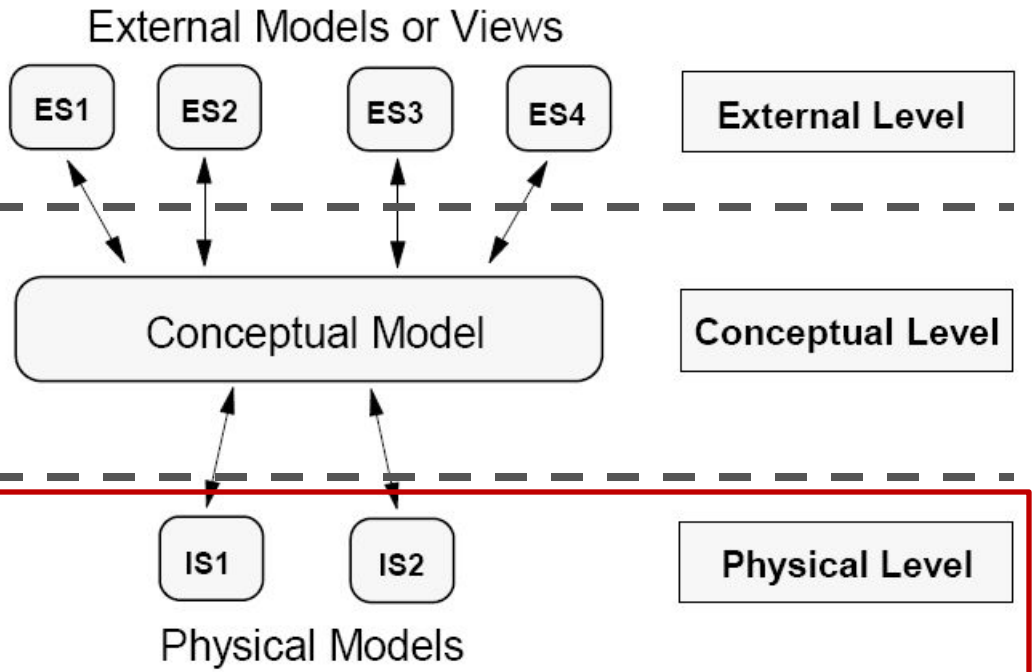
11 Physical Database Optimization

Strukturierungsebenen innerhalb einer Datenbank

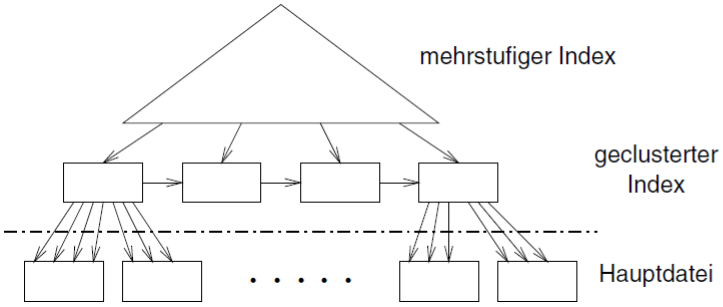
- Externes Schema/Nutzersicht
  - Verschiedene Nutzer haben unterschiedlichen Blick auf Daten
  - Implementierung: Views
- Konzeptionelles Schema
  - Beschreibt gesamte in Datenbank gespeicherte Daten und ihre Beziehungen
- Internes Schema
  - Wie werden Daten physisch repräsentiert?
  - Verschiedene Daten werden unterschiedlich abgelegt

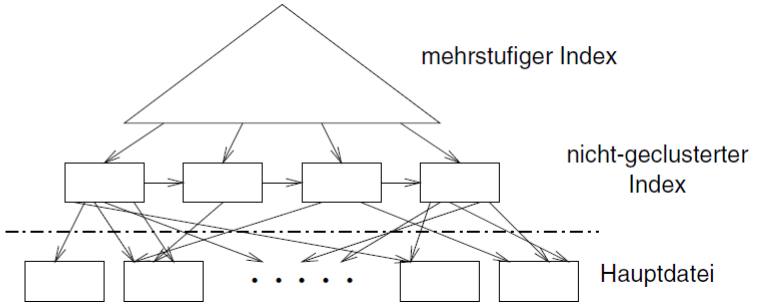


- beeinflusst wesentliche die Performanz des DBS
- Optimierung ist Aufgabe des DBA

Indizes

- Datenstruktur zur Beschleunigung bestimmter Operationen
- (Primär-)Index: bestimmt Dateiorganisationsform
  - Normalfall: Primärschlüssel über Primärindex/geclusterter Index
- Sekundärindex: Redundante Zugriffsmöglichkeit, zusätzlicher Zugriffspfad
- Vielfältige Implementierungen
  - Unsortierte Speicherung von Tupeln: Heap-Organisation
  - Sortierte Speicherung von internen Tupeln: sequentielle Organisation
  - Gestreute Speicherung von internen Tupeln: Hash-Organisation
  - Speicherung in mehrdimensionalen Räumen: mehrdimensionale Dateiorganisationsformen
- Beschleunigen Punkte-Anfrage, Bereichs-Anfrage und Sortierung
- Erhöhter Aufwand durch Wartung bei UDI-Operationen





Erstellung eines Index

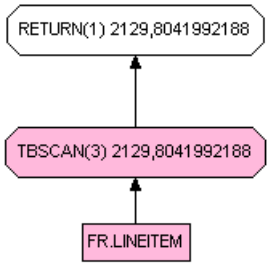
- **`CREATE [UNIQUE] INDEX <idx-name>`**
  `ON <tab-name> (<column-name> [ASC|DESC] [, …]*)`
  `[INCLUDE (<column-name> [, …]*)]`
  `[CLUSTER]`
  `[{DISALLOW|ALLOW} REVERSE SCANS]`
  `[COLLECT [[SAMPLED] DETAILED] STATISTICS]`
- **`UNIQUE`**
  - Sicherstellung der Eindeutigkeit (keine Duplikate, max. 1 Nullwert)
- **`ASC|DESC`**
  - auf-/absteigende Sortierung der Daten im Index → für sortiertes Lesen und Bereichsabfragen wichtig
- **`CLUSTER`**
  - Tupel mit ähnlichen Ausprägungen physisch nahe abspeichern (Primärindex)
  - keine Fragmentierung auf Blockebene
  - nur ein Primärindex pro Relation möglich

Entfernen eines Index

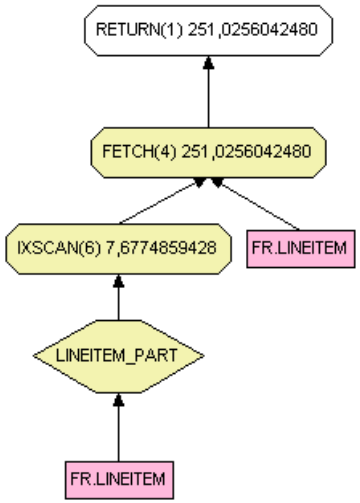
- **`DROP INDEX <idx-name>`**

```
SELECT * FROM LINEITEM WHERE L_PARTKEY = 1552
CREATE INDEX LINEITEM_PART ON LINEITEM(L_PARTKEY)
```

ohne Index



mit Index auf **L_PARTKEY**



Partitionierung

- Aufteilung schematisch gleicher Daten in unterschiedliche physische Ablagen
  - Vollständige und Disjunkte Aufteilung
  - Primärablage
- Idealerweise müssen Anfragen nur einen Partition lesen -> geringer Leseaufwand
- Vertikale Partitionierung: Trennung von Spalten, unterstützt Projektion
- Horizontal Partitionierung: Trennung von Zeilen, unterstützt Selektion
- Anfragen wider der Aufteilung können langsamer werden

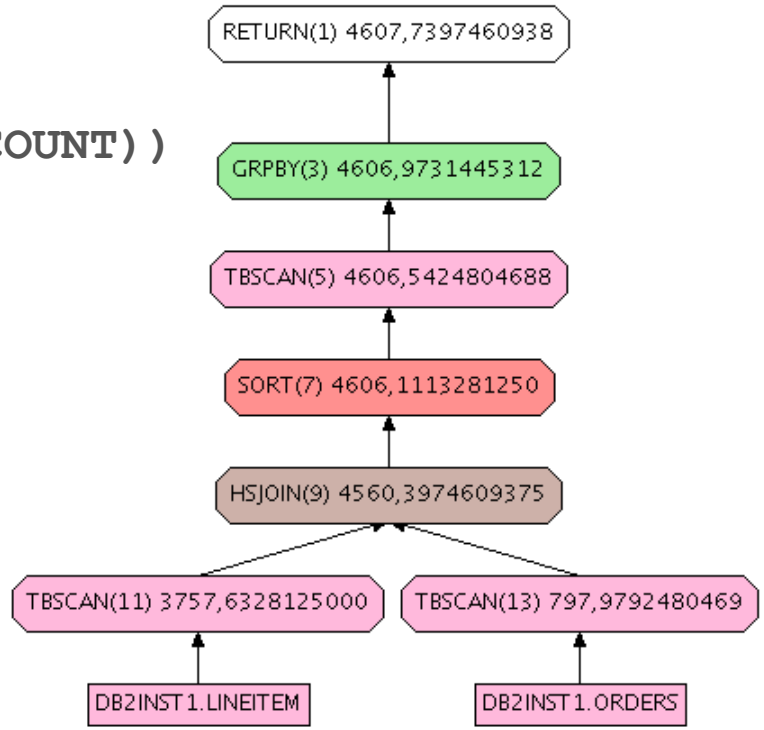
Materialisierte Sichten

- Vorberechnete Anfrageergebnisse
  - Unvollständig, möglicherweise überlappend
  - Sekundärablage
- Können Beantwortung der Vorberechnete Anfrage und ableitbarer Anfrage verwendet werden
- Anfrageergebnisse oft deutlich kleiner als Ausgangsdaten -> geringeren Leseaufwand
- Vor allem bei Selektion und Gruppierung, aber auch Projektion
- Erhöhter Aufwand durch Wartung bei UDI-Operationen

Beispiel (ohne materialisierte Sicht)

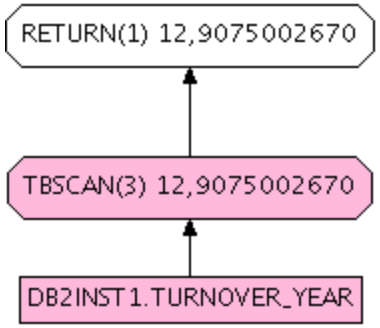
- Umsatz nach Jahren

- ```
  SELECT
    YEAR(O_ORDERDATE),
    SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
  FROM LINEITEM, ORDERS
  WHERE L_ORDERKEY = O_ORDERKEY
  GROUP BY YEAR(O_ORDERDATE)
  ```


  ```

Beispiel (mit materialisierter Sicht)

