

Institut für
Programmstrukturen und
Datenorganisation (IPD)

Prof. Dr.-Ing. Dr. h.c. Peter C. Lockemann
Universität Karlsruhe (TH)
Fakultät für Informatik



Konzeption und Evaluation eines Data Warehouse zur Analyse von Daten im Verkehrsbereich

Studienarbeit

von

Jan Sandberger

im April 2004

Verantwortlicher Betreuer: Prof. Dr.-Ing. Dr. h.c. Peter C. Lockemann

Betreuender Mitarbeiter: Dipl.-Inform. Heiko Schepperle

Diese Arbeit wurde von mir selbstständig angefertigt. Alle verwendeten Literaturstellen sind im Literaturverzeichnis aufgeführt; eine Verwendung anderer Hilfsmittel erfolgte nicht.

Ich versichere dies ausdrücklich mit nachstehender Unterschrift:

Jan Sandberger

Karlsruhe, den 10.4.2004

Kurzfassung

Diese Studienarbeit beschreibt einen Weg, Simulationsdaten aus dem Simulationsprogramm VISSIM der Firma PTV zur besseren Auswertung in einem Data Warehouse zu speichern.

In der ersten Phase wird ein Entwurf eines normalisiertes Datenbankschema dargestellt, das alle Zusammenhänge der Simulationsdaten abbildet. Im zweiten Schritt wird aufgezeigt, wie mit Hilfe von Java und JDBC ein Werkzeug verwirklicht wurde, welches die Simulationsdaten aus einer Simulationsdatei ausliest und im Nutzdatschema speichert. Wie mit Hilfe materialisierter Sichten in einem nächsten Schritt ein Data Warehouse Schema erzeugt, und im letzten Schritt einige beispielhafte OLAP Anfragen an das Data Warehouse selbst gestellt werden, wird ebenfalls dargestellt.

Inhaltsverzeichnis

1. Einleitung.....	9
1.1 Motivation und Projektumfeld.....	9
1.2 Aufgabenstellung und Zielsetzung	9
1.3 VISSIM.....	10
1.4 Übersicht über die folgenden Kapitel	10
2. Das Nutzdatenschema.....	11
2.1 Beschreibung der Logdaten	11
2.2 Besonderheiten einer Simulation.....	13
2.3 Besonderheiten der Simulation im relationalen Schema	13
2.4 Umsetzung der Daten in ein relationales Schema	15
2.5 Beschleunigung des Datenbankanfragen.....	16
3. Das Füllen der Nutzdatenbank.....	19
3.1 Das Projekt vissimLoader.....	19
3.2 Das Kopieren der Daten	21
3.3 Umsetzung der Besonderheiten einer Simulation.....	22
3.4 Datentypen des vissimLoader.....	23
3.5 Einschränkungen, Annahmen, sonstige Voraussetzungen	24
4. Das Data Warehouse Schema.....	25
4.1 Database Links	25
4.2 Materialisierte Sichten.....	26
4.3 Das Warehouse-Schema als materialisierte Sichten.....	27
4.4 Das Data Warehouse im Warehouse-Builder	33
5. Anfragen an das Data Warehouse.....	35
5.1 einfache Anfragen.....	35
5.2 OLAP Anfragen.....	36
6. Zusammenfassung und Ausblick.....	41
6.1 Zusammenfassung	41
6.2 Ausblick.....	42
7. Abbildungsverzeichnis	43
8. Literaturverzeichnis	45

1. Einleitung

In diesem Kapitel wird kurz auf die Motivation, auf das Projektumfeld und auf sonstige Aspekte der Arbeit eingegangen. Obwohl die Beschreibung der eigentlichen Arbeit erst in den nächsten Kapiteln erfolgt, werden in diesem Kapitel schon einige Begriffe erläutert und definiert.

1.1 Motivation und Projektumfeld

Die hohe Belastung der Straßen, im Besonderen die Belastung der Autobahnen, stellen ein immer größer werdendes Problem dar. Schnell wird aus der hohen Belastung eine Überlastung, die sich als Stau bemerkbar macht und die weitreichende ökologische und ökonomische Konsequenzen hat.

Intelligente und vernetzt arbeitende Routenplanungssysteme können ihren Beitrag zur Lösung dieses Problems leisten.

Das Projekt OVID hat zum Ziel, eine Plattform für die Modellierung und Bewertung verkehrsinfrastruktureller, verkehrstelematischer und logistischer Maßnahmen aufzubauen. Intelligente Routenplanungssysteme sind ein wichtiger Bestandteil solcher Maßnahmen (vgl. [OVID]).

Das Projekt OVID wird vom Bundesministerium für Bildung und Forschung gefördert. An diesem Projekt sind neben dem Institut für Programmstrukturen und Datenorganisation (IPD) noch das Institut für Verkehrswesen (IFV), das Institut für Fördertechnik und Logistiksysteme (IFL) und das Institut für Wirtschaftspolitik und Wirtschaftsforschung (IWW) der Universität Karlsruhe sowie das Institut für Informations- und Datenverarbeitung der Fraunhofergesellschaft (IITB) beteiligt. Als Industriepartner sind die PTV AG und die LOCOM Consulting GmbH, beide mit Sitz in Karlsruhe, beteiligt.

Diese Studienarbeit ist im Rahmen des Teilprojekts B1 „Verlässliche Datenbanken für die Informationsbereitstellung im Verkehr“ entstanden und beschäftigt sich mit der Auswertung von Daten aus dem Verkehrsbereich.

1.2 Aufgabenstellung und Zielsetzung

Ziel dieser Studienarbeit ist, ein besonderes System zur Speicherung großer Datenmengen, ein sog. Data Warehouse System, zu erstellen und mit simulierten Verkehrsdaten zu befüllen.

Die simulierten Verkehrsdaten werden mit dem Programm „VISSIM“, einem Produkt der PTV AG (siehe auch [PTV]), erzeugt und in einer Datei gespeichert. Diese Datei wird im Folgenden als Logdatei bezeichnet, die Daten in dieser Datei als Logdaten. Bevor die Daten in das Data Warehouse geschrieben werden, müssen sie in eine relationale Datenbank eingefügt und vorverarbeitet werden. Im Folgenden wird diese relationale Datenbank als Nutzdatenbank und die Daten in dieser Datenbank als Nutzdaten bezeichnet. Aus der Nutzdatenbank wird schließlich das Data Warehouse erstellt.

Das Ziel dieser Studienarbeit besteht darin, beispielhaft Daten aus dem Verkehrsumfeld auszuwerten, um diese Erfahrungen später in die Auswertung von realen Messdaten aus verschiedenen Quellen einfließen zu lassen.

1.3 VISSIM

Das Programm VISSIM 3.70 ist ein Produkt der Firma PTV AG. Mit diesem lassen sich Verkehrsflüsse auf Einzelfahrzeugebene simulieren. In einem Editor können einzelne Fahrspuren, Ampeln, Messstationen mit bestimmten Eigenschaften zu einem komplexen Netz zusammengefügt werden. Auf diese Weise entstehen Kreuzungen oder ganze Stadtteile.

Zusätzlich lassen sich verschiedene Fahrzeugtypen mit bestimmten Eigenschaften wie Länge, Wunschbeschleunigung, befahrbare Spuren uvm. definieren.

Eine genauere Betrachtung dieses Simulationswerkzeug und seiner Funktionsweise war nicht Teil dieser Studienarbeit. Es habe nur mit vorgefertigten Simulationsdateien gearbeitet, die von der PTV zu Demonstrationszwecken zur Verfügung gestellt wurden.

Das für diese Studienarbeit wichtigste Element sind Messstationen, die an bestimmten Stellen einer Fahrspur angelegt werden können. Diese Messstationen können von simulierten Fahrzeugen bestimmte Daten wie u.a. Geschwindigkeit und Beschleunigung aufnehmen. Die Messungen bilden die Logdaten und werden in der Logdatei gespeichert

VISSIM simuliert also auf Mikroebene, d.h. es werden die kleinsten Einheiten eines Verkehrsnetzes simuliert. Das IPD beschäftigt sich auch mit dem Produkt VISUM, das eine Simulation auf höherer Ebene ermöglicht. In der Makrosicht von VISUM werden also keine einzelnen Fahrzeuge mehr simuliert. Das Produkt VISUM bzw. die Ergebnisse aus der Arbeit damit sind nicht in diese Studienarbeit eingeflossen.

1.4 Übersicht über die folgenden Kapitel

In Kapitel 2 wird ein Nutzdatschema beschrieben, in dem die Simulationsdaten gespeichert werden. Dazu wird erst eine genauere Beschreibung der Logdaten erfolgen. Anschließend wird erläutert, wie diese Daten in ein relationales Schema umgesetzt werden und welche Besonderheiten sich durch die Simulation ergeben. Auch konzeptuelle Einschränkungen werden hier erläutert.

Das 3. Kapitel beschäftigt sich mit der Befüllung des Nutzdatschemas mit Hilfe von JDBC und mit konzeptuellen Einschränkungen, die in dieser Phase aufgrund von Besonderheiten simulierter Daten gemacht werden.

Im 4. Kapitel wird beschrieben, wie das Data Warehouse Schema angelegt wurde. In einer ersten Variante geschieht dies mit Hilfe von materialisierten Sichten. Besonderes Augenmerk wird dabei auf die Dimensionstabellen und ihre Erzeugung gelegt. Außerdem wird ein Blick auf die Dimensionen, ihre Ebenen und Hierarchien geworfen.

Anfragen an das Data Warehouse werden in Kapitel 5 behandelt.

2. Das Nutzdatenschema

In diesem Kapitel werden die Logdaten, die das Simulationsprogramm VISSIM generiert, genauer beschrieben und die Konzeption und die Umsetzung eines relationalen Schemas erläutert, welches diese Daten speichern kann. Besonderes Augenmerk wird dabei auf jene Besonderheiten gelegt, die sich dadurch ergeben, dass simulierte Daten verwendet werden.

2.1 Beschreibung der Logdaten

Die Logdatei ist in zwei Teile aufgeteilt; sie enthält zum Einen Informationen über die Messstationen zum Anderen Informationen über die Messungen, die an diesen Messstationen durchgeführt wird.

2.1.1 Beschreibung der Messstationen

Hier ein beispielhafter Ausschnitt für den ersten Teil, also Informationen über die Messstationen:

Messung	111:	Strecke	1	Spur	1	bei	137.509 m,	Länge	0.000 m.
Messung	112:	Strecke	1	Spur	2	bei	136.416 m,	Länge	0.000 m.
Messung	1311:	Strecke	13	Spur	1	bei	109.251 m,	Länge	0.000 m.
Messung	1312:	Strecke	13	Spur	2	bei	109.412 m,	Länge	0.000 m.
Messung	1611:	Strecke	16	Spur	1	bei	17.832 m,	Länge	0.000 m.
Messung	1612:	Strecke	16	Spur	2	bei	17.958 m,	Länge	0.000 m.
Messung	2811:	Strecke	28	Spur	1	bei	25.557 m,	Länge	0.000 m.

Abb. 1. Beschreibung der Messstationen

Jede dieser Zeilen beschreibt eine Messstation. Die Messstation mit der Nummer 111 steht demnach auf Spur 1 der Strecke mit der Nummer 1 an Position 137,509 Meter nach Spurfang. Jede Messstation hat laut Logdatei eine Länge von 0,000 Metern, die Länge kann daher (zumindest in den Logdaten dieser VISSIM-Version) vernachlässigt werden. Der erste Teil der Logdatei enthält also für jede Messstation genau eine Zeile und ist damit unabhängig von der Dauer einer Simulation und im Normalfall auch wesentlich kürzer als der zweite Teil, der die eigentlichen Messungen beschreibt.

2.1.2 Beschreibung der Messungen

Die Anzahl der Messungen hängt zusätzlich von der Dauer der Simulation ab. Auch hier wieder ein beispielhafter Ausschnitt.

Messung	t (Einf)	t (Ausf)	Fz-Nr	Fztyp	Linie	v [m/s]	b [m/s ²]	Bel	Pers	tStau	Temp	FzLaenge [m]
2811	600.62	-1.00	960	100	0	10.8	-1.69	0.08	2	0.0	0.0	4.40
2111	-1.00	600.62	954	100	0	11.9	-1.50	0.02	2	0.0	0.0	4.55
3211	600.99	-1.00	912	100	0	12.9	0.58	0.01	2	0.0	0.0	4.11
2811	-1.00	601.04	960	100	0	10.0	-1.68	0.04	2	0.0	0.0	4.40
3411	601.07	-1.00	952	100	0	17.1	-0.10	0.03	1	0.0	0.0	4.11
6912	601.24	-1.00	817	150	0	18.0	0.14	0.06	1	56.4	0.0	6.31
3211	-1.00	601.31	912	100	0	13.0	0.18	0.01	2	0.0	0.0	4.11
3411	-1.00	601.31	952	100	0	17.1	-0.11	0.01	1	0.0	0.0	4.11
6912	-1.00	601.59	817	150	0	18.0	0.14	0.09	1	56.4	0.0	6.31

Abb. 2. Beschreibung der Messdaten

Diese Daten bedürfen einer etwas ausführlicheren Erklärung.

Zum einfacheren Verständnis lassen sich Messstationen als Induktionsschleife oder Lichtschranke orthogonal zur Fahrtrichtung annehmen. Wenn ein Fahrzeug eine Messstation passiert, gibt es dazu zwei Einträge in der Logdatei.

Der erste Eintrag wird zu dem Zeitpunkt eingefügt, zu dem das Fahrzeug die Messstation erreicht – im Falle eines PKW also, wenn die vordere Stoßstange die Messstation überquert. In diesem Falle erhält $t(Einf)$ einen Zeitwert, $t(Ausf)$ wird auf -1.00 gesetzt.

