

# Datenbankoptimierung

## Lehrveranstaltung Datenbanktechnologien

Prof. Dr. Ingo Claßen   Prof. Dr. Martin Kempa

Hochschule für Technik und Wirtschaft Berlin

Hardware

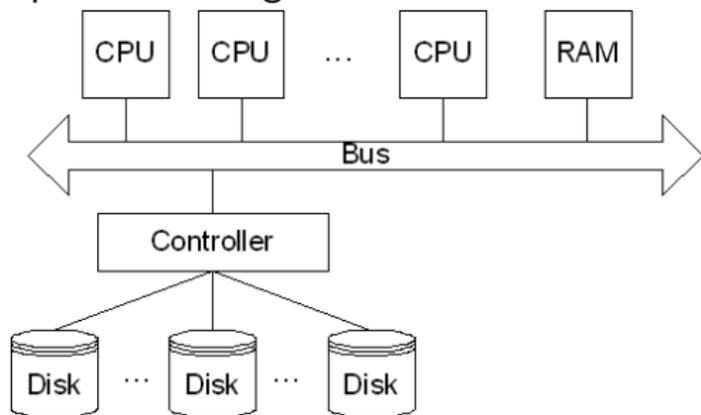
Partitionierung und Materialisierung

Indizes

Abfragebearbeitung

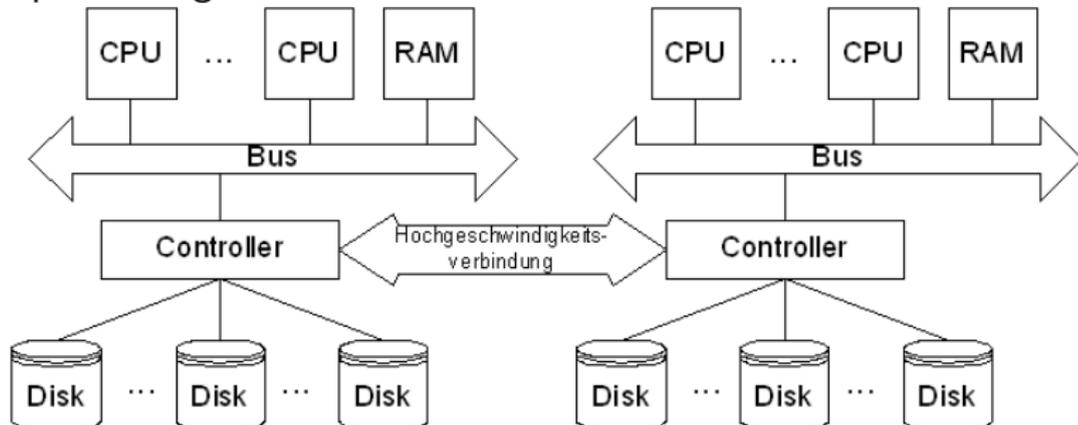
## Architektur: Shared Memory (Symmetrische Multiprocessing, SMP)

- ▶ Alle CPUs teilen sich einen gemeinsamen Speicher
- ▶ Gemeinsamer Zugriff auf Sekundärspeicher
- ▶ Schlechte Skalierbarkeit, Bus ist Flaschenhals
- ▶ Einfache Struktur: keine verteilte Zwischenspeicherung und Sperrverwaltung



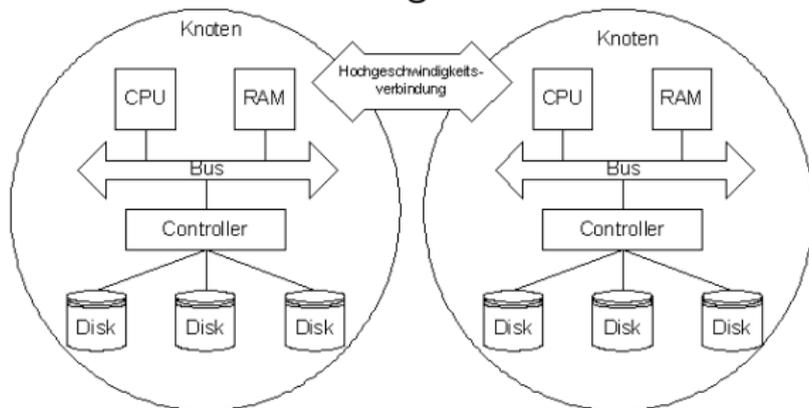
## Architektur: Shared Disk (Cluster aus SMP-Systemen)