- ```
  CREATE TABLE TURNOVER_YEAR AS (
    SELECT
      YEAR(O_ORDERDATE) AS YEAR,
      COUNT(*) AS CNT,
      SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS TURNOVER
    FROM LINEITEM, ORDERS
    WHERE L_ORDERKEY = O_ORDERKEY
    GROUP BY YEAR(O_ORDERDATE))
  DATA INITIALLY DEFERRED REFRESH IMMEDIATE
  ENABLE QUERY OPTIMIZATION
  MAINTAINED BY SYSTEM
  ```
- ```
  REFRESH TABLE TURNOVER_YEAR
  ```
- Anfrage verwendet jetzt automatisch die materialisierte Sicht → effizienter



Physische Strukturen

- Beschleunigen bestimmte Operationen
- Helfen aber nicht alle Operationen
- Behindern bestimmte andere Operationen
- Benötigen Platz

Gute Physische Strukturen

- Bringen viel Nutzen
- Behindern wenig
- Belegen wenig Platz

Welche Operation profitieren hängt ab von

- Anfrageoptimierer
  - Entscheidet, welche Strukturen er für die Anfrageausführung nutzt
  - Entscheidet, in Abhängigkeit seines Wissens über die angefragten Daten und die vorhanden Strukturen (Statistiken)
- Vorhanden Strukturen -> Konfiguration
  - Konfiguration bestimmt Auswahlmenge des Optimierers
  - Nutzen von Strukturen abhängig von anderen Strukturen (Strukturinteraktion)

Während der Entwicklungszeit

- Entwurf im eigentlichen Sinne
- Üblicherweise noch ein laufendes System zur Hand
- Nur Anwendung einfacher Regeln
  - Indizes auf jedem Primär-, Alternativ- und Fremdschlüssel
  - Indizes auf Attributen auf den die Anwendung Suchmasken vorsieht
  - …

Während der Laufzeit/Wartung

- Optimierung im eigentlichen Sinne
  (aber auch physischer Entwurf/physical design genannt)
- System läuft und in einem konkreten Zustand
  - Daten
  - Konfiguration
- Performanz von Anfrage ist bekannt
- Performanz soll gesteigert werden
- Beseitigung von Engpässen
- Aufgabe des Datenbankadministrators (DBA)

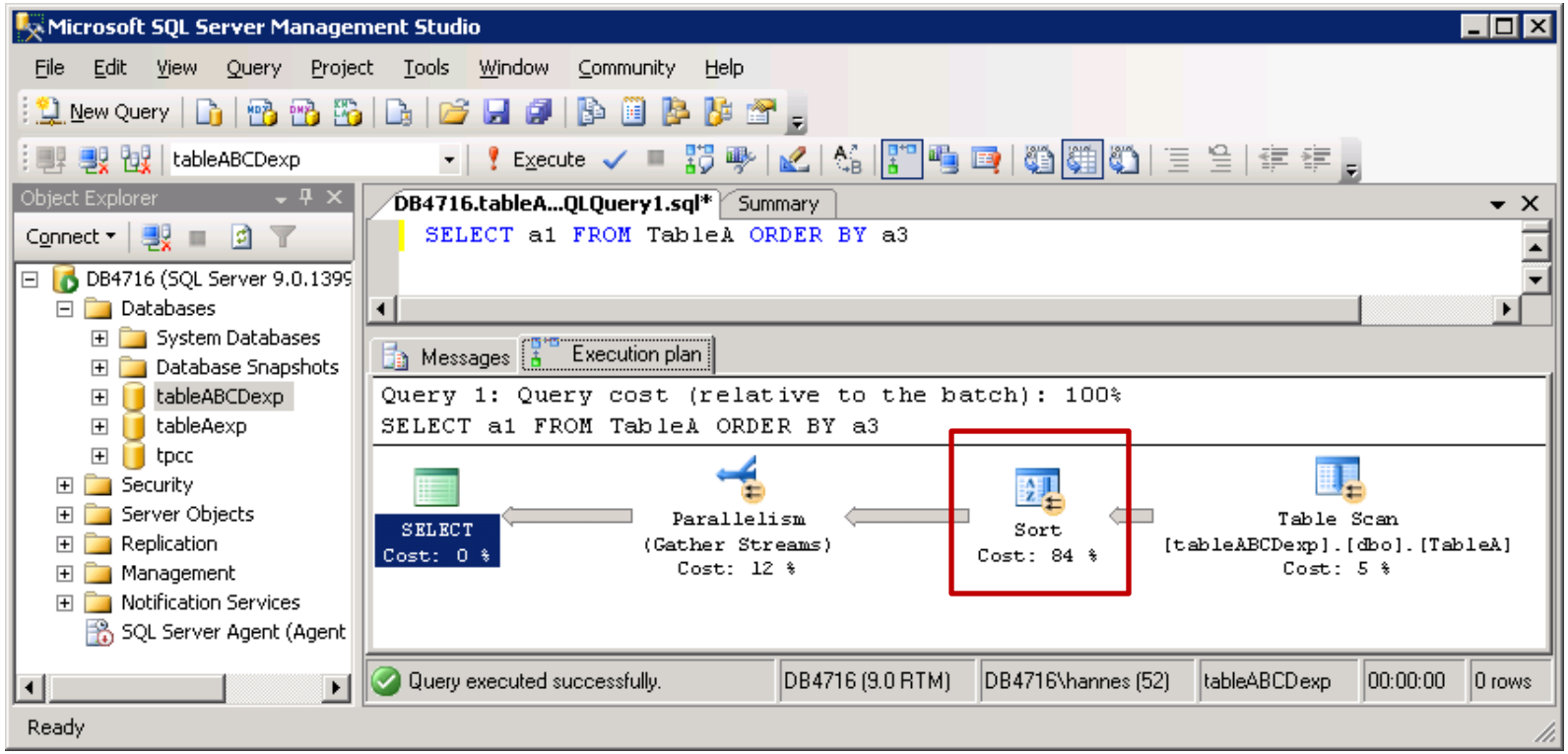
Monitoring und Benchmarking

- Durch Monitor-Werkzeuge zeichnet der DBA Statement und ihre Performanz auf
- Beispiel DB2

