER TABLE KUNDE_DB_FAHRT
   ADD CONSTRAINT FK_DB_FAHRT__KND_FAHRT FOREIGN KEY (FAHRPLAN_NR) REFERENCES
DB_FAHRT(FAHRPLAN_NR);


ALTER TABLE DB_FAHRT
   ADD CONSTRAINT FK_WAGGON_DB_FAHRT FOREIGN KEY (WAGNR) REFERENCES
WAGGON(WAGNR);


ALTER TABLE DB_FAHRT
   ADD CONSTRAINT FK_LOK_DB_FAHRT FOREIGN KEY (LOKNR) REFERENCES LOK(LOKNR);


ALTER TABLE DB_FAHRT
   ADD CONSTRAINT FK_STRECKE_DB_FAHRT FOREIGN KEY (STRECKEN_NR) REFERENCES
STRECKE(STRECKEN_NR);


ALTER TABLE STRECKEN_SEGMENTE
   ADD CONSTRAINT FK_STRECKE_STRECKEN_SEGMENTE FOREIGN KEY (STRECKEN_NR)
REFERENCES STRECKE(STRECKEN_NR);


ALTER TABLE STRECKEN_SEGMENTE
   ADD CONSTRAINT FK_DB_SEGMENT_STRECKEN_SEGMENTE FOREIGN KEY (SEG_NR)
REFERENCES DB_SEGMENT(SEG_NR);
```

```
ALTER TABLE STRECKE
  ADD CONSTRAINT FK_BAHNHOF_STRECKE1 FOREIGN KEY (B1) REFERENCES
BAHNHOF(BNR);

ALTER TABLE STRECKE
  ADD CONSTRAINT FK_BAHNHOF_STRECKE2 FOREIGN KEY (B2) REFERENCES
BAHNHOF(BNR);

ALTER TABLE DB_SEGMENT
  ADD CONSTRAINT FK_BAHNHOF_DB_SEGMENT1 FOREIGN KEY (B1) REFERENCES
BAHNHOF(BNR);

ALTER TABLE DB_SEGMENT
  ADD CONSTRAINT FK_BAHNHOF_DB_SEGMENT2 FOREIGN KEY (B2) REFERENCES
BAHNHOF(BNR);

ALTER TABLE KUNDE_DB_FAHRT
  ADD CONSTRAINT FK_BAHNHOF_KUNDE_DB_FAHRT1 FOREIGN KEY (B1) REFERENCES
BAHNHOF(BNR);

ALTER TABLE KUNDE_DB_FAHRT
  ADD CONSTRAINT FK_BAHNHOF_KUNDE_DB_FAHRT FOREIGN KEY (B2) REFERENCES
BAHNHOF(BNR);
```

11.4 Prüfbedingungen

```
ALTER TABLE WAGGON
    ADD CONSTRAINT CH_SITZPLAETZE CHECK (
  SITZPLAETZE >0
);

ALTER TABLE DB_SEGMENT
    ADD CONSTRAINT CH_PREIS CHECK (
  ( 10.00 <= PREIS ) AND (PREIS<=1000.00)
);