- ▶ Kein gemeinsamer Hauptspeicher
- ▶ Gemeinsamer Zugriff auf Sekundärspeicher
- ▶ Skalierbarkeit besser als „shared memory“ aber nicht optimal
- ▶ Komplexer verteilter Cache, komplexes verteiltes Sperrmanagement



## Architektur: Shared Nothing (Massive Parallelverarbeitung, MPP)

- ▶ Getrennte Ressourcen
- ▶ Optimale Skalierbarkeit, gut für Data-Warehouse-Systeme
- ▶ Keine verteilten Caches notwendig, keine verteilte Sperrverwaltung
- ▶ Problem: Partitionierung der Daten



# Speicher

## Festplatten

- ▶ Anbindung  
Lokal, NAS, SAN
- ▶ Technologie  
Raid 1, Raid 5, Raid 10, ...
- ▶ Verteilung der Daten auf verschiedene Festplatten
  - ▶ Eigene Festplatte für Protokolle
  - ▶ Getrennte Festplatten für Daten und Indizes

## Hauptspeicher

- ▶ Datenbankpuffer
- ▶ Sortierspeicher
- ▶ Speicher für Hashtabellen

# Partitionierung

- ▶ Zerlegung einer logischen Einheit in mehrere physische
- ▶ Verwaltung sehr großer Relationen
  - ▶ Effizientes Einfügen/Löschen ganzer Teile einer Datenbasis
- ▶ Effizienzsteigerung bei der Abfragebearbeitung
  - ▶ Überspringen von Partitionen
  - ▶ Parallele Bearbeitung von Partitionen

# Horizontale Partitionierung

- ▶ Formen der Aufteilung
  - ▶ Range-Partitionierung
  - ▶ Hash-Partitionierung

Student*			
<u>MatrNr</u>	Name	Vorname	Studiengang
50101	Clausen	Claus	WI
50102	Svenson	Sven	WI
50103	Jensen	Jens	AI
50104	Jansen	Jan	AI

Student (WI) <sup>†</sup>			
<u>MatrNr</u>	Name	Vorname	Studiengang
50101	Clausen	Claus	WI
50102	Svenson	Sven	WI

Student (AI) <sup>†</sup>			
<u>MatrNr</u>	Name	Vorname	Studiengang
50103	Jensen	Jens	AI
50104	Jansen	Jan	AI

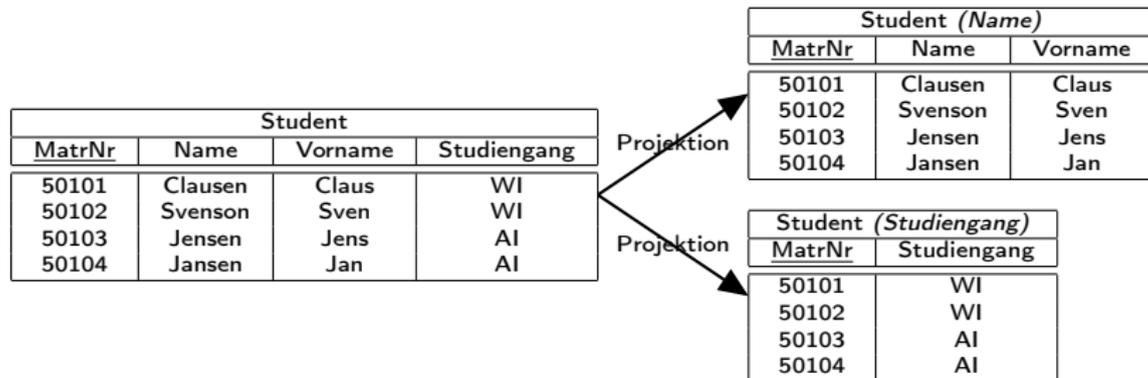
---

\* Masterrelation

<sup>†</sup> Partition

## Vertikale Partitionierung

- ▶ Verteilung der Spalten auf Partitionen
- ▶ Für Rekonstruierbarkeit muss gemeinsames Attribut in jeweils zwei Partitionen existieren



