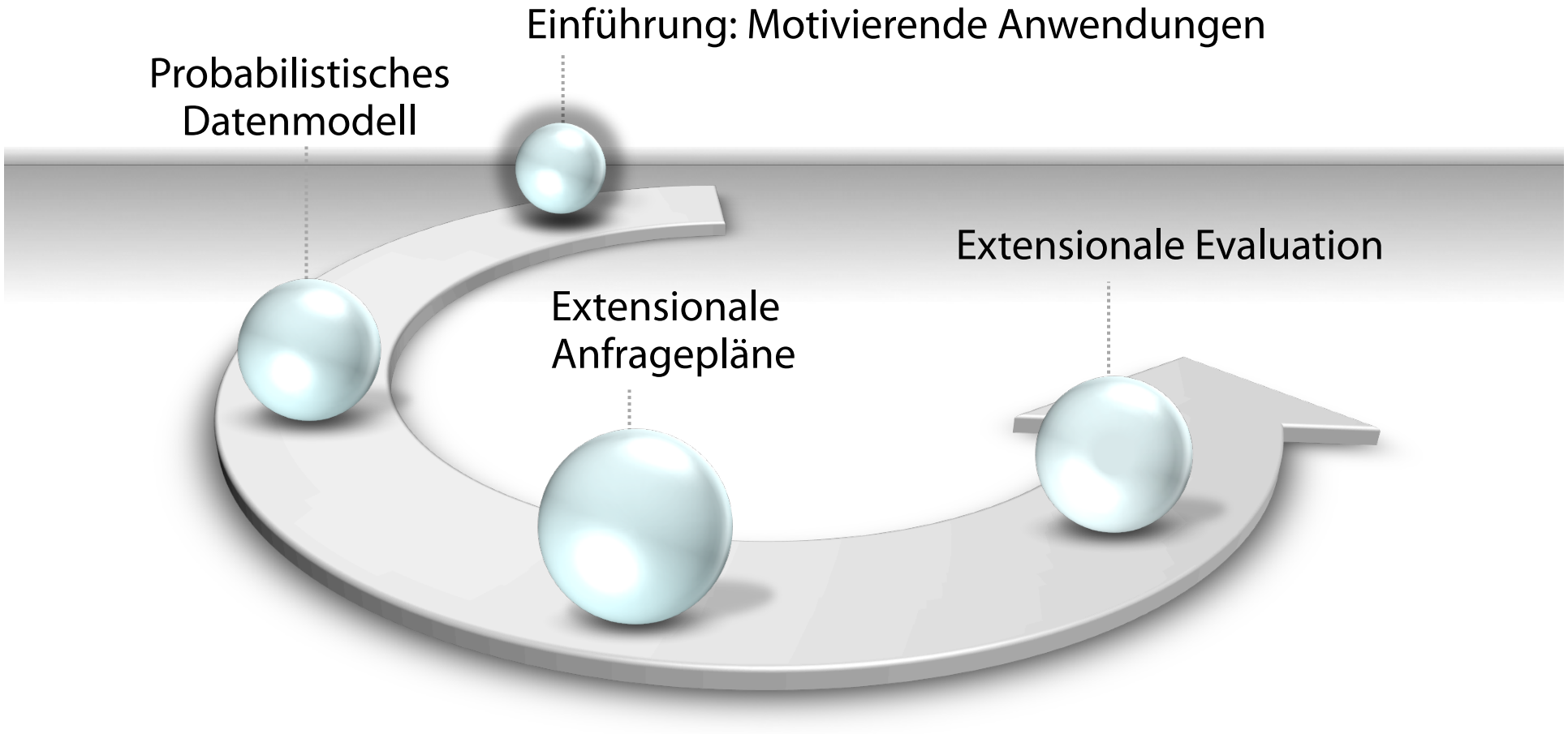
Non-Standard-Datenbanken

Probabilistische Datenbanken

Prof. Dr. Ralf Möller
Universität zu Lübeck
Institut für Informationssysteme

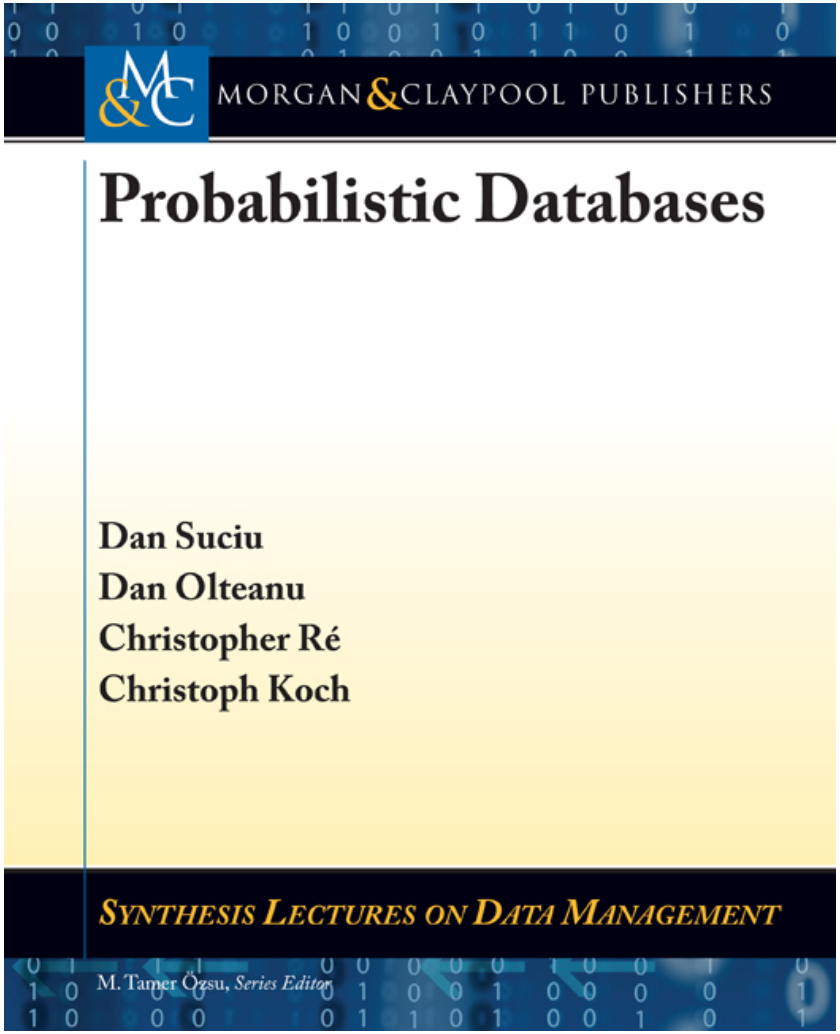
Non-Standard-Datenbanken

Probabilistische Datenbanken



Danksagung

Die Präsentationen sind nach einem Tutorial von Dan Suciu gestaltet und basieren auf dem Lehrbuch Probabilistic Databases



Probabilistische Datenbanken

- Daten: Relationale Daten plus Wahrscheinlichkeiten, um Grad der Unsicherheit auszudrücken
- Anfragen: SQL-Anfragen, deren Antworten annotiert sind mit Ausgabewahrscheinlichkeiten
- Formale Logik kombiniert mit Inferenzen über Wahrscheinlichkeiten
- Ermöglicht Ihnen einen neuen Blick auf beides, Datenbanken und Wahrscheinlichkeiten

Beispiel 1: Informationsextraktion

52-A Goregaon West Mumbai 400 076

Extraktion

Standard DB: Speichere nur wahrscheinlichste Extraktion

Id	House_no	Area	City	Pincode	Prob
1	52	Goregaon West	Mumbai	400 062	0.1
1	52-A	Goregaon	West Mumbai	400 062	0.2
1	52-A	Goregaon West	Mumbai	400 062	0.5
1	52	Goregaon	West Mumbai	400 062	0.2

Probabilistische DB: Speicher die meisten/alle Extraktionen um Recall zu erhöhen

Kernidee: Wahrscheinlichkeiten gegeben durch Extraktion korrelieren mit der Präzision der Extraktion

Gupta, Sarawagi: Creating Probabilistic Databases from Information Extraction Models. VLDB **2006**

Beispiel 2: Modellierung fehlender Daten

id	age	edu	inc	nw
t1	20	HS	?	?
t2	20	BS	50K	100K
t3	20	?	50K	?
t4	20	HS	100K	500K
t5	20	?	?	?
t6	20	HS	50K	100K
t7	20	HS	50K	500K
t8	?	HS	?	?
t9	30	BS	100K	100K
t10	30	?	100K	?
t11	30	HS	?	?
t12	30	MS	?	?
t13	40	BS	100K	100K
t14	40	HS	?	?
t15	40	BS	50K	500K
t16	40	HS	?	500K
t17	40	HS	100K	500K

Standard-DB: NULL

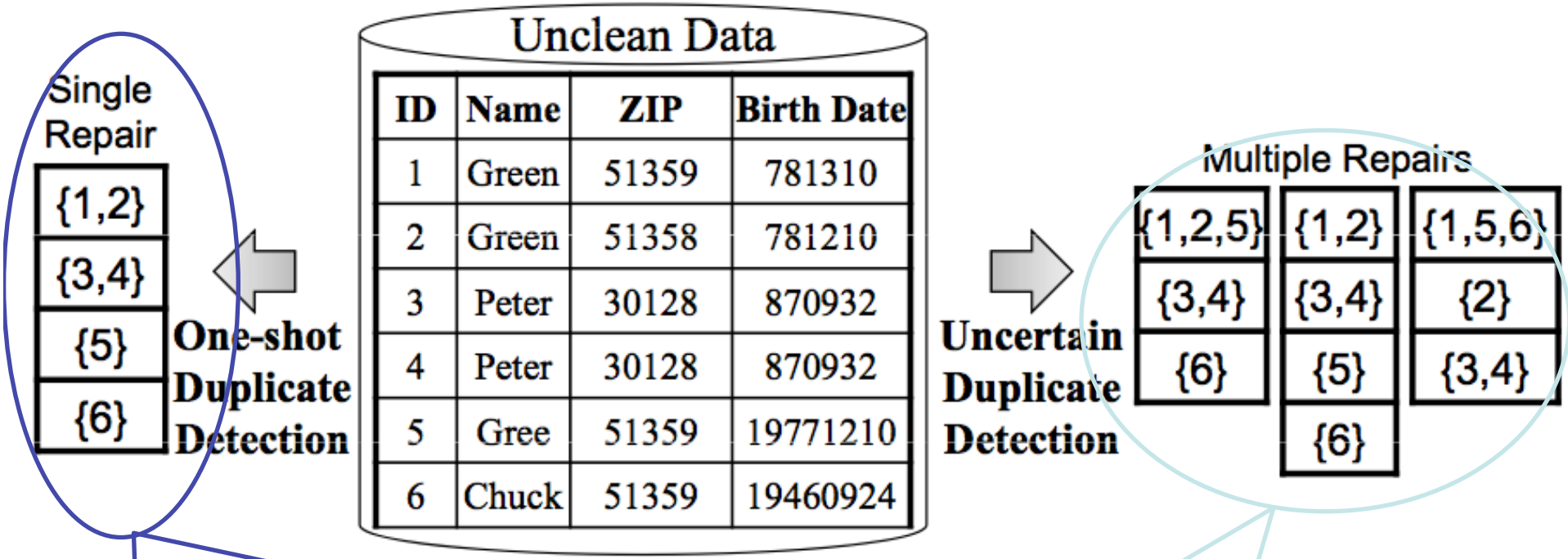
Kernidee:
Inferiere Verteilung für fehlende Daten.

