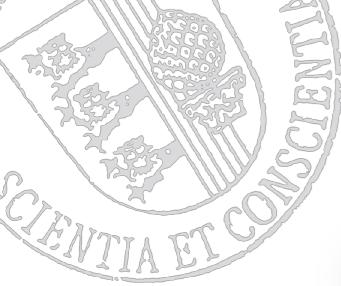


Hinweise

- Dieses Skript beinhaltet evtl. Fehler, die von mir gewollt sind.
- Vermutlich gibt es in diesem Skript auch Fehler, die nicht von mir gewollt waren.
- Manche Folien / Beispiele sind unvollständig. Dies ist Absicht.
- Die Lösungen zu den Beispielen werden in der Vorlesung besprochen.

6. Baumstrukturen in SQL (Teil 1)





Begriffsklärung

Was meinen wir mit “Baumstrukturen”?

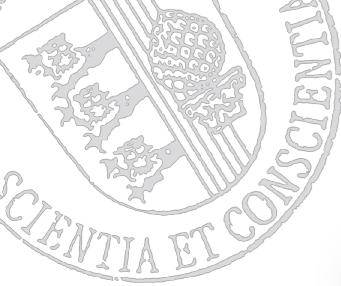


- Binärbäume?
- Rot-Schwarz-Bäume?
- AVL-Bäume?
- B-Bäume?

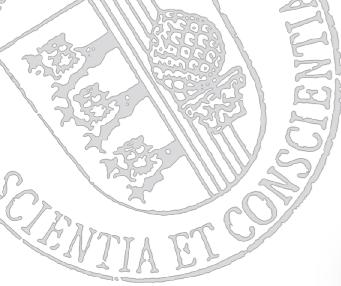
Das sind *Datenstrukturen*, die vielleicht für *Implementierer* von DBMS interessant sind!



Welche Baumstrukturen sind für den *Anwender* von DBMS interessant?



... audience participation time!



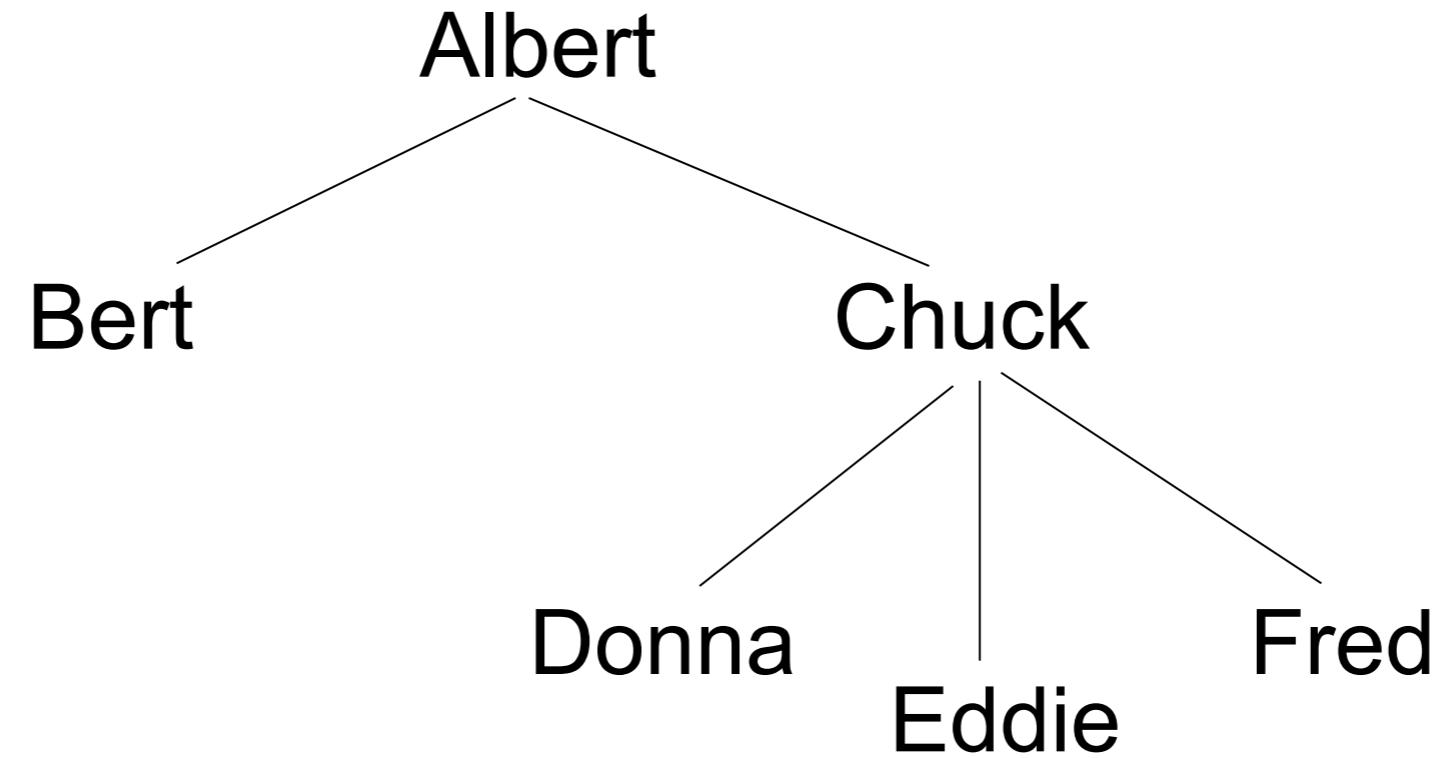
“Vorgänger-Zeiger” Ansatz

Wie kann man Bäume in SQL repräsentieren?

Knoten können als Tupel gespeichert werden!

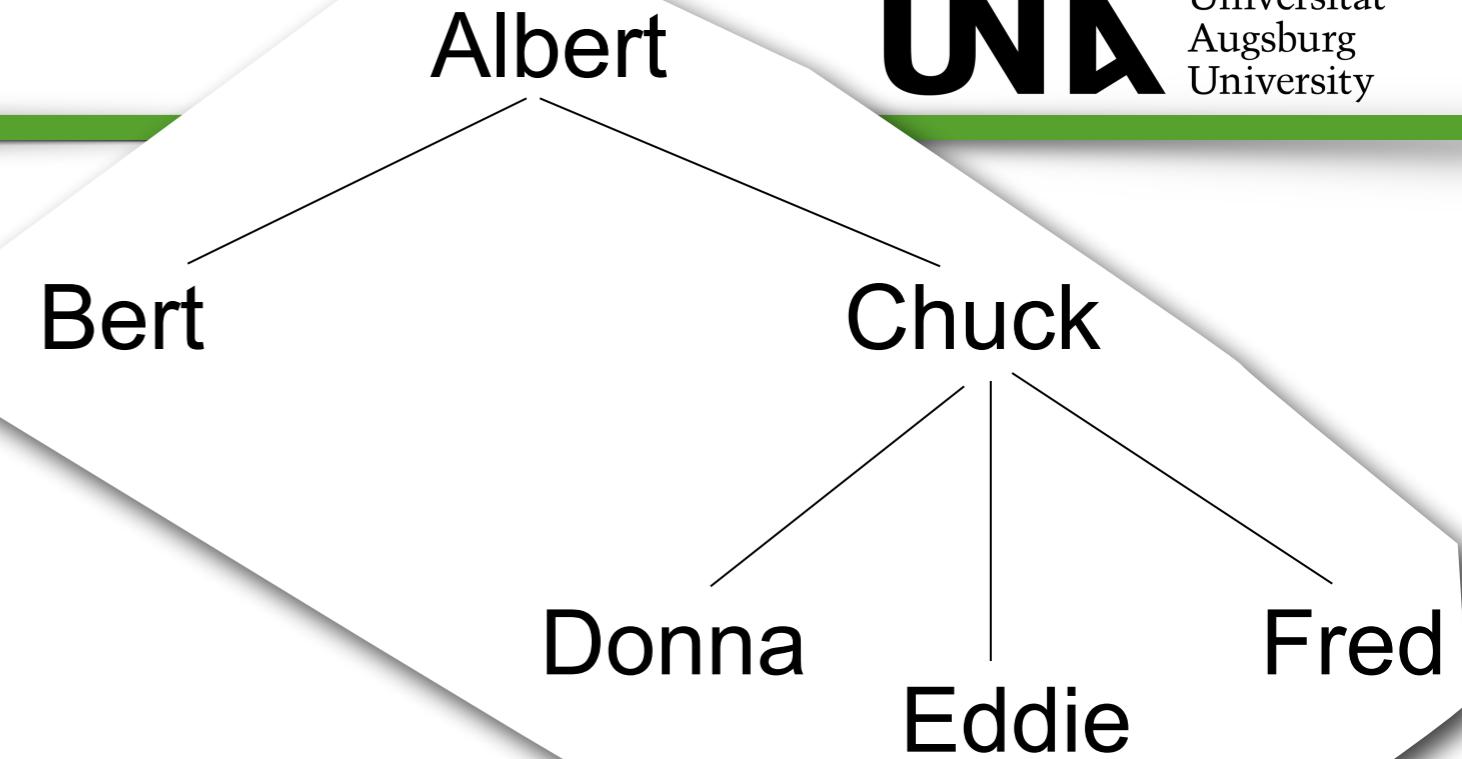
Wie erreicht man die “Verzeigerung”?

