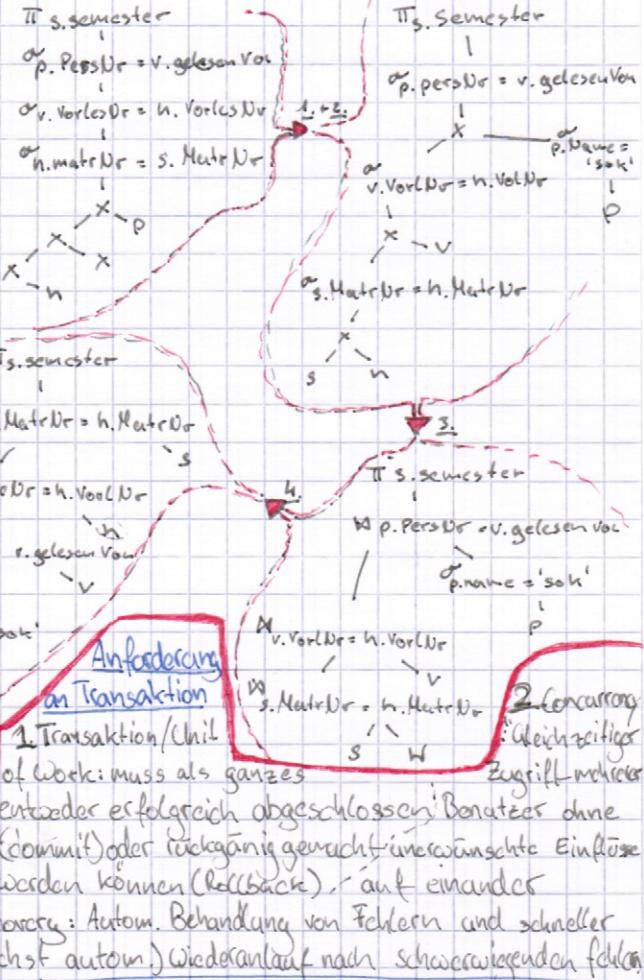


Transformation

Selektionsprädikate (σ) aufspalten
Selektionen "nach unten" schicken
Zusammenlegen von σ und x zu σx
Optimierung der N Reihenfolge
Kommutativität und Assoziativität
ausnutzen

Bsp.: SELECT DISTINCT s.Semester
FROM studenten s, Hören H, Vorlesung V,
Professoren P WHERE p.Name = 'Sok'
AND v.gelesenVon = p.PersNr
AND v.VorlesNr = h.VorlesNr
AND h.MatrNr = s.MatrNr;



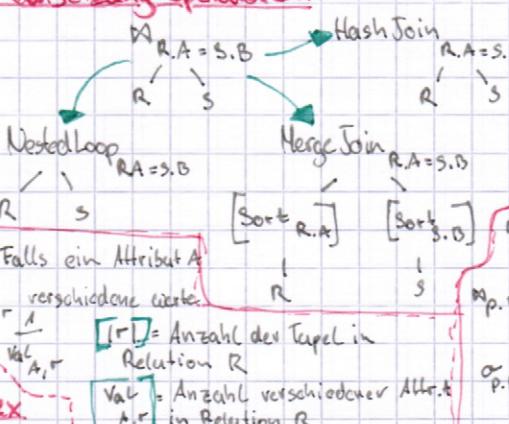
Selektivität

Selektivität einer Anfrage,
Schlüsselattribut einer
Relation R definiert = $\sigma_{A_1 \neq A_2}$
kennzeichnet wird, für das val
verschiedene Werte.
ist dies der Fall, ist Selektivität = $\frac{1}{|R|}$

Wann lohnt sich ein Index

Bsp.: Kosten Baumtiefe: lev
 $lev = (2^{C+1}) + 1$: $n < b$
 $lev < 1/lev = (c+1)$: r/b
 $< 1/(C+1) + 6 = \frac{1}{24} = 0.1\%$
Frage muss weniger als 4%
Sätze liefern damit sich Index lohnt Kardinalität (z.B. Geschlecht)

Achtung! Überindexierung kostet Zeit, Speicherplatz und kann Optimizer die Irc führen



Anforderungen an Transaktion

1. Freundschlüssel indexieren → besonders bei Master-Detail-Join
2. Attribute über die oft join läuft
3. Kein Index für Attrib. mit niedriger Kardinalität

ACID

Transaction / Recovery

Isolationsprobleme

Schedules

Folge von Read/Write op.

Atomicity: Zusammenhängende Operationen entweder ganz oder gar nicht.

Consistency: Alle Oper. hinterlassen DB in einem konsistenten Zustand

Isolation: Operationen von nebenläufigen Transaktions aufeinander ohne gegenseitige Beeinflussung

durability: Alle Resultate bleiben nach Beendigung der Transa. geprägt

Isolationsebenen:

at Uncommitted: • SCHWÄCHSTE STUFE!
Zugriff auf nicht ausgültig geschriebene Daten
für Read-Only Transaktionen verwenden
sonst Lost-Update möglich

at Committed: Nur lesen ausgültig
geschriebener Daten → Non-Repeat Read
Repeatable Read: Kein Non-Repeatable Read
oder Phantom Read möglich

serializable: Kein Phantom Read
→ garantisierte Serialisierbarkeit

Probleme von Sperren:

blocking: gesperrte Ressource zwingt diese zum Warten → weniger Transak.

