Datenbanksysteme

Wintersemester 2015/16

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

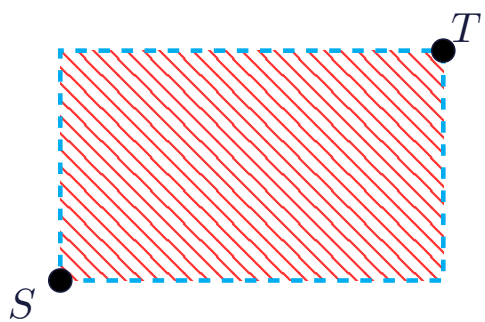
NN Suche im R-Baum

Wir betrachten nun das Problem der Nächsten Nachbar Suche im Falle des R-Baums.

Rechteck im d-dimensionalen Raum

$$R = (S,T)$$

mit $S = [s_1, s_2, \ldots, s_d]$
und $T = [t_1, t_2, \ldots, t_d]$ und $s_i \leq t_i \quad \forall 1 \leq i \leq d$



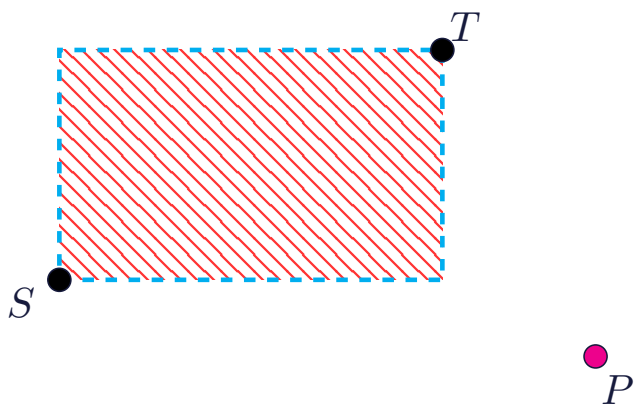
NN Suche im R-Baum

Wir betrachten nun das Problem der Nächsten Nachbar Suche im Falle des R-Baums.

Rechteck im d-dimensionalen Raum

$$R = (S,T)$$

mit $S = [s_1, s_2, \ldots, s_d]$
und $T = [t_1, t_2, \ldots, t_d]$ und $s_i \leq t_i \quad \forall 1 \leq i \leq d$



Was ist die minimale Distanz eines **Punktes** P zu irgendeinem von R umschlossenen Punkt?

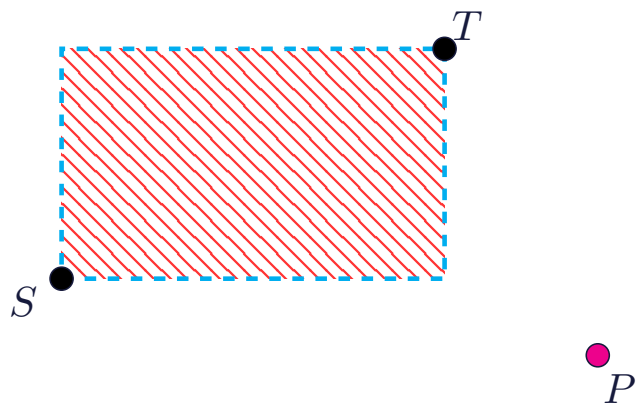
MINDIST

Wir folgen in der Notation dem Papier von N. Roussopoulos et al. Insbesondere, das Distanzmaß hier ist das Quadrat der Eukl. Distanz.

