Verteilte Datenbanksystemen und Client-/ Server-Systeme

Prof. Dr. Klaus Küspert

Semester: SS 200X, WS 200X/0Y

Nutzungsbedingungen und Danksagungen

Dieses Dokument wurde als Skript für die auf der Titelseite genannte Vorlesung erstellt und wird jetzt im Rahmen des Projekts „Vorlesungsskripte der Fakultät für Mathematik und Informatik“ weiter betreut. Das Dokument wurde nach bestem Wissen und Gewissen angefertigt. Dennoch garantiert weder der auf der Titelseite genannte Dozent, die Personen, die an dem Dokument mitgewirkt haben, noch die Mitglieder des Projekts für dessen Fehlerfreiheit. Für etwaige Fehler und dessen Folgen wird von keiner der genannten Personen eine Haftung übernommen. Es steht jeder Person frei, dieses Dokument zu lesen, zu verändern oder auf anderen Medien verfügbar zu machen, solange ein Verweis auf die Internetadresse des Projekts `http://uni-skripte.lug-jena.de/` *enthalten ist.*

Diese Ausgabe trägt die Versionsnummer 2555 und ist vom 4. Dezember 2009. Eine neue Ausgabe könnte auf der Webseite des Projekts verfügbar sein.

Jeder ist dazu aufgerufen, Verbesserungen, Erweiterungen und Fehlerkorrekturen für das Skript einzureichen bzw. zu melden oder diese selbst einzupflegen – einfach eine E-Mail an die Mailingliste `<uni-skripte@lug-jena.de>` *senden. Weitere Informationen sind unter der oben genannten Internetadresse verfügbar.*

Hiermit möchten wir allen Personen, die an diesem Skript mitgewirkt haben, vielmals danken:

- *Jörg Sommer* `<joerg@alea.gnuu.de>` *(2007)*

Inhaltsverzeichnis

Geplante Vorlesungsgliederung/-struktur

1 Einführung/Motivation/Historie

Wann und aus welchen Überlegungen heraus/mit welcher Zielsetzung entstanden verteilte Datenbanksystemen und Client-/Server-Systeme? Welche „Formen“ gibt es grob hinsichtlisch solcher Systeme und wie sind diese zu charakterisieren/abzugrenzen? → Begriffe, Pros und Cons

2 Datenspeicherung, -verteilung, -partitionierung (für globale Relationen in verteilten Datenbanksystemen)

*Partitionierungs*begriff und *Allokations*begriff, Partitionierungsformen (*Wie* kann man Daten *partitionieren* und Betrachtung einer Relation bzw. „darüber hinaus“? Welche *Anforderungen* müssen Partitionierungen erfüllen? Wie bestimmt man geeignete Partionierungen?) ... Quantifizierungsbestrebungen!

3 Schema-Architekturen verteilter Datenbanksysteme

- Ausgehend von bekannter ANSI/SPARC-Architektur (3 Ebenen) → wie sehen Architekturen verteilter Datenbanksystemen aus?
- Unterscheidung u. a. zwischen *lokalen* Schemata und *globalen* Schemata auf konzeptueller Ebene, Partitionierungseben, Allokationsebene → Vielfalt von Möglichkeiten (Orthogonalität?) deutet sich an
- Darstelllung, Klassifikation, Bewertung

4 Anfragebearbeitung in verteilten Datenbanksystemen

- *Basis:* Wissen zu Relationenalgebra/Äquivalenzumformungen/Query Optimization aus DBS 1 (insb.)
- What's new?

  - Globale Anfragen → lokale Anfragen wegen Verteilung (Zerlegung)
  - *Übertragungs*kosten/-dauer als zusätzlicher Faktor
  - Join-Thematik bei verteilten Datenbanksystemen
  - *Partitionierung/Allokation* → (noch) mehr „Stellschrauben“ bzw. *vergrößerter Parameterraum* für den Query Optimizer!

5 Transaktionen in verteilten Datenbanksystemen

- *Basis:* Bekannte Transaktionseigenschaften (Atomarity, Consistency, Isolation, Durability (ACID)), Schedule, Serialisierbarkeit, ...
- What's new?
  - *Globale* Transaktionen → (lokale) *Teil*transaktionen
  - Sicherstellen von ACID auch bei verteilter Transaktionsausführung → *Zwei-Phasen-Commit-Protokoll* (two-phase commit, 2PC): Abläufe und Realisierung-/Optimierungsfragen

6 Synchronisation in verteilten Datenbanksystemen

- *Basis:* Synchronisation-/Sperrthematik aus DBS 1, Zwei-Phasen-Sperrprotokoll (two-phase locking, 2PL), Optimistic Concurency Control (OCC), Deadlock-Thematik
- What's new?
  - *Verteilungs*implikationen globale Sperrtabelle versus lokale (dezentrale) Sperrtabellen
  - *Deadlock*-Erkennung und -Auflösung in Anbetracht von Verteilung und globalen Anfragen (Distributed Deadlock Detection)

7 Replikationsverfahren

- *Basis:* ?? „Replication considered harmful“? → im nicht verteilten (zentralistischen) Fall

  Replikation = bewußt, überlegt herbeigeführte Datenredundanz
- Replikations*verfahen* und u. a. Performance- und Verfügbarkeitsaspekte hiervon

...

Inhaltliche Bezüge zu Forschungsthemen am Lehrstuhl

- *Partitionierung* von Daten (bei *zentralisierten* Datenbanksystemen)! SAP-Projekte J. Nowitzky 1997–2001/02 Partitionierung zur Unterstützung der Datanarchivierung sowie der Performance-Steigerung (u. a.) in Data Warehouses (Partitionierungsausschluss, Nutzung von Parallelität) → Partitionierunsgedanke ist von verteilten Datenbanksystemen in zentralisierte „zurückgewandert“!
- *Replikation* von Daten (bei *mobilen* Datenbanksystemen) Forschungsthema Ch. Gollmick 2000–... *Mobile Datenbanken enthalten fast ausschließlich nur replizierte Daten!!* Größere Variabilität/Dynamik als bei verteilten Datenbanksystemen durch Lokationsabhängigkeit u. a.

Literaturverzeichnis

[1] Dadam, Peter: „Verteilte Datenbanken und Client/Server-Systeme – Grundlagen, Konzepte und Realisierungen“, Springer Berlin, Heidelberg, 1996

[2] Ceri, Pelagatti, 1984

[3] Lang, Lockemann, 1995

1 Einführung/Motivation/Historie

- Forschung zu *verteilten Datenbanksystemen* begann bereits *ab Mitte der 70er Jahre* → nur knapp zeitversetzt also zur Forschung an relationaler Datenbanktechnologie überhaupt (Codd 1970)! . . . 5 Jahre ca.
- Berühmte *frühe Projekte* und Prototypen/Systeme:
  - *System R** (verteiltes System R, IBM San Jose)
  - *Distributed Ingres* (Stonebraker, UC Berkeley)
  - VDN („Verteilte Datenbanken Nixdorf“) gemeinsam mit TU Berlin (R. Munz) → . . . → Adabas D (Software AG) bzw. SAP DB heute Open Source http://www.sapdb.org
  - PORTEL (Uni Stuttgart)

  . . .

  ⇒ haben Eingang in heutige DBMS-Produkte gefunden!!
- *Auslösende Momente für die Forschung* (zu jener Zeit): todo: Bild: Pfeile nach links und rechts, links: Trend zur Dezentralisierung (I), über den Pfeil „primär“ schreiben und Pfeil kräftiger, rechts: Trend zu Integration (II)

  I. Dezentralisierung
     - *Betriebswirtschaftliche* Dezentralisierung (Teileinheiten, Regionalzentren, Geschäftsstellen, . . .) verbunden mit IT-Dezentralisierung und Verarbeitungskapazität „vor Ort“.
     - 70er/80er Jahre: Großrechner (auch) vor Ort; Bsp.: *Großbanken mit Regionalstruktur*
     - „Ist für die Dresdner Bank DIE zentrale relationale Datenbank in Frankfurt die bessere Lösung oder eine Verteilung über die Standorte Hamburg/ Düsseldorf/Frankfurt/. . . ??“

       (Natürlich) nicht trivial beantwortbar, sondern $f(\dots)$:

* *Workload-Charakteristika* lokale versus globale Anteile

  rein global ⟵ ??? ⟶ rein lokal

  In der Extrempositionen würde man keine verteilte Datenbank einsetzen, aber dazwischen vielleicht?

* *Leistungscharakteristika* des eingesetzten bzw, vorgesehenen verteilten Datenbanksystems: Wie gut kommt es mit verteilten Anfragen klar? (Optimizer!), Wie gut mit verteilenten Datenbankänderungen (Tranksaktionsverwaltung) etc.?

* *Administrationsaspekte*: Mit Administration im verteilten Fall verbundene Mehraufwendungen?

* *Verfügbarkeitsfragen* (Availability)

... neue Möglichkeiten, neue Fragestellungen

II. Integration

– Einerseits als „Gegenbewegung“ zur Dezentralisierung, Ausschlagen des Pendels in andere Richtung (Industrietrends)

– Aber auch unabhängig davon zu sehen: Durch Forderung von *integrierter Auswertbarkeit* (dezentral, unabhängig entstandene Datenbestände)

  Durch *Firmenzusammenschlüsse/-kooperationen* (mit Einfluss auf IT, etwa Hypo- und Vereinsbank), Daimler und Chrysler, ...

  → *bottom-up*-Herangehensweise im Gegensatz zu *top-down* bei Dezentralisierung

1.1 Problemstellungen bei Dezentralisierung

Vom zentralen Datenhaltungsansatz zum verteilten.

Forderungen/Ziele:

- (Weiterhin) *Singel-System-Imag* (Verteilungstransparenz): Existierende Anwendungen sollen von der erfolgten Dezentralisierung möglichst nichts merken (Aspekt der Datenunabhängigkeit, der help: Soll das hier auch eine Datenmodellunabhängigkeit sein, so wie im nächsten Abschnitt Ortsunabhängigkeit der Anwendungsprogramme als zusätzliche Forderung).