## Beispiele für Range-Partitionierung

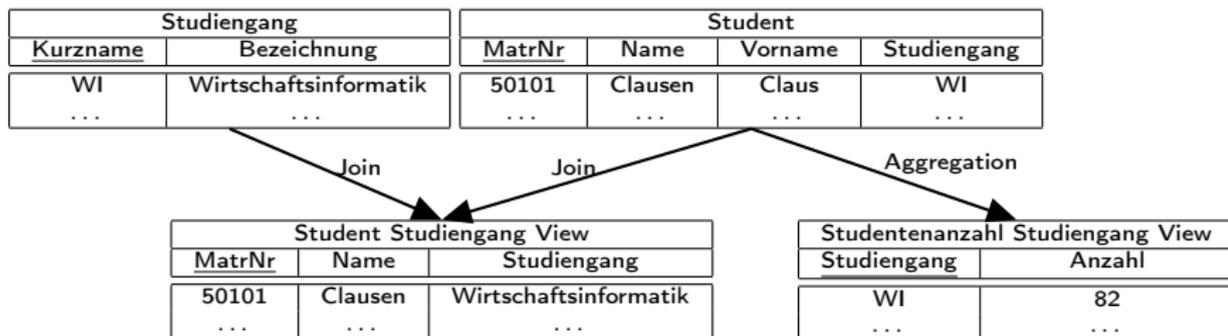
```
create table STUDENT (  
  MATRNR integer not null primary key,  
  NAME varchar(20) not null,  
  VORNAME varchar(20),  
  IMMATRIKULATIONSdatum date,  
  STUDIENGANG char(2),  
  foreign key (STUDIENGANG)  
    references STUDIENGANG(KURZNAME)  
)  
partition by range (MATRNR) (  
  partition STUDENT_10 values less than  
    (51000),  
  partition STUDENT_11 values less than  
    (52000),  
  ...  
)
```

## Beispiel für Hash-Partitionierung

```
create table STUDENT (  
  MATRNR integer not null primary key,  
  NAME varchar(20) not null,  
  VORNAME varchar(20),  
  IMMATRIKULATIONSdatum date,  
  STUDIENGANG char(2),  
  foreign key (STUDIENGANG)  
    references STUDIENGANG(KURZNAME)  
)  
partition by hash (NAME)  
partitions 4  
store in (STUDENT_01, STUDENT_02, STUDENT_03, STUDENT_04)
```

# Materialisierung

- ▶ Speicherung der Ergebnisse von Sichten (Abfragen)
- ▶ Automatisches (transparentes) Umschreiben von Abfragen durch den Abfrageoptimierer, so dass die gespeicherten Ergebnisse benutzt werden



## Beispiel für Materialisierung

```
create materialized view STUDENTENANZAHL_PRO_STUDIENGANG  
tablespace EXAMPLE  
build immediate  
refresh complete  
enable query rewrite as  
select SG.BEZEICHNUNG as STUDIENGANG, count(s.MATRNR) as ANZAHL_STUDENTEN  
from STUDENT S  
inner join STUDIENGANG SG on S.STUDIENGANG = SG.KURZNAME  
group by SG.KURZNAME, SG.BEZEICHNUNG;
```

## Selektivität von Indizes

- ▶ Bezeichnet das Verhältnis der Ergebnismenge zu allen Daten
  - ▶ Beispiel Primärschlüssel:  $1/\text{Anzahl Datensätze}$
  - ▶ Beispiel Geschlecht: Anzahl „M“ bzw. „W“ zu allen Datensätzen =  $1/2$
  - ▶ Je geringer der Wert für die Selektivität, desto besser der Index
  - ▶ Annahme: Gleichverteilung der Werte
  - ▶ Achtung: In der Literatur finden sich unterschiedliche Definitionen
- ▶ Bis zu welchem Wert ist eine Selektivität noch gut
  - ▶ Durch einen Index soll die Anzahl der Zugriffe auf Datenseiten minimiert bzw. sogar elimiert werden
  - ▶ Daumenregel: Selektivität sollte  $\leq 5\%$  sein
  - ▶ Ansonsten ist Tablescan wegen Vorausladen von Datenseiten besser

