4. Eine temporale relationale Datenbanksprache

Typische Sprachebenen

1. Sprachkonstrukte der Basissprache (etwa SQL); Anforderungen:
   - **abwärtskompatibel**: Syntax und Semantik bleibt bei Anwendung auf nicht-temporale Relationen erhalten.
   - **temporal-abwärtskompatibel**: Syntax bleibt bei Anwendung auf temporale Relationen erhalten, aber Semantik ändert sich: Kommandos beziehen sich nur auf den Datenbestand (Schnappschuss) zum aktuellen Zeitpunkt.
2. **sequentielle** Sprachkonstrukte: Ausführung so als ob Anfrage für jeden Zeitpunkt, d.h. in jedem Schnappschuss, gestellt wird; die Ergebnisse sind aber geeignet zusammenzufassen. Zeitstempel werden nicht explizit benutzt.
3. **nicht-sequentielle** Sprachkonstrukte: erlauben Verknüpfungen von Werten zu verschiedenen Zeitpunkten sowie explizite Zugriffe auf Zeitstempel

Datenbankstruktur

- Es gibt weiterhin auch nicht-temporale Relationen.
- Temporale Relationen seien *hier* wie folgt erweitert:
  - nur Gültigkeitszeit
  - nur eine Granularität
  - Tupel-Zeitstempel `VT` (ValidTime) als zusätzliches Attribut
  - halboffene Intervalle als Zeitstempel
- ausgezeichnete Zeitpunkte:
  - `NOW` (die aktuelle Zeit)
  - `FOREVER` (unbestimmt in der Zukunft)

Beispiel-Datenbank

```
CREATE VALIDTIME COALESCED TABLE ang(
  Nr       NUMBER PRIMARY KEY,
  Name     VARCHAR(20),
  Gehalt   NUMBER,
  Chefnr   NUMBER REFERENCES ang);
```

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	12	[01.03.2012, 01.05.2012)
12	Mueller	5000	27	[01.05.2012, 01.07.2012)
12	Mueller	5500	27	[01.07.2012, 01.10.2012)
12	Mueller	5000	27	[01.10.2012, FOREVER)
13	Meier	5300	13	[01.04.2012, 01.08.2012)
13	Meier	5300	27	[01.08.2012, 01.11.2012)
13	Meier	5400	27	[01.11.2012, FOREVER)
27	Boss	5200	27	[01.05.2012, 01.09.2012)
27	Boss	5700	27	[01.09.2012, FOREVER)

Beispiel-Anfragen in VTSQL2^{+} [1] an temporale Tabellen

1. Temporal abwärtskompatible Anfragen

```
SELECT * FROM ang;
```

liefert, falls `NOW` = 01.12.2012:

Nr	Name	Gehalt	ChefNr
12	Mueller	5000	27
13	Meier	5400	27
27	Boss	5700	27

(nicht-temporale Tupel)

[1] basiert auf:
Beier,F.: Objekt-relationale Realisierung einer temporalen Datenbanksprache. Diplomarbeit, Institut für Informatik, Uni Hannover, 2013
und dort zitierte Literatur, insbesondere zu TSQL2 von R.Snodgrass

2. Sequentielle Anfragen

```
SEQUENCED SELECT name FROM ang WHERE Gehalt>5250;
```

liefert:

Name	VT
Mueller	[01.07.2012, 01.10.2012)
Meier	[01.04.2012, 01.08.2012)
Meier	[01.08.2012, 01.11.2012)
Meier	[01.11.2012, FOREVER)
Boss	[01.09.2012, FOREVER)

(temporale Tupel, noch nicht verschmolzen)

2. Sequentielle Anfragen (Forts.)

```
SEQUENCED COALESCE SELECT name FROM ang WHERE Gehalt>5250;
```

liefert Verschmelzung:

Name	VT
Mueller	[01.07.2012, 01.10.2012)
Meier	[01.04.2012, FOREVER)
Boss	[01.09.2012, FOREVER)

```
SEQUENCED [01.01.2012, 01.09.2012)
    COALESCE SELECT name FROM ang WHERE Gehalt>5250;
```

schränkt das Ergebnis auf das angegebene Intervall ein:

Name	VT
Mueller	[01.07.2012, 01.09.2012)
Meier	[01.04.2012, 01.09.2012)

2. Sequentielle Anfragen (Forts.)

```
SEQUENCED SELECT * FROM ang a, ang b;
```

kombiniert u.a. die temporalen Tupel der Angestellten „Mueller“ und „Meier“; schnappschussweise Semantik führt zu Schnittbildung der VT-Intervalle:

a.Nr	a.Name	a.Gehalt	a.ChefNr	b.Nr	b.Name	b.Gehalt	b.ChefNr	VT
...								
12	Mueller	5000	12	13	Meier	5300	13	[01.04.2012, 01.05.2012)
12	Mueller	5000	27	13	Meier	5300	13	[01.05.2012, 01.07.2012)
12	Mueller	5500	27	13	Meier	5300	13	[01.07.2012, 01.08.2012)
12	Mueller	5500	27	13	Meier	5300	27	[01.08.2012, 01.10.2012)
12	Mueller	5000	27	13	Meier	5300	27	[01.10.2012, 01.11.2012)
12	Mueller	5000	27	13	Meier	5400	27	[01.11.2012, FOREVER)
...								

(temporale Tupel;
brauchen nicht mehr verschmolzen werden, da Ergebnis attributvollständig)

2. Sequentielle Anfragen (Forts.)

```
SEQUENCED COALESCE SELECT Nr, Name, Gehalt FROM ang a
    WHERE NOT EXISTS(
       SELECT * FROM ang b WHERE b.Gehalt > a.Gehalt);
```

wertet die Unteranfrage jeweils bezogen auf den gleichen Zeitpunkt wie die äußere Anfrage aus: → Maximalverdiener:

Nr	Name	Gehalt	VT
12	Mueller	5000	[01.03.2012, 01.04.2012)
12	Mueller	5500	[01.07.2012, 01.09.2012)
13	Meier	5300	[01.04.2012, 01.07.2012)
27	Boss	5700	[01.09.2012, FOREVER)

(temporale Tupel;
verschmolzen, was bei diesen Daten nicht auffällt)

3. Nicht-sequentielle Anfragen

```
NONSEQUENCED SELECT * FROM ang a, ang b;
```

kombiniert alle temporalen Tupel, hier u.a. zu den Angestellten „Mueller“ und „Meier“, unabhängig von den Zeitpunkten, lässt aber Zeitangaben weg: (nicht-temporale Tupel; Duplikate mit `DISTINCT` entfernen)

a.Nr	a.Name	a.Gehalt	a.ChefNr	b.Nr	b.Name	b.Gehalt	b.ChefNr
...							
12	Mueller	5000	12	13	Meier	5300	13
12	Mueller	5000	12	13	Meier	5300	27
12	Mueller	5000	12	13	Meier	5400	27
12	Mueller	5000	27	13	Meier	5300	13
12	Mueller	5000	27	13	Meier	5300	27
12	Mueller	5000	27	13	Meier	5400	27
12	Mueller	5500	27	13	Meier	5300	13
12	Mueller	5500	27	13	Meier	5300	27
12	Mueller	5500	27	13	Meier	5400	27
12	Mueller	5000	27	13	Meier	5300	13
12	Mueller	5000	27	13	Meier	5300	27
12	Mueller	5000	27	13	Meier	5400	27
...							