Lösung: ohne Sperren + Transa. Verwaltung

Verrangern/Live-Lock: Eine Transa. kommt von weil immer andere vorher berücksichtigt werden

Lösung: RDBMS muss fair wählen

Dead-Lock: Lösung: RDBMS erkennt DL durchzyklische

optimal Transaktion auswählen + zurücksetzen

"optimal" später ausführen

Lost-Update:

Problem: Überschreiben bereits getätigter Updates. Lösung: Sparen für die Dauer der Transaktion (write-lock)

Dirty-Read: Problem: Lesen von Veränderungen noch nicht abgeschlossener Transaktionen Lösung: Nur Updates bestätiger Transaktionen lesen

Non-Repeatable-Read: Problem:

Lesen von zwischenzeitlich von anderen Transaktionen durchgeföhrten veränderten

Lösung: Der DB-Zustand schon der bei Beginn der Transa. vorliegt

Phantom-Read: Problem: DRR Lösung: Benutzer sieht Zwischenzustände von neu. y

Sperrenverwaltung: 1. Abfrage der Sperrre vor jedem DB-Zugriff.

2. Setzen einer Sperrre vom verfügbar Erweiterung jeder Transaktion.

Lese-Sperre (Shared Lock): read lock (rl)

read unlock (ru) • andere Transaktionen können Read und lsc-Sperren setzen

aber KEINE SCHREIBSPERRE setzen

Schreib-Sperre (Exclusive-Lock): write lock (wl)

corite unlock (wu) • DB steht nur dieser Trans. zur Verfügung (zum anderen)

Unlock: read unlock (ru) und write

unlock (wu) werden üblicherweise zu unlock (u) zusammengefasst.

Vollständiger Schedule = History: • sämtliche Schritte aller austestenden Transa. inkl. Terminieren (commit)

• Jede Transa. wird als erfolgreich oder fehlerhaft festgestellt. • Können mehrere ACID realeben Darstellung: r = read, w = write, x = DB, n = Transf. Terminierungsp. C_n = commit, a_n = abort

Socialisierbarer Konflikt: Konflikt wenn 2 Transa. auf dasselbe Obj. reihenfolgeabhängig zugreifen. 1. Schreib/Lese $\rightarrow w_1(x), r_2(x)$

2. Lese/Schreib $\rightarrow r_1(x), w_2(x)$

3. Schreib/Schreib $\rightarrow w_1(x), w_2(x)$

Lösung: serialisierbarkeit mit Abhängigkeitsgraph prüfen. KEINE ZYKLEN

Transaktionsmanager: • Erzeugt serialisierbare Aufgabenplan für parallel ausführenden Operationen

• Verwaltet notwendige Synchronisationsinformationen

Regeln zur Sperredisziplin: abhängig von Transactionstyp anfangszeit

• Schreibzugriff $w(x)$ nur nach Setzen einer Schreibsperrre

$w(x)$ möglich • Lesezugriff $r(x)$ nur nach Lesesperrre

$r(x)$ möglich • $w(x)$ möglich • $w(x)$ kann nur gesetzt werden wenn keine anderen Transa. sperren haben

• Nach $w(x)$ darf die Transa. kein erneutes $w(x)$ oder $w(x)$ setzen. • Eine Transa.

darf eine Sperrre der selben Art auf dem selben Obj. nicht nochmals anfordern

• Bei COMMIT/ROLLBACK müssen alle Sperren aufgehoben werden

Daten und Schlüssel:

Dünne Besetzter Index

Wenn die interne Relation nach den Zugriffswerten sortiert ist, nicht im Index ein Eintrag pro Sektor verweist mit K_1, K_2 auf den Anfang der Seite, K_2, K_3 auf die nächste usw.

↪ Datensätze mit Zugriffswert K , $K \leq K_x < K_y$ sind auf Seite K_y zu finden

Dick Besetzter Index

→ jeden Datensatz (d.h. Zugriffswert, wert)

interne Relation ist ein Eintrag im Index vorhanden. Sek. Indexe sind IMMER dicht besetzt im Index KANN dicht besetzt sein (Dateiorga-

nutz: Heap-Datei) • Gevise Abfragen können

mit Index bearbeitet werden

Schlüsselzugriff: Zuordnung von Prim./

sek. schlüsselwerten zu Addr. in einer

Hilfsstruktur (z.B. B-Bäume). Attr. werte

in Addr. sind in einer Datei gespeichert.

Werttransformation: Tupeladdr.

d auf Basis einer Formel aus Prim. OR

sek. schlüsselwerten berechnet (z.B. Hashing)

wird nur die Berechnungsvorschrift

spezifiziert (Überlaufbehandlung).

datenbasierte Indexstrukturen

asortiert:

Heap

Sortiert:

Index

rechte Pfeile

geformt

Statische Verfahren

Indexsequentielle Datei:

mb. von seq. Hauptdatei und

Indexsequ. Dateiform kann geclusteter,

dann besetzter Index sein

ind. zweistufiger "Baum"

Blattebene = Hauptdatei

Alle anderen Stufen = Indexe

Aufbau der Indexdatei

(Prim.schlüsselwert, Seitennummer)

Jeder Seite der Hauptdatei hat genau

einen Index-Datensatz in der

Indexdatei

Problem: - Bei einstufigen Index

fürst "Wurzel" mehrere (verkettete)

an, die seq. durchsucht werden müssen

Con: Indexseq. Datei / Indexdatei-

Schnelleres Suchen nicht seq. Datei

Mehr Speicherplatzbedarf (Indexdaten)

Mehr Zeitbedarf Einfügen

und Löschen

Primär-/Sekundär Index/Schlüssel:

Primär index: - Zugriffspfad der die Dateiorganisation form nutzen kann

- i.d.R über Primärschlüssel-Merk.:!