MINDIST(P,R)

Für einen Punkt P und ein Rechteck R definieren wir die Distanz $MINDIST(P,R)$ als

$$MINDIST(P,R) = \sum_{i=1}^{d} |p_i - r_i|^2$$



mit

$$r_i = \begin{cases} s_i & \text{if } p_i < s_i \\ t_i & \text{if } p_i > t_i \\ p_i & \text{sonst} \end{cases}$$

Literatur: N. Roussopoulos, S. Kelley, F. Vincent. Nearest Neighbor Queries. SIGMOD 1995

MINDIST (2)

Die minimale Distanz eines Punktes P zu einem ausgedehnten (spatial) Objekt o, geschrieben als $||(P,o)||$ ist

$$||(P,o)|| = min_{[x_1,...x_d]\in o}(\sum_{i=1}^{d} |p_i - x_i|^2)$$

MBRs und MINDIST

Gegeben ein Punkt P und ein MBR R welches eine Menge von Objekten $O = \{o_i\}$ umfasst. Dann gilt

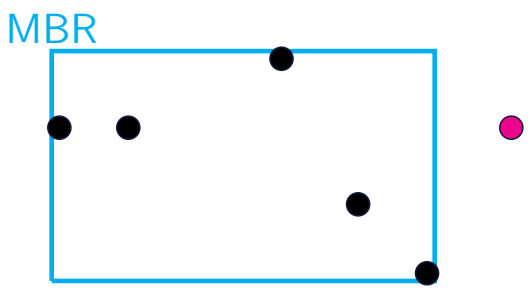
$$\forall o \in O \quad MINDIST(P,R) \leq ||(P,o||$$

MINDIST (3)

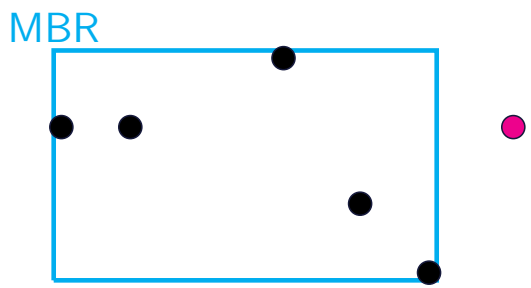
MINDIST wird benutzt, um die Distanz zum am nächsten zu P liegende Objekt in R abzuschätzen.

Suche im R-Baum

- Beim Besuch eines Knotens im R-Baum muss entschieden werden welches MBR als nächstes besucht werden soll.
- Für jedes MBR betrachten wir MINDIST, da dies eine Näherung der Distanz des nächsten Nachbarn ist.
- Wie gut funktioniert das?



MINDIST (4)

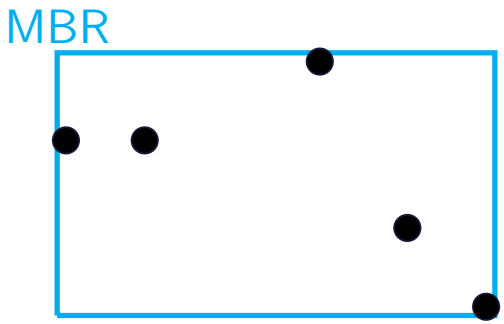


- Schätzung durch MINDIST kann weit daneben liegen
- Was können wir sonst noch über MBRs sagen?

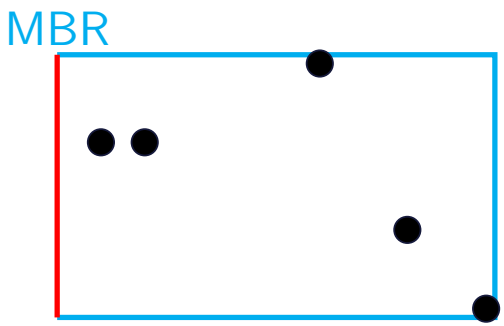
Beobachtung

- Das MBR ist laut Definition das minimale Rechteck, das die enthaltenen Punkte umschließt.
- D.h. auf jeder Kante (für $d = 2$) muss ein Punkt liegen. Für $d = 3$ entsprechend jedes Rechteck, ...
- Wieso? Weil es sonst nicht minimal wäre.
- Rote Kante in Abbildung unten ist z.B. nicht korrekt für ein MBR, müsste "nach rechts verschoben" werden um Rechteck minimal zu machen.