## Daumenregeln für Indizes

- ▶ Primärschlüssel sollten immer einen Index erhalten
- ▶ Fremdschlüssel sollten meistens einen Index erhalten
- ▶ Spalten, die in einer Where-Klausel auftauchen, sind gute Kandidaten für Indizes
- ▶ Die Selektivität darf aber nicht zu schlecht sein
- ▶ Indizes auf Spalten, die häufig verändert werden, sollten vermieden werden
- ▶ Ein gecusterter Index sollte sorgfältig ausgewählt werden, da nur einer pro Tabelle möglich ist
- ▶ Überdeckende Indizes sollten vorsichtig eingesetzt werden
- ▶ Redundante Indizes sollten vermieden werden
- ▶ Indizes sollten regelmäßig gewartet werden

## Daumenregeln für Indizes bei Spalten in Where-Klauseln

### Sinnvoll bei Spalten . . .

- ▶ die mit AND verknüpft sind (zusammengesetzter Index)
- ▶ die in Verbunden verwendet werden
- ▶ nach denen häufig sortiert wird
- ▶ nach denen häufig gruppiert wird

### Nicht sinnvoll bei . . .

- ▶ Funktionen oder arithmetischen Operationen, die auf Spalten angewendet werden
- ▶ Vergleichsoperator „ist ungleich“
- ▶ LIKE-Vergleich, der mit % beginnt
- ▶ Vergleichen, die auf NULL oder NOT NULL lauten
- ▶ NULL-Werten in Spalten, nach denen sortiert wird

## Abfragebearbeitungsschritte

- ▶ Interndarstellung
  - ▶ Syntaktische Analyse, Kontextanalyse (Syntaxbaum)
- ▶ Zugriffs- und Integritätskontrolle
  - ▶ Einfügen von Prüfoperatoren
- ▶ Abfrageoptimierung / Codeerzeugung
  - ▶ Transformation auf logischer Ebene (Relationenalgebra)
  - ▶ Abbildung auf physische Operatoren (Abfrageplan)
- ▶ Ausführung
  - ▶ Überwachung der Abfragebearbeitung
  - ▶ Pipelining
  - ▶ Ergebnisbereitstellung (Cursor)

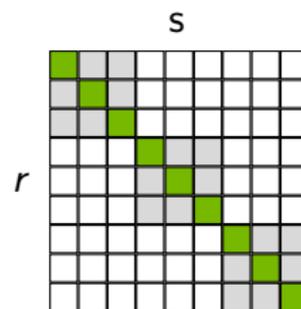
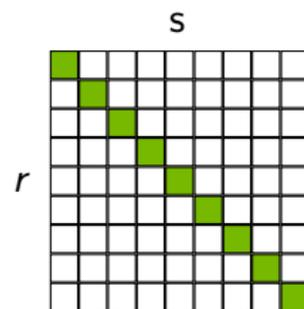
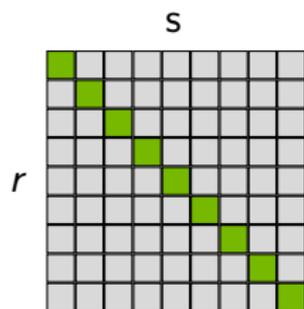
## Statistische Information zur Abfrageoptimierung

- ▶ Anzahl der Seiten pro Tabelle, Anzahl der nichtleeren Seiten
- ▶ Verhältnis von Überlaufzeilen zu Gesamtzahl der Zeilen pro Tabelle
- ▶ Gesamtzahl der Zeilen pro Tabelle
- ▶ Anzahl der verschiedenen Werte in einer Spalte
- ▶ Verteilung von Spaltenwerten
- ▶ Clustereigenschaft eines Indexes, d. h. Ausmaß der Übereinstimmung von Indexsortierung und physischer Zeilenreihenfolge
- ▶ Indexhöhe und Anzahl der Blattseiten