```
CREATE EVENT MONITOR EVENTS
FOR STATEMENTS WRITE TO FILE '/home/db2inst50/events/'
…
# db2evmon –db TPCH –evm EVENTS
38) Statement Event ...
  Appl Handle: 38
  Appl Id: *LOCAL.db2inst50.050620125518
  Appl Seq number: 0072
  …
  Text         : SELECT COUNT(*) FROM LINEITEM, ORDERS, …
  …
  Start Time   : 2008-12-04 14.29.36.372988
  Stop Time    : 2008-12-04 14.29.36.462530
  Exec Time    : 0.089542 seconds
  …
  User CPU     : 0.000005 seconds
  System CPU   : 0.000001 seconds
  Fetch Count : 32
```

Analyse und Optimierung

- Fokus auf schlecht laufende Statements
- Anfrageplan zeigt wo Hauptlast in der Bearbeitung eines Statements steckt
- Teiloperationen durch geeignete physische Strukturen beschleunigen



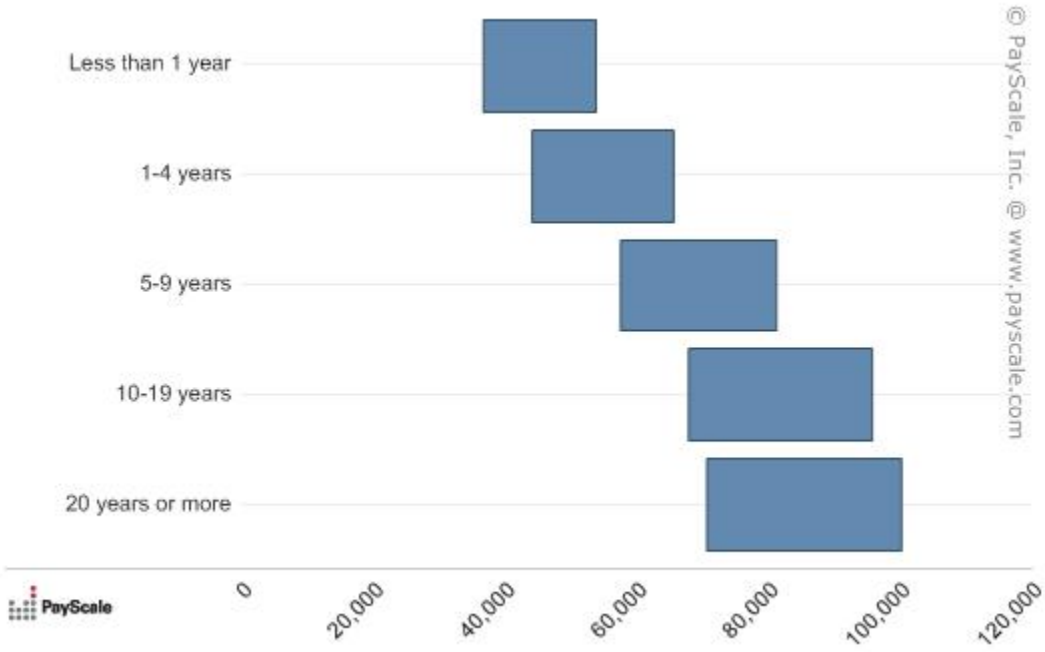
Probleme

- Zeitaufwändig
  - Monitoring, Analyse, Optimierung
- Abwägungen schwierig
  - Performanzgewinn vs. Speicherplatzbedarf
  - Performanzgewinn vs. Wartungsaufwand
- Hochqualifiziertes Personal erforderlich

Folgen

- Kosten
- Suboptimale Performanz

Wie kann DBA bei der Optimierung weiter unterstützt werden?



Werkzeuge des Physischen Entwurfs

- Unterstützung des DBA bei der Optimierung des physischen Entwurfs
- Über Monitoring, Profiling und Explain hinaus

- Empfehlung eines physischen Entwurfs (Empfehlung von physischen Strukturen)
- Automatische Optimierung des physischen Entwurf

Klassifikation

		Verwendung des Werkzeuges	
		Offline	Online
Art des empfohlenen Entwurfs	Statisch	Design Advisor	Alerter
	Dynamisch	Dynamic Design Advisor	Online-Tuning

Statischer Offline Entwurf

Definition 1 (Physische Entwurfskonfiguration)

- Eine physische Entwurfskonfiguration ist eine Menge C physischer Entwurfsstrukturen $\{I_1, I_2, \ldots, I_p\}$ und $C \subseteq P$, wobei P die Menge aller m möglichen Entwurfsstrukturen ist.

Problem 1 (Statischer Offline Entwurf)

- Geben ist eine Datenbank, eine Statement-Menge $W = \{S_1, S_2, \ldots, S_n\}$ und eine Speicherplatzbegrenzung b.
- Gesucht wird eine physische Entwurfskonfiguration C, so dass $SIZE(C) \leq b$ und die Statement-Ausführungskosten EXEC(W,C) minimal werden.

Definition 2 (Statische Ausführungskosten)

- Die Ausführungskosten EXEC(W,C) einer Statement-Menge $W = \{S_1, S_2, \ldots, S_n\}$ unter Verwendung einer physischen Entwurfskonfiguration C ist die Summe der Ausführungskosten $EXEC(S_i,C)$ jedes Statements S_i aus W.

Statement generality

- Entwurfswerkzeug muss mit jedem Statement umgehen können, jedes Statement berücksichtigen können, welches auch vom Datenbanksystem als gültiges Statement akzeptiert wird.

Access path generality

- Entwurfswerkzeug muss alle mögliche Zugriffspfade von sich aus berücksichtigen. Es brauchen keine Pfade vorgegeben werden.

Model correspondence

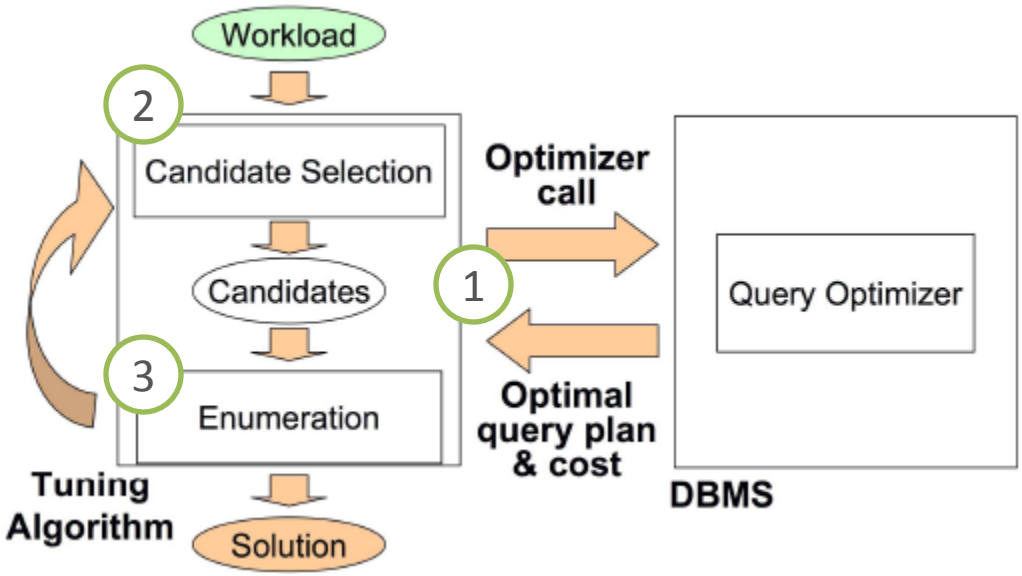
- Das Entscheidungsmodel des Entwurfswerkzeug muss mit dem Entscheidungsmodel des Anfrageoptimierers übereinstimmen.

[Schkolnick, M. & Tiberio, P. Considerations in Developing a Design Tool for a Relational DBMS. *COMPSAC*, **1979**, 228-235]

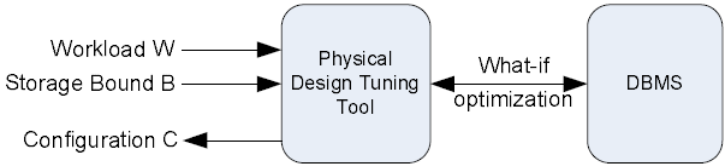
1. Bewertung von Konfiguration

2. Auswahl von Kandidaten-strukturen

3. Enumeration von Konfigurationen



- Spezielle Schnittstelle zum Datenbanksystem
- Ermöglicht model correspondence



Kernfunktionalität

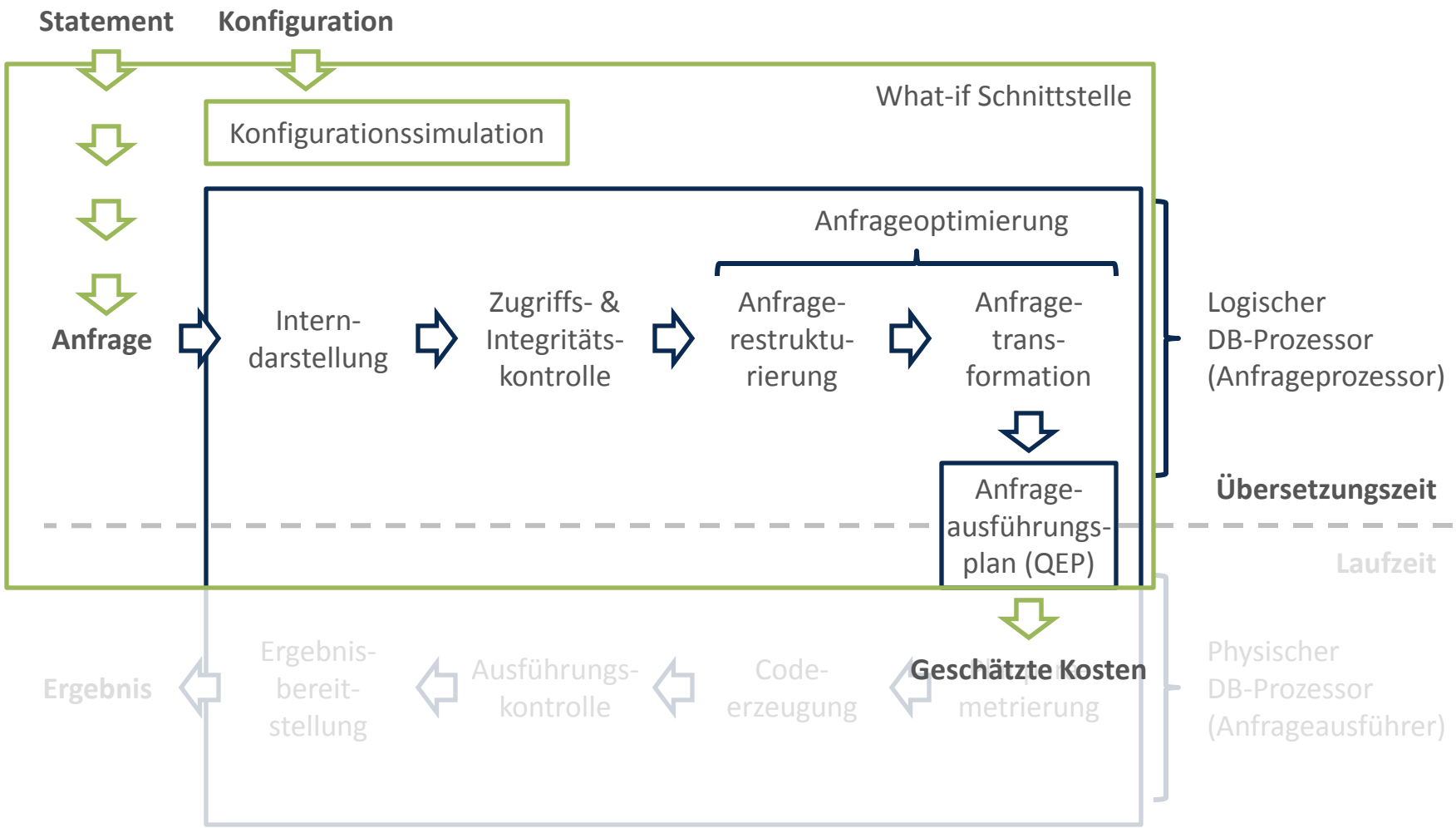
- Bestimmung von Ausführungskosten für Statements unter Verwendung einer bestimmten Konfiguration (EXEC(W,C))