Im Baum hat jeder Knoten höchstens einen Vorgänger => also genügt ein extra Feld pro Tupel (entspricht Adjazenzlisten-Modell)



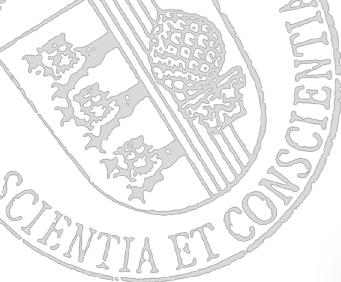
```
CREATE TABLE personal (
    emp VARCHAR(20) PRIMARY KEY,
    boss VARCHAR(20) DEFAULT NULL REFERENCES personal (emp),
    salary DECIMAL(6,2) NOT NULL
);
```

Beispiel



```
CREATE TABLE personal (
    emp VARCHAR(20) PRIMARY KEY,
    boss VARCAHR(20) DEFAULT NULL REFERENCES personal (emp),
    salary DECIMAL(6,2) NOT NULL
);
```

EMP	BOSS	SALARY
Albert	NULL	1000
Bert	Albert	900
Chuck	Albert	900
Donna	Chuck	800
Eddie	Chuck	700
Fred	Chuck	600



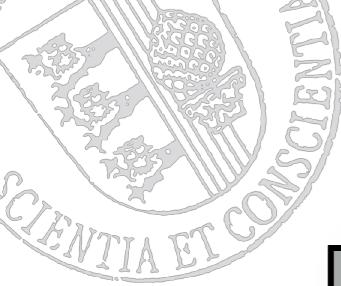
Beispiel

EMP	BOSS	SALARY
Albert	NULL	1000
Bert	Albert	900
Chuck	Albert	900
Donna	Chuck	800
Eddie	Chuck	700
Fred	Chuck	600

```
CREATE TABLE personal (
    emp VARCHAR(20) PRIMARY KEY,
    boss VARCAHR(20) DEFAULT NULL REFERENCES personal (emp),
    salary DECIMAL(6,2) NOT NULL
);
```

Nachteile:

- Jeder Chef muss auch ein Angestellter sein
- Wenn ich Albert umbenenne muss auch das boss-Feld bei allen von Albert abhängigen Angestellten geändert werden
- Der NULL-Wert zur Kennzeichnung der Wurzel kann nicht vermieden werden
- Kindknoten besitzen keine Reihenfolge
- Abfragen sind kompliziert



Abfragen bei Vorgänger-Zeigern

EMP	BOSS	SALARY
Albert	NULL	1000
Bert	Albert	900
Chuck	Albert	900
Donna	Chuck	800
Eddie	Chuck	700
Fred	Chuck	600

Finden den direkten Chef für
jeden Angestellten!

Query ?

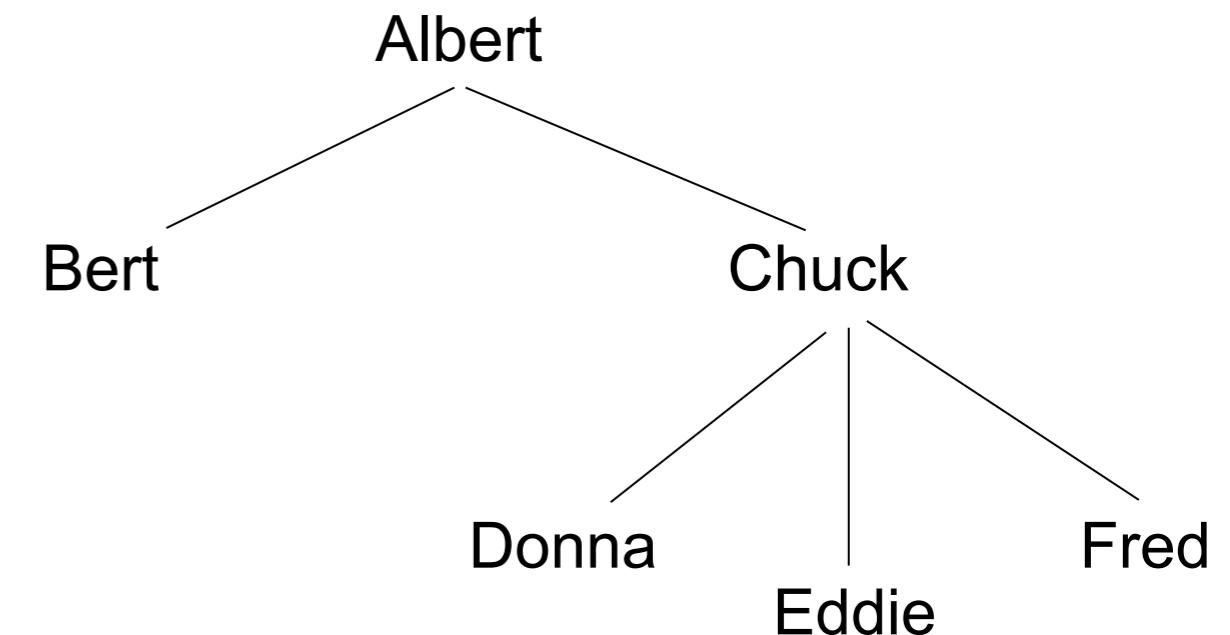
```
SELECT b1.emp, 'ist Chef von', e1.emp
FROM personal b1, personal e1
WHERE b1.emp = e1.boss;
```

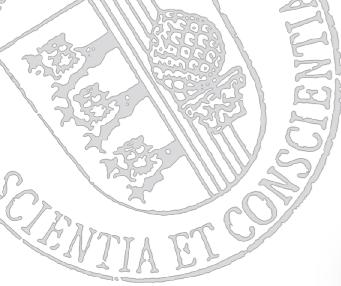
EMP	'IST CHEF VON'	EMP
Albert	ist Chef von	Bert
Albert	ist Chef von	Chuck
Chuck	ist Chef von	Donna
Chuck	ist Chef von	Eddie
Chuck	ist Chef von	Fred

Für **zwei** Ebenen in der Firmenhierarchie!

```
SELECT b1.emp, 'ist Chef vom Chef von ', e2.emp  
FROM personal b1,  
      personal e1,  
      personal e2  
WHERE b1.emp = e1.boss  
AND   e1.emp = e2.boss;
```

Aufwändig, 2 Self-Joins



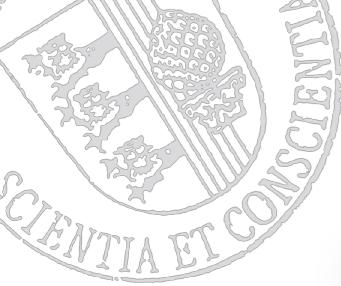


Abfragen mit Vorgänger-Zeigern

Für drei Ebenen in der Firmenhierarchie!

```
SELECT b1.emp, 'ist Chef von Chef von  
Chef von ', e3.emp  
FROM personal b1,  
      personal e1,  
      personal e2,  
      personal e3  
WHERE b1.emp = e1.boss  
AND   e1.emp = e2.boss  
AND   e2.emp = e3.boss;
```

Extrem aufwändig, 3 Self-Joins



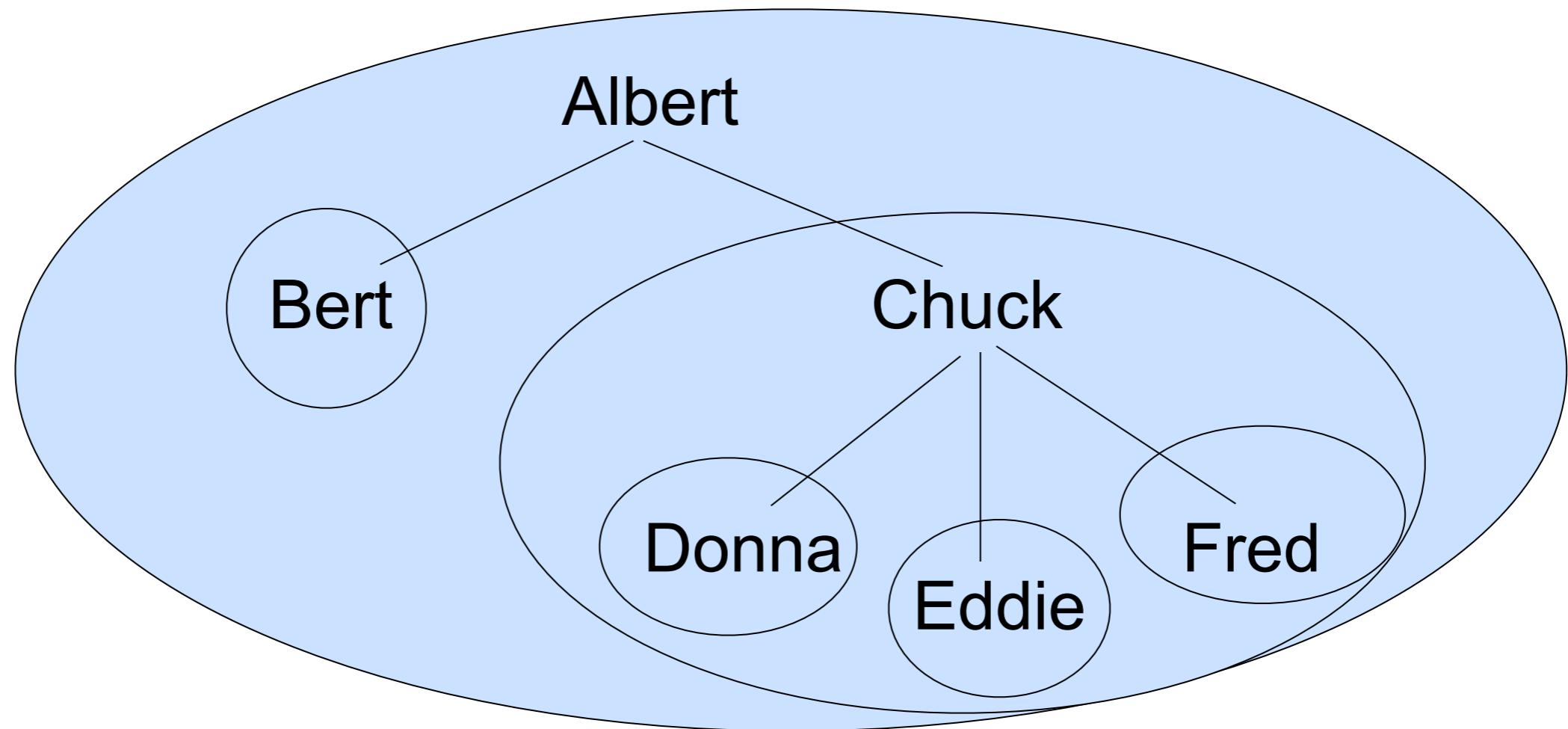
Abfragen mit Vorgänger-Zeigern (3)

```
SELECT b1.emp, 'ist Chef von', e1.emp  
FROM personal b1, personal e1  
WHERE b1.emp = e1.boss;
```

