

# 10. Satzorientierte DB-Schnittstelle

- **Ziele**
  - Entwurfsprinzipien für die satzorientierte Verarbeitung und die Navigation auf logischen Zugriffspfaden
  - Entwicklung einer Scan-Technik und eines Sortieroperators
- **Scheduling und Lastkontrolle von Transaktionen**
- **Bereitstellung der Objekte und Operationen**  
durch eine satzorientierten DB-Schnittstelle  
(z. B. entsprechend dem Netzwerk-Datenmodell oder (vielen) objektorientierten Datenmodellen)
- **Aufgaben und Funktionen der Metadatenverwaltung**  
(Data Dictionary / Directory)
- **Abbildung von externen Sätzen**
  - Optionen für die Satzspeicherung
  - Partitionierungsverfahren
- **Satzorientierte Verarbeitung**
  - Verarbeitungsprimitive
  - Cursor-Konzept zur satzweisen Navigation
- **Scan-Technik**
  - Implementierung durch Scan-Operatoren
  - verschiedene Scan-Typen
- **Sortierkomponente**
  - zur Unterstützung relationaler Operationen
  - Sort-Merge-Verfahren
  - Sortieren variabel langer Sätze

# Logische Zugriffspfade

- **Charakterisierung der Abbildung**

STORE <record>

FETCH <record> USING <attr1> = 400 AND <attr2> >=7

CONNECT <record> TO <set>

## Abbildungsfunktionen

- **Physischer Satz <-> Externer Satz**
- **Externer Satz <-> zugehörige Zugriffspfade**
- **Suchausdruck -> unterstützende Zugriffspfade**

insert <record> at ...

add <entry> to...

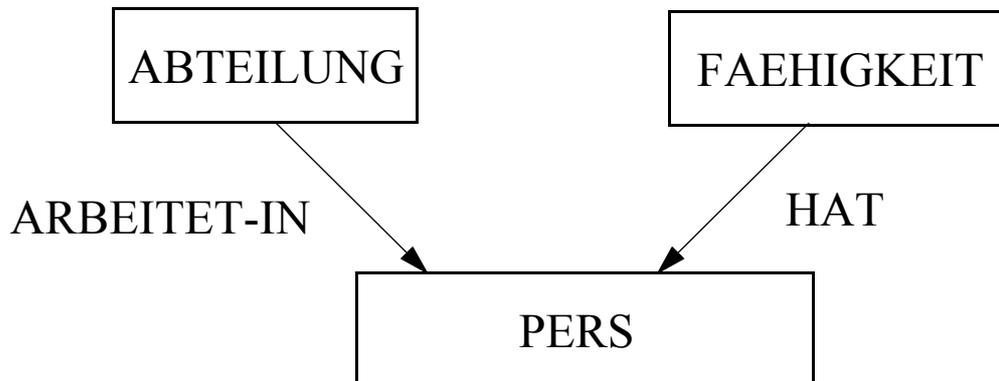
retrieve <address-list> from...

retrieve <record> with

- **Eigenschaften der oberen Schnittstelle**

- Logische Sätze (dynamische Format-Umsetzung)
- Logische Zugriffspfade (inhaltsadressierbarer Speicher, hierarchische Beziehungen zwischen Satztypen)
- Zugriff in Einheiten von einem Satz pro Aufruf
- Anwendungsprogrammierschnittstelle (API = application programming interface) mit navigierendem Zugriff (z. B. entsprechend dem Netzwerk-Datenmodell oder objektorientierten Datenmodellen)

## Das Netzwerk-Datenmodell - Ein Beispiel



### 1. Einfache Inhaltsadressierung

FETCH PERS USING PNR=12345

### 2. ... kombiniert mit hierarchischem Zugriff

FETCH ABTEILUNG USING ABT-NAME = 'VERTRIEB'

FETCH NEXT PERS WITHIN ARBEITET-IN

### 3. ... mit zwei hierarchischen Beziehungen

FETCH FAEHIGKEIT USING BERUF = 'PROGRAMMIERER'

FETCH NEXT PERS WITHIN HAT

FETCH OWNER WITHIN ARBEITET-IN

# Scheduling und Lastkontrolle

- **Transaktionsbezug von Objekten und Operationen**

geht in tieferen Schichten verloren

- Übernahme von transaktionsbezogenen Aktivierungs- und Überwachungsaufgaben in dieser Schicht
- Jedoch ist zur optimalen Steuerung ein schichtenübergreifender Informationsfluß zwingend geboten

↳ Aus Performance-Gründen wird hier die schichtenweise Kapselung unseres Architekturmodells aufgegeben

- **Kontroll- und Zuteilungsaufgaben**

- Festlegung und Erzeugung von Ablaufeinheiten für Transaktionen (multi-programming level)
- Scheduling zur Auswahl und Aktivierung der Transaktionslast
- Speicherverwaltung zur Zuteilung von Arbeitsspeicher oder DB-Pufferrahmen
- Lastkontrolle und -balancierung zur Behebung von Engpaßsituationen und zur Erreichung vorgegebener Leistungsziele

- **Leistungsziele können sehr unterschiedlich sein:**

- Durchsatzmaximierung für alle Transaktionen
- Gewährleistung von Antwortzeiten für verschiedene Klassen von Transaktionen
- Erledigung von Aufträgen gemäß vorgegebener Prioritäten

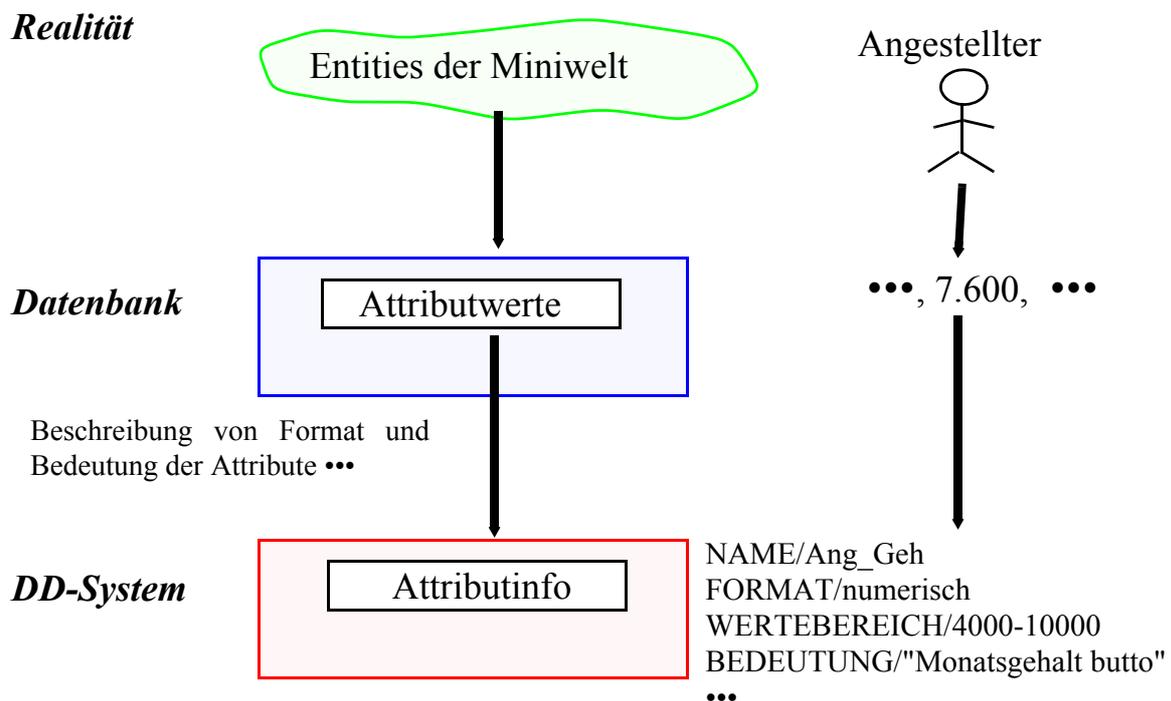
↳ automatische Lastanpassung bei einem vorgegebenen Leistungsziel ist sehr schwierig!

# Integriertes Data Dictionary-/Directory-System

- **Aufgabe:**

Ein D/D-System dient zur Verwaltung von Metadaten, die Informationen über Inhalt, Nutzung, Integritätsbedingungen usw. eines Datenverwaltungs- bzw. Datenbanksystems geben.

