Datenbankadministration

7. Anfrageoptimierung I

AG DBIS University of Kaiserslautern, Germany

Karsten Schmidt kschmidt@informatik.uni-kl.de
(Vorlage TU-Dresden)

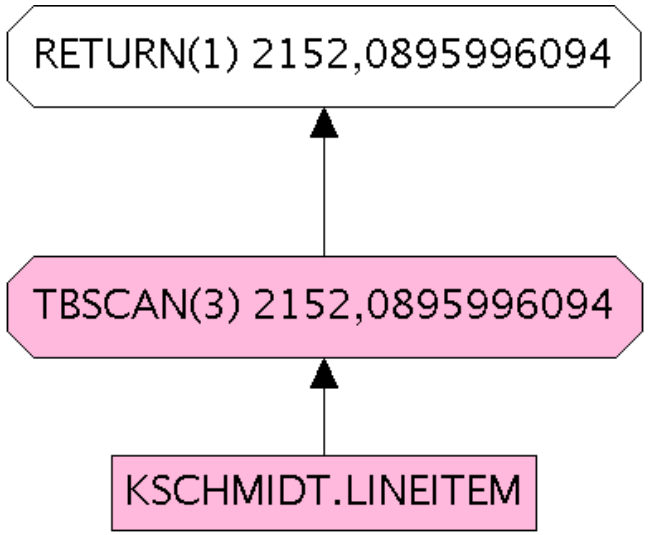
Wintersemester 2008/2009

Übersicht

- **Anfrageplan**
- **Grafische Tools**
  - Control Center (db2cc)
  - Command Editor (db2ce)
  - Achtung: X-Umleitung bei ssh-Session notwendig
- **Optimierung**
  - Indizes
  - Materialisierte Sichten
- **Zusammenfassung**

Anfrageplan

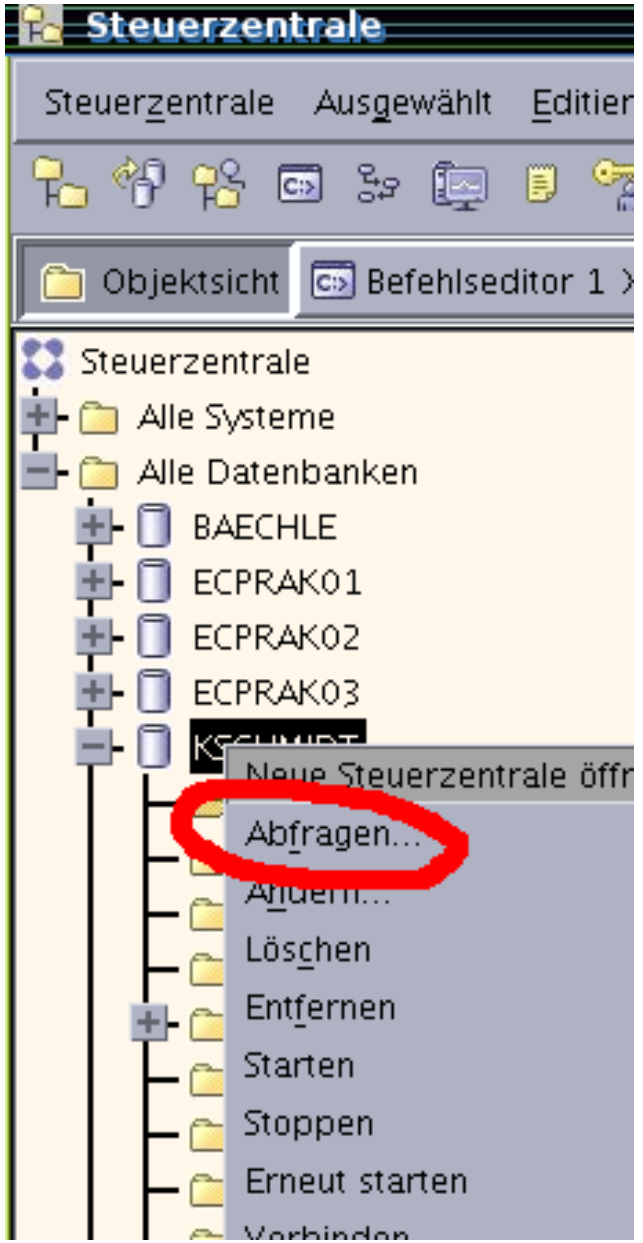
- **Anfrageplan**
  - Ergebnis der Anfrageoptimierung
  - Abfolge von Datenbankoperatoren mit geschätzten Kosten
  - Beispiel: `SELECT * FROM LINEITEM`



- **Kostenmodell in DB2**
  - basiert auf einer abstrakten Einheit: Timerons
  - Mischung aus CPU- und E/A-Kosten
  - Kostenschätzung verwendet Faustregeln und Statistiken über die Daten