Probabilistische DB:
Verteilung auf mögl. Werten

id	age	edu	inc	nw	prob
t12.1	30	MS	50K	100K	0.30
t12.2	30	MS	50K	500K	0.45
t12.3	30	MS	100K	100K	0.10
t12.4	30	MS	100K	500K	0.15

Stoyanovich, Davidson, Milo, Tannen: Deriving probabilistic databases with inference ensembles. ICDE **2011**

Beispiel 3: Datenreinigung



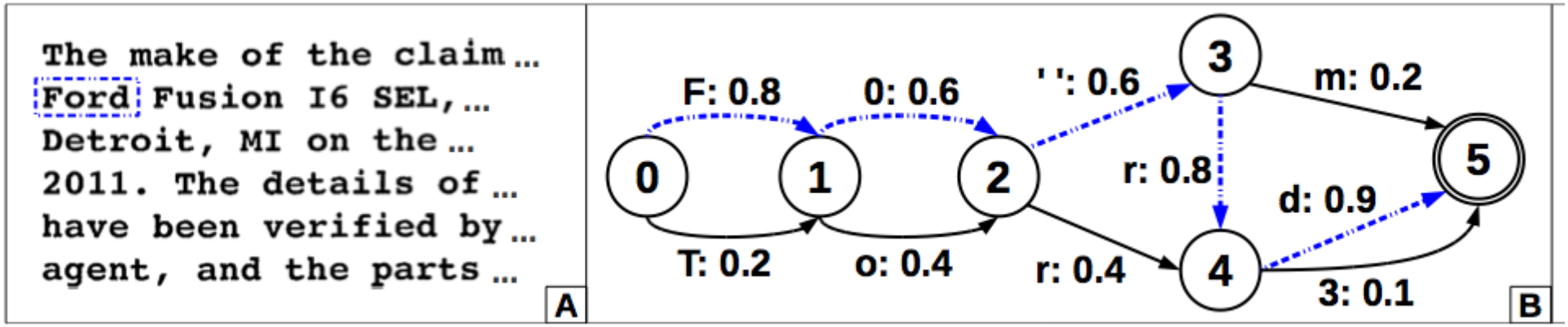
Standard-DB
Reinigung bedeutet eine mögliche Reparatur zu wähen

Probabilistische DB
Speichere viele/alle möglichen Reparaturen

Herausforderung: Representation von multiplen Reparaturen

Beskales, Soliman, Ilyas, Ben-David: Modeling and Querying Possible Repairs in Duplicate Detection. PVLDB **2009**

Beispiel 4: OCR



Verwendung von OCRopus von Google Books: Ausgabe ist stochastischer Automat
Üblicherweise wird nur Maximum Apriori Estimate (MAP) gespeichert
Mit probabilistischer Databasis: Speicherung verschiedener Möglichkeiten: Erhöhe Recall.

```
SELECT DocId, Loss
FROM Claims
WHERE Year = 2010
    AND DocData LIKE '%Ford%';
```

Kumar, Ré: Probabilistic Management of OCR Data using an RDBMS. PVLDB **2011**

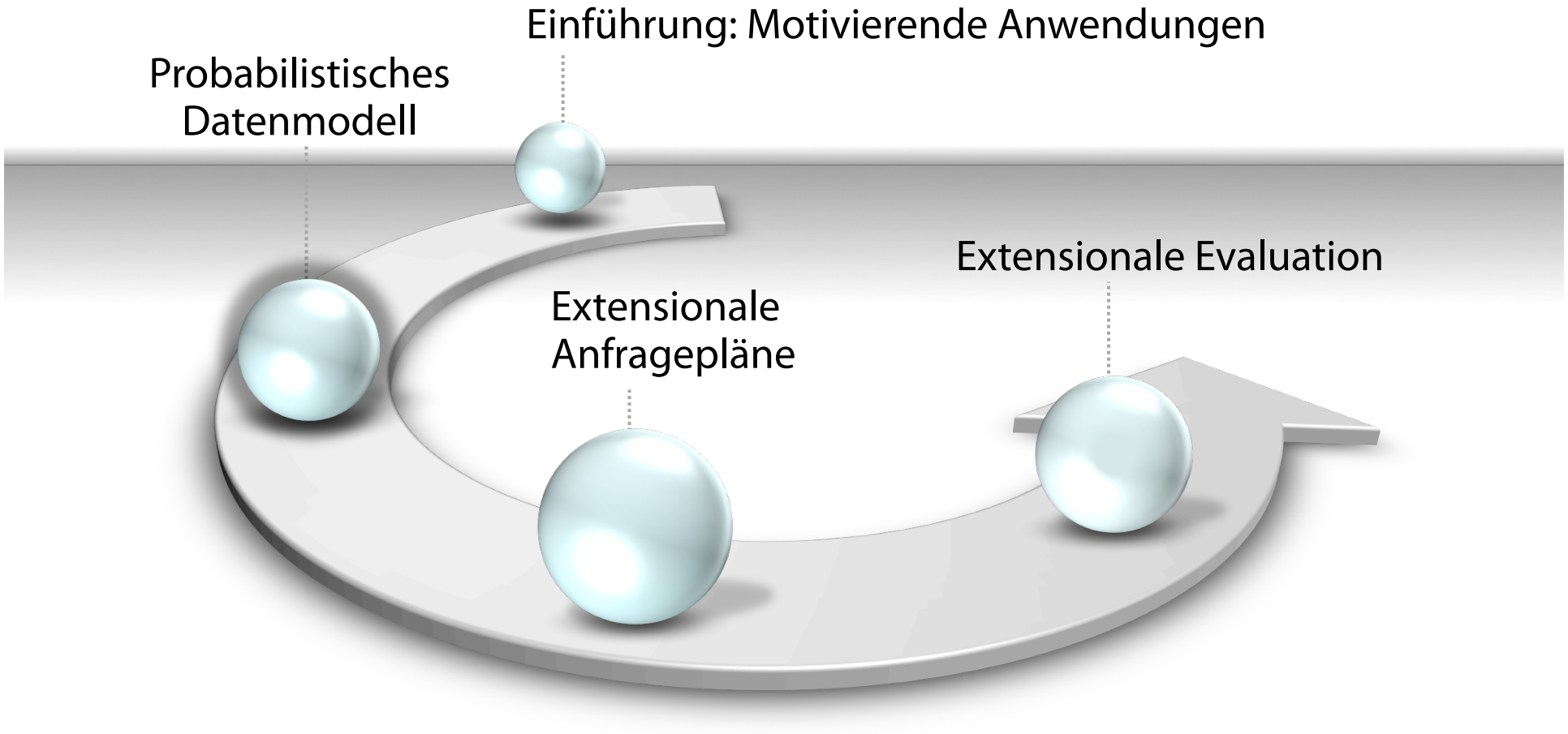
Zusammenfassung der Anwendungen

- Strukturierte, aber unsichere Daten
- Modelliert als probabilistische Daten
- Antworten für SQL queries annotiert mit Wahrscheinlichkeiten

Probabilistische Datenbank:
- Kombination aus Standard-Datenmanagement mit probabilistischer Inferenz

Non-Standard-Datenbanken

Probabilistische Datenbanken



Wiederholung: Relationales Datenmodell

Daten:
gespeichert in Relationen (= Tabellen)

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Wiederholung: Relationales Datenmodell

Daten:
gespeichert in Relationen (= Tabellen)

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Anfragen: SQL,

Find all owners of objects in the Office

```
-- SQL: z.B. Postgres

SELECT DISTINCT Owner.name
FROM Owner, Location
WHERE Owner.object = Location.object
    and  Location.loc = 'Office'
```

Wiederholung: Relationales Datenmodell

Daten:
gespeichert in Relationen (= Tabellen)

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Anfragen: SQL,

Find all owners of objects in the Office

```
-- SQL: z.B. Postgres

SELECT DISTINCT Owner.name
FROM Owner, Location
WHERE Owner.object = Location.object
    and  Location.loc = 'Office'
```

Vereinigung konjunktiver Anfragen
Unions of Conjunctive Queries (UCQs)

$$Q(z) = \text{Owner}(z,x), \text{Location}(x,t,y), y=\text{'Office'}$$

NB x,t sind existenzquantifiziert:

$$Q(z) = \exists x \; \exists t \; (\text{Owner}(z,x), \text{Location}(x,t,\text{'Office'}))$$

Wiederholung: Relationales Datenmodell

Daten:
gespeichert in Relationen (= Tabellen)

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Anfragen: SQL,

Vereinigung konjunktiver Anfragen
Unions of Conjunctive Queries (UCQs)

Find all owners of objects in the Office

```
-- SQL: z.B. Postgres

SELECT DISTINCT Owner.name
FROM Owner, Location
WHERE Owner.object = Location.object
    and  Location.loc = 'Office'
```

$Q(z) = Owner(z,x), Location(x,t,y), y='Office'$

NB x,t sind existenzquantifiziert:

$Q(z) = \exists x \exists t (Owner(z,x), Location(x,t,'Office'))$

Antwort: Q=

Name
Joe
Jim

Wiederholung: Relationales Datenmodell

Daten:
gespeichert in Relationen (= Tabellen)

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Anfragen: SQL,

Find all owners of objects in the Office

```
-- SQL: z.B. Postgres

SELECT DISTINCT Owner.name
FROM Owner, Location
WHERE Owner.object = Location.object
    and  Location.loc = 'Office'
```

Vereinigung konjunktiver Anfragen
Unions of Conjunctive Queries (UCQs)

$Q(z) = Owner(z,x), Location(x,t,y), y='Office'$

NB x,t sind existenzquantifiziert:

$Q(z) = \exists x \exists t (Owner(z,x), Location(x,t,'Office'))$

Antwort: Q=

Name
Joe
Jim

Wiederholung: Komplexität der Anfragebeantwortung

Anfrage Q, Datenbank D

- Datenkomplexität:
  fix Q, Komplexität = f(D)
- Anfragekomplexität:
  fix D, Komplexität = f(Q)
- Kombinierte Komplexität: Komplexität = f(D,Q)

Datenkomplexität wird im Bereich der Datenbankforschung betrachtet

Unvollständige Datenbank

Definition Eine unvollständige Datenbank ist eine endliche Menge von Datenbankinstanzen
$\mathbf{W} = (W_1, W_2, \ldots, W_n)$

Jedes W_i heißt mögliche Welt

Unvollständige Datenbank

Definition Eine unvollständige Datenbank ist eine endliche Menge von Datenbankinstanzen
$\mathbf{W} = (W_1, W_2, \ldots, W_n)$

Jedes W_i heißt mögliche Welt

W_1	W_2	W_3	W_4

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Unvollständige Datenbank

Definition Eine unvollständige Datenbank ist eine endliche Menge von Datenbankinstanzen
$\mathbf{W} = (W_1, W_2, \ldots, W_n)$

Jedes W_i heißt mögliche Welt

W_1

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

W_2

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

W_3

W_4

Unvollständige Datenbank

Definition Eine unvollständige Datenbank ist eine endliche Menge von Datenbankinstanzen
$\mathbf{W} = (W_1, W_2, \ldots, W_n)$

