
Non-Standard-Datenbanken

Temporale Daten und das relationale Modell

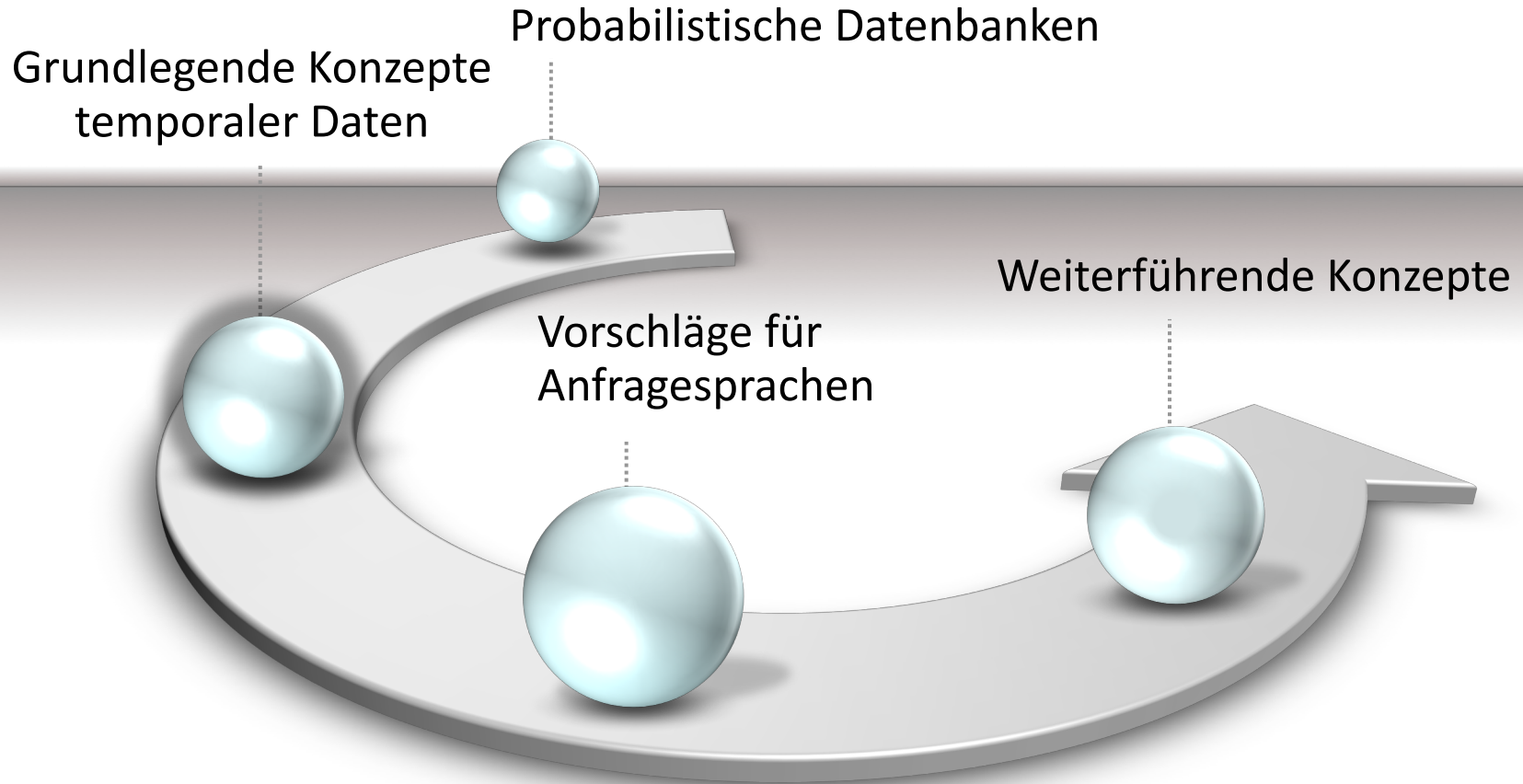
Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Non-Standard-Datenbanken

Von probabilistischen zu temporalen Datenbanken



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
- **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

Anfragen:

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

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

Zu prüfende Einschränkungen

S_FROM_TO

Ein Lieferant soll nicht einen Vertrag in unterschiedlichen aber überlappenden oder aneinander-grenzenden 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 aneinander-stoßenden 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.