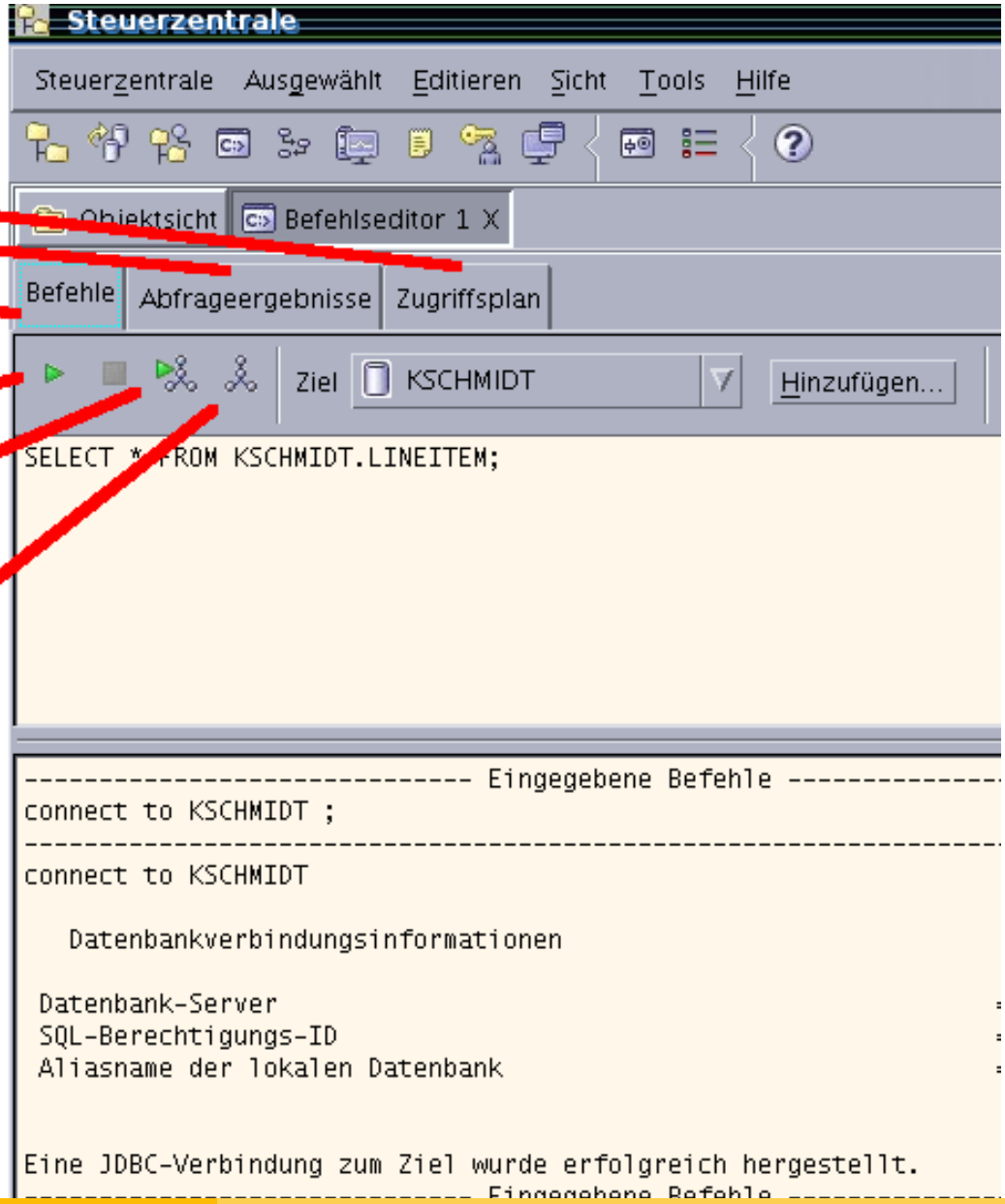
Visual Explain

- **Visual Explain**
  - DB2-Tools zum Anzeigen von Anfrageplänen
  - in Kommandozentrale (`db2cc`)
    - Rechtsklick auf alle Datenbanken
    - „Abfragen“auswählen
  - ermöglicht Anzeige von Detailinformationen zu jedem Operator
- `EXPLAIN`-**Kommando**
  - speichert Anfrageplan in Explain-Tabellen
  - Explain-Tabellen müssen erst erzeugt werden (automatisch → Visual Explain)
  - intern von Visual Explain verwendet