Sekundärindex: - Jeder Zugriffspfad ohne Nutzung der Datei orga.-Form

- Kann mehrere

Primärschlüssel: - wichtiger Kandidat für Pri. / Sek. Index - "Echter Schlüssel" Problem

↪ identifizierend, keine Duplikate - Merk.:!

Sekundärschlüssel: - Kandidat für Sek. Index

- Nicht vorrangig mit Fremdschlüssel

- Nicht zwingend ein Schlüssel - kann mehrere

Geduselter Index:

↪ Sortiert wie interne Rel.

Bsp.: 'Kunden' nach 'Kundennummer'

sortiert → Index über Attr. KDnr

geclustert (geignet für

Bereichsanfragen) • Jeder dünn-

besetzte Index ist geclustert aber nicht vice versa!

Nicht Geduselter Index: anders sortiert als interne Rel. Bsp.: über 'Name' sek.index Datei,

die selbst nach 'KDnr' sortiert ist • Sek. Index

können DURCH dicht-besetzt und NICHT-Geduselt sein

Statische vs. Dynamische Verfahren:

Statische Verfahren:

↪ Prim. Index: indexseg. Datei

↪ Sek. Index: indexiert - nicht seg. Datei

Sequentielle Datei:

• Daten werden sortiert gespeichert

↪ nach einem Attributwert oder Kombi.

Operationen:

• Einfügen - Seite suchen Datensatz einsetzen

↪ beim Anlegen/seq. Füllen Datei jede Seite nur bis z.B. 1/2 füllen • Löschen - suchen, dann Löschbit setzen

• Suchen - seq. durchsuchen Löschbit setzen

der Datei (keine Bin. Suche möglich trotzdem schnell)

Bei stark wachsenden Dateien:

↪ Wachstum der linear verketten Indexdateien

↪ Keine auto. Anpassung vorgesehen

• Bei stark schrumpfender Dateien:

↪ nur sogen. Verringerung der Index und Hauptdateiseiten

• Viele Mutationen -> unangemessenen Seiten

der Hauptdatei -> unnötig hoher Speicherplatz

↪ Längere Zugriffszeit

Klassifikation von Zugriffs-Grundoperationen:

Grundoperationen

Datenzugriff

Tupelzugriff

über TID

über Schlüsselwert

Mengenzugriff

Relationales Scan

Index Scan

Sortierung

Nested Loops

Block Nested Loops

Verbund

Hash Verbund

Classic Hash Join

Simple Hash Join

Partitioned Verfahren

Stored Procedure Vs

• Expliziter Aufruf

↪ Anwender / Programm

Transactionskontrolle

Kapselung von "Business Rules"

Reduktion von Netzwerktraffic

↪ Batch Processing

• Erhöhte Sicherheit benötigt

• Fehlerbehandlung kompliziert

• Dicht besetzter Index

• Dünne besetzter Index

• Geduselter Index

• Dicht besetzter Index

• Geduselter Index

• Statische Zugriffsstr.

• Optimal NUR bei

bestimmt (feste) Anzahl von zu verwendenden Datensätzen

• Periodische Reorganisation nötig

• Heap • Sequentielle Datei

• Indexsequentielle Dateien • Indexierte nicht-sequentielle Dateien • Statisches Hashing

• Dynamische Zugriffsstruktur

• Unabhängig von der Anzahl Datensätze optimal

• Dynamische Indexverfahren verändern dynamisch Anzahl der Indexstufen → in RDBMS üblich (B-Baum)

• Dynamische Addr. transformationsverfahren verändern dynamisch den Bildbereich der Transformation (dynamisches Hashing)

• Heap: • Daten unsortiert gespeichert

• Oft grundlegende Speichertechnik in RDBMS

• z.B. zeitlicher Reihenfolge der Speicherung

• Operationen / Komplexität:

• Einfügen - O(n) wenn Duplikate erlaubt, sonst

• Löschen - O(n) suchen dann Löschbit setzen

• Suchen - O(n) Durchsuchen der Gesamtdatei; mak. Aufwand (Heap-Datei mit Sek. Index eingesetzt oder sehr kleine Relat.)

• Indexiert - nicht sequentielle Datei:

• Unterstützt sek. Schlüssel • Mehrere Zugriffspfade dieser Form pro (Haupt) Datei möglich

• Ein-/Mehrstufig - höhere Indexstufen sind index-sequentiell organisiert

Aufbau der Indexdatei:

• Sek. Index: dicht besetzt, nicht geduselter (oder

• Jeder Satz der Hauptdatei hat 1 Indeksatz (w, s)

w: Schlüsselwert, s: zugeordnete Seite

• Weil Sek. Schlüsselwert mehrfach vorkommen kann

w für ein 'w' mehrere Sätze in Hauptdatei aufrufen

w für 'w' 1 Liste von Addr. in Hauptdatei angeben

Optimierung: Index-Scan: nutzt Index zum Auslesen des Tupel (External) Sortieren = Merge-Sort
in Sortierreihenfolge (Ausnahme: Hash-Index)

Relationen-Scan (full-Table scan): Bsp.: `SELECT * FROM kunde WHERE Nachname BETWEEN 'Heuer' AND 'Panik'`

Währt alle Tupel einer Relation in beliebiger Reihenfolge ab.