- Statements können dabei nicht ausgeführt werden (UDI-Operationen), daher nur Rückgriff auf die vom Optimierer geschätzten Kosten!

Realisierung

- Simulation einer Entwurfskonfiguration gegenüber dem Optimierer
- Entwurfsstrukturen
  - Werden nicht physisch angelegt
  - Nur im Datenbankkatalog eingetragen
  - Inklusive ihrer Statistiken
- Für logische Anfrageausführung (DB2: Explain, SQL-Server: Showplan) nur die Kataloginformation ausreichend

[Finkelstein, S. J.; Schkolnick, M. & Tiberio, P. Physical Database Design for Relational Databases. *ACM TODS*, **1988**, 13, 91-128]
[Chaudhuri, S. & Narasayya, V. R. AutoAdmin 'What-if' Index Analysis Utility. *SIGMOD'98*, ACM Press, **1998**, 367-378]

What-if Schnittstelle (2)



Problem: What-if-Aufrufe sind aufwändig

- Bei jedem What-if-Aufruf für eine Konfiguration muss das Statement komplett neu optimiert werden

Ansätze:

- Ableiten der Bewertung von Teil-Bewertung der Konfiguration (Bewertungsrecycling)
- Optimierer-Instrumentalisierung

Achtung: Kein Komplexitätsreduktion, nur weniger What-if Aufrufe

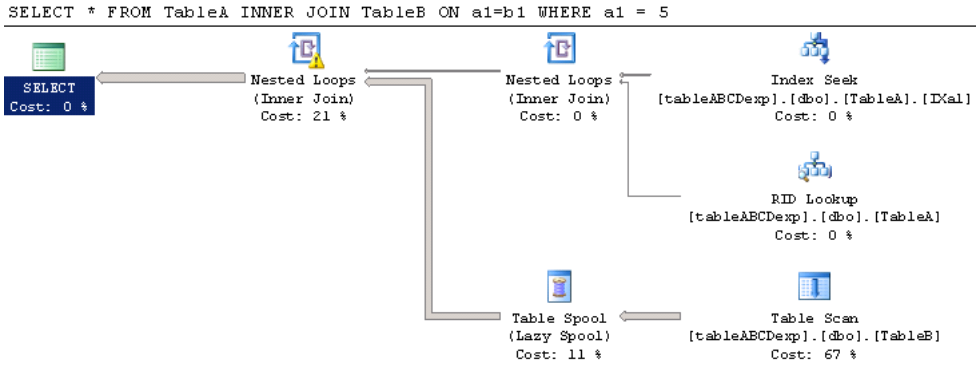
Ableiten von enthaltenen Entwurfsstrukturen

- Nützlichkeit einer Struktur I für ein Statement S:
  BENEFIT(S,I) = EXEC(S,{}) - EXEC(S,{I})
- Nützlichkeit einer Konfiguration C für ein Statement S:
  BENEFIT(S,C) = $MIN_{I \subseteq C}$ BENEFIT(S,I)
- Voraussetzung: Trennbarkeitseigenschaft (separability property):
  Es existiert kein $I_b \neq I_a$, so dass EXEC(W,{I_a}) ≠ EXEC(W,{I_a, I_b})

[Whang, K.-Y. et al. Separability - An Approach to Physical Database Design. *IEEE Transactions on Computers,* **1984**, *33*, 209-222]

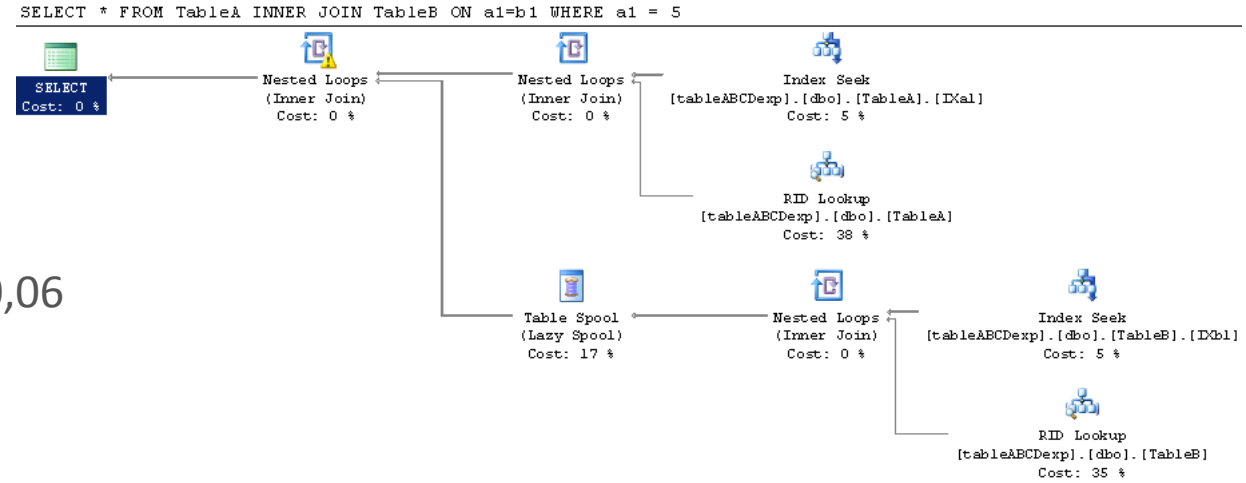
- Join-Methoden in heutigen DBS erfüllen Trennbarkeitseigenschaft nicht
  - Entwurfsstrukturen interagieren
  - Folge: Abgeleitete Bewertung sind deutlich schlechter als direkt ermittelte Bewertung
- Beispiel SQL-Server
  - Anfrage mit Index auf einem Verbundattribute

    geschätzte Kosten: 4,6



  - Anfrage mit Index auf beiden Verbundattributen

    geschätzte Kosten: 0,06

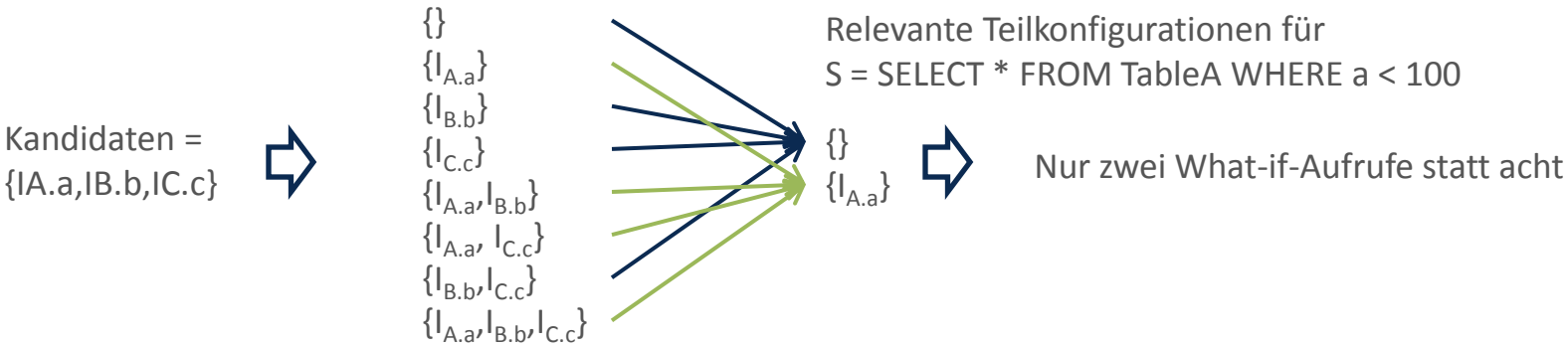


Ableiten von Teilkonfiguration

- Relevante Teilkonfiguration – auf Statement bezogene Teilkonfigurationsbildung
- Atomare Konfigurationen – auf Optimierer bezogenen Teilkonfigurationsbildung

Ableiten von Teilkonfiguration – Relevante Teilkonfiguration

- Nur bei Verarbeitung verwendbare Strukturen sind für ein Statement relevant
- Nicht relevante Strukturen kann man bei Bewertung weglassen
- Verschiedene Konfigurationen werden so auf die gleiche relevante Teilkonfiguration reduziert
- Relevante Teilkonfiguration muss nur einmal bewerte werden



[Chaudhuri, S. & Narasayya, V. R. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB'97, ACM,* **1997**, 146-155]

Ableiten von Teilkonfiguration – Atomare Konfigurationen

- Annahme
  - Optimierer wählt nur eine Teilmenge der vorhandenen, relevanten Strukturen
  - Optimierer wählt immer die beste Teilmenge
- Mögliche Teilmengen → atomare Konfigurationen
- Nur atomare Konfigurationen müssen bewertet werden
- Bewertung anderer Konfigurationen ist davon ableitbar

  $BENEFIT(S,C) = EXEC(S,\{\}) - MIN_{AC \subseteq C}\ EXEC(S,AC)$
- Beispiel:
  - Tabellen: A(a1,a2), B(b1,b2)
  - Nur einspaltige Indizes
  - Annahme: Optimierer verwendet maximale einen Index pro Tabelle
  - Mögliche Konfigurationen: 2^n (n Attribute)

    $|P(\{I_{a1},I_{a2},I_{b1},I_{b2}\})| = 2^4 = 16$
  - Atomare Konfigurationen: $\Pi_{Tabellen}(n_i+1)$ (n_i Attribute in Tabelle i)