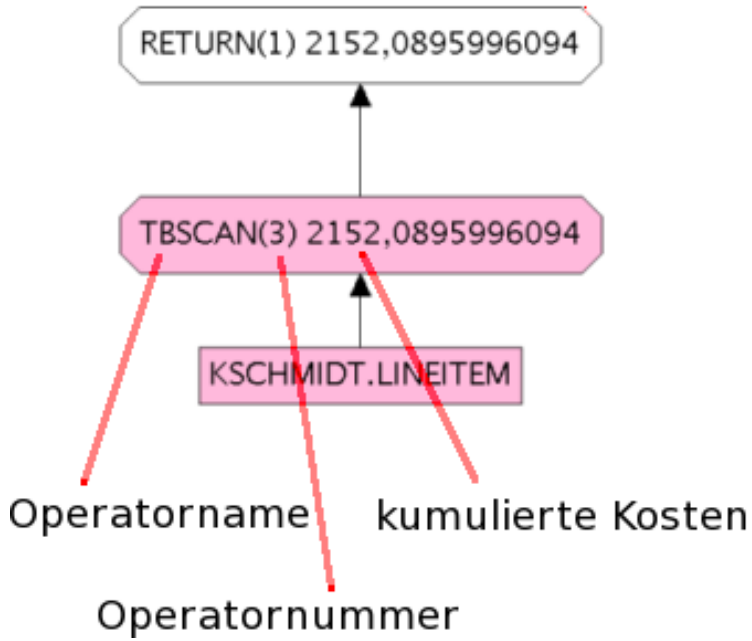
Visual Explain

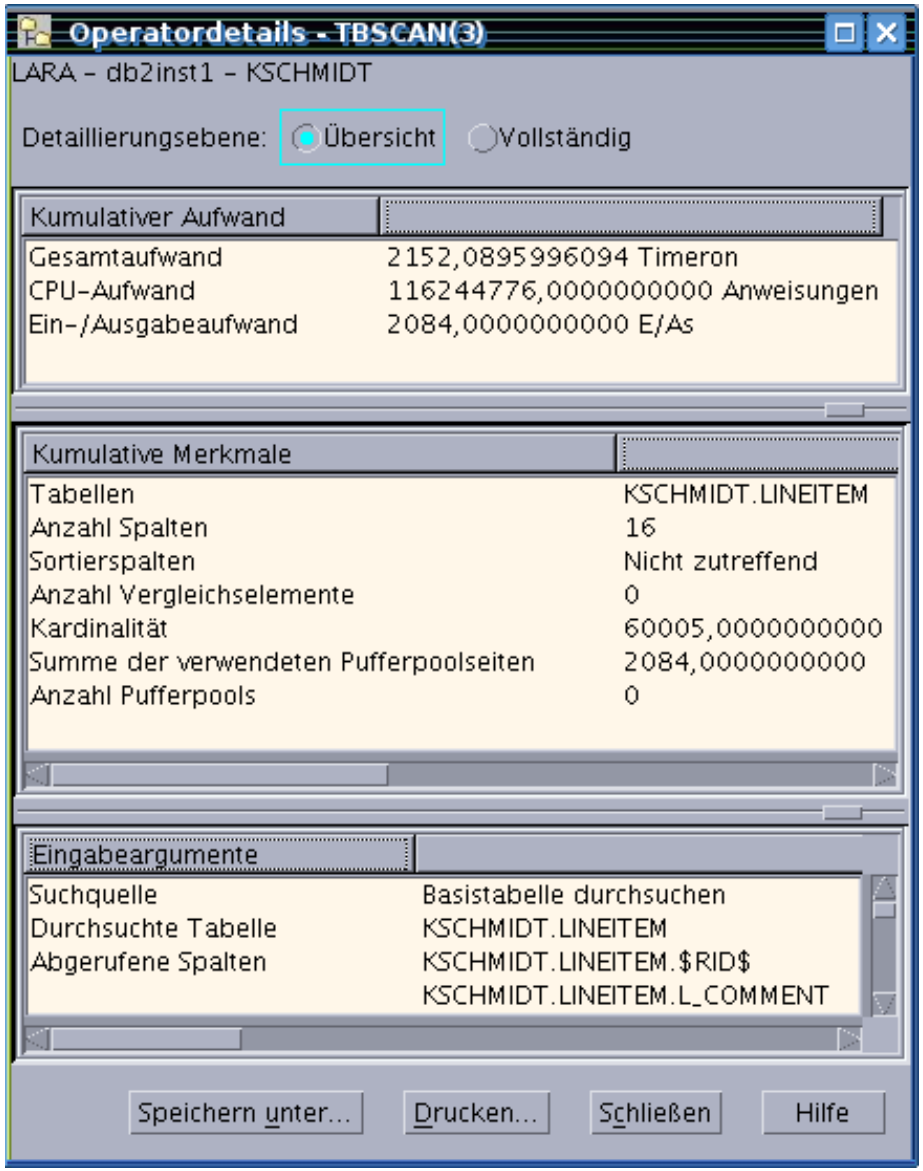
- Plan anzeigen
- Ergebnis anzeigen
- Anfrage eingeben
- Ausführen
- Ausführen und Plan
- Plan



Visual Explain

- **Operatordetails**
  - Doppelklick auf Operator
  - wichtig zum Auffinden von Engpässen
- **Beispiel**
  - `SELECT * FROM LINEITEM`





Operatoren

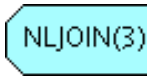
- **Ein-/Ausgabeoperatoren**
  - Eingaberelationen (kein Operator) 

    - stehen an den Blättern
    - repräsentiert Tupel
  - Relationenscan (*table scan*) 

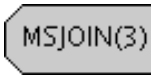
    - sequentielles Auslesen der gesamten Relation
    - ggf. direkt Anwendung von Selektionen (Filter)
    - Anwendung
      - falls kein Index vorhanden
      - niedrige Selektivität (hohe Anzahl von zurückgelieferten Tupeln)
      - kleine Tabelle
  - Rückgabe des Ergebnisses 

    - steht an Wurzel bei `SELECT`-Anweisung

Operatoren

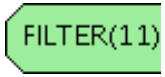
- **Verbundoperatoren** (*join*)
  - binär, für Verbund R⋈S mit N bzw. M Tupeln
  - Nested-Loop-Join 

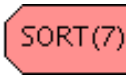
    - beliebige Verbundbedingungen
    - Verfahen: jedes Tupel aus R mit jedem Tupel aus S vergleichen
    - Aufwand: Laufzeit O(N*M), Speicher O(1)
  - Hash-Join 

    - nur Equi-Join
    - Verfahren: Hashtabelle von R aufbauen und S dagegen testen
    - Aufwand: Laufzeit O(N+M), Speicher O(N)
  - Merge-Scan-Join 

    - nur Equi-Join von nach Verbundattribut sortierten Relationen
    - Verfahren: schritthaltende Scans beider Relationen
    - Aufwand: Laufzeit O(N+M), Speicher O(1)

