Zusammenfassung Datenbanken

Heosibel

July 15, 2021

Sommersemester 2021
Professor: Dr.-Ing. Maik Thiele

Dieses Dokument enthält eine Zusammenfassung der Vorlesung Datenbanken vom SS 2021. Diese Dokument ist kein offizielles vom Lehrstuhl und sollte nicht als Referenz verwendet werden. Die Erstellung ist in Vorbereitung auf die Prüfung im SS 2021 aus den Folien geschehen und kann Ungenauigkeit und/oder Fehler enthalten!

Contents

1 Einführung

1.1 Definition Datenbank

- logisch konsistent
- besitzt eine bestimmte Bedeutung
- repräsentiert einen Ausschnitt der realen Welt

1.2 Ziel einer Datenbank

- effektives + effizientes Speichern
- Wiederfinden von Daten
- Analyse von Daten

1.3 Gründe für DBS

- Effizienz und Skalierbarkeit
- Fehlerbehandlung und Toleranz
- Mehrbenutzersynchronisation
- Sicherstellung der Datenintegrität
- Deklarative Anfragesprachen (Benutzer sagt was und nicht wie die Daten geholt werden)
- Datenunabhängigkeit

1.4 ANSI-SPARC-Architektur

- Externe: Definition externer Schemata (Nutzer oder anwendungsspezifische Sichten)
- Logische: definiert die logischen Datenstrukturen und deren Beziehungen
- Interne: Festlegung der Art und Weise der Speicherung

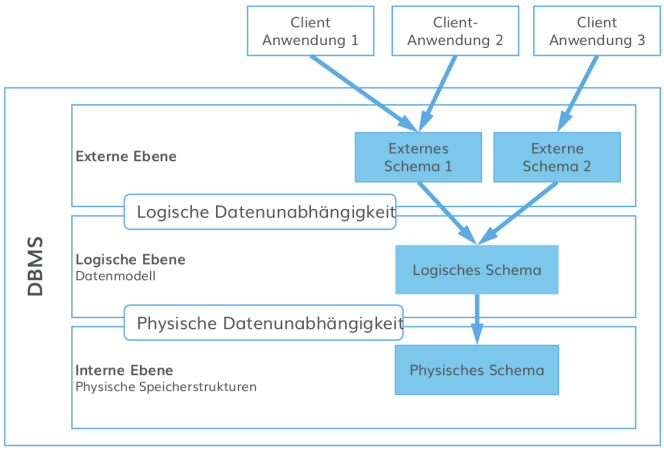


Figure 1: ANSI-SPARC-Architektur

1.5 konzeptuelle Modelle

- Entity-Ralationship-Modell (ER-Modell)
- Unified Modeling Language (UML)

1.6 Logische Modelle

- Hierarchische Modelle
- objektorientierte Modell
- Netzwerkmodelle
- Relationale Modelle
- ...

1.7 Datenverwaltung ohne DBMS (Probleme)

- kein standardisiertes Format
- Duplikate
- hoher Aufwand bei Kombination von Daten
- Dateninkonsistenz
- maßgeschneidert aber ineffizient
- kein Mehrbenutzerzugriff

1.8 Information Retrieval

- Def: finden von Information unstrukturierter Art das eine Informationsnachfrage genügt
- Aufgabe:
  - Suche nach Informationen
  - Strukturierung von meist unstrukturierten Daten
  - Befriedigung des Informationsbedürfnisses eines Benutzers
  - Bewältigung von großen Datenmengen

1.9 Arten von Daten

- strukturierte Daten (z.B. Relationen)
- unstrukturiert Daten
- semistrukturierte Daten

2 Konzeptueller Entwurf

2.1 Phasen des Entwurfes

2.1.1 Konzeptueller Entwurf

- semantisches Modell der Objekttypen
- Beziehungen der Objekttypen

2.1.2 Logischer Entwurf

- Transformation semantischen Modells in DBS-spezifisches Datenmodell
- z.B. ER-Modell

2.1.3 Physischer Entwurf

- Einrichten der Datenbank + internes Schemata
- eventuell laden von Daten

2.2 Entity-Relationship-Modell

- Entität: existiert in der Welt und unterscheidet sich von anderen Entitäten
- Attribut: relevantes Merkmal
- Beziehung: Zusammenhänge zwischen Entitäten

2.3 Funktionalitäten

- One-To-One: (1:1) Beziehung von einer Entität zu höchstens einer anderen
- Many-To-One: (1:N)
- Many-To-Many: (N:M) Es liegt keine Beschränkung vor
- n-stellige Beziehungen (z.B. betreuen: Professoren x Studenten → Seminarthemen)

2.4 Min/Max Notation

- min: jede Entität dieses Typs steht mind. min-mal in Beziehung
- max: jede Entität dieses Typs steht höchstens max-mal in Beziehung
- Sonderfälle
  - min = 0: braucht keine Beziehungen
  - max = *: kann beliebig oft in Beziehung stehen

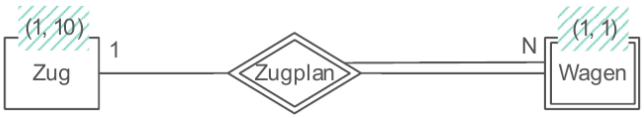


Figure 2: Min-Max-Notation Beispiel Zug

2.5 erweiterte Attribut-Typen

- zusammengesetzte Datentypen
- mehrwertige Datentypen
- abgeleitet Attributwerte

2.6 Spezialisierung und Generalisierung

- jede Instanz der Subklasse ist auch Instanz der Klasse

Generalisierung (bottom-up):	Unterdrückung der Unterschiede der Objekte
Spezialisierung (top-down):	Betonung der Unterschiede
d (disjunkt):	Instanzen der Unterklasse disjunkt
o (overlap):	Instanzen können sich überlappen
totale Spezialisierung:	Jede Instanz der Superklasse muss mind. eine Instanz der Subklasse sein
partielle Spezialisierung:	eine Instanz kann zu einer Subklasse gehören

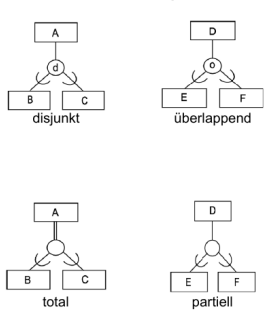


Figure 3: Spezialisierung und Generalisierung Symbole

3 Logischer Entwurf

3.1 Grundlagen

Tupel:	Element einer Relation
Kardinalität:	Kardinalität einer Relation: Anzahl der Tupel in einer Relation
Relation	Menge von Tupeln

3.2 Primärschlüssel

Eindeutigkeit:	jeder Schlüsselwert ist einzigartig und nicht doppelt vertreten
Definiertheit:	Jedes Objekt hat einen Schlüsselwert
Minimalität	es kann kein Teil des Schlüssels weggelassen werden sodass immer noch die Eigenschaften