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

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

Einschränkung:
keine „Lücken“

Nicht einfach zu
prüfen

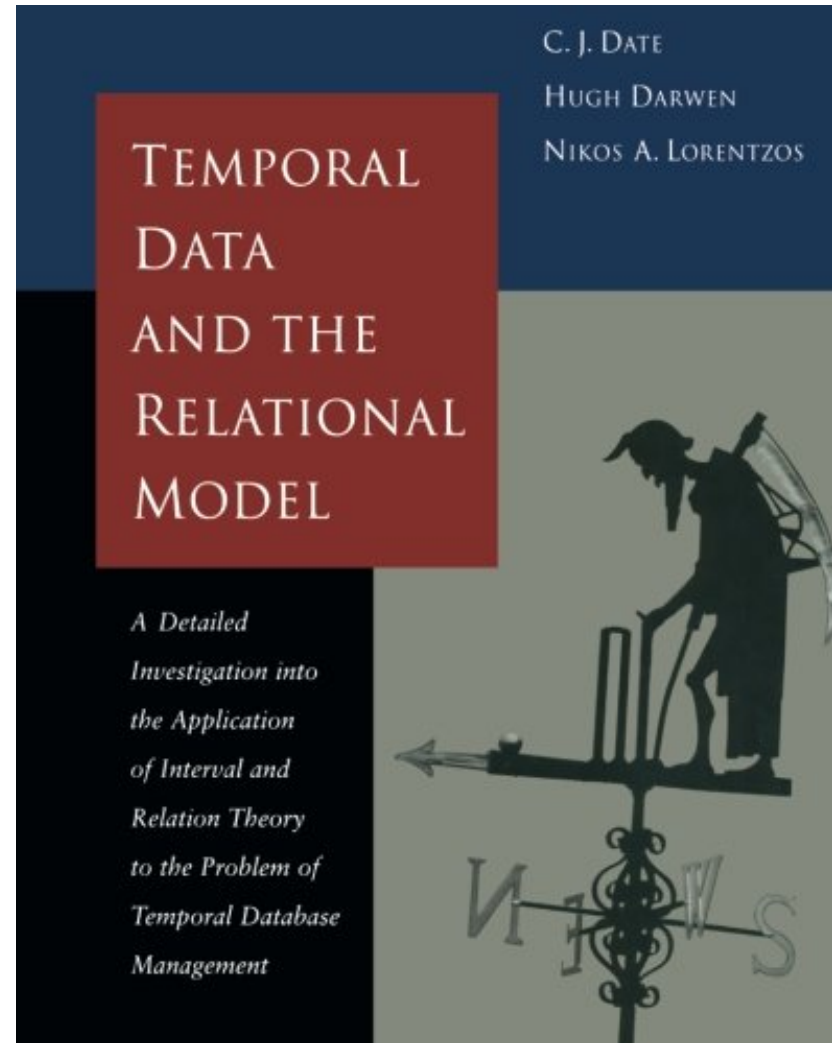
- ... weitere ...

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 and minus 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 $p2$



- $p1 = p2$

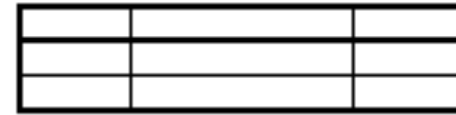


Dimensionen der Zeit

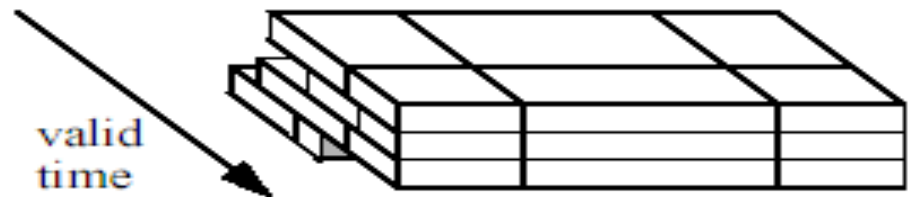
- Gültigkeit aus Sicht der Anwendung
(Gültigkeitszeit – valid time, **Anwendungszeit**, ...)
- Zeitpunkt der Eintragung eines Tupels in die Datenbank
„Bekannt“ aus Sicht des Systems
Zeitpunkt, an dem ein neuer Wert eingetragen wird
- (Transaktionszeit – transaction time, **Systemzeit**, ...)
- Bitemporal: Beides gleichzeitig repräsentiert
- Auch möglich: benutzerdefinierte Zeitangaben
- Seit langem untersuchtes Gebiet im Datenbankbereich¹

