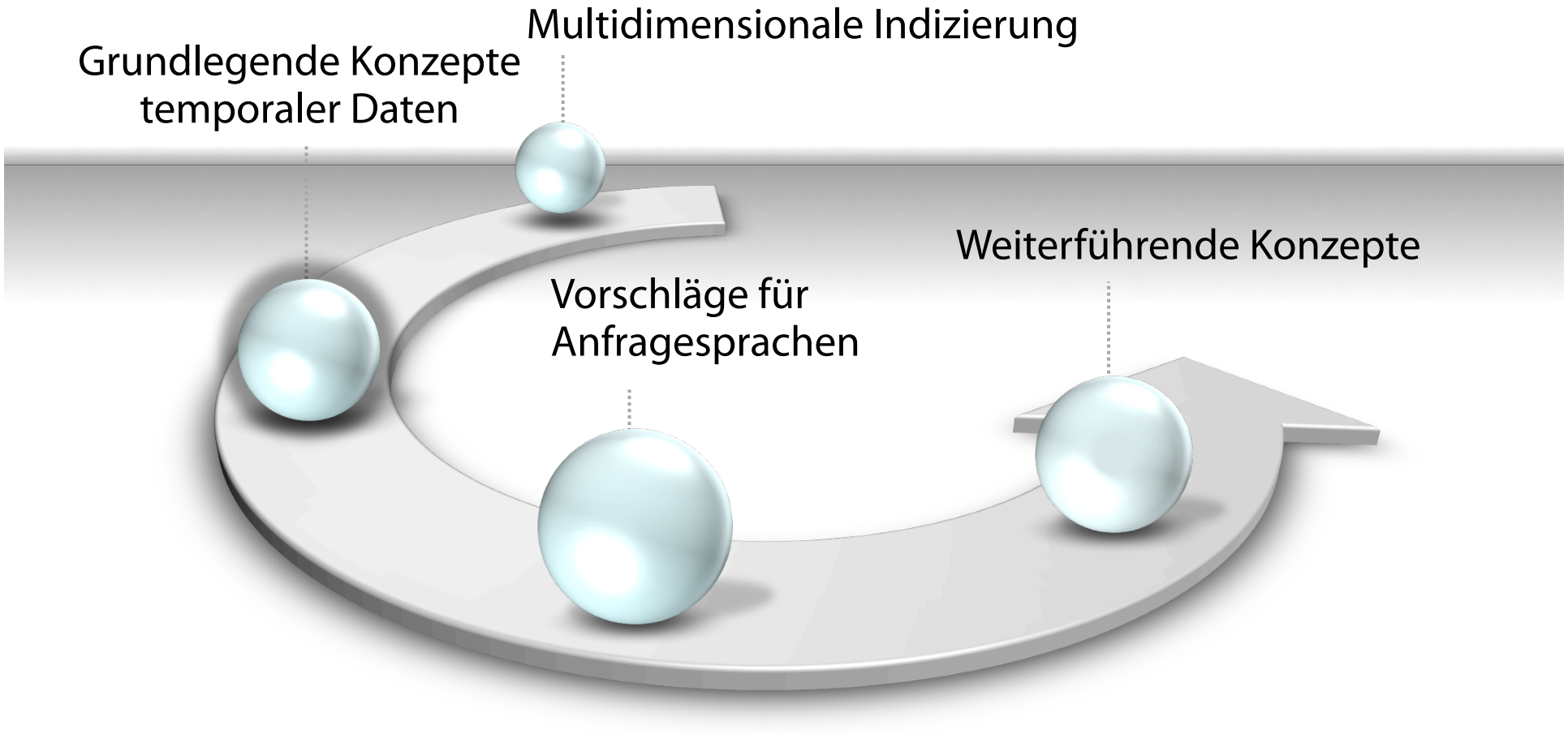
Non-Standard-Datenbanken

Temporale Daten und das relationale Modell

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

Non-Standard-Datenbanken

Von der multidimensionalen Indizierung zu temporalen Daten



Motivation: Ein klinisches Szenario

Ms. Jones was seen in the diabetes clinic on January 14 1997 at 11 A.M. Her blood-glucose value at that time was measured in the clinic as 220 mg/ 100ml. She complained of vomiting and dizziness for the past 2 or 3 days.

She was eventually hospitalized on the same day. A more accurate blood-glucose test that was taken at the same time as the one performed in the clinic returned from the laboratory on January 15 1997, with a value of 380 mg/100ml.

Ms. Jones was discharged on January 17 1997, and was seen again in the clinic on January 24 1997. At that time, several renal-function serum and urine tests were performed in addition to measuring blood-glucose values. A complete neurological assessment was carried out as well.

Entnommen aus: Temporal Databases and Maintenance of Time-Oriented Clinical Data, Yuval Shahar, M.D., Ph.D.

Standard-Datenbanken

"Suppliers and Shipments"

S

S#
S1
S2
S3
S4
S5

Prädikat
"Supplier S# is under contract"

SP

S#	P#
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

Prädikat
"Supplier S# is able to supply part P#"

Anfragen:

- *Which suppliers can supply something?*
- *Which suppliers cannot supply anything?*

Semitemporalisiert: Linear geordnete Struktur

S_SINCE

Prädikat:
"Supplier S# has been under contract since day SINCE"

S#	SINCE
S1	*d04*
S2	*d07*
S3	*d03*
S4	*d04*
S5	*d02*

SP_SINCE

Prädikat:
"Supplier S# has been able to supply part P# since day SINCE"

S#	P#	SINCE
S1	P1	*d04*
S1	P2	*d05*
S1	P3	*d09*
S1	P4	*d05*
S1	P5	*d04*
S1	P6	*d06*
S2	P1	*d08*
S2	P2	*d09*
S3	P2	*d08*
S4	P2	*d06*
S4	P4	*d04*
S4	P5	*d05*

- **Zeitstruktur** (T, <) wobei T eine Menge und < eine totale Ordnung über T ist
- Elemente von T auch **Chronons** genannt
- **Granularität** ist **anwendungsabhängig** zu wählen

Anfragen:

- *Since when has supplier S# been able to supply anything?* (einfach)
- *Since when has supplier S# been unable to supply anything?* (unmöglich)

Voll temporalisiert (Versuch 1)

S_FROM_TO

Predicate:
"Supplier S# was under contract from day FROM to day TO."

S#	FROM	TO
S1	*d04*	*d10*
S2	*d02*	*d04*
S2	*d07*	*d10*
S3	*d03*	*d10*
S4	*d04*	*d10*
S5	*d02*	*d10*

SP_FROM_TO

Predicate:
"Supplier S# was able to supply part P# from day FROM to day TO."

S#	P#	FROM	TO
S1	P1	*d04*	*d10*
S1	P2	*d05*	*d10*
S1	P3	*d09*	*d10*
S1	P4	*d05*	*d10*
S1	P5	*d04*	*d10*
S1	P6	*d06*	*d10*
S2	P1	*d08*	*d10*
S2	P1	*d02*	*d04*
S2	P2	*d08*	*d10*
S2	P2	*d03*	*d03*
S3	P2	*d09*	*d10*
S4	P2	*d06*	*d09*
S4	P4	*d04*	*d08*
S4	P5	*d05*	*d10*

Anfragen:

- *What does supplier S1 supply at day d6? (leicht)*
- *Does S1 supply something between d2 and d11? (leicht)*
- *During which times was supplier S# able to supply something?* (schwierig)
- *During which times was supplier S# unable to supply something?* (schwierig)

Zu prüfende Einschränkungen

S_FROM_TO

Ein Lieferant soll nicht einen Vertrag in unterschiedlichen aber überlappenden oder aneinandergrenzenden Intervallen haben

S#	FROM	TO
S1	*d04*	*d10*
S2	*d02*	*d04*
S2	*d07*	*d10*
S3	*d03*	*d10*
S4	*d04*	*d10*
S5	*d02*	*d10*

SP_FROM_TO

Ein Lieferant soll nicht die gleichen Dinge in überlappenden oder aneinanderstoßenden Intervallen liefern

S#	P#	FROM	TO
S1	P1	*d04*	*d10*
S1	P2	*d05*	*d10*
S1	P3	*d09*	*d10*
S1	P4	*d05*	*d10*
S1	P5	*d04*	*d10*
S1	P6	*d06*	*d10*
S2	P1	*d08*	*d10*
S2	P1	*d02*	*d04*
S2	P2	*d08*	*d10*
S2	P2	*d03*	*d03*
S3	P2	*d09*	*d10*
S4	P2	*d06*	*d09*
S4	P4	*d04*	*d08*
S4	P5	*d05*	*d10*

Einschränkungen sind nicht einfach repräsentierbar

Wie kann für **zeitlich begrenzt gültige Einträge** ein **Schlüssel** definiert werden? Will man das?

Interpretation? Sind die FROM und TO Daten enthalten? Z.B.: Hatte Lieferant S1 einen Vertrag am Tag 10?

Voll temporalisiert (Versuch 2)

S_DURING

S#	DURING
S1	[*d04*:*d10*]
S2	[*d02*:*d04*]
S2	[*d07*:*d10*]
S3	[*d03*:*d10*]
S4	[*d04*:*d10*]
S5	[*d02*:*d10*]

Einführung von ***Intervalltypen***[1] und ihren ***Punkttypen***

SP_DURING

S#	P#	DURING
S1	P1	[*d04*:*d10*]
S1	P2	[*d05*:*d10*]
S1	P3	[*d09*:*d10*]
S1	P4	[*d05*:*d10*]
S1	P5	[*d04*:*d10*]
S1	P6	[*d06*:*d10*]
S2	P1	[*d08*:*d10*]
S2	P1	[*d02*:*d04*]
S2	P2	[*d08*:*d10*]
S2	P2	[*d03*:*d03*]
S3	P2	[*d09*:*d10*]
S4	P2	[*d06*:*d09*]
S4	P4	[*d04*:*d08*]
S4	P5	[*d05*:*d10*]

Der Typ des Attributs DURING könnte sein:
INTERVAL_DATE (mit Punkttypen **DATE**)

DURING als Schlüssel? Möglich, aber hier nicht sinnvoll

Für einen Punkttypen ist eine Nachfolgerfunktion definiert – hier: **NEXT_DATE (*d*)**. Hier zeigt sich die Skalierung.

[1]Nicht zu verwechseln mit dem Typ INTERVAL in SQL-92

Intervallspezifikationen

- [a, b] – Anfangszeitpunkt a und Endzeitpunkt b enthalten
- (a, b] – Anfangszeitpunkt a nicht enthalten,
  Endzeitpunkt b enthalten
- [a, b) – Anfangszeitpunkt a enthalten,
  Endzeitpunkt b nicht enthalten
- (a, b) – Anfangszeitpunkt a nicht enthalten,
  Endzeitpunkt b nicht enthalten,
  d.h. nur die Chronons dazwischen

PRE (*i*)	Offener Anfang
BEGIN (*i*)	Geschlossener Anfang
END (*i*)	Geschlossenes Ende
POST (*i*)	Offenes Ende
COUNT (*i*)	Länge (Anzahl der Punkte)

Intervalle werden oft auch Perioden (periods) genannt.

Intervalle: Mehrere Deutungen

- Etwas gilt zu irgendeinem Zeitpunkt im Intervall (Unsicherheit)
- Etwas gilt zu jedem Zeitpunkt des Intervalls
- Etwas wurde als Wert über (alle) Zeitpunkte im Intervall bestimmt?
  - Beispiel: Inflation – hier DURING als Schlüssel!
- Intervall definiert Periode in der höheren Skala
- ... weitere ...

DURING	PERCENTAGE
[*m01*:*m03*]	18
[*m04*:*m06*]	20
[*m07*:*m09*]	20
[*m07*:*m07*]	25
.....	..
[*m01*:*m12*]	20

Einschränkung: keine „Lücken“

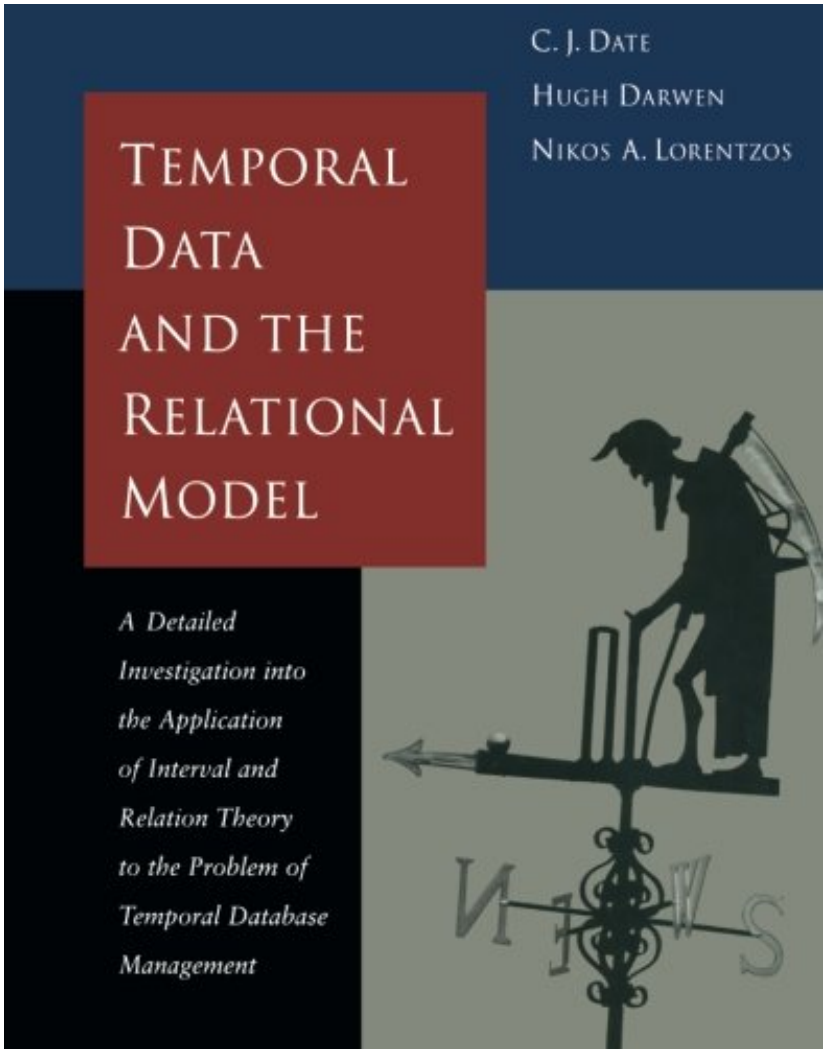
Nicht einfach zu prüfen

Automatische Verschmelzung von Intervallen?