- **Charakterisierung der erforderlichen Abbildungen**



- **Datenbankkataloge**

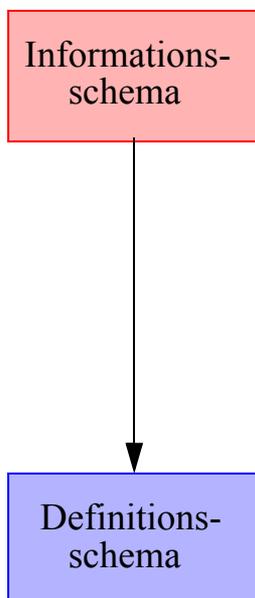
- normkonforme Implementierungen bieten einheitliche Sichten auf Metadaten
- Sichten werden durch sog. **Informationsschema** definiert
- Informationsschema ist für den Benutzer definierte Schnittstelle zum DB-Katalog

# Informations- und Definitionsschema

- **Ziel der SQL-Normierung**

- möglichst große Unabhängigkeit der DB-Anwendungen von speziellen DBS
- einheitliche Sprachschnittstelle genügt **nicht!**
- **Beschreibung der gespeicherten Daten** und ihrer Eigenschaften nach einheitlichen und verbindlichen Richtlinien ist genauso wichtig

- **Zweischichtiges Definitionsmodell für die Beschreibung der Daten**



- bietet **einheitliche Sichten** in normkonformen Implementierungen
- ist **für den Benutzer zugänglich** und somit die definierte Schnittstelle zum Katalog

- beschreibt **hypothetische** Katalogstrukturen
- erlaubt „**Altsysteme**“ mit abweichenden Implementierungen normkonform zu werden

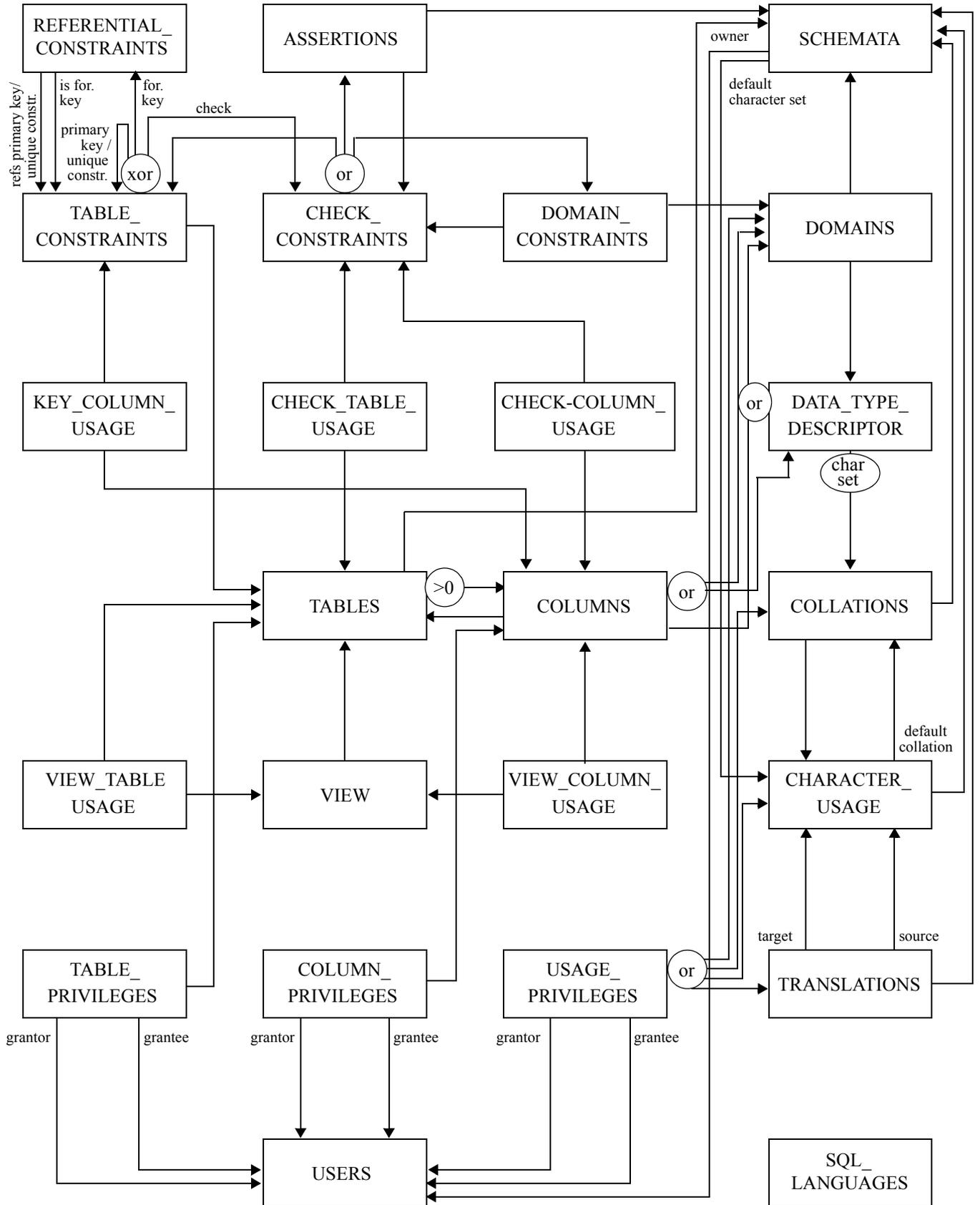
- **Komplexe Strukturen<sup>1</sup>**

- DEFINITION\_SCHEMA umfasst 24 Basistabellen und 3 Zusicherungen
- In den Tabellendefinitionen werden ausschließlich 3 Domänen verwendet: SQL\_IDENTIFIER, CHARACTER\_DATA und CARDINAL\_NUMBER

---

1. Das nicht normkonforme Schema SYSCAT von DB2 enthält 37 Tabellen

# Definitionsschema



# **Aufgaben und Funktionen des Data Dictionary/Directory**

- **Speicherung der Beschreibungs- oder Metainformation**
  - Angaben über Definition, Struktur und Benutzungsvorschriften
  - Angaben über Speicherung, Codierung und Zugriffspfade
  - direkte Zugriffsmöglichkeit auf Schema-Information durch Anwendungsprogramme (AP) vorteilhaft
  
- **Definition und Kontrolle durch DBA**
  
- **Ableitung von Subschemas durch DDL-Übersetzer**
  
- **Erweiterung des D/D**
  - Info über Herkunft, aktuelle Benutzung und Änderung der Daten
  - Namen und Charakteristika von AP (Datenverwendung)
  
- **Spezielle Funktionen zur Auswertung**
  - Dokumentation der Daten
  - Analyse der vorhandenen Datenobjekte
  - Unterstützung des Entwurfs von Anwendungen
  - Information über Gültigkeit und Verfügbarkeit der Daten
  - Analyse der Auswirkungen von Änderungen der Datendefinition
  - Optimierung der Speicherungsstrukturen aufgrund von Benutzungshäufigkeiten und Werteverteilungen

# Satzorientierte DB-Schnittstelle

- **Bereitstellung der Objekte und Operationen**
  - als externe Schnittstelle bei satzorientierten DBS
  - als interne Schnittstelle bei mengenorientierten DBS
- **Typische Objekte**
  - Segment (Area)
  - Satztyp (Tabelle) und Satz (Tupel)
  - Index (Search Key) und Set (Link)
- **Operatoren auf Segmenten**
  - Öffnen und Schließen von Segmenten (OPEN / CLOSE)
  - Erwerb und Freigabe von Segmenten (ACQUIRE / RELEASE)
  - Sichern und Zurücksetzen von Segmenten (SAVE / RESTORE)
- **Operatoren auf Sätzen und Zugriffspfaden**
  - Direktes Auffinden von Sätzen über Attributwerte (FIND RECORD USING ...)
  - Navigierendes Auffinden von Sätzen über einen Zugriffspfad (FIND NEXT RECORD WITHIN ...)
  - Hinzufügen eines Satzes (INSERT)
  - Löschen eines Satzes (DELETE)
  - Aktualisieren von Attributwerten eines Satzes (UPDATE)
- **Manipulation von benutzerkontrollierten Zugriffspfaden**
  - Einbringen eines Satzes (CONNECT)
  - Aufheben dieser Verknüpfung (DISCONNECT)
- **Weitere typische Operationen**
  - BEGIN / COMMIT / ABORT\_TRANSACTION
  - LOCK, UNLOCK

## Abbildung von externen Sätzen

- Beschreibung der externen Sätze durch Subschema-Konzept
- **Aufgaben des Subschema-Konzeptes**
  - Anpassung der Datentypen
  - Selektive Auswahl von Attributen
  - Abbildung eines externen Satzes auf interne Sätze eines oder mehrerer Satztypen
- **Partitionierte Speicherung großer Satzmengen:**