Korrektes MBR (minimal):



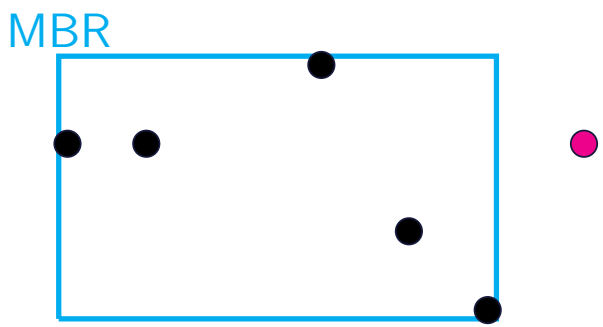
Nicht minimales "MBR":



MINMAXDIST: Idee

Können wir für ein MBR eine obere Schranke für die Distanz des NN Objekts angeben?

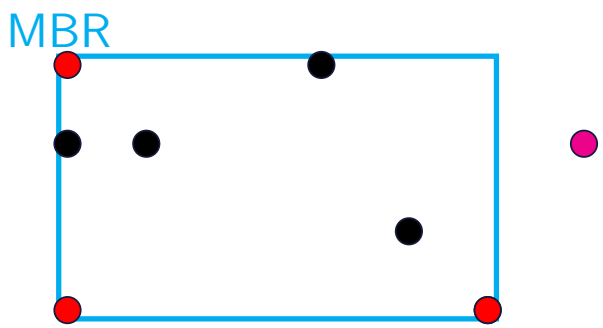
- Wir wissen ja, dass sich auf jeder Kante des MBRs ein Punkt befinden muss.
- Wo liegen diese Punkte maximal weit weg vom Anfragepunkt?



MINMAXDIST: Idee

Können wir für ein MBR eine obere Schranke für die Distanz des NN Objekts angeben?

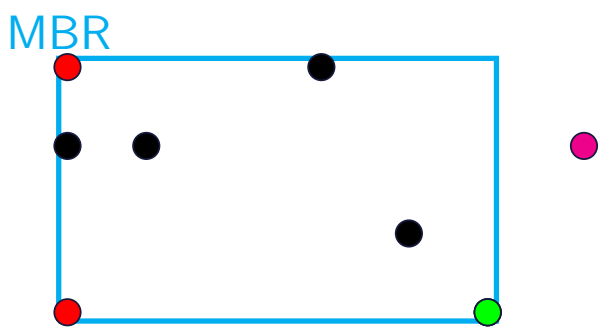
- Wir wissen ja, dass sich auf jeder Kante des MBRs ein Punkt befinden muss.
- Wo liegen diese Punkte maximal weit weg vom Anfragepunkt?



MINMAXDIST: Idee

Können wir für ein MBR eine obere Schranke für die Distanz des NN Objekts angeben?

- Wir wissen ja, dass sich auf jeder Kante des MBRs ein Punkt befinden muss.
- Wo liegen diese Punkte maximal weit weg vom Anfragepunkt?
- Nun betrachten wir den Punkt dieser Punkte mit der kleinsten Distanz zum Anfragepunkt.
- Wieso ist dieser Punkt interessant? Er beschreibt eine obere Schranke für die Distanz des NN. Und zwar die kleinste obere Schranke, die wir angeben können.