Relationen in zeitlichen Dimensionen

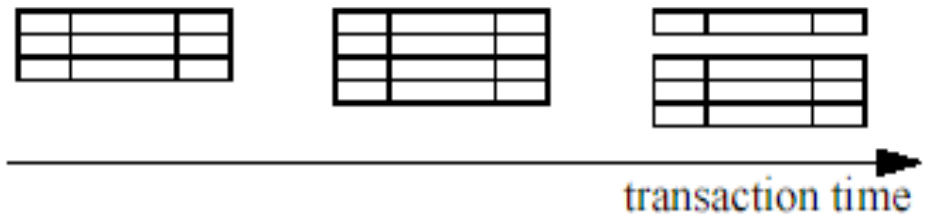
■ Schnappschuss



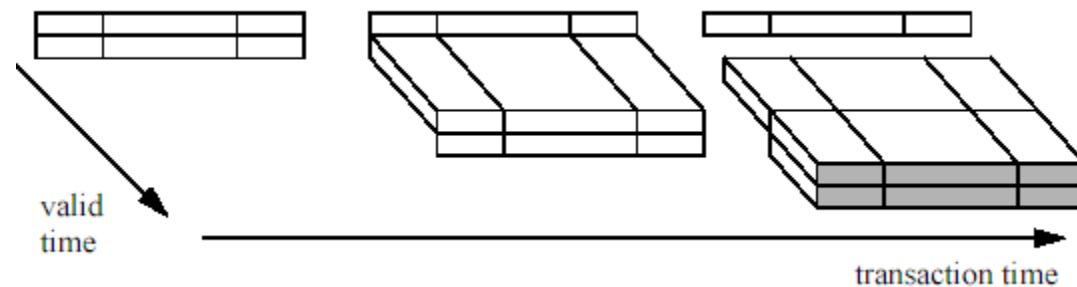
■ Gültigkeitszeiten



■ Transaktionszeiten



■ Bitemporale Relation

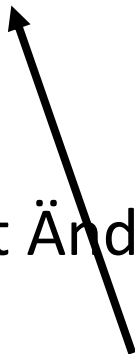


Transaktionszeit: Wofür benötigt?

- 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

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)