Nachteile:

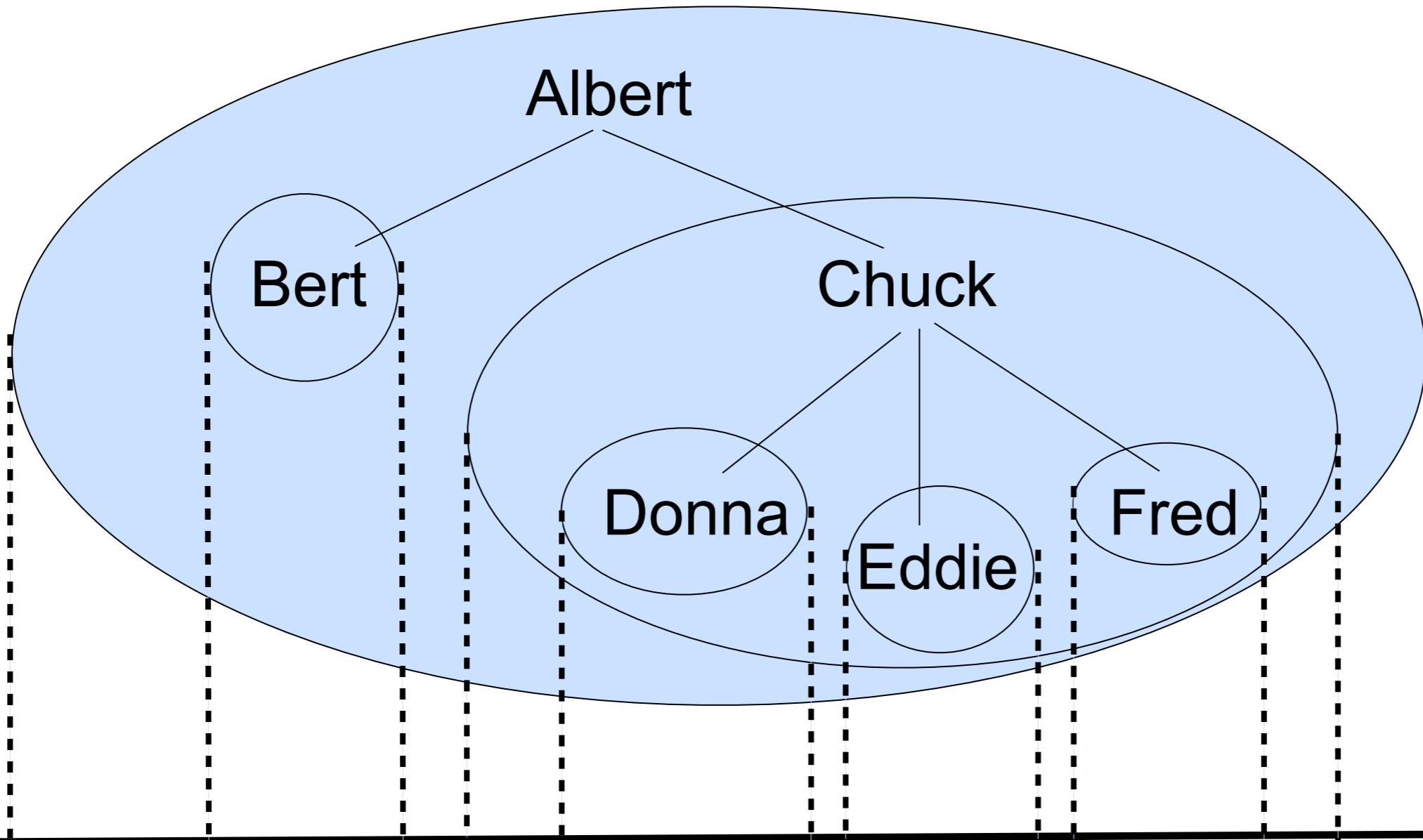
- Nur für eine feste Anzahl Ebenen anwendbar
- Nur unmittelbare Nachfolger-Relation anwendbar
- Iteration über mehrere Ebenen also prozedural zu lösen
- Für “tiefe” Bäume aufwändig und ineffizient

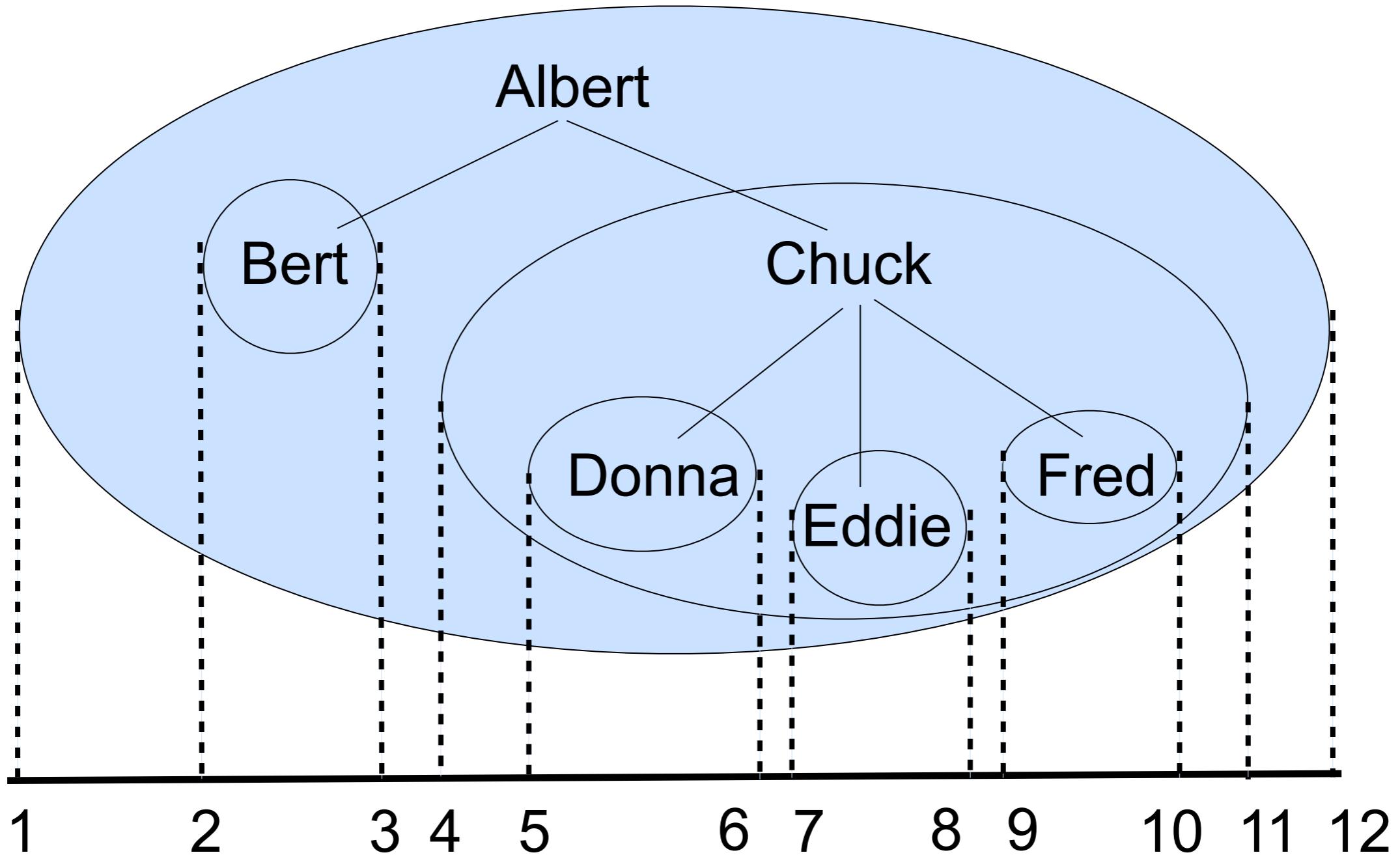
Idee: Betrachte Bäume als verschachtelte Mengen