3.3 Fremdschlüssel

Verwendung:	Ausdrücken von Beziehungen (verweist auf eine andere Relation wo dieser ein Primärschlüssel ist)
Eigenschaften:	Definiertheit und Minimalität

3.4 ER-Modell in Relationales Modell

1. Übersetzung von Entitäten
2. Übersetzung von Attributen
3. Übersetzung von 1:1 Beziehungen

4. Übersetzung von 1:N Beziehungen
5. Übersetzung von M:N Beziehungen
6. Übersetzung von Beziehungen zwischen mehr als zwei Relationen
7. Übersetzung rekursiver Beziehungen
8. Übersetzung von Attributen an Beziehungen
9. Übersetzung von Vererbungsbeziehungen

3.5 Abbildungsvarianten

3.5.1 Horizontale Partitionierung

- jedes Objekt ist genau ein Tupel einer Relation → gleiche ID heißt nicht das selbe Objekt
- Zusammenführung der Gesamtrelation entstehen viele NULL Objekte

3.5.2 Vertikale Zerlegung

- Gesamtheit aller Attribute einer Objektes nur durch den Verbund erhaltbar

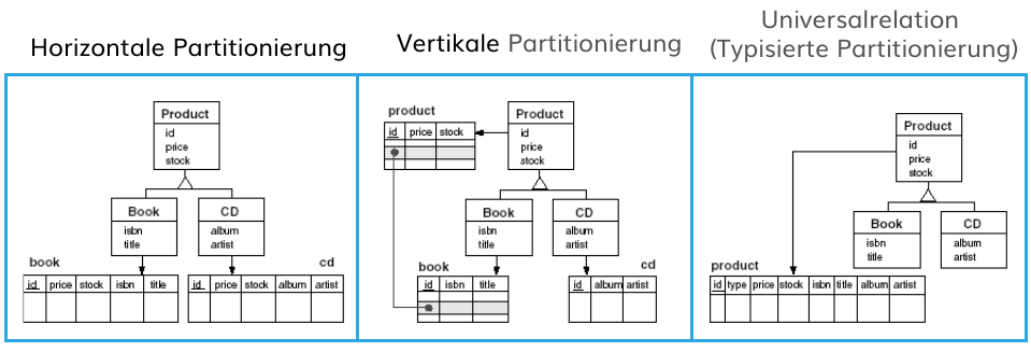


Figure 4: Vertikale und Horizontale Partitionierung

4 Relationale Algebra

- Formale Sprache von Anfrageergebnissen
- Nutzen zur Abfrageoptimierung
- enthält keine Operationen wie z.B. zum Erzeugen oder Löschen

4.1 Basisoperationen

Projektion	- Auswahl von bestimmten Spalten → erzeugen die neue Relational - es dürfen keine Duplikate in der Projektion vorhanden sein (zu SQL verschieden)
Selektion	- ist die Menge aller Tupel die der Selektionsbedingung genügen - Entspricht dem: SELECT * WHERE . . .
Kartesisches Produkt	Verknüpfung von allen Tupel der ersten Relation mit der zweiten
Vereinigung	entspricht der mathematischen Vereinigung von Mengen
Differenz	entspricht der mathematischen Differenz von Mengen

4.2 Abgeleitete Operationen

Durchschnitt

entspricht R-(R-S)

Division

- Bei der Division von R ÷ S bleiben die Attribute übrig, welche in jeder Kombination mit den Attributen aus S in R vorkommen
- $R \div S := \pi_{A-B}(R) - \pi_{A-B}((\pi_{A-B}(R) \times S) - R)$
- Beispielfrage: welche Eltern haben Kinder mit dem Namen Maria (4) und Sabine (2) (siehe 5)

Natürlicher Verbund

Verbund aller Elemente bei denen die Einträge übereinstimmen in in der Spalte mit gleicher Bezeichnung

Theta-Join

- Verbund von Relationen bezüglich beliebiger Attribute und einem Selektionsprädikat
- selbes Ergebnis, kann durch Bildung des kartesische Produkt und nachträglicher Kontrolle der Bedingung erzielt werden

Equi-Join

- entspricht dem INNER JOIN in SQL
- Bildung des kartesischen Produkt und Kontrolle bestimmter Spalten auf Äquivalenz

Left-Outer-Join

- alle Tupel der linken Relation, die kein Join Partner in der rechten Relation haben werden dennoch ausgegeben
- entsprechende Spalten haben NULL Werte

Right-Outer-Join

alle Tupel der rechten Relation, die kein Join-Partner in der linken Relation haben werden dennoch ausgegeben

Full-Outer-Join

alle Tupel sowohl der linken als auch der rechten Relation die keinen Join Partner haben werden trotzdem ausgegeben

Halbverbund

Für zwei Relationen R und S ist das Ergebnis des halben natürlichen Verbundes

R:

Vater	Mutter	Kind	Alter
Franz	Helga	Harald	5
Franz	Helga	Maria	4
Franz	Ursula	Sabine	2
Moritz	Melanie	Gertrud	7
Moritz	Melanie	Maria	4
Moritz	Melanie	Sabine	2
Peter	Christina	Robert	9

S:

Kind	Alter
Maria	4
Sabine	2

$R \div S$:

Vater	Mutter
Moritz	Melanie

Figure 5: Division Veranschaulichung

5 Entwurfstheorie

5.1 Anomalien

Insert-Anomalie	Vermischung zweier Entitätstypen
Update-Anomalie	Redundanz innerhalb der Relation
Delete-Anomalie	Verlieren der Verbindung zu einer Entität durch löschen einer anderen

5.2 Entwurfsziele

- Vermeidung von Redundanzen und Anomalien
- Vermeidung von Informationsverlust
- Effizienzüberlegungen

5.3 Normalisierung Übersicht

- Eliminierung von Redundanzen
- Vermeidung von Änderungsanomalien (Beseitigung von Abhängigkeiten)
- 1. NF: keine mehrwertigen Attribute
- 2. NF: keine Vermischung von Sachverhalten
- 3. NF: keine funktionalen Abhängigkeiten von Nichtschlüsselattributen

5.4 1. Normalform

- alle Attribute der Relation sind atomar (mehrwertige Attribute sind nicht erlaubt)
- Definition atomar hängt vom Sachverhalt ab
- Wertebereich ist STRING und INTEGER

5.5 Funktionale Abhängigkeiten

A und B sind Attributmengen, dann ist B von A funktional abhängig gdw. Für alle real möglichen Relationen r(RS) zu jedem Wert in A genau ein Wert in B gehört.

voll funktional abhängig	alle Attribute in X sind für die funktional Abhängigkeit notwendig
partielle abhängig	es sind nicht alle Attribute notwendig
Superschlüssel	A ist ein Superschlüssel wenn dies alle anderen Attributwerte bestimmt, A muss nicht minimal sein (RS $\rightarrow$ SR gilt immer)
Kandidatenschlüssel	• ist Vollständig und Minimal • es kann mehr als einen in einer Relation geben • bildet den Primärschlüssel einer Relation