# Operatoren

- ▶ Operatoren zum Durchwandern von Datensätzen (Scan-Operatoren)
  - ▶ Tabellen-Scan
  - ▶ Index-Scan
- ▶ Verbundoperatoren
  - ▶ Geschachtelte Schleifen (nested loop join)
  - ▶ Mischverknüpfung (merge join)
  - ▶ Hash-Verknüpfung (hash join)
- ▶ Sonstige
  - ▶ Extrahieren von Datensätzen über TID/RID
  - ▶ Anwendung von zusätzlichen Prädikaten über die im Tabellen- bzw. Index-Scan hinaus
  - ▶ Sortierung (mit oder ohne Gruppierung)
  - ▶ Gruppierungen
  - ▶ Operatoren auf TID/RID: Und-/Oder-Verknüpfung von Indizes
  - ▶ Erstellung temporärer Tabellen

## Verbundoperatoren\*



### Nested Loop Join

- ▶ vergleicht jedes Tupel von  $r$  mit jedem Tupel von  $s$
- ▶ Prinzip der geschachtelten Schleifen

### Merge Join

- ▶ Mischen vorsortierter Relationen
- ▶ vergleicht nur marginal mehr Tupel als verbunden werden

### Hash Join

- ▶ nutzt Hash-Verfahren
- ▶ bildet Unterbereiche (Buckets)
- ▶ Hashtabelle sollte in Hauptspeicher passen

## Verbundoperation: Nested-Loop-Join

T	
A	...
2	...
3	...
3	...

U	
...	A
...	3
...	2
...	2
...	3
...	1

Algorithmus:

- ▶ Durchlauf (scan) durch die Zeilen  $t$  von  $T$
- ▶ Für jede Zeile  $t$ :
  - ▶ Durchlauf (scan) durch die Zeilen  $u$  von  $U$ 
    - ▶ Übernahme von  $(t, u)$  in das Ergebnis wenn  $t.A = u.A$

## Verbundoperation: Merge-Join

Algorithmus:

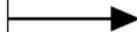
- ▶ Voraussetzung: Sortierung auf Spalte A liegt vor
- ▶ Lies erste Zeile  $t$  von  $T$ 
  - ▶ (\*) Lies Zeilen von  $U$  bis eine Zeile  $u$  mit  $t.A = u.A$  gefunden wird
  - ▶ Übernahme von  $(t, u)$  in das Ergebnis
  - ▶ Lies Zeilen von  $U$  bis eine Zeile  $ux$  mit  $t.A \neq ux.A$  gefunden wird oder keine Zeile in  $U$  mehr vorhanden ist
    - ▶ Übernahme aller bis dahin gelesener  $U$ -Zeilen in das Ergebnis
  - ▶ Gehe auf Zeile  $u$  zurück
- ▶ Solange noch Zeilen in  $T$  vorhanden sind: Lies nächste Zeile von  $T$  und verfare wie bei (\*)

## Verbundoperation: Hash-Join

- ▶ Erzeugung einer Hash-Tabelle für  $U$
- ▶ Durchlauf (scan) durch die Zeilen  $t$  von  $T$
- ▶ Für jede Zeile  $t$ :
  - ▶ Berechnung des Hash-Wertes von  $t.A$
  - ▶ Ermittlung der passenden  $u$  aus  $U$  über die Hash-Tabelle
  - ▶ Übernahme von  $(t, u)$  in das Ergebnis

## Datenmodell für Abfragebeispiele

Student			
<u>MatrNr</u>	Name	Vorname	Studiengang
50101	Clausen	Claus	WI
50102	Svenson	Sven	WI
50103	Jensen	Jens	AI
50104	Jansen	Jan	AI



Studiengang	
<u>Kurzname</u>	Bezeichnung
WI	Wirtschaftsinformatik
AI	Angewandte Informatik
WM	Wirtschaftsmathematik