Operatoren

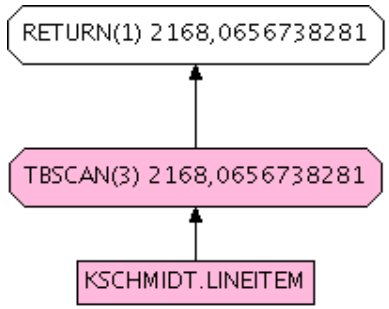
- **Weitere Operatoren**
  - Selektion 

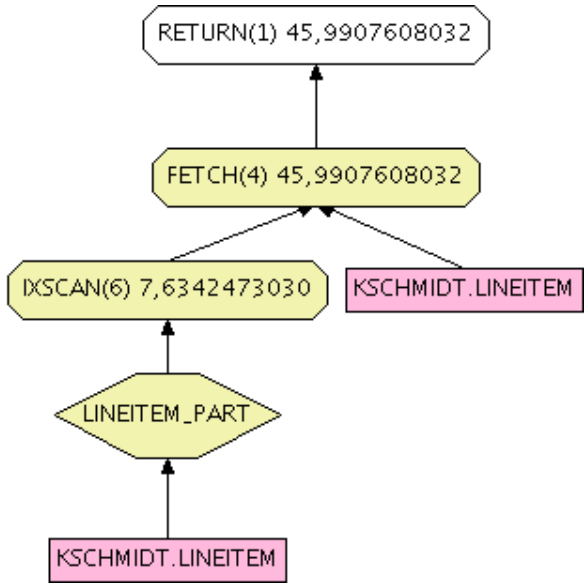
  - Sortierung 

  - Gruppierung 

  - in temporäre Relation zwischenspeichern 

- **Anfrageoptimierung**
  - basiert auf Kostenmodell
  - für eine SQL-Anweisung sind mehrere Anfragepläne möglich
    - Vertauschbarkeit von Operatoren
    - mehrere Implementationen für logische Operationen (z.B. Verbund)
    - optionale Nutzung von Zusatzinformationen wie Indizes
  - Ziel: Auswahl des Planes mit den geringsten Kosten
    - verschiedene Pläne aufstellen
    - Kosten schätzen





Indizes

Indizes

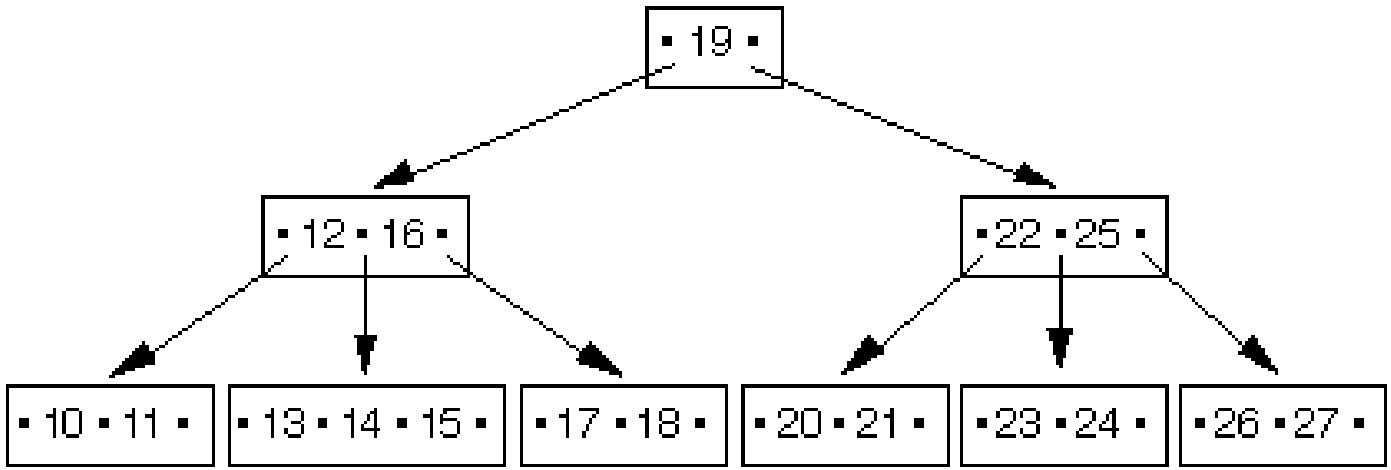
- **Index**
  - wird für ein oder mehrere Attribute angelegt
  - Einsatzgebiete
    - schneller Zugriff auf bestimmte Ausprägungen oder Bereiche der Indexattribute
    - materialisierte Sortierung der Ausprägungen der Schlüsselattribute (für `ORDER BY`, Merge-Scan-Join, `GROUP BY`)
    - Erzwingen von Eindeutigkeit
  - aber: zusätzlicher Aufwand für Indexaktualisierung notwendig!
  - Zielkonflikt: Zugriffsbeschleunigung vs. Aktualisierungsaufwand

    - sinnvoll für große Relationen, auf die selektiver Zugriff benötigt wird, z.B.:

      ```
      SELECT * FROM LINEITEM WHERE L_PARTKEY = 20
      ```

Indexstrukturen

- **Index-Strukturen**
  - B+-Baum (B für Bayer, balanciert, block-orientiert?)
    - balancierter Baum mit fester Anzahl Einträge pro Knoten
    - innere Knoten enthalten Schlüsselwerte
    - Blattknoten enthalten Satzadresse (RID) bzw. Daten



  - weitere Informationen - siehe Vorlesungen

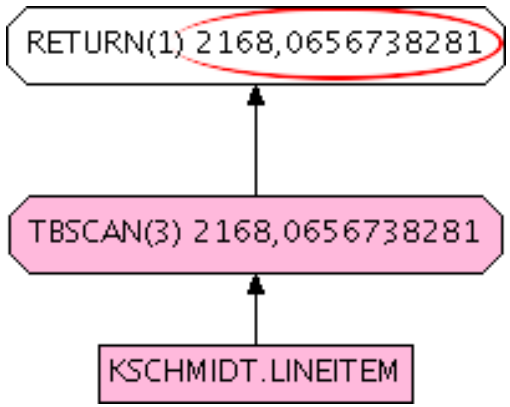
Index-DDL

- **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 Bereichsanfragen 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>`