Hülle von F
: Eine Hülle F+, ist die Menge aller funktionalen Abhängigkeiten die aus den funktionalen Abhängigkeiten in F ableitbar sind

Armstrong Axiome
: - Vereinigungsregel: falls $A \to B$ und $A \to C$, dann gilt auch $A \to B \cup C$
- Dekompositionsregel: falls $A \to B \cup C$,dann gilt auch $A \to B$ und $A \to C$
- Pseudotransivität: falls $A \to B$ und $B \cup C \to D$, dann auch $A \cup C \to D$

5.6 Zerlegung von Relationen

5.6.1 Verlustlosigkeit

- Die Ursprungsrelation R muss wieder aus den Teilrelationen hergestellt werden können (über joins)
- es liegt kein Informationsverlust vor wenn für die Teilrelationen RS1 und RS2 gilt:
  - $(RS_1 \cap RS_2 \to RS_1) \in F^+$ oder
  - $(RS_1 \cap RS_2 \to RS_2) \in F^+$

5.6.2 Abhängigkeitserhaltung

- die funktionalen Abhängigkeiten müssen auf die Schemata RS1, RS2, ... übertragbar sein
- es wird eine hüllentreue Zerlegung gefordert: $F^+_{RS} = (F_{RS_1} \cup ... \cup F_{RS_n})^+$

5.7 2. Normalform

- jedes Attribut ist entweder
  - teil eines Kandidatenschlüssel oder
  - von jedem Kandidatenschlüssel voll funktional abhängig
- Verstoß gegen 2NF weißt auf Vermischung von verschieden Beziehungen

5.8 3. Normalform

- für alle Abhängigkeit $X \to A$ mit $X \subset R, A \in R, A \notin X$ gilt:
  - X enthält einen Schlüssel von R oder
  - A ist Teil eines Schlüsselkandidaten
- 3NF beseitigt Abhängigkeiten von Nicht-Schlüsselattributen

5.9 Kanonische Überdeckung

1. Schritt: Linksreduktion
   - für alle $A \to B$ in F überprüfe ob durch weglassen von einem Element auf der linken Seite auch B anderes erreicht werden kann
   - Nur Betrachtung von Abbildungen mit mind. 2 Elementen auf der linken Seite
   - z.B.: $AB \to C, A \to E, E \to B$, es kann B weggelassen werden da B in der Hülle(F,A) = A,B,C,E ist
2. Schritt: Rechtsreduktion
   - ein Element auf der rechten Seite kann weggelassen werden und ist dennoch über andere Wege erreichbar

- z.B.: $A \to BC, E \to BF, A \to DE$, B kann weggelassen werden da $A \to E \to B$ abbildet

3. Schritt: Entfernen von FD's mit leerer Menge auf der rechten Seite
4. Schritt: Zusammenfassung von FD mit gleicher linken Seite

5.10 3NF-Synthesealgorithmus

- Zerlegung eines Relationsschemas R mit funktionalen Abhängigkeiten in Relationsschemata $R_1, ..., R_n$ es muss erfüllt sein:
  - kein Informationsverlust
  - Bewahrung der funktionalen Abhängigkeiten
  - $R_1, ..., R_n$ erfüllen dritte Normalform
- Generierung der Zerlegung
  1. bestimme die kanonische Überdeckung F_c der Menge F
  2. für alle FD in F_c erstelle eine Relation (z.B. $A \to BC$ erzeugt R_A = A,B,C)
  3. falls kein Schlüsselkandidat der Ursprungsrelation enthalten erzeuge einen zusätzlichen R_K = K wobei K ein Schlüsselkandidat von R ist
  4. Entferne doppelte Schemata (z.B. $R_1 : A, B, C, D$; $R_2 : A, C \Rightarrow R2$ ist schon in R_1)

5.11 Boyce-Codd-Normalform

- für alle Abhängigkeiten $X \to A$ gilt: X enthält einen Schlüssel von R
- beseitigt Abhängigkeiten unter Attributen, die Teil eines Schlüsselkandidaten sind
- Herstellung von BCNF
  - Zerlegung von R in R1 und R2
  - Erstelle eine List der Determinanten
  - für jede Determinante die kein Schlüsselkandidat ist erstelle eine neue Relation
  - Erhalte nur die Determinante in der Originalrelation

6 Transaktionsverwaltung

- Teilgebiete
  - Synchronisation von mehreren gleichzeitig ablaufenden Transaktionen
  - Recovery von eingetretenen, oft unvermeidbaren Fehlersituationen
- Transaktion: Zusammenfassung von aufeinanderfolgenden DB-Operationen

6.1 Operationen

BOT	Begin of transaction
Commit	Erfolgreiches Beenden einer Transaktion
Abort	Selbstabbruch der Transaktion → Zurücksetzen der Datenbasis
define savepoint	fest definierter Sicherungspunkt
backup transaction	zurücksetzen der aktiven Transaktion auf einen Sicherungspunkt

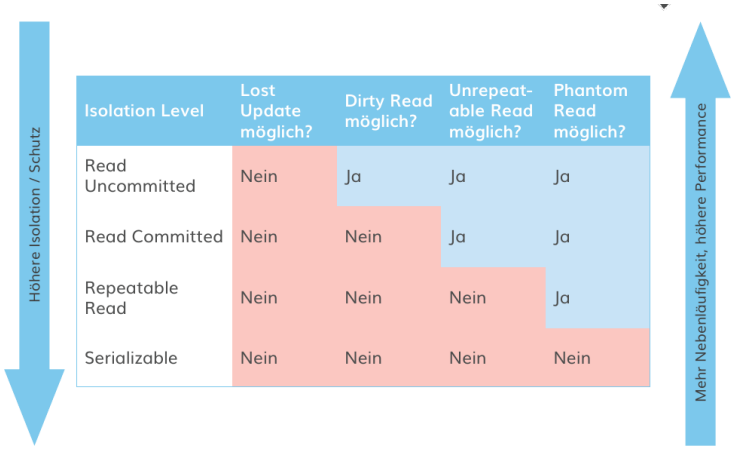
6.2 ACID-Eigenschaften

Atomarität	Transaktionen beginnen mit einem BOT und enden mit EOT (Alles oder nichts Prinzip)
Konsistenzerhaltung	einer erfolgreiche Transaktion garantiert die Konsistenzbedingungen
Isolation	unterschiedliche Transaktionen laufen isoliert voneinander ab
Dauerhaftigkeit	Ergebnisse erfolgreicher Transaktionen müssen persistent sein

6.3 Anomalien ohne Synchronisation

- Verlorengegangene Änderungen (lost update)
- Abhängigkeiten von nicht freigegeben Änderungen (dirty read, dirty overwrite)
- Inkonsistente Analyse (non repeatable read)
- Phantom-Problem