Zuordnung disjunkter Satzmengen zu separaten Speichereinheiten

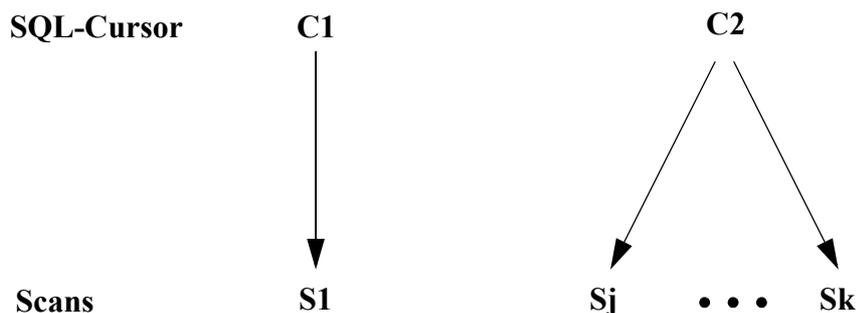
  - Leistungsgründe: E/A-Parallelität
  - Verfügbarkeit: Erstellung von Kopien, Migration, . . .
  - Partition ist Einheit der Reorganisation, des Backup, der Archivierung, des Ladens, von Zugriffsverfahren etc. in DB2
  - Spezifikation der Partitionierung über Werte (Schlüsselbereiche, Hashing) oder über Prozeduren (user exit)
- **Optionen der Speicherung bei internen Sätzen**
  - Aufteilung und Zuordnung der Felder nach Zugriffshäufigkeiten
  - redundante Speicherung
  - Verdichtung von Feldern und Sätzen
- ➔ **Möglichkeiten der Abstimmung/Verbesserung des Leistungsverhaltens (Tuning)**

# Satzorientierte Verarbeitung

- Wie wird die **satzorientierte Verarbeitung** durch die Schicht der Speicherungsstrukturen unterstützt?
  
- **Verarbeitungskonzepte**
  - Kontextfreie Operationen
  - Satzweise Navigation über vorhandene Zugriffspfade
  - Umordnung einer Satzmenge, falls passende Reihenfolge nicht vorhanden
  
- **Verarbeitungsprimitive**
  1. Direkter Zugriff  
(Hashing, Bäume, ...)
  2. Navigierender Zugriff über Scan  
(auf welchen Objekten?)
  3. Sortierung einer Satzmenge und Scan darauf  
(Welche Satzmenge?)
  4. Satzweise Aktualisierung  
(Insert, Delete, Update)
  
- **Einführung eines Navigationskonzeptes**
  - Bereitstellen und Warten von transaktionsbezogenen  
**Verarbeitungspositionen zur Navigation**
  - Verschiedene Cursor-Konzepte

## Satzorientierte Verarbeitung (2)

- **Implizites Cursor-Konzept** (Currency-Konzept bei CODASYL)
  - Änderung mehrerer Cursor durch Ausführung einer DB-Anweisung
  - Benutzerkontrolle durch RETAINING-Klausel
  - Hochgradige Komplexität, Fehleranfälligkeit
- **Explizites Cursor-Konzept**
  - Cursor werden durch AP definiert, verhalten sich wie normale Programmvariablen und werden unter expliziter Kontrolle des AP verändert
  - Definition von n Cursor auf einer Tabelle
  - Eindeutige Änderungssemantik
- **Was ist ein Scan?**
  - Ein Scan erlaubt das satzweise Durchmustern und Verarbeiten einer physischen Satzmenge
  - Er kann an einen Zugriffspfad (Index, Link, ...) gebunden werden
  - Er definiert eine Verarbeitungsposition, die als Kontextinformation für die Navigation dient
- Wie unterstützt das Scan-Konzept die **mengenorientierte Verarbeitung (SQL)**?



# Spektrum von Scan-Typen

- **Scan-Typen**

- Satztyp-Scan (Tabellen-Scan) zum Aufsuchen aller Sätze eines Satztyps (einer Tabelle)
- Index-Scan zum Aufsuchen von Sätzen in einer wertabhängigen Reihenfolge
- Link-Scan zum Aufsuchen von Sätzen in benutzerkontrollierter Einfügereihenfolge
- k-d-Scan zum Aufsuchen von Sätzen über einen k-dimensionalen Index.<sup>2</sup>

- **Implementierung von Scans**

- explizite Definition/Freigabe: OPEN/CLOSE SCAN
- Navigation: NEXT TUPLE
- Scans werden auf Zugriffspfaden definiert
- Optionen:  
 Start-, Stopp- und Suchbedingung (Simple Search Argument)  
 Suchrichtung: NEXT/PRIOR, FIRST/LAST, n-th
- Scan-Kontrollblock (SKB):  
 Angaben über Typ, Status, Position etc.

	Typ	Objekt	Start	Stopp	Status
SKB:					
	SSA		Richtung	TA	...

---

2. Es ist wünschenswert, für alle mehrdimensionalen Zugriffspfade ein einheitliches Auswertungsmodell anbieten zu können. Dadurch würde sich eine DBS-Erweiterung um einen neuen mehrdimensionalen Zugriffspfadtyp lokal begrenzen lassen. Jedoch dürfte die direkte Abbildung des Auswertungsmodells bei manchen Strukturen sehr komplex und gar unmöglich sein. Als Ausweg bietet sich hier an, das Anfrageergebnis mit Hilfe der verfügbaren Operationen des physischen Zugriffspfads abzuleiten, ggf. zu sortieren und in einer temporären Speicherstruktur zu materialisieren. Auf dieser Speicherstruktur könnte dann der k-d-Scan nachgebildet werden, um das abstrakte Auswertungsmodell für die satzweise Verarbeitung zu realisieren. Falls ein k-d-Scan nur ungeordnete Treffermengen abzuliefern braucht, sind sicherlich einfachere Auswertungsmodelle, die sich stärker an den Eigenschaften der darunterliegenden physischen Strukturen orientieren, denkbar.

# Tabellen-Scan

- **Anfragebeispiel:**

```
SELECT *
FROM PERS
WHERE ANR BETWEEN K28 AND K67
AND BERUF = 'Programmierer'
```

- **Scan-Optionen**

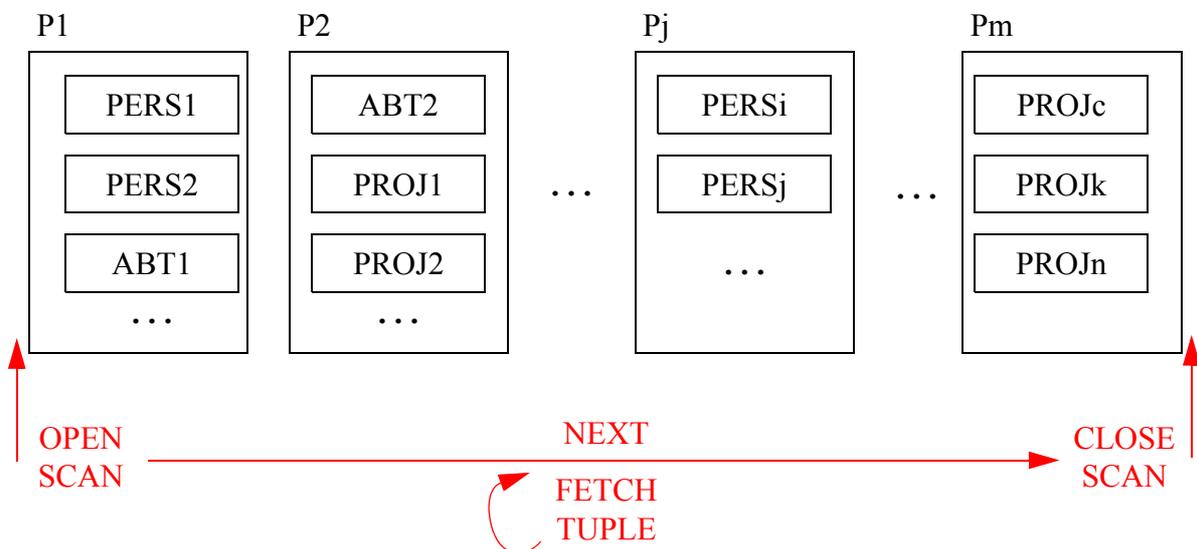
- Startbedingung (SB): BOS (Beginn von S1)
- Stoppbedingung (STB): EOS (Ende von S1)
- Suchrichtung: NEXT
- Suchbedingung (SSA):  $ANR \geq 'K28'$  AND  $ANR \leq 'K67'$   
AND BERUF = 'Programmierer'