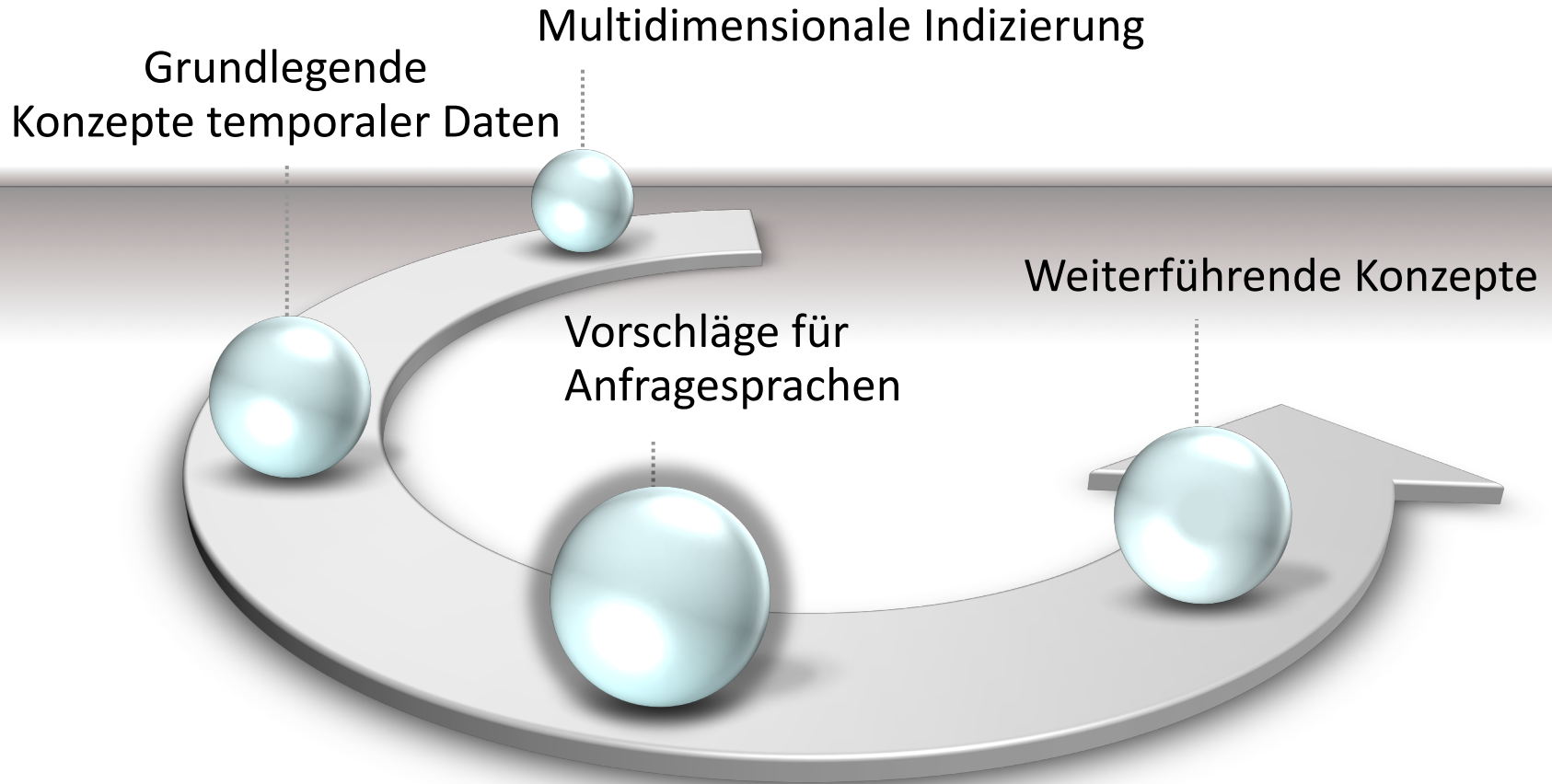
- Beispiel: Anstellung von Jake
 - Darstellung einer einzigen Anstellung, mit Änderungen in der modellierten Realität mit jeweils zugehörigem Änderungszeitpunkt
- 

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

Non-Standard-Datenbanken

Von der multidimensionalen Indizierung zu temporalen Daten



Standisierung schwierig

- TSQL2:
 - Bitemporale Repräsentation möglich
 - Zeitdimensionen implizit (keine normalen Attribute)
 - Zeitdimensionen vom Benutzer (dynamisch) angefordert
 - Modifizierer für SQL-Ausdrücke zur Berücksichtigung von zeitlichen Aspekten