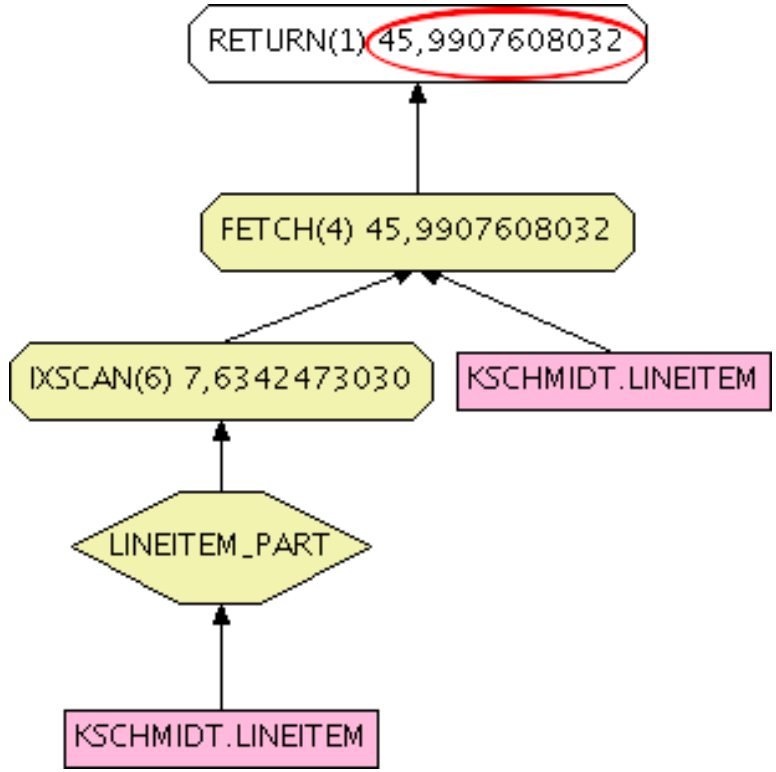
Beispiel für Indizes

- Anfrage: `SELECT * FROM LINEITEM WHERE L_PARTKEY = 20`
- `CREATE INDEX LINEITEM_PART ON LINEITEM(L_PARTKEY)`

ohne Index

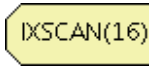


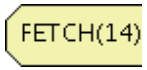
mit Index auf `L_PARTKEY`

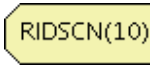


Operatoren für Indexzugriff

- **Operatoren für Indexzugriff**
  - Eingabeindex 

    - Operand für Indexscan
  - Indexscan 

    - entspricht Zugriff auf den Index
    - liefert Satzadressen (RID, *row identifier*)
  - Auslesen von Tupeln 

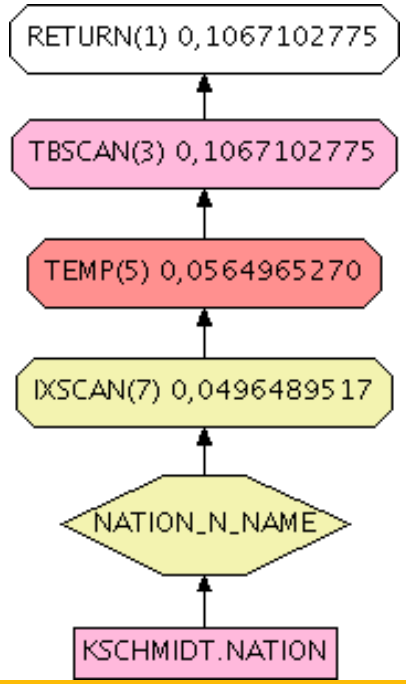
    - zu gegebenen RIDs Tupel aus der Eingaberelation lesen
  - Satzadressenscan 

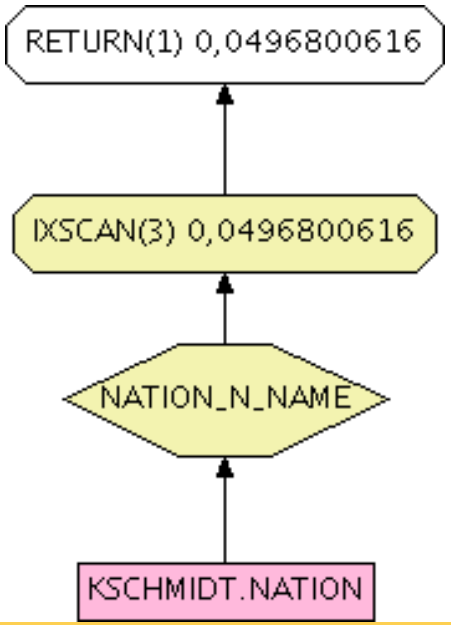
    - tupelweiser Zugriff auf RID-Liste
  - RID-Schnitt 