DURING	PRESIDENT
[1974 : 1976]	Ford
[1977 : 1980]	Carter
[1981 : 1984]	Reagan
[1985 : 1988]	Reagan
[1993 : 1996]	Clinton
[1997 : 2000]	Clinton
[2009 : 2012]	Obama
[2013 : 2016]	Obama

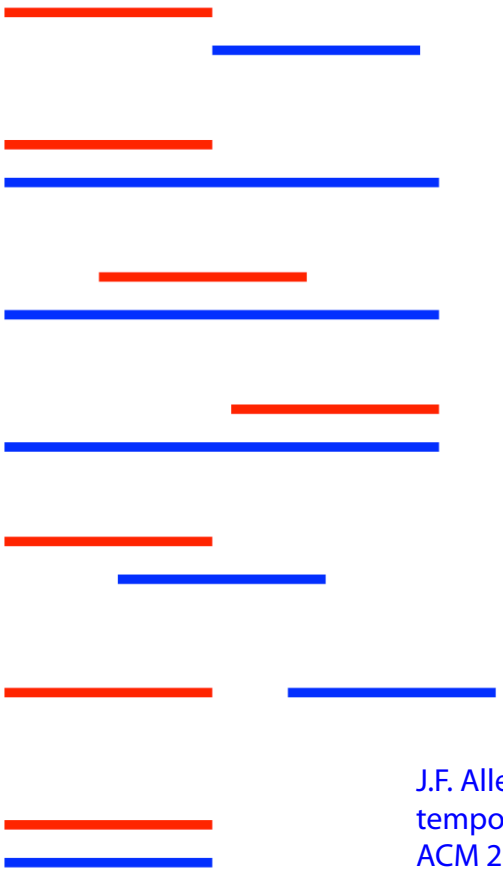
Danksagung und Literaturhinweis

Die vorigen 9 Präsentationen
sind in Anlehnung an Präsentationen
von Hugh Darwen zu folgendem Buch
gestaltet.



Operators on Time Periods

- Membership test of point in period
- Union, intersection, minus, count
  - union, intersection are not defined for all periods! Why?
- Comparison predicates: Allen's operators (are not symmetric!)
  - p1 meets p2
  - p1 starts p2
  - p1 during p2
  - p1 finishes p2
  - p1 overlaps p2
  - p1 before p2
  - p1 = p2



J.F. Allen, "Maintaining knowledge about temporal intervals", Communications of ACM 26(11), November **1983**

Dimensionen der Zeit

- Gültigkeit aus Sicht der Anwendung
  (Gültigkeitszeit – valid time, Anwendungszeit, ...)
- Zeitpunkt der Eintragung eines Tupels in die Datenbank
  „Bekannt“ aus Sicht des Systems
  Zeitpunkt, an dem ein neuer Wert eingetragen wird
- (Transaktionszeit – transaction time, Systemzeit, ...)
- Bitemporal: Beides gleichzeitig repräsentiert

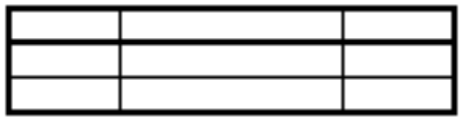
- Auch möglich: benutzerdefinierte Zeitangaben

- Seit langem untersuchtes Gebiet im Datenbankbereich[1]

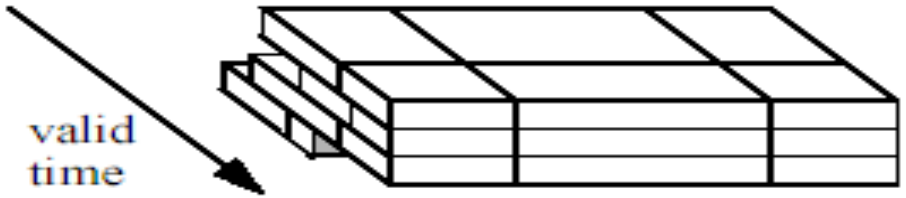
[1]R. Snodgrass, The temporal query language TQuel, ACM Transactions on Database Systems (TODS) TODS, 12:2, pp. 247-298, **1987**

Relationen in zeitlichen Dimensionen

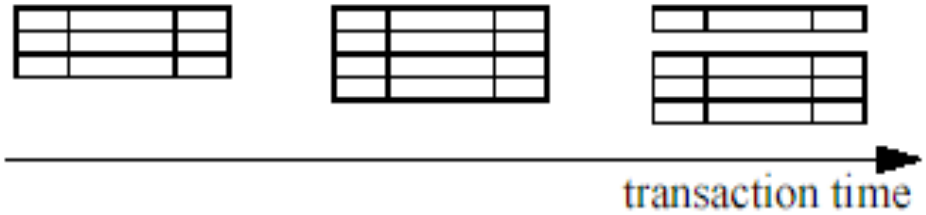
- Schnappschuss



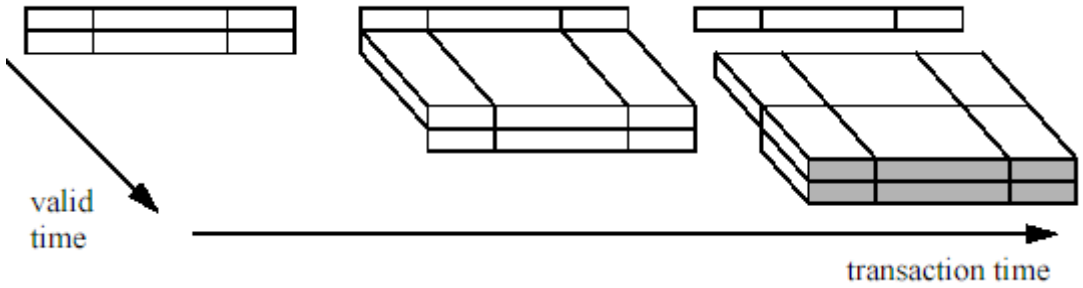
- Gültigkeitszeiten



- Transaktionszeiten



- Bitemporale Relation



Bitemporale Daten

Emp	Dept	Valid Time	Transaction Time
Jake	Shipping	[1995-06-10 - 1995-06-16)	[1995-06-05 - 1995-06-10)
Jake	Shipping	[1995-06-05 - 1995-06-21)	[1995-06-10 - 1995-06-15)
Jake	Shipping	[1995-06-10 - 1995-06-16)	[1995-06-15 - 1995-06-20)
Jake	Loading	[1995-06-10 - 1995-06-16)	[1995-06-20 - 9999-12-31)

- Beispiel: Anstellung von Jake
- Darstellung einer einzigen Anstellung, mit Änderungen in der modellierten Realität mit jeweils zugehörigem Änderungszeitpunkt

Temporale Datenbank: Definition

- Eine temporale Datenbank ist eine Datenbank mit Unterstützung für Anwendungszeit und/oder Transaktionszeit
- Anfrageevaluierung in temporalen Datenbanken ist nicht als einfaches multidimensionales Indizierungsproblem zu verstehen

Retrospektive: Ein klinisches Szenario

Ms. Jones was seen in the diabetes clinic on January 14 1997 at 11 A.M. Her blood-glucose value at that time was measured in the clinic as 220 mg/ 100ml. She complained of vomiting and dizziness for the past 2 or 3 days.

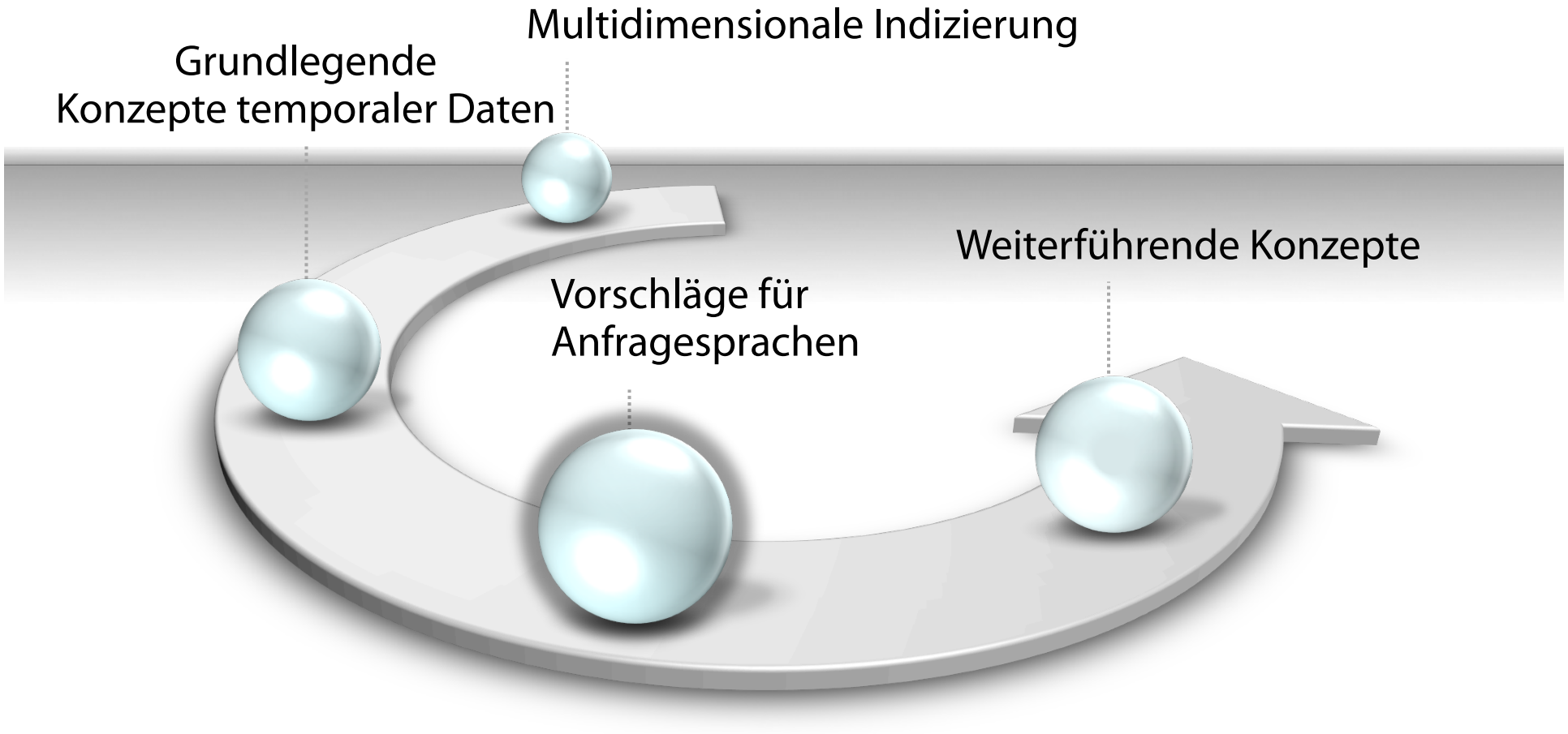
She was eventually hospitalized on the same day. A more accurate blood-glucose test that was taken at the same time as the one performed in the clinic returned from the laboratory on January 15 1997, with a value of 380 mg/100ml.

Ms. Jones was discharged on January 17 1997, and was seen again in the clinic on January 24 1997. At that time, several renal-function serum and urine tests were performed in addition to measuring blood-glucose values. A complete neurological assessment was carried out as well.

Entnommen aus: Temporal Databases and Maintenance of Time-Oriented Clinical Data, Yuval Shahar, M.D., Ph.D.