Der zweite Eintrag wird zu dem Zeitpunkt eingefügt, zu dem das Fahrzeug die Messstation wieder verlässt, also mit der hinteren Stoßstange über die Messstation fährt. In diesem Falle erhält $t(Ausf)$ einen Zeitwert, $t(Einf)$ wird auf -1.00 gesetzt.

Im Normalfall gibt es also stets zwei Zeilen in einer Logdatei, die in der Messstation, der Fahrzeugnummer und anderen Spalten übereinstimmen. Solche zwei Zeilen unterscheiden sich nur in der Geschwindigkeit und der Beschleunigung. Die Zeilen eins und vier aus obigem Beispiel entsprechen einem solchen Paar von Zeilen. In diesen beiden Zeilen wird das Fahrzeug mit der Nummer 960 an der Messstation mit der Nummer 2811 in der Zeit von 600,52 Sekunden bis 601,04 Sekunden nach Simulationsbeginn beschrieben. Da ein anderes Fahrzeug zum Zeitpunkt 600,99 Sekunden eine andere Messstation erreicht, stehen diese beiden Zeilen nicht hintereinander.

Die Spalte $v[m/s]$ gibt die Geschwindigkeit an, die Spalte $b[m/s^2]$ die Beschleunigung. $Pers$ gibt die Anzahl der Personen im Fahrzeug an, die Spalte $tStau$ gibt die Anzahl Sekunden an, die ein Fahrzeug während einer Simulation bereits gestanden hat. Die beiden Spalten $Temp$ sind für späteren Gebrauch vorgesehen; in der Version 3.70 von VISSIM enthalten sie immer den Wert 0.0.

Die Messungen werden erst nach einer sog. Einschwingphase in die Logdatei geschrieben, da Messdaten zu Anfang einer Simulation nicht unbedingt die Wirklichkeit widerspiegeln. Fahrzeuge müssen beispielsweise erst einmal die Messstationen erreichen können, bevor es sinnvoll ist, Messungen an diesen Messstationen vorzunehmen. Daher kann es passieren, dass sich Fahrzeuge zum Zeitpunkt der ersten Messung, die in der Logdatei aufgenommen wird, bereits über einer Messstation befinden. Für solche Fahrzeuge gibt es dann nur einen Eintrag in der Logdatei, laut Logdatei haben diese Fahrzeuge zu keinem Zeitpunkt diese Messstation erreicht.

Ein ähnliches Problem ergibt sich, wenn die Protokollierung beendet wird, die Simulation aber noch weiter läuft. Dann gibt es Fahrzeuge, die laut Logdatei eine bestimmte Messstation nie verlassen.

2.1.3 Die Begriffe Simulationsdatei und Messreihe

Unter dem Begriff Simulationsdatei werden alle Informationen zusammengefasst, welche benötigt werden, um eine Simulation in VISSIM durchzuführen. Die Simulationsdatei enthält also die Informationen über das Verkehrsnetz.

Der Begriff Messreihe bezeichnet einen Durchlauf einer Simulation. Eine Messreihe kann daher auch eindeutig derjenigen Simulationsdatei zugeordnet werden, aus der sie erzeugt wurde. Vereinfacht gesagt entspricht die Messreihe also der Aktion, die eine Logdatei erzeugt.

Werden zu einer Simulationsdatei mehrere Messreihen gestartet, so unterscheiden sich die Beschreibungen dieser Messreihen, also die Logdateien, nur im zweiten Teil, da der erste Teil nur die Messstationen beschreibt, die unabhängig von der Messreihe sind.

2.2 Besonderheiten einer Simulation

Die Verarbeitung simulierter Daten bereitet aus Datenbanksicht einige Schwierigkeiten. In Datenbanken spielen Konsistenzbedingungen wie Schlüssel und Eindeutigkeiten eine wichtige Rolle. Aus Simulationssicht machen diese jedoch Probleme, da beispielsweise das Fahrzeug mit der Nummer 911 innerhalb einer Messreihe zwar eindeutig ist, sobald aber mehrere Messreihen in der Datenbank gespeichert werden, ist eine Eindeutigkeit nicht mehr gegeben. Im Gegensatz dazu ist eine Messstation, wie wir bereits gesehen haben, solange eindeutig, wie nur Messreihen in die Datenbank eingefügt werden, die aus der gleichen Simulationsdatei erzeugt wurden. Sobald eine andere Simulationsdatei zugrunde liegt, ist eine Eindeutigkeit im Allgemeinen nicht gegeben.

Jedes Element, welches in die Datenbank eingefügt wird, kann also nur in Verbindung mit einer Simulationsdatei oder einer Messreihe eindeutig identifiziert werden.

Messstationen und Strecken sind nur innerhalb einer Simulationsdatei eindeutig. Diese können also auch nur in Verbindung mit einer Simulationsdatei eindeutig identifiziert werden. Da in VISSIM für jede Simulationsdatei eigene Fahrzeugtypen modelliert werden können, gilt dies auch für jeden Fahrzeugtyp.

Ein Fahrzeug ist, wie bereits gesehen, nur in Verbindung mit einer Messreihe eindeutig. Zusätzliche Angaben können eine Messreihe genauer beschreiben. Dazu gehören Simulationsparameter, die verwendeten Einheiten und das verwendete Fahrverhalten. Aus den Simulationsparametern können beispielsweise die Simulationszeit oder die Simulationsgeschwindigkeit herausgelesen werden. In den Einheiten ist angegeben, welche Längen- und Geschwindigkeitsmaße der Logdatei zugrunde liegen. Fahrverhalten gibt beispielsweise den Mindestabstand zum Vorderfahrzeug an. Simulationsparameter, Einheiten und Fahrverhalten lassen sich alle drei in VISSIM angeben.

2.3 Besonderheiten der Simulation im relationalen Schema

Eine der Relationen, welche im relationalen Schema der Nutzdatenbank, kurz Nutzdaten-schema, angelegt wurden, ist die Relation Fahrzeug. An dieser lassen sich alle Besonderheiten demonstrieren. Daher wird diese Relation herangezogen, um in diesem Unterkapitel die Besonderheiten zu erklären.

2.3.1 Umsetzung der Eindeutigkeit

Im Nutzdaten-schema wurden die im letzten Unterkapitel beschriebenen Eindeutigkeiten mit Unique-Constraints durchgesetzt. Das SQL-Kommando zur Erzeugung dieser Relation sieht folgendermaßen aus (auf die Angabe von Fremdschlüsselbedingungen wurde an dieser Stelle bewusst verzichtet):

```
CREATE TABLE OVIDSA01.FAHRZEUG (  
    FZ_NR NUMBER(9) NOT NULL,  
    MESSREIHE_ID NUMBER(4) NOT NULL,  
    FZ_TYP_ID NUMBER(3) NOT NULL,  
    LÄNGE NUMBER(2),  
    CONSTRAINT FahrzeugUnique UNIQUE(FZ_NR, MESSREIHE_ID))  
TABLESPACE USERS  
;
```

Für Fremdschlüsselbeziehungen sind Unique-Constraints allerdings nicht ausreichend.

2.3.2 Künstliche Schlüssel und Fremdschlüssel

Um innerhalb des Nutzdatschemas Fremdschlüsselbeziehungen zu ermöglichen, werden Primärschlüssel benötigt. Ich habe mich für einen einelementigen Primärschlüssel entschieden. Daher wurde in allen Relationen, auf welche in anderen Relationen verwiesen wird, ein künstlicher Schlüssel, also ein zusätzliches Attribut, benötigt. Dieses zusätzliche Attribut wurde in jeder solchen Relation „GENERATED_KEY“ genannt.

Das vollständige Kommando zur Erzeugung der Relation Fahrzeug lautet also:

```
CREATE TABLE OVIDSA01.FAHRZEUG (  
    GENERATED_KEY NUMBER(12) NOT NULL,  
    FZ_NR NUMBER(9) NOT NULL,  
    MESSREIHE_ID NUMBER(4) NOT NULL,  
    FZ_TYP_ID NUMBER(3) NOT NULL,  
    LÄNGE NUMBER(2),  
    CONSTRAINT FahrzeugKey PRIMARY KEY(GENERATED_KEY),  
    FOREIGN KEY(FZ_TYP_ID) REFERENCES OVIDSA01.FAHRZEUGTYP(GENERATED_KEY),  
    FOREIGN KEY(MESSREIHE_ID) REFERENCES OVIDSA01.MESSREIHE(GENERATED_KEY),  
    CONSTRAINT FahrzeugUnique UNIQUE(FZ_NR, MESSREIHE_ID))  
TABLESPACE USERS  
;
```

Hier ist zu beachten, dass Fremdschlüssel immer auf das Attribut GENERATED_KEY der jeweiligen Relation verweisen, auch wenn der besseren Lesbarkeit wegen die Attribute anders benannt sind. Im Beispiel verweist das Attribut FZ_TYP_ID der Relation Fahrzeug auf das Attribut GENERATED_KEY der Relation Fahrzeugtyp, und nicht auf das Attribut FZ_TYP_ID dieser Relation, da dieses nicht Primärschlüssel ist.

Natürlich genügt es nicht, im Schema einen Primärschlüssel vorzusehen, da dieser auch einen eindeutigen Wert enthalten muss. Daher wurde zu jedem künstlichen Schlüssel eine Sequenz angelegt, die, wenn sie aufgerufen wird, eindeutige Einträge erzeugt.

Die Erzeugung einer solchen Sequenz sieht in SQL folgendermaßen aus:

```
CREATE SEQUENCE FahrzeugSequence INCREMENT BY 1 START WITH 1;
```

Beim ersten Aufruf wird also der Wert 1 vergeben, bei jedem weiteren Aufruf die nächsthöhere Zahl. Der Aufruf einer Sequenz erfolgt innerhalb einer Insert Anweisung. Diese sieht dann für unser Beispiel wie folgt aus:

```
INSERT INTO FAHRZEUG VALUES (FahrzeugSequence.NEXTVAL, 911, 42, 23, 4.40);
```

Sequenzen sind also ein sehr einfaches Mittel, um künstliche Schlüssel zu erzeugen.

2.4 Umsetzung der Daten in ein relationales Schema

Aus dem Schema in Abbildung 3 wird ersichtlich, dass die Relationen Messreihe und Simulationsdatei eine zentrale Rolle spielen, da jede andere Relation mit einer von diesen beiden in Beziehung steht.

Die Relation Messung hat einen mehrelementigen Primärschlüssel, welcher aus den vier Attributen Messreihe_ID, Messstation_ID, Fahrzeug_ID und TEinfahrt besteht. Ein einzelnes Fahrzeug kann im Laufe einer Messreihe nur zu einem Zeitpunkt an einer bestimmten Messstation sein. Die drei Fremdschlüssel machen die Relation Messung zu einer Verbindungsrelation zwischen den Relationen Fahrzeug, Messreihe und Messstation. In der Relation Messung sind die eigentlichen Messdaten gespeichert.

Aus einer Eigenschaft von VISSIM ergibt sich, dass die Angabe über die Breite eines Fahrzeugs als Attribut der Relation Fahrzeugtyp modelliert wurde, während die Länge eines Fahrzeugs variabel ist, an den Messstationen gemessen wird und als Attribut der Relation Fahrzeug modelliert wurde. Das Attribut Länge wird zwar gemessen, taucht aber als einziger gemessener Wert nicht in der Relation Messung auf, sondern in der Relation Fahrzeug. Dies liegt an der Normalisierung des Schemas; da ein Fahrzeug an mehreren Messstationen gemessen werden kann, besteht zwischen den Relationen Fahrzeug und Messung eine 1:n Beziehung. Das Fahrzeug wird zwischen den Messstationen aber nicht seine Länge verändern.

Für Datums- und Zeitangaben habe ich das SQL-Format Timestamp gewählt, da sich dieses über bestimmte Klassen aus dem java.sql Package mit JDBC erzeugen lässt. Verwendet habe ich dies für das Attribut Dateidatum aus der Relation Simulationsdatei, für das Attribut Messreihendatum aus der Relation Messreihe und für das Attribut Startuhrzeit aus der Relation Simulationsparameter.

Die Grafik auf der folgenden Seite veranschaulicht das Nutzdatenschema.

Folgende Punkte sollen zur Erklärung des Schemas herausgestellt werden:

- Die Relationen Simulationsparameter, Einheiten und Fahrverhalten beschreiben die Eigenschaften einer Messreihe.
- In einer Simulationsdatei sind mehrere Strecken definiert, und jede dieser Strecken kann mehrere Messstationen enthalten.
- Die Relation Messung verbindet die Relationen Fahrzeug, Messreihe und Messstation.
- Eine Messreihe ist genau einer Simulationsdatei zugeordnet. Zu einer Simulationsdatei kann es aber mehrere Messreihen geben.
- Der Fahrzeugtyp, zu dem ein gemessenes Fahrzeug gehört, wird in der Simulationsdatei definiert.