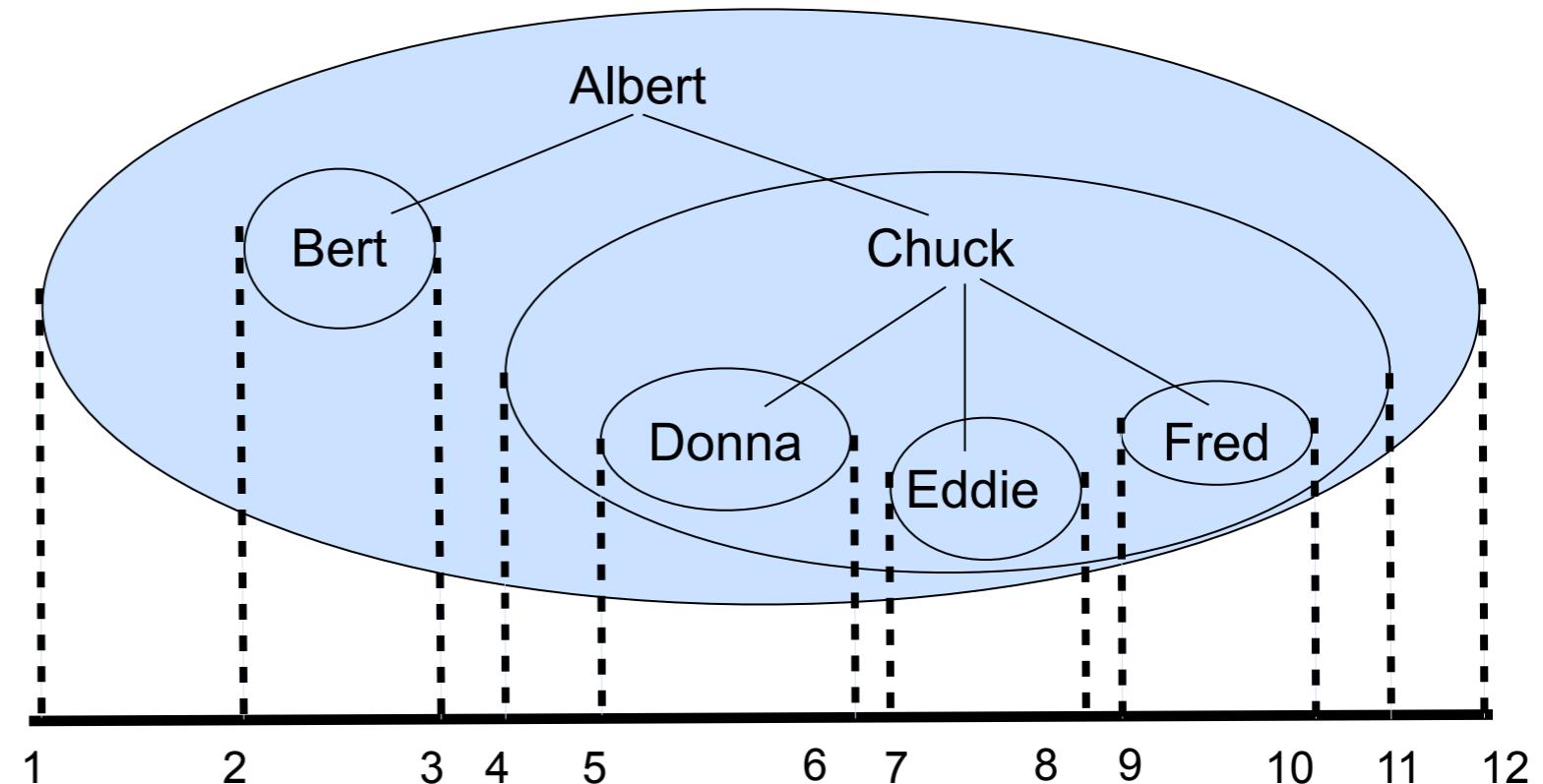


Was hilft das?

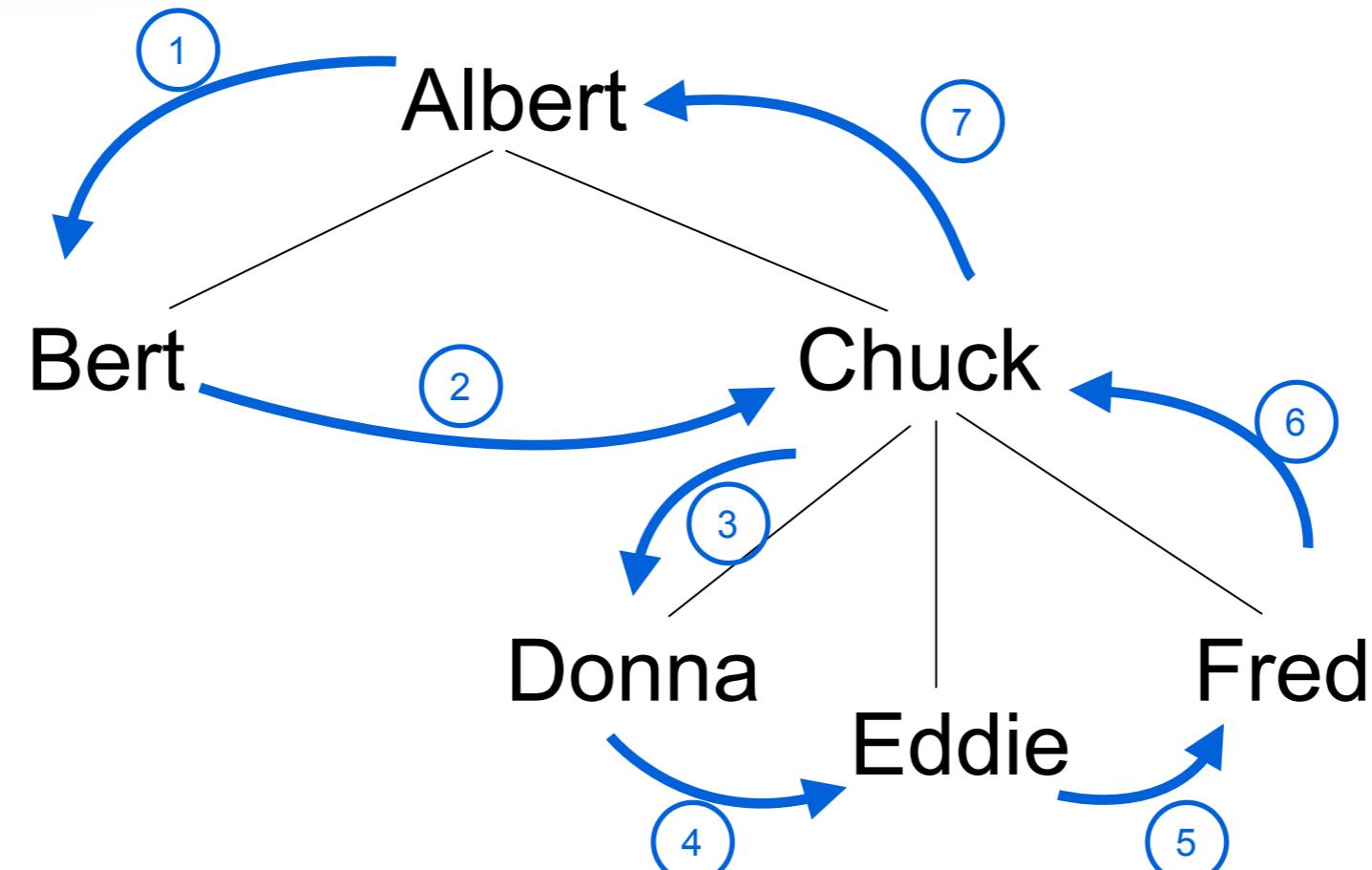
Verschachtelte Mengen lassen sich
effizient durch Intervalle repräsentieren







Knoten	von (LFT)	bis (RGT)
Albert	1	12
Bert	2	3
Chuck	4	11
Donna	5	6
Eddie	7	8
Fred	9	10



Knoten	LEFT	RIGHT
Albert	1	
Bert		
Chuck		
Donna		
Eddie		
Fred		

Knoten	LEFT	RIGHT
Albert	1	
Bert	2	3
Chuck		
Donna		
Eddie		
Fred		

Knoten	LEFT	RIGHT
Albert	1	
Bert	2	3
Chuck	4	
Donna		
Eddie		
Fred		

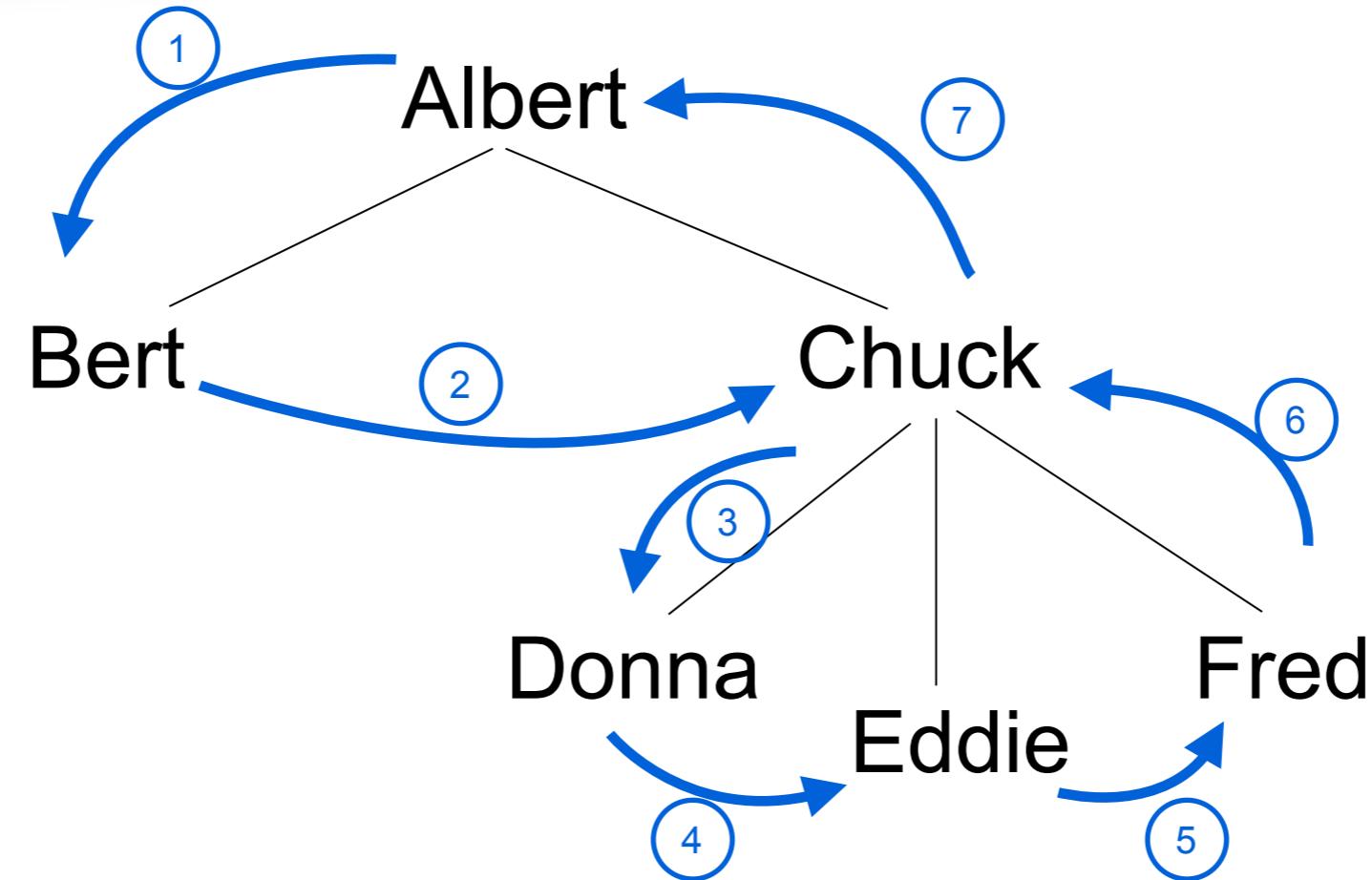
Knoten	LEFT	RIGHT
Albert	1	
Bert	2	3
Chuck	4	
Donna	5	6
Eddie		
Fred		