Jedes W_i heißt mögliche Welt

W_1

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

W_2

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

W_3

Owner

Name	Object
Jim	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

W_4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Unvollständige Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine unvollständige DB **W**:

- Eine Antwort t ist sicher (certain), falls $\forall W_i,\ t \in Q(W_i)$
- Eine Antwort t ist möglich (possible) falls $\exists W_i,\ t \in Q(W_i)$

Unvollständige Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine unvollständige DB **W**:

- Eine Antwort t ist sicher (certain), falls $\forall W_i,\ t \in Q(W_i)$
- Eine Antwort t ist möglich (possible) falls $\exists W_i,\ t \in Q(W_i)$

Q(z) = Owner(z,x),
Location(x,t,'Office')

W_1

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

W_2

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

W_3

Owner

Name	Object
Joe	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

W_4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Unvollständige Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine unvollständige DB **W**:

- Eine Antwort t ist sicher (certain), falls $\forall W_i,\ t \in Q(W_i)$
- Eine Antwort t ist möglich (possible) falls $\exists W_i,\ t \in Q(W_i)$

Q(z) = Owner(z,x),
Location(x,t,'Office')

W_1

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

W_2

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

Q=

Joe

W_3

Owner

Name	Object
Joe	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

Q=

Joe

W_4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

Unvollständige Datenbank: Anfragesemantik

Definition Given query Q, incomplete database **W**:

- An answer t is certain, if $\forall W_i$, $t \in Q(W_i)$
- An answer t is possible if $\exists W_i$, $t \in Q(W_i)$

$$Q(z) = \text{Owner}(z,x), \text{Location}(x,t,\text{'Office'})$$

Certain answers to Q: Joe
Possible answers to Q: Joe, Jim

W_1

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

W_2

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

Q=

Joe

W_3

Owner

Name	Object
Joe	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

Q=

Joe

W_4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

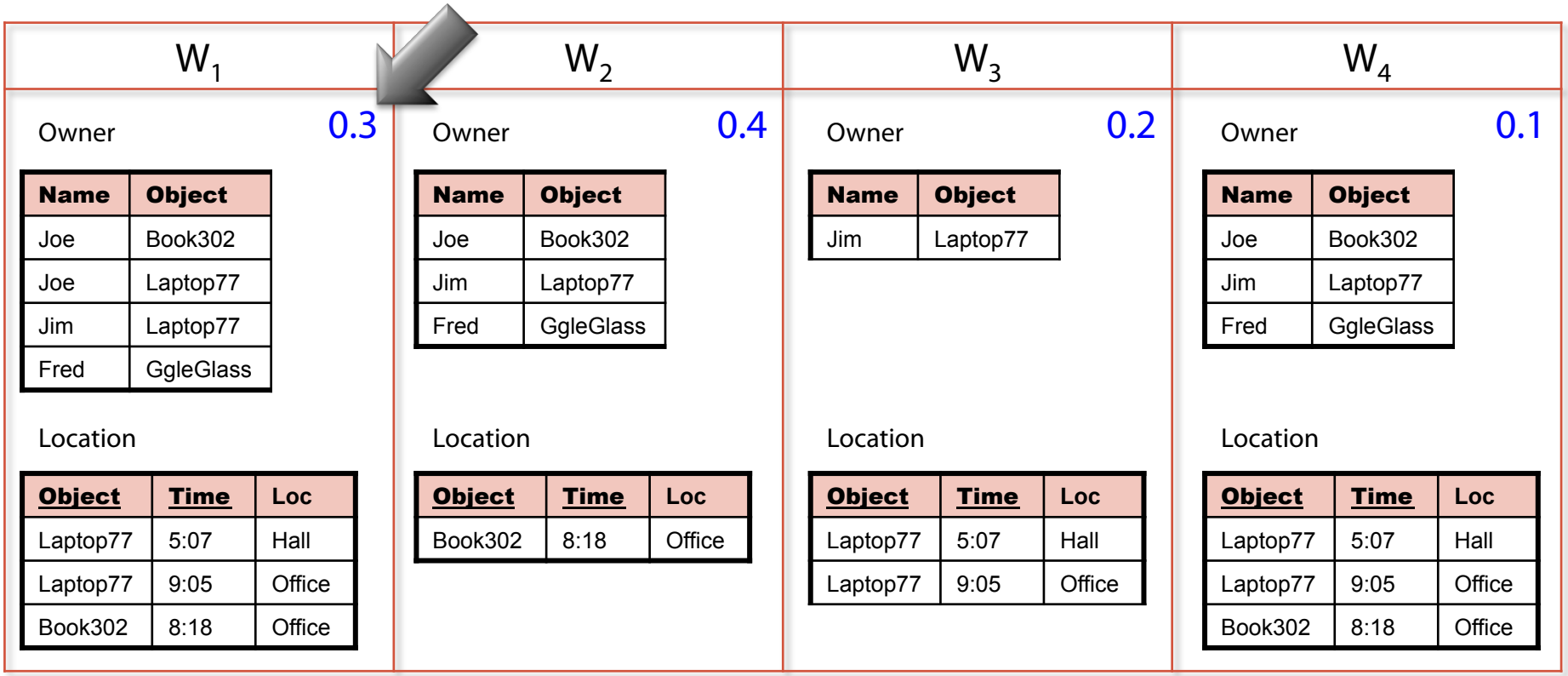
Joe
Jim

Probabilistische Datenbank

Definition Eine probabilistische DB ist ein Tupel (**W**, P), wobei **W** eine unvollständige DB und P: **W** → [0,1] eine Wahrscheinlichkeitsverteilung ist: $\Sigma_{i=1,n} P(W_i) = 1$

Probabilistische Datenbank

Definition Eine probabilistische DB ist ein Tupel (**W**, P), wobei **W** eine unvollständige DB und P: **W** → [0,1] eine Wahrscheinlichkeitsverteilung ist: $\Sigma_{i=1,n} P(W_i) = 1$



W_1 — 0.3

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

W_2 — 0.4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

W_3 — 0.2

Owner

Name	Object
Jim	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

W_4 — 0.1

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Probabilistische Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine probabilistische DB (**W**,P):

- Die Randwahrscheinlichkeit einer Antwort t ist:
  $P(t) = \Sigma \{ P(W_i) \mid W_i \in \mathbf{W}, t \in Q(W_i) \}$

Probabilistische Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine probabilistische DB (**W**,P):

- Die Randwahrscheinlichkeit einer Antwort t ist:
  $P(t) = \Sigma \{ P(W_i) \mid W_i \in \mathbf{W}, t \in Q(W_i) \}$

Q(z) = Owner(z,x),
Location(x,t,'Office')

W_1 — 0.3

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

W_2 — 0.4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

W_3 — 0.2

Owner

Name	Object
Jim	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

W_4 — 0.1

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Probabilistische Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine probabilistische DB (**W**,P):

- Die Randwahrscheinlichkeit einer Antwort t ist:
  $P(t) = \Sigma \{ P(W_i) \mid W_i \in \mathbf{W}, t \in Q(W_i) \}$

Q(z) = Owner(z,x),
Location(x,t,'Office')

W_1 — 0.3

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

W_2 — 0.4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

Q=

Joe

W_3 — 0.2

Owner

Name	Object
Joe	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

Q=

Joe

W_4 — 0.1

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

Probabilistische Datenbank: Anfragesemantik

Definition Gegeben eine Anfrage Q, eine probabilistische DB (**W**,P):

- Die Randwahrscheinlichkeit einer Antwort t ist:
  $P(t) = \Sigma \{ P(W_i) \mid W_i \in \mathbf{W}, t \in Q(W_i) \}$

Q(z) = Owner(z,x),
Location(x,t,'Office')

P(Joe) = 1.0
P(Jim) = 0.4

W_1

0.3

Owner

Name	Object
Joe	Book302
Joe	Laptop77
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

W_2

0.4

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Book302	8:18	Office

Q=

Joe

W_3

0.2

Owner

Name	Object
Joe	Laptop77

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office

Q=

Joe

W_4

0.1

Owner

Name	Object
Joe	Book302
Jim	Laptop77
Fred	GgleGlass

Location

Object	Time	Loc
Laptop77	5:07	Hall
Laptop77	9:05	Office
Book302	8:18	Office

Q=

Joe
Jim

Diskussion

- Intuition: Eine probabilistische Datenbank sagt aus, dass eine Datenbank in einem von verschiedenen möglichen Zuständen ist. Jeder Zustand hat eine Wahrscheintlichkeit

- Mögliche Anfrageantworten: Eine Menge von Antworten, annotiert mit Wahrscheinlichkeiten:

  $(t_1, p_1), (t_2, p_2), (t_3, p_3), \ldots$

  Üblicherweise: $p_1 \geq p_2 \geq p_3 \geq \ldots$

- Problem: Die Anzahl der möglichen Welten in einer probabilistischen Datenbank ist sehr groß.

- Ziel: Anfragebeantwortung ohne explizite Generierung aller möglichen Welten (eventuell Einschränkungen in der Ausdrucksstärke hinnehmen)

Unabhängige und disjunkte Tupel

Definition Gegeben eines probabilistische DB (**W**, P).
Zwei Tupel t_1, t_2 werden heißen:

- unabhängig, falls: $P(t_1\ t_2) = P(t_1)\ P(t_2)$
- disjunkt (or exklusiv), falls: $P(t_1\ t_2) = 0$

Unabhängige und disjunkte Tupel

Definition Gegeben eines probabilistische DB (**W**, P).
Zwei Tupel t_1, t_2 werden heißen:

- unabhängig, falls: $P(t_1\ t_2) = P(t_1)\ P(t_2)$
- disjunkt (or exklusiv), falls: $P(t_1\ t_2) = 0$

Definition Eine probabilistische DB heißt *block-unabhängig-disjunkt* (BUD), falls die Tupel in Blöcke gruppiert werden können, so dass:

- Tupel vom gleichen Block disjunkt sind
- Tuplel von verschiedenen Blöcken unabhängig sind.