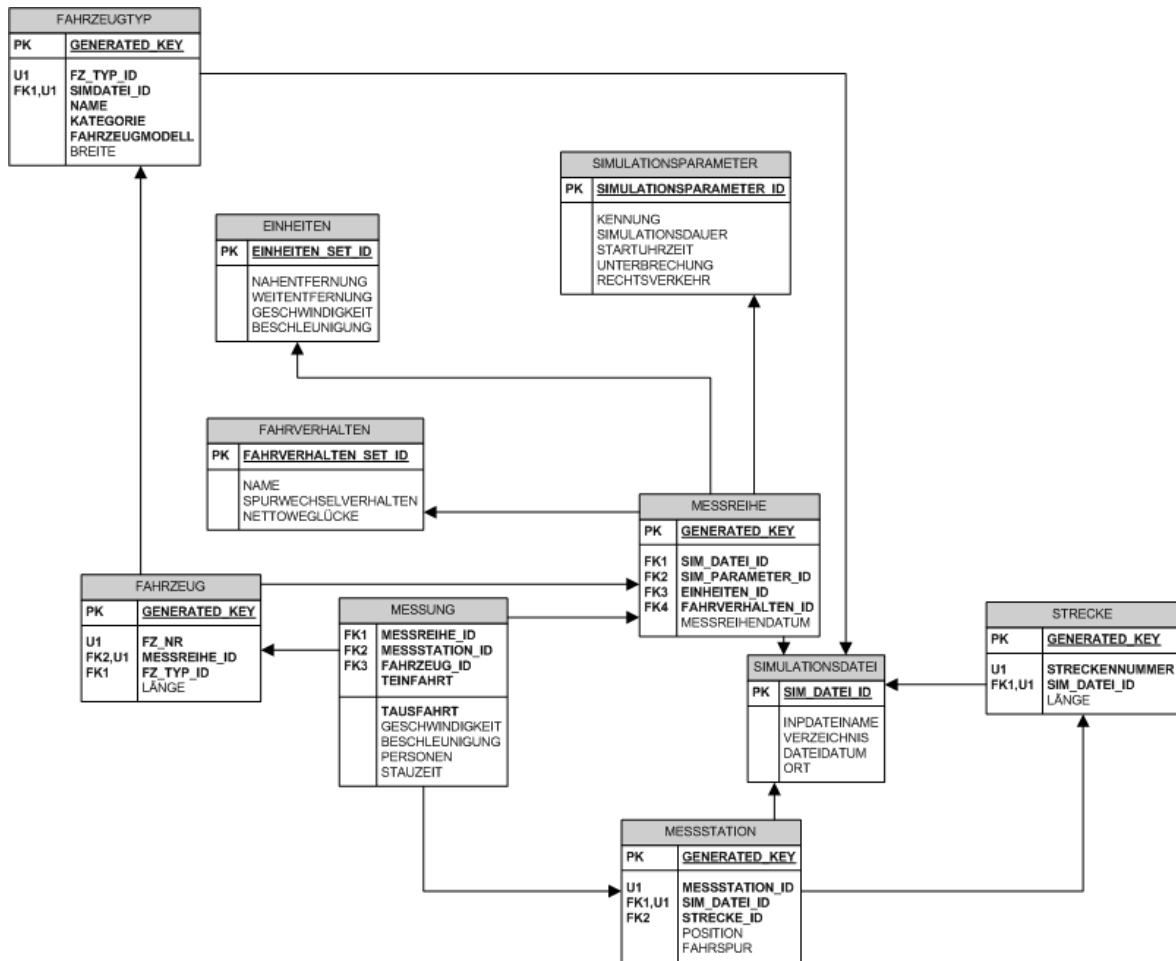


Abb. 3. Das Nutzdatschema

2.5 Beschleunigung des Datenbankankfragen

Die Nutzdatenbank unterscheidet sich von anderen Datenbanken dadurch, dass sehr wenig verschiedene Anfragen direkt an diese Datenbank gestellt werden. Die meisten Anfragen an diese Datenbank sind entweder Einfügeoperationen, die über JDBC angestoßen werden, oder Aktualisierungen des Data Warehouse. Da, wie in Kapitel 4 verdeutlicht, das Data Warehouse durch immer die gleichen SQL-Abfragen aktualisiert wird, bietet sich eine Beschleunigung genau dieser Anfragen mit Hilfe von Indizes an. Das Nutzdatschema ist also nur dazu da, die Daten für das Data Warehouse Schema aufzubereiten. An das Data Warehouse Schema werden dann die komplexen auswertenden Datenbankankfragen gestellt.

2.5.1 Die Einfügeoperationen

Wie in Kapitel 2.3 dargestellt ist in allen Relationen die Eindeutigkeit sowohl des Primärschlüssels als auch einer Kombination von Attributen gefordert. Der größte Beschleunigungsgewinn ist also zu erwarten, wenn die Überprüfung der Eindeutigkeit sehr schnell geht. Indizes sind eine Möglichkeit, dies zu erreichen. Indizes für die Primärschlüssel mussten nicht erstellt werden, da Oracle diese für Primärschlüsseln automatisch erzeugt.

Für diejenigen Nichtprimärschlüsselattribute, für die Eindeutigkeit gefordert wird, wurde einen Index über diese Attribute erzeugt.

Das Einfügen von vielen Tupeln mit sequentiell steigenden Schlüsseln in eine Relation ist vergleichbar mit dem Szenario in [Saa99], Seite 157, das „bulk-loading“ beschreibt. Allerdings müssen die Datensätze nicht, wie in [Saa97] beschrieben, vorher sortiert werden. „Alle Einfügungen konzentrieren sich auf den rechten Pfad des Baumes [...]. Da es wahrscheinlich ist, dass sich die Seiten auf diesem Pfad immer im Puffer befinden werden, kann man eine effiziente Bearbeitung erwarten.“ ([Saa97], S. 157)

2.5.2 Die Leseoperationen

Die Leseoperationen bestehen fast ausschließlich aus Anfragen zur Aktualisierung der materialisierten Sichten im Data Warehouse Schema. Sie beschränken sich also auf 6 verschiedene Select-Anfragen. Auf die einzelnen Anfragen wird in Kapitel 4.2 genau eingegangen. An dieser Stelle sei aber schon vorweggenommen, dass in vier von diesen Anfragen Verbindungsoperationen durchgeführt werden müssen. Da auf Primärschlüssel bereits Indizes definiert sind, habe ich mich darauf beschränkt, auf diejenigen Nichtprimärschlüsselattributen, welche an den Verbindungsoperationen beteiligt sind, jeweils einen Index zu erstellen.

3. Das Füllen der Nutzdatenbank

Es gibt mehrere Möglichkeiten, die Daten aus der Logdatei so auszulesen, dass sie konsistent mit dem Nutzdatenschema sind. Im ersten Versuch beschäftigte ich mich mit dem SQL*Loader, einem Werkzeug zum Einlesen von Daten aus einer Datei in eine Datenbank, welches Teil von Oracle 9i ist. Schnell zeigte sich jedoch, dass der SQL*Loader nicht mächtig genug war, da manche Daten nicht aus einer Datei kommen, sondern in VISSIM angegeben sind. Beispielsweise finden sich die Simulationsparameter in keiner Datei, genauso wie Angaben über die Fahrzeugtypen. Außerdem gibt es Daten, die zwar in der Datenbank stehen sollen, aber weder in der Logdatei noch in VISSIM zu finden sind. So zum Beispiel das Messreihendatum. Es müssen also viele Daten „von Hand“ in die Datenbank eingetragen werden, auf die es auch Fremdschlüsselverweise gibt.

Da es nicht praktikabel und zumutbar ist, für jede Messreihe die zugehörigen Simulationsparameter etc. über SQL-Kommandos in die Datenbank einzutragen, entschloss ich mich, eine JDBC-Anbindung an die Datenbank zu schreiben und so die Eingabe der Simulationsparameter, Fahrzeugtypen etc. komfortabler zu gestalten. Da der SQL*Loader nicht mächtig genug war, beschloss ich, den gesamten Vorgang des Befüllens der Nutzdatenbank über eine JDBC-Anbindung in Java zu implementieren.

JDBC birgt den großen Nachteil der schlechten Performanz. Ich habe mich dennoch für JDBC entschieden, da es möglich ist, die Logdatei in die Datenbank einzulesen, noch während VISSIM in die Logdatei hineinschreibt. Die Simulation und das Schreiben der Logdaten in die Nutzdatenbank können also parallel laufen. Das bedeutet weniger zusätzlichen Zeitaufwand zum Füllen der Nutzdatenbank nach Ende des Simulationsdurchlaufs.

3.1 Das Projekt *vissimLoader*

Im Projekt *vissimLoader* wurde der Ladevorgang der Logdatei in die Nutzdatenbank implementiert.

3.1.1 Voraussetzungen

Der Aufruf des *vissimLoader* muss innerhalb des institutseigenen Netzwerks im IPD erfolgen. Die Datenbank ist von externen Rechnern außerhalb des IPD durch eine Firewall abgeschirmt und nicht erreichbar. Hier kann auch ein SSH Tunnel keine Abhilfe schaffen, da die JDBC-Komponente des *vissimLoader* mit dem Datenbankserver dynamisch einen Port aushandelt, über den sie kommunizieren. Die Nummer des Ports kann also nicht bei der Konfiguration des SSH-Tunnels angegeben werden.

Von Oracle werden Bibliotheken für eine JDBC-Anbindung zur Verfügung gestellt. Diese Bibliotheken müssen als JAR-Archiv der Java-Umgebung bekannt sein. Benötigt wird die Version für Java 1.4 und Oracle 9i.

3.1.2 Benutzung

Das Hauptprogramm des `vissimLoader` liegt in der Klasse `File2Database`. Das Hauptprogramm erzeugt eine Instanz des Objekts `FileManager` und reicht die Aufrufparameter an dieses weiter. Als Aufrufparameter wird der Pfad zu einer Logdatei erwartet. Es kann sich dabei entweder um einen absoluten oder einen relativen Pfad handeln. Ein Aufruf von einer Kommandozeile sieht also folgendermaßen aus:

```
java File2Database S:\Verzeichnis\logdatei.mer
```

Wird das Projekt aus der Entwicklungsumgebung Eclipse heraus gestartet, muss der Dateiname in der Konfiguration als Programmargument angegeben werden. Hierbei ist zu beachten, dass Eclipse selbst in Java programmiert ist und daher das Zeichen `,` `\` als Steuerzeichen interpretiert. Als Aufrufargument muss daher

```
S:\\Verzeichnis\\logdatei.mer
```

angegeben werden.

Nur in die vier Relationen `Messung`, `Fahrzeug`, `Messstation` und `Strecke` werden Daten aus der Logdatei eingefügt. Die anderen Relationen müssen von Hand eingegeben werden. Nach dem Aufbau einer Verbindung zur Datenbank ist dies auch der erste Schritt des `vissimLoader`. Dazu wird eine Liste aller gespeicherten Tupel der Relationen `Fahrverhalten`, `Einheiten`, `Simulationsparameter` und `Simulationsdatei` angezeigt. Das Programm stellt für jede dieser Relationen die Option, ein vorhandenes Tupel als Referenztuplel zu verwenden oder ein neues zu erzeugen.

Die Werte für die Relation `Fahrverhalten` sind abrufbar als Unterpunkt des Menüs `Simulation`. In einem anderen Unterpunkt sind auch die `Simulationsparameter` zu finden.

Die Werte für die Relation `Einheiten` können als Unterpunkt des Menüpunktes `Optionen` angegeben werden. In den Verkehrswegenetzen, welche zu Demonstrationszwecken VISSIM beiliegen und auf deren Daten diese Studienarbeit aufbaut, wird immer mit den 4 Parametern `Meter`, `Kilometer`, `Meter pro Sekunde` und `Kilometer pro Stunde` gearbeitet.

Die Werte für die `Simulationsdatei` sind nicht in VISSIM angegeben. Es handelt sich um den Pfad zur `Simulationsdatei` (genauer: den Namen und das Verzeichnis der `*.inp` Datei) und den Ort, welchen diese `Simulationsdatei` simuliert.

Anschließend werden alle Fahrzeugtypen angezeigt, die zu einer bestimmten `Simulationsdatei` gespeichert sind. Auch hier können weitere Fahrzeugtypen hinzugefügt werden. Die Beschreibung der Fahrzeugtypen findet man als Unterpunkt des Menüpunktes `Netzeditor`. Durch das Auswählen des Feldes „Editieren“ werden zu jedem Fahrzeugtyp alle relevanten Daten aufgerufen.

Jetzt wird eine neue Messreihe erzeugt, welche neben den Primärschlüsselwerten mancher der eingegebenen Tupel als Fremdschlüssel auch einen Zeitstempel auf die aktuelle Zeit enthält.

Damit sind alle Werte bekannt, die als Fremdschlüssel in die Relationen `Messung`, `Fahrzeug`, `Messstation` und `Strecke` eingetragen werden müssen.

3.2 Das Kopieren der Daten

Nun beginnt das Einlesen der Logdatei. Dies geschieht zeilenweise. Für jede Zeile werden drei Schritte durchgeführt:

1. komplette Zeile aus der Datei einlesen
2. zerlegen der Zeile in ihre einzelnen Bestandteile.
3. überprüfen, ob diese Zeile als Tupel in die Datenbank eingetragen werden muss und evtl. eintragen des Tupels in die Datenbank.

3.2.1 Einfügen von Messstationen und Strecken

Das Einfügen der Messreihen aus dem ersten Teil der Logdatei gestaltet sich recht einfach. Nachdem die Zeile eingelesen und in ihre Bestandteile zerlegt wurde, wird ein Objekt vom Typ `MessstationLine` erzeugt. Dieses Objekt enthält alle Daten, die zum Einfügen einer Messstation und einer Strecke erforderlich sind. Wird aus einer Simulationsdatei mehrmals eine Messreihe erzeugt, so müssen und können die Werte für die Messstationen nicht mehrmals in die Datenbank eingetragen werden. Dies wird daher vor dem Einfügen überprüft. Dabei kann nicht anhand des Primärschlüssels überprüft werden, ob ein Tupel bereits vorhanden ist. Dies muss mit Hilfe der bereits beschriebenen Eindeutigkeitsforderungen im Nutzdatsenschema geprüft werden.

3.2.2 Einfügen von Fahrzeugen und Messungen

Auch Fahrzeuge müssen nicht immer in die Datenbank eingefügt werden, da ein Fahrzeug innerhalb einer Messreihe an mehreren Messstationen gemessen werden kann. Auch hier ist vor dem Einfügen eine Prüfung notwendig, ob das Fahrzeug bereits in der Datenbank vorhanden ist. Auch diese Prüfung kann nicht mit Hilfe des Primärschlüssels passieren.

Wie schon in der Beschreibung der Logdaten in Kapitel 2.1.2 erwähnt gibt es grundsätzlich zwei verschiedene Messungen. Die eine bei der Einfahrt in eine Messstation, bei welcher der Wert für `TAusfahrt` -1.00 beträgt, die andere bei der Ausfahrt aus einer Messstation mit -1.00 als Wert für `TEinfahrt`.