Non-Standard-Datenbanken

Von der multidimensionalen Indizierung zu temporalen Daten



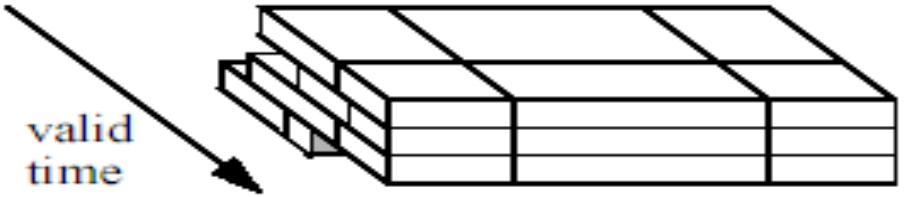
TSQL2

- Wir betrachten im Folgenden einige Konzepte von TSQL2[1]
- Wesentliche Kennzeichen von TSQL2:
  - Bitemporale Repräsentation möglich
  - Zeitdimensionen implizit (keine normalen Attribute)
  - Zeitdimensionen vom Benutzer (dynamisch) angefordert
  - Modifizierer für SQL-Ausdrücke zur Berücksichtigung von zeitlichen Aspekten (siehe nachfolgende Beispiele)

Präsentationen inspiriert von einem Vortrag von Puneet Mehta nach einem Aufsatz von Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen und Andreas Steiner, "Transitioning Temporal Support in TSQL2 to SQL3," in Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.), Springer, LNCS 1399, pp. 150––194, **1998**

[1]R.T. Snodgrass, et. al., TSQL2 Language Specification. SIGMOD Record (SIGMOD) 23(1):65-86, **1994**

Relation mit Gültigkeitszeit




```
CREATE TABLE Employee(
   Name    VARCHAR(30),
   Manager VARCHAR(30) VALIDTIME REFERENCES Employee (Name),
   Dept    VARCHAR(20)) AS VALIDTIME PERIOD(DATE)
```

VALIDTIME : Zu jedem Zeitpunkt im Intervall (Granularität: Datumsangaben)

Ziel dieser Präsentation: Problembewusstsein entwickeln

Relation mit Gültigkeitszeit (TSQL2)

LIST ALL EMPLOYES WHO WERE NOT MANAGERS

```
VALIDTIME SELECT Name FROM Employee EXCEPT SELECT Manager FROM Employee
```

VALIDTIME : Zu jedem Zeitpunkt im Intervall (Granularität: Datumsangaben)

Ziel dieser Präsentation: Problembewusstsein entwickeln

Relation mit Gültigkeitszeit (TSQL2)

EXTRACT THE SIZE HISTORY OF THE DEPARTMENT

```
VALIDTIME SELECT Dept, COUNT(*)
FROM Employee
GROUP BY Dept
```

VALIDTIME : Zu jedem Zeitpunkt im Intervall (Granularität: Datumsangaben)

Ziel dieser Präsentation: Problembewusstsein entwickeln

Relation mit Gültigkeitszeit (TSQL2)

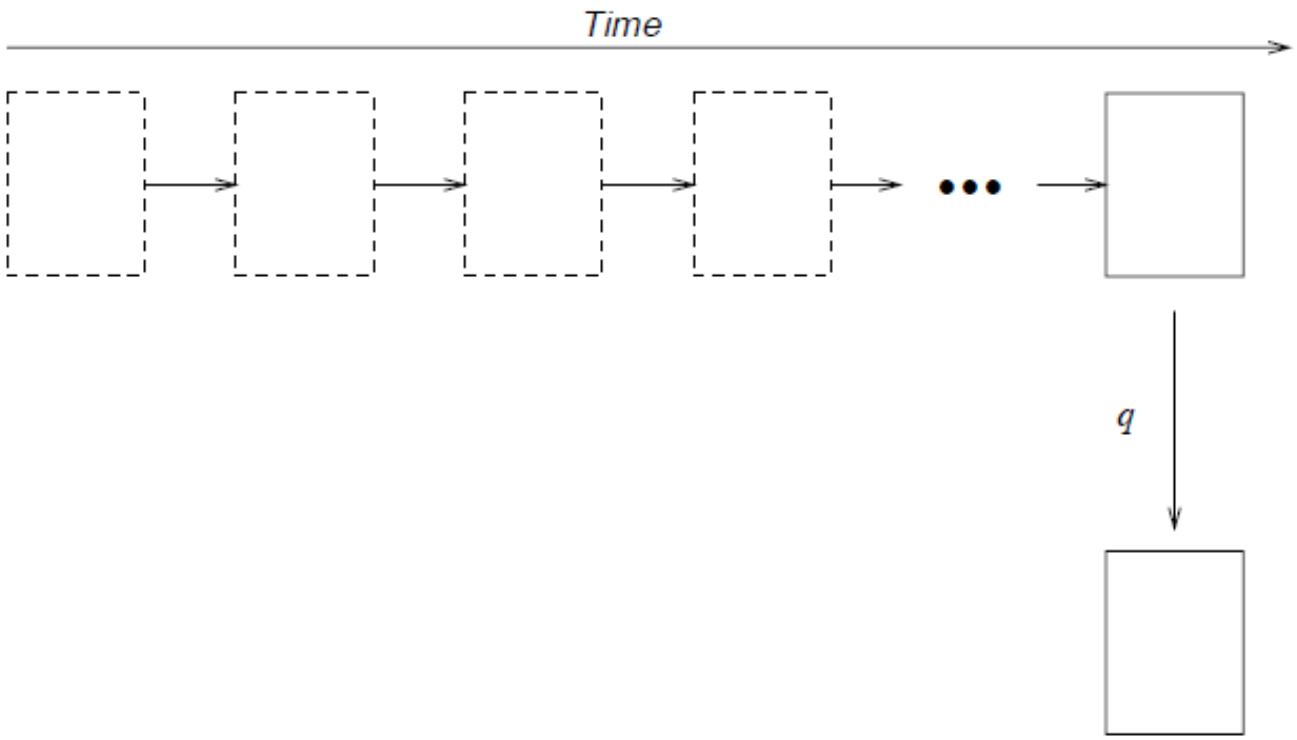
CHANGE THE MANAGER OF TOOLS DEPT FOR 1994 to BOB

```
VALIDTIME PERIOD '[1994-01-01 - 1994-12-31]' UPDATE Employee
SET Manager = 'Bob'
WHERE Dept = 'Tools'
```

VALIDTIME : Zu jedem Zeitpunkt im Intervall (Granularität: Datumsangaben)

Ziel dieser Präsentation: Problembewusstsein entwickeln

Schnappschussrelation

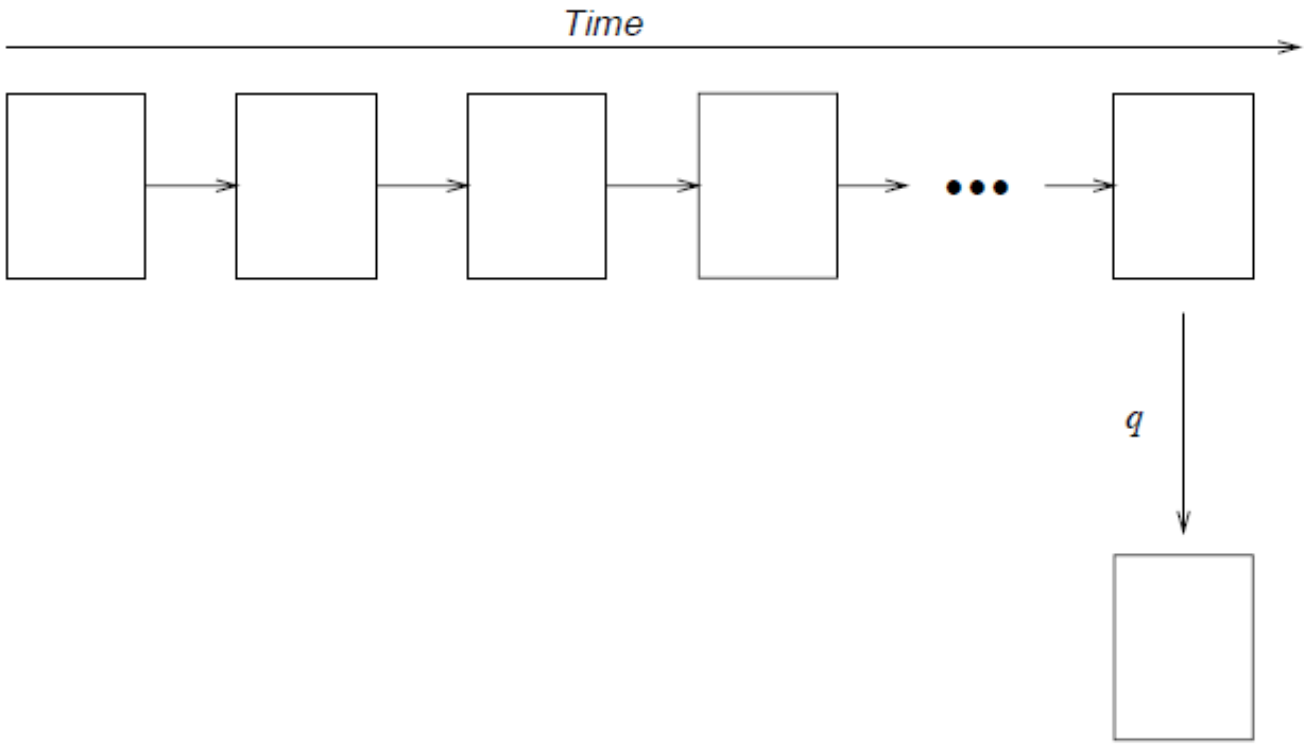


q: STANDARD-SQL-Anfrage

Schnappschussrelation

```
> CREATE TABLE employee(ename VARCHAR(12), eno INTEGER PRIMARY KEY,
                        street VARCHAR(22), city VARCHAR(10), birthday DATE);
> CREATE TABLE salary(eno INTEGER REFERENCES employee(eno), amount INTEGER);

> INSERT INTO employee
      VALUES ('Therese', 5873, 'Bahnhofstrasse 121', 'Zurich', DATE '1961-03-21');
> INSERT INTO employee
      VALUES ('Franziska', 6542, 'Rennweg 683', 'Zurich', DATE '1963-07-04');

> INSERT INTO salary VALUES (6542, 3200);
> INSERT INTO salary VALUES (5873, 3300);
```

Gültigkeitszeit und Standard-SQL-Anfrage



q: STANDARD-SQL-Anfrage
(arbeitet auf aktuell gültigem Schnappschuss)

Gültigkeitszeit und Standard-SQL-Anfrage

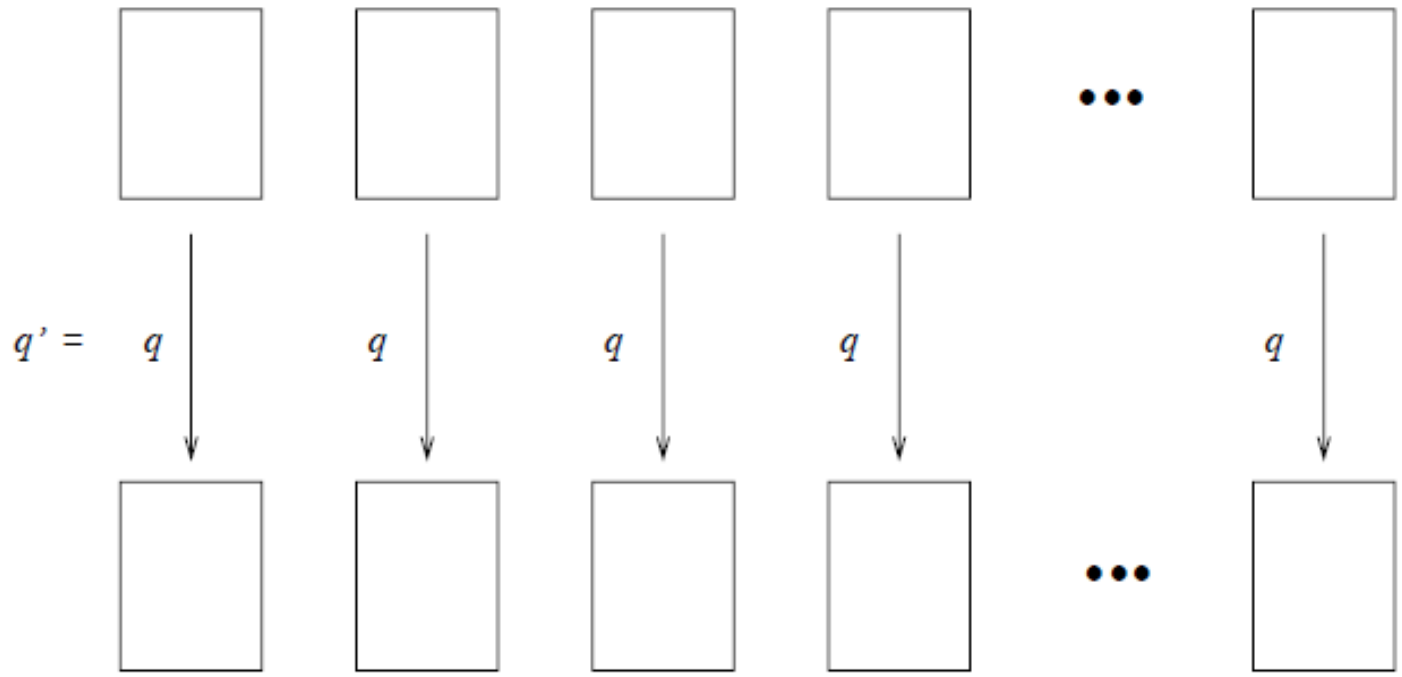
```
> ALTER TABLE salary ADD VALIDTIME PERIOD(DATE);
> ALTER TABLE employee ADD VALIDTIME PERIOD(DATE);

 > INSERT INTO employee
        VALUES('Lilian', 3463, '46 Speedway', 'Tuscon', DATE '1970-03-09');
 > INSERT INTO salary VALUES(3463, 3400);
 > COMMIT;
```

ename	eno	street	city	birthday	Valid
Therese	5873	Bahnhofstrasse 121	Zurich	1961-03-21	[1995-02-01 - 9999-12-31)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tuscon	1970-03-09	[1995-02-02 - 9999-12-31)

eno	amount	Valid
6542	3200	[1995-02-01 - 9999-12-31)
5873	3630	[1995-02-01 - 9999-12-31)
3463	3400	[1995-02-02 - 9999-12-31)

```
> SELECT ename, city
  FROM   high_salary AS s, employee AS e
  WHERE  s.eno = e.eno;
```