Knoten	LEFT	RIGHT
Albert	1	
Bert	2	3
Chuck	4	
Donna	5	6
Eddie	7	8
Fred	9	10

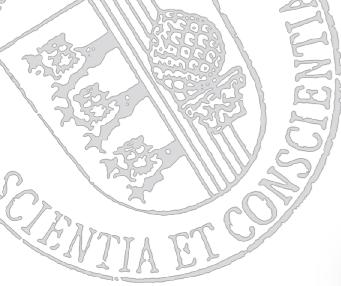
Knoten	LEFT	RIGHT
Albert	1	12
Bert	2	3
Chuck	4	11
Donna	5	6
Eddie	7	8
Fred	9	10

Knoten	LEFT	RIGHT
Albert	1	
Bert	2	3
Chuck	4	11
Donna	5	6
Eddie	7	8
Fred	9	10

Knoten	LEFT	RIGHT
Albert	1	
Bert	2	3
Chuck	4	
Donna	5	6
Eddie	7	8
Fred	9	10



Hinweis: Diese Art den Baum zu durchlaufen
heißt *Depth-First Traversierung*



Nested Sets – Invarianten 1

Die Wurzel hat immer den Wert $LFT = 1$

Da für jeden Knoten 2 Zahlen vergeben werden, gilt:
 $\max(RGT) = 2 \cdot \text{count}(LFT) = 2 \cdot \text{count}(RGT)$

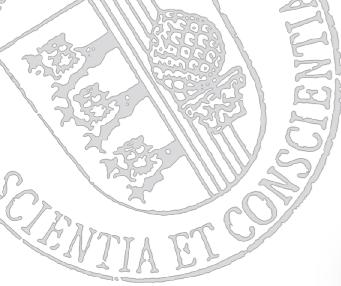
Für alle Knoten n gilt: $n_{RGT} > n_{LFT}$

Für alle Blätter b gilt: $b_{RGT} - b_{LFT} = 1$

Knoten	von (LFT)	bis (RGT)
Albert	1	12
Bert	2	3
Chuck	4	11
Donna	5	6
Eddie	7	8
Fred	9	10

⇒ Blätter mit

SELECT * FROM personal WHERE rgt - lft = 1;
ermitteln



Alle LFT - und RGT -Werte sind eindeutig

LFT und RGT definieren für sich eine totale Ordnung auf dem Baum

Für alle Knoten n gilt:

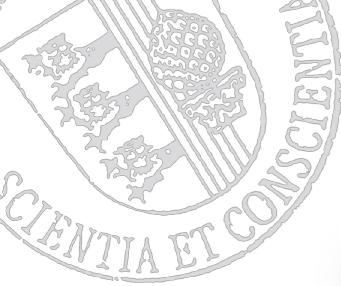
$$\lfloor \frac{n_{RGT} - n_{LFT}}{2} \rfloor = \text{Anzahl der Kind-Knoten}$$

Für die Mengenoperation “ x ist Element von y ” kann die SQL-Operation $BETWEEN$ verwendet werden:

“ x ist Element von y ” genau dann, wenn

$x.lft BETWEEN y.lft AND y.rgt$

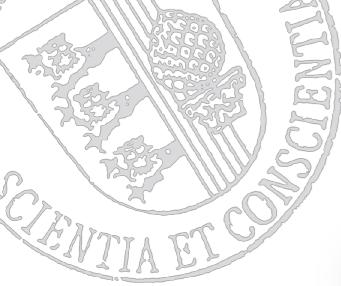
Knoten	von (LFT)	bis (RGt)
Albert	1	12
Bert	2	3
Chuck	4	11
Donna	5	6
Eddie	7	20
Fred	9	10



Nested Sets – DDL

```
CREATE TABLE personal (
    emp VARCHAR(20) PRIMARY KEY,
    salary DECIMAL(6,2) NOT NULL,
    lft INTEGER NOT NULL UNIQUE CHECK (lft > 0),
    rgt INTEGER NOT NULL UNIQUE CHECK (rgt > 1)
);
```

```
ALTER TABLE personal ADD CONSTRAINT orderOKAY
CHECK (lft < rgt);
```



Beachte: Sollen Änderungen an der Baumstruktur vorgenommen werden, so müssen diese *atomar* ablaufen.

- * Sperren der Tabelle oder
- * Transaktionen

Wir betrachten im folgenden:

- * Einen Bruder rechts einfügen
- * Einen Bruder links einfügen
- * Ein Blatt löschen
- * Einen Teilbaum löschen
- * Erweiterung des Baumes um ein Blatt

Nested Sets – Einen Bruder rechts einfügen

Wir fügen ein neues Blatt *rechts* von einem vorhandenen Knoten auf der gleichen Ebene ein:

Albert bekommt einen neuen Angestellten **Gilbert**, den wir zwischen Bert und Chuck einfügen wollen.

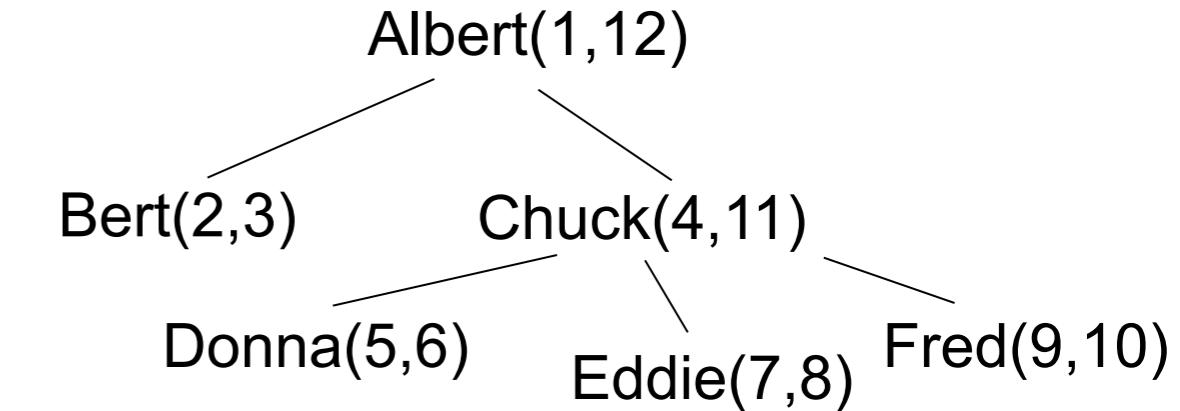
brother_rgt sei der RGT-Wert von Bert

-- Platz für den neuen Knoten

UPDATE personal

SET rgt = rgt+2

WHERE rgt > brother_rgt;