- **Tabellen-Scan**

```
OPEN SCAN (PERS, BOS, EOS) /* SCB1 */
WHILE (NOT FINISHED)
DO
    FETCH TUPLE (SCB1, NEXT,
                ANR  $\geq$  'K28' AND ANR  $\leq$  'K67' AND BERUF = 'Programmierer')
    ...
END
CLOSE SCAN (SCB1)
```

- **Ablauf beim Tabellen-Scan**

Segment S1:



# Index-Scan

- Anfragebeispiel:**

```

SELECT *
FROM PERS
WHERE ANR BETWEEN K28 AND K67
      AND BERUF = 'Programmierer'
    
```

- Scan-Optionen**

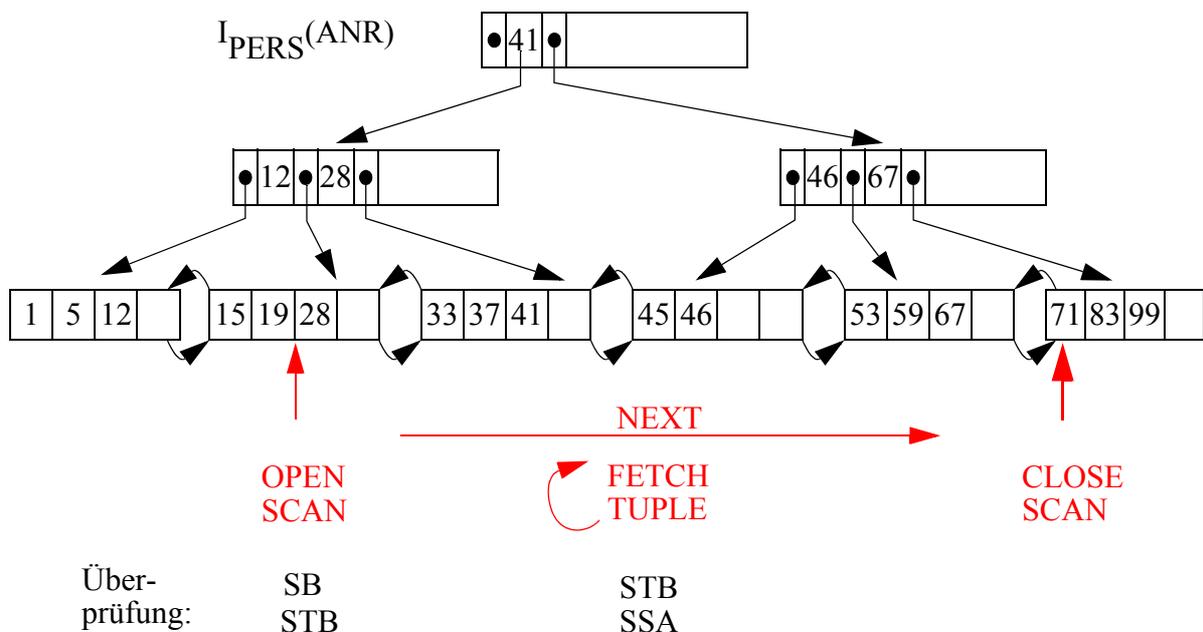
- Startbedingung:             $ANR \geq 'K28'$
- Stoppbedingung:            $ANR > 'K67'$
- Suchrichtung:             NEXT
- Suchbedingung:             $BERUF = 'Programmierer'$

- Index-Scan**

```

OPEN SCAN (IPERS(ANR), ANR ≥ 'K28', ANR > 'K67')                   /* SCB1 */
WHILE (NOT FINISHED)
DO
  FETCH TUPLE (SCB1, NEXT, BERUF = 'Programmierer')
  ...
END
CLOSE SCAN (SCB1)
    
```

- Ablauf beim Index-Scan**



- Die Verweise (TIDs) auf die PERS-Tupel sind in  $I_{PERS}(ANR)$  weggelassen.  
Die Suchbedingung (SSA = simple search argument) darf nur Wertvergleiche „Attribut  $\Theta$  Wert“ (mit  $\Theta \in \{<, =, >, \leq, \neq, \geq\}$ ) enthalten und wird auf jedem Tupel überprüft, das über den Index-Scan erreicht wird. Der Operator FETCH TUPLE liefert also nur Tupel zurück, die die WHERE-Bedingung erfüllen.

# Link-Scan

- Anfragebeispiel:**

```

SELECT *
FROM PERS
WHERE ANR BETWEEN K28 AND K67
      AND BERUF = 'Programmierer'
    
```

- Auffinden des Vaters**

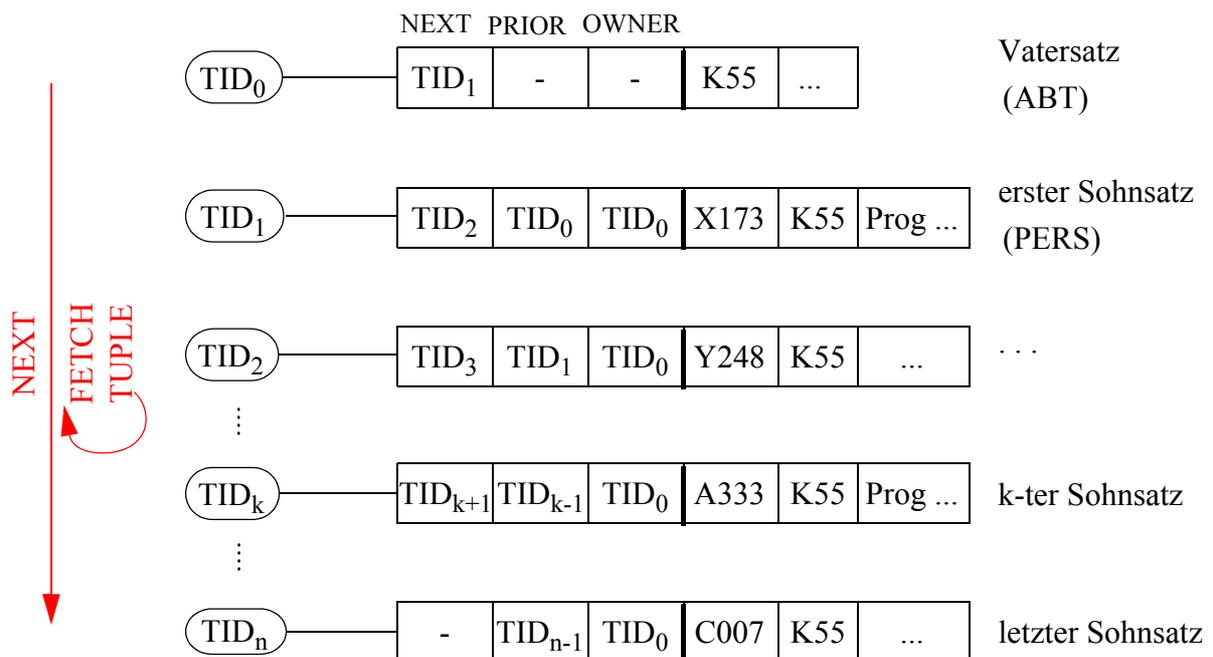
- Startposition bereits gefunden (ANR gegeben)
- Fortschalten in ABT erforderlich (ANR BETWEEN K28 AND K67)

- Einzelner Link-Scan**

```

OPEN SCAN (LABT-PERS(ANR), NONE, EOL)                                /* SCB1 */
WHILE (NOT FINISHED)
DO
    FETCH TUPLE (SCB1, NEXT, BERUF = 'Programmierer')
    ...
END
CLOSE SCAN (SCB1)
    
```

- Ablauf beim Link-Scan**



## Link-Scan (2)

- **Auffinden des Vaters**

- Nutzung einer Indexstruktur
- Schachtelung von Index-Scan (ANR BETWEEN K28 AND K67) und Link-Scan

- **Scan-Optionen**

	<b>Index-Scan</b>	<b>Link-Scan</b>
- Startbedingung:	ANR ≥ 'K28'	-
- Stoppbedingung:	ANR > 'K67'	EOL
- Suchrichtung:	NEXT	NEXT
- Suchbedingung:	-	BERUF = 'Programmierer'

