

Seminar Imperfektion und Datenbanken
Wintersemester 2003/2004

Der probabilistische Ansatz

Ying Zou

Betreut von Frau Jutta Mülle

Universität Karlsruhe
Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation
Lehrstuhl für Systeme der Informationsverwaltung

Karlsruhe, 15.01.2004

Zusammenfassung

Eine Datenbank ist eine Sammlung von Daten aus einem bestimmten Bereich der realen Welt. Aber es gibt auch viele unbestimmten Bereichen in der Welt, Z.B.: Unvollständigkeit, Ungenauigkeit, Vagheit, usw. Um solche Unbestimmtheiten zu beschreiben, führen wir einen neuen Begriff „Imperfekte Daten“ ein. Wie wird eine Datenbank mit imperfekten Daten funktionieren? Dazu werden viele Ansätze angeboten. Ein ist der probabilistische Ansatz. Die Wahrscheinlichkeitstheorie spielt als eine wichtigste Rolle in diesem Ansatz.

In dieser Ausarbeitung wird ein Ansatz — Erweiterung von Datenbanksprache—SQL für Unbestimmtheiten vorgestellt. Zuerst werden die grundsätzlichen Erklärungen über imperfekte Daten und Wahrscheinlichkeitstheorie kurz erläutert; Weiterhin wird, anhand von Beispielen und durch die Beschreibung des zugrundeliegenden Datenmodells, die Erweiterung von SQL aus zwei Seiten -- Syntax und Semantik betrachtet, und bei jeder Seite wird es gezeigt, dass die Glaubwürdigkeit und Plausibilität für eine Datenbank verfügbar sind.

Schlüsselwörter : Erweiterung von SQL; Imperfektion; Imperfekte Daten; Glaubwürdigkeit; Plausibilität; Wahrscheinlichkeitstheorie

1 Einleitung

1.1 Problemstellung

Eine Datenbank ist eine Sammlung von Daten der realen Welt. Ziel einer Datenbank ist es die reale Welt möglichst getreu abzubilden. Es werden alle relevanten Daten aus dem Bereich erfasst und in die Datenbank eingebracht. Aber wie kann man entscheiden, welche Daten die „relevanten Daten“ sind? Zum Beispiel: „zwischen 14:00 und 15:00“: Es ist gültig, wenn Datentyp Zeitraum ist. Aber wenn man nur einen genauen Zeitpunkt bekommen möchte, dann tritt für solche Zuweisung **Imperfektion** auf.

Imperfektion kann im Datenbankschema oder in der Datenbasis, innerhalb eines Datums oder zwischen mehreren Daten, auftreten.

Für allen Datenbanksysteme und Anfragesprachen ist es möglich, um imperfekte Informationen in irgendeiner Form abzulegen und zu verarbeiten. Aber nur unter der Voraussetzung ist es möglich, dass die unbestimmten Daten in einer bestimmten Datum quantifiziert werden können. Zur Quantifizierung von unbestimmten Daten ist die Verwendung der Wahrscheinlichkeiten eine mögliche Lösung.

1.2 Beispiel Datenbank

Folgende ist eine Beispieldatenbank, und diese Datenbank wird in der ganzen Ausarbeitung verwendet (Abb. 1.1):

<i>Sent_by_Boeing(Lot_Num,Part,Wann)</i>		
Lot_Num	Part	Wann
23	Flügel Strebe	6.Mai
24	Maschine	4.Juni

<i>Sent_by_Cessna(Lot_Num,Part,Wann)</i>		
Lot_Num	Part	Wann
30	Flügel Strebe	6.Mai
31	Flügel Strebe	9.Juni

<i>In_production(Model, Serial_Num, Während)</i>		
Model	Serial_Num	Während
Centurion	AB33	[1.März ~ 31.März – 1.Juni ~ 30.Juni]
Cutlass	Z19	[1.Juni ~ 30.Juni – 1.Juli ~ 31.Juli]
Centurion	AB34	[1.Juni ~ 30.Juni – 1.August ~ 31.August]
Caravan	FA2K	[1.April ~ 30.April – 1.Mai ~ 31.Mai]

<i>Received(Lager, Lot_Num, Part, Wann)</i>				
Lager	Lot_Num	Part	Wann	
Boeing	23	Flügel Strebe	10.Mai ~ 29.Mai	e1
Cessna	30	Flügel Strebe	30.Mai ~ 18.Juni	e2
Boeing	24	Maschine	8.Juni ~ 27.Juni	e3
Cessna	31	Flügel Strebe	13.Juni ~ 2.Juli	e4

Abb. 1.1 Eine Beispiel Datenbank

Ein Betrieb besteht aus einer Flugzeugfabrik und zwei Lagern: Boeing und Cessna. Die Lager versorgen die Fabrik mit Maschinenteilen. Jedes Lager erhält eine Relation--*Send*, die zeigt wann die Maschinenteile von Lager nach Fabrik abgeliefert werden; Die Fabrik erhält die Relation: *In_production*, die das Abbild der Flugzeugungen bei der Fabrik ist.

Wir nehmen an, dass die Basis Dateneinheit für Datum in den beiden Relationen ein Tag ist .