1 Rechnerknoten → n Rechnerknoten: Anwendungen auf beliebigen Rechnerknoten laufend sollen auf Daten(teil)bestände auf beliebigen Rechnerknoten transparent zugreifen können inklusive (natürlich) Gewährleistung von Transaktionseigenschaften (ACID) → Idealvorstellung

- *Performance*-Steigerung und ggf. auch Steigerung der *Verfügbarkeit* (Availability). Wodurch??
  - Hoher Anteil von *lokalen* Anfragen hilfreich für Performance
  - *Replikation* von Daten hilfreich/nötig für Verfügbarkeit (sonst eher gegenläufige Verfügbarkeitsveränderung zu erwarten . . .)

    Vorteile beim Lesen, Nachteile beim Ändern

1.2 Problemstellungen bei Integration

Von autonomen Datenhaltungslösungen (Einzelsysteme, „Insellösungen“) zur – zumindest logisch – integrierten Lösung.

Forderungen/Ziele/Probleme:

- Problemumfang stark abhängig von initial vorliegender Heterogenität (der Ausgangslösungen):
  - verschiedene Datenmodelle (relational, hierarchisch, netzwerk, Dateien, . . .)
  - gleiches Datenmodell (etwas relational), aber verschiedene DBMS-Versteller (Produkte)
  - gleiches Datenmodell und gleiche Produkte, aber verschiedene Schemata strukturell/semantisch
- *Singel-System-Image* für neuentwickelte, auf der integrierten Datenhaltungslösung beruhende Anwendungen: Orts- und Datenmodellunabhängigkeit idealerweise (z. B. einheitlich relationale Siche auf „alles und über alles“)
- (Weiterhin) Ablauffähigkeit für existierende (Alt-)Anwendungen auf jeweils lokaler Datenhaltung (über bisherige Schnittstellen) → Beibehaltung der *Autonomie* und *Performance* idealerweise

  nichttrivial!!

1.3 Formen und Charakteristika verteilter Informationssysteme

1.3.1 Räumliche Aspekte der Verteilung

geographisch verteiltes Informationssystem

(klassisch auslösende Vorstellung bei verteilten Datenbanksystemen). Realität?

- *Große Distanz* zwischen den Rechnerknoten
- wide area networt
- *Relativ teure,* störanfällige *Verbindungen* ggf. Verfügbarkeitsprobleme möglich → *Kommunikationskosten* spielen wesentliche Rolle
- Hilfreich hier: *Möglichst viel lokale Autonomie der Teilsysteme* mit entsprechenden Workload-Charakteristika
- Beispiel: Verteilte Datenhaltung weltweit über Produktionsstätten/Geschäftsstellen eines Unternehmens hinweg (Nebenbemerkung: *Orthogonal* zur Erörterung Integration vs. Dezentralisierung help: das fehlt auf meiner Kopie)

lokal verteiltes Informationssystem

(nicht der Klassiker unbedingt, aber heute oft bedeutender/eher anzutreffen als geographisch verteilt)

- *geringe Distanz* zwischen den Rechnerknoten
- local area network
- *Relativ schnelle,* wenig störanfellige *Verbindung*, hohe Verfügbarkeit
- Beispiel[1]: todo: Bild
- Kommunikation(skosten) auch hier natürlich ein Thema, aber weitaus geringere Bedeutung als bei geographisch verteilten Systemen → Autonomie der Teilsysteme deshalb nicht so wesentlicher Faktor
- Gründe für lokal verteilte Lösungen:
  - historisch gewachsen, nachfolgende Integration
  - Performance-Verteile (vielleicht . . .)!
  - Skalierbarkeitsvorteile (wenn im obigen Beispiel Anwendung $n + 1$ hinzukommt)!
  - Verfügbarkeitsvorteile (bei Replikation)!

1.3.2 Realisierungsebenen verteilter Informationssysteme

„wo ist die Verteilung implementiert?“ todo: Bild: Dreiteilung vDBMS, Anwendungen, Middleware

verteiltes DBMS

Transparenz todo: Bild

Realisierung der Verteilung in den Anwendungen

todo: Bild

Realisierung der Verteilung in Middleware (Zusatzschicht)

Transparenz todo: Bild

Bewertung:

- vDBMS: „reine Lehre“
- in Anwendung: problematisch
- Middleware: help: das fehlt auf meiner Kopie

1.3.3 Realisierungsformen verteilter DBMS

und ihre Abgrenzung/Einordnung (umfasst auch solche Fälle, die nicht im „engeren Sinne“ zu verteilten DBS zählen)

M. a. W.: Welche (wichtigen) Möglichkeiten gibt es, mehr als einen Rechner (mehr als einen Prozess) zur DBMS-Verarbeitung einzusetzen?

1. Shared everything: gemeinsamer Hauptspeicher, gemeinsame Platten
2. Shared disk: nur gemeinsame Platten
3. Shared nothing: nicht gemeinsam

→ Klassifikation nach Grad der gemeinsamen Nutzung von Resourcen ... Grad der Kopplung (eng ... lose)

Shared-Everything-Architektur

todo: Bild

- Unterstützt von übelichen DBMS à la Oracle, DB2, ...
- Größte Nähe zum klassischen Ein-Rechner-Ein-Prozessor-Szenarium
- *Skalierbarkeit* über Zahl der Prozessoren
- Probleme/Grenzen:
  - Konflikte/Engpässe bei zunehmender Zahl parallel (auf gleichen Hauptspeicher) zugreifender Prozessoren!
  - Cache-Koheränzprobleme!
  - *wenig* Nutzen bzgl. Verfügbarkeit (Availability)!

Shared-Disk-Architektur

todo: Bild

- *Komplette Rechner* wirken zusammen bei der DB-Verteilung
- Jeder einzelne Rechner darf mehrere CPUs (à la Fall a) oben) aufweisen → Skalierbarkeit nun auf zwei Ebenen (Ausbau einzelner Rechner und Ausbau des Rechner-Clusters)!
- großer Vorteil: von allen Rechnern alle Daten prinzipell erreichbar
- Argumente für Einsatz:
  - Skalierbarkeit
  - Hohe benötigte Gesamtverarbeitungsleistung
  - Verfügbarkeit (Availability)

Zur Verfügbarkeitsthematik:

- Hauptgrund für Einsatz von Shared-Disk-Architektur in der Praxis (sagt (nicht nur) SAP)
- Recher-Cluster kann – mit reduzierter Performance – weiterbetrieben werden auch bei Vorhandensein von nur noch $n-1$, $n-2$, ... Rechnern
  - „Absturz“ einzelner Rechner
  - Wartung einzelner Rechner
  - auch ggf. Hinzufügen weiterer Rechner zum Cluster im laufenden Betrieb! $\rightarrow$ $n+1$, $n+2$, ...
- zu lösenden Probleme in dem Zusammenhang:
  - Bei Absturz Betroffensein auf diesem Rechner laufender Transaktionen ($ACID$) $\rightarrow$ Wiederholung dieser (nur dieser) Transaktion auf $n-1$ übriggebliebenen Rechnern
  - Lastverteilung (Load Balancing) anzupassen an veränderte Konfiguration $(n-1)$

Zur Produktwelt (Mindestens) zwei sehr bekannte Produktlösungen für Shared-Disk-Architektur: Oracle Parallel Server (OPS) – heißt heute Oracle Real Application Cluster (RAC) – und IBM DB2 für Parallel Sysplex Mainframe!

Eigenschaften der Oracle-Lösung

- Shared-Disk-Ansatz orthogonal zu Betriebssystemplattform (Unix, Mainframe, ...)
- Oracle im Highend-Bereich (Mainframe) wenig verbreitet

Eigenschaften der DB2-Lösung

- Mainframe-DB2 (z/OS, OS/390) only
- Hardware-/Betriebssystemunterstützung für Kommunikation ($n+1$. Rechner nur dafür)

Allgemein zu lösende Probleme beim Shared-Disk-Ansatz (nur Problem, hier keine Lösungen)

Oder: Warum nicht einfach „normales“ DBMS verwendbar in Shared-Disk-Architektur?

- Load balancing → Lastverteilung über n Rechner (Datenbanklast),

  Affinity-Based-Routing (separate Systempuffer, Erzeilung von Lokalitätsverhalten, Kenntnis von Transaktionstyp, aber: einzelne Rechner trotzdem nicht zum Engpass werden lassen)

- *Konsistenzhaltung* Datenbanksystempuffer bei Datenänderungen (DB-Seite i kann sich gleichzeitig in mehreren Systempuffern befinden), Verwaltungsproblem → buffer coherancy
- *Sperrproblematik* bei (jetzt) Vorhandensein mehrerer Sperrtabellen → rechnerübergreifende Deadlocks ...
- Logging/Recovery ...

Beforscht seit 20 Jahren, Produkte auch fast so lang.

Probleme/Entsprechungen hiervon auch „oberhalb“ des DBMS Worum's geht? Anwendungebene hat u. U. vergleichbare Probleme zu lösen wie oben. Betrachte SAP System R/3 (mySAP.com) – 3-tier architecture:

- Präsentationsebene (Presentation Server)
- Anwendungsebene (Application Server)
- Datenbankebene (Database Server)

→ Database Server kann Shared-Disk-Ansatz nutzen (z. B. DB2 Parallel Sysplex)

→ Skalierbarkeit, Performance, Verfügbarkeit

→ transparent aus Anwendungssicht

Problem: Gleiche Argumentation (Skalierbarkeit, Performance, Verfügbarkeit) auch auf Anwendungsebene → n Application Server teilen sich Arbeit, puffern Daten, benötigen Load Balancing und Affinität, haben Konsistenzerhaltungsproblem. – Berühmtes Beispiel, dass DB-Probleme und -Lösungen auch auf höherer Ebene wieder auftauchen.