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

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

- Viel Kritik an TSQL2

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

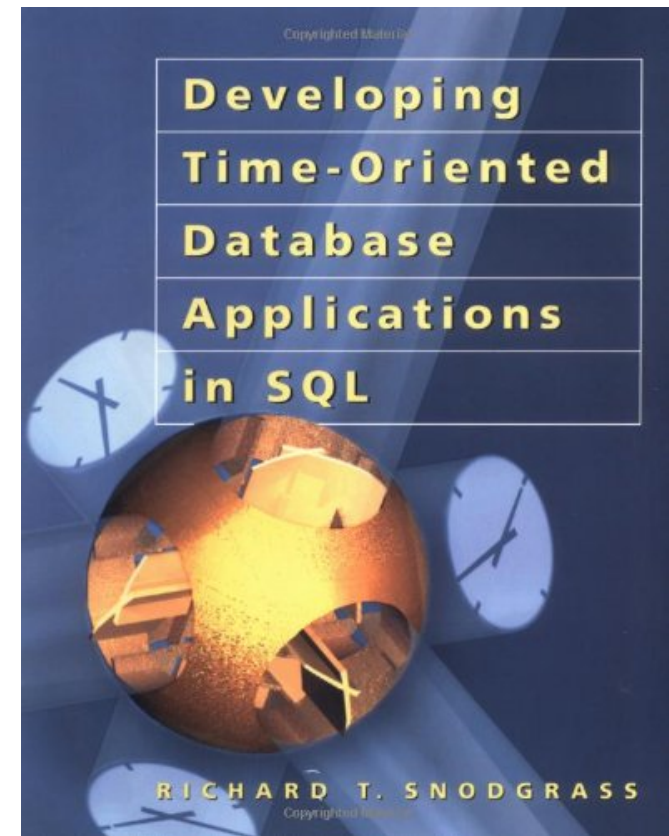
Standardisierung

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

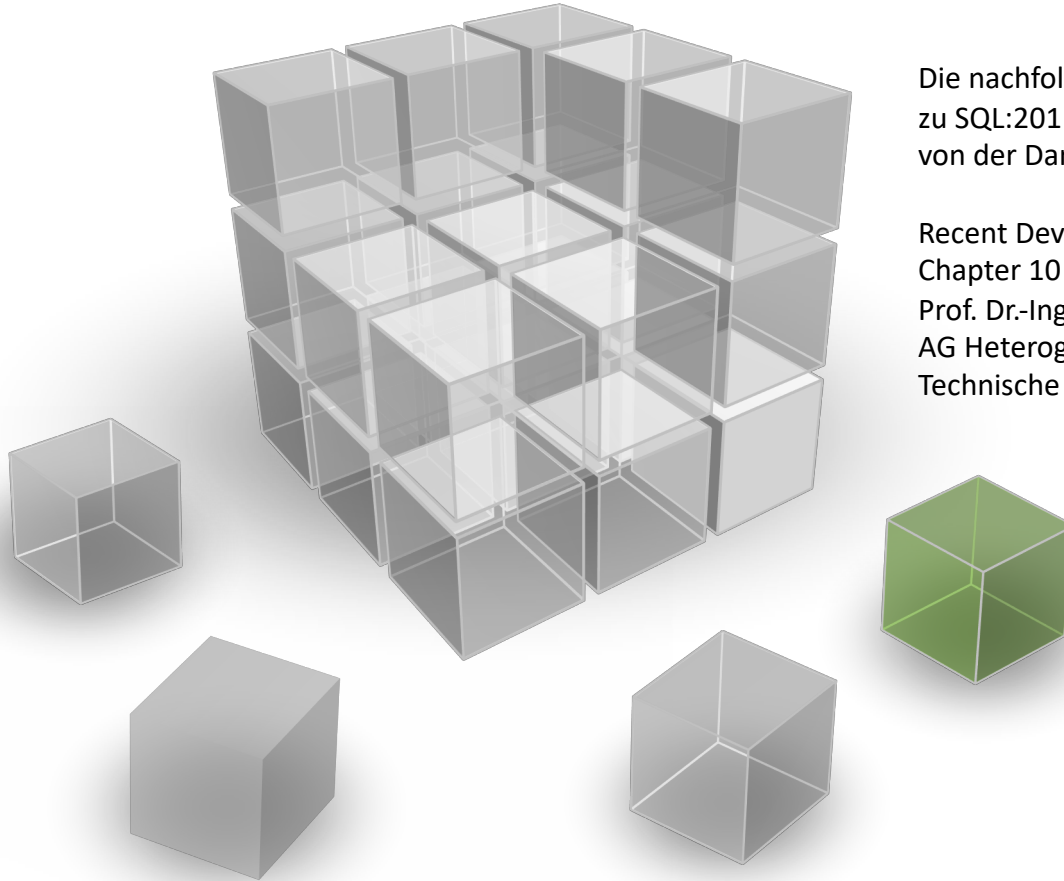
Snodgrass, Richard T., *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann, **1999**