Daher tauchen die Unbestimmtheiten in den beide Relationen auf: Die Eigenschaft— „*Während*“ in der Relation *In_Production* ist ein unbestimmter Zeitraum, das heißt, wir rechnen die Erzeugungsdauer per Einheit--Monat. Ein Monat ist ein unbestimmter Wert und bildet als eine Reihe der möglichen Tage ab. Eine Erzeugung des Flugzeuges beginnt bestimmt in diesem Monat, aber an welchem Tag sind wir nicht sicher. Zum Beispiel: Die Erzeugung an der Centurion mit Serial Nummer AB33 startet irgendwann zwischen 1, März und 31, März. So nehmen wir in diesem Beispiel an, dass die Erzeugungen am irgendeine Tag im einem sicheren Monat startet oder beendet.

Außerdem gibt es in Datenbank noch eine Relation: *Received*. Die gehört zu weder dem Lager noch der Flugfabrik, sondern ist sie eine geführte Relation basiert auf den Erzeugungen. Maschinenteile werden abgeliefert per LKW von einem Lager nach der Fabrik und nicht früher als 4 und auch nicht später als 24 Tage nach dem Abfahren an der Fabrik erreicht. Also, die Eigenschaften „*Wann*“ in der *Received* werden so berechnet: die Werte der Relation *Sent* von jeden Lager plus 4-24 Tage. Dann gibt es in der Relation *Received* unbestimmten Zeitpunkt: wir wissen dass Boeing 23, Flügel Strebe zwischen 10, Mai und 29, Mai an die Fabrik erreicht sind, aber wir wissen nicht den genauen Tag.

In der Beispieldatenbank wird eine Anfrage nicht normal verfügbar. Beispielweise möchten wir die Lot_num von Maschinenteile wissen, welche Flügel Strebe vor 30. Mai an Fabrik gekommen sind. Die Anfrage repräsentiert in SQL ist:

```
Select Lot_Num from Received
Where Part = „ Flügel Strebe “
And Wann <= 30.Mai
```

Abb. 1.2 Beispiel Anweisung

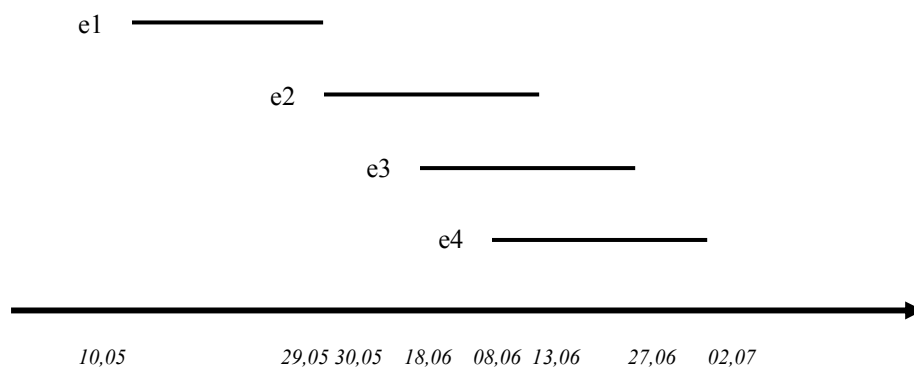


Abb. 1.3. Zeit Darstellung der „*Wann*“ in der Relation: *Received*

In der Relation—*Received* der Datenbank ist e1(10. Mai ~ 29. Mai) jedenfalls eine Antwort; Aber e2(30. Mai ~ 18. Juni) ist problematisch(Abb.1.3). Die Flügel Strebe könnten am 29, Mai angekommen werden, oder am 08, Juni; Jeder Tag hat gleich Gewicht. Es existieren inzwischen viele Möglichkeiten der Antworten für die Anfrage, davon ist

wegen der weiten Verbreitung natürlich eine Erweiterung von Anfragesprachen. In den folgenden Abschnitten(2,3) wird die Erweiterung von SQL vorgestellt.

2 Grundbegriffe

2.1 Wahrscheinlichkeitstheorien

Wahrscheinlichkeitstheorie ist das beste verstandene mathematisches Paradigma zur Modellierung und Behandlung von unbestimmten Informationen. Durch die Wahrscheinlichkeiten der Ereignisse werden die unbestimmten Daten von maximaler Granularität in einer bestimmten Daten zugeordnet.

Die Voraussetzung der Anwendung der Theorie in Datenbank ist: Die Wahrscheinlichkeiten der komplexen Fälle können von denen der grundlegenden Fälle berechnet werden.

Die Wahrscheinlichkeiten der Ergebnisse werden interpretiert in der Form:
 $P(A)$ und $P(A|B)$

2.2 Typen von Antworten

Es gibt zwei definierte Beschränkungen zu den Antworten der Anfrage in imperfekten Daten: **Bestimmte Antwort** und **Mögliche Antwort**.

Die bestimmte Antwort ist die Information, die für die Anfrage zu allen möglichen Erweiterungen der Datenbank gültig ist. Die Interpretation der Antwort in Wahrscheinlichkeiten ist: $P(A) = 1$.

Die mögliche Antwort ist solche Information, die für die Anfrage nur zum Teil der möglichen Datenbankerweiterungen gültig ist; Die interpretiert in Wahrscheinlichkeiten ist: $P(A) > 0$;

Wir überlegen wieder die Beispielanfrage in Abbild 1.2: Obwohl das genaue Datum der Ankunft der Flügel Strebe unbewusst ist, ist es klar, dass das Anliefern bevor 30. Mai fertig ist. Für die Anfrage ist e_1 eine bestimmte Antwort. Die Wahrscheinlichkeit von e_2 ist 0.10 ($P(e_2) = 2\text{Tage}/20\text{Tage} = 10\%$), dann ist e_2 eine mögliche Antwort.