Shared-Nothing-Architektur

todo: Bild

- „Richtiges“ vDBMS (wärend a) und b) nicht in jene Schublade gehören, sondern eigene Schublade bilden (insb. b))
- Hochgeschwindigkeitskommunikation bei lokaler Verteilung (LAN-basierte Kommunikation bspw.) → Ansatz gewähle für Hochleistungstransaktionsverarbeitung und/oder Verfügbarkeit
- Architekturell kein Unterschied zu geographischer Verteilung/‚langsamer‘ Kommunikation

Föderierte, verteilte Datenbanksysteme

- Ansatz vor allem gebräuchlich bei *nachträglicher* Integration bereits *existierender*, bislang dezentral organisierter Informationssysteme
- inwiefern Föderation? *Abgabe einiger*, aber *nicht aller* Zuständigkeiten/Kompetenzen an eine übergeordnete Instanz
- So auch beim föderierten, verteilten Ansatz: Existierende Informationssystem bringen nicht alle ihre (bisherigen, lokalen) Daten in globales Verbund ein, sondern nur jene, an denen ein globales Interesse besteht/wo man sich bzgl. des Einbringens verständigt hat → *lokale* Daten (wie zuvor) und – neu – *globale* Daten (die *lokal* gespeichert, aber *global* verfügbar sind)
- *(Teil-)Autonomie der dezentralenen Systeme*, ihre lokalen Daten betreffend und – nur eingeschränkt – die globalen

Schemaintegration bei föderierten, verteilten Datenbanksystemen

todo: Bild

Beobachtungen/Bemerkungen:

- Bei Festlegung des globalen Schemas wird entschieden, welche Teile der lokalen Schemata *global sichtbar* sein sollen → siehe im obigen Beispiel etwas Teile_1 (aus L-DB_1) komplett, Pers_2 (aus L-DB_2) teilweise, Lief (aus L-DB_3) gar nicht (d. h. Lief bleibt rein in lokaler Zuständigkeit, kein Einbringen in globalen Verbund)
- Lokale Ausführungsautonomie wird somit nur teilweie aufgegeben → Föderationsgedanke
- Vertiefung: Stefan Conrad, Föderierte Datenbanksysteme: Konzepte der Datenintegration. Springer-Verlag, Berlin, 1997

Probleme beim föderierten Ansatz u. a. (bereits kurz angesprochen)

- Umfang der Abgabe lokaler Autonomie; betrifft nicht nur à la Bsp. oben Datenumfang (Tabelle komplett, teilweise oder gar nicht), sinder natürlich auch Rechte: Darf globales Ebene nur lesen oder gar viel mehr (Schema ändern?)?
- Probleme bei inhomogenen lokalen Schemata; Dateien, hierarchisch, Netzwerk, relation
- Anfrageoptionierung bei globalen Queries
- Globale Transaktionen ...
- Aber: Praktisch *hochrelevantes* und interessantes Thema; erste Produktlösungen à la *IBM Data Joiner*, verschieden *Gateway-Lösungen* (die nicht in Reinkultur Föderationsgedanken umsetzen), ...

2 Datenspeicherung, -verteilung, -partitionierung (für globale Relationen in verteilten Datenbanksystemen)

Inhaltsübersicht:

- Formen verteiler Speicherung
- Zerlegungsmöglichkeiten (globaler Relationen) und physische Speicherung: Partitionierung und Allokation
- Partitionierungsformen
- Bestimmung geeigneter Partitionen
- Physische Verteilung der Daten (Allokation)

2.1 Vorbemerkungen

Ziele:

- *Datenverteilung* auf logischer Ebene (Partitionierung) und auf physischer Ebene (Allokation)
- ggf. unter Einführung von kontrollierter (!) Redundanz (Datenreplikation)
- möglichst TRANSPARENT realisiert aus Benutzerperspektive

(Gründe für jenen „Aufstand“ wie bereits angesprochen: Erhoffte/angestrebte Performance-Vorteile, Verfügbarkeitsvorteile, evtl. Autonomievorteile; Erfordernisse nachträglicher Integration (Verteilung existiert ...))

2.1.1 Formen verteilter Speicherung (bereits angesprochen)

- Aufteilung an sich zusammengehöriger Daten und Speicherung dort, wo am häufigsten benötigt (Partitionierung globaler Relationen) → *Homogenität*
- *Heterogenität*: Unterschiedliche Repräsentation von Daten derselben Art an verschiedenen Knoten:
  - strukturelle Heterogenität: gleiche Bedeutung, aber verschieden strukturiert
  - semantische Heterogenität: gleiche Bezeichnung, aber unterschiedlicher Bedeutungsinhalt

  (semantische Heterogenität ist der schlimmere, hinterhältigere Fall (schwer zu erkennen und i. d. R. auch schwerer per Automatismus zu beseitigen/durch Abbildung zu berücksichtigen))
- *Redundanz*: Replikation zwecks Performance oder Verfügbarkeit

2.2 Partitionierung und Allokation

Partitionierung: (Prädikative) Beschreibung der Verteilung von Daten auf logischer Ebene (Tabellenebene) → 1. Verteilungsschritt

Allokation: Festlegung des Speicherorts für die Partitionen auf physischer Ebene (Knoten, Table Spaces, . . .) → 2. Verteilungsschritt

2.2.1 Fälle und Zusammenspiel von Partitionierung und Allokation

todo: Bild

Bemerkungen/Beobachtungen

- Relation $R1$ wurde horizontal partitioniert
- Partitionierung erfolgte vollständig (nichts wurde „vergessen“, blieb übrig)
- Partitionierung erfolgte in disjunkte Teile, d. h. Partitionen überlappen nicht (Konsequenz aus Vollständigkeit und Disjunktheit: Jedes Tupel von $R1$ ist genau einer Partition zugeordnet)

  Bei nichtdisjunkten Teilen – überlappenden Partitionierungsprädikaten – läge „versteckte Redundanz“ vor (N. B.: Einige Produkte (Informix) erlauben überlappende Partitionierungsprädikate, andere (Oracle) tun’s nicht.)

- Es wurden fünf Partitionen drei Knoten physisch zugeordnet (im 2. Schritt). Partition P_{12} und P_{13} wurden dabei jeweils mehrfach gespeichert, an verschiedene Knoten → Redundante Allokation

Einschub

vgl. auch Vorlesung Archivierung in Datenbanksystemen

Partitionierung ggf. auch sinnvoll bei Vorhandensein nur EINES Knotens! (auch das erlauben DB2, Oracle, Informix, ...)!

Warum/Wie?

- *Table Space* übernimmt „Rolle des Knotens", d. h. Allokation der Partition zu (verschiedenen) Table Spaces auf u. U. verschiedenen Externspeichern (in Wahrheit wird ohnehin erst Table Space zugeordnet, auch in obiger Abbildung, nicht nur einfach direkt Knoten)

Nutzen?

- *Parallele Suche* auf verschiedenen Partitionen/Table Spaces (Parallel Query Option) *kann* Performance-Vorteil bieten
- Ausblenden von Partitionen/Einschränken der Suche auf bestimmte Partitionen, *kann* Performance-Vorteil bieten
- DROP PARTITION i. d. R. *viel* schneller als DELETE, DELETE, ...
- Administrationsvorteile

..... Ende des Einschubs .....

- Begriff Fragment wird teils anstelle von Partition verwendet/synonym → Informix bspw. (fragment, fragmentation) → hier in Vorlesung eher vermieden
- Wir nehmen disjunkte Partitionierung an, schließen als „versteckte Redundanz" aus!

2.2.2 Horizantale Partitionierung

- Aufteilung einer globalen Relation R in Teilrelationen $R_1, R_2, \ldots, R_p$ (mit gleichen Relationsschema), so dass gilt:

$$R = \bigcup_{1,\ldots,p} R_i$$

(Vollständigkeit steckt somit in der Definition bereits drin, Disjunktheit wäre zusätzlich zu spezifizieren $\forall 1 \leq i, j \leq p \wedge i \neq j \colon R_i \cap R_j = \emptyset$)

- Es werden stets ganze Tupel verteilt
- Hinweis (DBMS-Implementierungsperspektive): Teilrelationen aus Benutzersicht transparent (idealerweise), aus Systemsicht kann eine Teilrelation (intern) wie eine „normale“ Relation gehandhabt werden (vereinfacht Implementierung des Partitionierungskonzepts, keine Sonderbehandlung von Teilrelationen bzw. Sonderbehandlung hält sich in Grenzen).

  Bsp.: Statistiken fpr Teilrelationen wie für Relationen (einfache, genaue(re) Query Optimization), Sperrgranula Teilrelation etc.

Partitionierungsbeispiele

Schema: ANGEST(PersNr, AngName, Gehalt, AbtNr, Anschrift)
ABT(AbtNr, AbtName, Bereich, MgrPersNr, Budget)
TEILE(TeileNr, TeileBez, LiefNr, Preis)
LAGERORT(TeileNr, LagerNr)
LIEFERANT(LiefNr, LiefName, Stadt)

1. Intervall-/Bereichspartitionierung nach Primärschlüssel $\text{TEILE}_1 = \text{SL}_{0 \leq TeileNr < 300}$ TEILE
   $\text{TEILE}_2 = \text{SL}_{300 \leq TeileNr < 500}$ TEILE
   $\text{TEILE}_3 = \text{SL}_{500 \leq TeileNr < \infty}$ TEILE Vollständigkeit und Disjunktheit gegeben?
2. (Bereichs-)Partitionierung nach beliebigen Attribut $\text{LIEFERANT}_1 = \text{SL}_{Stadt = \text{'Hamburg'}}$ LIEFERANT
   $\text{LIEFERANT}_2 = \text{SL}_{Stadt = \text{'Jena'}}$ LIEFERANT
   $\text{LIEFERANT}_3 = \text{SL}_{Stadt \neq \text{'Hamburg'} \wedge Stadt \neq \text{'Jena'}}$ LIEFERANT Disjunktheit und Vollständigkeit? → LIEFERANT_3 übernimmt hier Rolle der Restpartition

Bemerkungen/Verallgemeinerungen dazu:

- Bereichspartitionierung wichtiger (Spezial-)Fall der Partitionierungsdefinition → in diesem Abschnitt ausschließlich betrachtet
- Gibt's noch einfachere Möglichkeiten einer Bereichsdefinition, betrachtet etwas am Bsp. a? Was wird minimal gebrauchst? „TeileNr; 300, 500“ also Partitionsattribut und Bereichsobergrenzen (außer bei letztem Bereich) → i. w. Oracle-Ansatz für Definition von Bereichspartitionierung

Vor- und Nachteile eines solchen „minimalistischen“ Ansatz’?