Gültigkeitszeit und historische Sequenzanfrage



q: Give history of monthly salaries paid to employees (Sequence Query)

Gültigkeitszeit und historische Sequenzanfrage

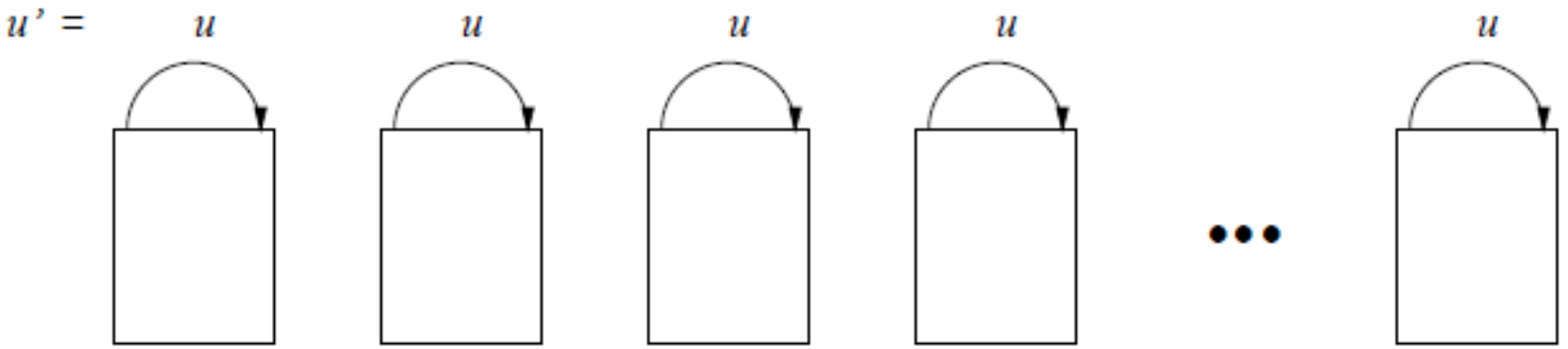
Ausdrucksmodifizierer

```
> VALIDTIME
    SELECT ename, amount
    FROM   salary AS s, employee AS e
    WHERE  s.eno = e.eno;
```

ename	amount	Valid
Franziska	3200	[1995-02-01 - 9999-12-31)
Therese	3630	[1995-02-01 - 9999-12-31)
Lilian	3400	[1995-02-02 - 9999-12-31)

Ziel dieser Präsentation: Problembewusstsein entwickeln

Gültigkeitszeit und Änderung nach Sequenzanfrage



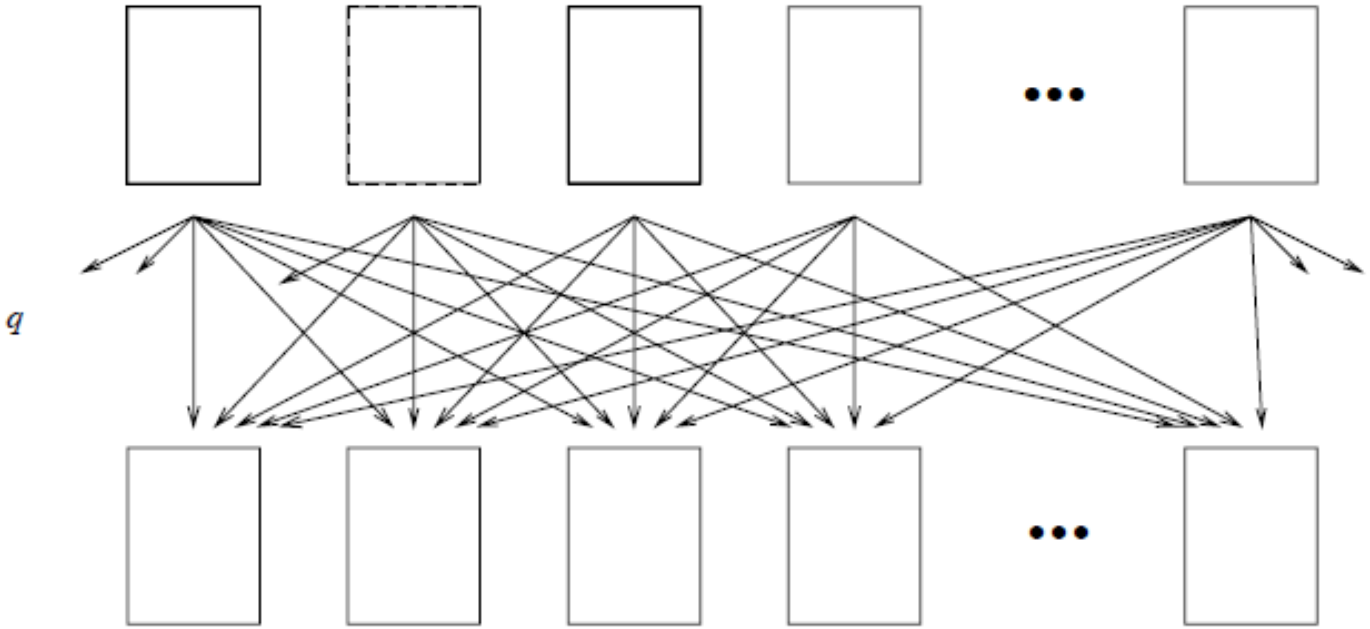
Change the town named ‘TUSCON‘ to ‘TUCSON’ (Sequence Query)

Gültigkeitszeit und Änderung nach Sequenzanfrage

```
> VALIDTIME UPDATE employee
            SET city = 'Tucson'
          WHERE city = 'Tuscon';
> COMMIT;
```

Ziel dieser Präsentation: Problembewusstsein entwickeln

Gültigkeitszeit und historische Nicht-Sequenz-Anfrage



q: Who was given SALARY raises ? (Non Sequence Query)

Gültigkeitszeit und historische Nicht-Sequenz-Anfrage

```
> UPDATE salary
  SET amount = 1.05 * amount
  WHERE eno = (SELECT S.eno
               FROM salary AS S, employee as E
               WHERE ename = 'Lilian' AND E.eno = S.eno);

> COMMIT;
```

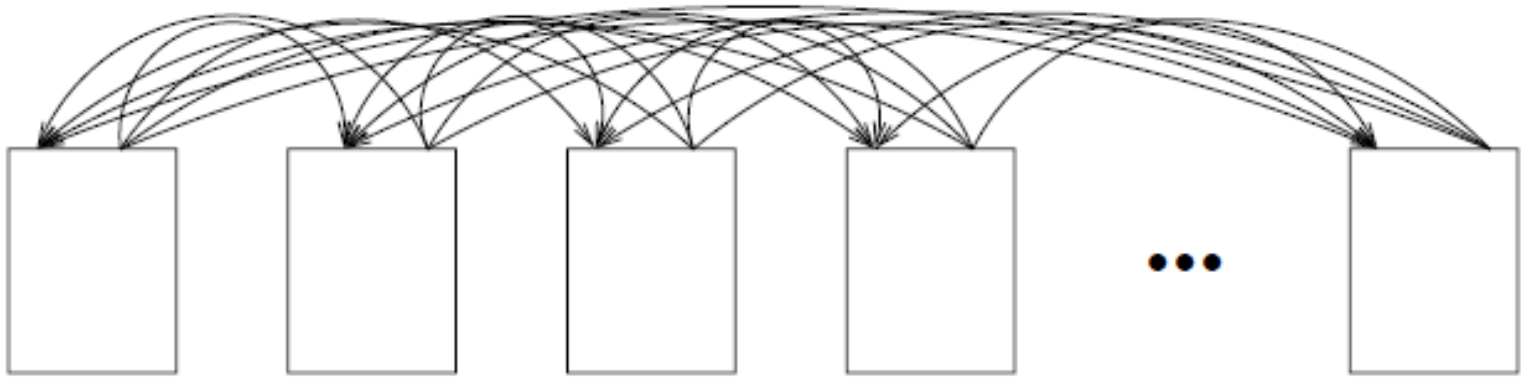
eno	amount	Valid
6542	3200	[1995-02-01 - 9999-12-31)
3463	3400	[1995-02-02 - 1995-04-01)
3463	3570	[1995-04-01 - 9999-12-31)

```
> NONSEQUENCED VALIDTIME SELECT ename
  FROM employee AS E, salary AS S1, salary AS S2
  WHERE E.eno = S1.eno AND E.eno = S2.eno
     AND S1.amount < S2.amount AND VALIDTIME(S1) MEETS VALIDTIME(S2);
```

Ziel dieser Präsentation: Problembewusstsein entwickeln

ename
Lilian

Gültigkeitszeit und Änderung mit Nicht-Sequenz-Anfrage



Give employees 5% raise if they never had a raise before?
(Non Sequence Query)

Vorbereitung: nicht verbundene Intervalle

```
> VALIDTIME PERIOD '[1995-07-01 - 1996-01-01)'
  DELETE FROM salary
  WHERE eno = 6542;

> VALIDTIME PERIOD '[1995-07-01 - 1996-01-01)'
  DELETE FROM employee
  WHERE eno = 6542;

> COMMIT;
```

ename	eno	street	city	birthday	Valid
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)

eno	amount	Valid
6542	3200	[1995-02-01 - 1995-07-01)
6542	3200	[1996-01-01 - 9999-12-31)
3463	3400	[1995-02-02 - 1995-04-01)
3463	3570	[1995-04-01 - 9999-12-31)

Ziel dieser Präsentation:
Problembewusstsein entwickeln

Gültigkeitszeit und Änderung mit Nicht-Sequenz-Anfrage

```
> UPDATE salary AS S
  SET amount = 1.05 * amount
  WHERE NOT EXISTS (SELECT *
                    FROM (NONSEQUENCED VALIDTIME SELECT *
                          FROM salary AS S1, salary AS S2
                          WHERE S1.amount < S2.amount
                              AND VALIDTIME(S1) MEETS VALIDTIME(S2)
                              AND S1.eno = S.eno and S2.eno = S.eno) AS S3
                    );
> COMMIT;
```

eno	amount	Valid
6542	3200	[1995-02-01 - 1995-06-01)
6542	3360	[1995-06-01 - 1995-07-01)
6542	3360	[1996-01-01 - 9999-12-31)
3463	3400	[1995-02-02 - 1995-04-01)
3463	3570	[1995-04-01 - 9999-12-31)

Ziel dieser Präsentation:
Problembewusstsein entwickeln

Transaktionszeit: Wofür benötigt?

- Repräsentation von früheren Zuständen (z.B. für Auditing-Zwecke)
- Schreibzugriffe für frühere Zustände könnten limitiert sein (z.B. für sicheres Auditing)
  - Stattdessen: Kompensationstransaktionen zur Fehlerkorrektur

Temporale Relation (Transaktionszeit)

Emp	Dept	Valid Time	Transaction Time
Jake	Shipping	[1995-06-10 - 1995-06-16)	[1995-06-05 - 1995-06-10)
Jake	Shipping	[1995-06-05 - 1995-06-21)	[1995-06-10 - 1995-06-15)
Jake	Shipping	[1995-06-10 - 1995-06-16)	[1995-06-15 - 1995-06-20)
Jake	Loading	[1995-06-10 - 1995-06-16)	[1995-06-20 - 9999-12-31)

- Beispiel: Anstellung von Jake
- Darstellung einer einzigen Anstellung, mit Änderungen in der modellierten Realität mit jeweils zugehörigem Änderungszeitpunkt

Temporale Relation (Transaktionszeit)

ename	eno	street	city	birthday	Valid
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)

```
ALTER TABLE employee ADD TRANSACTIONTIME;
COMMIT;
```

ename	eno	street	city	birthday	Valid	Transaction
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)	[1995-07-01 - 9999-12-31)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)	[1995-07-01 - 9999-12-31)

Temporale Relation (Transaktionszeit)

```
UPDATE employee
SET street = 'Niederdorfstrasse 2'
WHERE ename = 'Franziska';
COMMIT;
```

ename	eno	street	city	birthday	Valid	Transaction
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)	[1995-07-01 - 9999-12-31)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-07-01 - 1995-08-01)
Franziska	6542	Niederdorfstrasse 2	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-08-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)	[1995-07-01 - 9999-12-31)

Franziska hat 2 Adressen, das System kennt zu verschiedenen Zeiten nur eine.