Zwischen der möglichen Antwort und der bestimmten Antwort existiert noch einen anderen Antworttyp nämlich, Wahrscheinliche Antworten, die sind die Antworten mit Wahrscheinlichkeiten nicht kleiner als 0,5: $P(A) \geq 0,5$. Zum Beispiel sind e_1 und e_2 die wahrscheinlichen Antworten ($P(e_2) = 55\%$).

2.3 Glaubwürdigkeit und Plausibilität

Es gibt zwei Stufen um eine Antwort der Anfrage zu bestimmen: Die erste gewinnt die Daten wieder, die relevante zu der Anfrage sind. Die Zweite konstruiert eine Antwort, welche die Bedingungen von der Anfrage erfüllt. Dazu werden zwei getrennten Kontrollen auf die Unbestimmtheiten zu jeder Stufe angefordert.

Entsprechende Glaubwürdigkeit verändert die Informationen verfügbar zu einer Anfrage mit der Ersetzung jeden Unbestimmter Wert durch relevanter bestimmten Wert. Eine typische Ersetzung ist Erwartungswert oder Wahrscheinlichkeit. Zum Beispiel (in Abb.1.1), die erwartete Startdatum der Produktion für das Centurion mit Nummer AB33 ist 15, März. In SQL mit Unbestimmtheit kann ein Benutzer eine entsprechende Glaubwürdigkeit auswählen, und die Auswahl modifiziert alle unbestimmten Daten in der verbundenen Relation und entfernt die Unbestimmtheit.

Reihenfolge der Plausibilität kontrolliert die Konstruktion einer Antwort zu der Anfrage. Zum Beispiel, der Besitzer der Centurion kann anfragen, welcher Versand der Flügel Strebe vielleicht mittlerweile angekommen wird während der Produktion seines Flugzeugs. Für solche Anfrage betrachten wir zu dem Überlappen zwischen der Erzeugung und dem Ankunftstag der Versand drei Fälle:

1. Fall: der Ankunftstag tritt sicher während der Erzeugung auf;
2. Fall: der Ankunftstag tritt wahrscheinlich während der Erzeugung auf;
3. Fall: es kann sein, dass der Ankunftstag während der Erzeugung auftritt.

Es ist klar, dass die Wahrscheinlichkeiten des Überlappens von 1.Fall bis 3.Fall sinkend sind. Solchen Wahrscheinlichkeiten werden als die Reihenfolge der Plausibilität betrachtet, und der Benutzer wählt eine erwünschte Plausibilität von der Reihenfolge für seine Anfrage aus. In unserer Beispieldatenbank ist es möglich, dass die Flügel Strebe mit Lot_Num 31 von der Large Cessna während der Erzeugung des Centurion AB33 angekommen werden, aber niemand kann das absolut gewährleisten.

Die entsprechende Glaubwürdigkeit ersetzt die Unbestimmtheit in dem Datum, und die Reihenfolge der Plausibilität gibt die mögliche Beziehung zwischen dem Datum.

3. Erweiterungen des Datenmodells

3.1 Beschreibungen zu unbestimmten Daten

Jedes Datum wird durch *untere Beschränkung* (α_*), *obere Beschränkung* (α^*) und *Probabilistische Mass Funktion* beschrieben [Dyr&Sno93].

Die beiden Beschränkungen begrenzen die Bereiche der unbestimmten Daten. Das unbestimmte Datum ist nicht kleiner als untere Beschränkung und nicht größer als obere Beschränkung. Beispielsweise, für unbestimmte Daten *Während* [1.März ~ 31.März – 1.Juni ~ 30.Juni] in der Relation „*In_production*“ ist der Zeitraum [1.März – 1.Juni] die untere Beschränkung, und der Zeitraum [31. März – 30. Juni] die obere Beschränkung für *Während*. Gleichfalls ist in der Relation *Received* für die unbestimmte Eigenschaft *Wann* [10.Mai ~ 29.Mai] der Zeitpunkt 10.Mai eine untere Beschränkung und 29. Mai eine obere Beschränkung.

Obwohl ein unbestimmtes Datum kann an einigen möglichen Einheiten aufgeteilt werden kann, haben alle die möglichen Einheiten mit verschiedenen Gewichten. Die probabilistische Mass Funktion gibt die Wahrscheinlichkeit für jede mögliche Einheit. Die probabilistische Mass Funktion P_α für das unbestimmte Datum α ist:

$$P_{\alpha(i)} = \Pr[\alpha = i] \quad i \in \{0, 1, \dots, N\}$$

Wobei $\Pr[\alpha = i]$ die Wahrscheinlichkeit ist, dass das unbestimmte Datum α zum Einheit i angeordnet ist; Und jedes Datum α bleibt in dem Bereich der unbestimmten Daten, $\Pr[i < \alpha_*] = 0$ und $\Pr[i > \alpha^*] = 0$. Wir nehmen an, dass alle unbestimmten Daten unabhängig von anderen sind, d.h.:

$$\Pr[\alpha = i \wedge \beta = j] = \Pr[\alpha = i] \times \Pr[\beta = j]$$

3.2 Erweiterungen von SQL – Syntax

Um die Unbestimmtheiten unterstützen zu können, sind folgenden 4 Erweiterungen von SQL benötigt:

1. Gebe bei Erzeugen einer Relation an, dass die Eigenschaft der Daten unbestimmt ist; In den Anweisungen der Erzeugung einer Tabelle kann ein Benutzer die Einschränkung entweder **INDETERMINATE** oder **INDETERMINATE COMPACT** vor der Eigenschaft der Daten hinzufügen. Die beiden Einschränkungen werden anzeigen, dass die Daten vielleicht unbestimmt sind. Der Benutzer kann auch einen optionalen Satz am Ende der Beschreibung der Eigenschaften hinzufügen, um die Mass Funktion als **STANDARD** oder **NONSTANDARD** anzugeben. Untere Abb. 3.1 ist ein Beispiel der Erzeugung einer Tabelle mit der Erweiterung von SQL:

```
CREATE TABLE Received ( Lager          CHAR(30),
Lot_Num    INTERGER,
Part       CHAR(40),
Wann       INDETERMINATE DATUM);
CREATE TABLE In_Production ( Model    CHAR(30),
Serial_Num CHAR(10),
Während    INDETERMINATE ZEITRAUM(DATUM));
ALTER TABLE Received ALTER COLUMN Wann TO NONSTANDARD DISTRIBUTION;
```

Abb. 3.1. Erweiterung von SQL - Syntax

2. Bestimme die Stufe der Glaubwürdigkeit in der „*select...from...*“ Klausel. Wir wissen, dass die *from* Klausel in der *select* Anweisung die Relationen erklärt, mit denen wird die Anfrage berechnet und entsprechender Name(n) zu jeder Relation geordnet wird. Die Glaubwürdigkeiten werden durch Ausdruck **WITH CREDIBILITY** bezeichnet. Für jedes unbestimmte Datum wird Ersetzung (Replacement) Verfahren benutzt. Dazu gibt es 4 möglichen Stufen der Glaubwürdigkeit:

1. **INDETERMINAT** -- erhält alle Ungenauigkeiten; Keine unbestimmten Daten werden ersetzt;
2. **EXPECTED** -- Ersetzt die ungenauen Daten mit dem erwarteten Wert;
3. **MAX** -- Ersetzt die unbestimmten Daten mit unterer Beschränkung(α_*);
4. **MIN** -- Ersetzt die unbestimmten Daten mit oberer Beschränkung(α^*).

Die Angabe der Stufe der Glaubwürdigkeit ist optional und es gibt einen Defaultwert, **INDETERMINAT**. Der Defaultwert kann auch durch einen Ausdruck **SET DEFAULT CREDIBILITY**(Glaubwürdigkeit) ersetzt werden. Z.B., die folgenden zwei Anweisungen haben gleichen Bedeutungen, und die werden keinen Unbestimmtheiten durch bestimmte Daten ersetzt:

1. **SELECT Lager FROM *Received* WITH CREDIBILITY INDETERMINATE**
2. **SELECT Lager FROM *Received* SET DEFAULT CREDIBILITY**

3. Bestimme die Reihenfolge der Plausibilität. Der Wert der Plausibilität wird in der *Where* Klausel bezeichnet. Der Defaultwert der Plausibilität ist der Ausdruck: SET DEFAULT PLAUSIBILITY. Man kann auch die Reihenfolge der Plausibilität durch den Ausdruck WITH PLAUSIBILITY mit einem entsprechenden Interger zwischen 1 und 100 am Ende der *Where* Klausel zuweisen. Die Reihenfolge der Plausibilität 1 zeigt, dass die beliebige mögliche Antwort erwünscht ist, und 100 ist ein Defaultwert, das heißt, nur die bestimmte Antwort ist erlaubt. Wie in dem Beispiel in der Abbildung 3.2 ist die Reihenfolge der Plausibilität mit 60 zugewiesen, und die Antworten mit der Wahrscheinlichkeit höher als 60% sind gültig.

```
SET DEFAULT PLAUSIBILITY 60
SELECT Lager FROM Received
WHERE Model = „Centurion“
AND Wann < 29.Mai
WITH PLAUSIBILITY 60
```

Abb. 3.2 Beispiel zu 3. Erweiterung der Syntax

4. Füge die Zeichen in der Datenbank hinzu, um die Unbestimmtheit zu unterstützen; Die Beschränkungen der unbestimmten Daten können durch Zeichen „~“ getrennt werden, Z.B., DATE „10/5/2004 ~ 29/5/2004“ in der Datenbank repräsentiert der unbestimmte Zeitpunkt „von 10, Mai, 2004 bis 29, Mai, 2004“; Eine Mass Funktion kann auch als ein Zeichen genannt werden, Z.B., DATE „10/5/2004 ~ 29/5/2004 UNIFORM“ repräsentiert einen Zeitpunkt mit einer uniformen Mass Funktion.