Wird eine Zeile eingelesen, deren Wert für `TAusfahrt` -1.00 beträgt, so wird diese Zeile in die Datenbank eingefügt. Zusätzlich wird ein Objekt vom Typ `KomplexeMessungLine` erzeugt, welches neben der `MessungLine` noch den von der Datenbank automatisch erzeugten Schlüssel des aktuell gemessenen Fahrzeugs und den Schlüssel der aktuellen Messreihe enthält. Dieses Objekt wird in einer Kollektionsklasse vom Typ `java.util.Vector` mit dem Namen `messungenOhneAusfahrt` gespeichert.

Wird eine Zeile eingelesen, deren Wert für `TEinfahrt` -1.00 beträgt, so wird im Vektor `messungenOhneAusfahrt` derjenige Eintrag gesucht, welcher den gleichen Wert in der Variablen `MessungLine.fzNummer` hat wie die aktuell eingelesene Zeile. Diese Zeile enthält damit also den Wert für `TAusfahrt`, der in der letzten Zeile mit der gleichen Fahrzeugnummer fehlt. Anhand des automatisch erzeugten Schlüssels, der im Objekt `komplexeMessungLine` mitgespeichert ist, kann das Tupel in der Datenbank eindeutig qualifiziert werden, welches noch keinen Wert für `TAusfahrt` enthält. Da es mehrere Messungen zu einem Fahrzeug geben kann, handelt es sich um diejenige Messung mit dem höchsten Wert für `TEinfahrt`. In

diesem Tupel wird der Wert für TAusfahrt aktualisiert. Anschließend wird das entsprechende Element aus dem Vektor `messungenOhneAusfahrt` entfernt.

Jeder Messwert (z.B. Fahrzeuglänge) ist also doppelt in der Logdatei vorhanden, in derjenigen Zeile, welche ein Einfahrt in die Messstation beschreibt, und in jener, welche die Ausfahrt beschreibt. Die Messwerte für Geschwindigkeit, Beschleunigung und evtl. TStau sind in beiden Zeilen unterschiedlich. Ich habe mich jedoch dazu entschlossen, jeweils nur die Werte aus der Einfahrtszeile zu speichern. Zum Einen denke ich, dass diese Werte für die Auswertung der Simulationsdaten ausreichend sind. Ein anderer Grund liegt darin, dass diese Arbeit die Basis für die Auswertung realer Messdaten darstellt, welche an realen Messstationen vorgenommen werden. Laut [Sie2003], Seite 8, verwendet Siemens Induktionsschleifen in der Straßenoberfläche, passive Infrarotsensoren oder digitalisierte Videobilder. Diese wirklich verwendeten Sensoren messen jedes Fahrzeug ebenfalls nur einmal.

3.2.3 Unvollständige Messungen

Wie in Kapitel 2.1.2 beschrieben gibt es in der Logdatei Messungen, zu denen es keine Ein- bzw. Ausfahrten gibt. Die einen entstehen am Anfang der Protokollierung, die anderen am Ende. Da diese Datensätze teilweise inkonsistent sind, musste ich dafür Sorge tragen, dass diese Datensätze nicht dauerhaft in der Datenbank abgelegt werden.

Diejenigen Zeilen der Logdatei, welche für TEinfahrt den Wert `-1.00` enthalten und zu denen es keine passenden Zeilen gibt, sind wie besprochen diejenigen, welche zu Beginn der Protokollierung entstehen. Eine solche Zeile wird mit allen Einträgen im Vektor `messungenOhneAusfahrt` verglichen. Da es für diese Zeile keinen passenden Eintrag in diesem Vektor gibt, wird diese Zeile nicht in die Datenbank eingetragen. Es erscheint eine entsprechende Meldung auf der Textkonsole.

Am Ende der Protokollierung gibt es Zeilen, welche für TAusfahrt den Wert `-1.00` enthalten und zu denen es keine Werte für TEinfahrt mehr in der Logdatei gibt. Diese Zeilen stehen jedoch bereits in der Datenbank als inkonsistente Daten. Es handelt sich jedoch auch genau um diejenigen Datensätze, welche im Vektor `messungenOhneAusfahrt` am Ende der Auswertung der Logdatei noch enthalten sind. Die entsprechenden Tupel können mit Hilfe der Variablen im Objekttyp `komplexeMessungLine` eindeutig identifiziert und aus der Datenbank gelöscht werden. Dies ist die letzte Aktion, welche `vissimLoader` ausführt.

3.3 Umsetzung der Besonderheiten einer Simulation

Eine Besonderheit der simulierten Daten, welche für die Umsetzung des Befüllungsvorganges von Bedeutung war, war die Tatsache, dass für die meisten Einträge künstliche Schlüssel erzeugt wurden. Dies machte sich an zwei Stellen bemerkbar.

Zum Einen in der SQL Anweisung, welche das entsprechende Tupel in der Datenbank erzeugt. Dort wurde jeweils statt des Schlüsselwertes die Operation `SEQUENCE.NEXTVAL` angegeben.

Eine andere Problematik ergab sich dadurch, dass diese Schlüsselwerte aus anderen Relationen referenziert werden und daher bekannt sein müssen. Daher habe ich die Methoden, welche die Einfügeoperationen durchführen, als Funktionen mit einem Rückgabewert (`long` oder `int`) implementiert. Am Ende dieser Funktionen wird immer der Schlüssel des

zuletzt eingefügten Tupels aus der Datenbank gelesen. Hierbei habe ich die Eigenschaft von Sequenzen genutzt, monoton steigende Reihen zu erzeugen. Das letzte eingefügte Element ist dann dasjenige mit dem größten numerischen Schlüsselwert.

Folgende Programmtextskizze verdeutlicht, wie die Einfügeoperationen implementiert sind:

```
protected long insertFahrzeug(messungLine messung, int messreihe)
{
    sqlString = „INSERT INTO FAHRZEUG VALUES (FAHRZEUGSEQUENCE.NEXTVAL,
                messreihe, messung.fzTyp, messung.fzLaenge“;
    executeUpdate(sqlString);
    sqlString2 = „SELECT MAX (GENERATED KEY) FROM FAHRZEUG“;
    long schlüssel = execute(sqlString2);
    return schlüssel;
}
```

3.4 Datentypen des vissimLoader

Für die Relationen Einheiten, Fahrverhalten, Fahrzeugtyp, Simulationsdatei und Simulationsparameter habe ich jeweils einen eigenen Datentyp definiert. In diesem Datentyp habe ich eine Methode `printHeader()` und eine Methode `toString()` implementiert. Die Methode `printHeader()` schreibt die Namen der Attribute auf die Standardausgabe. Die Methode `toString()` gibt alle Werte der Attribute so aus, dass die Attribute unter dem Attributnamen stehen, welche die so aus, dass die Attribute unter dem Attributnamen stehen, welche die Methode `printHeader()` ausgibt. Um einen Vektor von Elementen eines bestimmten Datentyps in Tabellenform auf der Standardausgabe auszugeben, genügt folgender (gekürzter) Programmtext.

```
void printVector(Vector v)
{
    if (v.elementAt(0) instanceof Simulationsdatei)
    {
        dataTypes.Simulationsdatei.printHeader();
        for (int I; I < v.size(), I++)
        {
            simdatei = (Simulationsdatei) v.elementAt(i);
            System.out.println(simdatei.toString());
        }
    }
}
```

Die Methode `printVector()` in der Klasse `FileManager` ist auf diese Art aufgebaut. Sie kennt noch weitere Datentypen außer `Simulationsdatei`, wie z.B. `Fahrzeugtyp` oder `Einheiten`.

Die Methode `printVector()` geht davon aus, dass alle Elemente im Vektor den gleichen Datentyp haben. Diese Annahme ist auch gerechtfertigt, da die auszugebenden Vektoren in entsprechenden Funktionen in der Klasse `JDBCQuery` erzeugt wurden. Eine dieser Methoden ist beispielsweise die Methode `getSimulationsdateien()`, welche alle Tupel der Relation `Simulationsdatei` aus der Datenbank ausliest und aus jedem Tupel ein Objekt vom Typ `Simulationsdatei` erzeugt.

Zusätzlich wurde noch die bereits erwähnten Datentypen `MessungLine`, `KomplexeMessungLine` und `MessStationLine` implementiert. Für diese wurde keine Methode `toString()` imp-

lementiert, sondern nur die entsprechenden `get()` und `set()` Funktionen für die jeweiligen Attribute definiert.

3.5 Einschränkungen, Annahmen, sonstige Voraussetzungen

3.5.1 Das Format der Logdatei

Bei der Implementierung des `vissimLoader` musste ich einige Annahmen machen und Einschränkungen vornehmen. Die wichtigste Annahme ist natürlich, dass das Format, in welchem die Logdatei vorliegt, ein einheitliches Format ist. Dies ist für die Version 3.70 von VISSIM auch gegeben.

Da sich dieses Format in einer späteren Version von VISSIM ändern könnte, sei hier darauf hingewiesen, an welchen Stellen im Programmtext Änderungen vorzunehmen sind.

Die „äußere Form“ der Datei, also alles, was nicht in den Datenzeilen für die Messstationen und Messungen steht, wird in der Methode `readMerData()` in der Klasse `FileManager` überprüft. Diese Methode erwartet, dass in der ersten Zeile der Logdatei „Messungsprotokoll (Rohdaten)“ steht, gefolgt von einer Leerzeile. In der dritten Zeile ist eine geographische Angabe zu finden, an welche sich wieder eine Leerzeile anschließt. Es folgt die Beschreibung der Messstationen. Diese endet, wenn eine Leerzeile eingelesen wird. Im Anschluss an diese Leerzeile stehen die eigentlichen Messdaten.

Sollte sich am dem Format etwas ändern, mit welchem die Messstationen beschrieben werden, so sind Anpassungen in der Methode `readMessstationenLine()` auch in der Klasse `FileManager` nötig. Änderungen im Format der Messungen bedürfen einer Anpassung der Methode `readMessungenLine()`, welche auch in der Klasse `FileManager` implementiert ist.

3.5.2 Fahrzeugkategorien

Zwar sind Fahrzeugtypen von der Simulationsdatei abhängig, jeder Fahrzeugtyp gehört aber zu einer bestimmten Kategorie, welche unabhängig von der Simulationsdatei sind. Folgende Kategorien werden von `vissimLoader` unterstützt:

- Pkw
- Bus
- Zweirad
- Lkw
- Bahn
- Fußgänger

Wenn in späteren Versionen von VISSIM weitere Fahrzeugkategorien unterstützt werden, ist eine Anpassung der Methode `fahrzeugtypenEinlesen()`, ebenfalls in der Klasse `FileManager`, vorzunehmen.

4. Das Data Warehouse Schema

Aus Performanzgründen läuft ein Data Warehouse oft auf einem anderen Datenbankrechner als demjenigen Rechner, dessen Datenbank es auswerten soll. Zusätzlich ist es oft so, dass ein Data Warehouse Daten aus vielen verschiedenen Datenquellen zusammenführen und auswerten muss.

4.1 Database Links

Das Wichtigste, um im Warehouse Schema die Nutzdaten benutzen zu können, ist die Möglichkeit, auf die Daten zugreifen zu können. Da die Daten aber auf einem anderen DBMS zu finden sind, musste erst der Zugriff auf das andere DBMS geregelt werden. Dies wird in Oracle über Database Links geregelt.

Die Syntax zur Erzeugung eines Database Link geht aus folgendem Syntaxdiagramm hervor. Für genauere Erläuterungen dazu sei auf die Oracle SQL Reference verwiesen (z.B. [OraRef03])

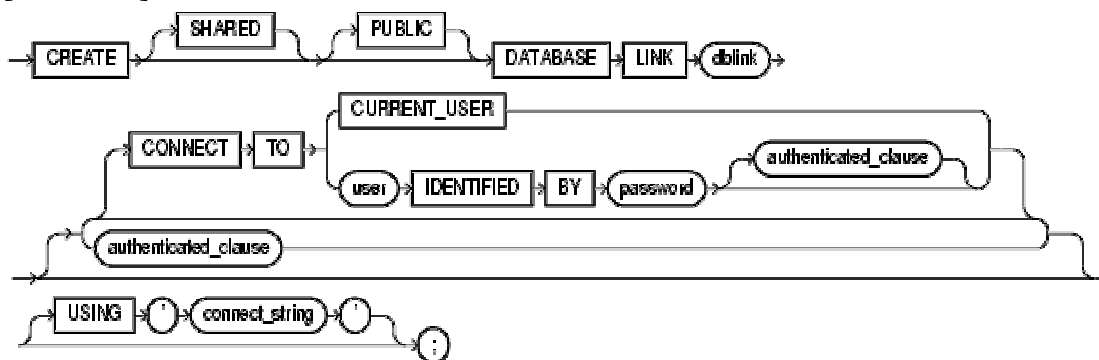


Abb. 4. Syntaxdiagramm zur Erzeugung eines Database Link

Die Angabe des Zusatzes „@dblink“ an einen Relationennamen in einem SQL-Kommando gibt nun an, dass diese Relation auf dem externen DBMS, welches durch den connect_string beschrieben ist, im Schema current_user oder user zu finden ist. Eine SQL Anweisung hat dann folgende Form:

```

SELECT *
FROM FAHRZEUG@OVID01;
    
```

Voraussetzung für die Benutzung von Database Links ist, dass das externe DBMS als ODBC Datenquelle bekannt ist. Dafür müssen folgende Zeilen in die Konfigurationsdatei tnsnames.ora von Oracle eingetragen sein:

```
OVID01 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)
                 (HOST = hostname)
                 (PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = servicename)
    )
  )
)
```

4.2 Materialisierte Sichten

Materialisierte Sichten bieten wie normale Sichten auch Zugriff auf bestimmte Attribute von Relationen und bieten die Möglichkeit, auf berechnete Zwischenergebnisse zuzugreifen. Eine Sicht wird dabei durch ein SQL-Kommando definiert, dessen Syntax hinreichend bekannt ist.