Beispiel: BUD-Tabelle

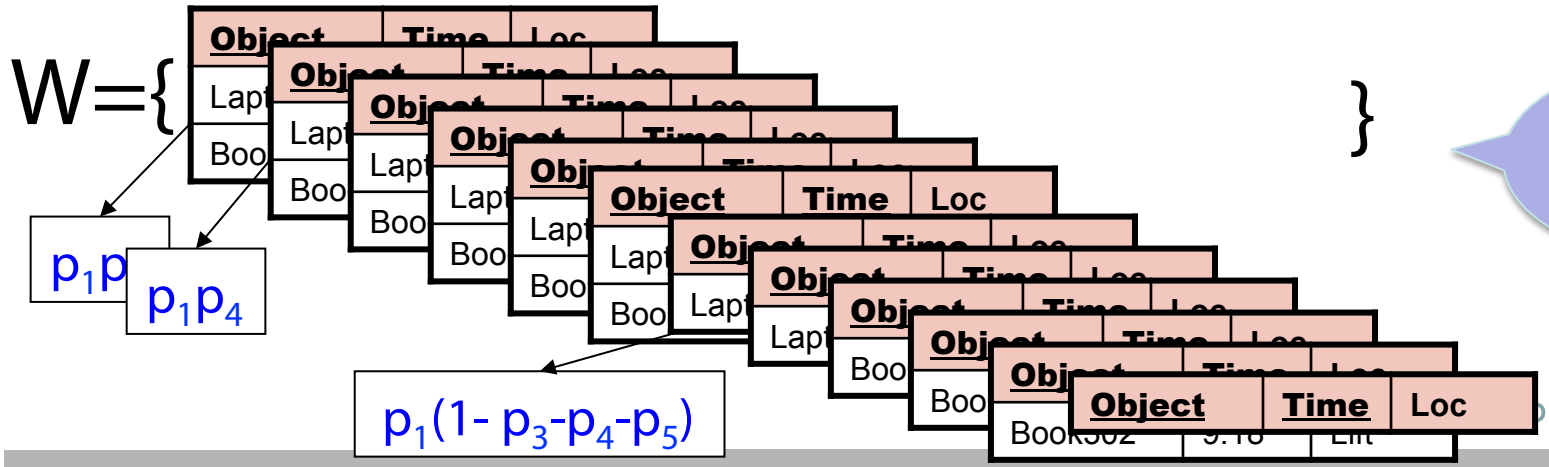
BUD Tabelle

Object	Time	Loc	P
Laptop77	9:07	Rm444	p_1
Laptop77	9:07	Hall	p_2
Book302	9:18	Office	p_3
Book302	9:18	Rm444	p_4
Book302	9:18	Lift	p_5

disjunkt

disjunkt

Unab-hängig



Mögliche Welten

Das Anfrage-Evaluationsproblem

Gegeben: BUD-Datenbank D, Anfrage Q, Ausgabetupel t

Berechne: P(t)

NB: D habe, sagen wir, 1.000.000 Tupel,
dann ist die Anzahl der möglichen Welten: $2^{1.000.000}$

Herausforderung: Berechne P(t) effizient, in der Größe von D

Datenkomplexität: die Komplexität von P hängt dramatisch von D ab.

Ein Beispiel

Boolesche Anfrage:
Join-Tupel vorhanden?

```
SELECT DISTINCT 'true'
FROM R, S
WHERE R.x = S.x
```

Q() = R(x), S(x,y)

$$P(Q) = 1- \{1- p1*[\ 1-(1-q1)*(1-q2)\ ]\} *$$

$$\{1- p2*[\ 1-(1-q3)*(1-q4)*(1-q5)\ ]\}$$

Man kann P(Q) in PTIME
bzgl. der Größe der DB D bestimmen

R

x	P
a1	p1
a2	p2
a3	p3

S

x	y	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

Zusammenfassung: Das probabilistische Datenmodell

- Mögliche-Welten-Semantik:
  Mächtig, aber schwierig zu repräsentieren

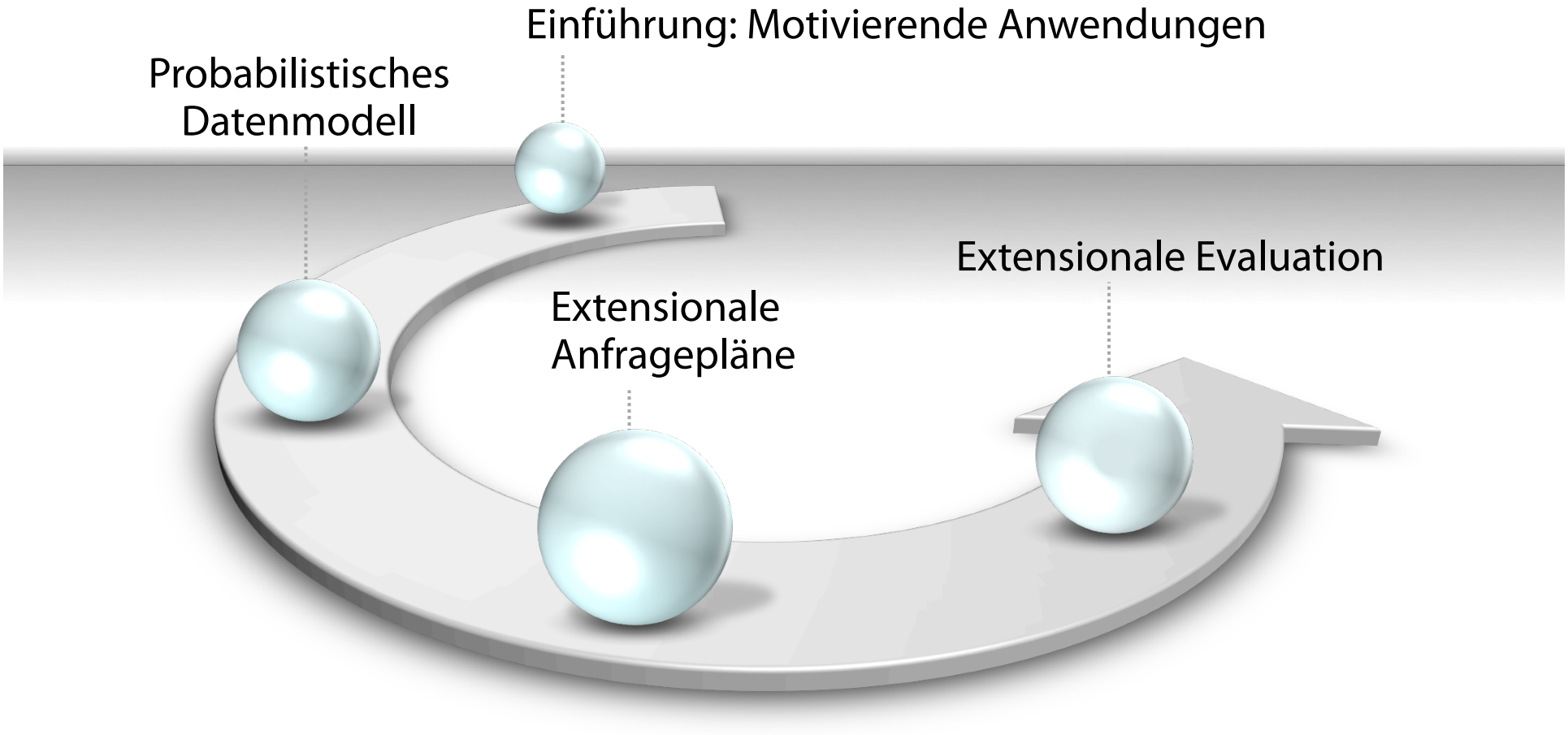
- Block-unabhängig-disjunkte Datenbasen haben effiziente Repräsentationen:
  D wird in traditioneller DB gespeichert

- Unabhängige Datenbasen: noch einfacher

Herausforderung: evaluiere Q effizient
bzgl. der Größe von D

Non-Standard-Datenbanken

Probabilistische Datenbanken



Relationale Algebra

1. Verbund (join) $\bowtie$
2. Projektion (mit Duplikat-Elimination) Π
3. Vereinigung $\cup$
4. Auswahl (selection) σ
5. Differenz: hier nicht verwendet -

Wiederholung: Anfragebearbeitungspläne

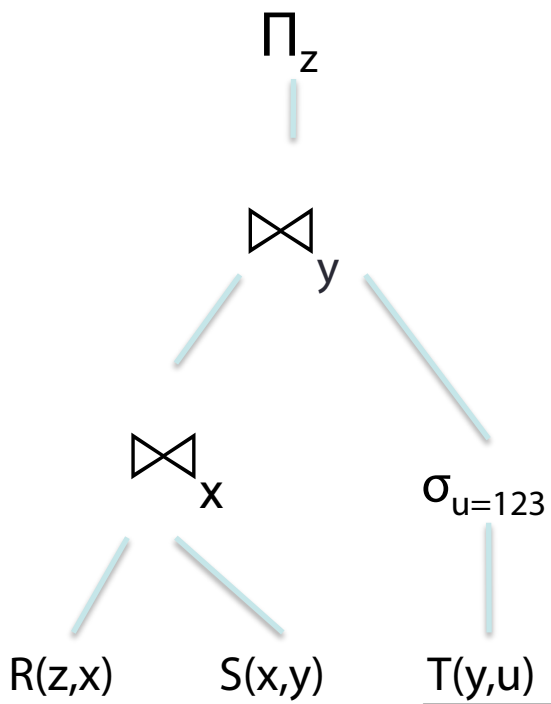
```
SELECT DISTINCT R.z
FROM R, S, T
WHERE R.x = S.x
      and S.y=T.y
      and T.u = 123
```

$$Q(z) = R(z,x), S(x,y), T(y,u)$$

Wiederholung: Anfragebearbeitungspläne

```
SELECT DISTINCT R.z
FROM R, S, T
WHERE R.x = S.x
      and S.y=T.y
      and T.u = 123
```

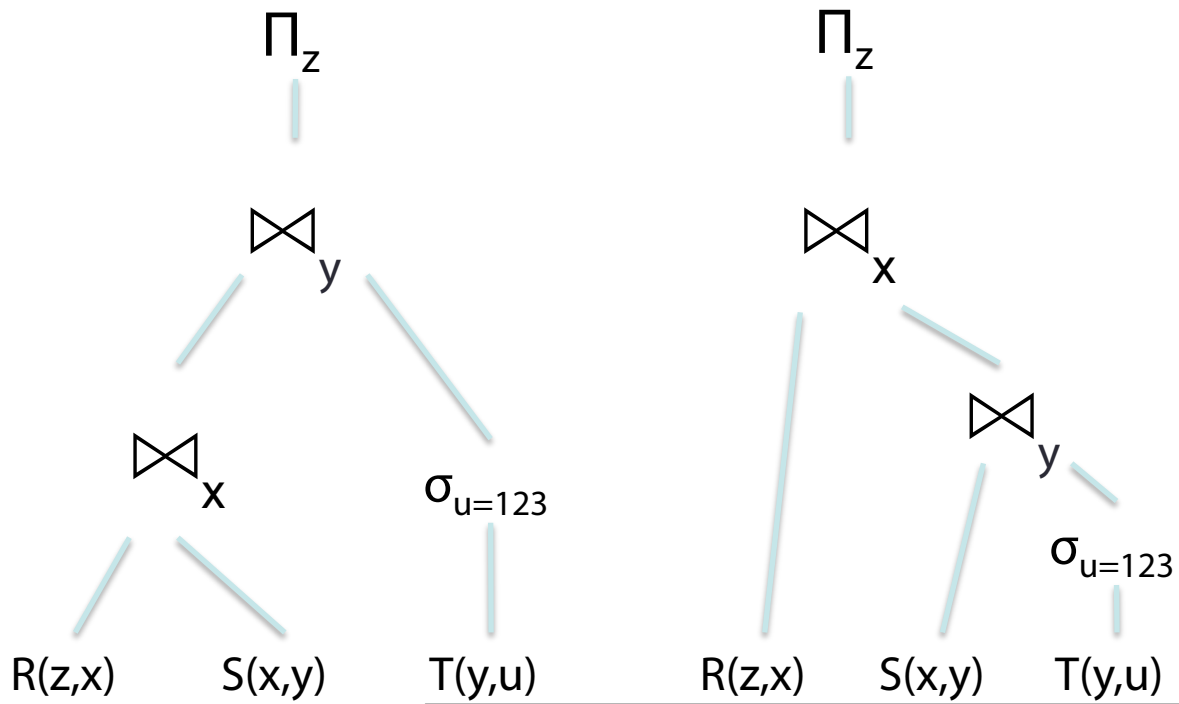
$Q(z) = R(z,x), S(x,y), T(y,u)$



Wiederholung: Anfragebearbeitungspläne

```
SELECT DISTINCT R.z
FROM R, S, T
WHERE R.x = S.x
      and S.y=T.y
      and T.u = 123
```