Bsp.: `SELECT * FROM kunde WHERE Nachname BETWEEN 'Heuer' AND 'Panik'`

currScanID < open-rel-scan (KUNDE-Relation ID), currTID < nextTID (currScanID)

while end-of-scan (currScanID) do

currRec < fetch-tuple (KUNDE-Relation ID, currTID)

if currRec.Nachname \geq 'Heuer' AND currRec.Nachname \leq 'Panik' then put (currRec)

end

currTID < nextTID (currScanID)

while end-of-scan (currScanID) do

currRec < fetch-tuple (KUNDE-Relation ID, currTID)

if currRec.Nachname \geq 'Heuer' AND currRec.Nachname \leq 'Panik' then put (currRec)

end

Entschachtelung von Anfragen:

• Erkennen von gemeinsamen Teilanfragen • Jede Anfrage müssen Alle Seiten in kanonische n-Relationen eingesetzt werden

• Anfrage umformen \rightarrow Verbund zwischen Blockung, ist aber sortiert

• Verbundsprädikaten (geschachtelte n-Relationen mit n-1)

• Reduzierung durch Ausnutzung

Typ-A-Schachtelung (Aggregation):

Inner Block Q enthält kein Verbundsprädikat, dass die äußere Relation referenziert. Q berechnet Aggregat

Bsp.: `SELECT BestNr FROM Bestellung WHERE ProdNr = (SELECT MAX(ProdNr) FROM Produkt WHERE Preis < 15)`

Ausführung: 1. Innere Anfrage + Aggregat führt zu Ergebnis in äußere und berechnen

Typ-N-Schachtelung (Nested):

wie Typ-A. 2. Q berechnet kein Aggregat

Bsp.: Ref. Bsp TypA but instead " = " wie "IN"

Ausführung: Vor A: Innere Anfrage berechnen (Tabelle) und in Äußere einsetzen. Vor B: Entschachtelung Anfrage entschachtelt: `SELECT BestNr FROM Bestellung B JOIN Produkt P ON B.ProdNr = ?ProdNr AND P.Preis < 15`

Typ-N-Anfragen mit 'IN' Prädikaten der Tiefe n-1 können semantische n-Rel.-Anfragen transformiert werden.

Typ-J-Schachtelung (Join):

1. Inner Block Q referenziert im Verbundsprädikat die Relation des äußeren Blocks ('korrelierte Unterabfrage')

2. Q liefert keinen Aggregationswert

Bsp.: `SELECT BestNr FROM Bestellung B WHERE B.ProdNr IN (SELECT ProdNr FROM Lieferung L WHERE L.LiefNr = ?LiefNr AND L.Datum = current_date)`

Ausführung: Entschachteln Anfrage entschachtelt: `SELECT BestNr FROM Bestellung B JOIN Lieferung L WHERE B.ProdNr = L.ProdNr AND L.Datum = current_date`

Zum Bestellung B JOIN Lieferung L WHERE B.ProdNr = L.ProdNr AND L.Datum = current_date

Regel 8: Blattknoten so vertauschen, das kleinste Zwischenergebnis zuerst ausgewertet wird

Regel 3, 4, 7, 10: Π -Operationen so weit wie möglich nach unten möglich

Forme x-Operation, die von or gefolgt wird, wenn möglich um

Anwendung der Alg. Regeln

a) Regel 1: σ Konjunktionen oder in Kaskaden von σ zerlegt

b) Regel 6, 4, 6, 9: σ so weit nach unten wie möglich

c) Regel 8: Blattknoten so vertauschen, das kleinste Zwischenergebnis zuerst ausgewertet wird

d) Regel 3, 4, 7, 10: Π -Operationen so weit wie möglich nach unten möglich

e) Regel 8: Blattknoten so vertauschen, das kleinste Zwischenergebnis zuerst ausgewertet wird

f) Regel 3, 4, 7, 10: Π -Operationen so weit wie möglich nach unten möglich

Verbundoperationen (Join): Hash-Join, Nested-Loop-Join, Block ULJ, Sort-Merge-Join

NLJ: 1. Sätze der ersten Tabelle der Reihe nach mit allen Sätzen der zweiten Tabelle verglichen. 2. Satz... Aut innere Tabelle wird über Index zugriffen. Geblcktes Lesen der Sätze beider Tabellen

SMJ: 1. Beide Relationen werden sortiert 2. SMJ wie Algo.