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

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
 - Anfragen beziehen sich auf alle gültigen Tupel
 - Prädikate über Perioden 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 überlappenden 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		DNo	DStart	DEnd	DName
22218	2010-01-01	2011-02-03	3	→	3	2009-01-01	2011-12-31	Test
22218	2011-02-03	2011-11-12	4	→	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)
 - $x \text{ CONTAINS } y \sim (x \text{ contains } y) \text{ or } (x \text{ starts } y) \text{ or } (x \text{ finishes } y) \text{ or } (x \text{ equal } y)$
 - $x \text{ OVERLAPS } y \sim (x \text{ overlaps } y) \text{ or } (y \text{ overlaps } x) \text{ or } (x \text{ contains } y) \text{ or } (y \text{ contains } x) \text{ or } (x \text{ starts } y) \text{ or } (y \text{ starts } x) \text{ or } (x \text{ finishes } y) \text{ or } (y \text{ finishes } x) \text{ or } (x \text{ equals } y)$
 - $x \text{ EQUALS } \sim x \text{ equals } y$
 - $x \text{ PRECEDES } \sim (x \text{ before } y) \text{ or } (x \text{ meets } y)$
 - $x \text{ SUCCEEDS } y \sim (y \text{ before } x) \text{ or } (y \text{ meets } x)$
 - $x \text{ IMMEDIATELY PRECEDES } y \sim x \text{ meets } y$
 - $x \text{ IMMEDIATELY SUCCEEDS } y \sim y \text{ meets } x$

Temporale Verbunde mit Anwendungszeit

ENo	EStart	EEnd	EDept		DNo	DStart	DEnd	DName
22218	2010-01-01	2011-02-03	3	→	3	2009-01-01	2011-12-31	Test
22218	2011-02-03	2011-11-12	4	→	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')