3. Nicht-sequentielle Anfragen (Forts.)

```
NONSEQUENCED SELECT *, a.VT, b.VT FROM ang a, ang b;
```

fügt Zeitstempel als normale Attribute hinzu, u.a.

a.Nr	a.Name	a.Gehalt	a.ChefNr	b.Nr	b.Name	b.Gehalt	b.ChefNr	a.VT	b.VT
... 12 ...	Mueller	5000	12	13	Meier	5300	13	[01.03.2012, 01.05.2012)	[01.04.2012, 01.08.2012)

Für solche Intervallattribute gibt es [2]

- Vergleichsoperatoren wie u.a. `OVERLAPS(I1,I2)`, `BEFORE(I1,I2)`
- Verknüpfungen wie `INTERSECTION(I1,I2)`, `UNION(I1,I2)`
  (`UNION` nur definiert, falls `OVERLAPS(I1,I2)` oder `MEETS(I1,I2)` gilt).
- Zugriffe auf Start- bzw. Endpunkt mit `BEGIN(I)` bzw. `END(I)` und die Berechnung der Dauer `LENGTH(I)`

Damit wäre auch das sequentielle Produkt (temporaler Verbund) als nicht-sequentielle Anfrage ausdrückbar.

[2] genaue Definitionen siehe Kapitel 5

3. Nicht-sequentielle Anfragen (Forts.)

```
NONSEQUENCED SELECT DISTINCT a.name FROM ang a, ang b
    WHERE a.Nr = b.Nr AND a.Gehalt < b.Gehalt
    AND   BEFORE(a.VT,b.VT);
```

→ die Namen der Angestellten, die jemals eine Gehaltserhöhung erhalten haben (hier: alle)[3]

```
NONSEQUENCED SELECT Nr, Name, Gehalt FROM ang a
    WHERE NOT EXISTS(
       SELECT * FROM ang b WHERE b.Gehalt > a.Gehalt);
```

→ die Angestellten, die irgendwann ein Gehalt hatten, das niemals übertroffen wurde

Nr	Name	Gehalt
27	Boss	5700

[3] `BEFORE(a.VT,b.VT)` ist äquivalent zu $\texttt{END(a.VT)} \leq \texttt{BEGIN(b.VT)}$

4. Mischformen (sequentiell/nichtsequentiell gemischt)

```
SEQUENCED [COALESCE] SELECT Nr, Name, Gehalt FROM ang a
    WHERE NOT EXISTS(
       NONSEQUENCED SELECT * FROM ang b
           WHERE b.Gehalt > a.Gehalt);
```

→ gleiches Ergebnis (nichtsequentielle Unteranfrage wird zeitlich losgelöst ausgewertet), aber mit Zeitstempel (sequentielle äußere Anfrage)

```
NONSEQUENCED SELECT DISTINCT x.Nr, x.Name
  FROM (SEQUENCED COALESCE SELECT a.Nr,a.Name FROM ang a,ang b
           WHERE a.ChefNr=b.Nr AND a.Gehalt>b.Gehalt) x
  WHERE LENGTH(x.VT) > "10 weeks"
```

verwendet eine sequentiell zwischenberechnete temporale Tabelle und wertet deren Zeitstempel (deshalb nichtsequentiell) aus:

→ Wer hat mehr als 10 Wochen am Stück mehr verdient als sein Chef ?

4. Mischformen (Forts.)

Verschmelzungen von Zeitstempeln einer nichtsequentiell berechneten (Zwischenergebnis-) Tabelle lassen sich auch per Gruppierung und Aggregierung nachbilden:

```
NONSEQUENCED SELECT z.Nr, COALESCE(z.intervall) FROM
    (SELECT a.Nr, ..., b.VT intervall FROM a,b
     WHERE ...) z
GROUP BY z.Nr
```

(kann pro Gruppe mehrere Tupel liefern, vgl. Kap. 5 !)