3.3. Semantik für Unbestimmtheiten

In diesem Abschnitt werden die Erweiterungen von Semantik der SQL geklärt. Wir stehen die Anweisung -- *select* der SQL im Brennpunkt. Zuerst wiederholen wir die Semantik der Anweisung -- *Select* in SQL. Danach erweitern wir die Semantik um die Unbestimmtheiten in Datenbank zu unterstützen. Die Bewertungen einer *Select* - Anweisung in der erweiterten Semantik haben zwei Interpretationen, bestimmte Interpretation und mögliche Interpretation, wie anderen Interpretationen, die zwischen Bestimmtheiten und Unbestimmtheiten stehen. Da Folgenden werden bewiesen: Die mögliche Interpretation ist sowohl zuverlässig, in denen keine unrichtigen Antworten auftreten dürfen, als auch maximal, D. h. es erzeugt keine weiteren Ergebnisse. Um die Unbestimmtheiten effizient zu implementieren ist noch die operationale Semantik benötigt. Die operationale Semantik bietet alle nötigen Unterstützungen für Unbestimmtheiten mit drei Veränderungen der SQL Semantik: Erste, es definiert die Relation – *Before* für eine vorläufige Reihenfolge; Nächste, es leitet einen 4 Wertebereich für die Bewertung der *where* – Klausel ein; Und endlich fügt eine Replace Operator ein, um die entsprechende Glaubwürdigkeit auszuführen.

3.3.1 Wiederholung der SQL Semantik

Folgend wird eine einfache Semantik für die Anweisung – *Select* präsentiert, wobei die Notiz $\llbracket x \rrbracket_{SQL}$ die Syntax der SQL von Konstrukt x seien; Und d eine Datenbank sei.

$$\begin{aligned} & \llbracket \text{[SELECT } \langle \textit{target list} \rangle \text{ FROM } \langle \textit{from list} \rangle \text{ WHERE } \langle \textit{predicate} \rangle \rrbracket_{SQL} (d) \\ & = \llbracket \langle \textit{Target list} \rangle \rrbracket_{SQL} (\llbracket \text{[WHERE } \langle \textit{predicate} \rangle \rrbracket_{SQL} (\llbracket \langle \textit{from list} \rangle \rrbracket_{SQL} (d))) \end{aligned}$$

Die *Select* – Anweisung legt erst die *from* Klausel in der Datenbank auf. Die Klausel berechnet das Cartesian Produkt der spezifizierten Relationen in der $\langle from\ list \rangle$. Die Bedeutung der *from* Klausel ist:

$$\begin{aligned} \llbracket \langle from\ list \rangle \rrbracket_{SQL}(d) &= \llbracket \langle Relation1 \rangle, \dots, \langle RelationN \rangle \rrbracket_{SQL}(d) \\ &= \llbracket \langle Relation1 \rangle \rrbracket_{SQL}(d) \times \dots \times \llbracket \langle RelationN \rangle \rrbracket_{SQL}(d) \end{aligned}$$

wobei

$$\llbracket \langle Relation \rangle \rrbracket_{SQL}(d) = r_i, r_i \in d, \text{ und } r_i \text{ ist als } \langle Relation \rangle \text{ genannt.}$$

Das Ergebnis, der berechnet bei der *from* Klausel wurde ist, eine unbestimmte Relation. Es wird als ein Argument für die *where* Klausel benutzt. Die Klausel wählt die Tuples t aus, die die Predikate in der *where* Klausel erfüllen.

$$\llbracket \text{WHERE } \langle predicate \rangle \rrbracket_{SQL}(r) = \{ t \mid t \in r \wedge \llbracket \langle predicate \rangle \rrbracket_{SQL}(t) \}.$$

Endlich wird die Ausgabe von *where* Klausel als Eingangsparameter bei der $\langle Target\ list \rangle$ benutzt.

$$\llbracket \langle Target\ list \rangle \rrbracket_{SQL}(r) = \prod \llbracket \langle Target\ list \rangle \rrbracket_{SQL}(r)$$

3.3.2 Erweiterung der Semantik von SQL für Unbestimmtheiten

Die Semantik zu unbestimmter Datenbank für die *Select* -- Anweisung ist wie folgende beschrieben:

$$\begin{aligned} \llbracket \text{SELECT } \langle target\ list \rangle \text{ FROM } \langle from\ list \rangle \text{ WHERE } \langle predicate \rangle \rrbracket_{ind}(\delta, \gamma, d) \\ = \llbracket \langle target\ list \rangle \rrbracket_{ind}(\gamma, \llbracket \text{WHERE } \langle predicate \rangle \rrbracket_{ind}(\gamma, \llbracket \langle from\ list \rangle \rrbracket_{ind}(\delta, d))) \end{aligned}$$

wobei δ die entsprechende Glaubwürdigkeiten seien und γ die Reihenfolge der Plausibilität sei; Die Datenbank wird auch durch d notiert. Die Notiz $\llbracket \rrbracket_{ind}$ unterscheidet sich mit der Semantik der SQL.

Zum Unterschied von der Semantik der SQL hat die Semantik für Unbestimmtheiten zwei Veränderungen: Zuerst fügen zwei zusätzlichen Parametern, δ und γ , hinzu, aber δ wird nur in *where* Klausel und $\langle target\ list \rangle$ benutzt; Zweit hat die *select* – Anweisung hat verschiedenen Sinn in der unbestimmten Semantik.