Hash-Join: 1. Beidell-Phase \rightarrow Tupel der kleineren Relation lesen und durch Hashfunktion entsprechende Tupel in Buckets ordnen 2. Probe-Phase \rightarrow Tupel der grösseren Relation lesen und unter Tupel Anwendung von 'Hash'-Bucket mit potentiellen Verbundpartnern identifizieren 2. Verbundbedingungen pro Bucket prüfen und Resultat anzeigen

Optimierung - Grundprinzipien:

- Möglichst kleine Zwischenergebnisse (damit Platz im Hauptspeicher) \Rightarrow Selektion so früh wie möglich ausführen
- Basisoperationen (Selektion / Projektion) wenn möglich zusammenfassen und ohne Zwischenspeicherung von Zwischenergebnissen als Berechnungsschritt realisieren
- Redundante Operationen oder leere Zwischenrelationen aus Berechnung entfernen
- Zusammenfassung gleicher Teilausdrücke
- Eigenschaften der relationalen Algebra ausnutzen

Rel. Alg.:

- $P_1 \wedge (P_2 \wedge P_3) \Leftrightarrow (P_1 \wedge P_2) \wedge P_3$ (same for 'v')
- \wedge, \vee, \wedge sind kommutativ und assoziativ $\Rightarrow R_1 \wedge R_2 = R_2 \wedge R_1$
- $\sigma_P^Q(R) = \sigma_Q^P(\sigma_P^Q(R)) \Rightarrow$ Selektion ist austauschbar $R_1 \wedge (R_2 \wedge R_3) = (R_1 \wedge R_2) \wedge R_3$
- $\sigma_{P_1}^Q \wedge \sigma_{P_2}^Q \wedge \dots \wedge \sigma_{P_n}^Q(R) = \sigma_{P_1}^Q(\sigma_{P_2}^Q(\dots(\sigma_{P_n}^Q(R)))) \Rightarrow$ Aufbrechen von konjugierten Selektionen (σ) $\Rightarrow \Pi_L^U(R_1 \wedge R_2) = \Pi_L^U(R_1) \vee \Pi_L^U(R_2)$
- $\neg(P_1 \wedge P_2) = \neg P_1 \wedge \neg P_2$
- $\neg(P_1 \wedge P_2) = \neg P_1 \vee \neg P_2$ \Rightarrow Umwandlung von Disjunktion zu Konjunktion

Typ-JA-Schachtelung (Join-Agg.):

1. wie Typ-J 2. Q liefert Aggr. Wert

Bsp.: Ref. Bsp Typ-J but instead 'SELECT ProdNr' use 'SELECT MAX(LiefNr)' Ablauf: 1. In Typ-J-Schachtelung umwandeln (Gruppierung über L.ProdNr) \rightarrow Aggr. verschwindet. 2. Typ-J Schachtelung Anfrage entschachtelt: `SELECT B.BestNr FROM Bestellung B JOIN (SELECT MAX(LiefNr) AS MAX_LiefNr, ProdNr FROM Lieferung WHERE versandart = 'exp' GROUP BY ProdNr) L ON B.BestNr = L.MAX_LiefNr AND B.ProdNr = P.ProdNr`

Logische Optimierung Regeln:

- Aufbrechen von Konjunktionen im Selektionsprädikat $\Rightarrow \sigma_{C_1 \wedge C_2 \dots}(R) = \sigma_{C_1}(R) \wedge \sigma_{C_2 \dots}(R)$
- 'or' ist kommutativ $\Rightarrow \sigma_{C_1}(R) = \sigma_{C_2}(R) \wedge C_1, C_2$
- Π -Kaskaden falls $L_1 \leq L_2 \leq \dots \leq L_n$ $\Rightarrow \Pi_{L_1}^U(\Pi_{L_2}^U(\dots(\Pi_{L_n}^U(R)))) = \Pi_{L_1}^U(R)$
- Vertauschen von σ und Π falls Selektion sich nur auf Attribute der Projektionsliste bezieht $\Pi_{A_1 \dots A_n}(\sigma_{C}(R)) = \sigma_{C}(\Pi_{A_1 \dots A_n}(R))$
- X, V, \wedge, \vee sind kommutativ
- Vertauschen von σ mit Π
- Falls Selektionsprädikat C nur auf Attribute der Relation R zugreift, können beide Operatoren vertauscht werden

$\sigma_{C}(R \wedge S) = \sigma_C(R) \wedge S$

b) Falls Selektionsprädikat C eine Konjunktion der Form " $C_1 \wedge C_2$ " ist und C_1 sich nur auf R und C_2 sich nur auf S bezieht gilt: $\sigma_C(R \wedge S) = \sigma_{C_1}(R) \wedge \sigma_{C_2}(S)$

3. Projektionsliste $L = \{A_1, \dots, B_{i-m}\}$ \rightarrow A \rightarrow Attribute aus R, B \rightarrow Attribute aus S

Falls Joinprädikat nur Attribute aus L bezieht $\Rightarrow \Pi_L^U(R \wedge S) = (\Pi_{A_1 \dots A_n}^U(R)) \wedge (\Pi_{B_{i-m}}^U(S))$

Falls Bezug ausserhalb von L, müssen diese für die Join-Operation erhalten bleiben $\Rightarrow \Pi_L^U(R \wedge S) = \Pi_L^U(A_1 \dots A_n, \text{außer } B_{i-m}, \text{ außer } B_{i-m}, \text{ außer } B_{i-m})$

Operationsfolgen zusammenfassen (z.B. Zusammenfassung (σ))

Fehlerklassifikation: (durch Recovery verursacht)