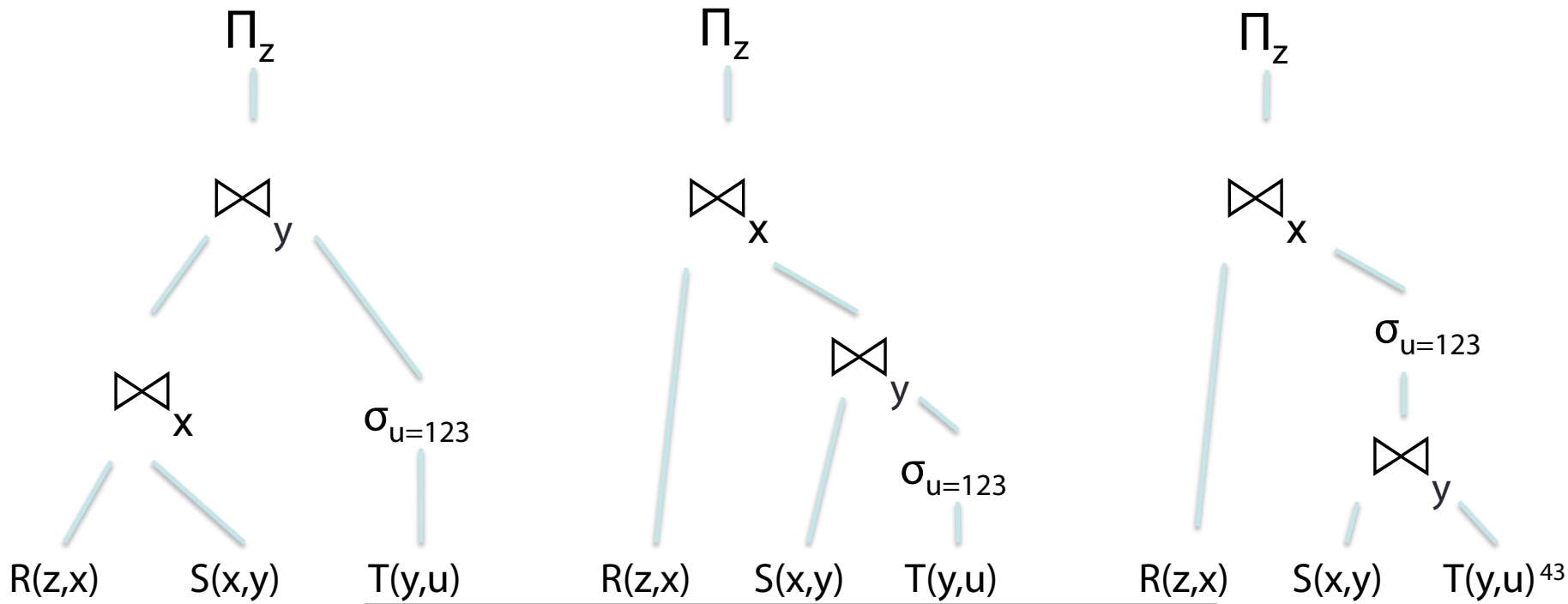
$Q(z) = R(z,x), S(x,y), T(y,u)$



Wiederholung: Anfragebearbeitungspläne

```
SELECT DISTINCT R.z
FROM R, S, T
WHERE R.x = S.x
      and S.y=T.y
      and T.u = 123
```

$Q(z) = R(z,x), S(x,y), T(y,u)$



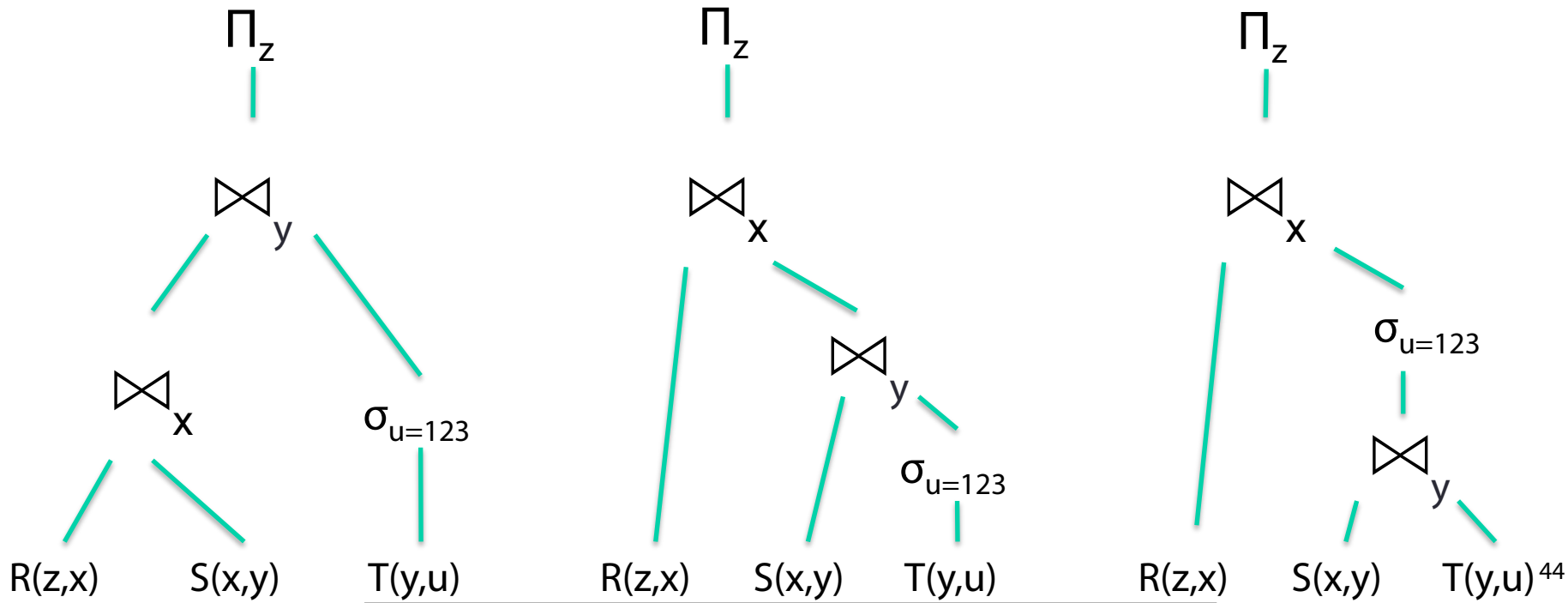
Wiederholung: Anfragebearbeitungspläne

```
SELECT DISTINCT R.z
FROM R, S, T
WHERE R.x = S.x
      and S.y=T.y
      and T.u = 123
```

$Q(z) = R(z,x), S(x,y), T(y,u)$

Diese Pläne sind äquivalent (liefern gleiche Ergebnisse)

Der Anfrageoptimierer wählt den Plan mit den geringsten Kosten



Extensionale Pläne

- Kernidee:
  - Modifiziere jeden Operator, sodass Wahrscheinlichkeiten für die Ausgabe berechnet werden

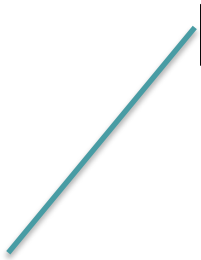
- Annahmen notwendig:
  - Ereignisse sind
    - unabhängig oder
    - disjunkt (exklusive)

Extensionale Operatoren

Independent join

A	B	P
a1	b1	p1*q1
a1	b2	p1*q2
a2	b3	p2*q3
a2	b4	p2*q4
a2	b5	p2*q5

i



i für independent

R(A)

A	P
a1	p1
a2	p2
a3	p3

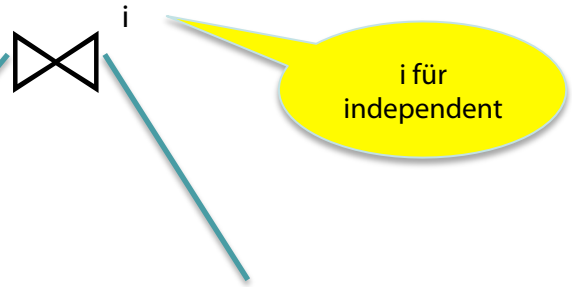
S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

Extensionale Operatoren

Independent join

A	B	P
a1	b1	p1*q1
a1	b2	p1*q2
a2	b3	p2*q3
a2	b4	p2*q4
a2	b5	p2*q5



R(A)

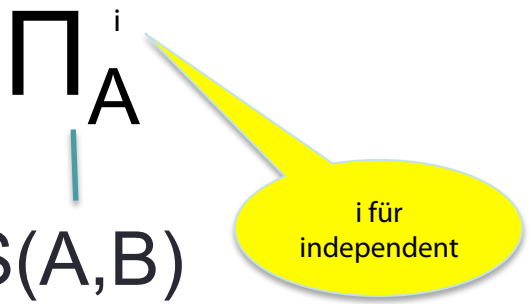
A	P
a1	p1
a2	p2
a3	p3

S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

Independent project

A	P
a1	1 - (1-q1)*(1-q2)
a2	1 - (1-q3)*(1-q4)*(1-q5)



S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

Extensionale Operatoren

Independent join

A	B	P
a1	b1	p1*q1
a1	b2	p1*q2
a2	b3	p2*q3
a2	b4	p2*q4
a2	b5	p2*q5

$\bowtie^{i}$

i für independent

R(A)

A	P
a1	p1
a2	p2
a3	p3

S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

Independent project

A	P
a1	1 - (1-q1)*(1-q2)
a2	1 - (1-q3)*(1-q4)*(1-q5)

Π_A^i

S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

i für independent

Selection

A	B	P
a2	b2	q3
a2	b3	q4
a2	b2	q5

$\sigma_{A=a2}$

S(A,B)

A	B	P
a1	b1	q1
a1	b1	q2
a2	b2	q3
a2	b3	q4
a2	b2	q5

```
SELECT DISTINCT 'true'
FROM R, S
WHERE R.x = S.x
```

Q() = R(x), S(x,y)