MINMAXDIST: Berechnung

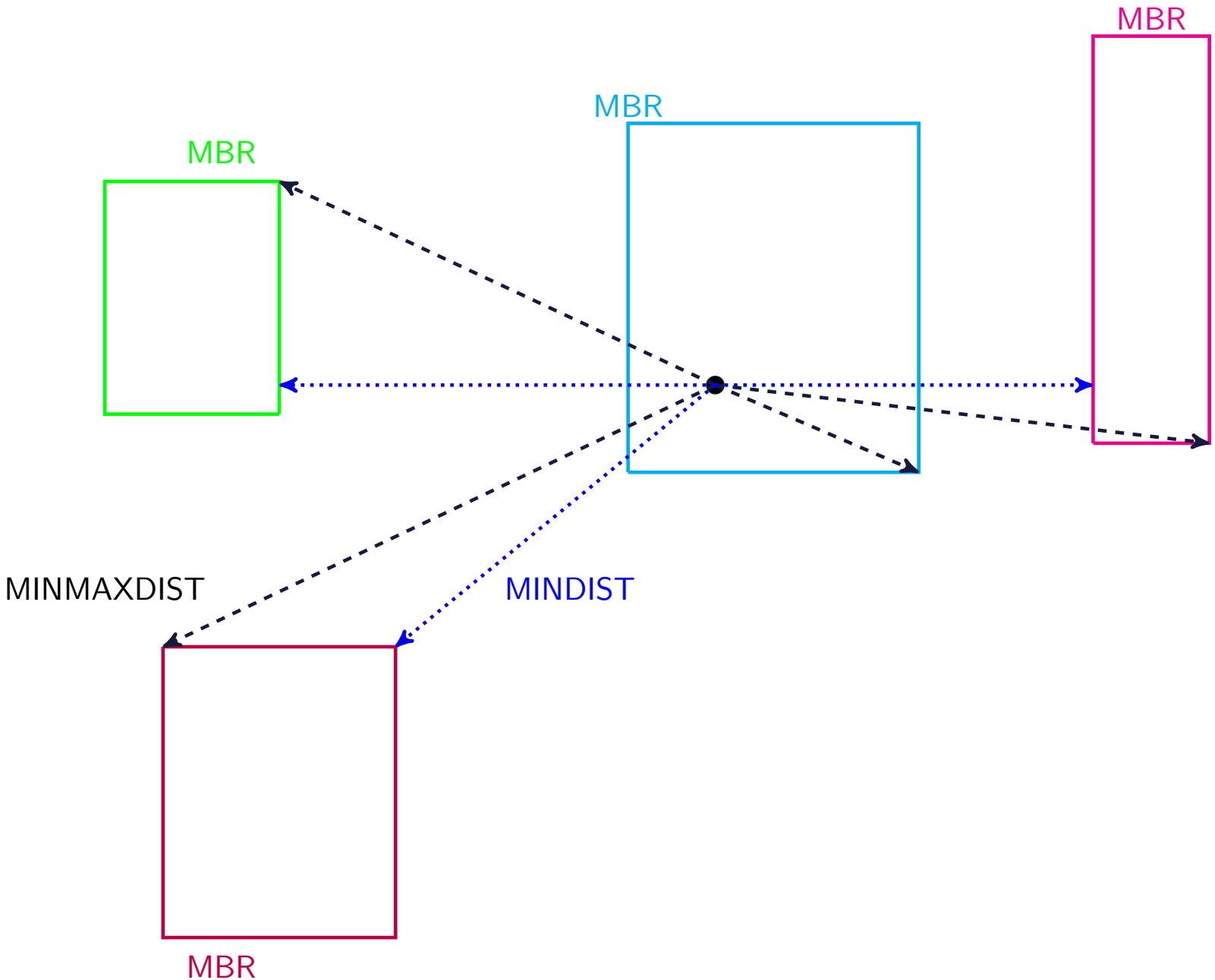
Gegeben ein Punkt P und ein MBR $R = (S,T)$, dann wird **MINMAXDIST(P,R)** berechnet durch

$$\text{MINMAXDIST}(P,R) = min_{1 \leq k \leq d}(|p_k - rm_k|^2 + \sum_{i \neq k, 1 \leq i \leq d} |p_i - rM_i|^2)$$

mit

$$rm_k = \begin{cases} s_k & \text{if } p_k \leq \frac{s_k + t_k}{2} \\ t_k & \text{sonst} \end{cases}$$

$$rM_i = \begin{cases} s_i & \text{if } p_i \geq \frac{s_i + t_i}{2} \\ t_i & \text{sonst} \end{cases}$$