UPDATE personal

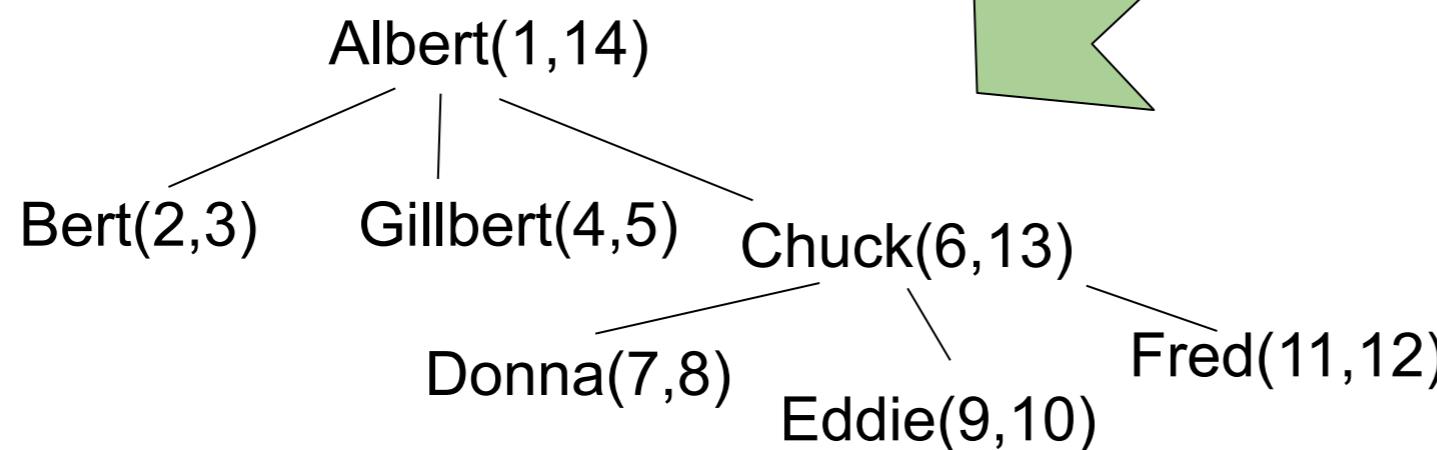
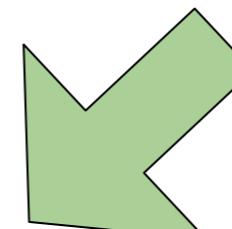
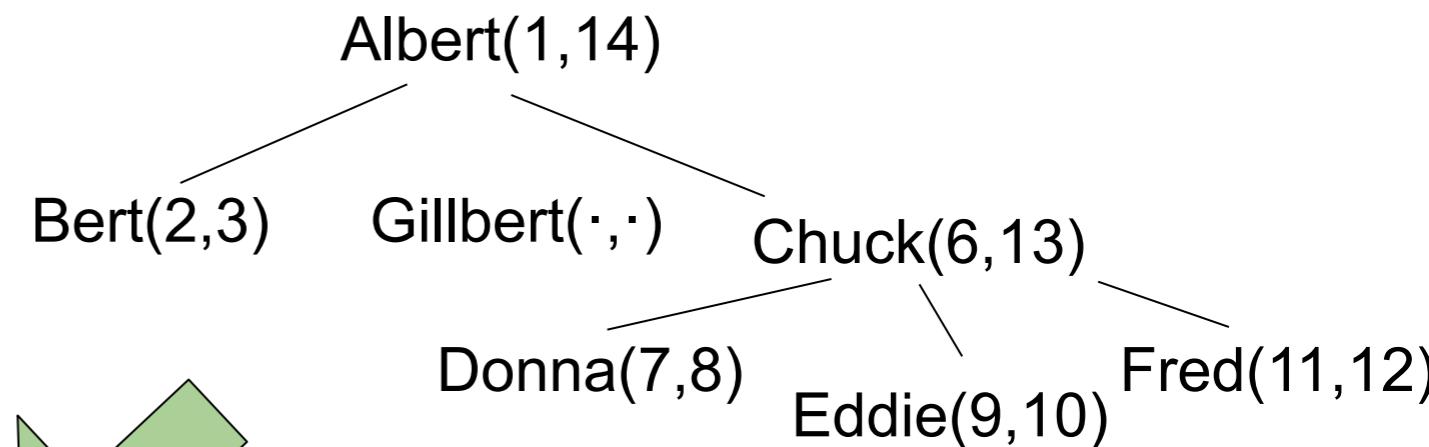
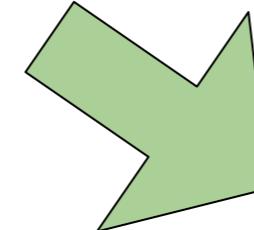
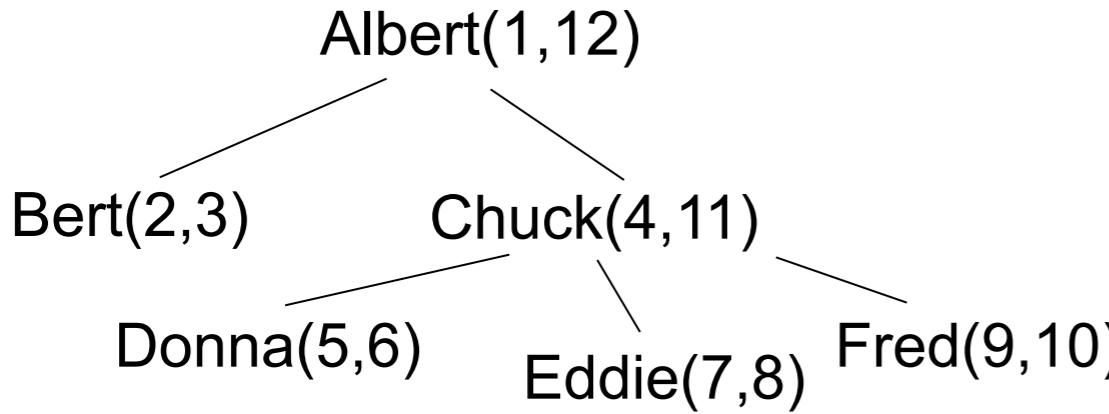
SET lft = lft + 2

WHERE lft > brother_rgt;

-- Einfügen des neuen Mitarbeiters

INSERT INTO personal

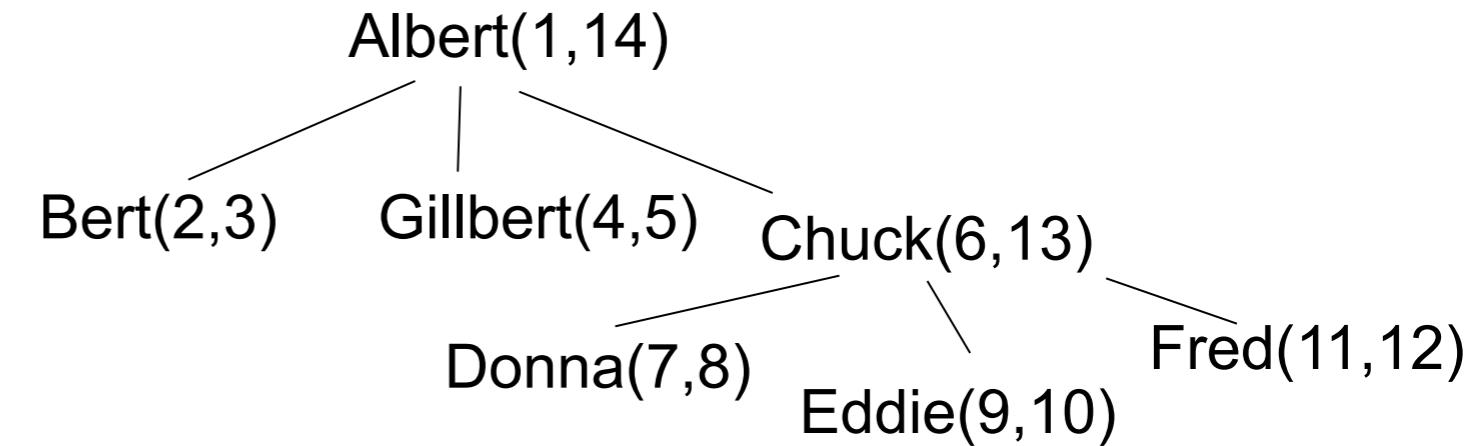
VALUES ('GILBERT', 900, brother_rgt+1, brother_rgt+2);



Wir fügen ein neues Blatt *links* von einem vorhandenen Knoten auf der gleichen Ebene ein:

Chuck bekommt einen neuen Angestellten 'Hans', den wir links vor Donna einfügen wollen.

brother_left = 7 ist der LFT-Wert von Donna.



-- Platz für den neuen Knoten

UPDATE personal

SET rgt = rgt+2

WHERE rgt >= brother_left;

-- Einfügen des neuen Mitarbeiters

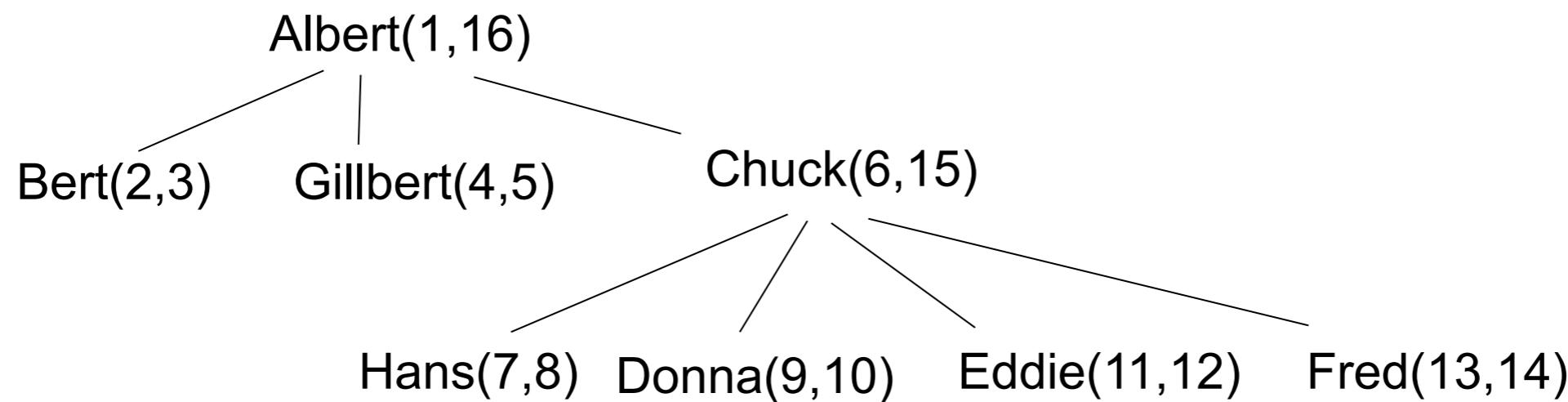
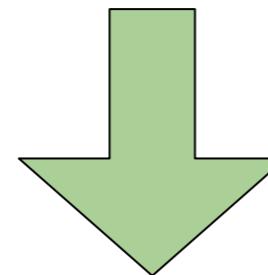
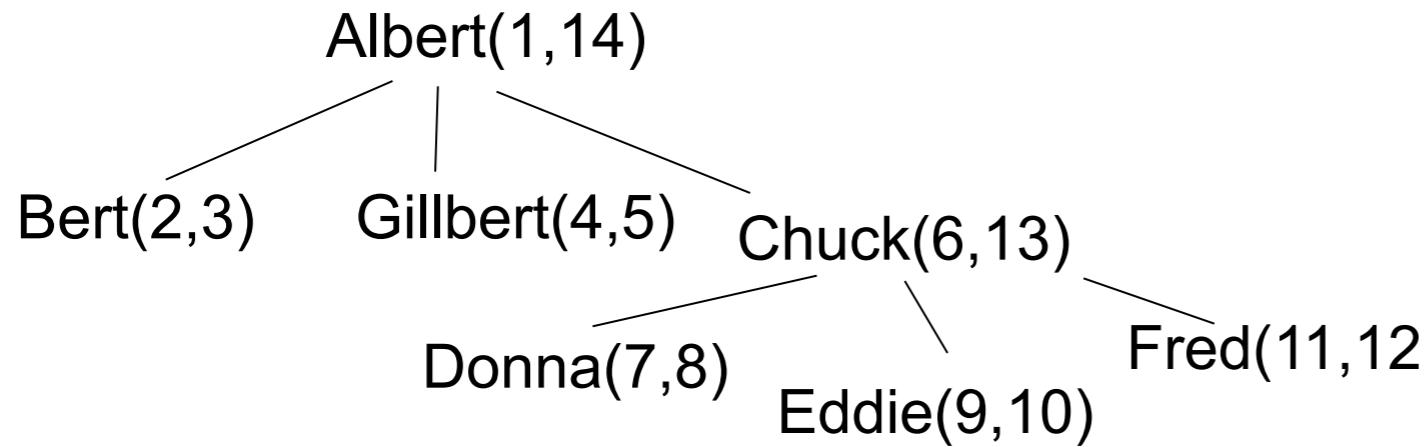
INSERT INTO personal