    - für konjunktiv verknüpfte Anfragebedingungen

Rückwärtsiteration

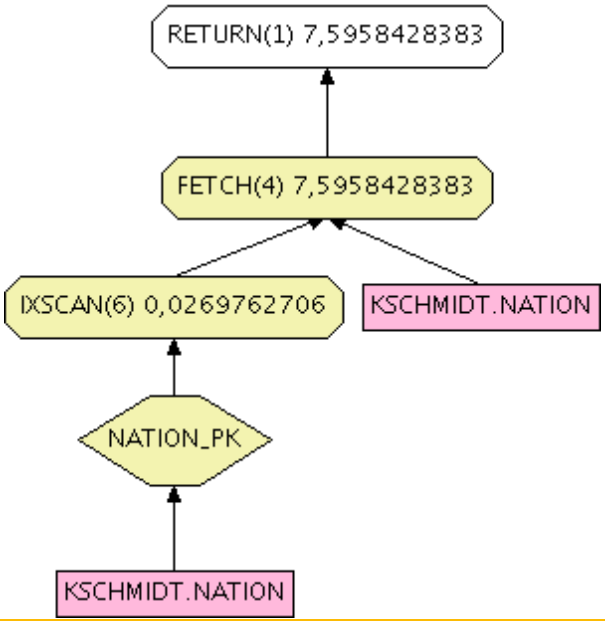
- **Rückwärtsiteration**
  - Schlüsselwort `REVERSE SCANS`
    - Zulassen (`ALLOW`) bzw. Verhindern (`DISALLOW`)
  - erlaubt Rückwärtslesen des Index
    - entspricht Doppelverkettung der Blätter des B+-Baums
  - Beispiel
    - `CREATE INDEX NATION_N_NAME ON NATION(N_NAME ASC) ALLOW REVERSE SCANS`
    - `SELECT N_NAME FROM NATION ORDER BY N_NAME DESC`

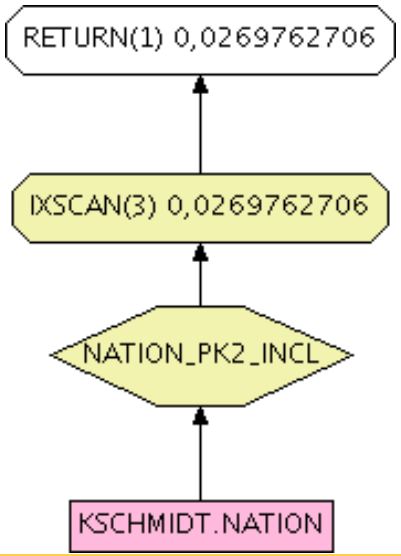




Index mit zusätzlichen Attributen

- **Index mit zusätzlichen Attributen**
  - Schlüsselwort `INCLUDE`, Voraussetzung: `UNIQUE`-Index
    - Aufnahme zusätzlicher Spalten in die Index-Knoten
  - Unterstützung des Index-only Datenzugriffes
    - kein Zugriff auf Relation mehr notwendig, wenn alle Attribute der Anfrage im Index enthalten
  - Beispiel
    - `CREATE UNIQUE INDEX NATION_PK2_INCL ON NATION(N_NATIONKEY) INCLUDE (N_NAME)`
    - `SELECT N_NAME FROM NATION WHERE N_NATIONKEY = 1`