Unter SQL Semantik wendet die *select* Anweisung nur die vollständigen Informationen an und sie hat eine einzelne Interpretation. Aber die Semantik braucht mindestens zwei Interpretationen für die unvollständigen Informationen. Eine Interpretation ist die Informationen, die die Anfrage durch *mögliche Zuordnung* ausgewählt hat, und als *mögliche Interpretation* genannt. Die zweite Interpretation, *bestimmte Interpretation* ist die Information, die die Anfrage durch *bestimmte Zuordnung* ausgewählt hat. Welche Interpretation übernommen ist wird von dem Benutzer per syntaktische Konstruktionen in der Anfrage spezifiziert. Es ist wichtig zu garantieren, dass eine Anfrage keine unmöglichen Antworten erzeugen wird, d.h.: die Antworten der Anfrage soll eine Submenge zu möglicher Interpretation, bzw. eine Obermenge zu bestimmter Interpretation sein. Dazu wird einen Begriff eingeführt -- *Vervollständigung (Completion)*.

Wir stellen uns einen unbestimmten Zeitpunkt als eine Menge von möglichen Zeitpunkten vor, wobei einer „echt“ ist, aber welcher ist unbekannt. Jeder mögliche Zeitpunkt stellt eine verschiedene, vollständige Realität dar. Jede Möglichkeit wird eine Vervollständigung eines Zeitpunktes benannt.

Vervollständigung (Completion) eines unbestimmten Zeitpunktes. Sei $\alpha = (\alpha^* \sim \alpha^*, P_\alpha)$.

Eine Vervollständigung eines unbestimmten Zeitpunktes α ist α_i , wobei α_i ein bestimmter Zeitpunkt ist, sodass $\alpha^* \leq \alpha_i \leq \alpha^*$. Die Menge aller Vervollständigungen für einen Zeitpunkt α bezeichnet man $C(\alpha)$.

Statt Zeitpunkt kann auch Zeitraum, Intervall usw. Daten verwendet werden. Wenn die unbestimmten Elemente in einem Tupel aufgetaucht werden, nennen wir die Menge des Tupels als $C(t)$.

In der Semantik für Unbestimmtheiten wird die mögliche Interpretation durch Verwendung der Plausibilität 1 erlangt; Und die bestimmte Interpretation wird nur mit Plausibilität 100 gewonnen. Folgende werden die Beispiele in *where* Klausel gegeben:

$$[\text{WHERE} \langle \text{predicate} \rangle]_{\text{ind}}(100, r) = \{ t \mid t \in r \wedge \forall t' \in C(t) ([\langle \text{predicate} \rangle]_{\text{SQL}}(t')) \}$$

Obere Formel beschreibt die bestimmte Interpretation in *where* Klausel, damit ist die Plausibilität 100; *Bestimmte Interpretation* wählt nur die tupel aus, dass die bei SQL Semantik in allen Completionen des Tupels ausgewählt sind.

Die *mögliche Interpretation* wählt die tupel aus, dass die bei SQL Semantik in einigen Completionen des Tupels ausgewählt sind; Die wird in der folgenden Formel beschrieben:

$$[\text{WHERE} \langle \text{predicate} \rangle]_{\text{ind}}(1, r) = \{ t \mid t \in r \wedge \exists t' \in C(t) ([\langle \text{predicate} \rangle]_{\text{SQL}}(t')) \}$$

wobei Plausibilität = 1.

Die Semantik für Unbestimmtheiten ist *zuverlässig*, D.h, keine unrichtigen Ergebnisse auftreten. Die Ergebnisse der *where* Klausel auf einer Vervollständigung von einer Relation r unter der SQL Semantik sollen konsistent mit der Ergebnisse unter der Semantik für Unbestimmtheiten von der Relation r . Es garantiert, dass die Semantik zu Unbestimmtheiten konsistent für SQL Semantik ist.

THEORIE 1. $[\text{WHERE} \langle \text{predcate} \rangle]_{\text{ind}}(1, r)$ ist zuverlässig, das heißt, für jede *where* Klausel W ,

$$\forall r' \in C(r) [[W]_{\text{SQL}}(r') \in C([W]_{\text{ind}}(1, r'))]$$

Die Semantik für Unbestimmtheiten ist auch *maximal*. Die Semantik erzeugt weitere Informationen und erhält mehr Vervollständigungen, und die Ergebnisse könnten nicht mehr zuverlässig sein. Aus Theorie 1 wissen wir, dass $[W]_{\text{ind}}(1, r')$ alle benötigten Vervollständigungen erhält. So muss es bestimmt sein, dass keine weiteren Vervollständigungen enthalten werden.

THEORIE 2. $[\text{WHERE} \langle \text{predcate} \rangle]_{\text{ind}}(1, r)$ ist maximal, das heißt, für jede *where* Klausel W ,

$$\forall c \in C(\llbracket W \rrbracket_{\text{ind}}(1, r)) [\exists r' \in C(r) (c = \llbracket W \rrbracket_{\text{SQL}}(r'))]$$

So für alle *where* Klauseln und unbestimmten Relationen *r* sind:

$$C(\llbracket W \rrbracket_{\text{ind}}(1, r)) = \bigcup_{r' \in C(r)} \llbracket W \rrbracket_{\text{SQL}}(r')$$

Wenn eine Datenbank nur die bestimmten Informationen hat, dann existiert es nur eine Vervollständigung und eine Interpretation für eine Anfrage. Die mögliche und bestimmte Interpretationen sind äquivalent, und die Glaubwürdigkeit und Plausibilität haben keine Wirkungen in der Datenbank.