MBR
MBR
MBR
MINMAXDIST
MINDIST
MBR

MINMAXDIST: Interpretation

Gegeben ein Punkt P und ein MBR R welches eine Menge $O = \{o_i\}$ von Objekten umfasst. Dann gilt

$$\exists o \in O \quad ||(P,o)|| \leq MINMAXDIST(P,R)$$

- **Das heißt wir können uns sicher sein, dass es ein Objekt gibt welches auf keinen Fall weiter als MINMAXDIST(P,R) entfernt ist.**

- **Wie können wir die Eigenschaften MINDIST UND MINMAXDIST nun ausnutzen, um einige MBRs gar nicht anschauen zu müssen** (d.h. den Teilbaum nicht betrachten zu müssen)?

Suchraum Pruning

Im Folgenden sind drei Strategien aufgeführt, um die Suche im R-Baum einzuschränken.

Strategie 1

- Ein MBR M mit MINDIST(P, M) größer als die MINMAXDIST(P,M') für ein anderes MBR M' braucht nicht besucht zu werden, da es den NN nicht enthalten kann. **"Downward Pruning"**

Suchraum Pruning (2)

Strategie 2

- Ist die Distanz von P zu einem Objekt O größer als MINMAXDIST(P,M) für ein MBR M, so kann Objekt ignoriert werden, da M ein Objekt O' enthält, das näher an P ist.

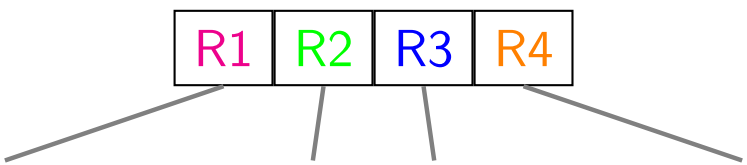
Strategie 3

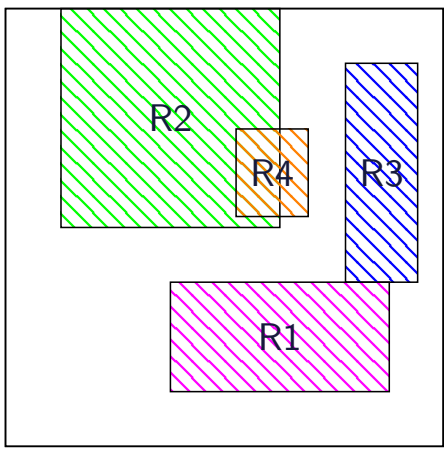
- Jedes MBR M mit MINDIST(P,M) größer als die Distanz von P zu einem Objekt O braucht nicht betrachtet zu werden, da es kein Objekt enthalten kann, das näher an P ist als O. **"Upward Pruning"**

Suche - Ablauf

Die Suche nach dem NN beginnt natürlich an der Wurzel des R-Baums und läuft dann top-down den Baum hinunter.

- Wie in der "normalen" Suche nach überlappenden Rechtecken haben wir hier nun potentiell wieder mehrere Möglichkeiten welche Teilbäume (MBRs!) wir besuchen müssen.
- Wir Suchen hier nach einem Punkt, den nächsten Nachbarn (NN) zum Anfragepunkt
- **Wir sollten hier natürlich so wenig wie möglich MBRs besuchen, aber welche?**





Suchalgorithmus benutzt eine **Prioritäts-Warteschlange** (priority queue). Verschiedene Möglichkeiten der Priorisierung (aka. Sortierung):

Anhand von MINDIST

- MINDIST betrachtet den minimalen Abstand des Anfragepunktes zum MBR
- Dies ist eine **optimistische Priorisierung**. Wie gesehen kann diese Schätzung weit daneben liegen.

Anhand von MINMAXDIST

- MINMAXDIST ist eine **pessimistische Priorisierung**.
- Garantiert immerhin, dass Distanz zu enthaltenen Punkten nicht größer als MINMAXDIST sein kann.

Man kann natürlich Fälle konstruieren wo die eine oder die andere Strategie der Priorisierung besser funktioniert.