- **Index- und Link-Scan**

```
OPEN SCAN (IABT(ANR), ANR ≥ 'K28', ANR > 'K67')           /* SCB1 */
...
WHILE (NOT FINISHED)
DO
    FETCH TUPLE (SCB1, NEXT, NONE)
    ...
    OPEN SCAN (LABT-PERS(ANR), NONE, EOL)                 /* SCB2 */
    ...
    WHILE (NOT FINISHED)
    DO
        FETCH TUPLE (SCB2, NEXT, BERUF = 'Programmierer')
        ...
    END
    CLOSE SCAN (SCB2)
END
CLOSE SCAN (SCB1)
```

## k-d-Scan

- **Anfragebeispiel:**

```
SELECT *
FROM PERS
WHERE ANR BETWEEN 'K28' AND 'K67'
      AND ALTER BETWEEN 20 AND 30
      AND BERUF = 'Programmierer'
```

- **Scan-Optionen**

	<b>Dimension 1</b>	<b>Dimension 2</b>
– Startbedingung:	ANR ≥ 'K28'	ALTER ≥ 20
– Stoppbedingung:	ANR > 'K67'	ALTER > 30
– Suchrichtung:	NEXT	NEXT
– Suchbedingung:	BERUF = 'Programmierer' (wird auf den PERS-Sätzen ausgewertet)	

- **2-d-Scan**

```
OPEN SCAN (IPERS(ANR, ALTER), ANR ≥ 'K28' AND ALTER ≥ 20,
           ANR > 'K67' AND ALTER > 30)                               /* SCB1 */
...
WHILE (NOT (ALTER > 30))
DO
  /* Zwischenspeichern der SCB1-Position in SCANPOS                    */
  WHILE (NOT (ANR > 'K67'))
  DO
    FETCH TUPLE (SCB1, NEXT(ANR), BERUF = 'Programmierer')
    ...
  END
  /* Zurücksetzen der SCB1-Position auf SCANPOS                        */
  ...
  MOVE SCB1 TO NEXT(ALTER)
END
CLOSE SCAN (SCB1)
```

# Anwendung von Scans

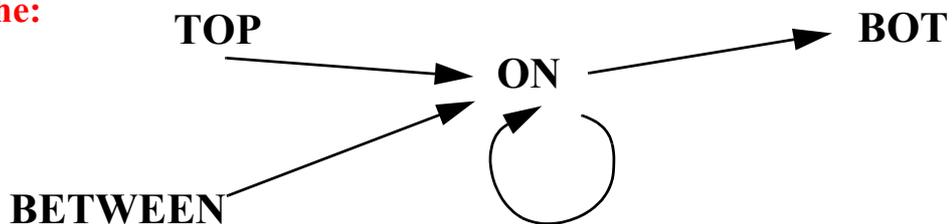
- **Scan-Zustände**

in Abhängigkeit von der Position in der Satzmenge

- vor dem ersten Satz (TOP)
- auf einem Satz (ON)
- in einer Lücke zwischen zwei Sätzen (BETWEEN)
- hinter dem letzten Satz (BOT)
- in einer leeren Menge (NULL)

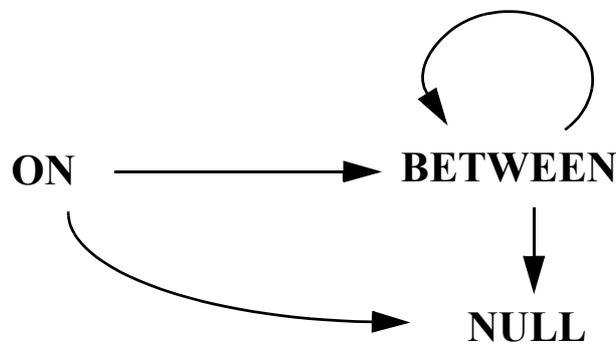
- **Zustandsübergänge beim Scan**

**Suche:**

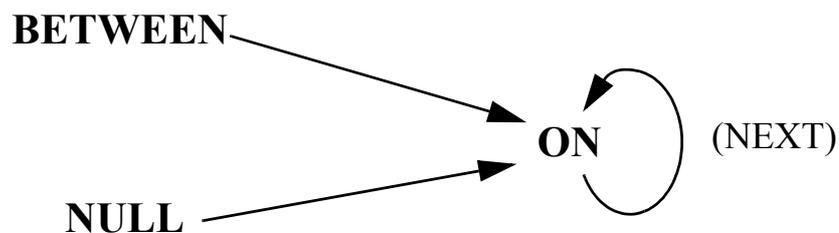


**Aktualisierung:**

**Delete:**



**Insert:**



➔ **Scan-Semantik festlegen!**

## Anwendung von Scans (2)

- **Problem**

**Mengenorientierte Spezifikation <---> satzorientierte Auswertung:**

Die navigierende Auswertung einer deklarativen SQL-Anweisung über einen Scan kann Verarbeitungsprobleme verursachen, insbesondere wenn das zu aktualisierende Objekt (Tabelle, Zugriffspfad) vom Scan benutzt wird (Verbot?)

- **Beispiel**

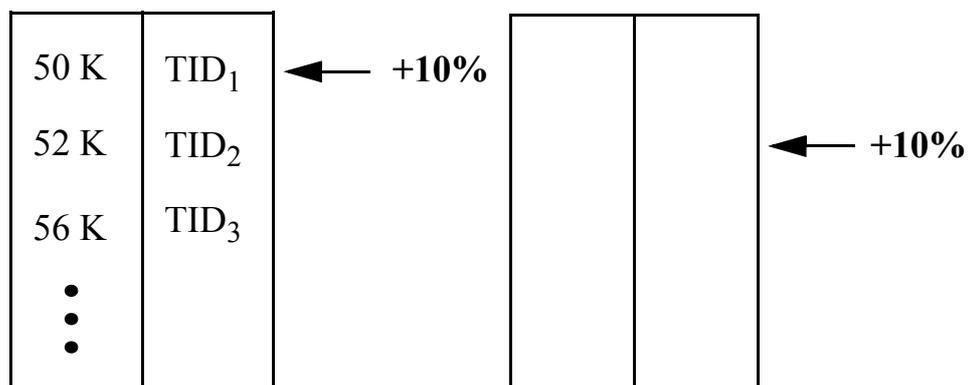
Anweisung: **UPDATE PERS**

**SET GEHALT = 1.1 \* GEHALT**

Ausführung:

Gehaltsverbesserung um 10% werde mit Hilfe eines Scans über Index GEHALT durchgeführt

**I<sub>PERS</sub>(GEHALT)**



→ **Halloween-Problem**

## Einsatz eines Sortieroperators

- Explizite Umordnung der Sätze gemäß vorgegebenem Suchschlüssel (ORDER-Klausel)

- Umordnung mit Durchführung einer Restriktion

```
SELECT * FROM PERS
WHERE ANR > 'K50'
ORDER BY GEHALT DESC
```

- Partitionierung von Satzmengen

```
SELECT ANR, AVG (GEHALT)
FROM PERS
GROUP BY ANR
```

- Duplikateliminierung in einer Satzmenge

```
SELECT DISTINCT BERUF
FROM PERS
WHERE ANR > 'K50' AND ANR < 'K56'
```

- Unterstützung von Mengen- und Verbundoperationen

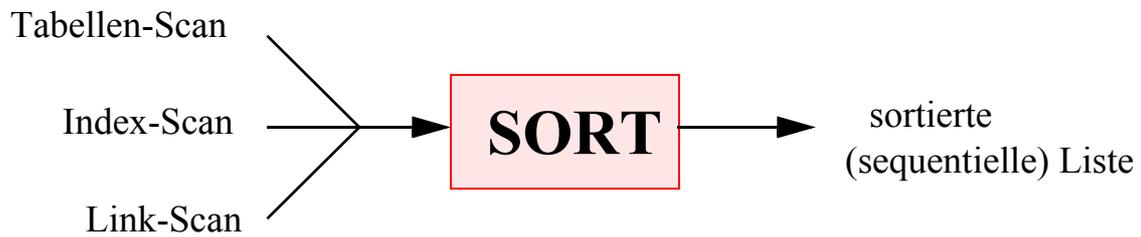
- Umordnen von Zeigern zur Optimierung der Auswertung oder Zugriffsreihenfolge

- Dynamische Erzeugung von Indexstrukturen ("bottom-up"-Aufbau von B\*-Bäumen)

- Erzeugen einer Clusterbildung beim Laden und während der Reorganisation

➔ **Reduktion der Komplexität** von  $O(N^2)$  nach  $O(N \log N)$  bei Mengen- und Verbundoperationen