Materialisierte Sichten unterscheiden sich von normalen Sichten dadurch, dass sie nicht nur virtuell sind, sondern die Sicht als Relation im DBMS abgelegt wird. Dies bietet besonders dann Vorteile, wenn das SQL Kommando aufwendige Berechnungen enthält, welche bei Verwendung einer materialisierten Sicht nicht bei jedem Zugriff auf die Sicht neu berechnet werden müssen. Auch wenn eine Sicht auf ein externes DBMS verweist, kann es sinnvoll sein, diese Sicht zu materialisieren, wenn die Datenmenge entsprechend groß und die Kapazität der Netzwerkverbindungen begrenzt ist.

Die Aktualität der Daten stellt bei der Verwendung einer normalen Sicht kein Problem dar, da diese immer auf die Basisrelationen zugreift. Bei der Definition von materialisierten Sichten muss aber angegeben werden, was zu tun ist, wenn sich die Basisrelation(en) ändern.

An einem Beispiel möchte ich die Definition einer materialisierten Sicht erklären:

```
CREATE MATERIALIZED VIEW fahrzeugtypdim
BUILD IMMEDIATE
REFRESH FORCE
ON DEMAND
AS
select f.generated_key, f.fz_typ_id, f.simdatei_id, f.name,
       f.kategorie, f.fahrzeugmodell, f.breite
from fahrzeugtyp@ovid01 f
;
```

Diese materialisierte Sicht erzeugt also eine exakte Kopie der Relation Fahrzeugtyp. Sie entspricht nicht genau der materialisierten Sicht, welche im Rahmen dieser Studienarbeit erstellt wurde. Es fehlen die Attribute „motorisiert“ und „transportiert“, auf die später noch genau eingegangen wird.

Die Definition dieser materialisierten Sicht unterscheidet sich von der einer normalen Sicht in den ersten vier Zeilen (vgl. [Hob01]).

Das Schlüsselwort **MATERIALIZED** gibt dabei an, dass für diese Sicht eine eigene Relation erzeugt werden soll.

Die Angabe von `BUILD IMMEDIATE` gibt an, dass diese Relation direkt beim Erzeugen mit Daten aus den Daten aus der Basisrelation gefüllt werden soll. Die Alternative besteht in der Angabe von `BUILD DEFERRED`. Dies erzeugt zwar die Relation, diese wird aber noch nicht mit Daten gefüllt. Dies ist beispielsweise sinnvoll, wenn auf der Relation, welche durch die materialisierte Sicht erzeugt wird, Trigger zum Einsatz kommen.

Das Schlüsselwort `REFRESH` gibt an, wie die Aktualisierung der materialisierten Sicht passiert. Mögliche Varianten sind `COMPLETE`, `FAST` und `FORCE`. `COMPLETE` weist das DBMS an, die materialisierte Sicht bei jeder Änderung in der Basisrelation komplett neu zu erstellen, während `FAST` nur neue Tupel einfügt bzw. alte Tupel aktualisiert. Die Option `FORCE` führt wenn möglich ein `FAST UPDATE` aus. Ist dies nicht möglich, ein `COMPLETE UPDATE`. Die nächste Option `ON DEMAND` weist das DBMS an, ein Update auf der materialisierten Sicht nur auf Anwenderwunsch auszuführen. Wurde die Option `ON DEMAND` angegeben, wartet das DBMS auf das Kommando zur Aktualisierung der materialisierten Sichten. Dieses Kommando wird über Prozeduren gegeben:

- `DBMS_MVIEW.REFRESH` zum aktualisieren einer bestimmten materialisierten Sicht
- `DBMS_MVIEW.REFRESH_DEPENDAND` zum aktualisieren aller materialisierten Sichten, welche auf eine bestimmte Tabelle als Basisrelation zugreifen
- `DBMS_MVIEW.REFRESH_ALL_MVIEWS` zum aktualisieren aller materialisierter Sichten .

Alternativ gibt es die Option `ON COMMIT`, welches ein Update ausführt, wenn auf der Basisrelation etwas geändert wurde und diese Änderung mit einem Commit Befehl festgeschrieben wurde.

Für eine detailliertere Beschreibung von materialisierten Sichten in Oracle sei auf das entsprechende White Paper von Oracle [Hob01] verwiesen.

4.3 Das Warehouse-Schema als materialisierte Sichten

In der ersten Variante eines Data-Warehouse-Schemas, habe ich materialisierte Sichten erstellt. Die Aktualisierung dieser Sichten geschieht in denjenigen materialisierten Sichten, deren Definition keine Verbindungsoperation enthält, `REFRESH FORCE ON DEMAND`. Für die anderen materialisierten Sichten ließ Oracle nur die Option `REFRESH COMPLETE ON DEMAND` zu. Die Option `FORCE` bietet sich besonders an, da selten Tupel im Nutzdatschema aktualisiert, sondern meist Tupel eingefügt werden. Die Option `ON DEMAND` wurde aufgrund der Tatsache gewählt, dass das Füllen des Nutzdatschemas über JDBC abläuft. Der `vissimLoader` erzeugt für jede Zeile der Logdatei ein Statement. Ist dieses Statement beendet, wird auf der Nutzdatsbank in den veränderten Relationen ein Commit Befehl ausgeführt. Die Option `ON COMMIT` in der Definition der materialisierten Sichten würde also bedeuten, dass beim Einlesen der Logdatei nach dem Eintrag jeder Zeile ein Update auf dem Data Warehouse ausgeführt wird. Die Leistungsfähigkeit der Nutzdatsbank würde damit zu sehr eingeschränkt.

Für das Data Warehouse Schema habe ich, wie in [BaGü01] vorgeschlagen, versucht, ein Star Schema umzusetzen. Auf die Gründe, weshalb sich ein reines Star Schema nicht durchsetzen ließ, wird in Kapitel 4.3.3 eingegangen. In einem Star Schema ist eine zentrale Faktentabelle vorgesehen, in welcher zum einen die zentralen Daten, in unserem Fall die

Messungen, stehen, zum anderen Verweise auf sog. Dimensionstabellen. In den Dimensionstabellen sind solche Daten gespeichert, welche sich in Kategorien einteilen lassen. Ein in der Literatur oft gewähltes Beispiel ist die Dimension Zeit. Eine Messung gehört zu einem bestimmten und eindeutigen Zeitpunkt. Dieser Zeitpunkt wird als Fremdschlüssel in die Faktentabelle aufgenommen. Die entsprechende Dimensionstabelle enthält Angaben darüber, zu welcher Sekunde dieser Zeitpunkt gehört, zu welcher Minute diese Sekunde gehört etc. Damit lassen sich beispielsweise sehr einfach alle Messungen herausfinden, welche in einer bestimmten Stunde aufgezeichnet wurden.

Dieses Schema wird durch die Stapelverarbeitungsdatei `createMaterializedViews.sql` erzeugt. Zum Erzeugen dieses Schemas werden Systemprivilegien zum Erstellen von Tabellen, materialisierten Sichten und Funktionen benötigt.

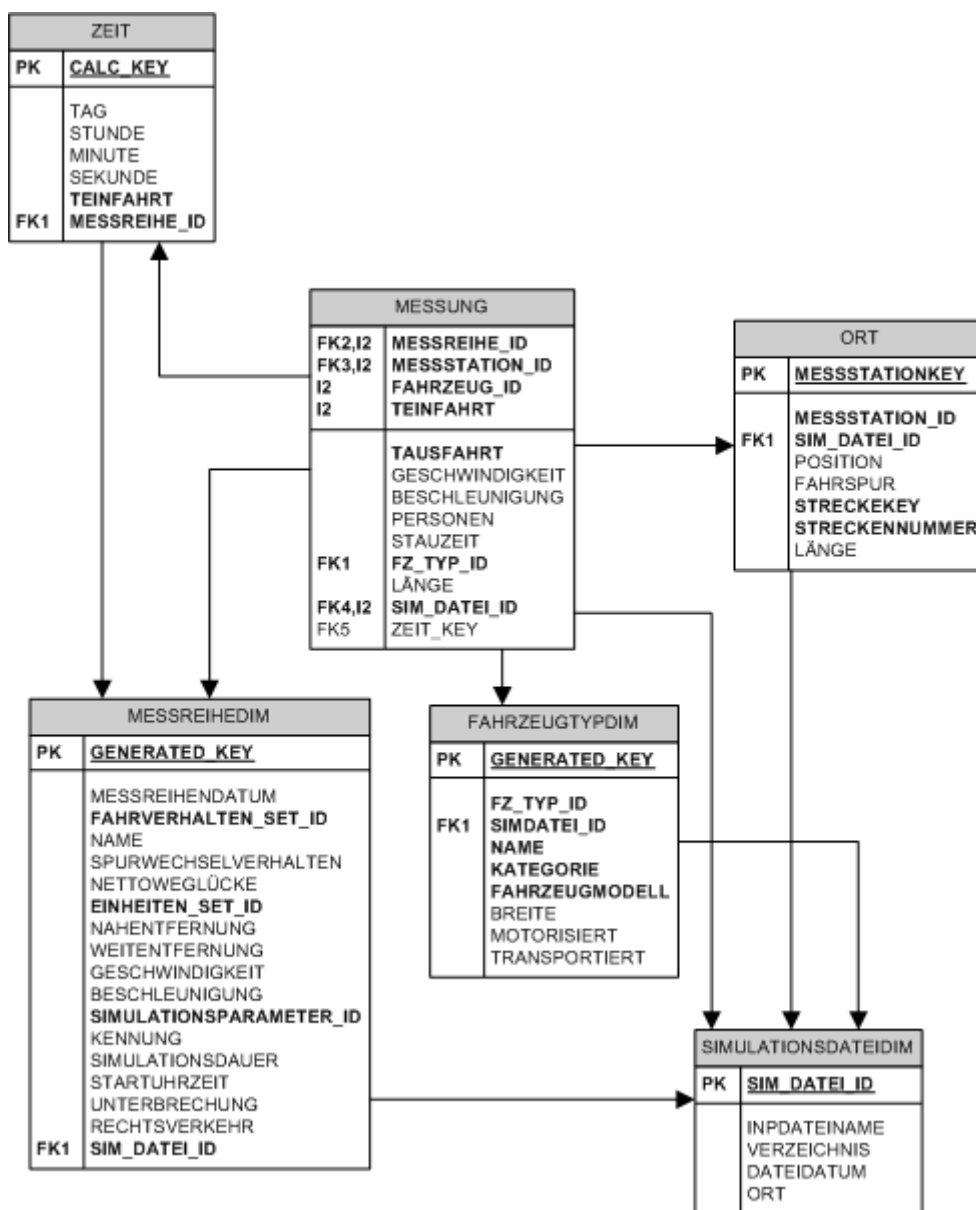


Abb. 5. Das Schema des Data Warehouse als materialisierte Sichten

4.3.1 Die Faktentabelle

Die materialisierte Sicht Messung im Data Warehouse ist aus einer natürlichen Verbindungsoperation der Relationen Messung und Fahrzeug im Nutzdatschema entstanden. Aus zwei Gründen habe ich mich dazu entschlossen, die Fahrzeugdaten mit in der Relation Messung zu speichern. Zum einen ist es ein intuitives Vorgehen, da alle Attribute der Relation Fahrzeug (bis auf das Attribut Messreihe_ID und dem künstlich generiertem Primärschlüssel) aus Messungen entstammen. Der andere Grund liegt darin, dass bei der vereinigten Relation nur das Attribut Fahrzeuglänge zu den Attributen der Relation Messung hinzukommt. Die Relation Messung wird also nur in geringem Maße umfangreicher. Das bedeutet, dass sehr wenig Redundanzen zu erwarten sind. Redundanzen treten genau in dem Fall auf, in welchem ein Fahrzeug innerhalb einer Messreihe an mehreren Messstationen gemessen wird. In diesem Fall ist die mehrfache Angabe der Fahrzeuglänge redundant. Durch diese geringe Redundanz entstehen keine Einfügeanomalien. Löschanomalien sind nicht zu erwarten, da aus einem Data Warehouse normalerweise keine Daten gelöscht werden.

Der große Vorteil der natürlichen Verbindung der Basisrelationen Messung und Fahrzeug liegt darin, dass beim Auswerten der Daten teure Verbindungsoperationen eingespart werden.

4.3.2 Die Dimensionstabellen

Auch die Dimensionstabellen sind teilweise aus natürlichen Verbindungsoperationen entstanden. Beispielsweise wurden in der Dimensionstabelle MESSREIHEDIM alle Angaben, welche eine Messreihe genauer beschreiben, zusammengefasst. Dies sind neben der Relation Messreihe noch die Basisrelationen Einheiten, Simulationsparameter und Fahrverhalten. Die Tabelle Ort entstand aus einer natürlichen Verbindung der Basisrelationen Messstation und Strecke. Die Tabelle SIMULATIONSDATEIDIM im Data Warehouse entspricht genau der entsprechenden Basisrelation.

4.3.2.1 Die Dimensionstabelle FAHRZEUGTYPDIM

Die Dimensionstabelle FAHRZEUGTYPDIM enthält zusätzlich zu den Attributen aus der Basisrelation die Attribute MOTORISIERT und TRANSPORTIERT. Das Attribut Motorisiert enthält entweder den Wert ‚Motorisiert‘ oder ‚Nicht Motorisiert‘. Das Attribut TRANSPORTIERT enthält entweder den Wert ‚Person‘ oder den Wert ‚Güter‘. Zur Entscheidung, welcher Wert das jeweilige Attribut enthalten soll, habe ich zwei Funktionen geschrieben, welche beim Einfügen eines Tupels in die Dimensionstabelle aufgerufen werden. Diese Funktionen werden zusammen mit dem Data Warehouse Schema durch die Stapelverarbeitungsdatei `createMaterializedViews.sql` erstellt.