6.4 ANSI-SQL Isolationsstufen



Isolation Level	Lost Update möglich?	Dirty Read möglich?	Unrepeat-able Read möglich?	Phantom Read möglich?
Read Uncommitted	Nein	Ja	Ja	Ja
Read Committed	Nein	Nein	Ja	Ja
Repeatable Read	Nein	Nein	Nein	Ja
Serializable	Nein	Nein	Nein	Nein

Figure 6: ANSI-SQL Isolationsstufen

6.5 Serialisierbarkeitstheorie

- Operationen einer Transaktionen
  - Leseoperation $r_j(A)$
  - Schreiboperation: $w_j(A)$
  - Abbruch a_j
  - Commit c_j
- Ausführungsplan: nebenläufiges Ausführung von mehrere TA
- Konfliktoperationen
  - zwei Operationen arbeiten auf dem selben Datenobjekt
  - eine ist mind. eine Schreiboperation
- Serialisierbarkeit
  - sequenzielle Ausführung führt zu lange Wartezeiten und schlechter Auslastung
  - serialisierbarer Ablaufplan durch

        * Beschränkung der "Parallelität" auf erlaubte Verarbeitungsreihenfolgen
        * Äquivalenz zu einer seriellen Historie
        * serialisierbare Ablaufpläne sind korrekt und frei von Anomalien
    - Achtung: zwei Ablaufpläne können zu unterschiedlichen Ergebnissen führen
- Serialisierbarkeitsgraph
    - Ablaufplan ist serialisierbar, wenn der Serialisierbarkeitsgraph keine Zyklen enthält
    - Serialisierbarkeitsgraph
        * Knoten: einzelne Transaktion
        * Kanten: Abhängigkeiten zwischen 2 Transaktionen

6.6 weitere Eigenschaften von Transaktionen

- Rücksetzbarkeit
    - abgeschlossene Transaktionen dürfen nicht zurückgesetzt werden
    - Abschluss einer TA nur wenn alle TA von denen sie gelesen hat abgeschlossen sind
    - Mindestvoraussetzung für die Dauerhaftigkeit
    - commit Reihenfolge muss eingehalten werden
- Vermeidung kaskadierenden Rücksetzens
    - TA dürfen nur Ergebnisse von abgeschlossen TA sehen
- Striktheit
    - Rücksetzen durch ein Befor-Image
    - veränderte Objekte dürfen nicht verändert werden bevor die aktuelle TA abgeschlossen ist

6.7 Datenbank-Scheduler

- ordnet eingehende Operationen und sorgt für serialisierbare und rücksetzbare Historien

6.7.1 Pessimistischer Scheduler

- verzögert entgegengenommene Operationen
- bei mehreren Operationen Festlegung einer geschickten Reihenfolge

6.7.2 Optimistischer Scheduler

- schnelle Auführung neuer eingehender Operationen
- eventuell später Schaden reparieren

6.8 Synchronisation durch Sperren (Pessimistische Synchronisation)

6.8.1 Pessimistische Synchronisation

- Sperren für exklusiven Zugriff auf Datenobjekte
- zentrale Sperrtabelle für die Nutzungsart
- Arten von Sperren
    - X (eXkclusive)-Sprre (=Schreibsperre)
    - S/R (shared/read)-Sperre (=Lesesperre)

6.8.2 Statisches Sperren

- zum Beginn der TA Anforderung aller Sperren
- Nachteil: Sperren von allem was man brauchen könnte

6.8.3 Dynamisches Sperren

- Anforderung von Sperren nach Bedarf
- Nachteil: Verklemmungen (Deadlocks)

6.8.4 Zweiphasen-Sperrprotokoll

- Wachstumsphase: Sperren werden angefordert aber keine freigegeben
- Schumpfungsphase: Sperren freigegeben aber keine angefordert

6.8.5 Striktes Zweiphase-Sperrprotokoll

- Freigabe aller Schreibsperren erst am Ende einer Transaktion
- Freigabe Lesesperre entsprechend Standard-2PL-Verfahren
- Vorteil: verhindert kaskadierendes Zurücksetzen
- Nachteil: Sperren werden zu lange gehalten

6.9 Deadlocks

- Verklemmungen können bei pessimistischen Methoden nicht verhindert werden
- Deadlock: Abhängigkeit von TA die wechselseitig auf Freigabe von Sperren warten

6.9.1 Deadlockerkennung

- Erzeugung eines Wartegraphen der TA
- prüfen auf Zyklen im Graphen → bei Zyklus eine TA zurücksetzen
- Kriterien für Zurücksetzen: Alter, Fortschritt, Anzahl Sperren, Abhängige TA, ...

6.9.2 Deadlockvermeidung

- versucht eine TA eine Sperre zu erwerben die vergeben ist setzt sich eine TA zurück
- Vermeidungsstrategien: Wait-Die und Wound-Wait
- Wait-Die
  - Fordert T1 eine Sperre an die von einer jüngeren T2 gehalten wird wartet T1 bis die Sperre freigegeben wird
  - Fordert T1 eine Sperre an die von einer älteren T2 gehalten wird, startet T1 mit dem alten Zeitstempel neu
- Wound-wait
  - Fordert T1 eine Sperre an die von einer jüngeren T2 gehalten wird, startet T2 neu und sie Sperre wird an T1 übergeben
  - Fordert T1 eine Sperre an die von einer älteren T2 gehalten wird, so wartet T1 bis die Sperre von T2 freigegeben wird

6.10 Optimistische Synchronisation

- Forderung: TA kann validieren, wenn alle zuvor validierten TA gesehen wurden

6.10.1 3 phasige Verarbeitung

- Lesephase
  - eigentliche TA-Verarbeitung
  - Änderungen werden im den privaten Puffer durchgeführt
- Validierungsphase
  - Überprüfung auf Konflikte mit parallel abgelaufenen TA
  - Konfliktlösung durch Zurücksetzen
- Schreibphase
  - nur bei positiver Validierung
  - Lese-TA ohne Zusatzaufwand
  - Schreib-TA schreiben LOG-Information und propagieren Änderungen

6.10.2 Backward Oriented

- Validierung gegenüber bereits beendete TA
- Nachteile:
  - Aufbewahrung der Write-Sets beendeter Transaktionen nötig
  - hohe Anzahl an Vergleichen bei Validierung

6.10.3 Forward Oriented

- Validierung gegenüber laufenden TA
- Vorteil: Frühzeitiges Rücksetzen möglich → Einsparung von Arbeit
- Probleme: hohe Rücksetzraten möglich

6.11 Recovery

- Alles oder Nichts Prinzip von TA
- Voraussetzung: Sammlung redundanter Informationen während des Betriebes
- Sicherungspunkte: Schnappschuss des Datenbankinhaltes
- Log-Datei: Protokollierung aller Änderungen in einer Datei

6.11.1 Fehlerarten

Transaktionsfehler:	Anwendungsbedingter Fehler (falsche Operation/Wert)
Systemfehler:	Systemzusammenbruch mit Verlust des Hauptspeichers
Gerätefehler:	Zerstörung Sekundärspeicher
Katastrophen:	Zerstörung des Rechenzentrums