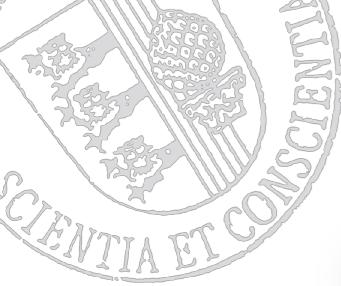
VALUES ('Hans', 850, brother_left, brother_left+1);

UPDATE personal

SET lft = lft + 2

WHERE lft >= brother_left;





Ein Blatt löschen

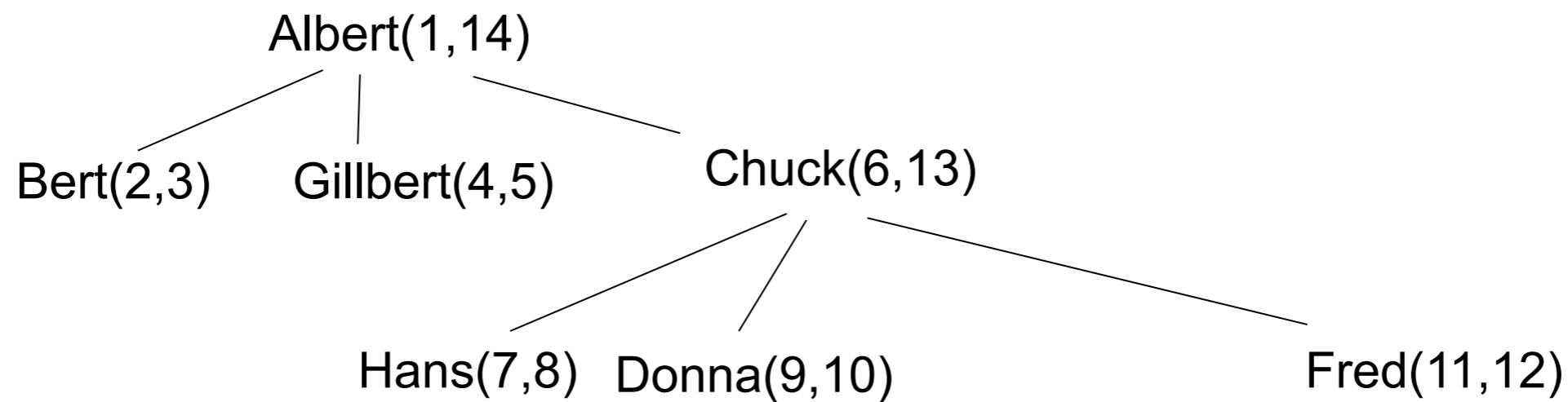
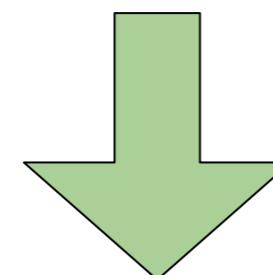
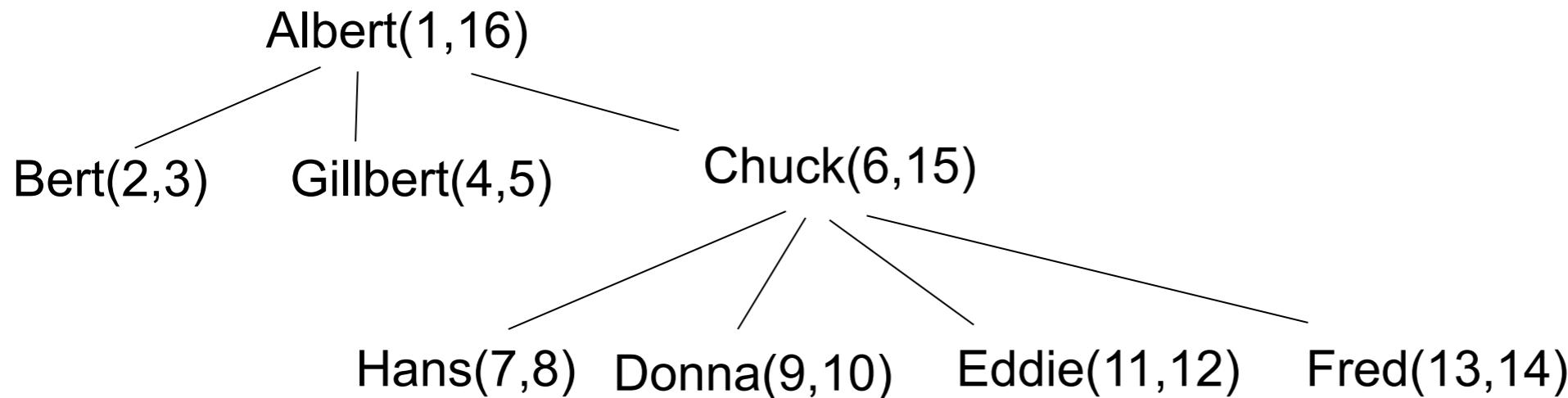
Eddie hat gekündigt. Wir wollen ihn aus der Baumstruktur entfernen. Zum Löschen eines einzelnen Blattes müssen die LFT und RGT-Werte (`v_lft` und `v_rgt`) von Eddie bekannt sein.

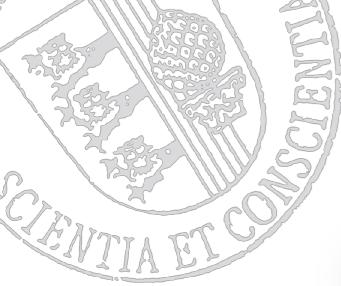
```
-- Eddie rauswerfen  
DELETE FROM personal  
WHERE emp = 'Eddie';
```

```
-- restliche Knoten updaten  
UPDATE personal  
  SET lft = lft - 2  
WHERE lft > v_rgt;
```

```
UPDATE personal  
  SET rgt = rgt - 2  
WHERE rgt > v_lft;
```

Natürlich muss erst geprüft werden, ob der zu löschende Knoten überhaupt ein Blatt ist. (Bedingung $rgt-lft = 1$)





Einen Teilbaum löschen

Analog zum Löschen von Blättern. Der Wert `v_move` gibt an, um wieviel die LFT/RGT-Werte der Folgeknoten vermindert werden müssen. `v_lft` und `v_rgt` seien die LFT- bzw. RGT-Werte des Knotens der den zu löschenen Teilbaum 'unter sich hat'.

1. `v_move := floor((v_rgt - v_lft)/2);`
2. `v_move := 2 * (1 + v_move)` 

Die komplette Abteilung von **Chuck** soll aufgelöst werden:

```
-- Knoten inkl. Kinder löschen
DELETE FROM personal
WHERE lft between v_lft AND v_rgt;
```