    $|\{\{I_{a1},I_{b1}\}, \{I_{a1},I_{b2}\}, \{I_{a2},I_{b1}\}, \{I_{a2},I_{b2}\}, \{I_{a1}\}, \{I_{a1}\}, \{I_{b1}\}, \{I_{b2}\}, \{\}\}| = (2+1)(2+1) = 9$
  - Bei 5 Tabellen à 5 Attributen: 33.554.432 vs. 7.776

[Finkelstein, S. J.; Schkolnick, M. & Tiberio, P. Physical Database Design for Relational Databases. *ACM TODS*, **1988**, 13, 91-128]

Ableiten von Teilkonfiguration – Atomare Konfigurationen – Auswahl

- Zu bewertende Konfiguration C enthält große Menge A atomare Konfigurationen
- Welche Teilmenge B $\subseteq$ A die C vollständig abgedeckt ($\cup_{AC \in B}$=C) ist für die Bewertung von C ausreichend?
- Heuristik Größenbeschränkung:
  Optimierer verwendet …
  - … maximal j Strukturen pro Tabelle
    (Idee: Mit jeder weiteren Struktur steigt der Leseaufwand und sinkt der Gewinn)
  - … Strukturen auf maximal t Tabellen
    (Idee: Strukturen auf den ersten paar Joins bringen den größten Gewinn)
- Heuristik Strukturinteraktion:
  - Sukzessiver Aufbau von Teilkonfigurationen
  - Stopp-Kriterium ist zu geringer Unterschied zwischen abgeleiteten und direkt ermittelter Bewertung

[Chaudhuri, S. & Narasayya, V. R. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB'97, ACM,* **1997**, 146-155]

In designing our cost evaluation module, we found that values of **j** = 2 and **t** = 2 provide good quality solutions while dramatically reducing the number of atomic configurations for complex workloads. [...]

Consider a SELECT query with conditions $T_1.A < 20$, $T_1.A = T_2.B$, $T_3.C$ BETWEEN [30,50], $T_3.C = T_2.B$. In this case, one 3-table atomic configuration is ($T_1.A$, $T_2.B$, $T_3.C$) since all three indexes may be used together to answer the query. However, due to the single-join atomic configuration based pruning step, the above atomic configuration is not evaluated. Rather, the cost of this query for the 3-table configuration is estimated by taking minimum of the costs of the atomic configurations: ($T_1.A$, $T_2.B$), ($T_1.A$, $T_3.C$), and ($T_2.B$, $T_3.C$).

1. n = 2; A = {*atomic* configurations of size <= 2}.
2. A'={}; Evaluate all configurations in A.
3. For each configuration C in A, determine if the indexes in C interact strongly. We do this by testing to see if the *evaluated* cost of the configuration is at least **x**% less than its *derived* cost.
4. If C meets the above condition, add all atomic configurations of size n+1 that are supersets of C to A'.
5. If A' = {}, then exit.
   Else A = A'; n = n + 1, Goto Step 2.

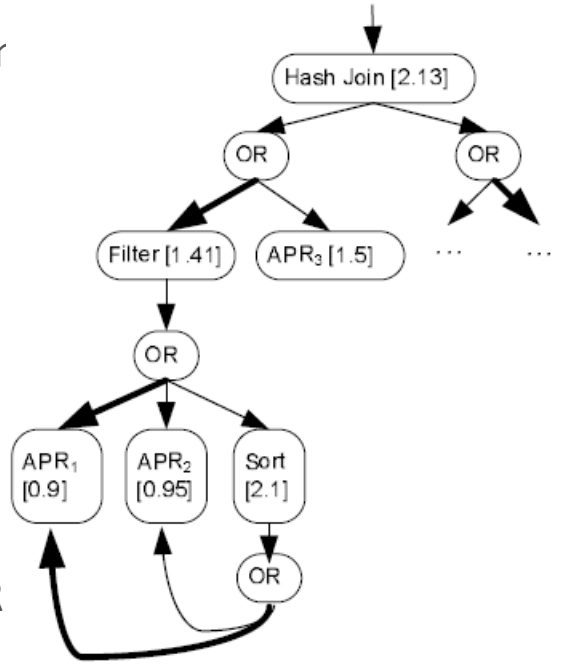
Aufwändige Teil des Optimierer-Aufrufes ist die Join-Enumeration

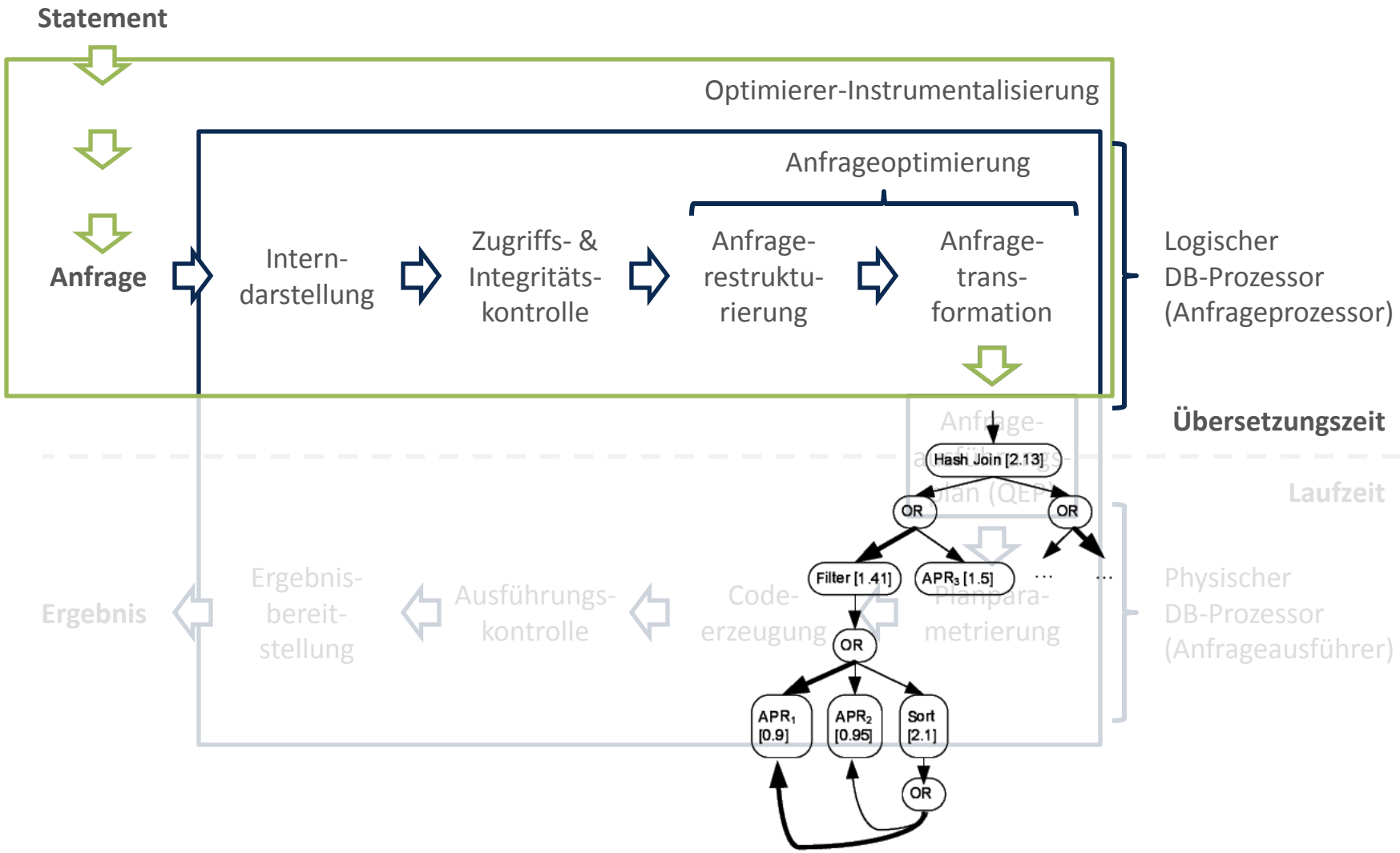
- Idee: Join-Enumeration trennen von Enumeration der Einzeltabellen-Zugriffspfade
- Einzeltabellen-Zugriffe lassen sich hinter abstrakten Planoperator verstecken
- Access plan request: APR(S,O,A,N)
  - S – Menge an Attribute in Prädikaten für die ein Index verwendet werden kann
  - O – Sequenz von Attribute nach denen sortiert sein soll
  - A – Menge von zusätzlich benötigter Attribute
  - N – Anzahl der Ausführung des Teilplans (> 1 bei NL-Joir

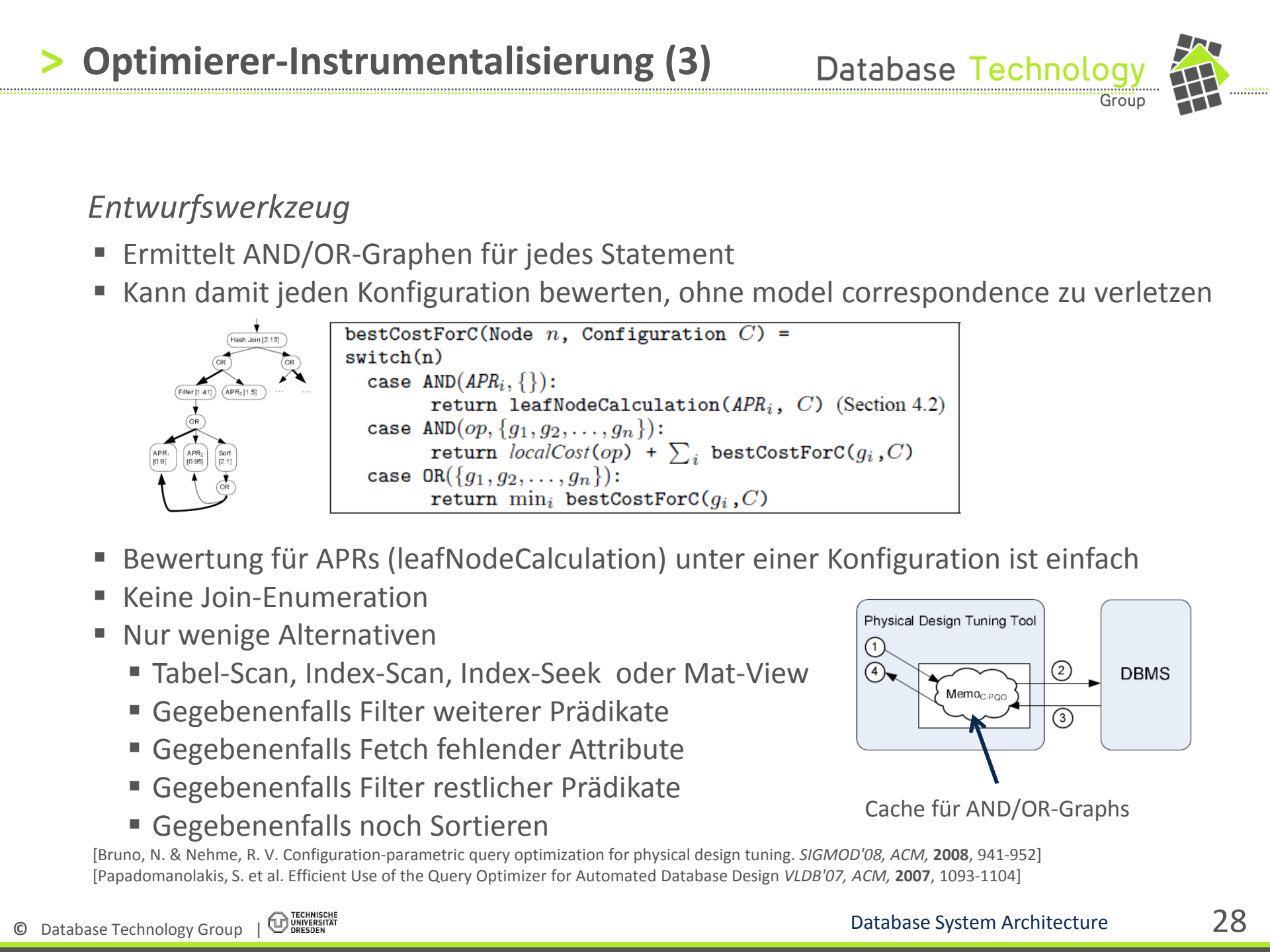
Instrumentalisierung des Optimierers

- Optimierer transformiert Anfrage auf Planoperatoren ohne Einzeltabellen-Zugriffe
- Für all Einzeltabellen-Zugriffe verwendet er APRs
- Ausgabe ist ein AND/OR Graph
  - Stellt alle betrachteten Teilpläne mit Kosten dar
  - Mit den verwendeten Planoperatoren
  - Welche Planoperatoren sich bedingen (AND)
  - Welche Alternativen sich gegen seitig Ausschließen (OR