6.11.2 Recorvery-Klassen

Lokaler Fehler:	Fehler in einer nicht abgeschlossen TA → lokales undo (R1)
Fehler mit Hauptspeicherverlust:	abgeschlossen bleiben erhalten, andere werden zurückgesetzt
Fehler mit Hintergrundspeicherverlust:	Behebung mittels Archivkopie+Logarchiv (R4)

6.11.3 Ersetzungsstrategien

- STEAL
  - geänderte Seiten können vor EOT in den Hintergrundspeicher verdrängt werden
  - Speicherung von Undo-Informationen nötig (Write-Ahead-Log Prinzip)
- NOSTEAL : Seiten mit schmutzigen Änderungen dürfen nicht ersetzt werden
- FORCE
  - geänderte Seiten werden spätestens bei EOT persistent gespeichert
  - hoher Schreibaufwand, DB Puffer werden schlecht genutzt
- NOFORCE
  - geänderte Seiten können später erst persistent gespeichert werden
  - Einhaltung der Commit Regel
  - Redo-Recovery nach Rechnerausfall

6.11.4 Einbringstrategien

Update in Place:	jede Seite hat genau eine Heimat, wo der alte Zustand überschrieben wird
Twin-Block-Verfahren:	jede Seite hat 2 Seiten im Hintergrundspeicher, die vorletzte wird immer überschrieben
Schattenspeicherkonzept:	nur geänderte Seiten werden dupliziert

6.11.5 Protokollierungsarten

- Logische Protokollierung
  - Undo-Operation → vorherigen Zustand zu erzeugen
  - Redo-Operation → Nachfolgerzustand zu erzeugen
- Physische Protokollierung
  - before-Image des Datenobjektes → statt Undo-Operation
  - fter-Image des Datenobjektes → statt Redo-Operation
- Redo-Informationen: Wiederholung von Änderung möglich
- Undo-Informationen: rückgängig machen von Änderungen

6.11.6 Zusätzliche Log Komponenten

LSN:	Log Sequenz Number - Eindeutige Kennungsnummer
Transaktionserkennung	
PageID:	Seitenkennung der Änderung
PrevLSN:	Zeiger auf vorherigen Log Eintrag

6.11.7 Wiederanlauf nach einem Fehler

1. Analyse: Ermittlung Winner (abgelossene TA) /Looser TA
2. Redo: erneute Ausführung aller Änderungen (Winner und Looser), Vergleich mit LSN
3. Undo: rückgängige Ausführung der Looser TA(Compensations Log Records)

6.12 Sicherungspunkte

- Begrenzung des Redo-Aufwandes nach Systemfehler
- kritisch: Hot-Spot-Seiten (Seiten die fast nie verdrängt werden)

6.12.1 Sicherungspunktarten

- transaktionskonsistente Sicherungspunkte
  - DBS Überführung in ein Ruhezustand → aktive TA beenden, neue verschieben
  - sehr aufwendig, nur im Ausnahmefall möglich
- aktionskonsistente Sicherungspunkte
  - Abschluss aller elementaren Änderungsoperationen
  - Übertragung aller modifizierten Seiten in den Hintergrundspeicher
  - Redo-Informationen können gelöscht werden bis zum Sicherungspunkt, Undo nicht
- unscharfe (fuzzy) Sicherungspunkte
  - modifizierte Seiten werden nicht ausgeschrieben nur deren Kennung
  - Dirty-Pages = Menge modifizierter Seiten
  - MinDirtyPagesLSN: min LSN deren Änderung noch nicht ausgeschrieben wurde

7 Indexstrukturen

7.1 Zugriffsarten

- sequenzieller Zugriff auf alle Sätze
- sequenzieller Zugriff in Sortierreihenfolge
- direkter Zugriff auf Primärschlüssel
- direkter Zugriff auf Sekundärschlüssel
- direkter Zugriff auf zusammengesetzte Schlüssel
- navigierender Zugriffe

7.2 DB-Scan vs Index-Nutzung

7.2.1 DB-Scan

- alle Blöcke müssen gelesen und untersucht werden
- ausreichend bei Anfragen mit großer Treffermenge
- Optimierung durch Prefetching

7.2.2 Index-Nutzung

- Schlüsselwerte werden transformiert (Hash-Verfahren)
- Schlüsselwert werden redundant in eigener Struktur gehalten (z.B. Baum)
- wenn kann Zugriffspfad vorhanden → DB-Scan

7.3 Primär und Sekundärindex

- Primärindex: Dateiorganisationsformen
  - unsortierte Speicherung von Tupel (Heap)
  - sortierte Speicherung von internen Tupeln
  - gestreute Speicherung von internen Tupeln
  - Speicherung in mehrdimensionalen Räumen
  - Normalfall: Primärindex über Primärschlüssel / geclusterter Index
- Sekundärindex
  - redundante Zugriffsmöglichkeiten, zusätzliche Zugriffspfade

7.4 B-Baum der Ordnung k

- jeder Knoten enthält höchstens 2k Schlüssel und mind k Schlüssel
- die Wurzel enthält mind einen Schlüssel → mind Speicherplatznutung $\geq 50\%$
- Knoten mit k Schlüsseln hat k+1 Kinder
- alle Blätter sind auf dem selben Level

7.4.1 Einfügen eines neuen Schlüssels (stets ein Blatt)

- Einfügen des neuen Schlüssel in der entsprechenden Stelle
- kann zu Überlauf führen (s = 2k +1) → splitten des Knoten
- Überläufe können sich bis zur Wurzel propagieren

7.4.2 Entfernen eines Schlüssels

- entweder löschen aus Blatt oder innerer Knoten → Rückführung auf ein Blatt
- Anzahl der Schlüssel nach Entfernen
  - $s > k \rightarrow Stop$
  - $s = k - 1 \rightarrow$ Unterlauf
    - Fall 1: Geschwisterknoten hat $\geq k + 1$ Schlüssel → Ausgleich Geschwisterknoten
    - Fall 2: Geschwisterknoten hat k Schlüssel → Verschmelzung mit Geschwister und Hinzunahme Vaterschlüssel

7.5 $B^+ - Baum$

- jeder Pfad von Wurzel zu Blatt hat die gleiche Länge
- jeder Knoten (außer Wurzel) hat mind k und höchstens 2k Einträge
- alle Sätze werden in Blattknoten abgelegt
- innere Knoten enthalten nur die Verzweigungsinformation aber keine Daten
- jeder Blattknoten hat eine Referenz zu seinen beiden Nachbarn

7.5.1 Einfügen eines Eintrages

Wie im B – Baum nur das die Referenz nach oben geschoben werden ohne die Daten

7.5.2 Entfernen von Einträgen

- Fall 1: $s > k \rightarrow$ Stop
- Fall 2: $s = k - 1 \rightarrow$ Unterlauf: Mische Blatt mit Geschwisterknoten
  - Fall 1: Summe Einträge $\leq 2k \rightarrow$ Zusammenfassung Blätter + Unterläufe behandeln
  - Fall 2: Summe Einträge $> 2k \rightarrow$ Teile Sätze neu auf beide Knoten auf (Hälfte) + aktualisiere Diskiminator im Vaterknoten