$P(Q) = 1 - [1-p_1*(1-(1-q_1)*(1-q_2))]$
$*[1- p_2*(1-(1-q_3)*(1-q_4)*(1-q_5))]$

Beispiel

R

x	P
a_1	p_1
a_2	p_2
a_3	p_3

S

x	y	P
a_1	b_1	q_1
a_1	b_2	q_2
a_2	b_3	q_3
a_2	b_4	q_4
a_2	b_5	q_5

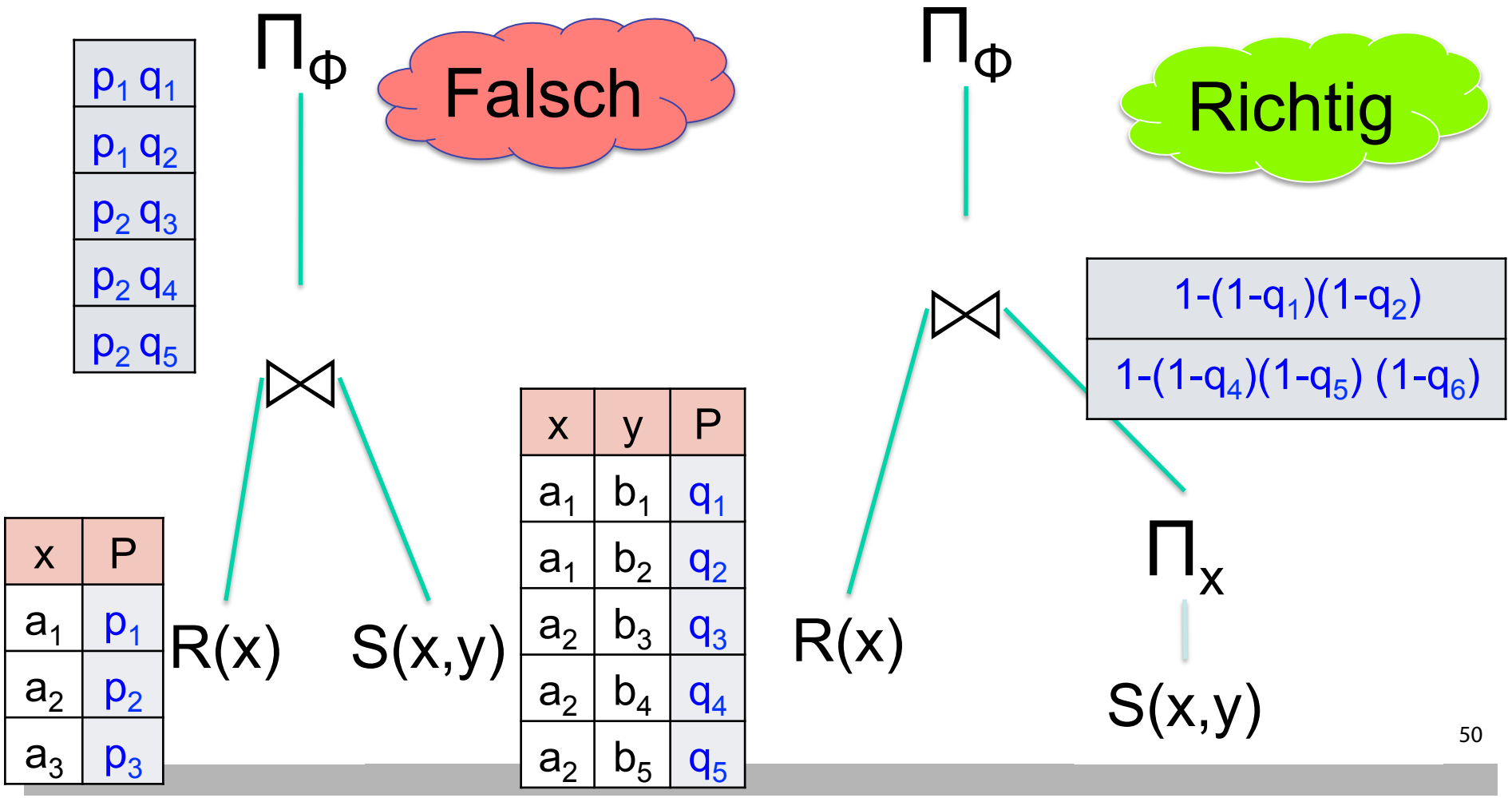
```
SELECT DISTINCT 'true'
FROM R, S
WHERE R.x = S.x
```

Q() = R(x), S(x,y)

$P(Q) = 1 - [1-p_1*(1-(1-q_1)*(1-q_2))]$
$*[1- p_2*(1-(1-q_3)*(1-q_4)*(1-q_5))]$

$1-(1-p_1q_1)(1-p_1q_2)(1-p_2q_3)(1-p_2q_4)(1-p_2q_5)$

$1-\{1-p_1[1-(1-q_1)(1-q_2)]\}*$
$\{1-p_2[1-(1-q_4)(1-q_5)\ (1-q_6)]\}$



Sichere Pläne

- Sei ein Schema für eine probabilist. DB gegeben
  - Relationen tupel-unabhängig oder BUD bei gegebenem Schlüssel

Definition: Ein Plan heißt *sicher*, wenn er die Wahrscheinlichkeiten für die Ausgabe richtig berechnet

- Anfrageoptimierung: Finde kostengünstigen aber sicheren Plan

Einsichten 1

- Äquivalente Pläne können unter Betrachtung der Wahrscheinlichkeiten inäquivalent werden
- Ein korrekter Plan wird sicher genannt
- Ziel: Finde sicheren Plan!
- Gibt es für jede Anfrage einen sicheren Plan?

Unsichere Anfragen

R

X	P
x1	p1
x2	p2

S

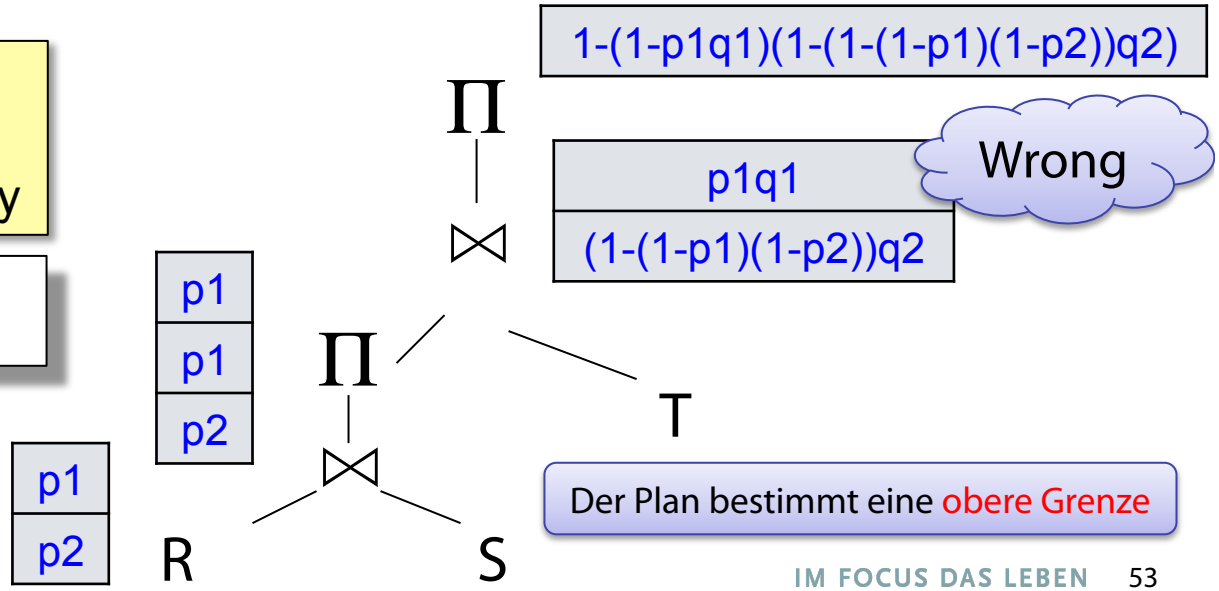
X	Y
x1	y1
x1	y2
x2	y2

T

Y	P
y1	q1
y2	q2

```
SELECT DISTINCT 'yes'
FROM R, S, T
WHERE R.x = S.x and S.y = T.y
```

H_0 :- R(x),S(x,y),T(y)



W Gatterbauer, D Suciu, Oblivious bounds on the probability of Boolean functions, ACM Transactions on Database Systems (TODS) 39 (1), 5, **2013**

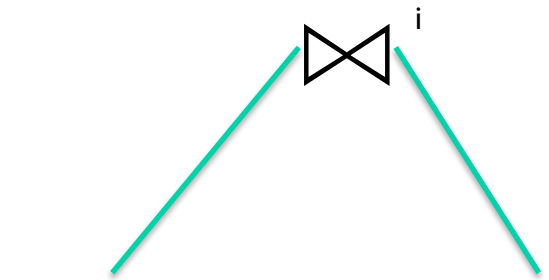
Diskussion

- Sjchere Anfragen haben einen sicheren Plan und können effektiv berechnet werden
- Für unsichere Anfragen kann kein sicherer Plan bestimmt werden, und es kann gezeigt werden, dass sich nicht effizient berechnet werden können
- Jeder extensionale Plan (sicher oder unsicher) kann direkt in SQL ausgedrückt werden – gezeigt am Beispiel von PostgreSQL
- Jede Anfrage (sicher oder unsicher) hat extensionale Anfragen, die obere und untere Grenzen der Wahrscheinlichkeiten berechnen.

Extensionale Pläne in PostgreSQL

A	B	P
a1	b1	p1*q1
a1	b2	p1*q2
a2	b3	p2*q3
a2	b4	p2*q4
a2	b5	p2*q5

```
SELECT R.A, S.B, R.P*S.P
FROM    R, S
WHERE R.A=S.A
```



R(A)

A	P
a1	p1
a2	p2
a3	p3

S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

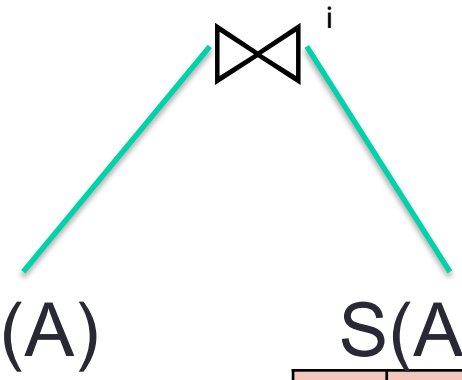
Extensional Plans in Postgres

A	B	P
a1	b1	p1*q1
a1	b2	p1*q2
a2	b3	p2*q3
a2	b4	p2*q4
a2	b5	p2*q5

```
SELECT R.A, S.B, R.P*S.P
FROM    R, S
WHERE R.A=S.A
```

```
SELECT S.A, 1.0-prod(1.0 - S.p)
FROM    S
GROUP BY S.A
```

A	P
a1	1 - (1-q1)*(1-q2)
a2	1 - (1-q3)*(1-q4)*(1-q5)



R(A)

A	P
a1	p1
a2	p2
a3	p3

S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

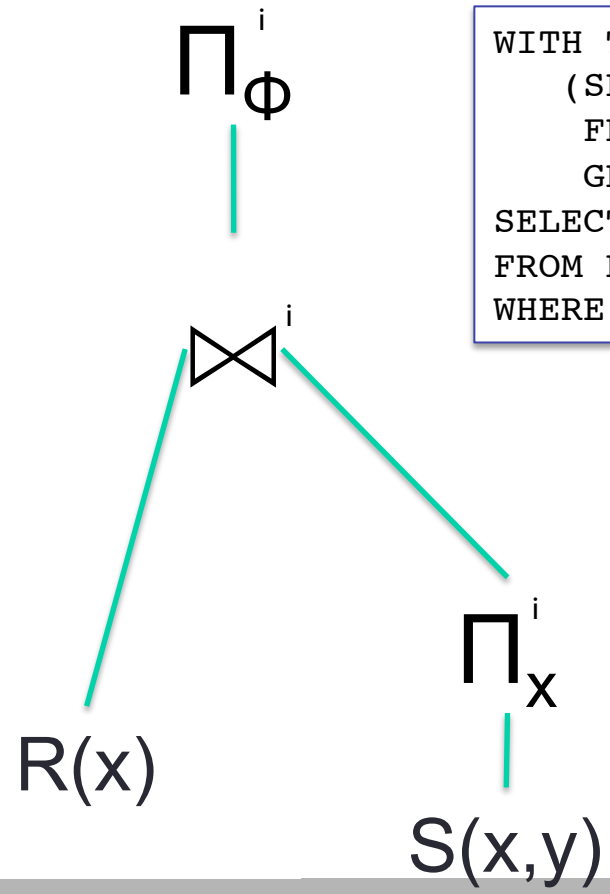
S(A,B)