Algorithmus: NNSuche

NN.point und NN.dist sind NN bzw. Distanz (initialisiert mit nil und ∞)

Für inneren Knoten (also kein Blatt)

Füge Teilbäume (beschrieben durch MBR) in Queue ein
Sortiere Queue anhand von Priorisierungs-Verfahren
Wende Pruning Strategie 1 und 2 an
while Queue **not empty**
N := Queue.pop
Wende Algorithmus NNSuche rekursiv auf N an
Wende Pruning Strategie 3 an

Für Blattknoten

for each Objekt o **in** Node
dist := $d(q, o)$
if (dist $<$ NN.dist)
NN.dist := dist
NN.point := o

Übersicht Kapitel Indexstrukturen

- Mehrdimensionale Indexstrukturen: Quadtree, PR Quadtree, kd-Tree
- R-Baum, insbesondere NN-Suche im R-Baum
- Indexstrukturen für den metrischen Raum: GH-Tree, Vantage Point Tree, M-Tree
- Hashing: Standardverfahren, Sondierungen bei Überlauf, Erweiterbares Hashing, Lineares Hashing
- Invertierter Index und Top-k Algorithmen
- Skyline Anfragen

Ausblick auf kommende Vorlesungen

- **Weiterführende SQL Konzepte**
  - Fensteranfragen
  - Rekursion in SQL
- **PL/pgSQL**
  - Sprache/Konzept um prozeduralen Code zu schreiben und direkt in der Datenbank (nicht im Client mit Java/JDBC) auszuführen.
- **Trigger**
  - Beschreibe welche Anweisungen ausgeführt werden sollen, wenn bestimmte Bedingung erfüllt ist. Z.B. füge Tupel in Relation Bestellungen ein, wenn ein Produkt im Lager weniger als 10 Mal vorrätig ist.

Zur Erinnerung: Die relationale Uni-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesen von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten			
PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Weiterführende SQL Konzepte

Ranking, Partitionierung und Fensteranfragen

- SQL:2003 Standard
- Ermöglicht das Gruppieren/Partitionieren von Tupeln der Ergebnismenge.
- Und Anwendung einer Aggregat-Funktion auf diese Partitionen
- Z.B. **select** .... , count(*) **over** (**partition by** MatrNr) as vlcount **from** ...

Rekursion

- Beispiel: Gegeben eine Relation `setztVoraus`, in der für eine Vorlesung jeweils die direkten Vorgänger (welche vorausgesetzt werden) angegeben sind
- Wie können dann rekursiv (beliebig tief) Vorlesungen gefunden werden, auf denen eine Vorlesung aufbaut?

```
select s.name, count(*) over
(partition by s.matrnr) as vlcount,
h.vorlnr
from hoeren h, studenten s
where h.matrnr=s.matrnr
order by s.name asc;
```

	name character varying(30)	vlcount bigint	vorlnr integer
1	Carnap	4	5259
2	Carnap	4	5216
3	Carnap	4	5052
4	Carnap	4	5041
5	Feuerbach	2	5001
6	Feuerbach	2	5022
7	Fichte	1	5001
8	Jonas	1	5022
9	Schopenhauer	2	5001
10	Schopenhauer	2	4052
11	Theophrastos	3	5049
12	Theophrastos	3	5041
13	Theophrastos	3	5001

Im Vergleich dazu:

```
select s.name, count(*) as vlcount,
h.vorlnr
from hoeren h, studenten s
where h.matrnr=s.matrnr
group by s.name, h.vorlnr
order by s.name asc;
```

	name character varying(30)	vlcount bigint	vorlnr integer
1	Carnap	1	5041
2	Carnap	1	5052
3	Carnap	1	5216
4	Carnap	1	5259
5	Feuerbach	1	5001
6	Feuerbach	1	5022
7	Fichte	1	5001
8	Jonas	1	5022
9	Schopenhauer	1	4052
10	Schopenhauer	1	5001
11	Theophrastos	1	5001
12	Theophrastos	1	5041
13	Theophrastos	1	5049