7.6 Vergleich B-Baum und B^+-Baum

B-Baum	**B^+-Baum**
Keine Redundanz	Teilweise redundant
Einbettung der Datensätze $\rightarrow$ große Höhe	Blattknotenkette liefert Sortierte Schlüsselwerte
Wenige 1 Schritt Zugriffe in der Wurzel	Geringe Höhe durch hohe Verzweigung

- B-Bäume sinnvoll für Prädikate mit geringem Verhältnis zwischen Ein-und Ausgangskardinalität
- Daumenregel: Grenztrefferrate ca. 5%

7.7 Hashverfahren

Ziele:

- direkter Zugriff nach Schlüsselwert
- Anzahl der Seitenzugriffe nahe 1 $\rightarrow$ Kollisionen unerwünscht
- effiziente Speichernutzung

7.7.1 Divisionsrestverfahren (Restklassenbildung)

- Bitdarstellung der Zahlen
- $h(s) =$ s mod q (q größte Primzahl $\leq |N| \leftarrow$ liefert eine zulässige Adresse)

7.7.2 Faltung

- Zerlegung von k in einzelne Bestandteile
- Deren Verknüpfung additiv, multiplikativ oder logisch
- Ergebnis als Binärzahl interpretieren und dem Adressraum anpassen

7.7.3 Techniken zur Kollisionsbehandlung

1. Verkettung
   - Seperates Verketten
     - Verkettung der Elemente mit gleichem Hash-Wert/Bucketnummer
     - Problem: Speicherung vieler Pointer
   - Überlaufbereiche
     - Buckets fester Größe pro Hash-Wert
     - Verkettung von Buckets $\rightarrow$ weniger Pointer
     - Problem: Variierende Seitenzugriffe beim Suchen
2. Open Addressing (Lineares Sondieren)
   - Mehr Buckets als Schlüssel
   - h(k) als Hash-Funktion für die Position $\rightarrow$ falls belegt Nutzung von weiteren Funktionen $h_i(k)$ mit z.B. = (h(k) + i) mod m
   - Alterantive quadratisches Sondieren $h_i(k) = (h(k) + i^2)$ mod m

3. Mehrfach Hash-Funktionen

- Cuckoo Hashing
  - zwei Tabellen mit je m Elementen
  - zwei Hash-Funktionen h(k), h‘(k)
  - Einfügen eines Wertes
    (a) zuerst in Tabelle 1 mit h(k) → belegt verdränge den aktuellen Werte
    (b) verdrängter Wert in Tabelle 2 mit h‘(k) → falls belegt wieder verdrängen
    (c) Prozess wiederholen falls nötig
  - max Schleifenkonstante verwenden gegen Loops → falls überschritten Neuaufbau mit neuwahl von 2 neuen Hashfunktionen
- Lineares Hashing
  - Folge von Hashfunktionen h0, h1, ...
  - Belegungsfaktor: $F = N/(n * 2^L + p) * b$
    N Anzahl aller Elementen die eingefügt wurden
    n Größe der Ausgangsdatei in Buckets
    L Level
    p Splitzeigerposition
    b Größe der Buckets
  - Ablauf
    (a) Der Pointer p startet mit dem ersten Bucket und wandert nach jedem Split + 1
    (b) ein Split wird durchgeführt wenn der Belegungsfaktor überschritten wird
    (c) Erhöhung des Levels des aktuell referenzierten Buckets
    (d) hinzufügen eines neuen Buckets am Ende
    (e) alle Elemente im gespiteten Bucket werden neu berechnet und neu verteilt

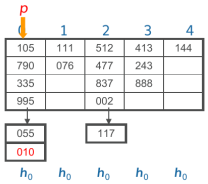


Figure 7: lineares Hashing Schritt 1

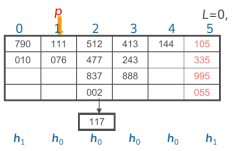
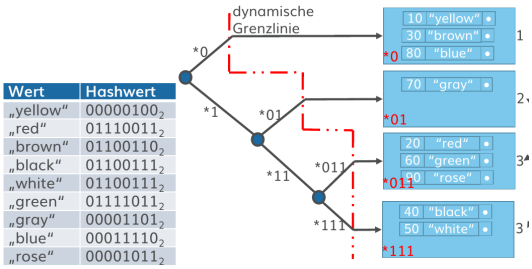


Figure 8: lineares Hashing Schritt 2

- Erweiterbares Hashing
  - Buckets werden nur nach Bedarf angelegt
  - Maximal zwei Seitenzugriffe notwendig
  - Hashfunktion erzeugt einen Pseudoschlüssel für den Schlüssel
  - In Buckets werden nur Sätze gespeichert deren Pseudoschlüssel in den ersten d‘ Bit übereinstimmt
  - d = MAX(d‘) über alle Seiten (globale Tiefe)
  - Einfügen neuer Schlüssel:
    (a) läuft eine Seite über setze d‘ um 1 herauf → verteile alte Seiteninhalt über zwei Seiten
    (b) falls d dadurch um 1 wächst lege neuen Index mit doppelter Zahl von Einträgen an
- Externes Hashing mit Seperator
  - auch bei Überlauf nur 1 Seitenzugriff

- Verwendung einer Signaturfunktion zur Zuordnung
- Für jeden Bucket gibt es pro Key eine neue Signatur
- Ablauf
  (a) Bucket Überlauf → setzen des Bucket Seperator auf die kleinste abgewiesene Signatur
  (b) abgewiesene Elemente werden in den nächsten Bucket verschoben (siehe Vorgabe)
- Einfügen eines Elementes
  (a) ist die Signatur größter als der Seperator → Element kann nicht eingefügt werden → wahl des nächsten Buckets
  (b) ist die Signatur kleiner als der Seperator → einfügen



Wert	Hashwert
„yellow“	00000100_2
„red“	01110011_2
„brown“	01100110_2
„black“	01100111_2
„white“	01100111_2
„green“	01111011_2
„gray“	00001101_2
„blue“	00011110_2
„rose“	00001011_2

Figure 9: erweiterbare Hashing Beispiel

8 Anfrageverarbeitung

- Ziel: Optimierung der Anfrage durch effiziente Anfrageauswertung
- Vorgehen: Verwendung der relationalen Algebra zur Standardisierung des SQL Ausdrucks
- Operatorbaum:
  - Darstellungsform für Ausdrücke der relationalen Algebra
  - Erstellung von unten nach oben
  - in Blätter sind die Relationen, innere Knoten enthalten Operationen und Kanten
- kanonischer Operatorbaum: direkte Ableitung aus einer SQL Abfrage