- Disjunktheit und Vollständigkeit bei der Partitionierung automatisch gegeben, es kann nicht vergessen werden oder versehentlich überlappen
- Kompakte Katalogbeschreibung (Metadaten) fpr die Partitionen → Speicher- und Effizienzvorteile (NB.: Oracle erlaubt *Zehntausende* von Partitionen für eine Tabelle! Die dann natürlich nicht alle auf getrennte Knoten/Table Spaces gelegt werden.)
- Effiziente Bestimmung der betroffenen Partitionen beim INSERT oder update (update kann „Umpositionierung“ von Tupeln erfordern) → bei Zehntausenden von Partitionen und „beliebiger“ Bereichspartitionierung (à la Informix) ist Partitionsbestimmung aufwändig

2.2.3 Abgeleitete horizontale Partitionierung

Bisher betrachtet: Zerlegungsinformation für eine Relation X in dieser Relation selbst enthalten (Primärschlüssel der Relation, anderes Attribut der Relation, ...) → über Partitionierung (welches Tupel kommt in welche Partition?) konnte lokal entschieden werden, rein unter Betrachtung dieser Relation X.

Abgeleitete horizontale Partitionierung: Zerlegungsinformation für eine Relation X muss aus einer anderen Relation Y abgeleitet werden (weil das für die Zerlegung relevante Attribut sich in Y befindet; Relation X und Y stehen in Beziehung zueinander.)

Betrachtet am Beispiel: TEILE(TeileNr, TeileBez, LiefNr, Preis)
LAGERORT(TeileNr, LagerNr) TEILE soll entsprechend dem Lagerort der Teile partitioniert werden (d. h. gemäß dem Attribut LagerNr in der Relation LAGERORT) → nicht mehr lokal entscheidbar (lokaler Blick)

Bildliche Darstellung

TEILE (virtuelle, globale Relation)

TeileNr	TeileBez	...
35181	Schraube M8x3	...
38799	Mutter M8	...
37244	Mutter M7	...
42351	Bolzen B33	...
43749	Schraube M4x6	...
44812	Bolzen B27	...
45438	Schraube M3,5x5	...
⋮	⋮	⋮

LAGERORT

TeileNr	LagerNr
35181	3
38799	1
37244	3
42351	2
43749	2
44812	1
45438	1

TeileNr	TeileBez	...
38799	Mutter M8	...
44812	Bolzen B27	...
45438	Schraube M3,5x5	...
⋮	⋮	⋮

TEILE_1 ($LagerNr = 1$)

TeileNr	TeileBez	...
42351	Bolzen B33	...
43749	Schraube M4x6	...
⋮	⋮	⋮

TEILE_2 ($LagerNr = 2$)

TeileNr	TeileBez	..
35181	Schraube M8x3	..
37244	Mutter M7	..
⋮	⋮	⋮

TEILE_3 ($LagerNr = 3$)

todo: Hier fehlen die Pfeile zwischen den Relationen

Bemerkungen zur abgeleiteten horizontalen Partitionierung

Wann „funktioniert“ dieser Ansatz? (welche Bedingung muss erfüllt sein?) Wurde obiges Beispiel willkürlich gewählt? Beziehung zwischen den betrachteten Relationen X und Y wesentlich, d. h. am Beispiel, für ein Teil muss ein eindeutiger Lagerort existierren $\rightarrow$ hier gegeben wegen der Annahme, dass zu jedem TEILE-Tupel *genau ein* korrespondierendes LAGERORT-Tupel existiert.

Verallgemeinerbar?

Referentielle Integritätsbedingung existiert todo: Bild partitioniert wird Relation X nach Attribut D in Relation Y

für jeden X-Tupel existiert ein eindeutiger D-Wert (in der Relation Y)

Formale Beschreibung der Partitionierung (via Algebra-Sprachausdruck): Benutze (der kompakten Schreibweise wegen) *Semi-Join-Operator* $\text{R SJ}_F \text{ S} := \text{PJ}_{\{\text{Attr}(R)\}}(\text{R JN}_F \text{ S})$, d. h. Semi-Join ist eine Join, wo nur die Attribute des einen Join-Operanten (Relation) im

Join-Ergebnis auftauchen (der zweite Operant wird nur zur Auswertung der Join-Bedingung F herangezogen/betrachtet) → wichtiger Spezialfall des Joins.

Auf unser Beispiel angewandt: $\text{TEILE}_1 = \text{TEILE NSJ (SL}_{LagerNr=1}\text{ LAGERORT)}$
$\text{TEILE}_2 = \text{TEILE NSJ (SL}_{LagerNr=2}\text{ LAGERORT)}$
$\text{TEILE}_3 = \text{TEILE NSJ (SL}_{LagerNr=3}\text{ LAGERORT)}$ mit NSJ als Natural-Semi-Join.

2.2.4 Alternativen zur Bereichspartitionierung

Bisher (horizontale) Bereichspartitionierung diskutiert, am Beispiel aus Abschnitt 2.2.2

todo: Bild

Von DBMS-Produkten darüber hinaus teils angeboten:

1. *Hash-Partitionierung*
   - Sei A Attribut von Relation X und seien a_i Attributwerte von A, so wird die Partition eines Tupels t über $h(a_i)$ mit der Hashfunktion h bestimmt.
   - n Partitionen für Relation X → h bildet ab in Wertebereich $[0, \ldots, n-1]$
   - h kann systembereitgestellt sein oder vom Benutzer/Administrator ggf. auf X zugeschnitten (h_X)

   Vorteile/Nachteile?

   ☼ (Angestrebt) Gleichmäßige Verteilung von Tupeln über Partitionen, was durch Bereichspartitionierung i. d. R. nicht erreicht werden kann und soll.

   🌧 Zerstörung inhaltlicher Zusammenhangseigenschaften (die einzelne Partition enthält „heterogene Daten“)

   → Anwendung weniger bei (geographisch) verteilten Datenbanken, sondern im *zentralisierten Fall*, zur Ausnutzung von Parallelität (vgl. Abschnitt 2.2.1)

2. *Round-Robin-Partitionierung*
   - Verteilung der Tupel jeweils bei Einfügung „reihum“ auf die Partitionen, also nicht werteabhängig!

   ☼ Gleichmäßige Verteilung von Tupeln

   ☼ Billig beim Insert, keine Metadaten etc benötigt

   🌧 Zerstörung inhaltlicher Zusammenhangseigenschaften

   🌧 Partition eines Tupels (bei Suche) nicht bestimmbar

   → Anwendung wie bei Hashtabelle

3. *Random-Partitionierung*

- Zufällige Verteilung der Tupel auf die Partition, also nicht werteabhängig

→ entspricht den Pros und Cons der Round-Robin-Partitionierung, auch hinsichtlich des Anwendungsbereichs

Zusammengefasst: todo: Bild: Baum: „horizontale Partitionierung“ → „werteabhängig“ (→ „Bereichspartitionierung“ + „Hashpartitionierung“) + „werteunabhängig“ (→ Round-Robin + Random), unter Bereichspart: „für (geografisch) verteilte Datenbanken, unter die anderen eine Klammer mit “für ‚sonstige‘ Anwendungsbereiche (lokal verteilt . . . vgl. Abschnitt 1.3)

2.2.5 Vertikale Partitionierung

todo: Bild

- Realisierung über *Projektionen* (im Unterschied zur *Selektion* im Abschnitt 2.2.2)
  $R_1 := PJ_{\{PS,A,B,C\}}\ R$
  $R_2 := PJ_{\{PS,D,E\}}\ R$
- „Wiederherstellen“ der globalen Relation („Sicht“) durch *Natural-Join* über die Partitionen
- Hier angenommen: *Disjunkte* Attributmenden zur Redundanzvermeidung; kein „Muss“ jedoch help: da geht es bei mir auf der Folie nicht weiter
- Ebenfalls angenommen: *Primärschlüssel* als Bestandteil jeder Partition (gestattet einfaches „Wiederherstellen“ der globalen Relation über Natural-Join) kein „Muss“ jedoch help: jedoch was? Mitten im Satz aufhören ist nicht nett

  [NB.: Falls Partitionen bei vertikaler Partitionierung *nicht* den Primärschlüssel enthalten sollen (etwa um auch diese Redundanz zu vermeiden), so muss auf Verlustfreiheit der Zerlegung geachtet werden (vgl. formale Grundlagen hierzu etwas in [1, S. 49ff.]): Originalrelation lässt sich durch natürlichen Verbund der Teilrelation (Partitionen) wiederhersellen, ohne dass Tupel „verlorengegangen“ oder hinzugekommen sind hierbei → *nicht* jede Zerlegung einer Relation ist verlustfrei in diesem Sinne! (ist eigentlich auch Aspekt der Normalformlehre/bei Normalisierung zu beachten, haben wir in DB 2 aber ignoriert)]
- Vertikale Partitionierung auch aus klassischen, zentralisieren Datenbanksystemen bekannt und (implizit) benutzt: *„Abspaltung“ von Large-Object-Attributen* (LOBs, CLOBs, BLOBs, . . .) mit Ablage i. d. R. in separaten Table Space (aus implementierungstechnischen Gründen (Resttupel passt in eine Datenbankseite) und Performance-Gründen) → Vermischung von Partitionierung und Allokation

2.2.6 Gemischte Partitionierung

Kombination der bisher vorgestellten Partitionierungsarten, also vertikal und (vor allem) horizontale Bereichspartitionierung.