## **SORT-Operator - Optionen und Anwendung**



- **Scans können mit Suchbedingungen eingeschränkt sein**  
(SSA = Simple Search Arguments)
- **SORT-Optionen zur Duplikateliminierung:**
  - N = keine Eliminierung
  - K = Duplikateliminierung bezüglich Sortierkriterium
  - S = STOPP, sobald Duplikat entdeckt wird
- **SORT dient als Basisoperator für Operationen auf höherer Ebene**

Beispiel.: Einsatz von Scan- und Sortier-Operator

```
OPEN SCAN (R1, SB1, STB1)                               /* SCB1 */
SORT  R1 INTO S1 USING SCAN (SCB1)
CLOSE SCAN (SCB1)
OPEN SCAN (R2, SB2, STB2)                               /* SCB2 */
SORT  R2 INTO S2 USING SCAN (SCB2)
CLOSE SCAN (SCB2)
OPEN SCAN (S1, BOS, EOS)                                /* SCB3 */
OPEN SCAN (S2, BOS, EOS)                                /* SCB4 */
WHILE (NOT FINSHED)
DO
  FETCH TUPLE (SCB3, NEXT, NONE)
  FETCH TUPLE (SCB4, NEXT, NONE)
  ...
END
```

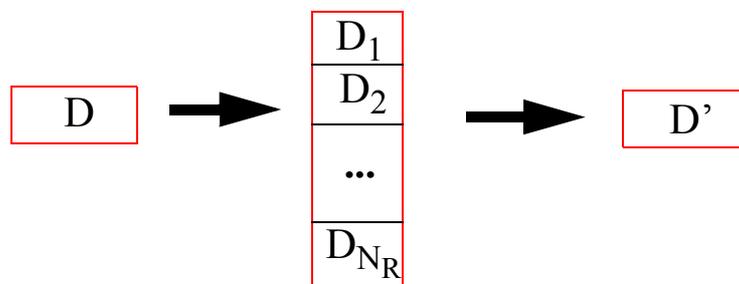
# Externes Sortieren

- **Wie wird sortiert?**

- Zu sortierende Datenmenge (n Sätze) i. allg. viel größer als der zum Sortieren verfügbare HSP (q Sätze)  
(DB-Pufferbereich oder spezieller Arbeitsspeicher)
- Erzeugung sog. Runs ( $N_R \geq 1$ ) mit einem internen Sortierverfahren
- Welche Sortierverfahren sind geeignet?

- **Mehrmalige Durchführung der Sortierung**

- Zerlegung der Eingabe in mehrere Läufe (runs)
- Sortieren und Zwischenspeichern der sortierten Läufe
- Lesen der Eingabedaten aus Datei D und Schreiben der Runs nach  $D_i$

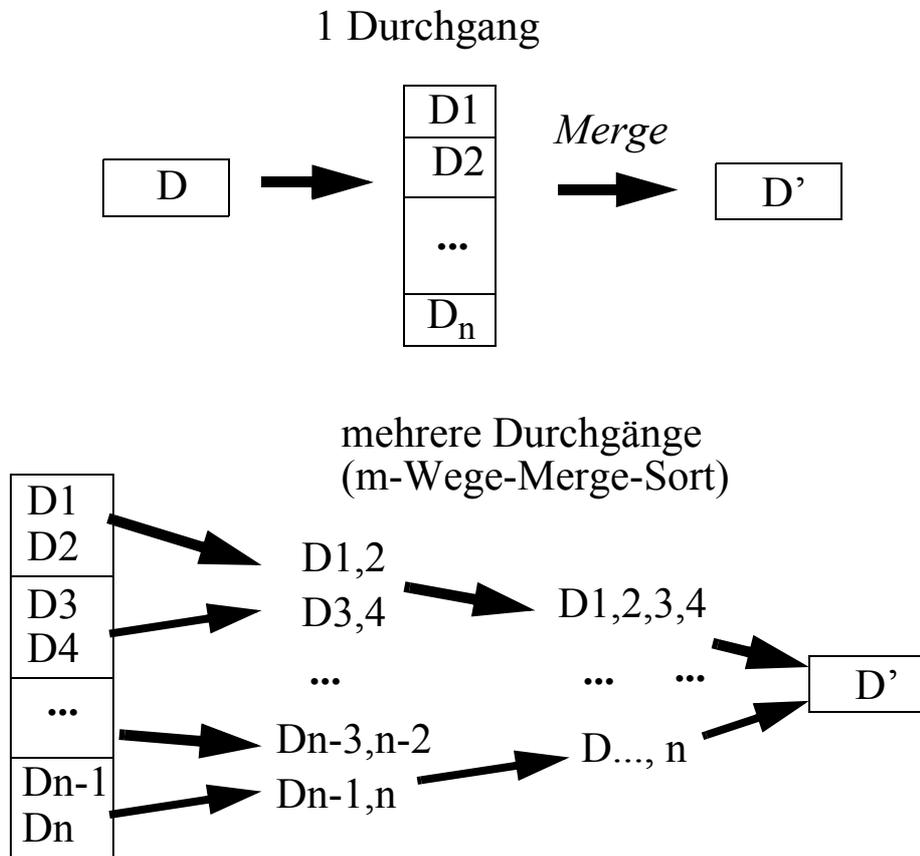


Mischen aller  $N_R$  anfänglichen Runs erforderlich

## Externes Sortieren (2)

- **DBS: Externes Sortieren**

- Sukzessives Mischen bis ein sortierter Lauf entsteht



- **m-Wege-Mischen bei Magnetplatten**

- Es stehen  $m_{\max} + 1$  Plätze (Seiten) im HSP zur Verfügung.  
 $m_{\max}$  bestimmt die maximale Mischordnung.
- $N_R$  anfängliche Runs sind i. allg. über mehrere Magnetplatten verteilt
- Jeder Run wird nach Möglichkeit sequentiell geschrieben.  
Er kann jederzeit wahlfrei gelesen werden.
- Typischerweise sind die anfänglichen Runs gleich lang.

## Externes Sortieren (2)

- **Optimale Mischbäume**

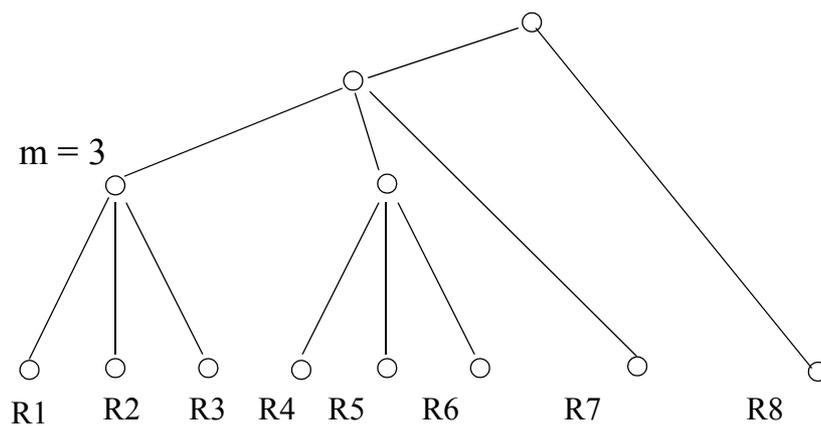
- Optimierung der Zugriffsbewegungen auf den Magnetplatten ist sehr schwierig: wird gewöhnlich nicht berücksichtigt
- Ziel: Minimierung der E/A und der Vergleiche
- einfach bei idealen Anzahlen  $N_R$

$$N_R = m_{\max}^p$$

Es ergibt sich ein vollständiger Mischbaum der Höhe  $p$  vom Grad  $m_{\max}$

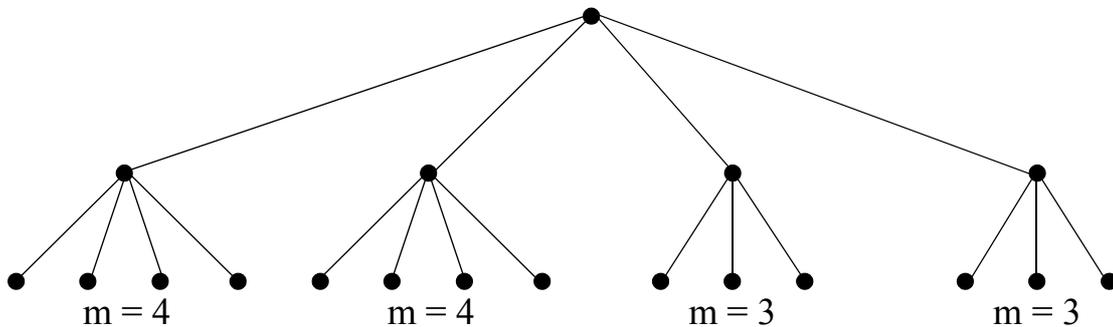
- Wann sind **unvollständige Mischbäume** optimal?

z. B.