8.1 Äquivalenzerhaltende Transformationsregeln

1. Aufbrechen von Konjuktionen im Selektionsprädikat
   - $\sigma_{C1 \wedge C2 \wedge ... \wedge Cn}(R) \equiv \sigma_{C1}(\sigma_{C2}(...(\sigma_{C3}(R))...))$
2. σ ist kommutativ
   - $\sigma_{C1}(\sigma_{C2}(R)) \equiv \sigma_{C2}(\sigma_{C1}(R))$
3. π-Kaskaden: Falls $L_1 \subseteq L_2 \subseteq ... \subseteq L_n$ dann gilt
   - $\pi_{L_1}(\pi_{L_2}(...(\pi_{L_n}(R))...)) \equiv \pi_{L_1}(R)$
4. Vertauschen von σ und π
   - Selektion darf sich nur auf die Attribute $A_1, ..., A_n$ der Projektionsliste beziehen
   - $\pi_{A_1,...,A_n}(\sigma_{C1})R)) \equiv (\sigma_{C1}(\pi_{A_1,...,A_n}(R))$
5. $\times, \cap, \cup$ und $\bowtie$ sind kommutativ
   - $R \times S \equiv S \times R$
   - $R \cap S \equiv S \cap R$
   - $R \cup S \equiv S \cup R$

- $R \bowtie_C S \equiv S \bowtie_C R$

6. Vertauschen von σ und $\bowtie$
   - Falls das Selektionsprädikat c nur auf Attribute von R zugreift: $\sigma_C(R \bowtie S) \equiv \sigma_C(R) \bowtie S$
   - Falls das Selektionsprädikat c eine Konjunktion der Form $c_1 \wedge c_2$ ist und c_1 sich nur auf Attribute von R und c_2 nur auf Attribute von S bezieht: $\sigma_C(R \bowtie S) \equiv \sigma_{C_1}(R) \bowtie \sigma_{C_2}(S)$
7. Vertauschen von π mit $\bowtie$
   - Projektionsliste L sei $\{A_1, ..., A_n, B_1, ..., B_m\}$, Attribute A_i sind aus R und B_i aus S : $\pi_L(R \bowtie_C S) \equiv (\pi_{A_1,...,A_n}(R)) \bowtie (\pi_{B_1,...,B_m}(S))$
   - Bezieht sich der Join auf weitere Attribute in R $A'_1, ..., A'_p$ und $B'_1, ..., B'_q$ aus S müssen diese erhalten bleiben: $\pi_L(R \bowtie_C S \equiv \pi_L(\pi_{A_1,...,A_n,A'_1,...,A'_p}(R)) \bowtie (\pi_{B_1,...,B_m,B'_1,...,B'_q}(S))$
8. Operationen $\bowtie, \times, \cap$, und $\cup$ sind jeweils assoziativ. Φ bezeichnet einen dieser Operatoren: $(R\Phi S)\Phi T \equiv R\Phi(S\Phi T)$
9. Operation σ ist distributiv mit $\cap, \cup, -$. Φ bezeihcnet einen dieser Operatoren: $\sigma_C(R\Phi S) \equiv (\sigma_C(R))\Phi(\sigma_C(S))$
10. Operation π ist distributiv mit $\cup$: $\pi_C(R \cup S) \equiv (\pi_C(R)) \cup (\pi_C(S))$
11. De Morgan's Regeln gelten für Join und Selektionsprädikate: $\neg(c_1 \wedge c_2) \equiv (\neg c_1) \vee (\neg c_2)$
12. kartesisches Produkt gefolgt von Selektions-Operation deren Attribute aus beiden Operanden des kartesischen Produktes enthält kann zum Join umgeformt werden : $\sigma_C(R \times S) \equiv R \bowtie_C S$

8.2 Optimierung des Operationsbaumes

1. konjunktive Selektionsprädikate in Kaskaden von $\sigma-$Operationen zerlegen (Regel 1)
2. Selektionsoperationen so weit wie möglich nach unten propagieren (Regel 2,4,6,9)
3. vertauschen der Blattknoten sodass kleine Zwischenergebnisse entstehen (Regel 8)
4. Umformung der X-Operationen gefolgt von eine $\sigma-$Operation in eine Join um
5. Projektionen nach unten propagieren (Regel 3,4,7,10)
6. Zusammenfassung von Operationsfolgen (Regel 1,3)

8.3 Physische Anfrageoptimierung

8.3.1 Anpassung der Daten

- Basisrelation
  - Beispielrelationen sind in einer Datei gespeichert
  - zusätzliche Indexdateien
  - ablegen der Datei nach einem Hashverfahren
- Zwischenergebnisse
  - Wahlfreiheit der Dateiorganisation
  - einfügen von zusätzlichen Operatoren z.B. Sortierung, Index-Generierung

8.3.2 Anpassung Operanden

- Zuordnung von konkreter Implementierung (Algorithmen) zu den Operatoren
- Zuordnung der Operatoren auf den Blättern richtet sich nach der statischen vorgegebenen Dateiorganisation

8.3.3 Verarbeitungsvarianten

- Materialisierung: jeder Knoten wird vollständig abgearbeitet und dann weiter gereicht
- Fließverarbeitung (Pipeline)
  - jeder Operator leitet jedes Ergebnistupel sofort an sein übergeordneten Operator weiter
  - Problem: langsamstes Glied bestimmt die Geschwindigkeit (Pufferung erforderlich)
  - Pipeline Breaker
    - Unäre Operationen: Sortierung, Duplikatelimination
    - Binäre Operationen: Mengendifferenz
    - Join, Union (je nach Implementation)

8.3.4 Iteratorkonzept

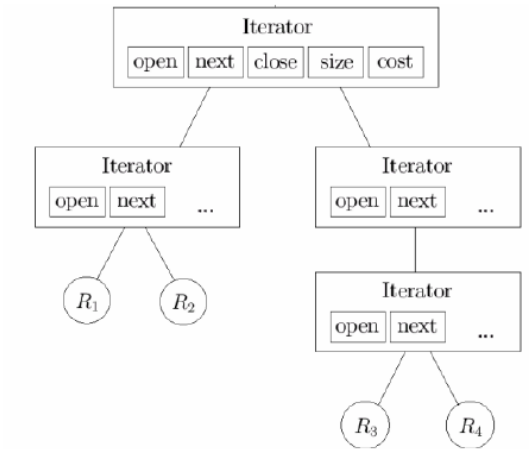


Figure 10: Iteratorenkonzept

8.3.5 Selektion und Projektion

- Planoperatoren für Projektion
  - Spalteneleminierung (meist kombiniert mit Sortierung, Selektion oder Verbund)
  - Duplikateneleminierung durch Gruppierung
- Planoperatoren zur Selektion
  - Nutzung des Scan-Operators
  - Relationen-Scan
  - Index-Scan
  - Auswahl kostengünstigster Index

8.3.6 Nested-Loop Verbund

- Sätze in R und S sind nicht sortiert
- Verwendung eines beliebigen Join Prädikat (z.B. $=, \leq, >, <, \dots$)
- jeder Satz in R muss mit S verglichen werden $\rightarrow$ Komplextiät: $O(N^2)$