Temporale Relation (Transaktionszeit)

When was the street corrected, and what were the old and new values?
(Nonsequenced tx time & sequenced vt time)

```
NONSEQUENCED TRANSACTIONTIME AND VALIDTIME
SELECT e1.ename, e1.street AS old_street, e2.street AS new_street,
       BEGIN(TRANSACTIONTIME(e2)) AS trans_time
FROM employee AS e1, employee AS e2
WHERE e1.eno = e2.eno AND TRANSACTIONTIME(e1) MEETS TRANSACTIONTIME(e2)
```

ename	old_street	new_street	trans_time	Valid
Franziska	Rennweg 683	Niederdorfstrasse 2	1995-08-01	[1996-01-01 - 9999-12-31)

Temporale Relation (Transaktionszeit)

When did we think that someone lived somewhere for more than six months?

```
VALIDTIME AND TRANSACTIONTIME SELECT ename, street
FROM employee
WHERE INTERVAL(VALIDTIME(employee) MONTH) > INTERVAL '6' MONTH
```

ename	street	Valid	Transaction
Franziska	Rennweg 683	[1996-01-01 - 9999-12-31)	[1995-07-01 - 1995-08-01)
Franziska	Niederdorfstrasse 2	[1996-01-01 - 9999-12-31)	[1995-08-01 - 1995-09-01)
Lilian	46 Speedway	[1995-02-02 - 9999-12-31)	[1995-07-01 - 1995-09-01)

Temporale Relation (Transaktionszeit)

Assume it is now October 1, 1995. Lilian moved last June 1.
(PostActive update)

```
VALIDTIME PERIOD '[1995-06-01 - 9999-12-31)' UPDATE employee
SET street = '124 Alberca'
WHERE ename = 'Lilian'
COMMIT;
```

ename	eno	street	city	birthday	Valid	Transaction
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)	[1995-07-01 - 9999-12-31)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-07-01 - 1995-08-01)
Franziska	6542	Niederdorfstrasse 2	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-08-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)	[1995-07-01 - 1996-10-01)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 1995-06-01)	[1995-10-01 - 9999-12-31)
Lilian	3463	124 Alberca	Tucson	1970-03-09	[1995-06-01 - 9999-12-31)	[1995-10-01 - 9999-12-31)

Temporale Relation (Transaktionszeit)

"When was an employee's address for 1995 corrected?"
(run on Nov 1 95)

```
NONSEQUENCED TRANSACTIONTIME AND VALIDTIME PERIOD '[1995-01-01 - 1996-01-01)'
SELECT e1.ename, e1.street AS old_street, e2.street AS new_street,
       BEGIN(TRANSACTIONTIME(e2)) AS trans_time
FROM employee AS e1, employee AS e2
WHERE e1.eno = e2.eno AND TRANSACTIONTIME(e1) MEETS TRANSACTIONTIME(e2)
      AND e1.street <> e2.street
```

ename	old_street	new_street	trans_time	Valid
Lilian	46 Speedway	124 Alberca	1995-10-01	[1995-06-01 - 1996-01-01)

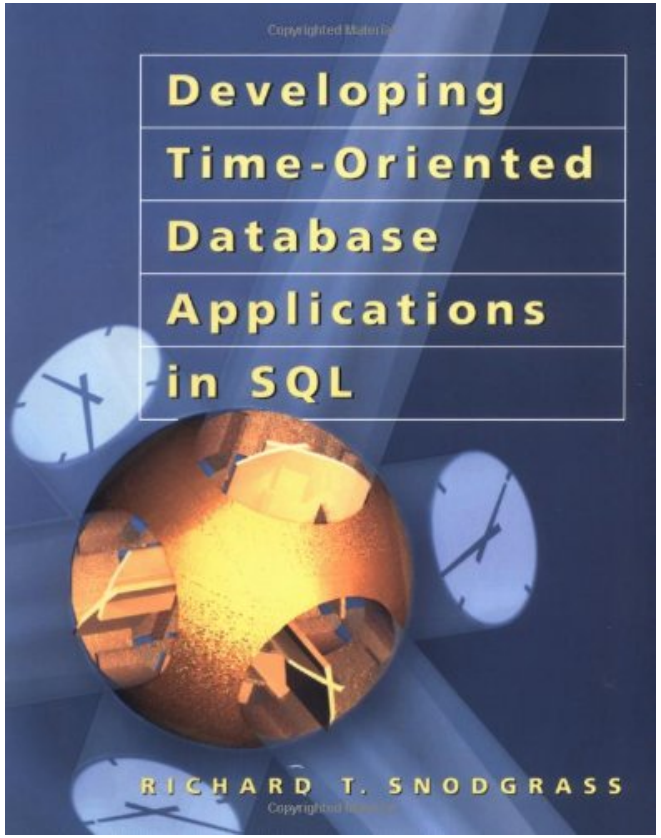
Es gab viel Kritik an TSQL2

- Viele nützliche Anfragen immer noch schwierig zu formulieren
- Aufwärtskompatibilität durch „versteckte“ Attribute für Gültigkeitszeit und Transaktionszeit sei nicht zielführend
  - sondern eher Ursache für viele Probleme
- Ausdrucksmodifizierer wie z.B. VALIDTIME, NONSEQUENCED logisch unklar
  (z.B. in Bezug auf Anfrageumformung zur Optimierung)
- Normalformen nicht einfach einzuhalten

Hugh Darwen, C.J. Date, An overview and Analysis of Proposals Based on the TSQL2 Approach, In Date on Database: Writings 2000-2006, C.J. Date, Apress, pp. 481-514, **2006**

Standardisierung

- TSQL2[2] konnte sich nicht durchsetzen
- Auch nicht SQL/Temporal[3] als Teil von SQL:1999 (SQL3)
- Neuer Versuch: SQL:2011[4]



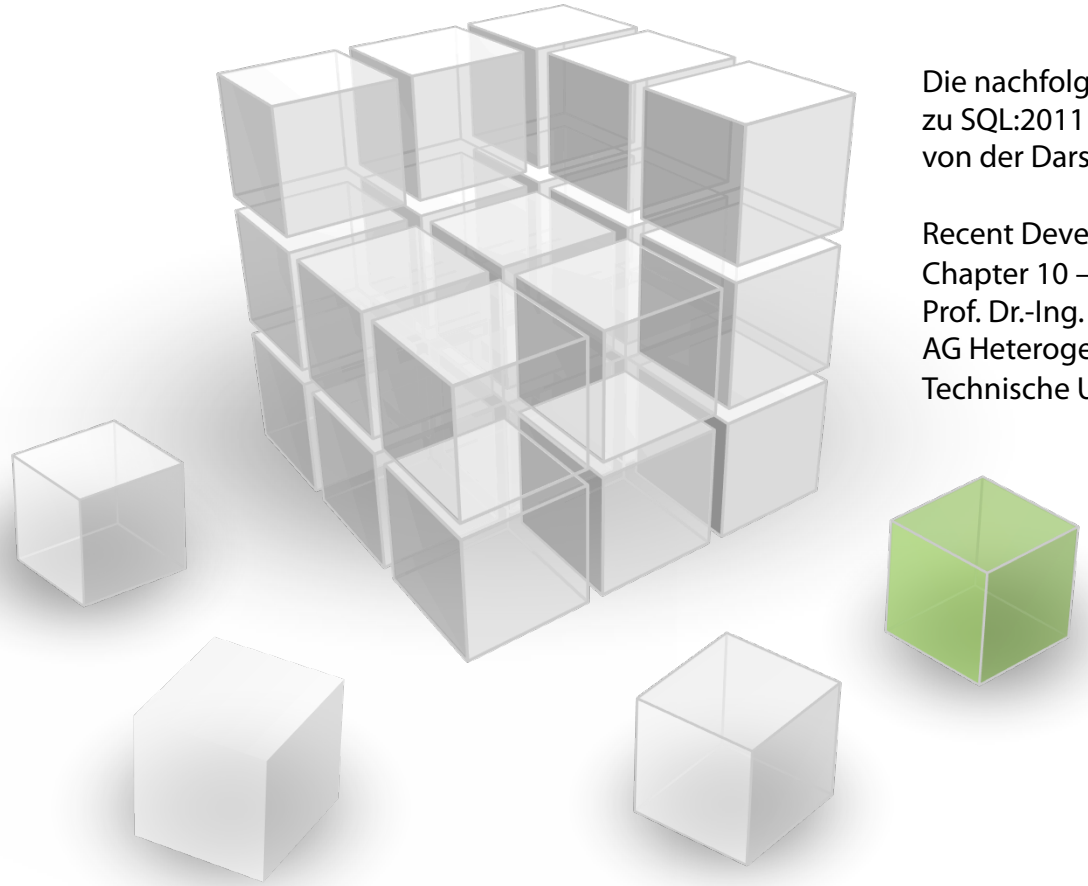
[2]R.T. Snodgrass, et. al., TSQL2 Language Specification. SIGMOD Record (SIGMOD) 23(1):65-86, **1994**

[3]Snodgrass, Richard T., Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, **1999**

Vgl. auch: C. J. Date, H. Darwen, N.A. Lorentzos, Temporal Data and the Relational Model, Morgan Kaufman, **2003**

[4]Kulkarni, Krishna, and Jan-Eike Michels. Temporal features in SQL: 2011. ACM SIGMOD Record 41.3, pp. 34-43, **2011**

Danksagung



Die nachfolgenden 15 Präsentationen zu SQL:2011 sind inspiriert von der Darstellung in

Recent Developments for Data Models
Chapter 10 – Temporal Data Models
Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Technische Universität Kaiserslautern

SQL:2011 – Unterstützung für temporale Daten

- Kein neuer Datentyp Zeitperiode
- Definition von Perioden für Tabellen mit Bezug auf Paaren von Spalten für Beginn und Ende
  (→ **Tabellenperioden**)
- Zugrundeliegendes Intervallmodell: [Start, Ende)

SQL:2011 – Unterstützung für temporale Daten

- System-Versionierung von Tabellen
  - Unterstützung für **Transaktionszeit**
  - Standardanfragen beziehen sich auf „aktuelle“ Daten
  - Bezugnahme auf „historische“ Version möglich durch FOR SYSTEM TIME-Schlüsselwort
- Anwendungszeit von Tabellen durch Perioden
  - Unterstützung für **Gültigkeitszeit** (Valid Time)
  - Primärschlüssel und Referentielle Integrität
  - Anfrage beziehen sich auf alle gültigen Tupel
  - Prädikate über Periode für Zugriff auf temporale Bereiche
- **Bitemporale** Tabellen

Tabellenperioden für Anwendungszeit

- Anwendungszeit/Gültigkeitszeit durch Nutzer bestimmt
- Neues Schlüsselwort PERIOD FOR
- Beispiel:

```
CREATE TABLE Emp(
    ENo    INTEGER,
    EStart DATE,
    EEnd   DATE,
    EDept INTEGER,
    PERIOD FOR EPeriod (EStart, EEnd) )
```

- Namen für Perioden frei wählbar
- Anfangs- und End-Attribute als normale Spalten
- Einfügen von Tupeln über INSERT wie bekannt

```
INSERT INTO Emp
VALUES (22217, DATE '2010-01-01', DATE '2011-11-12', 3)
```

Anwendungszeit – Änderung und Löschung

- Modifikationen beziehen sich auf eine effektive Periode
- Änderungen wirksam
  auf allen Tupeln mit über-
  lappenden Zeitperioden

Anwendungszeit und Primär-/Fremdschlüssel

- Primärschlüssel auf Anwendungszeitperiode bezogen
  - Überlappungen von Beginn/Ende-Spalten vermeiden

```
ALTER TABLE EMP
ADD PRIMARY KEY (ENo, EPeriod WITHOUT OVERLAPS)
```

- Referentielle Einschränkungen generalisiert auf jeden gültigen Zeitpunkt

ENo	EStart	EEnd	EDept
22218	2010-01-01	2011-02-03	3
22218	2011-02-03	2011-11-12	4

DNo	DStart	DEnd	DName
3	2009-01-01	2011-12-31	Test
4	2011-06-01	2011-12-31	QA

- Zeitperiodenangaben für Primär- und Fremdschlüssel

```
CREATE TABLE Dept
(DNo INTEGER, ..., PERIOD FOR DPeriod (DStart, DEnd),
PRIMARY KEY (DNo, DPeriod WITHOUT OVERLAPS))

ALTER TABLE Emp
ADD FOREIGN KEY (Edept, PERIOD EPeriod)
REFERENCES Dept (DNo, PERIOD DPeriod)
```

Anwendungszeit in Anfragen

- Verwendung von SELECT wie gehabt
  - Zeitperiodenspalten verwendbar wie gewohnt
- Spezielle Prädikate für Zeitperioden

```
SELECT EName, EDept
FROM Emp
WHERE ENo = 22217 AND EPeriod CONTAINS DATE '2011-01-02'
```

- Semantik (basierend auf Allen'sche Intervallbeziehungen)
  - x CONTAINS y ~ (x contains y) or (x starts y) or (x finishes y) or (x equal y)
  - x OVERLAPS y ~ (x overlaps y) or (y overlaps x) or (x contains y) or (y contains x) or (x starts y) or (y starts x) or (x finishes y) or (y finishes x) or (x equals y)
  - x EQUALS ~ x equals y
  - x PRECEDES ~ (x before y) or (x meets y)
  - x SUCCEEDS y ~ (y before x) or (y meets x)
  - x IMMEDIATELY PRECEDES y ~ x meets y
  - x IMMEDIATELY SUCCEEDS y ~ y meets x

Temporale Verbunde mit Anwendungszeit

ENo	EStart	EEnd	EDept
22218	2010-01-01	2011-02-03	3
22218	2011-02-03	2011-11-12	4

DNo	DStart	DEnd	DName
3	2009-01-01	2011-12-31	Test
4	2009-06-01	2011-01-31	Quality
4	2011-02-01	2011-12-31	QA