A	B	P
a1	b1	q1
a1	b2	q2
a2	b3	q3
a2	b4	q4
a2	b5	q5

```
create or replace
  function combine_prod(float, float)
  returns float as
  'select $1 * $2' language SQL;
create or replace
  function final_prod(float)
  returns float as
  'select $1' language SQL;
drop aggregate if exists prod (float);
create aggregate prod(float)
(   sfunc = combine_prod,
    stype = float,
    finalfunc = final_prod,
    initcond = '1.0'
);
```

Extensional Plans in Postgres

```
SELECT DISTINCT 'true'
FROM R, S
WHERE R.x = S.x
```



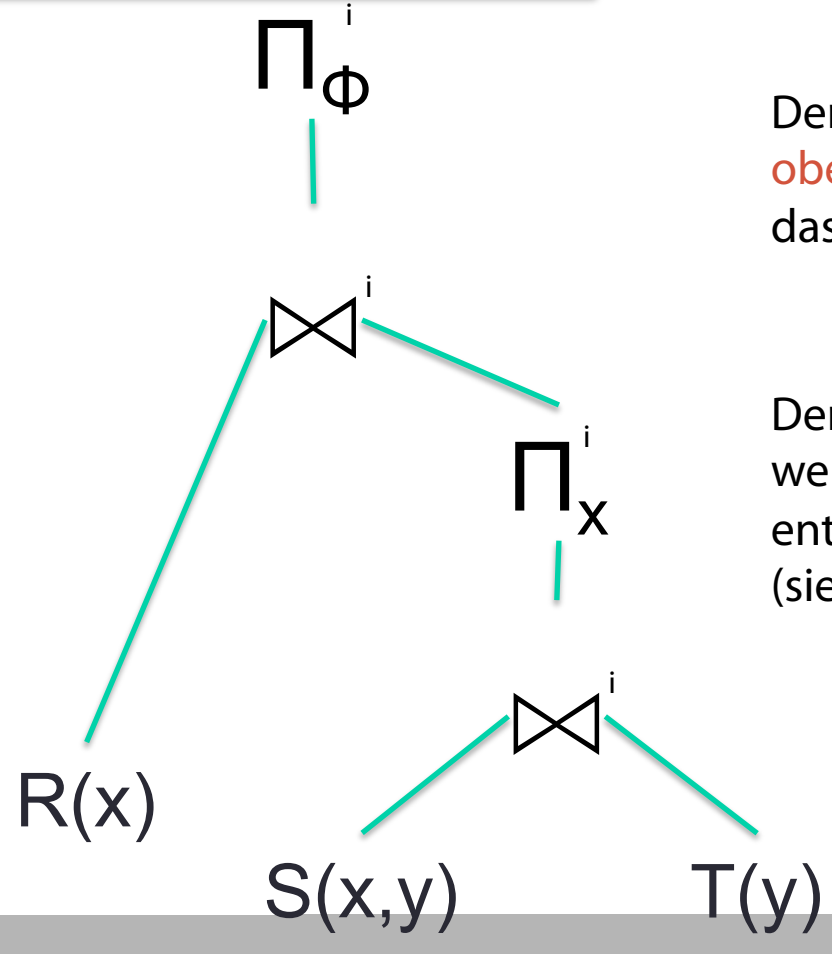

```
WITH Temp AS
   (SELECT S.x, 1.0-prod(1.0 - S.p) as p
    FROM S
    GROUP BY S.x)
SELECT 'true' as z, 1.0-prod(1.0 – R.P * Temp.P) as p
FROM R, Temp
WHERE R.x = Temp.x
```

Eingaben für PostgreSQL:

```
-----------------------------------------------------
-- First step: download postgres from http://www.postgresql.org/
-- Second step: run the command "createdb pdb"
-- Third step: run the command "psql pdb" then cut/paste commands below
-----------------------------------------------------
-- define an aggregate function to compute the product
create or replace function combine_prod (float, float) returns float as 'select $1 * $2' language SQL;
create or replace function final_prod (float) returns float as 'select $1' language SQL;
drop aggregate if exists prod (float);
create aggregate prod (float)
(  sfunc = combine_prod,
   stype = float,
   finalfunc = final_prod,
   initcond = '1.0'
);

-----------------------------------------------------
-- simple tables, similar to those used in the tutorial
create table R(z char(8), x char(8), p float);
create table S(x char(8), y char(8), p float);

insert into R values('c', 'a1', 0.5);
insert into R values('c', 'a2', 0.5);
insert into R values('c', 'a3', 0.5);

insert into S values('a1', 'b1', 0.5);
insert into S values('a1', 'b2', 0.5);
insert into S values('a2', 'b2', 0.5);
insert into S values('a2', 'b3', 0.5);
insert into S values('a2', 'b4', 0.5);

-- computing the query Q(z) = R(z,x),S(x,y)
-- a safe plan:
with Temp as
  (select S.x, 1.0-prod(1.0-p) as p
   from S
   group by S.x)
select R.z, 1.0-prod(1-R.p*Temp.p)
from R, Temp
where R.x=Temp.x
group by R.z;

-- an unsafe plan; guaranteed to return an upper bound on the probability
select R.z, 1.0-prod(1-R.p*S.p)
from R, S
where R.x=S.x
group by R.z;
```

Extensionale Pläne in PostgreSQL

```
SELECT DISTINCT 'yes'
FROM R, S, T
WHERE R.x = S.x and S.y = T.y
```



Diese Anfrage ist unsicher.

Der Plan ist unsicher, aber er liefert eine obere Grenze für den Wahrscheinlichkeitswert, dass die Anfrage mit 'yes' beantwortet wird.

Der Plan generiert eine untere Grenze, wenn die Wahrscheinlichkeiten in T entsprechend angepasst werden (siehe Literatur)

W Gatterbauer, D Suciu, Oblivious bounds on the probability of Boolean functions, ACM Transactions on Database Systems (TODS) 39 (1), 5, **2013**

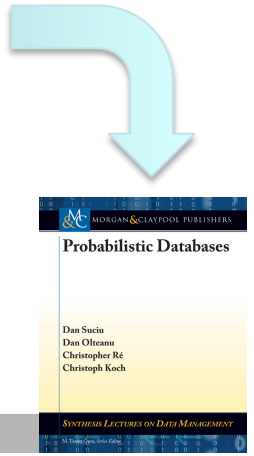
Eingaben für PostgreSQL:

```
-------------------------------------------------------
-- The following approximation plans for unsafe queries are from
-- Gatterbauer, Suciu: Oblivious Bounds on the Probability of Boolean Functions

-- create a third table
create table T(y char(8), p float);

insert into T values('b1', 0.5);
insert into T values('b2', 0.5);
insert into T values('b3', 0.5);
insert into T values('b4', 0.5);

-- computing the query Q(z) = R(z,x),S(x,y),T(y)
-- This query has no safe plans

-- Next two unsafe plans compute upper bounds on the probability:
-- Unsafe plan #1
with Temp as
    (select S.x, 1.0-prod(1.0-S.p*T.p) as p
     from S,T
     where S.y=T.y
     group by S.x)
select R.z, 1.0-prod(1-R.p*Temp.p)
from R, Temp
where R.x=Temp.x
group by R.z;

-- Unsafe plan #2
with Temp as
   (select R.z,S.y,1.0-prod(1.0-R.p*S.p) as p
    from R,S
    where R.x=S.x
    group by R.z,S.y)
select Temp.z, 1.0-prod(1-Temp.p*T.p)
from Temp, T
where Temp.y=T.y
group by Temp.z;
```

```
-- Next two unsafe plans compute lower bounds on the probability:
with newT as
  (select T.y, 1-exp((ln(1-T.p))/count(*)) as p
   from S,T
   where S.y=T.y
   group by T.y, T.p),
Temp as
    (select S.x, 1.0-prod(1.0-S.p*newT.p) as p
     from S,newT
     where S.y=newT.y
     group by S.x)
select R.z, 1.0-prod(1-R.p*Temp.p)
from R, Temp
where R.x=Temp.x
group by R.z;

with newR as
  (select R.z, R.x, 1-exp((ln(1-R.p))/count(*)) as p
   from R,S
   where R.x=S.x
   group by R.z,R.x,R.p),
Temp as
    (select newR.z, S.y, 1.0-prod(1.0-newR.p*S.p) as p
     from newR, S
     where newR.x=S.x
     group by newR.z, S.y)
select Temp.z, 1.0-prod(1-Temp.p*T.p)
from Temp, T
where Temp.y=T.y
group by Temp.z;
```

Einsichten 2

- Man benötigt kein neues probabilistisches DB-System für eine probabilistische Datenbasis!
- Was man benötigt, sind SQL-Kenntnisse und Kenntnisse in Wahrscheinlichkeitstheorie
- Im Buch über probabilistische Datenbanken steht, wie's geht!



Überblick

- Wiederholung: Unions of Conjunctive Queries, UCQ (Vereinigung von Selektion, Projektion, Verbund)

- Vier Regeln zur Bestimmung von sicheren Anfragen

Wiederholung: Unions of Conjunctive Queries

Owners of items in either "Office444" or "Hall7":

Atom

$$Q(z) = \exists x_1 \exists t_1 \, (\text{Owner}(z,x_1) \wedge \text{Location}(x_1,t_1,\text{"Office444"})) \vee \exists x_2 \exists t_2 \, (\text{Owner}(z,x_2) \wedge \text{Location}(x_2,t_2,\text{"Hall7"}))$$

Ohne Quantoren:

$$Q(z) = \text{Owner}(z,x_1),\text{Location}(x_1,t_1,\text{"Office444"}) \vee \text{Owner}(z,x_2),\text{Location}(x_2,t_2,\text{"Hall7"})$$

Wiederholung: Unions of Conjunctive Queries

Owners of items in either "Office444" or "Hall7":

$$Q(z) = \exists x_1 \exists t_1 \, (\text{Owner}(z,x_1) \wedge \text{Location}(x_1,t_1,\text{"Office444"})) \vee \exists x_2 \exists t_2 \, (\text{Owner}(z,x_2) \wedge \text{Location}(x_2,t_2,\text{"Hall7"}))$$

Ohne Quantoren:

Union of conjunctive queries

$$Q(z) = \text{Owner}(z,x_1),\text{Location}(x_1,t_1,\text{"Office444"}) \vee \text{Owner}(z,x_2),\text{Location}(x_2,t_2,\text{"Hall7"})$$

Wiederholung: Unions of Conjunctive Queries

Owners of items in either "Office444" or "Hall7":

$$Q(z) = \exists x_1 \exists t_1 \, (\text{Owner}(z,x_1) \wedge \text{Location}(x_1,t_1,\text{"Office444"})) \vee \exists x_2 \exists t_2 \, (\text{Owner}(z,x_2) \wedge \text{Location}(x_2,t_2,\text{"Hall7"}))$$

Ohne Quantoren:

Union of conjunctive queries

$$Q(z) = \text{Owner}(z,x_1),\text{Location}(x_1,t_1,\text{"Office444"}) \vee \text{Owner}(z,x_2),\text{Location}(x_2,t_2,\text{"Hall7"})$$

Nach Umformung:

$$Q(z) = \text{Owner}(z,x) \wedge \exists t \, [\text{Location}(x,t,\text{"Office444"}) \vee \text{Location}(x,t,\text{"Hall7"})]$$

Wiederholung: Unions of Conjunctive Queries

Owners of items in either "Office444" or "Hall7":

$$Q(z) = \exists x_1 \exists t_1 \, (\text{Owner}(z,x_1) \wedge \text{Location}(x_1,t_1,\text{"Office444"})) \vee \exists x_2 \exists t_2 \, (\text{Owner}(z,x_2) \wedge \text{Location}(x_2,t_2,\text{"Hall7"}))$$

Ohne Quantoren:

Union of conjunctive queries

$$Q(z) = \text{Owner}(z,x_1),\text{Location}(x_1,t_1,\text{"Office444"}) \vee \text{Owner}(z,x_2),\text{Location}(x_2,t_2,\text{"Hall7"})$$

Nach Umformung:

$$Q(z) = \text{Owner}(z,x) \wedge \exists t \, [\text{Location}(x,t,\text{"Office444"}) \vee \text{Location}(x,t,\text{"Hall7"})]$$

Unter Verwendung von:

1. Distributivegesetz für $\vee$, $\wedge$
2. Kommutativegesetz für $\exists$, $\vee$: $(\exists x \, P(x)) \vee (\exists y \, T(y)) = \exists z \, (P(z) \vee T(z))$

Vier Regeln, um sichere Anfragen zu erzeugen

- Independent join
- Independent project
- Independent union
- Inclusion/exclusion

Wir beschränken uns auf Boolesche Anfragen.