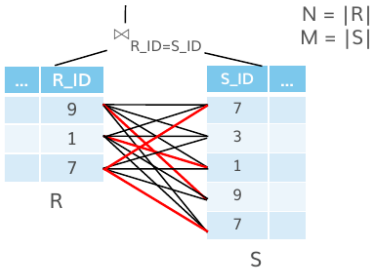


Figure 11: Nested Loop Veranschaulichung

8.3.7 Sort-Merge Verbund

- es existieren Indexstrukturen $I_R(VA), I_S(VA)$
- nur verwendung von Gleichheit als Prädikat
- Komplexität: $O(N \log N)$
- Algorithmus:
  1. Sortierung von R und S nach den Indexstrukturen
  2. Schrittweiser Scan über die Relationen
- Es müssen nicht alle Elemente verglichen werden, da bei einer gößerem Element als das aktuell gestoppt werden kann

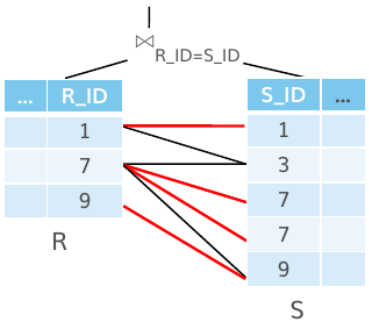


Figure 12: Sort Merge Verbund Veranschaulichung

8.3.8 Hash-Verbund - Classic Hashing

1. Schritt:
   - lesen der kleineren Relation
   - Aufbau einer Hashfunktion $h_A(R.VA)$
   - Aufteilen in p Abschnitte, sodass jeder Abschnitt in den Hauptspeicher passt
2. Schritt:
   - Scan über S
   - Überprüfen für jeden Satz von S mit P_S
   - Erfolg $\rightarrow$ Durchführung Verbund
3. Schritt: Wiederholen von 1. Schritt und 2. Schritt

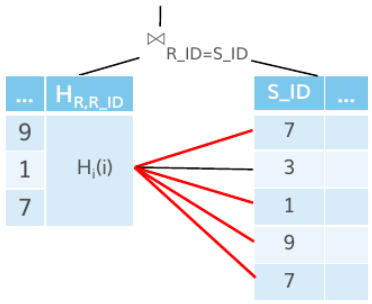


Figure 13: Hash Verbund Veranschaulichung

8.4 Kostenbasierte Auswahl

8.4.1 Optimierungskriterien

- Benötigt eine Kostenfunktion
- Schätzt die Gesamtkosten der Anfrage anhand der Kostenabschätzungen der Einzeloperationen
- wichtige Rolle Selektivitätsabschätzungen

8.4.2 Optimierung

- für jeden Kandidaten-Auswertungsplan müssen die Kosten abgeschätzt werden
- Suche nach schnell einschränkenden Optimierungsverfahren ohne viele Lösungen zu verlieren

8.4.3 Kostenmodell

- Basis Kostenschätzung eines Anfrageplans
- Operatiorspezifische Kostenmodelle
- Verwendung von Statistiken bzw. Schätzungen
- Arten von Kosten
  - Berechnungskosten: CPU Kosten, Pfadlängen
  - E/A Kosten: Anzahl Seitenreferenzen
  - Speicherkosten: Temporäre Allokation
  - Kommunikationskosten: Anzahl der Nachrichten

8.4.4 Join-Reihenfolge

- Beobachtung: Verbundoperationen dominieren in praktischen Anfragen
- Optimierung:
  - Gesucht: äquivalente und effiziente Folgen von 2 Wege-Joins → Ausnutzung Parallelität
  - Beschränkung auf lineare Folgen von Operatorbäumen zur Reduzierung des Aufwandes (left oder right deep tree)

8.4.5 Join-Bäume

Veranschaulichung der verschieden Arten von Bäumen. Bevorzugt werden die Left oder Right-Deep-Trees. (siehe Figure: 14)

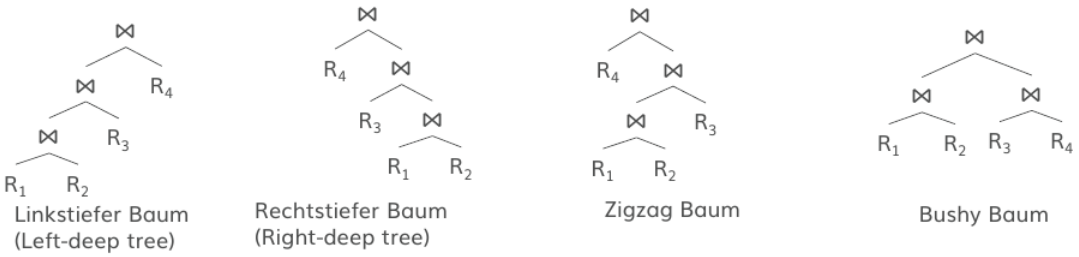


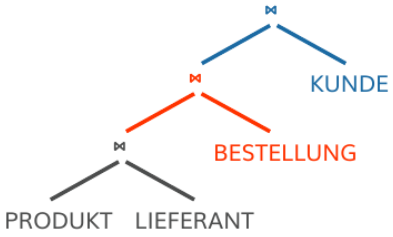
Figure 14: Klassen von Join-Bäumen

8.4.6 Greedy-Suche

- keine Garantie das der optimale Plan gefunden wird
- Vorteil: Anzahl der Pläne ist sehr gering
- Ablauf:
  1. Berechnung der Ergebnisgrößen des Join aller Relationen miteinander
  2. Auswahl des besten Ergebnisses
  3. erneute Berechnung der Ergebnissgrößen mit den restlichen Relationen
  4. Wiederholung bis keine Relation mehr übrig ist

Schritt 1

Plan	Ergebnisgröße
KUNDE ⋈ PRODUKT	5.000.000
KUNDE ⋈ LIEFERANT	1.000.000
KUNDE ⋈ BESTELLUNG	20.000
PRODUKT ⋈ LIEFERANT	5.000
PRODUKT ⋈ BESTELLUNG	20.000
LIEFERANT ⋈ BESTELLUNG	2.000.000



Schritt 2

Plan	Ergebnisgröße
(PRODUKT ⋈ LIEFERANT) ⋈ BESTELLUNG	20.000
(PRODUKT ⋈ LIEFERANT) ⋈ KUNDE	5.000.000

Figure 15: ersten 2 Schritte zur Veranschaulichung

8.4.7 Dynamisches Programmieren

- Problem: greedy Ansatz findet nicht immer den besten globalen Anfrageplan
- Lösung: Bellmannsches Optimalitätsprinzip: Lösung besteht aus den optimalen Lösungen seiner Teilprobleme
- Grundlegender Ansatz:
  - teilen des Problems in abhängige Teilprobleme
  - finden der Lösung für diese Teile
  - Kombinierung der einzelnene Teilprobleme