- Normale Verbunde betrachten Anwendungszeit nicht

  SELECT ENo, EDept, DName FROM Emp, Dept
  WHERE EDept = DNo

- Temporale Verbunde unter Verwendung von Zeitperioden

  SELECT ENo, EDept, DName
  FROM Emp, Dept
  WHERE EDept = DNo AND **EPeriod OVERLAPS DPeriod**

Tabellen und Versionierung mit Systemzeit

- Versionierung von Tabellen durch Systemzeitattribute (auch Transaktionszeit genannt)
- Beispiel

```
CREATE TABLE Emp (
        ENo INTEGER,
        Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
        Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
        EName VARCHAR(30),
        PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end))
WITH SYSTEM VERSIONING
```

Tabellen und Versionierung mit Systemzeit

- Versionierung von Tabellen durch Systemzeitattribute (auch Transaktionszeit genannt)
- Nur das DBMS kann (und muss) Start- und Endzeitpunkte für Systemzeitperioden liefern
  - Datentyp für Zeitpunkte systemabhängig
- Modifikation → zusätzliches Tupel für den alten Zustand vor der Änderung
  - Aktueller Zeilenzustand: Zeitstempel ist der neueste
  - Historischer Zeilenzustand: Zeitstempel nicht der neueste

Änderung von systemversionierten Tabellen

INSERT

- Systemzeitstart wird auf Transaktionszeit gesetzt
- Systemzeitende wird auf Maximalwert gesetzt
  INSERT INTO Emp (ENo, EName) VALUES (22217, 'Joe')

ENo	Sys_start	Sys_end	EName
22217	2012-01-01 09:00:00	9999-12-31 23:59:59	Joe

UPDATE und DELETE

- Operationen nur auf aktuellen Zeilen
- Automatisches Erzeugen von historischen Zeilen für jede geänderte oder gelöschte Zeile
- Bei UPDATE wird der neue Startzeitpunkt auf die Transaktionszeit gesetzt

ENo	Sys_start	Sys_end	EName
22217	2012-01-01 09:00:00	2012-02-03 10:00:00	Joe
22217	2012-02-03 10:00:00	9999-12-31 23:59:59	Tom

Einschränkungen und Anfragen

- Primärschlüssel und Fremdschlüssel für systemversionierte Tabellen
  - Sollen nur für die aktuellen Zeilen geprüft werden
  - Systemzeitperiode muss nicht als Teil des Schlüssels deklariert werden
- Anfragen auf systemversionierten Tabellen:
  - Beziehen sich standardmäßig auf aktuelle Zeilen
  - Bezugnahme auf historische Zeilen durch Schlüsselwort FOR SYSTEM TIME

Beispiele

- Bestimme Zeilen, die zu einem gegebenen Zeitpunkt die aktuellen Zeilen sind

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME AS OF TIMESTAMP '2011-01-02 00:00:00‘
```

- Bestimme Zeilen, die in einer Zeitperiode aktuell waren (ohne Endpunkt der Periode)

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME
        FROM TIMESTAMP '2011-01-02 00:00:00’ TO TIMESTAMP '2011-12-31 00:00:00'
```

- Bestimme Zeilen, die in einer Zeitperiode aktuell waren (mit Endpunkt der Periode)

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME
        BETWEEN TIMESTAMP '2011-01-02 00:00:00‘ AND TIMESTAMP '2011-12-31 00:00:00'
```

Bitemporale Tabellen

- Anwendungszeit und Systemzeit kombiniert
- Beispiel

```
CREATE TABLE Emp(
        ENo INTEGER,
        EStart DATE,
        EEnd DATE,
        EDept INTEGER,
        PERIOD FOR EPeriod (EStart, EEnd),
        Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
        Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
        EName VARCHAR(30),
        PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end),
        PRIMARY KEY (ENo, EPeriod WITHOUT OVERLAPS),
        FOREIGN KEY (EDept, PERIOD EPeriod)
            REFERENCES Dept (DNo, PERIOD DPeriod))
WITH SYSTEM VERSIONING
```

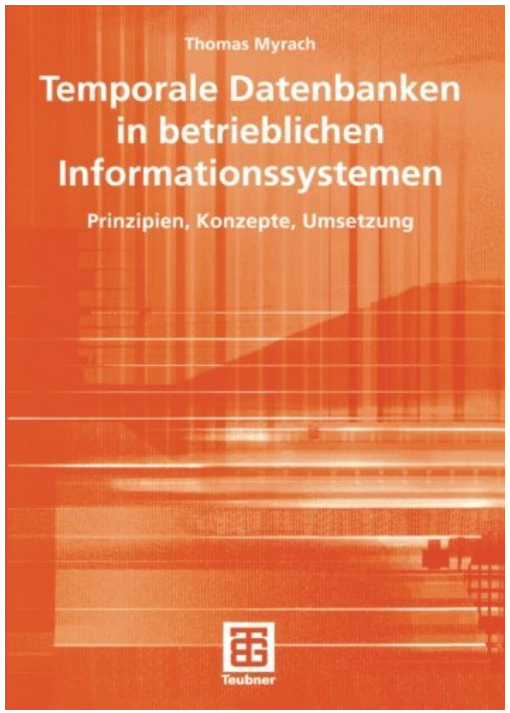
Anfragen an bitemporale Tabellen

- Anfragen können Prädikate für Systemzeitperioden und Anwendungszeitperioden enthalten
- Beispiel