Vorher wurden zwei wichtigsten Interpretationen, die bestimmte Interpretation und mögliche Interpretation, vorgestellt. Es existiert noch andere Interpretation, die mit anderen Wert von Glaubwürdigkeiten und Plausibilität zwischen 1~100.

3.3.3 Operationale Semantik für Unbestimmtheit

Um die Effizienz zu erhöhen, wird eine operationale Semantik eingeführt. Dazu gibt es drei Erweiterungen zur Semantik für Unbestimmtheiten: ein Prädikat – *Before*, zugehörige Ausgabewerte bzw. *Ersetzung(Replace)* Verfahren.

In SQL Semantik können wir die Reihenfolge von bestimmten Daten durch nachfolgende Formel bestimmen: Wir nehmen die unbestimmten Daten *E* als Zeitpunkte an, dann wie die Reihenfolge der drei Zeitpunkte, *E1, E2, E3*, ist, wird von dem Ausgabewert von *Before* abhängen. *E1* ist im Zeitraum $\langle E2, E3 \rangle$ aufgetaucht, wenn *E1* später als *E2* und früher als *E3* ist. Der Wahrheitswert des Prädikates hängt von der Ausgabewerte von *Before*.

$$[\langle E1 \rangle \text{ OVERLAPS Zeitraum}(\langle E2 \rangle, \langle E3 \rangle)] = \text{Before}([\langle E2 \rangle], [\langle E1 \rangle]) \wedge \text{Before}([\langle E1 \rangle], [\langle E3 \rangle])$$

Aber für Unbestimmte Daten ist das nicht genug. Zum Beispiel, wenn $E1 = [01,05,2003 \sim 31,05,2003]$, $E2 = [15,04,2003 \sim 15,05,2003]$, $E3 = [16,05,2003 \sim 31,06,2003]$, dann kann die Reihenfolge von *E1, E2, E3* nicht einfach anzuordnen werden. Für die unbestimmte Daten, fügen wir im Prädikat *Before* einen neuen Sinn hinzu; *Before* wird mit Plausibilität zusammen gespielt und durch Zeichen „ \leq “ dargestellt:

$$\text{Before}(\alpha, \beta, \gamma) = \{ \text{True} \mid \text{Pr}[\alpha \leq \beta] \times 100 \geq \gamma \} \\ \cup \{ \text{False} \mid \text{Pr}[\beta < \alpha] \times 100 \geq \gamma \}$$

Wobei γ die Plausibilität ist.

Zu der Ausgabe des Prädikates *Before* definieren wir vier Ausgabewerte, leer Menge, Menge von True, Menge von False, und Menge von True und False:

- { } : gilt weder $\alpha \leq \beta$, noch die Negation;
- { True } : gilt nur $\alpha \leq \beta$, aber die Negation nicht;
- { False } : gilt nicht $\alpha \leq \beta$, aber die Negation gilt;
- { True, False } : $\alpha \leq \beta$ und die Negation haben gleiches Gewicht.

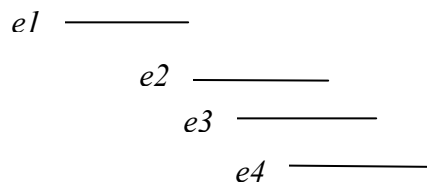
Für weitere Darstellung der operationalen Semantik benutzen wir die folgenden Prädikate und logische Formel in der *where* Klausel:

$$\begin{aligned}
[<Präd1> \text{ AND } <Präd2>](\gamma, r) &= [<Präd1>](\gamma, r) \cap [<Präd2>](\gamma, r) \\
[<Präd1> \text{ OR } <Präd2>](\gamma, r) &= [<Präd1>](\gamma, r) \cup [<Präd2>](\gamma, r) \\
[\text{NOT } <Prädikat>](\gamma, r) &= \{x \mid \neg x \in [<Prädikat>](\gamma, r)\}
\end{aligned}$$

Wir illustrieren die Semantik mit einigen Beispiele(Abb3.3):

		β			
		$e1$	$e2$	$e3$	$e4$
α	$e1$	1.00	1.00	1.00	1.00
	$e2$	0	1.00	0.86	0,96
	$e3$	0	0.16	1.00	0.73
	$e4$	0	0.05	0.27	1.00

(a) Tabelle für $Pr[\alpha \leq \beta]$



(b) Darstellung der Zeitlinear

$[e2 \leq e3](100, r) = before(e2, e3, 100) = \{\}$	$[\text{NOT } <e2 \leq e3 >](100, r) = \{\}$
$[e2 \leq e3](50, r) = \{True\}$	$[\text{NOT } <e2 \leq e3 >](50, r) = \{False\}$
$[e2 \leq e3](1, r) = \{True, False\}$	$[\text{NOT } <e2 \leq e3 >](1, r) = \{True, False\}$
$[e2 \leq e3 \text{ AND } e1 \leq e4](100, r) = \{\}$	$[\text{NOT } (<e2 \leq e3 > \text{ AND } <e1 \leq e4 >)](100, r) = \{\}$
$[e2 \leq e3 \text{ OR } e1 \leq e4](100, r) = \{True\}$	$[\text{NOT } (<e2 \leq e3 > \text{ OR } <e1 \leq e4 >)](100, r) = \{True\}$

Abb.3.3 Beispiel für Before und logische Formel

Die Tabelle (a) zeigt die Wahrscheinlichkeiten des Prädikates *Before* von 4 unbestimmten Zeitpunkte $e1, e2, e3, e4$, (b) ist grafische Darstellung von den Zeitpunkten. Mit verschiedener Plausibilität wird die *where* Klausel wie Untere Formeln konstruiert.