ALTER TABLE DB_SEGMENT
    ADD CONSTRAINT CH_DAUER CHECK (
  ( 0.5 <= DAUER ) AND (DAUER<=12.00)
);
```

12 Datenbank

Diese Kapitel übersichtlich zeigt alle Tabellen.

12.1 Waggon

WAGNR	SITZPLAETZE
1	5
2	6
3	12
4	8
5	8
6	12
7	15
8	20
9	25
10	10
11	11
12	8
13	15
14	15
15	15
16	15
17	10
18	11
19	12
20	12

12.2 Lok

LOKNR	TYP
2001	DAMPF 300
2002	Starlight Express
2003	ICE Magdeburg
2004	ICE Nano-Train
2005	ICE WR
2006	DZug Speedie
2007	Bummelbahn Hanswurst
2008	ICE Essen
2009	ICE Hamburg
2010	ICE Darmstadt
2011	ICE Berlin
2012	ICE Stendal
2013	ICE Hannover
2014	ICE Braunschweig
2015	ICE Nürnberg
2016	ICE Goethe
2017	ICE Schiller
2018	ICE Seneca
2019	ICE Donnervogel
2020	ICE Transrapid

12.3 Bahnhof

BNR	NAME
1	Hamburg HBF
2	Hannover HBF
3	Braunschweig HBF
4	Madgeburg HBF
5	Berlin HBF
6	Göttingen HBF
7	Fulda
8	Frankfurt / Main HBF
9	Würzburg HBF
10	Nürnberg HBF
11	München HBF

12.4 Kunde

KNR	NAME	VORNAME	PLZ	ORT	STRASSE
1	Müller	Andrea	39343	Wernigerode	Breiter Weg 77
2	Meyer	Hans	39114	Magdeburg	Kurze Gasse 4a
3	Schulz	Ralf	39113	Magdeburg	Langer Weg 12
4	Young	Angus	12343	Weiterstadt	Gleichstrom 75
5	Mctell	Ralf	23434	Cochstedt	Franweinstraße 4
6	Blackmore	Ritchie	56443	Essen	Regenbogenweg 33

12.5 DB_SEGMENT

SEG_NR	PREIS	DAUER	B1	B2
100	12,33	0,5	1	2
101	23,33	1	2	3
102	12,5	0,5	3	4
103	17	1	4	5
104	22	1,5	2	6
105	17,4	1,4	6	7
106	25	2	7	8
107	12	0,7	8	9
108	15	0,5	9	10
109	22	1,5	10	11

12.6 Strecke

STRECKEN_NR	NAME	B1	B2
10	Hamburg nach Berlin	1	5
30	Frankfurt nach München	8	11
20	Hamburg nach Frankfurt	1	8

12.7 Strecken_Segmente

STRECKEN_NR	SEG_NR	L_NR
10	100	1
10	101	2
10	102	3
10	103	4
20	100	1
20	105	3
20	104	2
20	106	4
30	107	1
30	108	2
30	109	3

12.8 DB_FAHRT

FAHRPLAN_NR	WAGNR	LOKNR	STRECKEN_NR	DATUM	ABFAHRT
1	1	2001	10	02.01.2005	08:00:00
2	2	2002	10	02.01.2005	09:00:00
3	3	2002	10	02.01.2005	10:00:00
4	4	2003	10	02.01.2005	11:00:00
5	5	2003	10	02.01.2005	12:00:00
6	1	2001	10	03.01.2005	08:00:00
7	2	2002	10	03.01.2005	09:00:00
8	3	2003	10	03.01.2005	10:00:00
9	4	2004	10	03.01.2005	11:00:00
10	5	2005	10	03.01.2005	12:00:00
11	6	2006	20	02.01.2005	08:00:00
12	7	2007	20	02.01.2005	09:00:00
13	8	2008	20	02.01.2005	09:00:00
14	9	2009	20	02.01.2005	10:00:00
15	10	2010	20	02.01.2005	11:00:00
16	11	2011	20	02.01.2005	12:00:00
17	6	2006	20	03.01.2005	08:00:00
18	7	2007	20	03.01.2005	09:00:00
19	8	2008	20	03.01.2005	09:00:00
20	9	2009	20	03.01.2005	10:00:00
21	10	2010	20	03.01.2005	11:00:00
22	11	2011	20	03.01.2005	12:00:00
23	12	2012	30	02.01.2005	08:00:00
24	13	2013	30	02.01.2005	10:00:00
25	14	2014	30	02.01.2005	12:00:00
26	15	2015	30	02.01.2005	14:00:00
27	12	2016	30	03.01.2005	08:00:00
28	13	2017	30	03.01.2005	10:00:00
29	14	2018	30	03.01.2005	12:00:00
30	15	2019	30	03.01.2005	14:00:00

12.9 KUNDE_DB_FAHRT (Reservierungen)

L_NR	KNR	FAHRPLAN_NR	ANZ_PLAETZE	B1	B2
1	4	1	2	1	5
2	1	1	1	1	2
3	2	7	2	1	6
4	3	7	3	6	8

13 Stichwortverzeichnis