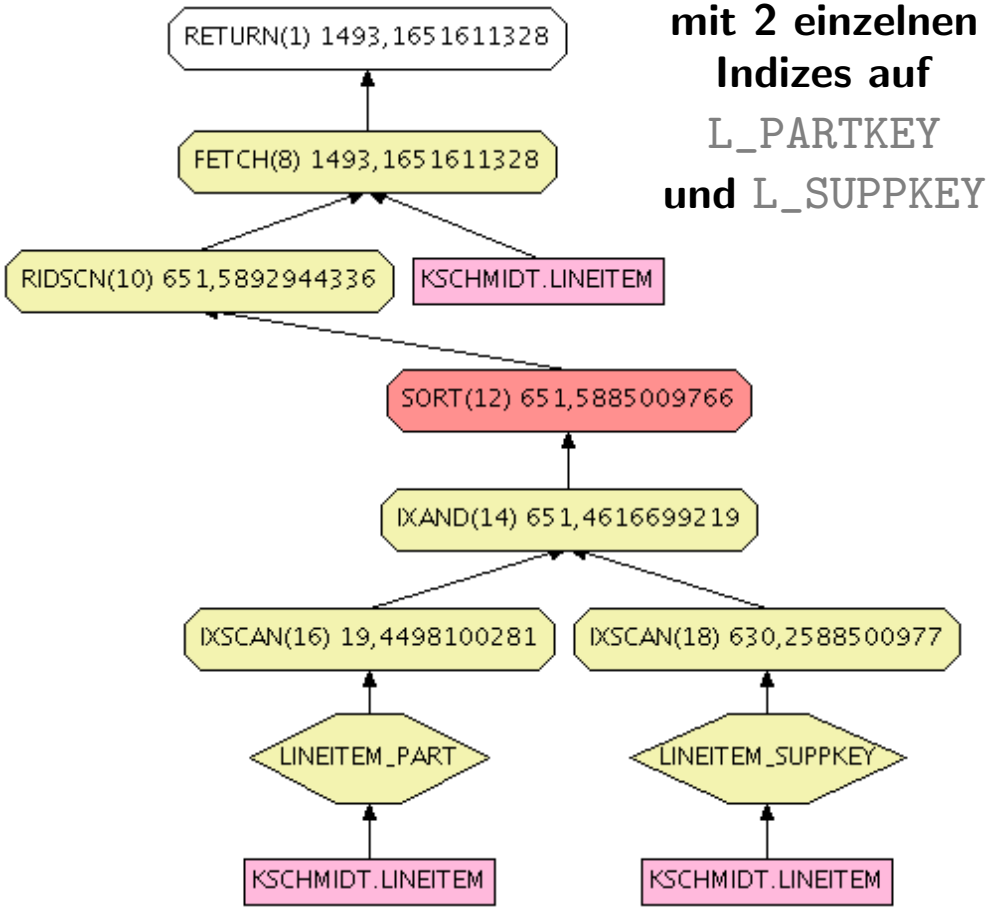




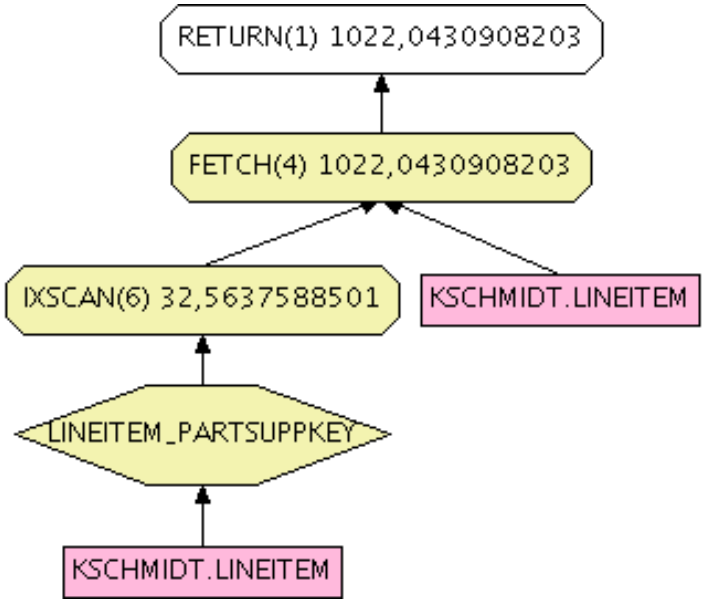
Mehrdimensionale Indizes

- **Mehrdimensionale Indizes**
  - besteht aus mehr als einem Schlüsselattribut in `ON`-Klausel
  - Umsetzung z.B. mittels Schlüsselkonkatenation oder MDB-Baum
  - Beispiel: `SELECT * FROM LINEITEM WHERE L_PARTKEY < 20 AND L_SUPPKEY < 30`

mit 2 einzelnen Indizes auf `L_PARTKEY` und `L_SUPPKEY`



mit kombiniertem Index auf (`L_PARTKEY`,`L_SUPPKEY`)



Materialisierte Sichten

Rückblick: Sicht

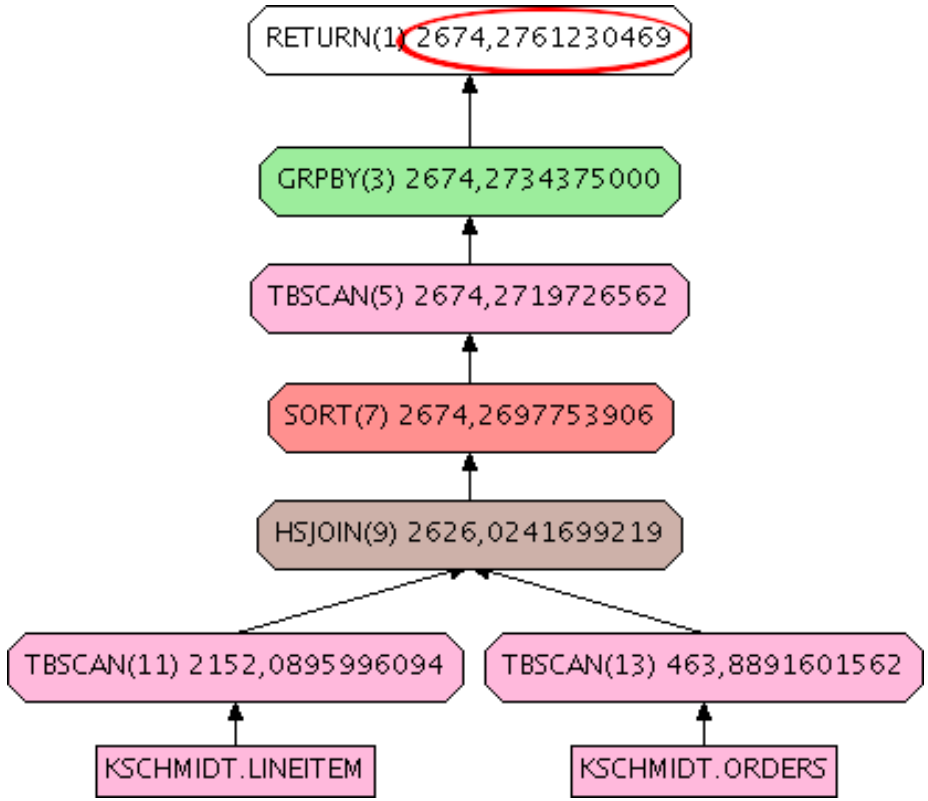
- **Sicht** (*view*)
  - virtuelle Relation zur Vereinfachung von Anfragen bzw. nutzerspezifische Datendarstellung
  - Beispiel
    - ```
      CREATE VIEW MYVIEW AS
      SELECT C_NATIONKEY, COUNT(*) AS CNT
      FROM CUSTOMER
      GROUP BY C_NATIONKEY
      ```
  - Definition im Datenbankkatalog abgelegt (`SYSCAT.VIEWS`)
  - Verwendung der Sicht explizit innerhalb SQL-Anfragen
    - `SELECT * FROM MYVIEW`
  - Inhalt wird bei Anfrage auf der Sicht berechnet $\rightarrow$ aufwendig, wenn oft verwendet
  - Alternative: Inhalt in der Datenbank ablegen = materialisieren
      ```