Die ersten zwei Erweiterungen der Operationalen Semantik haben die Reihenfolge der Plausibilität unterstützt in der *where* Klausel. Und das Ersetzung(Replace) Verfahren unterstützt die Glaubwürdigkeit für target List per Ersetzung Unbestimmtheiten durch Bestimmtheiten in der *from* Klausel.

Ersetzung(Replace): Sei Tuple $t = (X, \alpha_1, \dots, \alpha_n)$, wobei X bestimmte Werte und $\alpha_1, \dots, \alpha_n$ unbestimmte Werte sind. Dann

$$Replace(\delta, t) = (X, R(\alpha_1), \dots, R(\alpha_n))$$

R ist die Ersetzung Strategie. Abb.3.4 verzeichnet die Strategien für die Kombinationen der Glaubwürdigkeitwerte und Type der Zeitwerte. $E[\alpha]$ ist Erwartungswert.

	Zeitpunkt	Zeitraum		Abstan
		Start	Ende	
INDETERMINAT	α	α	α	α
EXPECTED	$E[\alpha]$	$E[\alpha]$	$E[\alpha]$	$E[\alpha]$
MIN	α^*	α^*	α^*	α^*
MAX	α^*	α^*	α^*	α^*

Abb. 3.4 Ersetzung Strategien

Beispielweise wird die Tuple

$t = (\text{Centurion}, \text{AB33}, [1, \text{März} \sim 31, \text{März} - 1, \text{Juni} \sim 30, \text{Juni}])$ betrachtet

Der unbestimmte Zeitraum wird durch bestimmte Werte ersetzt. Es gibt 4 möglichen Ergebnisse:

$\text{Replace}(\text{INDETERMINATE}, t) = t$

$\text{Replace}(\text{EXPECTED}, t) = (\text{Centurion}, \text{AB33}, [15, \text{März} - 15, \text{Juni}])$

$\text{Replace}(\text{MIN}, t) = (\text{Centurion}, \text{AB33}, [31, \text{März} - 1, \text{Juni}])$

$\text{Replace}(\text{MAX}, t) = (\text{Centurion}, \text{AB33}, [1, \text{März} - 30, \text{Juni}])$

4. Zusammenfassung mit Beispiel

Vorher haben wir die Erweiterungen der SQL für die Unbestimmtheiten auf zwei Aspekte, Syntax und Semantik, erläutert. Nun sehen wir eine Anfrage wie die in erweiterten Syntax und erweiterten Semantik für Unbestimmtheiten.

Die Syntax der Anfrage ist:

```
SET DEFAULT PLAUSIBILITY 60
SELECT r.Lager, r.Lot_Num, p.Serial_Num, r.Wann
FROM Received AS r WITH CREDIBILITY INDETERMINATE,
     In_Production AS p WITH CREDIBILITY INDETERMINATE
WHERE p.Model = „Centurion“ AND r.Part = „Flügel Strebe“
AND r.Wann OVERLAPS p.Während
```

Die Anfrage wird übersetzt in Semantik wie Folgendes:

$$\llbracket [Q] \rrbracket (d) = \{(r.Lager, r.Lot_Num, p.SerialNum, r.Wann) \mid$$

$$r \in \text{Replace}(\text{INDETERMINATE}, \text{Received})$$

$$\wedge p \in \text{Replace}(\text{INDETERMINATE}, \text{In_production})$$

$$\wedge p.Model = \text{‘Centurion’} \wedge r.Part = \text{‘Flügel Strebe’}$$

$$\wedge True \in \{\text{Before}(p.Während_{start}, r.Wann, 60) \cap$$

$$\text{Before}(r.Wann, p.Während_{ende}, 60)\}$$

Mit der Reihenfolge der Plausibilität 60 und Glaubwürdigkeit als INDETERMINATE sind die Antworten wie in folgender Tabelle:

Lager	Lot_Num	Serial_Num	Wann
Boeing	23	AB33	10,Mai ~ 29,Mai
Cessna	30	AB33	30,Mai ~ 18,Juni
Cessna	31	AB34	13,Juni ~ 02,Juli

Literatur

- [Dyr&Sno97] Dyreson, C. E. und Snodgrass, R. T. : *Supporting Valid-time Indeterminacy*. ACM Transactions on Database Systems, 23 (1), 1997, pp. 1-57
- [Dyr&Sno93] Dyreson, C. E. und Snodgrass, R. T. : *Valid-time Indeterminacy*. ICDE 1993, pp. 335-343
- [Lak&Leo] Lakshmanan, L. V. S. und Leone, N., Ross, R., Subrahmanian, V.S.: *ProbView: A Flexible Probabilistic Database System*. ACM Transactions on Database Systems, 22 (3), 1997, pp. 419-469
- [Reinhard] Reinhard, C.: *Realisierung einer temporalen Erweiterung von SQL auf einem objekt-relationalen Datenbankmanagementsystem*. <http://www.dbs.uni-hannover.de/ftp/theses/reinhard/DpA.pdf>, 1999, pp.1-38