Beispiel: ANGEST(PersNr, AngName, Gehalt, AbtNr, Anschrift)
$\text{ANGEST}_{\text{vertraulich}} := \text{PJ}_{\{PersNr,Gehalt\}}$ ANEST
$\text{ANGEST}_{\text{öffentlich}_1} := \text{SL}_{AbtNr<300}\ \text{PJ}_{\{PersNr,AngName,AbtNr,Anschrift\}}$ ANEST
$\text{ANGEST}_{\text{öffentlich}_2} := \text{SL}_{AbtNr\geq 300}\ \text{PJ}_{\{PersNr,AngName,AbtNr,Anschrift\}}$ ANEST

Ergebnis graphisch veranschaulicht: todo: Bild (hier nicht zu erkennen/dargestellt: Schlüssel PersNr in allen Partitionsschemata enthalten → sonst Probleme mit Zuordnung zu entsprechenden Tupeln der globalen Relation, Duplikateleminierung (wobei wir ja wissen, dass Duplikateleminierung „in Realität“ (Structured Query Language (SQL)) nicht automatisch stattfindet im Gegensatz zur Relationenalgebra))

2.3 Physische Verteilung der Daten – Allokation

Aspekte, die hierbei zu betrachten sind: Effizienz und Verfügbarkeit

- Effizienz: Verteilung der Daten (auf Rechnerknoten, Table Spaces, ...) so, dass Verarbeitungskosten (für Anfragen, Änderungen, ...) möglichst gering sind.

  Bemerkungen hierzu/Probleme hierbei:

  - Was steht im Vordergrund, was ist zu berücksichtigen?

    1. *Gesamtverarbeitungskosten* für eine gegebene Last (Workload) – Durchsatz als Optimierungsziel – und/oder
    2. vorgegebene *Obergrenzen* für die Kosten (Dauer) bestimmter einzelner Lastbestandteile

    → meist spielt beides eine Rolle

  - Beispeil:
    * Punkt 1 Materialbuchungen, nachts, Batch-Betrieb: nur Gesamtverarbeitungskosten relevant
    * Punkt 1+Punkt 2 Kontenbuchungen, tags, Dialog-Betrieb: Gesamtverarbeitungskosten relevant („Resourcenverbrauch“) und keine Einzelbuchung soll länger als drei Sekunden dauern (möglichst)
    * Punkt 2 Maschinensteuerung, Realzeit-Betrieb: nur Zeitobergrenzen relevant („harte“ Obergrenzen.)
  - Problem hierbei?

    * Nur Punkt 1 allein gibt es kaum, d. h. DB-Entwurf (inkl. Verteilung/ Allokation) muss meist auch Punkt 1+Punkt 2 berücksichtigen

      [Nebenbemerkung/Einschub: Außer man legt *separate*, redundante Datenbank an für jeweilige spezifische Lastcharakteristika + Anforderungen wie etwas für Date Warehouseing der Fall]

    * Punkt 2 lieben die Datenbänkler überhaupt nicht, DBMS erlauben, insb. im Mehrbenutzerbetrieb, kein garantiertes Einhalten *harter* Zeitschranken für die Dauer von DB-Anweisungen → Locking

  Was geht in Kosten alles ein?

  – Lokale Verarbeitungskosten (I/O, CPU) auf einzelnen Knoten
  – Übertragungskosten

- *Verfügbarkeit*
  – Durch Redundanz (Datenreplikation) → bei der Allokation!
  – help: das fehlt auf meiner Kopie Zusatzforderung, dass Performance-Einbuße bei Ausfall einzelner Knoten „noch tolerierbar“ ist (etwa, in dem Replikate *nicht beliebig* (anderen) Knoten zugeordnet werden, sondern solchen *in der Nähe* oder solchen mit *ähnlichen Charakteristika* (Mainfraim vs. PC))

Es ist prinzipell möglich, ein *mathematisches Modell* aufzustellen, zur Bestimmung einer optimalen Allokation [1, S. 76ff.] → praktisch irrelevant, wegen erforderlicher exakter Angaben zur DB, (vor allem) zur Lastbeschreibung, zu den Verarbeitungskosten, zu den Übertragungskosten etc. (detaillierte Lastbeschreibung ist immer ein Problem, muss bis auf Tupelebene, Verteilung über Partitionen etc. heruntergehen → Angaben liegen nicht vor, können nur abgeschätzt werden help: das fehlt auf meiner Kopie nicht stabil)

3 Schema-Architekturen verteilter Datenbanksysteme

3.1 Grundlagen/Abgrenzung

- bekannt: Drei-Schema-Architektur nach ANSI/SPARC: externes Schema, konzeptuelles Schema, internes Schema

  todo: Bild einfügen

  Ziele vor allem (DBS 1 ...) physische und logische Datenunabhängigkeit! Relationale DBMS realisieren physische in hohem Maße, logische in maßen (View-Update-Problem; Umgang mit Materializes Views noch in den Anfängen (letztere in DB2/Oracle erst seit wenigen Jahren verfügbar))

  Zusätzliche Forderungen (Ziele) der Schema-Architektur in vDBMS:

  - Verteilungsunabhängigkeit: Anwendungsprogramm sieht die Verteilung der Daten, inklusive der Partitionierung und Allokation, nicht. Ebenso wenig die Replikation. ⇒ Änderungen möglich (durch Datenbankadministrator), ohne Anwendungsprogramm anpassen zu müssen!
  - Datenmodellunabhängigkeit: Anwendungsprogramm kann in einer (einheitlichen) Anfragesprache operieren, unabhängig davon, auf welchem Knoten die Daten abgelegt sind. Wann tritt dieses Problem/diese Forderung auf?

    todo: Bild einfügen ⇒ globales Schema + DB-Sprache soll „verstecken“:

    * Unterschiedliche lokale Datenmodelle, Anfragesprachen, strukturelle + semantische Heterogenitäten
    * Partitionierung
    * Replikation
    * Allokation

    Für Datenbankadministrator/Query Processor erforderlich sind:

    * Informationen über Partitionierung, Allokation, Replikation, Kopien-Aktualisierungs-Strategie (wann, welche Reichenfolge, ...)

Fallunterscheidung im Folgenden: todo: Bild einfügen

3.2 Homogene, präintegrierte Datenbanksysteme

Charakteristika: Datenbanksystem, das von vornherein als verteiltes Datenbanksystem (anstelle eines zentralen Datenbanksystems) realisiert wird → Grüne-Wiese-Ansatz

→ Keine Altlasten (Anwendungsprogramme, die bislang (auf lokalen Datenbeständen) laufen und weiterhin laufen sollen)

→ (Top-Down-)Datenbankentwurf: (E/R →) globale Relationen → Partitionen → Allokation

→ Homogene lokale Schemata (auch keine strukturellen/semantischen Heterogenitäten)

→ einfachster und friedlichster Fall

3.2.1 Schema-Grobarchitektur bei homogenen, präintegrierten Datenbanksystemen

todo: Bild einfügen: AP → globale externe Schemata → globales Schema (GKS, GPS, GAS) → lokales konzeptuelles Schema → lokales internes Schema

- Globales konzeptuelles Schema (GKS) realisiert globale Außensicht
- Globales Partitionierungsschema (GPS) beschreibt die Partitionierung der Relationen des GKS
- Globales Allokationsschema (GAS) beschreibt die physische Platzierung der Partitionen (also die Allokation)

Was jeweils zu den verschiedenen Schemata dazugehört:

- GKS: Globaler Katalog ABT(AbtNr, AbtName, Bereich, MgrPersNr, Budget)
- GPS: Globales Partitionierungsschema

Globale Relationen	
Globale Relation	Defintion
ABT	ABT1 UN ABT2
⋮	⋮

Partitionen			
Partition	Attribute	Globale Relation	Selektionsprädikat
ABT1	AbtNr, AbtName, Bereich, MgrPersNr, Budget	ABT	AbtNr≤ 300
ABT2	AbtNr, AbtName, Bereich, MgrPersNr, Budget	ABT	AbtNr> 300
⋮	⋮	⋮	⋮

- GAS Globales Allokationsschema

Allokationen			
Partition	Lokaler Name	Primärkopie	...
ABT1	ABTEILUNG@KNOTEN_A	Knoten_A	...
ABT2	ABTEILUNG@KNOTEN_C	Knoten_C	...
⋮	⋮	⋮	⋮

- lokales konzeptuelles Schema:

ABTEILUNG(...)		ABTEILUNG(...)
⋮		⋮
Knoten A	Knoten B	Knoten C

Hinweise:

- Vereinfachte Darstelllung in obiger Abbildung, speziell Partitionen würden im Katalog ‚real' sicher anders dargestellt werden (Schemainformation, bei horizontaler Partitionierung, nur einmal – bei globaler Relation –, dafür Algebraausdrücke zur Ableitung der Partitionen hieraus)
- worauf muss der Optimizer Bezug nehmen beider Übersetzung einer (globalen) Anfrage aus AP_{Gi}?
  - Globales Schema (GKS, GPS, GAS)
  - lokalen Schemata (Lokales konzeptuelles Schema (LKS)-x, Lokales internes Schema (LIS)-x) nur dort liegen die Informationen über die lokale Speicherung (physische Realisierung, Datenverteilung auf Table Spaces, Indexe, Performance-Charakteristika bei leistungsmäßig unterschiedlichen Knoten (CPU-Geschwindingkeit, Caches, Plattencharakteristika) → benötigt für Performance-Vorhersage (Kostenschätzung)) vor

3.3 Heterogene, präintegrierte Systeme

Charateristika:

- von vornherein als verteiltes System (getrennte Datenverwaltung) konzipiert
- heterogene lokale Systeme zugrundeliegend

Warum handelt man sich solche Probleme (Heterogenität!) ‚freiwillig' ein/geht diesen Weg? → Systeme, wo unterschiedliche (Teil-)Aufgabestellungen innerhalb eines Gesamtsystems berücksichtigt werden müssen!