Einfacher wäre die Rückwandlung in eine temporale Tabelle:

```
(NONSEQUENCED SELECT a.Nr, ..., b.VT intervall FROM a,b
     WHERE ...)
AS VALIDTIME COALESCED TABLE WITH TIMESTAMP intervall
```

5. Spezialfall: Aggregierungen

```
SEQUENCED SELECT AVG(Gehalt) FROM ang;
```

→ Durchschnittsgehalt pro Zeitpunkt, mit größten gemeinsamen Teilintervallen, u.a. $(5150, [01.04.2012, 01.05.2012))$, $((5167, [01.05.2012, 01.07.2012))$, ...

```
NONSEQUENCED SELECT AVG(Gehalt) FROM ang;
```

→ Durchschnittsgehalt über 9 Tupel *(wenig hilfreich)*

```
NONSEQUENCED SELECT Nr,AVG(Gehalt) FROM ang GROUP BY Nr;
```

→ Durchschnittsgehalt pro Angestellter, über jew. Tupelanzahl

```
SEQUENCED SELECT Nr,AVG(Gehalt) FROM ang GROUP BY Nr;
```

→ Durchschnittsgehalt pro Angestellter und Zeitpunkt = Ausgangswerte

wünschenswert: nach Zeitdauer (Intervalllänge) gewichtetes Durchschnittsgehalt pro Angestellter; erfordert andere Aggregierungsfunktion:

```
NONSEQUENCED SELECT Nr, WEIGTHED_AVG(Gehalt,LENGTH(VT))
FROM ang GROUP BY Nr;
```

Updates temporaler Tabellen

SEQUENCED- (evtl. mit Intervallangabe) und NONSEQUENCED- Updates wählen solche Zeitpunktinformationen bzw. temporale Tupel zur Änderung aus, die im Sinne von sequentiellen bzw. nichtsequentiellen Anfragen die Bedingung erfüllen, und ändern genau diese.

Bei COALESCED-Tabellen sind evtl. möglich werdende Verschmelzungen eingeschlossen.

Pure Updates an temporalen Tabellen wählen für den Zeitpunkt NOW aus, ändern aber zu den ausgewählten tiks alles für den kompletten Zeitbereich [NOW, FOREVER).

Gegeben sei im folgenden immer wieder:

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.07.2012)
12	Mueller	5500	13	[01.07.2012, 01.10.2012)
12	Mueller	5200	13	[01.10.2012, FOREVER)

Updates (Forts.)

```
SEQUENCED UPDATE ang SET chefnr = 27
  WHERE Gehalt < 5500 AND EXISTS (...);
```

(wobei die EXISTS-Bedingung an eine andere Relation im Zeitraum [01.06.2012, 01.01.2013) erfüllt sei:)

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.06.2012)
12	Mueller	5000	27	[01.06.2012, 01.07.2012)
12	Mueller	5500	13	[01.07.2012, 01.10.2012)
12	Mueller	5200	27	[01.10.2012, 01.01.2013)
12	Mueller	5200	13	[01.01.2013, FOREVER)

Updates (Forts.)

```
SEQUENCED UPDATE ang SET Gehalt = 6000 WHERE chefnr = 13;
```

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	6000	13	[01.03.2012, FOREVER)

```
SEQUENCED [01.05.2012, 01.08.2012) UPDATE ang
SET Gehalt = 6000 WHERE chefnr = 13;
```

ändert höchstens im Zeitraum [1.5.2012, 1.8.2012)

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.05.2012)
12	Mueller	6000	13	[01.05.2012, 01.08.2012)
12	Mueller	5500	13	[01.08.2012, 01.10.2012)
12	Mueller	5200	13	[01.10.2012, FOREVER)

Updates (Forts.)

```
UPDATE ang SET Gehalt=GREATEST(Gehalt,5400) WHERE Nr<16;
```

wegen temporaler Abwärtskompatibilität am 01.11.2012 = NOW:

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.07.2012)
12	Mueller	5500	13	[01.07.2012, 01.10.2012)
12	Mueller	5200	13	[01.10.2012, 01.11.2012)
12	Mueller	5400	13	[01.11.2012, FOREVER)

wegen temporaler Abwärtskompatibilität am 01.06.2012 = NOW:

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.06.2012)
12	Mueller	5400	13	[01.06.2012, FOREVER)

Updates (Forts.)

```
NONSEQUENCED UPDATE ang a
  SET a.Gehalt =
    (SELECT MIN(Gehalt) FROM ang b
     WHERE a.Nr=b.Nr AND
     (EQUALS(b.VT,a.VT) OR AFTER(b.VT,a.VT))
    );
```

ändert ganze temporale Tupel gemäß Bedingung; da die Tabelle temporal (und "coalesced") bleibt, erfolgen auch Verschmelzungen;

das Beispiel soll nicht monoton steigende Gehaltsverläufe rückwirkend korrigieren:

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.07.2012)
12	Mueller	5200	13	[01.07.2012, FOREVER)

Updates (Forts.)

Durch

```
NONSEQUENCED UPDATE ang a
  SET a.VT = TIMESTAMP [01.01.2012, END(a.VT))
  WHERE nr=12 AND BEGIN(a.VT) < 01.07.2012
```

wird der Zeitstempel eines temporalen Tupels geändert, hier nach früher erweitert (sofern im Rahmen der inhärenten IBen erlaubt)

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.01.2012, 01.07.2012)
12	Mueller	5500	13	[01.07.2012, 01.10.2012)
12	Mueller	5000	13	[01.10.2012, FOREVER)

Alternativ könnte hier ein temporales Tupels mit dem zusätzlichen (früheren) Zeitabschnitt eingefügt und implizit verschmolzen werden (s.u.).

Updates (Forts.)

`DELETE` arbeitet ganz analog zu `UPDATE`.

```
DELETE FROM ang WHERE Nr=12;
```

→ (am 16.06.2012=`NOW`)

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 16.06.2012)

```
SEQUENCED DELETE FROM ang WHERE Gehalt<5500 AND ...;
```

(wobei die `...`-Bedingung in [01.06.2012, 01.01.2013) erfüllt sei[4])

→

Nr	Name	Gehalt	ChefNr	VT
12	Mueller	5000	13	[01.03.2012, 01.06.2012)
12	Mueller	5500	13	[01.07.2012, 01.10.2012)
12	Mueller	5200	13	[01.01.2013, FOREVER)

[4] Da eine Lücke entsteht, ist diese Operation nur erlaubt, falls für die Tabelle nicht "COMPLETE" gefordert ist.

Updates (Forts.)

Temporales INSERT benötigt die Angabe temporaler Tupel (Wertekombination plus Zeitstempel), z.B. explizit

```
INSERT INTO ang(
  VALUES(42,'Chef',6400,42) TIMESTAMP [01.06.2012,01.11.2012),
  VALUES(42,'Chef',9600,42) TIMESTAMP [01.11.2012,FOREVER),
  VALUES(12,'Mueller',5000,13) TIMESTAMP [01.01.2012,01.03.2012)
);
```

oder durch passende Anfragen gebildet; kein Zusatz `[NON]SEQUENCED`.
Bei fehlendem Zeitstempel ist wieder `[NOW,FOREVER)` gemeint (Abwärtskompatibilität).

Entsprechend den inhärenten IBen (s. Kap. 3) darf es noch keine Information zum gleichen tik und zu den angegebenen Zeitpunkten geben.

Übliche Ausnahme: `[...,FOREVER)` darf ab `NOW` überschrieben werden.

Updates (Forts.)

Deshalb:

```
INSERT INTO ang VALUES(42,'Chef',10000,13);
```

→ (am 24.12.2012=NOW)

Nr	Name	Gehalt	ChefNr	VT
...				
42	Chef	6400	42	[01.06.2012, 01.11.2012)
42	Chef	9600	42	[01.11.2012, 24.12.2012)
42	Chef	10000	13	[24.12.2012, FOREVER)