## Abfrage ohne Index

► Abfrage

```
select *  
from STUDENT  
where STUDIENGANG = 'WI'
```

► Explain

OPERATION	OBJECT_NAME	OPTIONS	COST
  SELECT STATEMENT			3
  TABLE ACCESS	STUDENT	FULL	3
  Filterprädikate			
 STUDIENGANG='WI'			

- **Table-Scan:** Durchlauf durch alle Zeilen der Tabelle

## Abfrage von Zeilen mit Index

► Index

```
create index IX_STUDENT_SG on STUDENT(STUDIENGANG)
```

► Abfrage

```
select *  
from STUDENT  
where STUDIENGANG = 'WI'
```

► Explain

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			2
TABLE ACCESS	STUDENT	BY INDEX ROWID	2
INDEX	IX_STUDENT_SG	RANGE SCAN	1
Zugriffsprädikate			
STUDIENGANG='WI'			

- **Index-Scan:** Durchlauf nur durch Zeilen der Tabelle, die den Wert STUDIENGANG = 'WI' haben

## Abfrage von Zeilen nur mit Index

► Abfrage

```
select MATRNR  
from STUDENT  
where MATRNR > 50000
```

► Explain

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			2
INDEX	SYS_C00122822	FAST FULL SCAN	2
Filterprädikate			
MATRNR>50000			

- Der Zugriff über den Index ist ausreichend, da keine weiteren Werte aus den Zeilen benötigt werden

## Abfrage mit Verbund

### ▶ Abfrage

```
select S.MATRNR, S.NAME, SG.BEZEICHNUNG
from STUDENT S
inner join STUDIENGANG SG on S.STUDIENGANG = SG.KURZNAME
where S.STUDIENGANG = 'WI'
```

### ▶ Explain

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			4
NESTED LOOPS			4
TABLE ACCESS	STUDIENGANG	BY INDEX ROWID	1
INDEX	SYS_C00122819	UNIQUE SCAN	1
Zugriffsprädikate			
SG.KURZNAME='WI'			
TABLE ACCESS	STUDENT	FULL	3
Filterprädikate			
S.STUDIENGANG='WI'			

- ▶ Table-Scan notwendig, da kein Index auf dem Fremdschlüssel STUDIENGANG in der Tabelle STUDENT

## Abfrage mit Verbund und Index

► Index

```
create index IX_STUDENT_SG on STUDENT(STUDIENGANG)
```

► Abfrage

```
select S.MATRNR, S.NAME, SG.BEZEICHNUNG  

from STUDENT S
```

```
inner join STUDIENGANG SG on S.STUDIENGANG = SG.KURZNAME
```

```
where S.STUDIENGANG = 'WI'
```

► Explain

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			2
NESTED LOOPS			2
TABLE ACCESS	STUDIENGANG	BY INDEX ROWID	1
INDEX	SYS_C00122819	UNIQUE SCAN	1
Zugriffsprädikate SG.KURZNAME='WI'			
TABLE ACCESS	STUDENT	BY INDEX ROWID	1
INDEX	IX_STUDENT_SG	RANGE SCAN	0
Zugriffsprädikate S.STUDIENGANG='WI'			

► Index-Scan mit Index auf Fremdschlüssel

## Abfrage mit Verbund

► Abfrage

```
select S.MATRNR, S.NAME, SG.BEZEICHNUNG
from STUDENT S
inner join STUDIENGANG SG on S.STUDIENGANG = SG.KURZNAME
```

► Explain

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			7
HASH JOIN			7
Zugriffsprädikate S.STUDIENGANG=SG.KURZNAME			
TABLE ACCESS	STUDIENGANG	FULL	3
TABLE ACCESS	STUDENT	FULL	3

- Da die Tabelle STUDIENGANG nur wenige Zeilen enthält, wird der Hash-Join verwendet, der eine Hashtabelle im Speicher aufbaut.
- Das ist effizienter als ein Nested-Loop-Join und ein mehrmaliger Index-Scan auf Tabelle STUDIENGANG.