Beispiel: todo: Bild einfügen: CAD-System mit Verwaltungsinformationen in einer DB und Geometriedaten in einer anderen DB

Gründe für diesen Ansatz:

1. historisch Gewachsen, etwa aus erweiterten Anforderungen an ein CAD-System heraus, zusätzliche Berücksichtigung betriebswirtschaftlicher Anforderungen (Produktdaten, nicht technisch)
2. System-/Datenaufteilung aus Performance-Überlegungen (Polygondaten u. U. kosteneffizienter direkt in Dateien abgelegt als über Umweg über DBMS, auch in Ermangelung adäquater DBMS-Modelle- und -Speicherkonstrukte → Objektrelationalität als Lösung? → eNF²-artige Strukturen in heutigen relationalen Produkten nur rudimentär vorhanden und mit nichtadäquater Speicherung (Large Objects (LOBs), Side Tables))
3. System-/Datenaufteilung aus Modularisierungsüberlegungen (alles, auch „Spezialfälle“, wirklich in einem großem DBMS realisieren oder modular abtrennen) [Bsp.: DB2 Text Extender hat (Text-)Index und Indexverwaltung „außerhalb“. Gründe? Punkt 1, Punkt 2]

3.3.1 Exkurs: DB2 Text Extender

todo: Bild einfügen

- Leichte Variation des obigen Ansatz: obige Sicht (Anwendung) „spricht nicht direkt“ mit zwei Datenhaltungssystemen, sondern – in erweiterter Sprache (User Defined Function (UDF)-Verwendung!) – nur mit einem, dem DBMS (welches die „Verzweigung“ in das andere System unter der Decke erledigt)
- Texte in Character Large Objects (CLOBs) in den DB-Tabellen (vertikal Partitioniert!), Textindexe separat in Dateien
- Wie funktioniert Textsuche?
  - Per UDF-Verwendung in SQL-Select-From-Where (SFW)-Ausdruck CONTAINS (...) mit Textspalte + Text Search Expression
  - DBMS kennt/analysiert Text Search Expression nicht! (Syntax, Semantik)
  - UDF reicht Text Search Expression durch (verzweigt) an separate Komponente Search Manager, dieser liefert Trefferliste zurück
- Wie funktioniert Texteinfügung/-änderung/-löschung? Wo liegt das Problem? Aktualisierung des separat liegenden Textindex!!
  - Trigger-Verwendung (Insert-/Update-/Delete-Trigger)
  - Trigger rufen antsprechende Prozeduren auf, die in Search Manager (Index Maintenance) verzweigen und dort Indexaktualisierungen vornehmen
- Was IST der Text Extender (help: Hier kann ich was auf meiner Kopie nicht lesen ein DB2 Extender allgemein)?

- Vordefinierte UDFs
- Vordefinierte Trigger
- Administrations-/Installationshilfmittel (Skripte)
- Separater Search Manager

Alles paketiert = Extender

(gleiches Prinzip bei Oracle Cartridges und Informix Data Blades) (andere bekannte Extender: Image Extender, Spatial Extender)

Vor- und Nachteile des Text-Extender-Ansatz'/Separierung?

- ☼ „Einfache" und schnelle Integration verhandener Systeme – DBMS einerseits, Information-Retrival-System andererseits – unter teilweiser Beibehaltung der separaten Datenverwaltung
- ☼ Modularisierung; keine Erweiterung des DBMS nötig um Spezialindex (Text) → keine Vermischung von allgemeiner DBMS-Funktionalität („universell nutzbarer" Funktionen) und Spezialfunktionalität eines bestimmten Anwendungsbereichs (hier: Text)
- 🌤 kein DBMS-Overhead (Schichtenmodell . . .) für Textindex
- 🌧 Textindex außerhalb der Datenbank; problematisch in Bezug auf Transaktionseigenschaften (ACID), Locking, . . . → Wiederanlauf im Fehlerfall (Crash, Externspeicherversagen, . . .) skriptgestützt
- ☁ Schutzaspekt (gewisse Daten – Textindex – nicht unter DBMS-Kontrolle/-autorisierung) → in der Praxis durch speziellen DB-Server gelöst
- 🌧 Problem bei Anfrageoptimierung, UDF als Black Box (extern!) aus Sicht des Optimizers; Selektivitätsabschätzung bei Textsuchprädikaten → Defaults, Optimizer Hints

Resümee (dito [1, S. 95]): Vollständig integrierter Ansatz wäre vielleicht „schöner", eher der reinen Lehre entspechend, aber Tendenz geht eher zur Modularisierung, Andocken separater Datenhaltungs- und Spezialbausteine, aber unter einheitlicher Oberfläche (SQL + UDF) → gegen die eierlegende Wollmilchsau

. . . . . Ende des Exkurs' . . . . . .

Schemaarchitektur: Bei Ansatz à la Abschnitt 3.3 Schemaarchitektur im Prinzip à la Abschnitt 3.2, aber mit möglicherweise aufwendigem Mapping (Schema- und Sprachtransformationen zur Überbrückung der Heterogenität)

3.4 Postintegrierte Systeme

(Unterscheidung zwischen Homogenität und Heterogenität erst weiter unten)

Ausgangspunkt allgemein: Nachträgliche Integration vorhandener Datenbanken zu einer verteilten Datenbank.

Problem: Bisherige, lokal operierende Anwendungen können nicht einfach – oder überhaupt nicht – auf neues, globales Schema umgestellt werden („auf Ebene des globalen Schemas angehoben werden“) → ggf. schrittweise Umstellung (Anhebung), aber Aufwand, Performance-Nachteile

Sichtweise der lokalen Anwendungen: „Stülpt rüber, was Ihr wollt, aber lasst uns in Ruhe/unverändert weiter leben.“ → hinsichtlich der Funktionalität, möglichst auch der Performance!

Mögliche Herangehensweisen:

1. voll integrierte lokale Datenbanken, alle lokalen Daten werden nur global zur Verfügung gestellt

   todo: Bild einfügen

   Vorgehensweise:

   a) Festlegung von KS-neu, den neuen einheitlichen (globalen) konzeptuellen Schemas

   b) Transformationsanpassungen „nach unten“, von KS-neu auf die vorhandenen internen Schemata IS-i bzw. Modifikation des IS-i

   c) Transformationsanpassungen „nach oben“, Abbildungsanpassungen zur „Simulation“ der bisherigen ES_{ij} auf dem neuen konzeptuellen Schema

   Bemerkungen/Bewertung:

   ☼ Anwendungen bleiben *unbeeinfluss*, bekommen weiterhin ihre externen Schemata ES_{ij} dargeboten. ToDo: Anpassung der View-Definitionen!

   ☼ Anpassungen der internen Schemata, wenn nötig, unkritisch (Datenunabhängigkeit, wie immer . . .)

   🌧 Performance-Probleme, wenn die bisherigen ES_{ij} u. U. aufwendig „simuliert“ werden müssen, um – auch auf dem neuen konzeptuellen Schema KS-neu – den Anwendungen ihre alte Umgebung „vorgaukeln“ zu können!!

   Wann machen sich solche Performance-Probleme besonders bemerkbar (beispielsweise)? Bei großer Heterogenität der Ausgangssysteme, der KS-i, der IS-i!!

   Beispiele für große Heterogenität:

- Ausgangssysteme teils hierarchisch (Information Managment System (IMS)), teils relational (KS-1, KS-2). KS-neu relational → aufwendige Abbildungen, um für Anwendungen „hierarchisch ‚on top of' rational" zu simulieren

  (ähnliches Problem auch bei Datenbankmigration: es gibt Ansätze (Praxis!), um nach erfolgter Daten*bank*migration (Bestand hierarchisch → relational) bisherigen Altanwendungen weiter die Existenz der hierarchisch Datenbank „vorzugaukeln": Umsetzung (Runtime-Abbildung) von Data Language/One (DL/1)-Calls auf SQL-Sprachanweisungen, performancemäßiges Desaster!! aber es läuft ...)

- Ausgangssysteme sind bereits relational (KS-i), dito KS-neu: Ursprungstabelle (in KS-1 etwa) nun aber in vielleicht viel umfangreicherer KS-neu „versteckt" oder – schlimmer – dort aufgeteilt über mehrere Tabellen etc

- Umstellungsproblem: Übergang von KS-i auf KS-neu müsste zu *einem* Zeitpunkt geschehen („big bang"), alle Anpassungen/Modifikationen ES_{ij}-KS-neu entsprechend verfügbar: keine schrittweise Migration in neue, integrierte Welt möglich → kaum praktisch durchführbar

2. partiell integrierte lokale Datenbanken

   - nicht alle lokalen Daten (aus KS-i) werden global zur Verfügung gestellt (in KS-neu)

   - lokale Anwendungen dürfen weiterhin direkt auf lokalen Daten operieren (ohne Umweg über KS-neu, sozusagen „nativ" ohne „Simulation")

   Was ist NEU vor allem? → Lokale Repräsentationsschemata (LRS) = Exportschemata

   todo: Bild von Folie 73 einfügen

   Zwei Aufgaben/Ziele bei der LRS-Verwendung:

   a) Legt fest, welcher Ausschnitt der lokalen Datenbank global zur Verfügung gestellt werden soll (im einfachsten Fall, welche lokalen Relationen „globalisiert" werden sollen, aber auch feinere Abstufungen strukturell/werteabhängig möglich)

   b) Führt Vereinheitlichung durch: Unterschiedliche lokale strukturelle Formen und Semantiken werden datenbankweit (knotenübergreifend), einheitlich dargestellt! (Export berücksichtigt globel Vorgaben bzgl. Struktur und Semantik)

   → Homogenisierung der (Darstellung der) lokalen Schemata!!