```
SELECT * FROM R,S,T
WHERE R.x=S.x AND S.y=T.y
```




Statement
Optimierer-Instrumentalisierung
Anfrageoptimierung
Anfrage
Intern-darstellung
Zugriffs- & Integritäts-kontrolle
Anfrage-restruktu-rierung
Anfrage-trans-formation
Logischer DB-Prozessor (Anfrageprozessor)
Übersetzungszeit
Laufzeit
Hash Join [2.13]
OR
OR
Filter [1.41]
APR3 [1.5]
OR
APR1 [0.9]
APR2 [0.95]
Sort [2.1]
OR
Ergebnis
Ergebnis-bereit-stellung
Ausführungs-kontrolle
Code-erzeugung
Physischer DB-Prozessor (Anfrageausführer)



Entwurfswerkzeug

- Ermittelt AND/OR-Graphen für jedes Statement
- Kann damit jeden Konfiguration bewerten, ohne model correspondence zu verletzen


```
bestCostForC(Node n, Configuration C) =
switch(n)
  case AND(APR_i, {}):
        return leafNodeCalculation(APR_i, C) (Section 4.2)
  case AND(op, {g_1, g_2, ..., g_n}):
        return localCost(op) + Σ_i bestCostForC(g_i, C)
  case OR({g_1, g_2, ..., g_n}):
        return min_i bestCostForC(g_i, C)
```

- Bewertung für APRs (leafNodeCalculation) unter einer Konfiguration ist einfach
- Keine Join-Enumeration
- Nur wenige Alternativen
  - Tabel-Scan, Index-Scan, Index-Seek oder Mat-View
  - Gegebenenfalls Filter weiterer Prädikate
  - Gegebenenfalls Fetch fehlender Attribute
  - Gegebenenfalls Filter restlicher Prädikate
  - Gegebenenfalls noch Sortieren

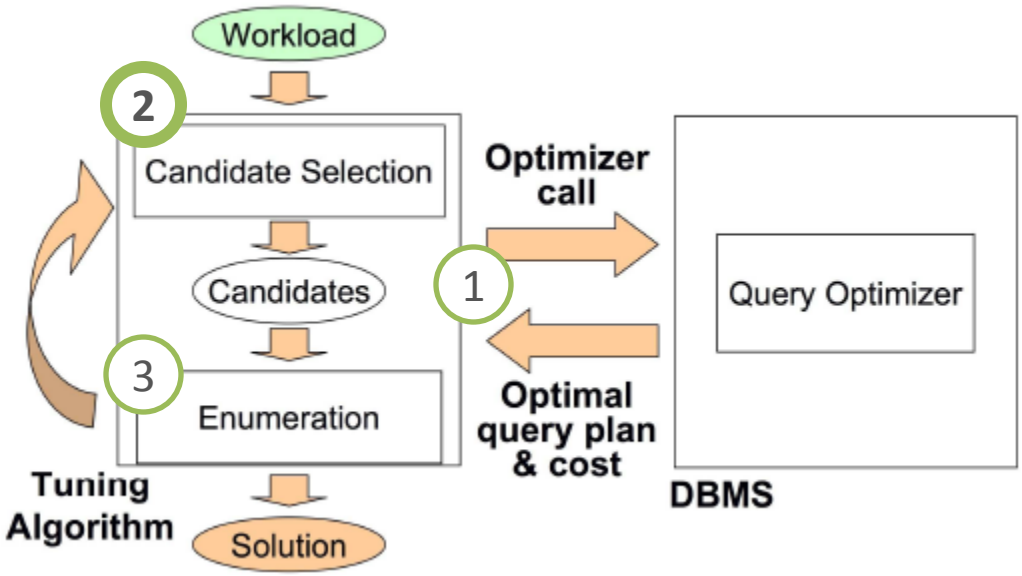
Cache für AND/OR-Graphs

[Bruno, N. & Nehme, R. V. Configuration-parametric query optimization for physical design tuning. *SIGMOD'08, ACM,* **2008**, 941-952]
[Papadomanolakis, S. et al. Efficient Use of the Query Optimizer for Automated Database Design *VLDB'07, ACM,* **2007**, 1093-1104]

1. Bewertung von Konfiguration

2. Auswahl von Kandidaten-strukturen

3. Enumeration von Konfigurationen



Kandidaten aus dem

- Zu viele mögliche Er
- Auf einer m-spaltige
  - 2^m mögliche meh
  - 2^m mögliche Proje
  - 2^n mögliche disju
  - 2^m mögliche Grup
  - 2^m mögliche Sorti
  - 2^{m-1} möglicher ve
  - 2^{n-1} möglicher ho
- Die meisten schema

Grafik: Andreas Hahn

Kandidaten aus der Arbeitslast

- Sinnvoll indizierbare Attribute
  - Attribute aus „sargable“ Prädikaten -> Prädikate die über einen Index auswertbar sind (SARG = Search ARGument)
  - Sortierungsattribute
  - Gruppierungsattribute
- Sinnvolle Indizes eines Statements indizieren ein oder mehrere sinnvoll indizierbare Attribute des Statements
- Sinnvolle Indizes einer Arbeitslast ist die Vereinigung der sinnvollen Indizes aller Statements der Arbeitslast

Definitions:
(i) *An indexable column for a query* in the workload is a column R.a such that there is a condition of the form *R.a operator Expression* in the WHERE clause. The operator must be among {=, <, >, <=, >=, *BETWEEN, IN*}.
Columns in GROUP BY and ORDER BY clauses are also considered indexable. For an Update query, the updated columns of the table are considered indexable.

Example 1: Indexable Columns of a Query
Consider the following query $\mathbf{Q_1}$:
SELECT * FROM onektup, tenktup1
WHERE (onektup.unique1 = tenktup1.unique1)
AND (tenktup1.unique2 between 0 and 1000)
From the above definition, it follows that the indexable columns of $\mathbf{Q_1}$ are {onektup.unique1, tenktup1.unique1, tenktup1.unique2}.

- Analog lassen sich auch sinnvoll Materialisierte Sichten oder Partitionieren aus der Arbeitslast ableiten

[Chaudhuri, S. & Narasayya, V. R. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB'97,* **1997**, 146-155]

Eingrenzung der Kandidaten auf die Besten pro Statement

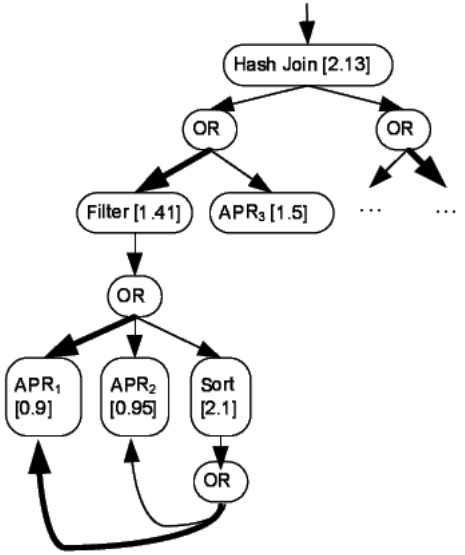
- Idee: Eine Struktur, die nicht in der besten Konfiguration für ein einzelnes Statement ist, wird auch nicht Teil der besten Gesamtkonfiguration
- Heuristik
- Kann gute Kandidaten für die Gesamtkonfiguration verpassen
- Beispiel:
  - Anfrage: SELECT a,b,c,d FROM Table A WHERE a < 100
    - Index I(a,b,c,d) ist optimal, benötigt aber viel Speicherplatz
    - Index I(a) ist etwas schlechter (Fetch auf b,c,d nötig), aber weniger Speicherplatz
    - Für die Gesamtkonfiguration könnte I(a) günstiger sein
- Zwei Implementierungsvarianten
  - Optimierer trifft direkt die Auswahl
  - Optimiererinstrumentalisierung
  - Entwurfswerkzeug ruft sich rekursiv auf
    - Arbeitslast ist das einzelne Statement
    - Kandidatenmenge sind sinnvollen Strukturen des Statements

1. For the given workload W that consists of n queries, we generate n workloads $W_1..W_n$ each consisting of one query each, where Wi = $\{Q_i\}$
2. For each workload W_i, we use the set of indexable columns I_i of the query in W_i as starting candidate indexes.
3. Let C_i be the configuration picked by index selection tool for W_i, i.e., C_i = Enumerate (I_i, W_i).
4. The candidate index set for W is the union of all C_i's.

[Chaudhuri, S. & Narasayya, V. R. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB'97,* **1997**, 146-155]

Eingrenzung der Kandidaten mittels Optimierer-Instrumentalisierung

- Ermitteln der besten Konfiguration für ein Statement aus AND/OR Graphen
- Beste Konfiguration C_o für Graphen bestimmen
  - Beste Konfiguration für jeden Einzel-Tabellenzugriff (APR) bestimmen (einfach)
  - APRs damit bewertet und mit Strukturen assoziiert
  - OR-Knoten auflösen, bester Zweig gewinnt
  - Strukturen an verbleibenden APRs bilden beste Konfiguration



[Bruno, N. & Chaudhuri, S. To Tune or not to Tune? A Lightweight Physical Design Alerter. *VLDB'06, ACM*, **2006**, 499-51]

Optimierer trifft direkt die Auswahl (erweitertes EXPLAIN)

- Ähnlich einer Anfrageausführung

- Optimierer generiert Kandidaten
  - Basierend aus Menge relevanter Indizes
    - Smart column Enumeration for Index Scans: Nutzt Expertenwissen über den Optimierer (SAEFIS)
    - Brute Force and Ignorance: Einfacher rekursiver Algorithmus (FBI)
- Kandidaten-Indizes werden im Schema-katalog eingetragen
- Statistiken für die Indizes werde erzeugt

ALGORITHM 1:
RECOMMEND_INDEXES(Statement S)

1. Enable "RECOMMEND INDEXES" mode
2. Enter the DB2 Optimizer
3. Inject the schema with virtual indexes using SAEFIS and generate their statistics
4. Inject the schema with virtual indexes using BFI and generate their statistics
5. Construct the best plan for S by calling the DB2 Optimizer
6. Scan the optimal plan, searching for virtual indexes
7. Submit these indexes back to the user as "recommended".

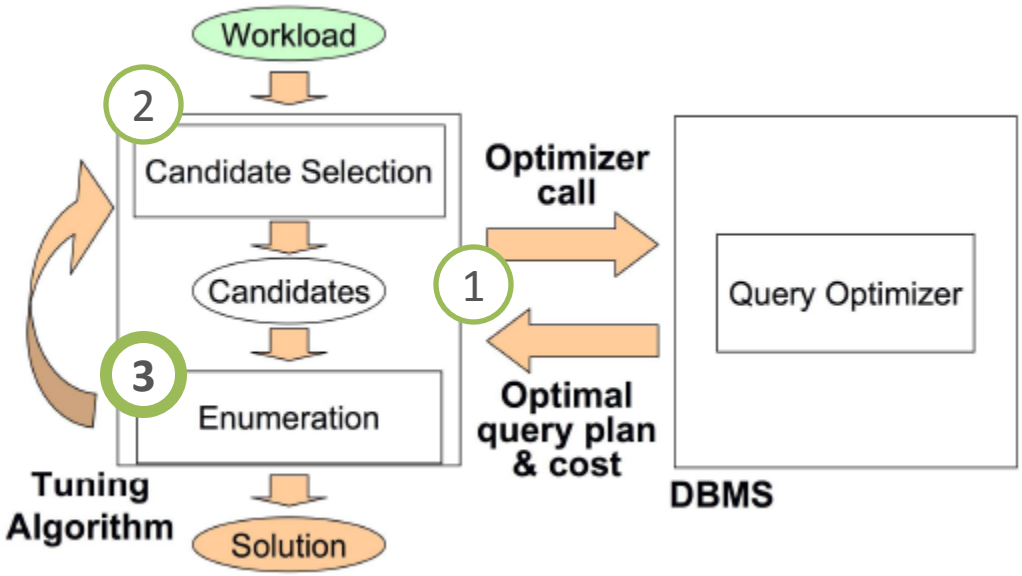
- Optimierer wählt Kandidaten aus
  - Optimierer erzeugt ganz normal für die Anfrage einen Anfrageausführungsplan
  - Plan wird NICHT ausgeführt, sondern nach verwendeten Indizes durchsucht

[Valentin, G. et al. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. ICDE'00, IEEE Computer Society, 2000, 101-110]

1. Bewertung von Konfiguration

2. Auswahl von Kandidaten-strukturen

3. Enumeration von Konfigurationen



Naive Enumeration

- Alle Teilmengen der Menge der Kandidatenstrukturen werden betrachtet
- 2^n Kandidatenkonfigurationen bei n Kandidatenstrukturen
- Skaliert nicht -> heuristische Enumeration notwendig

Konstruktive, heuristische Enumeration

- Schrittweises Zusammensetzen von Konfigurationen aus Kandidatenstrukturen
- Bewertung bei jedem Schritt
- Konfiguration werden größer, nähern sich der Speicherplatzbegrenzung an
- Stopp wenn sich keine besser Konfiguration mehr erreichen lässt

Destruktive, heuristische Enumeration

- Menge aller Kandidatenstrukturen ist Ausgangskonfiguration
- Reduzieren der Ausgangskonfiguration, so dass Speicherplatzbegrenzung eingehalten wird
- Reduktionsverfahren
  - Abschneiden und Variieren
  - Mischen und Reduzieren

Konstruktive, heuristische Enumeration

- Zwei Parameter
  - k: Konfiguration mit maximal k Strukturen sollen ausgewählt werden (alternativ Speicherplatzbegrenzung)
  - m: Bis zu einer Größe von m Strukturen werden Konfigurationen naive (nicht heuristisch) enumeriert

1. Let S = the best **m** index configuration using the *naïve enumeration algorithm*. **If m = k then exit**.
2. Pick a new index I such that Cost (S U {I}, W) <= Cost(S U{I'}, W) for any choice of I' != I
3. **If** Cost (S U {I}) >= Cost(S) **then exit**
   Else S = S U {I}
4. If |S| = **k** then exit
5. **Goto 2**

- Anteil naiver Enumeration wählt eine optimale Konfiguration der Größe m also Ausgangpunkt für die Heuristik
- Zweck: Berücksichtigung von Strukturinteraktion
- Einfluss von m
  - m=0: Ausschließlich heuristische Enumeration
  - m=k: Entspricht naiver Enumeration

[Chaudhuri, S. & Narasayya, V. R. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *VLDB'97*, **1997**, 146-155]

- Verallgemeinerung zur Baumexpansion
  - n besten Konfigurationen kommen in die nächste Runden (oben n=1)

[Finkelstein, S. J.; Schkolnick, M. & Tiberio, P. Physical Database Design for Relational Databases. *ACM TODS*, **1988**, 13, 91-128]

Destruktive, heuristische Enumeration – Abschneiden und Variieren

- Kandidatenmenge R nach Benefit-Größe-Verhältnis sortieren
- (Ähnliche Indizes werden kombiniert)
- Konfiguration ergibt sich durch Abschneiden der sortierten Kandidatenmenge unterhalb der Speicherplatzbegrenzung
- Wenn noch Zeit bleibt, wird Konfiguration leicht variiert und evaluiert
  - Austausch kleiner Teilmenge der Konfiguration mit Teilmenge der abgeschnittene Kandidatenmenge

For each index I in R

(a) I.benefit = S.cost_with_existing_indexes - S.cost_with_virtual_indexes

(b) I.size = bytes in index

Sort indexes in R by decreasing benefit-to-cost ratio.

Combine any index subsumed by an index with a higher ratio with that index.

Accept indexes from set R until disk constraint is exhausted.

while (time did not expire) repeat

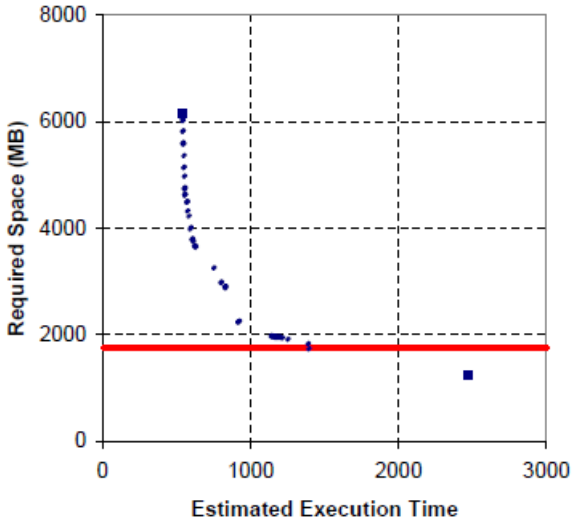
(a) TRY_VARIATION

[Zilio, D. C.; et al. DB2 Design Advisor: Integrated Automatic Physical Database Design. VLDB'04, Morgan Kaufmann, **2004**, 1087-1097]

Destruktive, heuristische Enumeration – Mischen und Reduzieren

- Mischen – Zusammenführen zwei Strukturen zu einer Strukturen
  - Definition: $I_1 = (K_1)$ und $I_2 = (K_2) \Rightarrow I_{1,2} = (K_1 + (K_1 - K_2))$.
  - Beispiel: $I_1 = (a, b, c)$ und $I_2 = (a, d, c) \Rightarrow I_{1,2} = (a, b, c, d)$
- Reduzieren – Verkleinern einer Strukturen
  - Definition: $I = (K) \Rightarrow I' = (K')$ wobei K' ist ein Präfix von K ist
  - Beispiel: $I = (a, b, c) \Rightarrow I' = (a,b)$
- Operationen erzeugt jeweils eine neue kleinere Konfiguration
- Die Beste, die in den Speicherplatz passt gewinnt