Im Vergleich dazu:

```
select s.name, count(*) as vlcount
from hoeren h, studenten s
where h.matrnr=s.matrnr
group by s.name
order by s.name asc;
```

	name character varying(30)	vlcount bigint
1	Carnap	4
2	Feuerbach	2
3	Fichte	1
4	Jonas	1
5	Schopenhauer	2
6	Theophrastos	3

.... mit mehreren Partitionen

```
select s.name, count(*) over (partition by s.matrnr) as vlcount,
count(*) over (partition by h.vorlnr) as studentcount, h.vorlnr
from hoeren h, studenten s
where h.matrnr=s.matrnr
order by s.name asc;
```

- **vlcount:** Anzahl Tupel mit demselben Namen
- **studentcount:** Anzahl Tupel mit derselben VorlNr

	name character varying(30)	vlcount bigint	studentcount bigint	vorlnr integer
1	Carnap	4	1	5052
2	Carnap	4	2	5041
3	Carnap	4	1	5216
4	Carnap	4	1	5259
5	Feuerbach	2	2	5022
6	Feuerbach	2	4	5001
7	Fichte	1	4	5001
8	Jonas	1	2	5022
9	Schopenhauer	2	4	5001
10	Schopenhauer	2	1	4052
11	Theophrastos	3	1	5049
12	Theophrastos	3	2	5041
13	Theophrastos	3	4	5001

rank()

```
select s.name, count(*) over (partition by s.matrnr) as vlcount,
rank() over (partition by s.name order by h.vorlnr) as rank, h.vorlnr
from hoeren h, studenten s
where h.matrnr=s.matrnr
order by s.name asc;
```

- **rank:** Position in Gruppe wenn partitioniert nach s.name

	name character varying(30)	vlcount bigint	rank bigint	vorlnr integer
1	Carnap	4	1	5041
2	Carnap	4	2	5052
3	Carnap	4	3	5216
4	Carnap	4	4	5259
5	Feuerbach	2	1	5001
6	Feuerbach	2	2	5022
7	Fichte	1	1	5001
8	Jonas	1	1	5022
9	Schopenhauer	2	1	4052
10	Schopenhauer	2	2	5001
11	Theophrastos	3	1	5001
12	Theophrastos	3	2	5041
13	Theophrastos	3	3	5049

rank() vs. dense_rank()

Berechne den Rang von Studenten basierend auf GPA.

select ID, **rank() over (order by GPA desc)** as student_rank
from student_grades
order by student_rank;

- Was passiert, wenn zwei Studenten den gleichen GPA haben?
- Z.B. wenn bei Studenten den höchsten GPA haben bekommen beide Rang 1
- die nächsten Studenten bekommen dann Rang 3, etc.
- Wenn es drei Studenten geben würde mit dem zweithöchsten GPA, dann würde der nächste zu vergebene Rang 6 sein.
- Bei Funktion **dense_rank** gibt es keine fehlenden Ränge, d.h. die Tupel mit dem zweithöchsten Wert würden Rang 2 bekommen, etc.

Beispiel ohne rank() berechnet

Die Anfrage von zuvor kann auch wie folgt ausgedrückt werden:

```
select ID, (1+(select count(*)
               from student_grades B
               where B.GPA > A.GPA)) as student_rank
from student_grades A
order by student_rank;
```

- **Die naive Ausführung dieser Anfrage hat quadratischen Aufwand: Für jeden Studenten wird der Rang berechnet durch linearen Aufwand** (laufen über alle anderen Studenten).