   → Annäherung – durch jenen Zwischenschritt – an Situation (Integrationsaufgabe) im präintegrierten homogenen Fall oben, erkauft durch u. U. sehr aufwendige (rechenintensive) Transformationsregeln, Update-Probleme, ...

   Weitere Hinweise/Bemerkungen zum neuen Vorgehen:

- „Langer Weg“ (sechs Kanten im obigen Graphen) zwischen AP_{Gi} (globaler Anwendung) und Datenbestand, performancerelevant (s. o.) und mit möglicherweise funktionalen Auswirkungen (Einschränkungen, Update-Problematik)
- Lokale externe Schemata (LES) unverändert, lokale Anwendungen (AP_{Ai}, AP_{Bj}) dito
- Lokale Exportschemata, lokale externe Schemata müssen natürlich nicht disjunkt sein: ggf. kann gleicher Datenausschnitt global (mit global angepasster Struktur und Semantik) und (weiterhin) lokal (zum „Schutz“ der vorhandenen Anwendungen/Weiternutzung) zur Verfügung gestellt werden

  → Disjunktheit macht Dinge jedoch einfacher, vor allem bei Schemarevolution (also wenn die global und die lokal genutzten Ausschnitte nicht überlappen)

3.4.1 Schemintegration

Aufgabe: Definition eines „integrierenden“ globalen Schemas ausgehend von lokalen (konzeptuellen) Schemata, die unabhängig voneinander entstanden sind:

- verschiedene Attributnamen (für gleiche Information)
- verschiedene Wertebereiche
- verschiedene strukturelle Repräsentation (Tabellenverteilung, LOB vs. Side Table etc.)
- verschiedene Schlüsselsemantiken/Wertevergabe, Primärschlüssel, Schlüsselkandidaten, Fremdschlüssel betreffend
- verschiedene Integritätsbedingungen
- Widersprüche zwischen den Daten an verschiedenen Knoten

⇒ NICHT TRIVIALES Problem

- taucht ähnlich auch z. B. bei Data Warehousing auf (also nicht ‚exotisch‘)
- Vorgehensmodell für Schemaintegration gefordert

4-Phasen-Vorgehensmodell für die Schemaintegration

Präintegration, Vergleichsphase, Vereinheitlichungsphase, Restrukturierungs- und Zusammenfassungsphase

1. Präintegration
   - Bestimmen/Festlegen der zu integrierenden Schemaausschnitte (in E/R-Terminologie: Edity-Typen, Beziehungstypen, . . .)

- Festlegen der Vorgehensweise: binäre oder n-stellige Integration (Mehrschritt vs. Einzelschrittverfahren)

  todo: Bild einfügen

  Pros und Cons:

  - Binäre Integration:
    - erlaubt Vorgehen in kleinen Schritten (Arbeitseinheiten)
    - Führt nicht (oder nicht einfach) zum globalen Optimum: Frühe, lokalgetroffenen Entwurfs-(Integrations-)Entscheidungen in Schritt i erweisen sich u. U. in Schritt $j \gg i$ als ungünstig → Nacharbeiten, Revidieren u. U.
  - n-stellige Integration
    - Komplexität des „großen Schritts“
    - Dann sinnvoll, wenn Ausgangsschemata S_i *stark voneinander abweiche* (um sich erst einmal bzgl. des globalen Zielschemas „zusammenzuraufen“ und nicht einfach „kleinkleinloszuwurschteln“, wie bei binärer Integration die Gefahr besteht bei lokaler, „kurzsichtiger“ Betrachtungsweise)

  → eventuell Verwendung von binärer und n-stelliger Integration

2. Vergleichsphase Vor allem: Auflösung von *Namens-* und *Strukturkonflikten* (für die jeweils aktuell betrachteten Ausgangsschemata S_i)

   Homonym- und Synonymproblem, Beispiel dafür todo: Bild einfügen: E/R-Schema
   Homonym: EIN Begriff mit VERSCHIEDENEN Bedeutungen, Synonym: VERSCHIEDEN Begriffe für EINEN Sachverhalt

   Was kann beim Erkennen von Homonymen und Synonymen helfen? → Betrachtung von Attributen, zugehörigen Beziehungstypen ... allgemein des „Kontexts“

   (im obigen Beispiel für Homonyme: Die beiden Entity-Typen Ausstattung besitzen mit hoher Wahrscheinlichkeit verschiedene Attributmengen bzgl. Anzahl, Attributnamen → deutet auf Homonyme hin) → aber Vorsicht: Homonymproblem kann sich natürlich auf Ebene der Attribute fortsetzen, d. h. (einige) Attribute heißen links und rechts möglicherweise gleich, aber besitzen verschiedene Bedeutungen!

   Tückisch bzgl. Synonymen: Kunde und Klient (im oben Beispiel für Synonyme) werden mit hoher Wahrscheinlichkeit verschiedene Attributmengen aufweisen, d. h. sie haben unterschiedliche (Schema-)Entwicklungen durchemacht (d. h. Synonyme „natürlich“ nicht immer sofort als solche erkennbar)

Weitere Inhalte der Vergleichsphase:

- „Verhältnis“ der S_i zueinander:
  - äquivalent?
  - überlappend?
  - disjunkt?
  - enthalten (S_i in S_j)?
- Konfliktarten auf Strukturebene (Schema):
  - Typkonflikte (Modellierung als Attribut oder Entity-Typ) help: fällt auch „virtuelles (berechnetes) vs. echtes Attribut“ mit in diese Kategorie?
  - Beziehungskonflikte (1:n oder n:m)
  - Schlüsselkonflikte (unterschiedliche Primärschlüssel)
  - Verhaltenskonflikte (kaskadiertes Löschen oder „manuelles“ Löschen)
- Konflikte auf Instanzebene (Ausprägung, Daten)
  - Ambiguitätsprobleme
    - Mehrfachspeicherung derselben Entities – mit unterschiedlichem Schlüsselwert – in verschiedenen lokalen Datenbanken
    - Verwendung desselben Schlüsselwerts für verschiedene Entities (in verschiedenen lokalen Datenbanken)

      todo: Tabelle einfügen
- Funktionale Konflikte: Nicht alle (potentiell möglichen) Operationen gegen das globale Datenmodell lassen sich in semantisch äquivalente Operationen gegen die lokalen Datenmodelle umsetzten. Problem woher bekannt? → View-Update-Problematik

  Lösung? Reduzierter Funktionsumfang auf globaler Ebene, etwas nur lesender Zugriff auf Teile des Bestands [Beliebige Operationen wird man auf globaler Ebene ohnehin nicht zulassen wollen/können, etwa Data Definition Language (DDL)?]

3. Vereinheitlichungsphase: Konfliktauflösungs, d. h. „(Fehler-)Behandlung“ für die in der zweiten Phase erkannten Konflikte und Probleme, u. a.:
   - *Namenskonflikte*: Auflösbar durch geeignete *Umbenennung* (bzw. geeigenete Abbildungen, dess es erfolgt natürlich *kein* Eingriff in die vorhandenen *lokalen* Schemata/Datenbanken); etwa Namenserweiterungen bei Homonymen (Attributname)

- *Strukturkonflikte*: *Umformungen* etwa zwischen Entity-Typen und Attributen (Abbildungen!), etwa wenn ein (lokales) Attribut global als Entity-Typ gesehen werden soll etc. (oder in mehreren Tabellen gespeicherte Daten einer lokalen Datenbank global als „nested table“ (NF^2-artig) → Probleme à la View-Update-Problematik → nur lesender Zugriff)
- Ambiguitätsprobleme (auf Instanzebene): (Schlüsselproblematik (Was sind globale Schlüssel?) im Zusammenhang mit Integration)

  Lösungsvarianten: Einführung neuer, global einheitlicher Schlüssel oder globale Verwendung von erweiterten lokalen Schlüsseln

  a) Einführung neuer, global einheitlicher Schlüssel todo: Tabelle einfügen

     Bemerkungen hierzu:

     – Hier Schlüssel-Umsetzungsteil integriert in globale Tabelle (separate, „reine“ Schlüssel-Umsetzungstablle wäre auch denkbar)

     – Umsetzungsteil sollte „verborgen“ werden vor globalen Anwendungen (letzten beiden Spalten von GLOBLIEFTAB)

     – Problematisch (natürlich) bei Änderungen aller Art:

       i. ABC-Firma wird – durch globale Anwendung – umbenannt in ABZeh-Firma → Änderung entsprechend auf Knoten A und B

       ii. ABTse-Firma wird – durch globale Anwendung – neu eingefügt: Wohin damit? Auf Knoten A oder B oder beide?

       iii. Lokale Anwendung auf Knoten A benennt ABC-Firma um in ABChe-Firma: Fortan zwei separate Tupel – ABC-Firma und ABChe-Firma – auf globaler Ebene? ↯

       (kein „neues“ Problem (durch Integration hinzugekommen), sondern ohnehin durch redundante Datenspeicherung A/B gegeben)

  b) globale Verwendung von erweiterten lokalen Schlüsseln todo: Tabelle einfügen

     Bemerkungen hierzu:

     – *Ortstransparenz* aufgeweicht (Knoten-ID auf globaler Ebene sichtbar)

     – Eigentlich gar keine separate Umsetzungstabelle (wärend sie im vorherigen Fall zumindest in Frage käme) → nur (virtuelle) Schlüsselerweiterung (im vorherigen Fall muss Umsetzungstabelle physisch existieren)

     – Mehr Tupel als im vorherigen Fall → aber ja nur virtuell

     – Änderungsprobleme?