- Transaktionsfehler:** • Abbruch der jeweiligen lokalen Transaktion. • kein Einfluss aufs restliche System (Fehler) • technisch: Anwendungsprogramm (z.B. divide by 0)
- ausw. abbruch durch Benutzer (z.B. Time-out, invalidade)
- ausw. abbruch durch System (z.B. Deadlock, ACID, zwifflung; "lokales" Zurücksetzen aller Änderungen in abgebrochenen Transa. (Atomicity needs "undo")
- Systemfehler:** • Folge: Zerstörung der Daten im Hauptspeicher (flüchtige DB) • NICHT TROFFEN: Hintergrundspeicher (permanente DB)
- technisch: • RDBMS-Fehler (config file Fehlerhaft) • Betriebssystemfehler (Seitenfehler, ungültiger Befehl) • hardware Fehler (Stromausfall, Fehler im Memory) • zwifflung: Nachvollziehen der Änderungen die von geschlossenen Transac. nicht in die DB gebracht wurden (REDO) • Zurücksetzen in Änderungen von nicht abgeschlossenen ausw. (undo)

- Mediafehler:** • Verlust der Daten in stabilen DB
- technisch: • "Head-Crashes", Controller-Fehler, Naturgewalten (Feuer, Erdbeben)
- operator-Fehler
- zwifflung: Voraussetzung: Sicherungskopien (B-Backup, Log-Backup) auf sep. Medium inspielen vom Backup • Anwendung der Transactionlogs, die seit dem Log vom Backup erzeugt worden sind DB muss durch UNDO und REDO in einen konsistenten Zustand gebracht werden!
- B-Backup: 1x Tag/Woche
- Log-Backup: 1x 30 Minuten!

- | Ziel | Wiedergabe | Zeitbezug | Entscheidung | Zustand | Update | Quellen |
|------------------------------------|--|-----------|--------------|--|--------|---------|
| • Abwicklung der Geschäftsprozesse | | | | | | |
| | • Detaillierte, granulare Geschäftsvorfall Daten | | | | | |
| | • Aktuell • Zeitpunktbezogen | | | | | |
| | • Auf die Transac. ausgerichtet | | | | | |
| | | | | • Altbestände oft nicht modelliert (funktionsorientiert) | | |
| | | | | • Häufig redundant | | |
| | | | | • In konsistenter - | | |
| | | | | • Lautend und Konkurrenzend | | |
| | | | | • Strukturiert • Meist statisch im Programmcode | | |

Sicherungspunkte

Problem bei Recovery: Alter der ältesten Seite im Puffer nicht bekannt. Gesamtes Log-Buch (seit letzten DB-Backup) muss eingespielt werden → **Hohes Zeitaufwand!**

Lösung: Sicherungspunkte (SP) (Checkpoints). Aufbau vom SP:

1. Geänderte Seiten (dirty Pages) vom Puffer (flüchtige DB) in die Persistente DB geschrieben.

2. Puffer vom Log auf stabilen Speicher.

3. SP im Log registriert

Arten von SP:

1. Transaction-konsistente SP

2. Aktionskonsist. SP (einzelner SQL-Befehl)

3. Unscharfe SP

Transaktionskonsistente SP:

- Alle Änderungen werden in einem Moment, in dem keine Transa. aktiv sind, vom Puffer in die DB geschrieben

Ablauf:

1. SP anmelden

2. Nur ankommende Transa. müssen warten.

3. Aktive Transa. zuende führen.

4. Geänderte Seiten auf Platte schreiben.

5. SP im Log-Buch registrieren.

- ⊕ Späteres Recovery keine Veränderungen vor diesem Punkt berücksichtigen

- ⊕ Löst Benutzer unter Umständen lange warten (unscharfe SP)

Dispositive Daten (OLAP)

- Information für Mgt.
- Entscheidungsunterstützung

- Verdichtete, transformierte Daten
- umfassendes Metadatangebot

- Unterschiedliche, aufgaben-abhängige Aktualität
- Historien betrachtung

- Sachgebiete o. themenbezogen standardisiert + Benutzerlangfristig

- Konsistent modelliert • kontinuierliche Redundanz

- Ergänzend, Fortschreibung abgeglichen, aggregierter Daten

- AdHoc für komplexe, ständig wechselnde Fragen und Standardauswertungen

Non-Standard Datenbanken:

Data-Warehouse:

Operative Daten: lieferiert von Admin - Plan - Abschlussystemen

- Große Teile von OLTP-Systmen erzeugt Online Transaction Processing - Tage geschäft
- Mehrere Benutzer greifen das selbe System und DB zu (Auskunft, Buchen, etc.)

- Dispositive Daten:** • Unterstützen das Management (OLAP-Systeme), Direkter durchgriff von Mgmt-Unterstützenden Systemen nicht zielführend. • Werden aus operativen Daten erzeugt.

OLAP (Online Analytical Processing)

↳ Entscheidungsunterstützend

Begriffsvielfalt: OLAP sind Software-systeme, die für menschliche Entscheidungs-treträger relevante Informationen für operative und strategische Aufgaben ermittelt, aufbereitet, übersichtlich zusammenstellt und bei der Auswertung "hilft" (wikipedia)

System Einführung Fokus

Decision Support System (DSS) Anfang 1970 Modellfokus

Executive (EIS) Information Ende 1980 Präsentations-fokus

DataWarehouse Anfang 1990 Datenfokus (DW)