Da in den Demonstrationsdateien, welche mit VISSIM mitgeliefert werden, Bahnen immer Straßenbahnen waren, transportiert ein Fahrzeug, welches zur Kategorie Bahn gehört, Personen, und keine Güter. Ein Zweirad wird als nicht motorisiert angenommen. Ansonsten entsprechen die Werte für TRANSPORTIERT und MOTORISIERT den intuitiven Vorstellungen dieser Fahrzeugkategorien.

4.3.2.2 Die Dimensionstabelle ZEIT

Die Dimensionstabelle Zeit ist aus einer Verbindungsoperation der Basisrelationen Messung und Messreihe entstanden, wobei nur die Attribute `TEinfahrt`, `Startuhrzeit` und `Messreihe_ID` zur Berechnung anderer Attribute in der materialisierten Sicht verwendet werden. Das Problem in dieser Relation war, einen geeigneten Schlüssel zu finden. Das Attribut `TEinfahrt` allein ist wie bereits gesehen nicht eindeutig. Eindeutig wird dieses Attribut erst durch die Angabe der `Messreihe_ID`. An dieser Stelle habe ich mir zu Nutze gemacht, dass das Attribut `TEinfahrt` in der Basisrelation Messung der Nutzdatenbank den Datentyp `NUMBER(8)` zugewiesen bekommen hat. Dies beschränkt zwar die maximale Dauer einer Messreihe (auf etwa 3,17 Jahre), ermöglichte es aber, einen eindeutigen Schlüssel in der Dimensionstabelle Zeit über die Formel

$$\text{CALC_KEY} = (\text{Messreihe_ID} * 100\,000\,000) + \text{TEinfahrt}$$

zu erzeugen. Natürlich kann es passieren, dass Fahrzeuge innerhalb einer Messreihe zum gleichen Zeitpunkt (innerhalb der Messgenauigkeit) eine Messstation erreichen.

Das Attribut `TEinfahrtTime` in der Zeitdimensionstabelle enthält einen Wert vom Datentyp `Timestamp`. Dieser Wert gibt die Zeit an, zu welcher eine Messung durchgeführt wurde – unabhängig von der Startuhrzeit. Berechnet wird dieser Wert in der Funktion `zeitberechnung()`. In dieser wird auf die Startuhrzeit, welche vom Typ `Timestamp` ist, der Wert von `TEinfahrt` aufaddiert. Startet eine Simulation beispielsweise um 15:50 Uhr und beträgt der Wert von `TEinfahrt` 602,56, so erhält das Attribut `TEinfahrtTime` den Wert 16:00:02,56 Uhr. Da dieser Wert von `TEinfahrtTime` nicht von der `Messreihe_ID` abhängt, ist er innerhalb dieser Relation auch nicht eindeutig.

Das Attribut Sekunde berechnet sich folgendermaßen:

$$\text{Sekunde} = (\text{Messreihe_ID} * 100\,000\,000) + \text{AnzahlSekundenDesTages}(\text{TEinfahrtTime})$$

Es handelt sich also um ein Attribut, welches alle Tupel, welche innerhalb einer Messreihe in der gleichen Simulationssekunde erzeugt wurden, zusammenfasst. Es handelt sich also um eine Zusammenfassung von Zeitpunkten. Eine Sekunde ist durch die Verrechnung mit der `Messreihe_ID` eindeutig.

Die Attribute `Minute`, `Stunde` und `Tag` sind analog definiert. Das Tupel aus vorigem Beispiel würde also die Werte 100 057 602 für `Sekunde`, 100 000 960 für `Minute`, 100 000 016 für `Stunde` und 100 000 000 für `Tag` zugewiesen bekommen. In diesem Beispiel hatte die `Messreihe` die Nummer 1. Zu beachten ist, dass diese Attribute rein numerische Werte haben und keine speziell in SQL definierten Datentypen wie `Timestamps` oder `Interval Day To Second` sind. Da der Umgang mit speziellen SQL Datentypen für Zeitwerte umständlicher wäre, vereinfacht die Darstellung als numerischer Wert die Arbeit mit dem Data Warehouse, obwohl eine andere Repräsentation (z.B. als `Interval Day To Second`) evtl. intuitiv zu erwarten wäre. Des Weiteren ließ die Forderung nach Eindeutigkeit der Werte spezielle Datentypen nicht zu.

Das zusätzliche Führen der berechneten Attribute in der Dimensionstabelle Zeit ist redundant. Sie ist jedoch so angelegt, dass vielfältige Anfragen an das Warehouse möglich sind. Zum Einen aggregierte Anfragen über die beschriebenen Attribute `Sekunde`, `Minute`, `Stunde`, `Tag`, zum Anderen natürlich auch Anfragen, welche mit dem Datentyp `Timestamp` gestellt werden. Die Relation Zeit ist also an sich komplett redundant, soll aber die Auswer-

tion der Daten im Data Warehouse erleichtern. Zum besseren Verständnis der Dimensionstabelle Zeit sei auf die Anfragebeispiele in Kapitel 5 hingewiesen.

4.3.3 Orthogonalität der Dimensionstabellen

Nach [BaGü01] sollten die Dimensionstabellen orthogonal zueinander sein. Diese Forderung ließ sich nur bedingt durchsetzen. Der Grund hierfür ist wieder die zentrale Rolle der Simulationsdatei. Da nur mit Hilfe der Simulationsdatei die Elemente der einzelnen Dimensionen eindeutig sind, nimmt die materialisierte Sicht der Simulationsdatei, im Data Warehouse Schema als SIMULATIONSDATEIDIM bezeichnet, die Rolle des obersten Hierarchieelements jeder Dimension ein. Dies widerspricht der Forderung, dass die Dimensionstabellen orthogonal zueinander sein sollten.

Wenn die Daten in den Dimensionstabellen reale Daten wären, fiel das gemeinsame Element Simulationsdatei weg. Damit wären die Dimensionen zueinander orthogonal. Eine Ausnahme wäre dann die Dimension Zeit, welche noch Verbindungen zur Dimension Messreihe hat. Dies hat - analog zu den anderen Dimensionstabellen - den Grund, dass die einzelnen Zeitwerte erst über die Angabe der Messreihe eindeutig identifiziert werden können. Wenn es sich also nicht um simulierte Daten handeln würde, würde auch die Verbindung zwischen der Dimension Zeit und der Dimension Messreihe entfallen.

Das Schaubild sähe dann wie folgt aus:

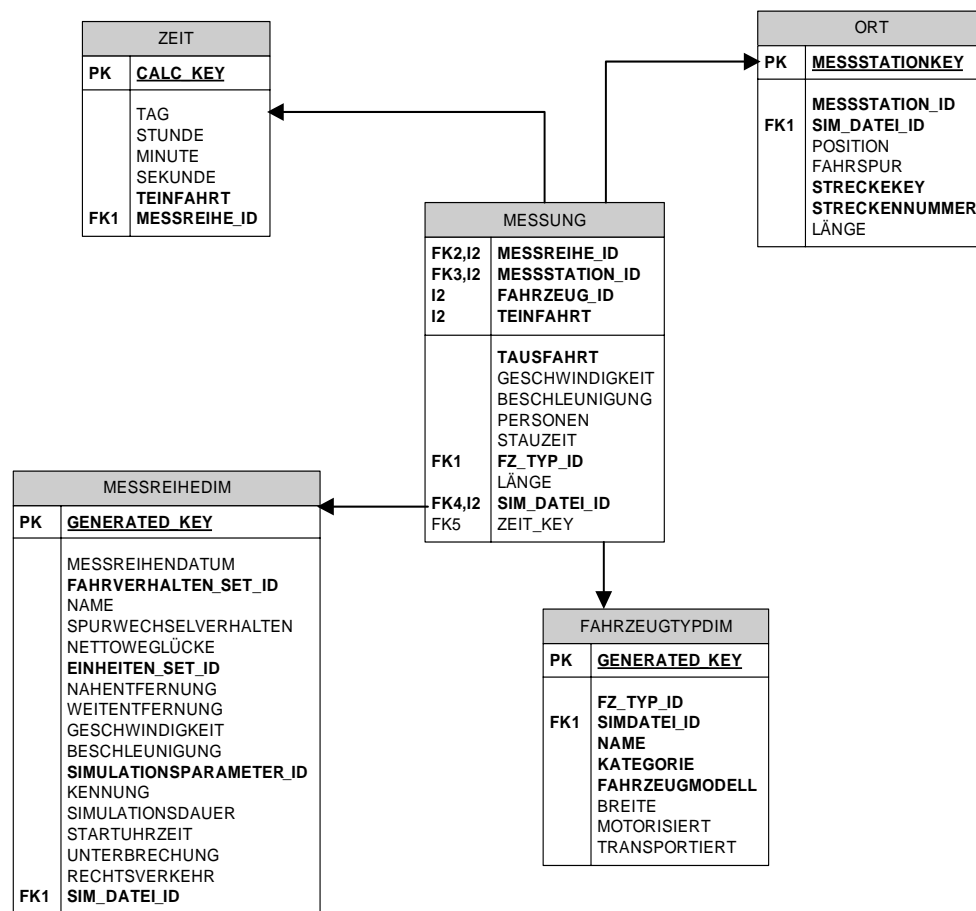


Abb. 6. Das Data-Warehouse für nicht simulierte Daten

4.3.4 Konsistenzbedingungen

Da die Daten im Data Warehouse exakt die Daten im Nutzdatschema widerspiegeln, sind alle Konsistenzbedingungen, welche im Nutzdatschema vorliegen, auch im Data Warehouse automatisch erfüllt. Dies gilt für Primärschlüssel als auch für Fremdschlüsselbedingungen. Die Angabe von Primärschlüsseln in den Dimensionstabellen ist also unnötig, auch in der Dimension Zeit, welche wie gesehen eindeutige Schlüssel enthalten muss. Auch die Angabe von Fremdschlüsselbedingungen ist obsolet.

Dennoch habe ich der Vollständigkeit halber Primärschlüssel und Fremdschlüssel definiert. Die Definition von Primärschlüsseln hat den angenehmen Effekt, dass auf die Schlüsselattribute Indizes angelegt werden. Die Fremdschlüsselbedingungen habe ich zur besseren Veranschaulichung im Schaubild eingefügt.

Die Definition dieser Konsistenzbedingungen ist auf den materialisierten Sichten selbst nicht machbar. Oracle erzeugt jedoch für jede materialisierte Sicht eine eigene Relation, welche als Schemaelement angelegt wird. Auf diesen Relationen wurden die Konsistenzbedingungen definiert. Beispielhaft möchte ich hier wieder zwei SQL Kommandos angeben, welche die Primärschlüsselforderung und die Fremdschlüsselbedingung im Schema durchsetzen.

```
ALTER TABLE MESSREIHEDIM ADD CONSTRAINT FAHRZEUGDIMPRIMARYKEY PRIMARY
KEY(GENERATED_KEY);
```

```
ALTER TABLE MESSREIHEDIM ADD CONSTRAINT
MESSREIHE2SIMULATIONSDATEIFOREIGNKEY FOREIGN KEY (SIM_DATEI_ID)
REFERENCES SIMULATIONSDATEIDIM(SIM_DATEI_ID);
```

Aufgrund der Angabe von Fremdschlüsselbedingungen ist die Reihenfolge, in welcher die materialisierten Sichten aktualisiert werden, nicht mehr beliebig. Die Stapelverarbeitungsdatei `dbmsmv_refresh.sql` führt die Updates auf den materialisierten Sichten in der richtigen Reihenfolge durch.

4.3.5 Dimensionen, Ebenen und Hierarchien

Der Begriff der Dimension ist nicht synonym mit dem Begriff der Dimensionstabelle zu betrachten. Trotz einer Verwechslungsgefahr möchte ich beide Begriffe verwenden, da der Begriff der Dimensionstabelle in einschlägiger Literatur (z.B. [BaGü01]) eher theoretisch verwendet wird, der Begriff Dimension im praktischen Einsatz in Oracle bzw. SQL auftaucht.

Dimensionen können über PL/SQL definiert werden, es bietet sich aber aufgrund der Komplexität an, zur Erstellung der Dimensionen Assistenten zu verwenden. Im Oracle Enterprise Manager ist dieser Assistent unter der Rubrik Warehouse zu finden.

Eine Dimension bezeichnet ein abstraktes Schemaelement, welches multidimensionale Anfragen über OLAP Werkzeuge ermöglicht. Eine Dimension besteht aus sog. Ebenen. Jeder Ebene wird ein Attribut einer Relation aus dem Schema zugewiesen. Die Dimension Zeit, welche im Rahmen dieser Studienarbeit definiert wurde, besteht beispielsweise aus den Ebenen `Zeit.CALC_KEY`, `Zeit.SEKUNDE`, `Zeit.MINUTE`; `Zeit.STUNDE`, `Zeit.TAG`, `Zeit.MESSREIHE_ID`, `Messreihedim.GENERATED_KEY` und `Messreihe.SIMULATIONSDATEI_ID`. Bei der Definition der Ebenen muss beachtet werden, dass die Primärschlüsselattribute der beteiligten Relationen vorhanden sind. Sind mehrere Rela-

tionen beteiligt, so müssen auch die Fremdschlüssel als Ebene definiert sein. Zusätzlich muss in dem Assistenten angegeben werden, welches Attribut als Fremdschlüssel auf ein anderes Primärschlüsselattribut verweist.

Mit Hilfe dieser Ebenen lassen sich in den Dimensionen verschiedene Hierarchien erstellen, welche angeben, in welcher Reihenfolge welche Attribute gruppiert werden. Beispielsweise wurden in der Dimension Fahrzeugtyp neben den Ebenen Generated_Key, Kategorie, Motorisiert, Transportiert und Simulationsdatei_ID die Hierarchien FahrzeugtypTransportiert und FahrzeugtypMotorisiert erstellt. Die Hierarchie FahrzeugtypTransportiert gibt die Abstufung Simulationsdatei_ID, Transportiert, Kategorie, GeneratedKey an, während die Hierarchie FahrzeugtypMotorisiert die Abstufung Simulationsdatei_ID, Motorisiert, Kategorie, Generated_Key angibt. Mit Hilfe eines OLAP Werkzeuges lassen sich so z.B. Anfragen stellen, welche alle Fahrzeuge betreffen, die Güter transportieren.