## Externes Sortieren (3)

- Allgemein gilt:  $N_R \leq (m_{\max})^{p_{\min}}$
- Annahmen:  
nur 2 Dateien für Ein-/Ausgabe, ungewichtete Runs
- Heuristik 1: Harmonisiere Mischbaum
  - (1) Bestimme  $p_{\min}$
  - (2) Finde kleinstes  $m$ , so daß gilt:  
$$N_R \leq m^{p_{\min}}$$
  - (3) Wende nur Mischordnungen von  $m$  und  $m-1$  an.
- Beispiel:  $N_R = 14$ ,  $m_{\max} = 8$



## Externes Sortieren (4)

- **Annahmen:** keine Beschränkung für Ein-/Ausgabe, gewichtete Runs

- **Heuristik 2:**

**Minimiere Anzahl der Ein-/Ausgaben und Vergleiche**

(1)  $N_R \leq (m_{\max} - 1) * p + 1$ ; bestimme minimales  $p$

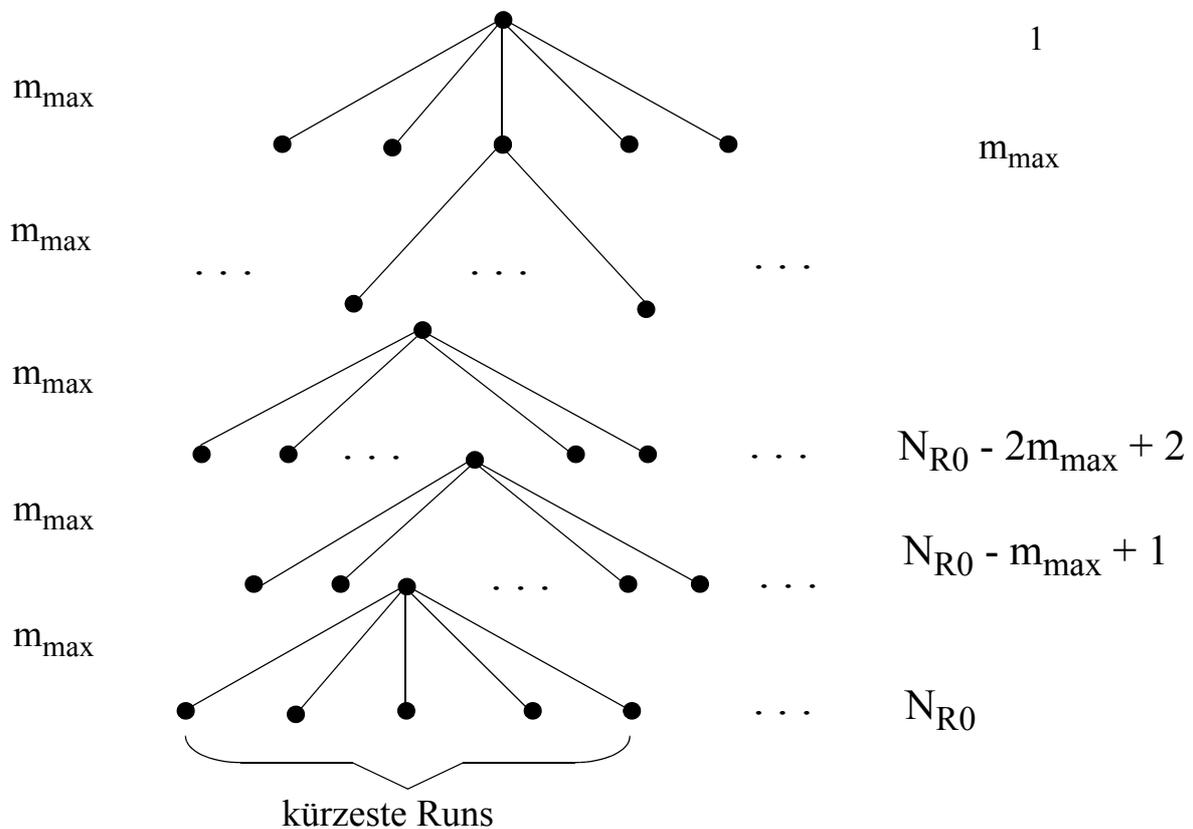
(2) Erzeuge zusätzliche leere Runs, so daß gilt:

$$N_{R0} = (m_{\max} - 1) * p + 1$$

(3) Wähle bei jedem Durchlauf jeweils die  $m_{\max}$  kürzesten Runs und bilde daraus einen neuen Run, bis ein Run übrig bleibt.

- **Schema:**

**Mischordnung**



## Externes Sortieren (5)

- **Annahmen:** keine Beschränkung für Ein-/Ausgabe, gewichtete Runs

- **Beispiel für Heuristik 2:**

$$N_R = 11, m_{\max} = 4;$$

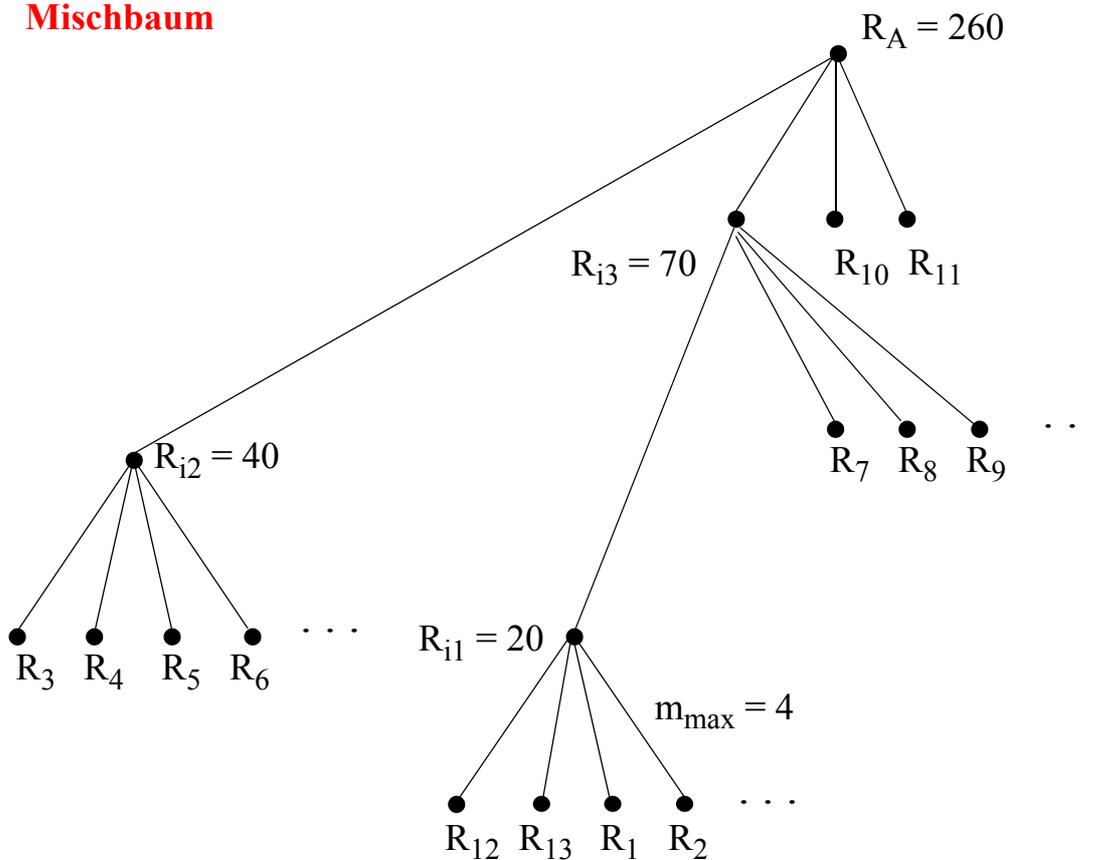
$$\implies p = 4, N_{R0} = 13;$$

Längen von:

$$R_1, \dots, R_7 = 10; R_8, R_9 = 20; R_{10} = 50$$

$$R_{11} = 100; R_{12}, R_{13} = 0;$$

### Mischbaum



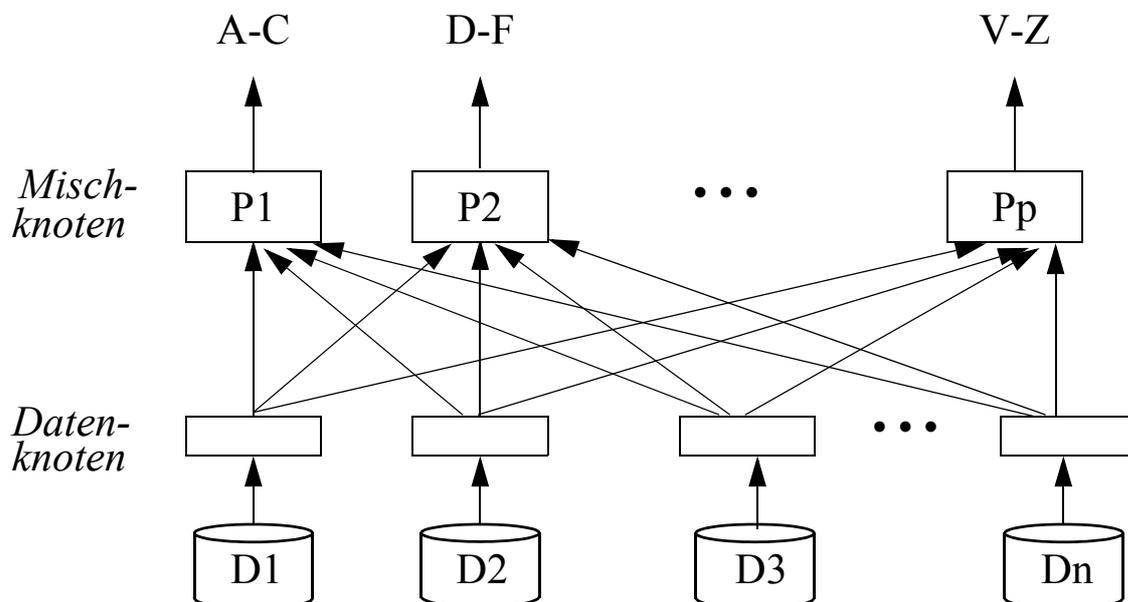
# Parallele Sortierung

- **Anforderungen an parallele Sortierung**

- parallele Eingabe (multiple input)
- parallele Sortierphasen
- paralleles Mischen
- Partitionierung der sortierten Ausgabe (multiple output)

- **Ansatz**

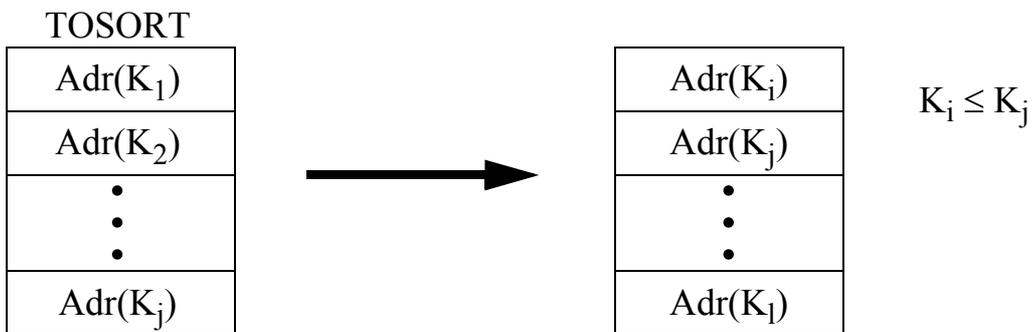
- lokale Sortierung der Partitionen an den Datenknoten
- dynamische Umverteilung der sortierten Läufe unter p Mischknoten
- Umverteilung wird über eine dynamische Bereichsfragmentierung auf dem Sortierattribut gesteuert
- paralleles Mischen in den p Mischknoten
- partitionierte Ausgabe



## Sortieren von variablen Sätzen

- **Datensätze in Dateien sind oft variabel lang**
  - bestehend aus festen (f) und variabel langen (v) Feldern
  - entsprechend sind Sortierschlüssel variabel lang
- **Wie organisiert man den Sortierbereich im HSP?**

S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>
S <sub>4</sub>		S <sub>5</sub>
• • •		



Wann werden die Datensätze in die Sortierreihenfolge gebracht?

- **Weitere Probleme**

- Ausgabeformat weicht vom Eingabeformat ab

Eingabe:

	f	v	v	f	v
	A1	A2	A3	A4	A5

Ausgabe:

	v	f	v	v
	A2	A1	A5	A3

- Sortierschlüssel besteht aus mehreren Feldern, nach denen aufsteigend (+) und/oder absteigend (-) zu sortieren ist, z. B. Sortierschlüssel: A5 +, A3 +, A1 -

## Sortieren von variablen Sätzen (2)

- Sortierfolge bei variabel langen Schlüsseln

aufsteigend: '0'	absteigend: AAA
A	AA
AA	A
AAA	'0'

- Wie vergleicht man zwei zusammengesetzte Schlüssel variabler Länge?

K sei definiert durch  $A5+ | A3+$ :

Was ist das Ergebnis des Vergleichs von K1 und K2?

$$K1 = 12345 | 67$$

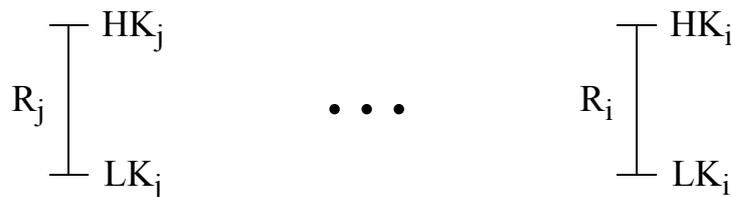
$$K2 = 123 | 6789$$

- Wie kann man eine Vorsortierung ausnutzen?

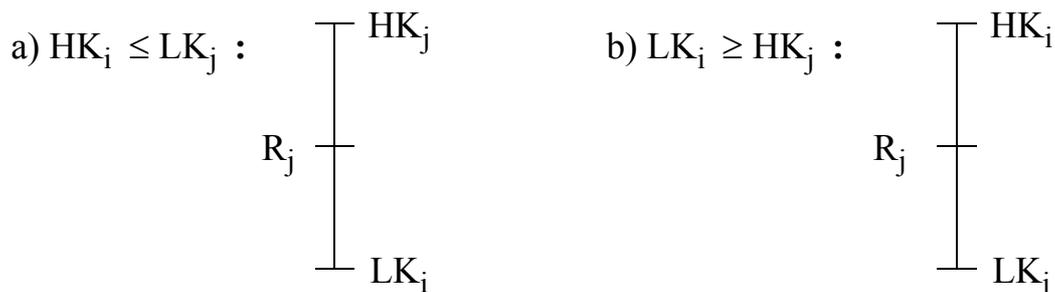
(Einsatz einer Peerage-Technik)

Run  $R_j$  besitzt  $HK_j$  (Highkey) und  $LK_j$  (Lowkey)

**1. Vergleich von  $R_j$  mit dem neu erzeugten Run  $R_i$**



**2. Wenn möglich, Verlängerung von  $R_j$  durch  $R_i$**



**3. Sonst:** separate Speicherung von  $R_i$

# Zusammenfassung

- **Trennung von internen und externen Sätzen und flexible Abbildungskonzepte erforderlich**
- **Transaktionsbezogene Kontroll- und Überwachungsaufgaben**
  - Sie verlangen einen schichtenübergreifenden Informationsfluß
  - Lastkontrolle und -balancierung ist komplexes Forschungsthema
- **Data Dictionary-/Directory-System verwaltet Metadaten zur Beschreibung aller Aspekte der Benutzerdaten**
  - Inhalt, Nutzung
  - Integritätsbedingungen, Zugriffskontrollbedingungen
- **Weitergehende Einsatzmöglichkeiten von D/D**
  - Unterstützung des Programm- und Anwendungssystementwurfs
  - Revisionsunterstützung (Sammlung und Auswertung von zeitbezogenen Informationen), integriertes D/D erforderlich
- **Abbildung von externen Sätzen**
  - Optionen für die Satzspeicherung
  - Trennung von internen und externen Sätzen und flexible Abbildungskonzepte erforderlich
  - implizites und explizites Cursor-Konzept
- **Scan-Technik**
  - Scan-Technik zur satzweisen Navigation auf Zugriffspfaden
  - flexibler Einsatz durch Start-, Stopp- und Suchbedingung sowie Suchrichtung
- **Sortierkomponente**
  - wichtig zur Implementierung relationaler Operationen
  - große Tabellen (Relationen) erfordern Sortier-/Mischverfahren