```
SELECT ENo, EDept
FROM Emp FOR SYSTEM_TIME
        AS OF TIMESTAMP '2011-07-01 00:00:00'
WHERE ENo = 22217
        AND EPeriod CONTAINS DATE '2010-12-01'
```

Temporale Daten – Zusammenfassung

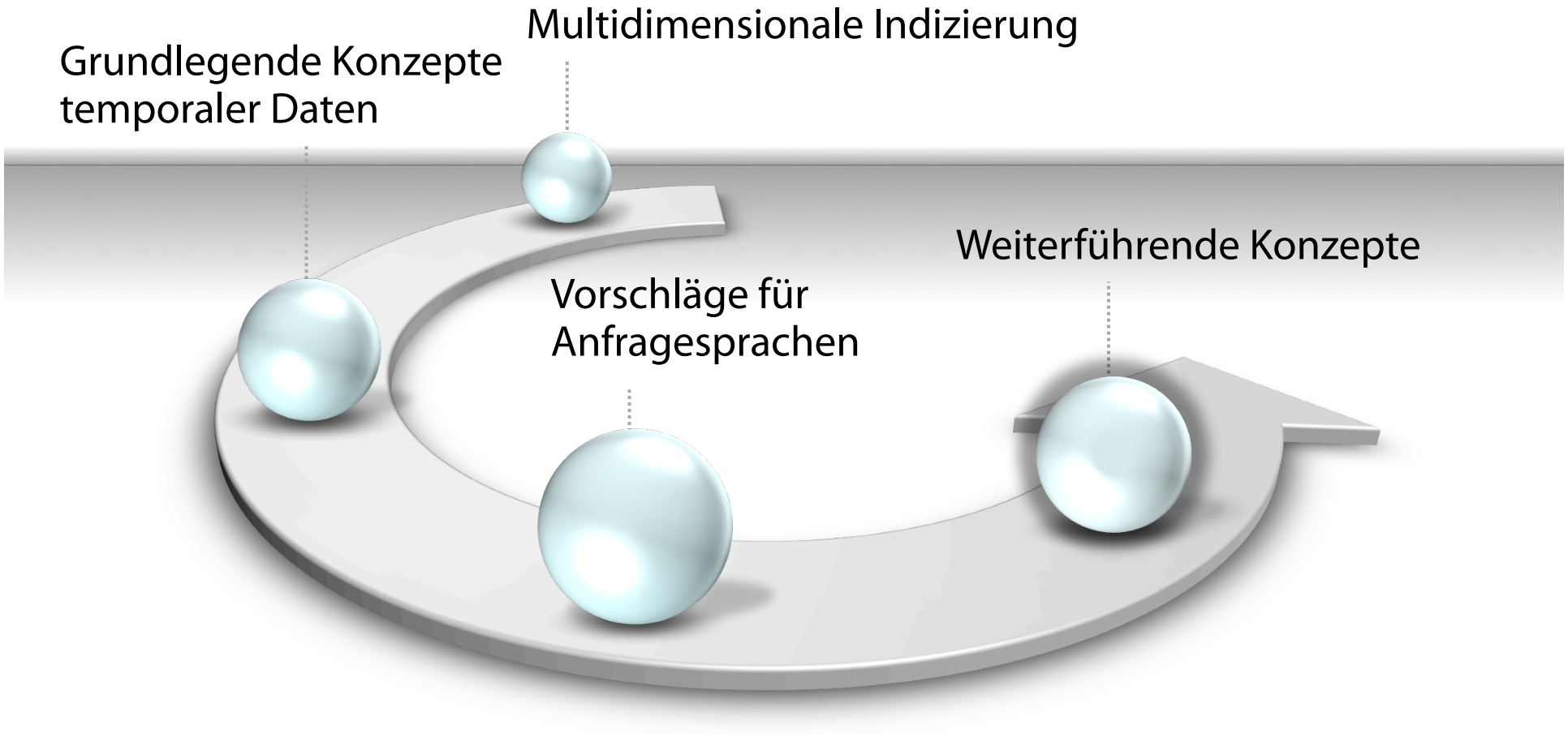
- Zentralen Anforderung in vielen Anwendungen
  - Zeitperioden, Prädikate (z.B. mit Allen-Operatoren)
  - Anwendungszeit vs. Systemzeit, bitemporale Tabellen
- Wichtige frühere Ansätze[1]
  - TSQL2, SQL/Temporal
- SQL:2011
  - Definition von Perioden, Anwendungszeit und Systemzeit
- SQL: Weitere Entwicklungen
  - Outer Join, Gruppierung, Aggregation mit Zeitperioden
  - Normalisierung (z.B. Zusammenfassung von Perioden)
  - Viele weitere



[1] CS Jensen, RT Snodgrass, Semantics of time-varying information, Journal of Information Systems, 21:4, pp. 311-352, **1996**

Non-Standard-Datenbanken

Von der multidimensionalen Indizierung zu temporalen Daten



Einsichten

- Unterscheidung von Sequenzanfragen und Nicht-Sequenzanfragen mit Gültigkeitszeit und Transaktionszeit sind nützlich
- Integration in SQL (Tupelkalkül) sperrig, wie wir gesehen haben
- Neues Anfragekonzept notwendig mit geeigneten syntaktischen Abstraktionen

Vereinfachung der Zeitrepräsentation

- Zeitstruktur $T_P = (T, <)$, eine lineare Ordnung von Zeitpunkten (Zeitstempel, Zeitinstanzen, Chronons, …)
- Wir betrachten nur noch die Zuordnung von Zeitpunkten (nicht Intervallen) zu Tupeln
  - Kann Gültigkeitszeit sein (z.B. Messwerterfassungszeitpunkt)
  - Kann Transaktionszeit sein (z.B. Eintragung eines Messwertes in ein DB-System)
- Ggf. Zuordnung von mehreren Zeitattributen für Gültigkeitszeit, Transaktionszeit, usw. (multidimensionale Zeitrepräsentation)

David Toman: Point-Based Temporal Extensions of SQL and Their Efficient Implementation. Temporal Databases, Dagstuhl , 211-237, **1997**

Das Zeitstempelmodell (Anwendungszeit)

Beispiel: **Booking (Meeting, Room, Time)**

Ein Tupel (m, r, t) denotiert das Faktum, dass eine Besprechung m in Raum r zur Zeit t stattfindet

Booking		
Meeting	*Room*	*Time*
DB Group	DC1331	06-Jan-04.10:00
DB Group	DC1331	06-Jan-04.10:01
DB Group	DC1331	06-Jan-04.10:02
...	...	...
DB Group	DC1331	16-Jan-04.11:59
Intro to Databases	MC4042	06-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	06-Jan-04.11:19
Intro to Databases	MC4042	08-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	08-Jan-04.11:19

Schnappschussmodell

Unterschiedliche Sicht auf Zeitstempel-basierte Daten

Time	`booking`
06-Jan-04.10:00	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
06-Jan-04.10:01	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
...	
06-Jan-04.11:19	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
06-Jan-04.11:20	{ (DB Group,DC1331) }
...	
06-Jan-04.11:59	{ (DB Group,DC1331) }
06-Jan-04.12:00	{ }
...	
06-Jan-04.12:00	{ }
08-Jan-04.10:00	{ (Intro to Databases,MC4042) }
...	
08-Jan-04.11:19	{ (Intro to Databases,MC4042) }

Definition: Temporale Schnappschussdatenbank

Eine **temporale Schnappschussdatenbank**

über einer Grundmenge D,

einer Zeitstruktur T_P und

einem Schema ρ (z.B. $\rho = \{ \text{Booking} \}$)

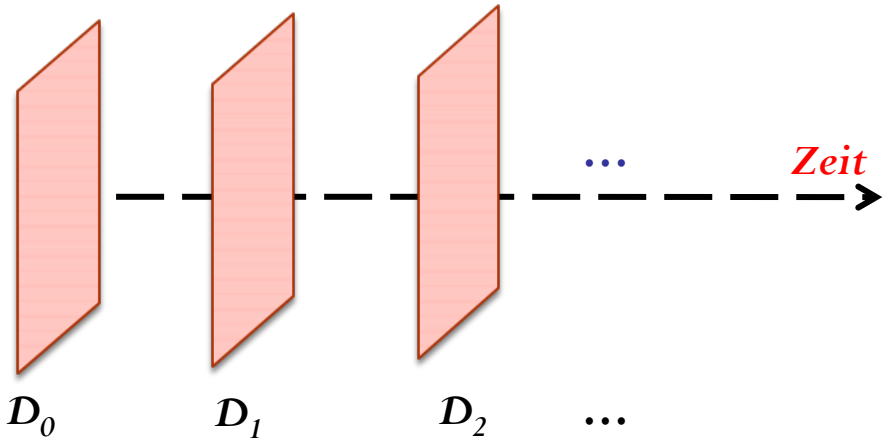
ist eine **Abbildung** $DB: T_P \rightarrow \mathcal{DB}(D, \rho)$,

wobei $\mathcal{DB}(D, \rho)$ die Klasse der endlichen relationalen Datenbanken über D und ρ bezeichnet.

Historie von relationalen Datenbanken

Eine Historie über einem Datenbankschema ρ und einer Grundmenge D ist eine Sequenz $\boldsymbol{H : (D_0, \ldots, D_n)}$ von Datenbankinstanzen, so dass

1. Alle Zustände $D_0, \ldots, D_n$ dasselbe Schema ρ und den gleichen Wertebereich verwenden
2. D_0 ist der initiale Zustand der Datenbasis
3. D_i resultiert aus der Änderung von D_{i-1}, für i > 0



Entwurfstheorie für temporale Datenbanken

- Rekonstruktion des formalen Rahmenwerks von Jensen[1]
- Basierend auf dem Begriff der **temporalen funktionalen Abhängigkeit** (functional dependency, FD): $X \xrightarrow{T} Y$

Eine temporale FD $X \xrightarrow{T} Y$ gilt in einer temporalen Schnappschussdatenbank, wenn die klassische FD $X \rightarrow Y$ in jeder Datenbank gilt

- Beispiel: die temporale FD $Meeting \xrightarrow{T} Room$ bedeutet, dass in jedem D_i jede Besprechung in höchstens einem Raum stattfindet
- Vorteile: Klassische FD-Behandlung für Normalformen usw. kann übernommen werden

[1] Jensen, C. S., R. T. Snodgrass, "Semantics of Time-Varying Information, Information Systems, Vol. 21, No. 4, pp. 311-352, **1996**

Beispiel

- Find all rooms in which the last meeting was 'DB group'.

Booking		
Meeting	*Room*	*Time*
DB Group	DC1331	06-Jan-04.10:00
DB Group	DC1331	06-Jan-04.10:01
DB Group	DC1331	06-Jan-04.10:02
...	...	...
DB Group	DC1331	16-Jan-04.11:59
Intro to Databases	MC4042	06-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	06-Jan-04.11:19
Intro to Databases	MC4042	08-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	08-Jan-04.11:19

- Return this for all time points before Christmas 2008

Beispiel

- Find all meetings that always met in the same room.

Booking		
Meeting	*Room*	*Time*
DB Group	DC1331	06-Jan-04.10:00
DB Group	DC1331	06-Jan-04.10:01
DB Group	DC1331	06-Jan-04.10:02
...	...	...
DB Group	DC1331	16-Jan-04.11:59
Intro to Databases	MC4042	06-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	06-Jan-04.11:19
Intro to Databases	MC4042	08-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	08-Jan-04.11:19

- Return this for all time points where 'Intro to Databases' is scheduled

Temporale Logik

- Wir definieren T_P *als* eine Sprache erster Ordnung mit Variablen X_i :

$$O ::= t_i < t_j \mid O \wedge O \mid \neg O \mid \exists t_i . O \mid X_i$$

- Damit lassen sich k-äre temporale Prädikate definieren:
  - *O*-Formel mit genau einer freien Variable t_0 und *k* Variablen $X_1,.., X_k$
  - Wir nehmen dabei an, t_i ist die temporale Variable, die in die Formel für die Variable X_i eingefügt wird
    - Aussagenlogischer Fall $X_i = p$ (p sei eine Aussage)
      - *p zum Zeitpunkt t_i wahr*
    - Prädikatenlogischer Fall $X_i = r(\boldsymbol{x})$ ($r(\boldsymbol{x})$ sei atomare Formel)
      - $R(\boldsymbol{x}, t_i)$, d.h. Tupel $\boldsymbol{x}$ gilt zum Zeitpunkt t_i

Temporale Operatoren und deren Semantik

- Wir nutzen die Temporale Logik, um neue Ausdrücke mit temporalen Operatoren zu definieren

- Einstellige Operatoren:

  $\Diamond \Phi$ *irgendwann in der Zukunft git* Φ

  $\Box \Phi$ *zu jedem Zeitpunkt in der Zukunft gilt* Φ

  Auch für Vergangenheit $\blacklozenge \Phi$ $\blacksquare \Phi$

- Zweistellige Operatoren: X_1 **until** X_2, X_1 **since** X_2

$$X_1 \textbf{ until } X_2 \triangleq \exists t_2 . t_0 < t_2 \wedge X_2 \wedge \forall t_1 (t_0 < t_1 < t_2 \rightarrow X_1)$$

$$X_1 \textbf{ since } X_2 \triangleq \exists t_2 . t_0 > t_2 \wedge X_2 \wedge \forall t_1 (t_0 > t_1 > t_2 \rightarrow X_1)$$

Temporale Operatoren und deren Semantik

- Sometime in the future: $\Diamond X_1 \triangleq \text{true } \mathbf{until}\ X_1$
- Sometime in the past: $\blacklozenge X_1 \triangleq \text{true } \mathbf{since}\ X_1$
- Always in the future: $\Box X_1 \triangleq \neg\Diamond\neg X_1$
- Always in the past: $\blacksquare X_1 \triangleq \neg\blacklozenge\neg X_1$
- Next: $\bigcirc X_1 \triangleq \exists t_1 . t_1 = t_0 + 1 \wedge X_1$
- Previous: $\bullet X_1 \triangleq \exists t_1 . t_1 + 1 = t_0 \wedge X_1$

Dov M. Gabbay, Mark A. Reynolds, Marcelo Finger: Temporal Logic - Mathematical Foundations and Computational Aspects – **Volume 1**; Clarendon Press Oxford **1994**

Dov M. Gabbay, Mark A. Reynolds, Marcelo Finger: Temporal Logic - Mathematical Foundations and Computational Aspects – **Volume 2**; Clarendon Press Oxford **2000**

Temporale Logik **erster Stufe** (FOTL)

- Ω : Eine Menge temporaler Junktoren (z.B. {**since**, **until**}) (üblicherweise in Prä- oder Infixschreibweise geschrieben)
- L^{Ω} : Temporale Logik erster Stufe (FOTL) mit Schema ρ

$$F ::= r(x_{i_1}, \ldots, x_{i_k}) \mid x_i = x_j \mid F \wedge F \mid \neg F \mid \omega(F_1, \ldots, F_k) \mid \exists x.F$$

$r \in \rho$ *and* $\omega \in \Omega$.

- Standard FOTL: $\mathcal{L}^{\{\textbf{since, until}\}}$:

$$F ::= r(x_{i_1}, \ldots, x_{i_k}) \mid x_i = x_j \mid F \wedge F \mid \neg F \mid F_1 \textbf{ since } F_2 \mid F_1 \textbf{ until } F_2 \mid \exists x.F$$

- Semantik von **since** und **until** siehe Temporale Logik von oben für prädikatenlogischen Fall $F_i = X_i = r(\boldsymbol{x})$

Temporale Operatoren für Anfragen

- Freie Variablen (beachte mögl. t_0) bestimmen Ergebnis
  - Find all rooms in which the last meeting was 'DB group‘:

$$(\neg\exists y.\mathtt{booking}(y,x))\ \mathbf{since}\ \mathtt{booking}(\mathtt{DB\ \ group},x)$$

  - Find all meetings with a scheduled break:

$$\blacklozenge\exists y.\mathtt{booking}(x,y)\wedge\neg\exists y.\mathtt{booking}(x,y)\wedge\Diamond\exists y.\mathtt{booking}(x,y)$$

- Antworten: Tupelmengen (Bindungen freier Variablen)
  - Eventuell will man t_0 eliminieren oder letztes t_0 betrachten
    - Siehe NONSEQUENCED bzw. Transaktionszeit in in TSQL2
  - Und wenn keine freien Variablen vorhanden sind?
    - Boolesche Anfrage: true = {()}, false = {}

Beispiel

Find all meetings with a scheduled break :

$$\blacklozenge\exists y.\texttt{booking}(x,y) \wedge \neg\exists y.\texttt{booking}(x,y) \wedge \Diamond\exists y.\texttt{booking}(x,y)$$

Booking		
Meeting	*Room*	*Time*
DB Group	DC1331	06-Jan-04.10:00
DB Group	DC1331	06-Jan-04.10:01
DB Group	DC1331	06-Jan-04.10:02
...	...	...
DB Group	DC1331	16-Jan-04.11:59
Intro to Databases	MC4042	06-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	06-Jan-04.11:19
Intro to Databases	MC4042	08-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	08-Jan-04.11:19

Time	booking
06-Jan-04.10:00	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
06-Jan-04.10:01	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
...	
06-Jan-04.11:19	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
06-Jan-04.11:20	{ (DB Group,DC1331) }
...	
06-Jan-04.11:59	{ (DB Group,DC1331) }
06-Jan-04.12:00	{ }
...	
06-Jan-04.12:00	{ }
08-Jan-04.10:00	{ (Intro to Databases,MC4042) }
...	
08-Jan-04.11:19	{ (Intro to Databases,MC4042) }

Beispiel

Find all meetings with a scheduled break :

$$\blacklozenge \exists y.\mathtt{booking}(x,y) \wedge \neg \exists y.\mathtt{booking}(x,y) \wedge \Diamond \exists y.\mathtt{booking}(x,y)$$

```
select r1.Meeting
from   Booking r1, Booking r2
where  r1.Meeting = r2.Meeting
  and  r1.time < r2.time
  and  not exists ( select *
                    from   Booking r3
                    where  r3.Meeting = r1.Meeting
                      and  r1.time < r3.time
                      and  r3.time < r2.time )
```

Effiziente Auswertung ...

- ... von Anfragen oder Integritätsbedingungen
- Kodierung von Historien in Zuständen
  - Datenbankschema erweitert mit zusätzlichen Relationen
  - Jeder Datenbankzustand erweitert mit Instanzen der zusätzlichen Relationen als erweiterter Zustand
  - Erweiterte Zustände H_k' kodieren die gesamte Historie
    $H : (H_0, \ldots, H_k)$
- Eigenschaften der Kodierung
  - Inkrementell berechenbar
  - mit Informationsverlust behaftet für „neue“, nicht bekannte Anfragen (nur für spezielle, vorher bekannte Anfragen nutzbar)
  - Platz-effizient

Chomicki, J. (1995). Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. ACM Transactions on Database Systems, 20(2):149–186.

Zusätzliche Relationen

- Beispiel Integritätsbedingung *C*
- Eine zusätzliche Relation r_α für jede temporale Unterformel α von *C*
  (also Formeln der Form ●*A* oder *A* ***since*** *B*)
- Stelligkeit von r_α ergibt sich aus der Anzahl der freien Variablen in der Formel α plus 1 (wegen Zeitparameter)
- Zusätzliche Relation r_C (1-stellig wegen t_0)
- Eine Integritätsbedingung *C* ist erfüllt, wenn r_C nicht leer ist (für alle t_0)

Beispiel

- Employees cannot be hired if they have been fired in the past and not subsequently reinstated

$$\neg(\exists x)(hired(x) \wedge (\neg reinstated(x) \ \mathbf{since} \ fired(x)))$$

- Zusätzliche Relationen

$$r_\alpha(x) \stackrel{df}{=} \neg reinstated(x) \ \mathbf{since} \ fired(x)$$

$$r_C \stackrel{df}{=} \neg(\exists x)(hired(x) \wedge r_\alpha(x))$$

- Induktive Definition

$$r_\alpha^0(x) \stackrel{df}{=} False$$

$$r_\alpha^{k+1}(x) \stackrel{df}{=} (r_\alpha^k(x) \vee fired^k(x)) \wedge \neg reinstated^{k+1}(x)$$

Begrenzte Historienkodierung

- Annahme: Menge der in der DB verwendeten Namen konstant (sog. Active Domain)
- Für relevante Formeln werden
  - wahre Formeln in die Zukunft „projiziert“, so dass zu einem Zeitpunkt *t* sichtbar wird, ob z.B. *p* in der Vergangen wahr war (dann ist bei *t* die Formel ◆ *p* wahr)
  - wahre Formeln aus der Zukunft in die Vergangenheit „projiziert“, so dass „lokal“ sichtbar wird, ob z.B. *p* in der Zukunft war ist.

Begrenzte Kodierung

Beispiel 1:

	0	1	2	3	⋯
$\{x : p(x)\}$	$\{a\}$	$\{\}$	$\{a\}$	$\{\}$	⋯
$\{x : \blacklozenge p(x)\}$	$\{\}$	$\{a\}$	$\{a\}$	$\{a\}$	⋯

Beispiel 2:

	0	1	2	3	⋯
$\{x : p(x)\}$	$\{a_0\}$	$\{a_1\}$	$\{a_2\}$	$\{a_3\}$	⋯
$\{x : \blacklozenge p(x)\}$	$\{\}$	$\{a_0\}$	$\{a_0, a_1\}$	$\{a_0, a_1, a_2\}$	⋯

Platzbedarf für ◆p(x) **linear** in der Anzahl der Namen in der DB, ist aber nicht von der Anzahl der Tupel in der DB abhängig

Unbegrenzte Kodierung

	0	1	2	3	...
$\{x : p(x)\}$	$\{a\}$	$\{\}$	$\{a\}$	$\{\}$	...
$\{x : \blacklozenge p(x)\}$	{}	{(a,0)}	{(a,0)}	{(a,0) (a, 2)}	

- Vorteil: Wäre für beliebige Formeln nutzbar

Temporale Logik: Zusammenfassung

- **Ziel der Sprachdefinition**: knappe Spezifikation von Modellen (oder gesuchten Objekten)
- **Ziel des Algorithmenentwurfs**: Linearer Scan z.B. um Anfrageergebnisse zu ermitteln oder Integritätsbedingungen zu testen
- **Tradeoff des besprochenen Ansatzes**: Viele Hilfsrelationen müssen (inkrementell) materialisiert werden
- **Problem**: Man kann kaum etwas bestimmen, was nach einer gewissen Zeit wieder weggelassen werden kann
- **Neue Idee**: Lasse den Benutzer ein Fenster bestimmen, in dem die Datenauswertung stattfinden soll

Sequenzen in SQL:2011

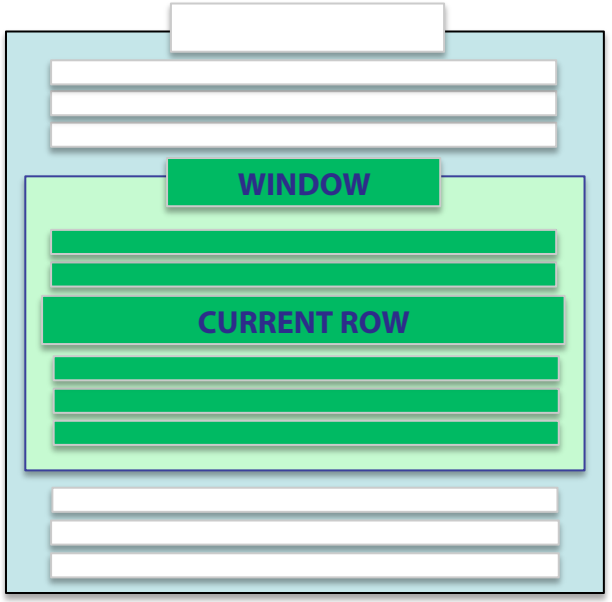
- Unterstützung für Datenanalyse in SQL:2011 durch Bildung von Sequenzen aus einer Relation
- Auch für Analyse zeitlicher Daten geeignet

- Nachfolgende 11 Präsentationen sind angelehnt an ein Tutorial „Advanced SQL“ von Marcin.Blaszczyk@cern.ch

Analytische Funktionen – Überblick

- Allgemeine Syntax:

```
SELECT analytical-function(col-expr)
       OVER (window-spec) [AS col-alias]
FROM [TABLE];
```



- Fensterspezifikation – Syntax

```
[PARTITION BY [expr list]]
ORDER BY [sort spec] [range spec]
```

- Beispiel für Bereichspezifikation (siehe Dokumentation)

```
•  ROWS UNBOUNDED PRECEDING AND CURRENT ROW (default)
•  ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
•  RANGE BETWEEN 2 PRECEDING AND 2 FOLLOWING
```

Geordnetes Analytisches Fenster

Analytische Funktion auf alle Tupel des Fensters angewandt

```
SQL>  select employee_id, last_name, manager_id, salary
      sum(salary) over (order by  employee_id, last_name, salary)
      as cumulative from employees;

EMPLOYEE_ID LAST_NAME MANAGER_ID SALARY CUMULATIVE
----------- --------- ---------- ------ ----------
        100 King                24000      24000
        101 Kochhar          100 17000      41000
        102 De Haan          100 17000      58000 = 24000 + 17000 + 17000
        103 Hunold           102  9000      67000
        104 Ernst            103  6000      73000
        105 Austin           103  4800      77800
        106 Pataballa        103  4800      82600
        107 Lorentz          103  4200      86800
        108 Greenberg        101 12000      98800
        109 Faviet           108  9000     107800
        110 Chen             108  8200     116000
```

Bereichsangabe – Beispiel (1)

RANGE BETWEEN 2 PRECEDING AND 2 FOLLOWING

```
SQL> select manager_id, last_name, salary, sum(salary) over (order by
          last_name, salary rows between 2 preceding and 1 following) as
          cumulative from employees;

MANAGER_ID LAST_NAME SALARY CUMULATIVE
---------- --------- ------ ----------
       103 Austin      4800      10800
       103 Ernst       6000      22800
       101 Greenberg  12000      31800
       102 Hunold      9000      51000 = 6000 + 12000 + 9000 + 24000
           King       24000      62000
       100 Kochhar    17000      54200
       103 Lorentz     4200      45200
```

Bereichsangabe – Beispiel (2)

ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

```
SQL> select manager_id, last_name, salary, sum(salary) over (order by
          last_name, salary rows between current row and unbounded
          following) as cumulative from emp_part;

MANAGER_ID LAST_NAME  SALARY CUMULATIVE
---------- --------- ------ ----------
       103 Austin      4800      77000
       103 Ernst       6000      72200
       101 Greenberg  12000      66200
       102 Hunold      9000      54200 = 9000 + 24000 + 17000 + 4200
           King       24000      45200
       100 Kochhar    17000      21200
       103 Lorentz     4200       4200
```

Partitioniertes analytisches Fenster

Analytische Funktion wird auf jede Partition angewandt

```
SQL> break on manager_id
SQL> SELECT manager_id, last_name, employee_id, salary,
     sum(salary) over (PARTITION BY manager_id order by employee_id) as cumulative
     FROM employees order by manager_id, employee_id, last_name;

MANAGER_ID LAST_NAME       EMPLOYEE_ID SALARY CUMULATIVE
---------- --------------- ----------- ------ ----------
       100 Kochhar                 101  17000      17000
           De Haan                 102  17000      34000
           Raphaely                114  11000      45000
           Weiss                   120   8000      53000
       101 Greenberg               108  12000      12000
           Whalen                  200   4400      16400
           Mavris                  203   6500      22900
           Baer                    204  10000      32900
       102 Hunold                  103   9000       9000
       103 Ernst                   104   6000       6000
           Austin                  105   4800      10800
           Pataballa               106   4800      15600
```

Partitioniertes analytisches Fenster

Analytische Funktion wird auf jede Partition angewandt

```
SQL> break on manager_id
SQL> SELECT manager_id, last_name, employee_id, salary,
     sum(salary) over (PARTITION BY manager_id order by employee_id) as cumulative
     FROM employees order by manager_id, employee_id, last_name;

MANAGER_ID LAST_NAME       EMPLOYEE_ID SALARY CUMULATIVE
---------- --------------- ----------- ------ ----------
       100 Kochhar                 101  17000      17000
           De Haan                 102  17000      34000
           Raphaely                114  11000      45000
           Weiss                   120   8000      53000
       101 Greenberg               108  12000      12000
           Whalen                  200   4400      16400
           Mavris                  203   6500      22900
           Baer                    204  10000      32900
       102 Hunold                  103   9000       9000
       103 Ernst                   104   6000       6000
           Austin                  105   4800      10800
           Pataballa               106   4800      15600
```

Partitioniertes analytisches Fenster

Analytische Funktion wird auf jede Partition angewandt

```
SQL> break on manager_id
SQL> SELECT manager_id, last_name, employee_id, salary,
     sum(salary) over (PARTITION BY manager_id order by employee_id) as cumulative
     FROM employees order by manager_id, employee_id, last_name;

MANAGER_ID LAST_NAME       EMPLOYEE_ID SALARY CUMULATIVE
---------- --------------- ----------- ------ ----------
       100 Kochhar                 101  17000      17000
           De Haan                 102  17000      34000
           Raphaely                114  11000      45000
           Weiss                   120   8000      53000
       101 Greenberg               108  12000      12000
           Whalen                  200   4400      16400
           Mavris                  203   6500      22900
           Baer                    204  10000      32900
       102 Hunold                  103   9000       9000
       103 Ernst                   104   6000       6000
           Austin                  105   4800      10800
           Pataballa               106   4800      15600
```

Partitioniertes analytisches Fenster

Analytische Funktion wird auf jede Partition angewandt

```
SQL> break on manager_id
SQL> SELECT manager_id, last_name, employee_id, salary,
     sum(salary) over (PARTITION BY manager_id order by employee_id) as cumulative
     FROM employees order by manager_id, employee_id, last_name;

MANAGER_ID LAST_NAME       EMPLOYEE_ID SALARY CUMULATIVE
---------- --------------- ----------- ------ ----------
       100 Kochhar                 101  17000      17000
           De Haan                 102  17000      34000
           Raphaely                114  11000      45000
           Weiss                   120   8000      53000
       101 Greenberg               108  12000      12000
           Whalen                  200   4400      16400
           Mavris                  203   6500      22900
           Baer                    204  10000      32900
       102 Hunold                  103   9000       9000
       103 Ernst                   104   6000       6000
           Austin                  105   4800      10800
           Pataballa               106   4800      15600
```

Partitioniertes analytisches Fenster

Analytische Funktion wird auf jede Partition angewandt

```
SQL> break on manager_id
SQL> SELECT manager_id, last_name, employee_id, salary,
     sum(salary) over (PARTITION BY manager_id order by employee_id) as cumulative
     FROM employees order by manager_id, employee_id, last_name;

MANAGER_ID LAST_NAME       EMPLOYEE_ID SALARY CUMULATIVE
---------- --------------- ----------- ------ ----------
       100 Kochhar                 101  17000      17000
           De Haan                 102  17000      34000
           Raphaely                114  11000      45000
           Weiss                   120   8000      53000
       101 Greenberg               108  12000      12000
           Whalen                  200   4400      16400
           Mavris                  203   6500      22900
           Baer                    204  10000      32900
       102 Hunold                  103   9000       9000
       103 Ernst                   104   6000       6000
           Austin                  105   4800      10800
           Pataballa               106   4800      15600
```

= 6000 + 4800 + 4800

Vordefinierte analytische Funktionen

- Aggregatsfunktionen
  - SUM
  - MAX
  - MIN
  - AVG
  - COUNT
- Zusätzlich für Fensterbereiche:
  - LAG
  - LEAD
  - FIRST
  - LAST
  - FIRST VALUE
  - LAST VALUE
  - ROW_NUMBER
  - DENSE_RANK

Analytische Funktion – Beispiel LAG

```
SQL> select * from currency order by 1;

DAY                    EURCHF
-------------------- ------
01-JUN-2012 00:00:00  1.240
02-JUN-2012 00:00:00  1.223
03-JUN-2012 00:00:00  1.228
04-JUN-2012 00:00:00  1.217
05-JUN-2012 00:00:00  1.255
06-JUN-2012 00:00:00  1.289
07-JUN-2012 00:00:00  1.291
08-JUN-2012 00:00:00  1.247
09-JUN-2012 00:00:00  1.217
10-JUN-2012 00:00:00  1.265
```

```
SQL> select day, EURCHF, lag(EURCHF,1) over
(order by day) as prev_eurchf from currency;

DAY                    EURCHF    PREV_EURCHF
-------------------- ------ --------------
01-JUN-2012 00:00:00  1.240
02-JUN-2012 00:00:00  1.223          1.240
03-JUN-2012 00:00:00  1.228          1.223
04-JUN-2012 00:00:00  1.217          1.228
05-JUN-2012 00:00:00  1.255          1.217
06-JUN-2012 00:00:00  1.289          1.255
07-JUN-2012 00:00:00  1.291          1.289
08-JUN-2012 00:00:00  1.247          1.291
09-JUN-2012 00:00:00  1.217          1.247
10-JUN-2012 00:00:00  1.265          1.217
```

Offene Punkte bei der Datenmodellierung

- Periodische Gültigkeit
- Unsicherheit:
  - Gültigkeit zu mindestens einem Zeitpunkt in einem Intervall
- Abhängigkeiten zwischen Anwendungszeitperioden bzw. Systemzeitperioden
  - Etwas gilt, nachdem/bis/bevor/während etwas anderes gilt, ohne dass die tatsächlichen Intervallgrenzen bekannt sind
- Open-World-Annahme: Wenn etwas nicht als gültig eingetragen ist, ist es nicht notwendigerweise ungültig
- Komplexere Zeitstrukturen
  - Verzweigende Zeit (branching time)
  - Dichte Zeitstrukturen
- Repräsentation und Erkennung von Ereignissen

Weitere Aspekte und Ausblick

- Zeit in **XML** (valid time / transaction time in XPath)
  - Ähnliche Prinzipien wie im relationalen Modell
- Zeit auf der Ebene der **konzeptuellen Datenmodellierung**
- Deklarationen für **physikalische Ebene** von temporalen Datenbanken (z.B. Indexe)
- **Strom-orientierte Datenverarbeitung**
  - Hohe Datenraten, Zwischendaten pro Fenster akkumulierend
  - Sekundärspeicher nötig, aber ggf. auch zu langsam, um Zwischenresultate aufzunehmen
  - Speicherbegrenzung und damit Approximation nötig