Neben den bereits erwähnten Dimensionen Zeit und Fahrzeugtyp wurde noch eine Dimension Ort mit den Ebenen Messstation_Key, Strecke_key definiert. Die Hierarchie dieser Dimension fasst alle Messstationen, welche auf einer Strecke liegen, und alle Strecken innerhalb einer Simulationsdatei zusammen.

Zwar dient der Assistent zum Erstellen der Dimensionen zum besseren Verständnis der Dimensionen selbst, er trägt aber sicher nicht dazu bei, dass das Data Warehouse sich einfach auf andere Datenbankschemata übertragen lässt oder nach einem Fehler leicht wiederherstellen lässt. Daher habe ich die SQL Kommandos, welche der Assistent erstellt, wieder in einer Stapelverarbeitungsdatei gespeichert. Diese heißt `createDimensions.sql`.

4.4 Das Data Warehouse im Warehouse-Builder

Der Data-Warehouse-Builder ist ein Werkzeug von Oracle, mit welchem sich ein Data-Warehouse-System erstellen lässt. Im Rahmen dieser Studienarbeit habe ich zusätzlich zur Definition der materialisierten Sichten untersucht, ob sich dieses Werkzeug für die Erstellung des gewünschten Data Warehouse einsetzen lässt. Hierbei zeigten sich einige Probleme.

Das erste Problem war die Mächtigkeit dieses Werkzeugs. Die Komplexität erfordert einen Einarbeitungsaufwand, der im Rahmen dieser Studienarbeit nicht möglich war. Zwar wurden testweise einige Schemaelemente angelegt, jedoch war nicht klar, wo diese Elemente angelegt werden und wie darauf wieder zugegriffen werden kann.

Ein weiteres Problem stellte die Inkompatibilität der verwendeten Datentypen dar. So unterstützt der Oracle-Warehouse-Builder den SQL-Datentyp `Timestamp` nicht.

Aus diesen Gründen habe ich eine zweite Implementierung eines Data Warehouse System mit dem Oracle Warehouse Builder nicht umgesetzt.

5. Anfragen an das Data Warehouse

Mit Anfragen an das Data Warehouse soll unter anderem das Schema getestet werden und evaluiert werden, inwiefern sich dieses Data Warehouse in der Praxis einsetzen lässt.

Im ersten Teil dieses Kapitels möchte ich anhand einiger beispielhafter SQL-Anfragen versuchen, die Semantik des Schemas und der Daten im Data Warehouse genauer zu beschreiben. Im zweiten Teil sollen multidimensionale Anfragebeispiele aus dem OLAP-Werkzeug von Oracle beschrieben werden, welche Aggregationen aufgrund der Dimensionen vornehmen.

5.1 einfache Anfragen

Da die Dimensionstabelle Zeit mir am wenigstens intuitiv erscheint, möchte ich mit einem Anfragebeispiel deren Aufbau noch einmal genauer beschreiben.

Folgende Anfrage soll ausgewertet werden: Wie lange steht ein Fahrzeug, welches an irgendeiner Messstation gemessen wird, während des Feierabendverkehrs zwischen 17:00 Uhr und 18:00 Uhr im Stau?

```
SELECT AVG(TStau)
FROM   MESSUNG m, Zeit t
WHERE  m.ZEIT_KEY = t.CALC_KEY
AND    ((t.STUNDE%100000000)%24) = 17
;
```

Besonders auffällig in dieser Anfrage sind die Modulooperationen. Daher eine kurze Erklärung dieser Operationen: Da die Anfrage unabhängig von der Messreihe (und auch unabhängig von der Simulationsdatei) sein soll, werden alle Messungen gesucht, welche zu einem Zeitpunkt zwischen 17:00 Uhr und 18:00 Uhr aufgezeichnet wurden. Die Operation $(t.STUNDE \% 100\ 000\ 000)$ löst den Zusammenhang zwischen der um 17 Uhr beginnenden Stunde und der Messreihe auf. Dies liegt daran, dass der Wert, welcher im Attribut STUNDE gespeichert ist, berechnet wurde, nämlich genau $(MESSREIHE_ID * 100000000) +$ Anzahl Stunden seit Mitternacht. Die anschließende Modulooperation (Modulo 24) trägt der Tatsache Rechnung, dass Messreihen länger 24 Stunden (in der Simulationszeit) dauern können. Außerdem kann es vorkommen, dass eine Messung um 23 Uhr beginnt. In diesem Fall wäre die Simulation noch keine 24 Stunden gelaufen, wenn die Fahrzeuge zwischen 17:00 Uhr und 18:00 Uhr gemessen werden. Trotzdem würde die Operation $(t.STUNDE \% 100\ 000\ 000)$ nicht das gewünschte Ergebnis 17 ergeben, sondern das Ergebnis 41, also $17 + 24$.

Eine andere einfache Anfrage an das Data Warehouse könnte lauten: War das Fahrzeug, welches um 16:00:02,56 Uhr gemessen wurde, motorisiert?

```
SELECT f.MOTORISIERT
FROM   MESSUNG m, FAHRZEUGTYP f, ZEIT t
WHERE  m.ZEIT_KEY = t.CALC_KEY
AND    m.FZ_TYP_ID = f.GENERATED_KEY
AND    TEinfahrtTime =
TO_TIMESTAMP('01.01.0001 16:00:02,56', 'DD.MM.YYYY HH24:MI:SSSS');
```

Die in diesem Beispiel verwendete Methode `TO_TIMESTAMP()` erzeugt ein Objekt vom Typ `Timestamp`, welches mit dem Attribut `TEinfahrtTime` verglichen wird. Dabei gibt das zweite Argument dieses Methodenaufrufs an, in welchem Format die Daten für den `Timestamp` vorliegen.

5.2 OLAP Anfragen

Die multidimensionalen Anfragen an das Data Warehouse wurden mit Produkten der Firma Cognos (siehe [Cog]) umgesetzt. Cognos stellt mehrere „Business Intelligence“-Produkte zur Verfügung, darunter Cognos Impromptu, Power Play Transformer und Power Play.

Die Cognos Produkte greifen über die ODBC Schnittstelle von Microsoft Windows auf das Data Warehouse zu. Die Datenbank, auf welcher das Data Warehouse gestartet wurde, muss also in der Systemkonfiguration als ODBC Datenquelle bekannt gemacht werden. Zum Erstellen und Benutzen der ODBC Schnittstelle reichen normale Systemprivilegien nicht aus. Erst mit Administratorrechten ausgestattet gelingt der Versuch, auf die ODBC Schnittstelle zuzugreifen.

5.2.1 Cognos Impromptu

Das Werkzeug Impromptu bildet die Basis für die Arbeit mit den weiterführenden Cognos Produkten. Es dient dazu, diejenigen Daten zu identifizieren, mit denen später gearbeitet werden soll. In einem ersten Schritt wird dazu ein sog. Katalog angelegt, in welchem die einzelnen Relationen angegeben werden, auf die später zugegriffen werden soll. In einem zweiten Schritt wird ein sog. Bericht angelegt, mit dessen Hilfe diejenigen Attribute aus den Relationen im Katalog bekannt gemacht werden, welche weiter verwendet werden sollen. Es werden verschiedene Arten von Berichten angeboten. Die hier dargestellten Ergebnisse wurden mit einer „einfachen Liste“ erzielt.

Geschwindigkeit	Personen	Stauzeit	Motorisiert	Transportiert	Kategorie
17,30	1	0,00	Motorisiert	Person	Pkw
15,00	55	0,00	Motorisiert	Person	Bus
7,70	2	0,00	Motorisiert	Person	Pkw
6,10	1	0,00	Motorisiert	Person	Pkw
4,90	2	0,00	Motorisiert	Person	Pkw
2,20	1	33,00	Motorisiert	Güter	Lkw
7,00	2	27,60	Motorisiert	Person	Pkw
8,00	55	21,00	Motorisiert	Person	Bus
9,50	1	15,10	Motorisiert	Person	Pkw
11,00	1	7,30	Motorisiert	Person	Pkw
12,80	1	0,00	Motorisiert	Person	Pkw
12,80	1	0,00	Motorisiert	Güter	Lkw
16,40	1	0,00	Motorisiert	Person	Pkw
17,30	1	0,00	Motorisiert	Person	Pkw
7,80	1	0,00	Motorisiert	Güter	Lkw
4,00	1	44,00	Motorisiert	Güter	Lkw
8,50	2	45,70	Motorisiert	Person	Pkw
8,10	1	37,60	Motorisiert	Person	Pkw
9,00	1	29,00	Motorisiert	Güter	Lkw
9,30	1	11,20	Motorisiert	Güter	Lkw
12,00	1	0,00	Motorisiert	Güter	Lkw
14,90	2	8,80	Motorisiert	Person	Pkw
16,30	2	0,00	Motorisiert	Person	Pkw
17,00	1	0,00	Motorisiert	Person	Pkw
7,80	1	0,00	Motorisiert	Person	Pkw
4,00	1	0,00	Motorisiert	Person	Pkw
2,90	1	42,50	Motorisiert	Person	Pkw
7,60	35	37,80	Motorisiert	Person	Bus
10,20	1	31,30	Motorisiert	Person	Pkw

Abb. 7. Eine einfache Liste dargestellt in Cognos Impromptu

Aus Impromptu heraus lässt sich ein Datenwürfel (vgl. [BaGü01]) erstellen. Dazu werden die Werkzeuge Power Play Transformer und Power Play gestartet. Power Play Transformer berechnet einen Würfel mit bestimmten Standardeinstellungen und zeigt den Würfel mit Power Play an.

5.2.2 Cognos Power Play Transformer

Power Play Transformer erkennt die im Data Warehouse definierten Hierarchien und stellt sie in einer Dimensionsübersicht grafisch dar. Der Begriff der Dimension wird in der Literatur bzw. von Oracle und von den Cognos Produkten homonym verwendet. Im Folgenden wird mit Dimension das gemeint, was vorher als Hierarchie bezeichnet wurde.

Diejenigen Attribute, die nicht zu Dimensionen gehören, stellt Power Play Transformer als Kennzahlen dar. Aus der Kategorieübersicht wird ersichtlich, wie der Datenwürfel aussehen wird. In der Kategorieübersicht werden Daten aus der Datenbank angezeigt. Dadurch lässt sich leicht kontrollieren, ob die Dimensionen richtig erkannt wurden.

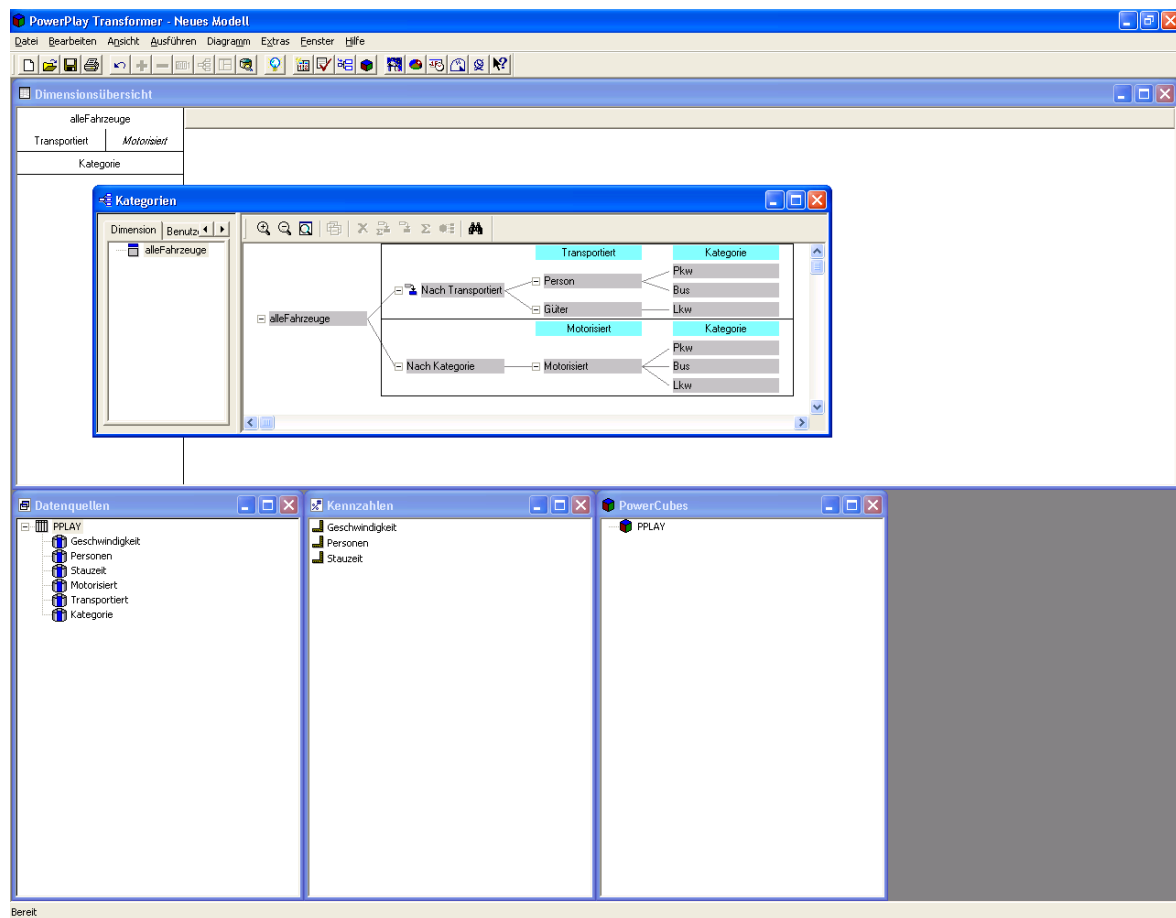


Abb. 8. Die Dimension FahrzeugtypDim visualisiert im Power Play Transformer

Im Power Play Transformer können die Dimensionen angepasst und die Verarbeitung der Kennzahlen festgelegt werden. Kennzahlen können durch Aufsummierung, Maximumbestimmung, Mittelwertbestimmung u.a. verarbeitet werden. Im Beispiel wurde von den Kennzahlen Geschwindigkeit, Personen und Stauzeit der Mittelwert gebildet.