```
Search_Strategy (W:workload, B:space constraint)
01 Get optimal configurations for each q ∈ W // Section 2
02 c_best = ∪_{q∈W} optimal configuration for q
03 CP = { c_best }; c_best=NULL;   // cost(NULL)=∞
04 while (time is not exceeded)
05    Pick c ∈ CP that can be relaxed   // template
06    Relax c into c_new                // template
07    CP = CP ∪ { c_new }
08    if ( size(c_new) ≤ B ∧ cost(c_new)<cost(c_best) )
09       c_best=c_new
10 return c_best
```



[Chaudhuri, S. & Narasayya, V. R. Index Merging. *ICDE'99, IEEE Computer Society*, **1999**, 296-303]
[Bruno, N. & Chaudhuri, S. Automatic Physical Database Tuning: A Relaxation-based Approach. *SIGMOD'05, ACM*, **2005**, 227-238]
[Bruno, N. & Chaudhuri, S. Physical Design Refinement: The "Merge-Reduce" Approach. *EDBT'06, Springer*, **2006**, 3896, 386-404]

Statischer Online Entwurf

Problem 2 (Statischer Online Entwurf)

- Geben ist
  - eine Datenbank,
  - eine beobachtete Arbeitslast $W = \{S_1, S_2, \ldots, S_n\}$,
  - eine aktuelle Entwurfskonfiguration C,
  - eine Speicherplatzbegrenzung b
  - und eine Schwelle t,

  herauszufinden ist ob eine physische Entwurfskonfiguration C‘ existiert, so dass
  - $SIZE(C') \leq b$
  - und $EXEC(W,C)/EXEC(W,C) < t$.

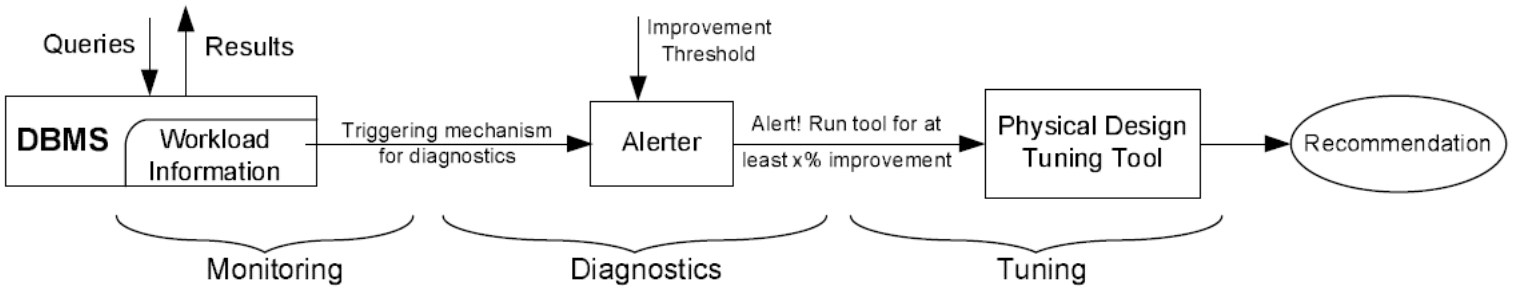
- Ist das Entscheidungsproblem einfacher zu beantworten als das Optimierungsproblem?

Szenario

- Arbeitslast ändert sich
- Wann soll der Entwurf re-optimiert werden?

Idee: Alerter-Werkzeug

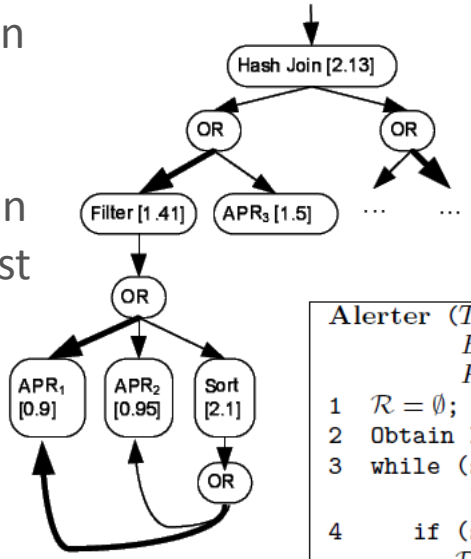
- Trennung von Diagnose und Optimierung
- Hoffnung: Diagnose ist weniger aufwendig als Optimierung?



[Bruno, N. & Chaudhuri, S. To Tune or not to Tune? A Lightweight Physical Design Alerter. *VLDB'06, ACM*, **2006**, 499-51]

Basis AND/OR Graph der Optimierer-Instrumentalisierung

- AND/OR Graphen mit der Arbeitslast aufzeichnen (Online-What-if)
- Beste Konfiguration C_0 aus AND/OR Graphen
- Suchen nach Konfiguration C‘
  - Verkleinern von C_0 durch Mischen und Reduzieren
  - Aggressive Greedy-Strategie
  - Wird eine Konfiguration gefunden ist die um die Schwelle t besser ist als die aktuelle -> Alarm
- Falschalarme nicht möglich (kein false positives)
- Verpasste Alarme möglich (false negatives)



Alerter (T:AND/OR request tree, B_{min}, B_{max}:storage constraints, P:minimum percentage improvement)

1 $\mathcal{R} = \emptyset$; i=0
2 Obtain locally optimal configuration $\mathcal{C}_0$
3 while (size($\mathcal{C}_i$)>B_{min} and $100\% \cdot \Delta^{T}_{\mathcal{C}_i} / cost_{current} > P$)
4 if (size($\mathcal{C}_i$)<B_{max}) $\mathcal{R} = \mathcal{R} \cup \mathcal{C}_i$
5 Pick transformation TR that minimizes penalty($\mathcal{C}_i$, TR($\mathcal{C}_i$))
6 $\mathcal{C}_{i+1}$ = TR($\mathcal{C}_i$)
7 i=i+1
8 if ($\mathcal{R} \neq \emptyset$) ALERT($\mathcal{R}$)

[Bruno, N. & Chaudhuri, S. To Tune or not to Tune? A Lightweight Physical Design Alerter. *VLDB'06, ACM*, **2006**, 499-51]

Dynamischer Online Entwurf

Problem 3 (Dynamischer Online Entwurf)

- Geben ist
  - eine Datenbank,
  - ihre aktuelle Arbeitslast W,
  - ihre aktuelle Entwurfskonfiguration C,
  - eine Speicherplatzbegrenzung b
- prognostizieren die kommende Arbeitslast W‘ und finde eine Entwurfskonfiguration C‘, so dass
  - SIZE(C‘) $\leq$ b
  - und EXEC(W‘,C‘) minimal wird.

Blick in die Glaskugel?

- Nein, keine echte Vorhersage
- Arbeitslastvorhersage ist ein offene und interessante Forschungsfrage

Epochenansatz

- Gibt die Länge einer zusammenhängenden Arbeitslastbeobachtung vor
- Nach jeder Epoche arbeitet das Online-Tuning-Werkzeug
- Annahme: Arbeitslast ist autokorreliert
  → Arbeitslast der nächsten Epoche ist hinreichend ähnlich zu beobachteten
- Konfiguration wird auf beobachteter Epoche optimiert, auf nächste angewendet
  - Einbeziehung mehrere Epochen über Gewichtung denkbar
- Optimale Epochenlänge?
  - Bei zu kleinen Epochen
    - Gesamtcharakteristik der Arbeitslast geht verloren
    - Zu viel Overhead durch Online-Tuning-Werkzeug
  - Bei zu großen Epochen
    - Zu langsame Reaktionszeit bei Arbeitslaständerungen
  - Intuition: Relativ kurze Zeitspanne (30sec – 5min)

Materialisierte Strukturen vs. virtuelle Strukturen (Kandidatenstrukturen)

- Materialisierte Strukturen M existieren in der aktuellen Konfigurationen
- Virtuelle Strukturen V sind Kandidaten für die nächste Konfiguration
- M∪V werden kontinuierlich bewertet (Online-What-if)
  - Anzahl der virtuellen Strukturen beeinflusst Overhead
- M∪V sind Kandidaten für Optimierung der Konfiguration

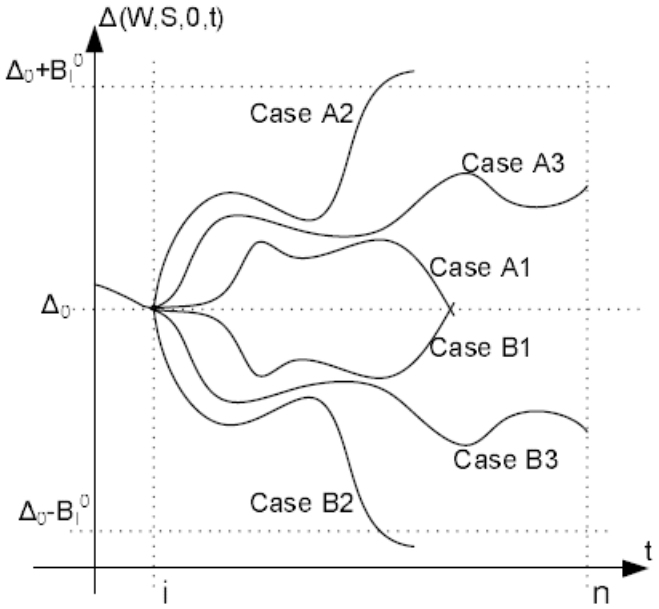
Virtuelle Strukturen vs. heiße Strukturen

- Zusätzliche Gruppe der heißen Strukturen H (vielversprechende Kandidaten)
- Einfach Kostenabschätzung (ohne What-if) unterscheidet zwischen V und H
- Bewertet und optimiert wird M∪H
- Adaptiver Ansatz
  - Last stetig, die Konfiguration gut → wenig vielversprechende Kandidaten
    → keine Notwendigkeit zum Optimieren → Menge H klein → geringer Overhead
  - Last in Veränderung, die Konfiguration nicht mehr optimal
    → mehr vielversprechende Kandidaten → Notwendigkeit zum Optimieren
    → Menge H groß → mehr Overhead
  - Overhead passt sich Notwendigkeit zum Optimieren an

[Schnaitter, K.; Abiteboul, S.; Milo, T. & Polyzotis, N. On-Line Index Selection for Shifting Workloads. SMDB'07, **2007**]

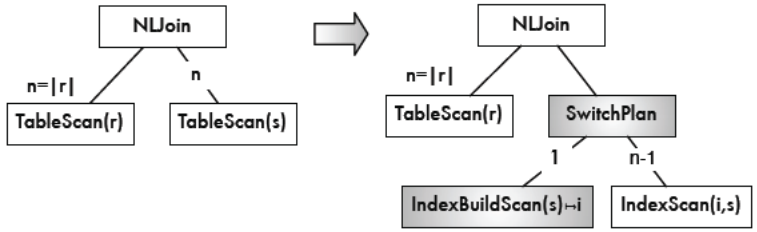
Konfigurationsänderung ist nicht umsonst

- Aufwand fürs Verändern der Konfiguration muss berücksichtigt werden
  - Anlegen von Strukturen
  - Löschen von Strukturen
- Transformationskosten bilden Schwellwert für möglichen Gewinn durch eine Konfigurationsänderung
- Veranschaulichung
  - Δ_0: Gewinn durch eine Struktur I am Anfang einer Epoche
  - B_I^0: Aufwand fürs Erzeugen von Struktur I
  - Fälle A1, A2, A3: I existiert noch nicht
  - Fälle B1, B2, B3: I existiert bereits