4. Restrukturierungs- und Zusammenfassungsphase
    - Endgültige Festlegung des *globalen* konzeptuellen Schemas → bestimmt Vorgaben für die lokalen Repräsentationsschemata
        - Was muss in den jeweiligen LRS enthalten sein? (Ausschnittsbildeung?)
        - Was muss durch die jeweiligen LRS an Vereinheitlicung geschehen?
    - Festlegung der *Abbildungen* („Implementierung“ derselben), durch SQL-Sprachausdrücke
    - Ziele hierbei:
        - Vollständigkeit (globales Schema enthält alle – gewünschten – Informationen aller Teilschemata)
        - Minimalität/Redundanzfreiheit
        - Vollständigkeit

Zusammenfassung hiervon (Thema Schemaintegration)

- *Aufwendiger*, mehrstufiger Prozess bei postintegriertem Ansatz todo: Verweis auf den Abschnitt einfügen
- Nur *teilweise* automatisierbar (etwas *Vergleichsphase* (2) durch Werkzeuge unterstützbar bzw. bei Vorhandensein eines globalen Data Dictionary (globaler Datenkatalog) im Unternehmen; dieser existiert u. U. in Teilen schon unabhängig davon, ob Integration angestrebt wird)
- Betrifft Schema- und Datenebene (Instanzen)
- *Integritätsbedingungen* der lokalen Systeme zu berücksichtigen („globale Integritätsbedingungen“ ableiten?) bzw. – schlimmer noch – Integritätsbedingungen, die in lokalen Anwendungen verborgen sind
- Nichtsdestotrotz: Beschrittener Weg!

4 Anfragebearbeitung in verteilten Datenbanksystemen

4.1 Allgemeines/Grundlagen

Bekanntes aus DBS 1/2 u. a.:

- Algebraausdrücke als formale Grundlage der internen Darstellung von (**SLQ!** (**SLQ!**)-)Sprachanweisungen sowie als Ausführungs-/Optimierungsgrundlage (vgl. insb. DBS 1 Kapitel 6.3)
- Reale Repräsentation hiervon in DBMS: Operatorbäume/Query-Graphen → Ausführungsplan
- Auf dieser Basis: Äquivalenztransformationen der Algebraausdrücke (berühmte 26 Regeln aus DS 1) → Query Rewrite (durch das DBMS/den Optimizer)
- Rule-base Optimization vs. cost-based Optimization
- heutige DBMS-Produkte arbeiten alle (?) mit cost-based Optimization
- Grundlage/Kostenmaß? Geschätzte (vorhergesagte) Ausführungskosten auf Basis von I/O-Kosten (Anzahl der I/O-Operationen) (plus CPU-Kosten)
- „Datenbasis“ des Optimizers: Datenbankkatalog = Schemadaten + Statistikdaten; beides ja „Daten über Daten“, also Metadaten
- Probleme der Optimierung:
  - Statistikdaten nie in allen Fällen detailiert genug bzw. aktuell genug für eine „perfekte“ Kostenvorhersage bzw. auch nicht umfassend genug (Beispiele!), auch weil natürlich # Statistikdaten $\ll$ # Daten sein soll! (und nicht umgekehrt ;-)
- „Vorhersagen sind schwierig, insbesondere wenn sie die Zukunft betreffen“ ...
  - (genaue) I/O-Kostenvorhersage (Pufferung, Caching, „Buffer-Hit-Ratio“) ... vor allem bei
    - Mehrbenutzerbetrieb (parallele Anfragen)
    - Multi-Query-Fall (serielle Anfragen)

  - (genaue) CPU-Kostenvorhersage (Pfadlängenberechnung durch Optimizer? ϟ) quasi unmöglich → Ersatzmaße, etwa Anzahl bestimmter Subsystemaufrufe im DBMS (berechneter Wert)

- Vorhersageaufwand (Kosten für die Optimierung) Lösungen:
  - Verwendung (Benutzer, *Administrator*!) verschiedener Optimization Levels (à la DB2)
  - keine vollständige Enumeration (kein komplettes Durchrechnen aller möglichen (äquivalenten) Ausführungspläne, sondern „intelligents Abschneiden“ mit bekannten Techniken der heuristischen Optimierung)

Resümee: ALLE genannten Probleme (klassischer, zentralisierter DBMS) treten NATÜRLICH auch im verteilten Fall auf und diese sogar VERSCHÄRFT/mit zusätzlichen Lösungsanforderungen!

→ Forderung nach Bestimmung DES „global optimalen“ Ausführungsplans wird relaxiert wegen zu hohen Brechnungsaufwands, zu erwartenden zu genauen Eingabedaten für die Berechnung (denn: Lösungsraum bei Optimierung in verteilten Datenbanksystemen ist nochmal wesentlich größer als bei zentralisierten (Beispiel?), Problem der Datengenauigkeit (Statistiken) ist verschärft (wegen eventueller Heterogenität der Knoten-DBS, Autonomie, ...), Übertragungskosten, ...)

Unterscheidung zwischen globaler Optimierung und lokaler Optimierung.

Vorgehen „im Großen“

- Analyse und Zerlegung einer globalen Anfrage (die also Daten an mehreren Knoten tangiert) in lokale Anfragen, Versuch dabei insbesondere, Zwischenergebnisse bzw. zu übertragende Tupelmengen möglichst klein zu halten (Heuristik! Hängt in wahrheit natürlich von Übertragungsraten und -geschwindigkeit ab, Verhältnis zwischen Übertragungsgeschwindigkeit einerseits und I/O- und CPU-Geschwindigkeit andererseits)
- Lokale Anfragen werden lokal („vor Ort“) optimiert (und natürlich ausgeführt)

  → Es gibt i. d. R. nicht DEN globalen Optimizer, der in seine Kalkulation alle, auch lokal vorliegene Informationen einzubeziehen versucht („Herkules-Aufgabe“)
- Erkennen überflüssiger Teilaufgaben im verteilten Fall noch wichtiger als im zentralen!! (warum?)

4.2 Transformation von globalen Anfragen in lokale

Situation: Relation R existiert nicht physisch „am Stück“, sondern nur virtuell: Teile der Relation (Partitionen) liegen physisch auf verschiedenen Knoten der teilten Datenbank

Anfrage wird nun gegen R gestellt (impliziert also eine globale Anfrage)

Möglichkeiten der Anfrageausführung (prinzipiell):

- mit Materialisierung von R
  - einfach zu implementieren
  - (meist) zu kostspielig (insbesondere wenn für Anfrageausführung gar nicht „ganz R“ gebraucht wird)
- ohne Materialisierung von R
  - erfordert Transformation der globalen Anfrage in lokale ausführbare Teilanfragen
  - help: Das kann ich bei mir nicht lesen

Voraussetzungen für Anfragetransformationen:

- Formales Regelwerk für Transformationen → Äquivalenztransformationen vorhanden
- Formale Beschreibung (algebraische Spezifikation) der Abbildungen zwsichen globaler Relation auf Partitionen

→ gestattet dann insgesamt algebraische Substitution

Beispiel: Relation TEILE sei partitioniert $\text{TEILE}_1 = \text{SL}_{0 \leq TeileNr < 300}$ TEILE
$\text{TEILE}_2 = \text{SL}_{300 \leq TeileNr < 500}$ TEILE
$\text{TEILE}_3 = \text{SL}_{500 \leq TeileNr < \infty}$ TEILE → TEILE := TEILE_1 UN TEILE_2 UN TEILE_3

Anfrage: Q1 := $\text{SL}_{Preis > 700}$ TEILE

Mögliche Herangehensweise:

- Einsetzen: Q1’ := $\text{SL}_{Preis > 700}$ (TEILE_1 UN TEILE_2 UN TEILE_3)
- Ausmultiplizieren (Regel aus der Menge der zulässigen Äquivalenztransformationen) Q1” := $\text{SL}_{Preis > 700}$ TEILE_1 UN $\text{SL}_{Preis > 700}$ TEILE_2 UN $\text{SL}_{Preis > 700}$ TEILE_3

In *Operatorbaumschreibweise* (und damit nahe an tatsächlicher interner Repräsentation im DBMS Query Processor): todo: Bild einfügen

Bemerkungen hierzu:

- Q1' und Q1" beziehen sich nicht mehr auf (virtuelle) Relation TEILE, sondern auf Basis-Relationen (Partitionen) $\text{TEILE}_{1,2,3}$ → physische Relationen („Relationsteile")
- Q1' und Q1" enthalten – verteilte Datenbank! – Freiheitgrade:
  - Wo (an welchem Knoten) werdei die Vereinigungsoperationen ausgeführt?
  - Wo (an welchem Knoten) die Selektionen?

Für den Ausführungsplan müssen derartige Festlegungen getroffen werden mit eventuell folgendem Ergebnis: todo: Bild einfügen

Ob es so gemacht wird, entscheidet der globale Optimizer (berücksichtigt, wie erwähnt, Übertragungskosten (vielleicht sind Verbindungen B-A, C-A, D-A unterschiedlich schnell?), Knotenleistungscharakteristika etc.)

Frage: Könnte es eventuell noch „ganz anders" gehen? Benötigt man wirklich die Knoten B,C,D alle? → nur, wenn alle drei wirklich potentiell zum Ergebnis beitragen können

OCC Optimistic Concurency Control

LOB Large Object

CLOB Character Large Object

BLOB Binary Large Object

GKS Globales konzeptuelles Schema

GPS Globales Partitionierungsschema

GAS Globales Allokationsschema

LKS Lokales konzeptuelles Schema

LIS Lokales internes Schema

UDF User Defined Function

SQL Structured Query Language

SFW Select-From-Where Klausel im SQL

ACID Atomarity, Consistency, Isolation, Durability DIE Eigenschaft von Transaktionen

IMS Information Managment SystemDatenbank von IBM

DL/1 Data Language/One Data Modelling Language (DML) für IMS

DML Data Modelling Language

LRS Lokale Repräsentationsschemata

LES Lokale externe Schemata

DDL Data Definition Language