Mit der neuen Konfiguration des Datenwürfels muss dieser neu erstellt und in Power Play geöffnet werden.

5.2.3 Cognos Power Play

Power Play ermöglicht es, in dem erzeugten Datenwürfel zu navigieren. Der Datenwürfel wird entweder in Tabellenform oder grafisch dargestellt. Es ist möglich, die Anzeige auf bestimmte Kennzahlen zu beschränken oder innerhalb der Dimensionen mit den Operatoren Verdichten bzw. Detaillieren auf verschiedene Granularitätsebenen der Daten zu wechseln. Die Operatoren Verdichten und Detaillieren entsprechen den in [BaGü01] genannten Operatoren RollUp und Drill Down.

Ein Beispiel zur Verdeutlichung:

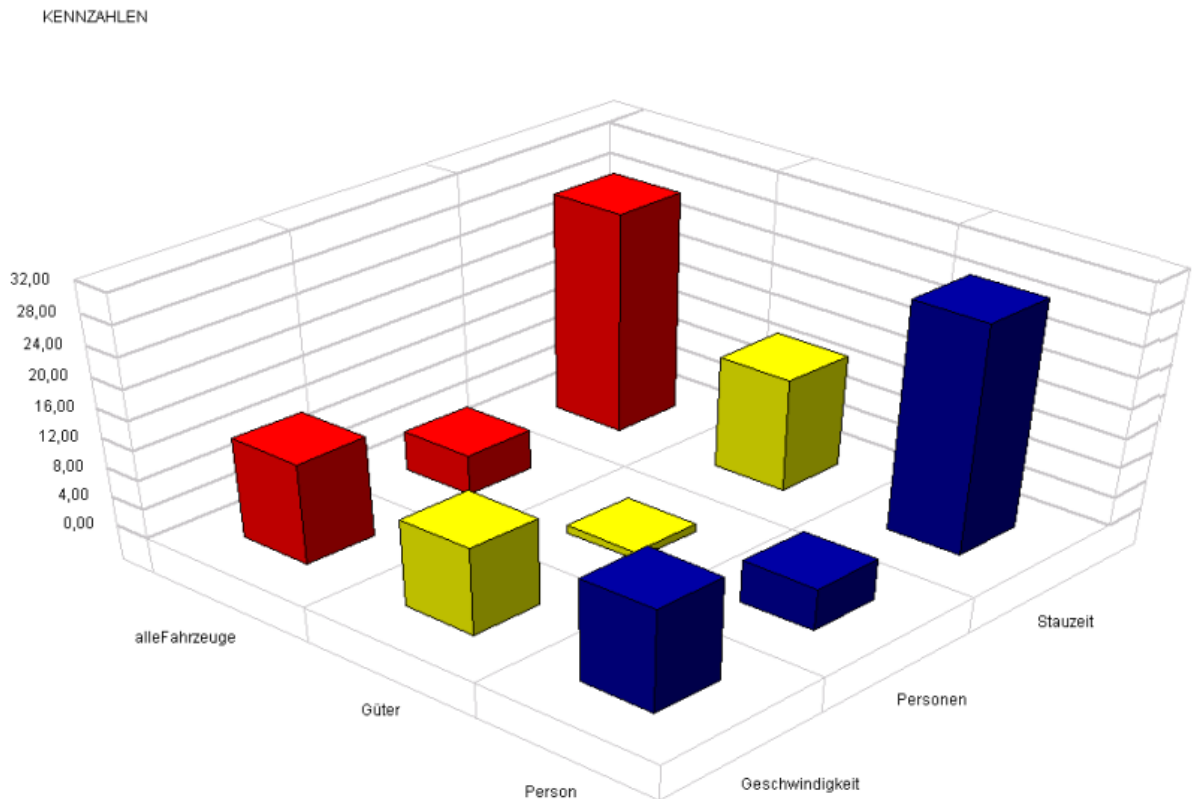


Abb. 9. Anzeige aller Kennzahlen für die Hierarchie „Transportiert“

Diese Grafik veranschaulicht den erstellten Datenwürfel. Man erkennt die drei Kennzahlen Geschwindigkeit, Personen und Stauzeit auf der einen und die Dimension Transportiert auf der anderen Seite.

Die nachstehende Grafik zeigt das Schaubild aus Power Play nach dem Anwenden des Detaillierungsoperator angewandt auf das Attribut Person aus der Dimension „transportiert“. Das Schaubild zeigt also alle Fahrzeugtypen, die Personen transportieren.

Die Werte Bus und Pkw, die im Schaubild dargestellt werden, stellen dabei die beiden Ausprägungen des Attributs Kategorie dar, welche tatsächlich dem Personentransport dienen. Diese Werte liest Power Play aus der Datenbank aus. Da im diesem Beispiel die Messstationen in VISSIM nur auf Straßen definiert waren, sind keine Fußgänger oder Zweiräder aufgelistet.

KENNZAHLEN

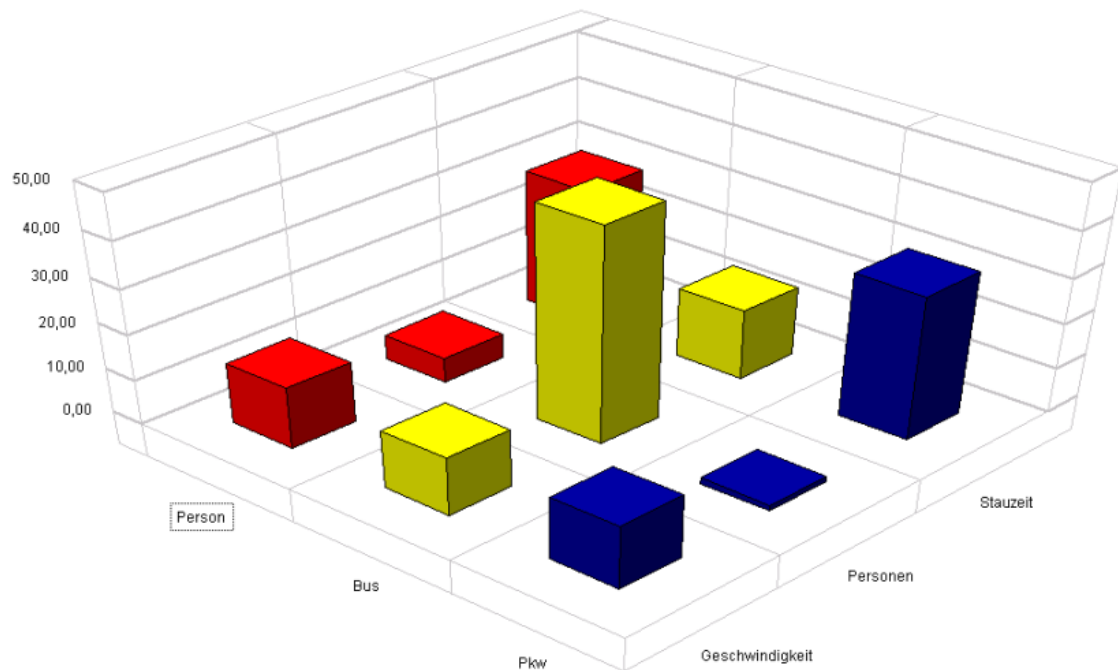


Abb. 10. Detaillierung der Dimension „transportiert“ auf die Ebene „Person“

5.2.4 Bemerkung

Probleme gibt es bei Attributen aus den Dimensionstabellen, welche einen numerischen Wert enthalten. Diese werden im Power Play Transformer als Kennzahlen dargestellt. Abhilfe schafft hier das manuelle Löschen dieser Attribute aus der Liste der Kennzahlen. Anschließend muss eine neue Dimension erzeugt werden, in welche diese Kennzahlen als Ebenen eingefügt werden können.

6. Zusammenfassung und Ausblick

In diesem Kapitel möchte ich das bisher erreichte bewerten und Anregungen für weitere Arbeiten auf diesem Gebiet geben.

6.1 Zusammenfassung

Ich habe aufgezeigt, dass insbesondere simulierte Daten problembehaftet sind, da aufgrund der Simulation Eindeutigkeiten und damit auch Fremdschlüsselbedingungen nicht mehr intuitiv gegeben sind. Für dieses Problem habe ich einen Lösungsweg vorgeschlagen.

Ich habe ein relationales Datenbankschema erstellt, welches trotz der Problematik der Eindeutigkeit jedes Tupel eindeutig identifizierbar macht. Die weitreichenden Folgen dieser Problematik machten die Konzeption dieses Schemas als aufwändiger als gedacht heraus.

Das Data Warehouse mit Hilfe von materialisierten Sichten zu erstellen war eine Entscheidung, welche die Konzeption des Data Warehouse recht einfach macht und es flexibel gegenüber Veränderungen der zugrunde liegenden Daten macht. Dies ist aber nicht die mächtigste Lösung.

Da diese Arbeit hauptsächlich auf dem Datenformat von VISSIM 3.70 beruht, habe ich meine Arbeiten so strukturiert, dass eine Anpassung an ein verändertes Datenformat möglich bleibt. Erleichtert wird eine Anpassung dadurch, dass sie Stück für Stück vorgenommen werden kann, da die einzelnen Elemente meiner Arbeit (Erstellen eines Nutzdatenschemas, Füllen des Nutzdatenschemas mit JDBC, Konzeption des Data Warehouse durch materialisierte Sichten und OLAP-Anfragen über die ODBC-Schnittstelle) strikt voneinander getrennt wurden.

Die Möglichkeit der Verteilung der einzelnen Elemente auf verschiedene Rechner war ein wichtiges Kriterium, um ein performantes Gesamtsystem zu erhalten. So ist es möglich, VISSIM, vissimLoader, ein DBMS für die Nutzdaten, ein DBMS für das Data Warehouse und die OLAP Software auf jeweils getrennten Rechnern auszuführen. Dies hat den Vorteil, dass die Rechenlast auf mehrere Rechner verteilt werden kann und kein Rechner mit einer zu großen Last die Performanz des gesamten Systems zu sehr einschränkt.

6.2 Ausblick

Besonders die Funktionsweise des Data Warehouse Builder zum Erstellen eines Data Warehouse mit einem DBMS von Oracle bedarf einer weitergehenden Evaluierung. Aufgrund der vielfachen Probleme, welche sich bereits am Anfang der Arbeit mit diesem Werkzeug eröffnet haben, war eine tiefgehendere Evaluierung im Rahmen dieser Studienarbeit nicht möglich. Dennoch verspricht dieses Werkzeug, Data-Warehouse-Systeme von größerer Komplexität als es im Rahmen dieser Studienarbeit nötig war beherrschbar zu machen.

Ich habe eine Möglichkeit vorgeschlagen, der Problematik der nicht eindeutig identifizierbaren Tupel im Nutzdatenschema zu begegnen. Da dieses Problem in ähnlicher Weise bei vielen simulierten Daten auftreten kann, wäre eine Untersuchung wichtig, wie meine Erfahrungen generalisiert werden können oder welche anderen Lösungsmöglichkeiten es für dieses Problem gibt.

7. Abbildungsverzeichnis

- Abb. 1.** Beschreibung der Messstationen, Seite 11
- Abb. 2.** Beschreibung der Messdaten, Seite 11
- Abb. 3.** grafische Veranschaulichung des Nutzdatenschemas, Seite 16
- Abb. 4.** Syntaxdiagramm zur Erzeugung eines Database Link, Seite 25, nach [OraRef03]
- Abb. 5.** Das Schema des Data-Warehouse als materialisierte Sichten, Seite 28
- Abb. 6.** Das Data-Warehouse für nicht simulierte Daten, Seite 31
- Abb. 7.** Eine einfache Liste dargestellt in Cognos Impromptu, Seite 37
- Abb. 8.** Die Dimension FahrzeugtypDim visualisiert im Power Play Transformer, Seite 38
- Abb. 9.** Anzeige aller Kennzahlen für die Hierarchie „Transportiert“, Seite 39
- Abb. 10.** Detaillierung der Dimension „transportiert“ auf die Ebene „Person“, Seite 40

8. Literaturverzeichnis

- [BaGü01] Andreas Bauer, Holger Günzel: Data Warehouse Systeme, dpunkt Verlag 2001
- [Cog] Homepage der Cognos GmbH, <http://www.cognos.com>, Stand: 18.3.2004
- [Hob01] Dr. Lilian Hobbs, Oracle 9i Materialized Views – An Oracle White Paper, 2001, http://otn.oracle.com/products/oracle9i/pdf/o9i_mv.pdf, Stand:18.3.2004
- [OraRef03] Oracle 8i SQL Reference, <http://www.cs.bris.ac.uk/Database/OracleDocs/server.816/a76989/ch4f5.htm>, Stand:18.3.2004
- [OVID] Homepage des Projekts Ovid, <http://ovid.uni-karlsruhe.de>, Stand:18.3.2004
- [PTV] Homepage der PTV AG, <http://www.ptv.de>, Stand:18.3.2004
- [Saa97] Gunter Saake, Andreas Heuer: Datenbanken: Implementierungstechniken, MITP Verlag, 1. Auflage 1999, ISBN 3-8266-0513-6
- [Sie03] Siemens Telematics: Verkehrstelematik – Das Zusammenspiel entscheidet, http://www.is.siemens.de/traffic/de/de_its/sol_ser/vkt/pdf/telematics_de.pdf, Stand:18.3.2004