Regel 1: Independent Join

$$P(Q1 \wedge Q2) = P(Q1)P(Q2)$$

Wenn Q1 und Q2 unabhängig sind
(also keine gemeinsamen Atome haben)

Regel 1: Independent Join

$$P(Q1 \wedge Q2) = P(Q1)P(Q2)$$

Wenn Q1 und Q2 unabhängig sind
(also keine gemeinsamen Atome haben)

Regel 2: Independent Project

$$P(\exists z\, Q) = 1 - \prod_{a \in \text{Domain}} (1 - P(Q[a/z]))$$

Wenn z eine "Separatorvariable" in Q
ist, also für Konstanten a,b, Q[a/z]
und Q[b/z] unabhängig sind

Regel 1: Independent Join

$$P(Q1 \wedge Q2) = P(Q1)P(Q2)$$

Wenn Q1 und Q2 unabhängig sind
(also keine gemeinsamen Atome haben)

Regel 2: Independent Project

$$P(\exists z\, Q) = 1 - \prod_{a \in Domain} (1 - P(Q[a/z]))$$

Wenn z eine "Separatorvariable" in Q
ist, also für Konstanten a,b, Q[a/z]
und Q[b/z] unabhängig sind

Regel 3: Independent Union

$$P(Q1 \vee Q2) = 1 - (1 - P(Q1))(1 - P(Q2))$$

Wenn Q1 und Q2 unabhängig sind
(also keine gemeinsamen Atome haben)

Beispiel

$$Q_U = R(x_1),S(x_1,y_1) \vee T(x_2),S(x_2,y_2)$$

$$= \exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1) \vee \exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)$$

Beispiel

$$Q_U = R(x_1),S(x_1,y_1) \vee T(x_2),S(x_2,y_2)$$

$$= \exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1) \vee \exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)$$

$$Q_U = \exists z\,[R(z) \wedge S(z,y_1) \vee T(z) \wedge S(z,y_2)]$$

Commute $\exists$ with $\vee$

Beispiel

$$Q_U = R(x_1),S(x_1,y_1) \vee T(x_2),S(x_2,y_2)$$

$$= \exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1) \vee \exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)$$

$$Q_U = \exists z\,[R(z) \wedge S(z,y_1) \vee T(z) \wedge S(z,y_2)]$$

Kommutiere $\exists$ mit $\vee$

$$P(Q_U) = 1 - \Pi_{a \in \text{Domain}} (1 - P[R(a) \wedge \underline{S(a,y_1)} \vee T(a) \wedge \underline{S(a,y_2)})]$$

Independent project: Für $a \neq b$, sind $Q_U[a/z]$ und $Q_U[b/z]$ unabhängig weil die Atome $R(a),S(a,y_1),T(a),S(a,y_2)$ disjunkt sind von $R(b),S(b,y_1),T(b),S(b,y_2)$

Beispiel

$Q_U = R(x_1),S(x_1,y_1) \vee T(x_2),S(x_2,y_2)$

$= \exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1) \vee \exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)$

$Q_U = \exists z\,[R(z) \wedge S(z,y_1) \vee T(z) \wedge S(z,y_2)]$

Kommutiere $\exists$ mit $\vee$

$P(Q_U) = 1 - \Pi_{a \in \text{Domain}} (1 - P[R(a) \wedge \underline{S(a,y_1)} \vee T(a) \wedge \underline{S(a,y_2)})]$

Independent project: Für $a \neq b$, sind $Q_U[a/z]$ und $Q_U[b/z]$ unabhängig weil die Atome $R(a),S(a,y_1),T(a),S(a,y_2)$ disjunkt sind von $R(b),S(b,y_1),T(b),S(b,y_2)$

$P(Q_U) = 1 - \Pi_{a \in \text{Domain}} (1 - P[\underline{(R(a) \vee T(a))} \wedge \underline{\exists y.\, S(a,y)}]$

Distribution $\wedge$ über $\vee$

Beispiel

$Q_U = R(x_1),S(x_1,y_1) \vee T(x_2),S(x_2,y_2)$

$= \exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1) \vee \exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)$

$Q_U = \exists z\, [R(z) \wedge S(z,y_1) \vee T(z) \wedge S(z,y_2)]$

Kommutiere $\exists$ mit $\vee$

$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - P[R(a) \wedge \underline{S(a,y_1)} \vee T(a) \wedge \underline{S(a,y_2)})]$

Independent project: Für $a \neq b$, sind $Q_U[a/z]$ und $Q_U[b/z]$ unabhängig weil die Atome $R(a),S(a,y_1),T(a),S(a,y_2)$ disjunkt sind von $R(b),S(b,y_1),T(b),S(b,y_2)$

$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - P[\underline{(R(a) \vee T(a))} \wedge \underline{\exists y.\, S(a,y)}]$

Distribution $\wedge$ über $\vee$

$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - P[R(a) \vee T(a)]\, P[\exists y.\, S(a,y)]$

Independent join

Beispiel

$Q_U = R(x_1),S(x_1,y_1) \vee T(x_2),S(x_2,y_2)$

$= \exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1) \vee \exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)$

$Q_U = \exists z\,[R(z) \wedge S(z,y_1) \vee T(z) \wedge S(z,y_2)]$

Kommutiere $\exists$ mit $\vee$

$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - P[R(a) \wedge \underline{S(a,y_1)} \vee T(a) \wedge \underline{S(a,y_2)})]$

Independent project: Für $a \neq b$, sind $Q_U[a/z]$ und $Q_U[b/z]$ unabhängig weil die Atome $R(a),S(a,y_1),T(a),S(a,y_2)$ disjunkt sind von $R(b),S(b,y_1),T(b),S(b,y_2)$

$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - P[\underline{(R(a) \vee T(a))} \wedge \underline{\exists y.\, S(a,y)}]$

Distribution $\wedge$ über $\vee$

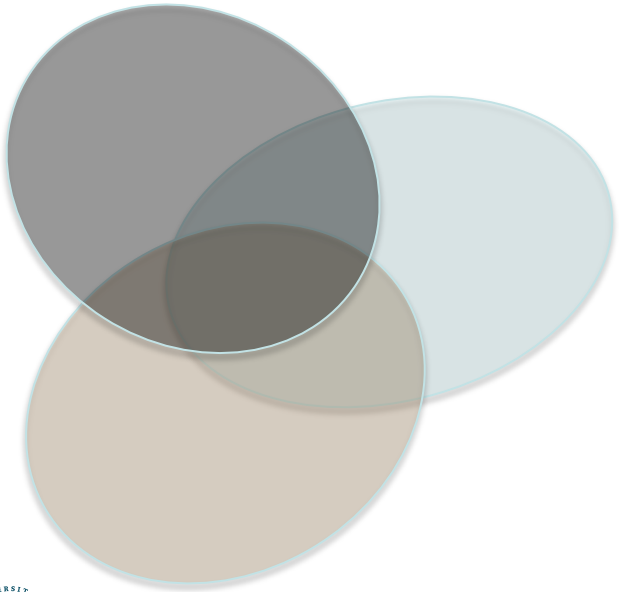
$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - P[R(a) \vee T(a)]\, P[\exists y.\, S(a,y)]$

Independent join

$P(Q_U) = 1 - \Pi_{a \in Domain} (1 - (1-(1-P[R(a)])(1-P[T(a)]))\, (1-\Pi_{b \in Domain} (1 - P[S(a,b)])))$

Regel 4: Inclusion-Exclusion

$$P(Q1 \wedge Q2 \wedge Q3) = P(Q1) + P(Q2) + P(Q3) - P(Q1 \vee Q2) - P(Q1 \vee Q3) - P(Q2 \vee Q3) + P(Q1 \vee Q2 \vee Q3)$$



NB: Dieses ist dual zur häufiger verwendeten Formel:

$$P(Q1 \vee Q2 \vee Q3) = P(Q1) + P(Q2) + P(Q3) - P(Q1 \wedge Q2) - P(Q1 \wedge Q3) - P(Q2 \wedge Q3) + P(Q1 \wedge Q2 \wedge Q3)$$

Beispiel

$$Q_J = R(x_1), S(x_1, y_1), T(x_2), S(x_2, y_2)$$

$$= [\exists x_1 \exists y_1 R(x_1) \wedge S(x_1, y_1)] \wedge [\exists x_2 \exists y_2 T(x_2) \wedge S(x_2, y_2)]$$

Beispiel

$Q_J = R(x_1), S(x_1,y_1), T(x_2), S(x_2,y_2)$ $= [\exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1)] \wedge [\exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)]$

$Q_J = Q_1 \wedge Q_2$ wobei $Q_1 = R(x_1), S(x_1,y_1)$ $Q_2 = T(x_2), S(x_2,y_2)$

Beispiel

$Q_J = R(x_1), S(x_1,y_1), T(x_2), S(x_2,y_2)$ $= [\exists x_1 \exists y_1 R(x_1) \wedge S(x_1,y_1)] \wedge [\exists x_2 \exists y_2 T(x_2) \wedge S(x_2,y_2)]$

$Q_J = Q_1 \wedge Q_2$ wobei $Q_1 = R(x_1), S(x_1,y_1)$ $Q_2 = T(x_2), S(x_2,y_2)$

$$P(Q_J) = P(Q_1) + P(Q_2) - P(Q_1 \vee Q_2)$$

Q_1 = eine hierarchische CQ ohne Self-Joins

Q_2 = dito

$Q_1 \vee Q_2 = Q_U$, siehe vorige Folien

Einsicht 3

Vereinigung (union) für Self-Joins!

- Conjunctive Queries = Keine “natürliche” Klassen von Anfragen von Probabilistische DBs
- Unions of Conjunctive Queries = die “natürliche” Klasse von Anfragen

Non-Standard-Datenbanken

Probabilistische Datenbanken