Materialisierte Sicht

- **Materialisierte Sicht**
  - auch *materialized view*, *materialized query table*, *summary table*
  - für SQL-Anfrage definiert, Ergebnis wird in der Datenbank abgespeichert
  - Änderung der Quelldaten können ggf. an die materialisierte Sicht weitergegeben werden
  - Anlegen einer materialisierten Sicht
    - `CREATE [SUMMARY] TABLE <mv-name> [(<col-name>, ...)]`
      `AS (<fullselect>)`
      `{WITH NO DATA |`
      `DATA INITIALLY DEFERRED REFRESH`
      `{DEFERRED|IMMEDIATE}`
      `[{ENABLE|DISABLE} QUERY OPTIMIZATION]`
      `[MAINTAINED BY {SYSTEM|USER}]}`
      - `<fullselect>` entspricht `SELECT`-Anweisung (alle Spalten benennen oder angeben!)
  - Erstellen von Indizes ist möglich

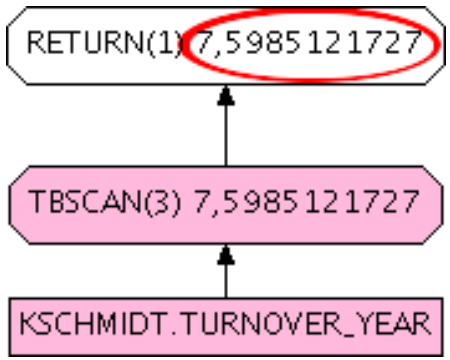
Beispiel ohne materialisierte Sicht

- **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

- **Beispiel (mit materialisierter Sicht)**
  - ```
    CREATE TABLE TURNOVER AS (
        SELECT YEAR(O_RDERDATE) 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 $\longrightarrow$ effizienter


    ```

Materialisierte Sicht - Wartung

- **Wartung von materialisierten Sichten** (*maintenance*)
  - Art der Wartung wird bei Erstellung angegeben
  - keine Wartung
    - `WITH NO DATA`
    - erzeugt keine materialisierte Sicht, sondern kopiert nur die Attributdefinitionen der Quelltabelle
  - vollautomatische Wartung durch das DBMS
    - `DATA INITIALLY DEFERRED REFRESH IMMEDIATE`
    - inkrementelle Aktualisierung nach jeder Operation auf den Daten
    - initiales Befüllen der materialisierten Sicht notwendig
      - `REFRESH TABLE <mv-name>`
    - aber: nicht mit jeder Anfrage möglich

Materialisierte Sichten - Wartung

- **Beschränkung für vollautomatische Wartung**
  - keine Referenzen auf typisierte Tabellen, materialisierte Sichten, temporäre Tabellen und Systemobjekte
  - keine Scalar Full Selects
    - liefern nur ein Ergebnistupel zurück
    - sonst enormer Wartungsaufwand
  - nur inkrementelle Wartung unterstützt, erfordert
    - keine Duplikate
    - keine Duplikateliminierung mit `SELECT DISTINCT`
    - Gruppierung erfordert `COUNT(*)`-Spalte
    - keine Outer-Joins
- **Verwendung zur Anfrageoptimierung**
  - beantwortet Anfragen mit Hilfe der materialisierten Sicht, falls möglich $\longrightarrow$ effizienter
  - `{ENABLE|DISABLE} QUERY OPTIMIZATION`

Materialisierte Sichten - Wartung

- **Weitere Möglichkeiten zur Wartung**
  - mit `DATA INITIALLY DEFERRED REFRESH DEFERRED`
  - halbautomatische Wartung
    - Wartung wird vom Nutzer angestoßen, aber vom System durchgeführt
      - `REFRESH TABLE <mv-name>`
    - inkrementelle Wartung mit Staging-Tabelle (nächste Folie)
    - `MAINTAINED BY SYSTEM`
  - durch den Nutzer
    - Wartung wird vom Nutzer durchgeführt (mittels `INSERT`, `UPDATE` und `DELETE`-Anweisungen)
    - kein `REFRESH` möglich
    - `MAINTAINED BY USER`

Staging-Tabellen

- **Staging-Tabelle (Zwischenspeichertabelle)**
  - ermöglicht inkrementelle halbautomatische Wartung
  - nimmt alle Änderungen an den unterliegenden Relationen einer materialisierten Sicht auf
    - Vermeidung von langen Sperrzeiten der MatView
  - für die Anfrage `<fullselect>` gelten die gleichen Bedingungen wie für die vollautomatische Wartung
  - besitzt 3 Spalten mehr, als materialisierte Sicht
    - `GLOBALTRANSID`: ID der Transaktion
    - `GLOBALTRANSTIME`: Zeitstempel der Transaktion
    - `OPERATIONTYPE`: Insert, Update oder Delete
  - `CREATE TABLE <st-name> FOR <mv-name> PROPAGATE IMMEDIATE`
  - Stagging-Table muss in Normalzustand versetzt werden (vorher keine Datenänderungen mehr möglich)
    - `SET INTEGRITY FOR <st-name> IMMEDIATE CHECKED`
  - Beispiel: in Übung

Zusammenfassung

- **Anfrageplan**
  - beschreibt Abarbeitung der Anfrage
  - ist Gegenstand der Anfrageoptimierung
- **Indizes**
  - erlauben effizienten selektiven Zugriff
  - aber: Wartungsaufwand
- **Materialisierte Sichten**
  - Vorberechnung von Anfragen
  - Steigerung der Effizienz
  - vom Nutzer oder vom System gewartet