  - Vorgehen:
    Anlegen von I bei A2
    Nichts tun bei A1 und A3 (nicht anlegen von I)
    Löschen bei von I bei B2
    Nichts tun bei B1 und B3 (behalten von I)



[Bruno, N. & Chaudhuri, S. An Online Approach to Physical Design Tuning. *ICDE'07, IEEE*, **2007**, 826-835]

Strukturerzeugende Anfrageverarbeitung



- Anfrage lesen die zur Strukturerzeugung notwendigen Daten
- Beispiel Indexerzeugung: Spezielle Planoperatoren
  - IndexBuildScan ist eine TableScan der einen Index mit erzeugt
  - SwitchPlan schalten nach dem ersten Scan-Aufruf auf eine IndexScan um

[Graefe, G. Dynamic Query Evaluation Plans: Some Course Corrections? *IEEE Data(base) Engineering Bulletin,* **2000***, 23*, 3-6]
[Sattler, K.-U.; Luehring, M.; Schmidt, K. & Schallehn, E. Autonomous Management of Soft Indexes. *SMDB'07,* **2007**]

Strukturpassivierung

- Strukturen können (in einigen DBMS) passiviert werden
  - Struktur bleibt erhalten, belegt werden Speicherplatz (-> Verletzung der Speicherplatzbegrenzung)
  - Kann nicht für Anfrageverarbeitung genutzt werden
  - Wird nicht gewartet
- Beim Reaktivieren von Strukturen wird Wartung mit Hilfe REDO-Logs nachvollzogen
- Passivieren/Aktivieren kann deutlich günstiger sein Löschen/Erzeugen

[Bruno, N. & Chaudhuri, S. An Online Approach to Physical Design Tuning. ICDE'07, IEEE, **2007**, 826-835]

Dynamischer Offline Entwurf

Problem 4 (Dynamischer Offline Entwurf)

- Geben ist
  - eine Datenbank,
  - eine sequenzielle Arbeitslast W = [S_1, S_2, …, S_n],
  - eine initiale Entwurfskonfiguration C_0,
  - eine Speicherplatzbegrenzung b
- gesucht ist eine Sequenz von Entwurfskonfigurationen C = [C_1, C_2, …, C_n], so dass
  - $\forall i : SIZE(C_i) \leq b$
  - und die dynamischen Ausführungskosten EXEC(W,C) minimal werden.

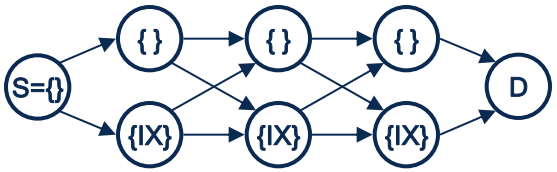
Definition 3 (Dynamische Ausführungskosten)

- Die dynamischen Ausführungskosten EXEC(W,C) einer Statement-Sequenz W = [S_1, S_2, …, S_n] unter Verwendung einer physischen Konfigurationssequenz C = [C_1, C_2, …, C_n] ist die Summe der Ausführungskosten EXEC(S_i,C_i) jedes Statements S_i aus W unter Verwendung der Konfiguration Ci und der Transformationskosten von Ci-1 nach Ci.

$$EXEC(W,C) = \sum_i^n (TRANS(C_{i-1},C_i) + EXEC(S_i,C_i))$$

Problem lässt sich als Graph beschreiben (Sequenzgraph)

- Start- und Endknoten
- Pro Statement und möglicher Konfiguration ein Knoten
- Zwischen jeder Paarung eine Kante
- Knoten repräsentieren Statement-Ausführungskosten
- Kanten repräsentieren Konfigurationstransformations-kosten



Optimale Lösung

- Kürzester Pfad im Sequenzgraph
- Komplexität $O(n2^{2m})$

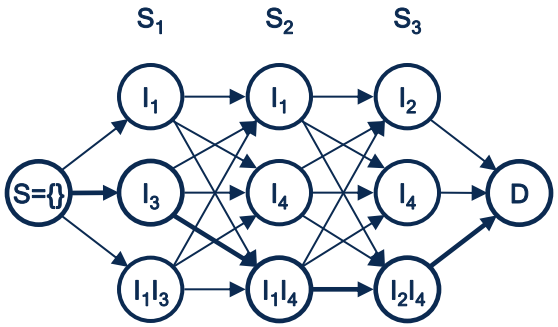
Generalizing the graph to N-statement sequence and M structures to generate an optimal solution is conceptually straightforward. In each stage 1 through N, there are 2^M nodes, each representing a configuration. This is because each subset of input structures defines a configuration. We refer to the solution that enumerates all 2^M configurations exhaustively at each stage as EXHAUSTIVE. It is important to note that the graph has O ($N*2^M$) nodes and O ($N*2^{2M}$) edges, i.e., the graph is exponential in the number of input structures. Though the shortest path can be solved in linear complexity of number of nodes and edges as above, these however are exponential in number of structures.

[Agrawal, S.; Chu, E. & Narasayya, V. R. Automatic Physical Database Tuning: Workload as a Sequence. *SIGMOD'06, ACM,* **2006**, 683-694]

Heuristische Lösung

- Erforderlich: Bestimmung von Konfigurations-kandidaten unter Berücksichtigung von Transformationskosten
- UnionPair
  - Bestimmt aus zwei Konfigurationssequenzen die optimale Mischung
  - Kontenzahl pro Statement konstant = 3
  - Komplexität O(n)
- Gesamtvorgehen
  - Optimale Sequenz für jede Struktur einzeln bestimmen
  - Sequenzen schrittweise mit UnionPair mischen keine besser gefunden wird
  - Alle dabei entstanden Konfiguration sind Kandidaten

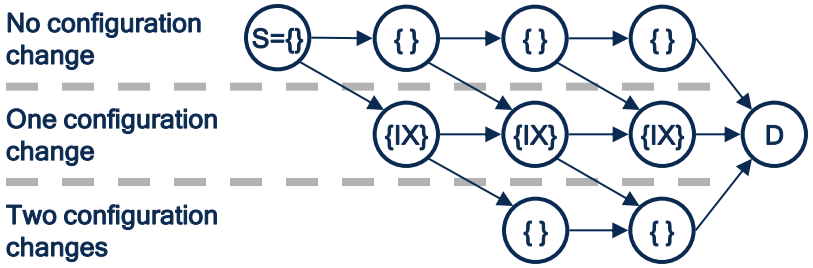
[Agrawal, S.; Chu, E. & Narasayya, V. R. Automatic Physical Database Tuning: Workload as a Sequence. *SIGMOD'06, ACM,* **2006**, 683-694]



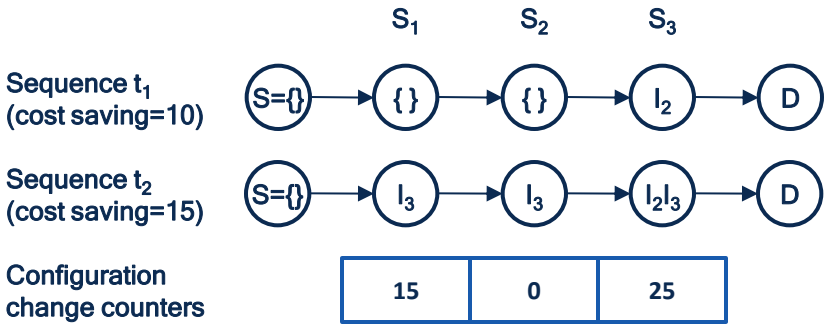
1. For every structure in the set **S**={s_1,..s_M}, find the optimal solution using the graph formulation as described in Section 3. At this point, we have a set of solutions **P** for individual structures. Let **P**= {p_1,…p_M} and $\mathbf{p_i}$=[a_{i1},S_1…S_N,a_{iN+1}].
2. Let C be the set of all configurations over all p_i's.
3. Run a greedy search over **P** as follows.
   a. Let **r** = [c_1, S_1, … c_N, S_N, c_{N+1}] represent the least cost solution in **P**. **P**=**P**-{**r**}. Let C = C ∪ {c_1, … c_{N+1}}
   b. Pick an element **s** from **P** such that **t**=**UnionPair**(**r**,**s**) has the minimal sequence execution cost for among all elements of **P** and sequence execution cost of **t** is less than that of **r**. If no such element exists go to step 4. **P**=**P**-{**s**}. **P**=**P** ∪{**t**}. Go to a.
4. Generate the graph with all the configurations in **C** at each stage. Run the shortest path over this graph and return the solution.

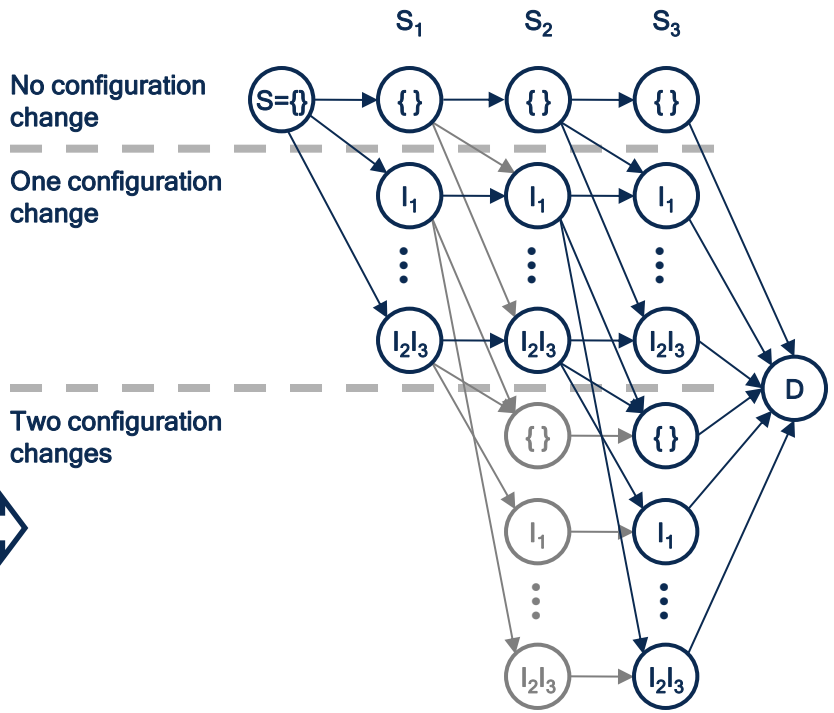
Limitierung der Konfigurationsänderungen

- Optimale Lösung



- Heuristische Lösung





[Voigt, H.; Lehner, W. & Salem, K. Constrained Dynamic Physical Database Design. *SMDB'08,* **2008**]

Einleitung

- Physisches Schema
- Physischer Entwurf

Automatische Entwurfswerkzeuge

- Statischer Offline Entwurf
  - Bewertung von Konfiguration
  - Auswahl von Kandidatenstrukturen
  - Enumeration von Konfigurationen
- Statischer Online Entwurf
- Dynamischer Online Entwurf
  - Epochenkonzept
  - Transformationskosten
  - Senken von Transformationskosten
- Dynamischer Offline Entwurf