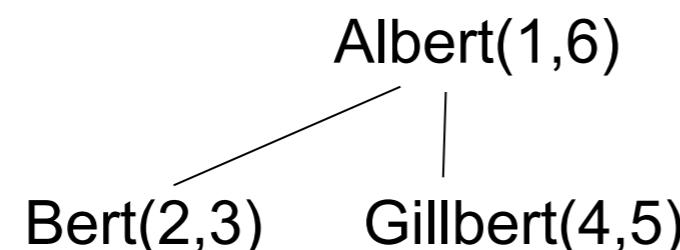
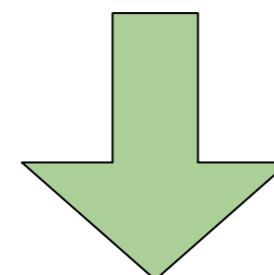
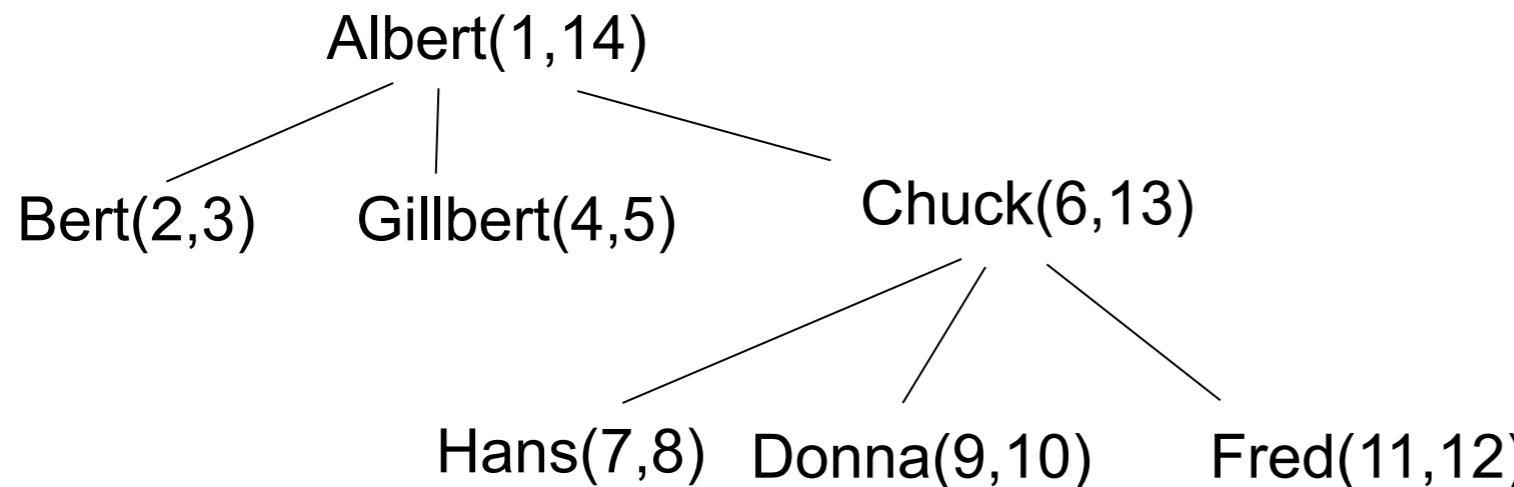
-- restliche Knoten verschieben

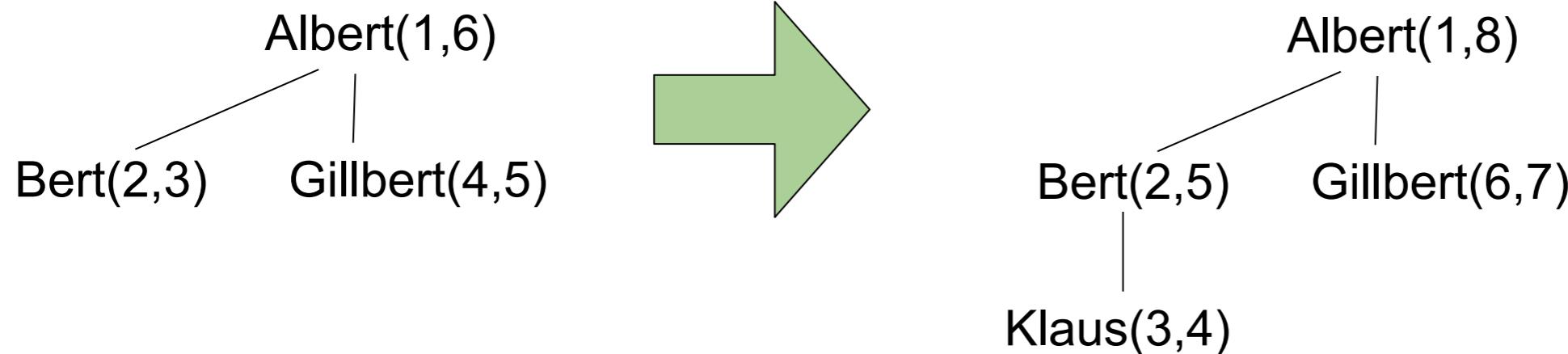
```
UPDATE personal
  SET lft = lft - v_move
 WHERE lft > v_rgt;
```

```
UPDATE personal
  SET rgt = rgt - v_move
 WHERE rgt > v_lft;
```

Anzahl
Kinder
(s. Folie 20)

Beachte: Dies ist eine
Verallgemeinerung des
Löschens von Blättern (Dort
ist `v_move = 2`)





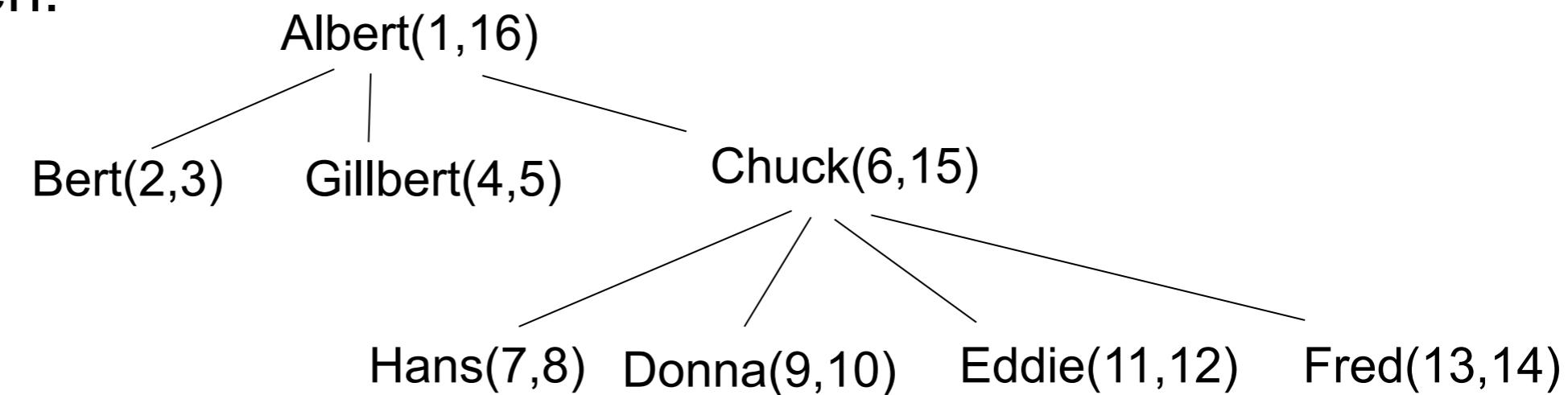
Seien v_{lft} bzw v_{rgt} die LFT- und RGT-Werte von Bert

```
UPDATE personal
  SET lft = lft + 2
WHERE lft > v_rgt;
```

```
UPDATE personal
  SET rgt = rgt + 2
WHERE rgt >= v_rgt;
```

```
INSERT INTO personal VALUES ('KLAUS', 800, v_rgt, v_rgt+1)
```

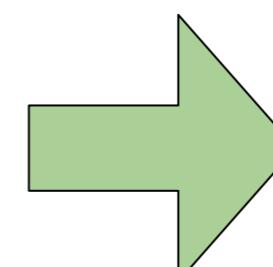
Für die nachfolgenden Anfragen sollen uns folgende Daten zur Verfügung stehen:



EMP	SALARY	LFT	RGT
Albert	1000	1	16
Bert	900	2	3
Chuck	900	6	15
Donna	800	9	10
Eddie	700	11	12
Fred	600	13	14
Gilbert	900	4	5
Hans	850	7	8

* Finde die Mitarbeiter von Chuck

```
SELECT b.emp boss, e.emp emp
FROM personal b, personal e
WHERE e.lft BETWEEN b.lft AND b.rgt
AND e.rgt BETWEEN b.lft AND b.rgt
AND b.emp = 'Chuck' ;
```



EMP	SALARY	LFT	RGT
Albert	1000	1	16
Bert	900	2	3
Chuck	900	6	15
Donna	800	9	10
Eddie	700	11	12
Fred	600	13	14
Gilbert	900	4	5
Hans	850	7	8

BOSS	EMP
Chuck	Hans
Chuck	Donna
Chuck	Eddie
Chuck	Fred
Chuck	Chuck

* Finde alle Chefs von Eddie und die Anzahl derer Mitarbeiter

```
SELECT e.emp emp, b.emp boss,  

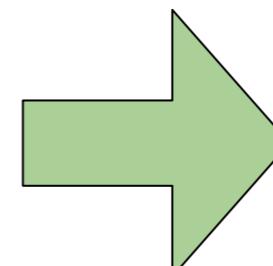
        FLOOR( (b.rgt - b.lft) /2 ) ang  

FROM personal b, personal e  

WHERE e.lft BETWEEN b.lft AND b.rgt  

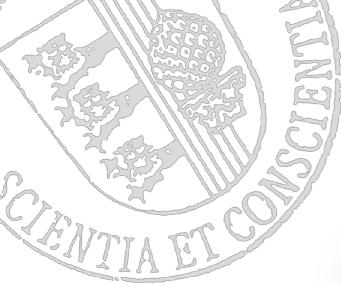
AND e.rgt BETWEEN b.lft AND b.rgt  

AND e.emp = 'Eddie' ;
```

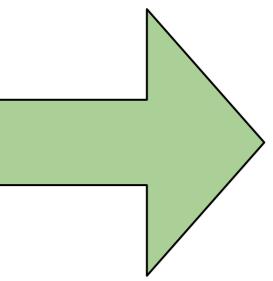


EMP	SALARY	LFT	RGT
Albert	1000	1	16
Bert	900	2	3
Chuck	900	6	15
Donna	800	9	10
Eddie	700	11	12
Fred	600	13	14
Gilbert	900	4	5
Hans	850	7	8

EMP	BOSS	EMP
Eddie	Eddie	0
Eddie	Chuck	4
Eddie	Albert	7



- * Finde die direkten Untergebenen von Chuck
- * Finde die direkten Untergebenen von Albert



Diese Anfragen sind mit dem bisher vorgestellten Modell nicht einfach zu lösen. Deshalb werden bei Nested-Sets häufig die direkten Vorläufer im Baum zusätzlich gespeichert.