- **In der vorhergehenden Notation mit rank() kann das Datenbanksystem geschickter vorgehen**: Relation sortieren und den Rang einfach berechnen.

window frames

select h.matrnr, count(h.matrnr) over ()
from hören h;

- keine Partitionierung
- alle Werte gleich

	matrnr integer	count bigint
1	26120	13
2	27550	13
3	27550	13
4	28106	13
5	28106	13
6	28106	13
7	28106	13
8	29120	13
9	29120	13
10	29120	13
11	29555	13
12	25403	13
13	29555	13

select h.matrnr, count(h.matrnr) **over (order by** h.matrnr)
from hören h;

- keine Partitionierung
- ABER: **laufendes Fenster**
- Fenster enthält
  - alle Werte bis hierhin
  - einschließlich derjenigen mit **derselben** matrnr

	matrnr integer	count bigint
1	25403	1
2	26120	2
3	27550	4
4	27550	4
5	28106	8
6	28106	8
7	28106	8
8	28106	8
9	29120	11
10	29120	11
11	29120	11
12	29555	13
13	29555	13

Regeln für Fensterfunktion

- **Nur im SELECT und ORDER BY erlaubt**
- Nicht in **group by**, **having** und **where**
- Warum? Ausführung der Fensterfunktionen passiert logisch nach diesen Statements
- **kann mit normaler Aggregation kombiniert werden:**

select h.matrnr, count(*) as countstar, count(*) over (order by h.matrnr) as countpartition
from hören h
group by h.matrnr;

- **countpartition:** laufendes Fenster über Ergebnis der Gruppierung!

	matrnr integer	countstar bigint	countpartition bigint
1	25403	1	1
2	26120	1	2
3	27550	2	3
4	28106	4	4
5	29120	3	5
6	29555	2	6

select h.matrnr, count(*) as countstar, count(*) over () as countpartition
from hören h
group by h.matrnr;

- keine Partitionierung!
- kein laufendes Fenster!

Auswertungsreihenfolge:

- erst **where**
- dann **group by**
- dann **having**
- dann **Fenster**

Angenommen wir haben eine Tabelle tot_credits, in der die Summe aller von Studenten erreichten Credit Points stehen, pro Jahr, also ein Eintrag pro Jahr.

```
select year, avg(num_credits)
             over (order by year rows 3 preceding)
             as avg_total_credits
from tot_credits;
```

- Diese Anfrage berechnet die durchschnittliche Anzahl von Credits über je 3 Tupel, in der durch order by angegebenen Reihenfolge.
- Wir haben also z.B. für das Jahr 2009 einen Durchschnitt, der über die Jahre 2007, 2008 und 2009 berechnet wurde.
- Ähnlich mit Angaben von following:

```
select year, avg(num_credits)
             over (order by year rows
                 between 3 preceding and  2  following)
             as avg_total_credits
from tot_credits;
```

Bereichsbasierte Fenster-Spezifikation

- Die Fenster zuvor waren spezifiziert durch Anzahl der Tupel (z.B. 3 preceeding and 2 following)
- Nun: Fenster-Spezifikation berücksichtigt Bereich (Range) von Daten

```
select year, avg(num_credits)
             over (order by year range between year-4 and year)
             as avg_total_credits
from tot_credits;
```

Bereichsbasierte Fenster-Spezifikation

- Angenommen wir haben nun eine Relation tot_credits_depts(dept_name, year, num_credits)
- Also wie zuvor bloß per Department
- Dann können wir nun noch anhand von Department partitionieren:

```
select dept_name,year, avg(num_credits)
                    over (partition by dept_name
                    order by year range between year-4 and year)
                    as avg_total_credits
from tot_credits_dept;
```

Hinweis: Einige Beispiele sind zum Teil direkt übernommen aus dem Buch "Datebase System Concepts" von Silberschatz, Korth, Sudarshan.

Übersicht Syntax (in Postgresql)

http://www.postgresql.org/docs/9.3/static/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS

```
function_name ([expression [, expression ... ]]) OVER window_name
function_name ([expression [, expression ... ]]) OVER ( window_definition )
function_name ( * ) OVER window_name
function_name ( * ) OVER ( window_definition )
```

where window_definition has the syntax

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS
                                                { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

and the optional frame_clause can be one of

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

where frame_start and frame_end can be one of

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```