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	9999-12-31 23:59:59	Joe

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
```

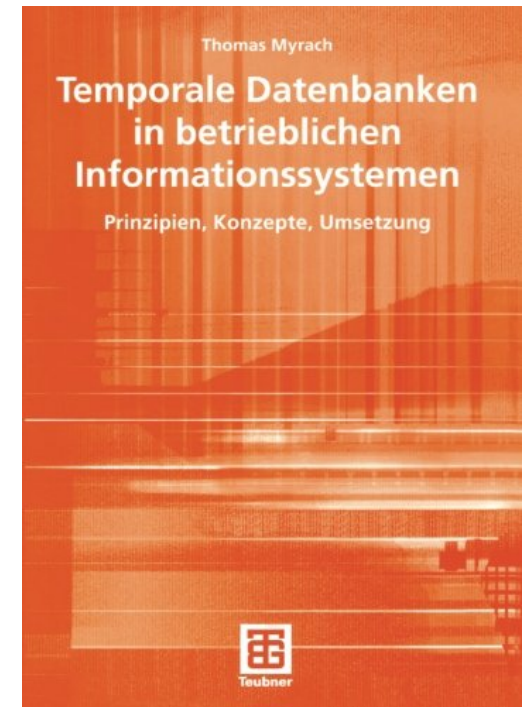
Anfragen an bitemporale Tabellen

- 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 Ansätze¹
 - 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



Non-Standard-Datenbanken

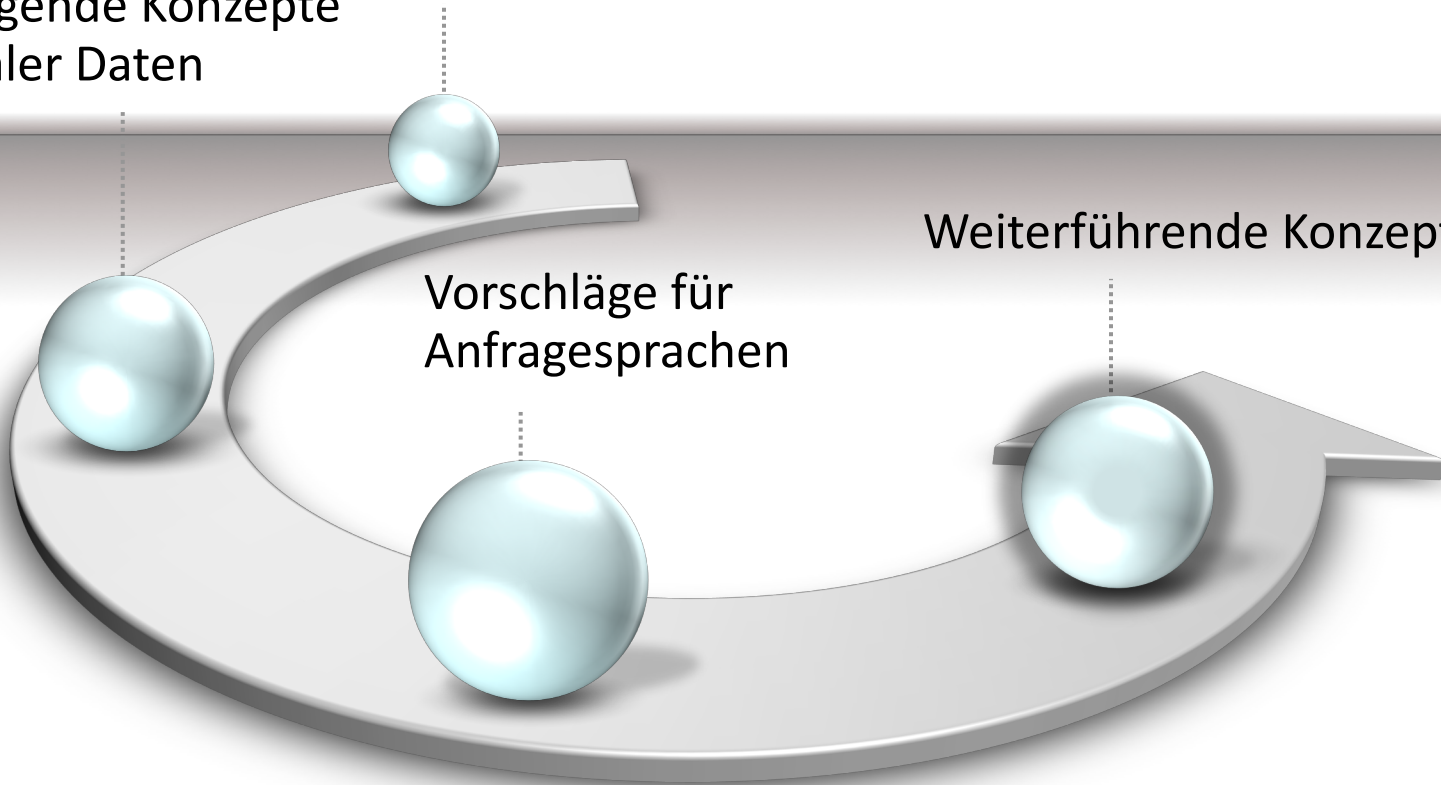
Von der multidimensionalen Indizierung zu temporalen Daten

Grundlegende Konzepte
temporaler Daten

Multidimensionale Indizierung

Vorschläge für
Anfragesprachen

Weiterführende Konzepte



Sequenzen in SQL:2011

- Unterstützung für Datenanalyse in SQL:2011 durch Bildung von Sequenzen aus einer Relation
 - Siehe auch OLAP-Funktionen in SQL:2003
 - 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:

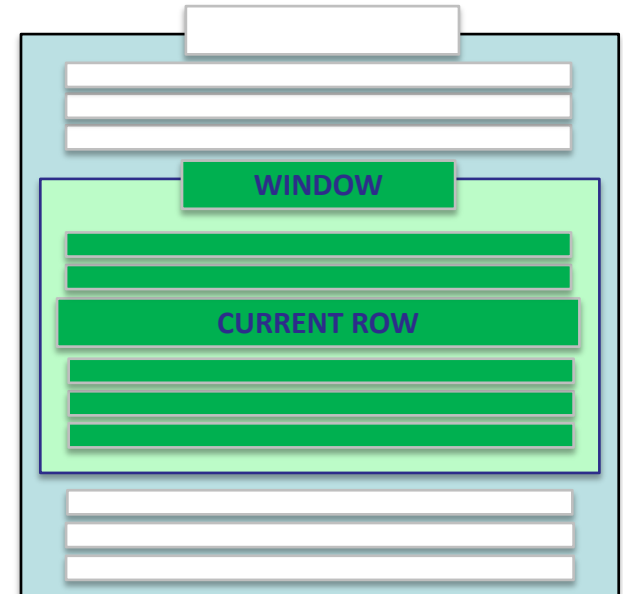
```
SELECT analytical-function(col-expr)
      OVER (window-spec) [AS col-alias]
FROM [TABLE];
```

- 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
	King	24000	62000
100	Kochhar	17000	54200
103	Lorentz	4200	45200

$= 6000 + 12000 + 9000 + 24000$



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
	King	24000	45200
100	Kochhar	17000	21200
103	Lorentz	4200	4200

$54200 = 9000 + 24000 + 17000 + 4200$



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
 - RANK
 - 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