OLAP Anfang 1990 Modellfokus

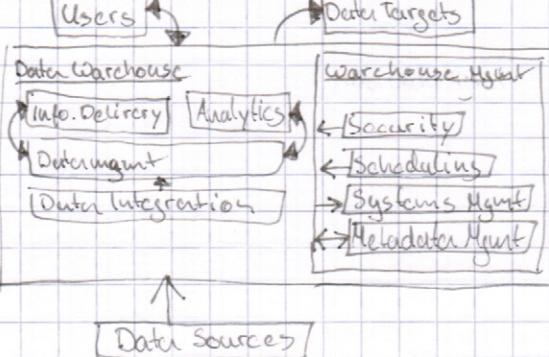
BI Anfang 1990 Präz.-Fokus

Advanced Analytics Ab ca. 2012 Analysefokus

Data Warehouse: Theorieorientierte, zeitorientierte, integrierte und unverständliche Datensammlung, deren Daten sich für Mgt.-entscheidungen auswerten lassen "Innen" 1994

Merkmale: • strikte Trennung von operativen und dispositiven Daten • Integration vielfältiger (auch externer) Datenquellen • Langfristige Datenaufbewahrung, Historisierung • Mehrdimensionale Auswertung • Detailliert und verdichtete Daten • Holz-Datenstrukturen (keine Löschung) • immer unterschneidet/ anwender spezifisch aufgebaut.

DWH Referenzarchitektur



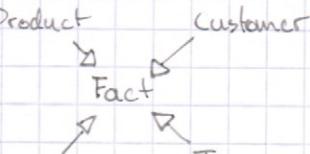
Daten-Mart

Langfristig gehaltener Datenbestand innerhalb eines DW/Hopie vom Teil und Dimensionen (Raum, Zeit, Organisation...) • Geschaffen für bestimmte Anwendung
Organisationsbereich • Viele Architekturvarianten, Übergänge oft nicht scharf. • "Analysedatenbanken" können auch Excel-File sein
• Hierarchie von oben nach unten traversieren. "Verfeinerung" entlang einer Dimension
• Hierarchie von unten nach oben traversieren. "Aggregation" nach einer Dimension

Roll-up

Hierarchie von oben nach unten traversieren. "Verfeinerung" nach einer Dimension

Star-schema



Facttable: Enthält Fakten und Fremdschlüssel auf Dimensionstabellen

Dimensionstabelle: Enthält Dimension und Primärschlüssel, die mit Facttable "verknüpft" ist

No-SQL (Not only):
• nicht relational • schemafrei • einfache API • Auf horizontale Skalierung ausgerichtet • einfache Datenmodellierung • Konsistenzmodelle
• BASE und Eventual Consistency

NoSQL

Key/Value Stores • Wide Column Stores • Document Stores

Graphdatenbank

oft NoSQL:

Obj. DB • Grid/Cloud DBAnw.

Multidimensionalität

- Unterscheidung von Fakten (gemessene Werte)
- Unterschied von Dimensionen (Raum, Zeit, Organisation...)
- Hierarchische Struktur von Daten
- Fakten (Measures):**
 - Eigentliche Obj. der Analyse
 - Ansammlung von Einzelmesswerten / Kennzahlen, verknüpft mit Kontext (Dimension)
 - Jedes Faktum stellt einen Businesswert, eine Transaktion oder ein Ereignis dar.
 - I.d.R. numerische Werte (Verkaufszahlen, Lagerbestand, Umsatz, Gewinn etc.)
 - Analyse der Fakten soll es dem Nutz. ermöglichen die Unternehmensleistung zu überwachen/steuern
 - Numerische Werte lassen sich auf verschiedene Arten verdichten / aggregieren.

Dimensionen

- Kontext der Daten
- Eindeutige Strukturierung des Datenraums
- Hoffentlich orthogonal (unabhängig) • Verfügen über Attrib., welche geordnet werden können
- Anzahl nicht beschränkt • 'Zeit' ist häufig
- "Non-Standard DB-Systeme"** (alle die nicht Standard)
 - Quasi Echtzeit • Riesiges Datenvolumen (Petabytes)
 - Lange Transa. (z.B. Konstruktionsbereich)
 - Abbildung obj.-orientierter Konstrukte (z.B. Vererbung, Polymorphismus) und Speicherung von Objekten

No-SQL: CAP

Consistency: Alle Knoten sehen zu jeder Zeit die selben Daten. ACID hat nur innere Konsistenz

Availability: Ausgefallene Knoten beeinflussen die Verfügbarkeit anderer nicht

Partition Tolerance: System arbeitet trotz Verlust von Nachrichten, einzelner Knoten / Partitionen vom Netz weiter

Anwendung: Klassische vert. Systeme ist 'C' am wichtigsten → C.U.A eingeschränkt
• Web-Anwendungen können Fokus auf A + P, dabei Einschränkung von 'C'

Bsp.: DVS → AP, RDBMS → CA

Geldautomat → CP

Eventual Consistency (BASE):

- Konsistenz erst nach definiertem Zeitraum erreicht, nicht nach jeder Transa.

Big Data (5 - V)

Volume: Tera- bis Petabyte

Velocity: Hohe Stream/Verarbeitungsgeschwindigkeit • Viele Datensätze pro Tag/Stunde/Minute

Variety: Unterschiedliche Datenquellen (Text, Bilder, Videos, Blogs etc.)

Veracity: Schwankende Datenqualität

Value: Wert des Daten

Herausforderung: Erfassung, Speicherung, Auswertung und Visualisierung von Daten

"Standard - DB - Systeme" Anforderungen

- DBanwendungen kommen aus dem Betriebswirtschaftlichen Umfeld.
- Transaktionssicherheit • Hohe Performance bei Search/Update • Verwaltung 'einfacher', stark strukturierter Daten
- i.d.R. zentraler DB-Server
- Relationale DB-Systeme

↳ Grenzen: • Fokus auf strukturierte Daten • Begrenzte Skalierbarkeit • Schweränderungen sind komplex

Skalierung

Vertikal: Leistungssteigerung durch hinzufügen von Ressourcen (z.B. + CPU, + RAM, + HDD, etc.) • Unabhängig von der Implementierung möglich. Begrenzt durch Stand der Technik und Kosten → Transparent für DBMS

• Admin. aufwand ist konstant.

• Hardwarekosten • Skalierung nur in grossen Stufen möglich

↳ hohe Kosten für ungenutzte Leistung

Horizontal: Hinzufügen von Rechnern/Knoten • System muss für Parallelisierbarkeit (Anwendung) ausgerichtet sein → Kostengünstig

• Skalierung in kleinen Stufen

• Last/Datenverteilung notwendig

• ggf. verteilte Protokolle

• Höhere Fehlerrate • höherer Administrationsaufwand

DAB 2 Zusammenfassung:

Trigger:
 - Code der vom DBMS ausgeführt wird
 - Aufführung auf Grund von sich ändernden Datenbankzuständen
 ent-condition • geeignet für Action

- Realisierung von Integritätsbedingungen
- Berechnungen (z.B. nachführen von Summen)
- Protokollieren von Datenänderungen
- + Entlastung der Anwendungsprogramme (Fehlerbehandlung)
- + effiziente Ausführung möglich (Testen, Debuggen)
- + einfacher Rollback (Ergebnis event. abhängig von Aufruf-Reihenfolge) (Terminierung von geschachtelten Triggern)

SQL: CREATE / DROP / ALTER Trigger...
 ↳ Auslösung: BEFORE / AFTER / INSTEAD OF

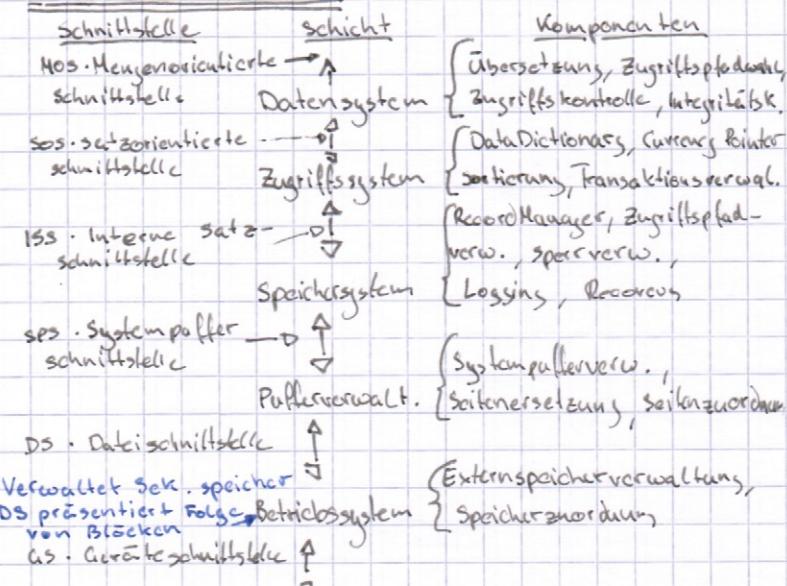
INSTEAD OF:
 + Verhindern unerlaubte DML-Operationen
 + SQL-constraints nicht ausreichend
 + Nicht überbare Sichten "änderbar" machen

AFTER:
 + weitere operationen auslösen
 + Historisierung

SQL Example:

```
CREATE TRIGGER NewPODetail
ON Purchasing.PurchaseOrderDetail
AFTER INSERT
AS
IF @@ROWCOUNT = 1
BEGIN
    UPDATE Purchasing.PurchaseOrderHeader
    SET Subtotal = Subtotal + LineTotal
    FROM inserted
    WHERE PurchaseOrderHeader.PurchaseOrderID =
        inserted.PurchaseOrderID
END;
```

5-Schichten-Architektur:



Speicherhierarchie

Tertiärespeicher im Zusammenhang mit Datenbanken:

- Speichern der aktuellen Datenbankinhalte für laufenden Betrieb
- (Anst sekundärer Speicher)
- Datensicherung
- Archivierung
- Eigenschaften Ebene X

Externer Speicher

1. Primärspeicher + sehr schnell
 ↳ sehr teuer, Volatil (Inhalt nur zur Systemlaufzeit verfügbar)
2. Sekundärspeicher (HDD, SSD)
 - Faktor 10⁹-10¹⁶ langsam (SSD 10³)
 - Günstig, Persistent
3. Tertiärespeicher
 - Langsam
 - Billig, Hohe Kapazität, nicht permanent zugreifbar, Persistent

Speicher- und Zugriffszeit: $x < x+1$

Kosten pro Speicherplatz: $x > x+1$

Kapazität: $x < x+1$

Lebensdauer: $x < x+1$

Problem: Zugriffslücke zwischen Primärspeicher und Sekundärspeicher ca. 10⁵

Caching kann helfen wenn 1) Zeitliche Lokalität (in kurzer Zeit auf gleiche Daten zugreifen) und Räumliche Lokalität (Data Cluster: Zusammen angefragte Daten sind auf dem Sekundärspeicher eng zusammen)