Datenbanken zur Entscheidungsunterstützung - Data Warehousing

Einführung

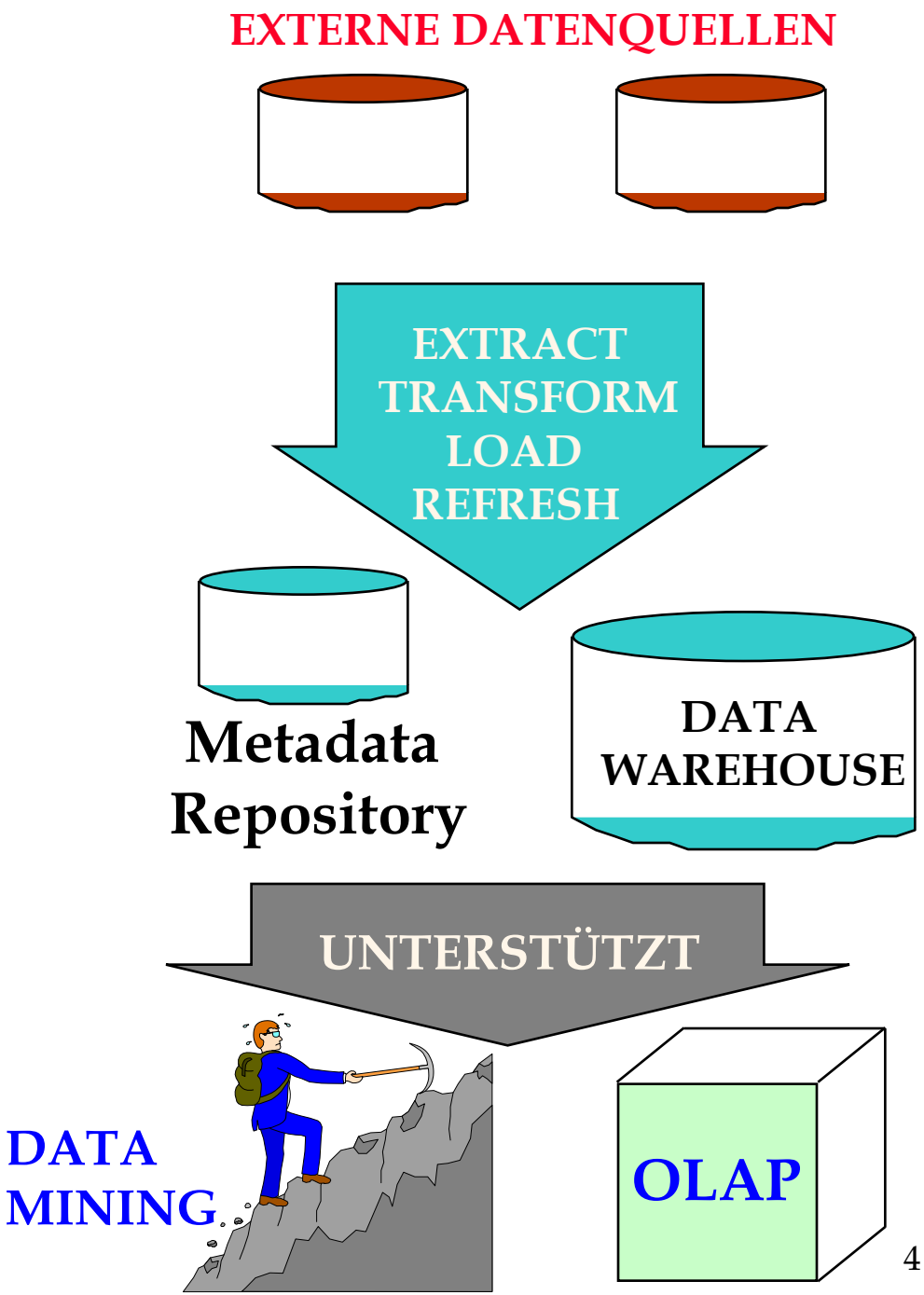
- Zunehmender Bedarf nach Analyse aktueller und historischer Daten
  - Identifizierung interessanter Patterns
  - Entscheidungsfindung (Decision Support) zur Unterstützung von Business-Strategien (z.B. Marketing)
- Schwerpunkt liegt auf komplexer, interaktiver Analyse sehr großer Datenmengen
  - Integration von Daten aus allen Teilen des Unternehmens
  - Natur der Daten ist statisch (keine Updates)
- **On-Line Analytic Processing (OLAP)**
  - Lange Lese-Transaktionen
- **On-line Transaction Processing (OLTP)**
  - Traditionelle Verarbeitung
- Integration von OLAP-Features in DBMS (Zusammenwachsen beider Technologien)
- Angebot eigenständiger Decision Support-Produkte

Drei Komplementäre Trends

- Data Warehousing: Konsolidieren von Daten aus vielen Quellen in einem großen Repository
  - Laden, periodische Synchronisation der Replikate
  - Syntaktische Integration (z.B. Datenformate)
  - Semantische Integration
- OLAP:
  - Komplexe SQL-Queries und Views
  - Queries basieren auf Spreadsheet-artigen Operationen und "mehrdimensionaler" Sicht der Daten
  - Interaktive und "online" Anfragen
- Data Mining:
  - Suche nach interessanten Trends und Abweichungen (wird hier nicht näher behandelt!)

Data Warehousing

- *Data Warehouse*
  Integrierter Datenbestand, der sich über lange Zeitperioden erstreckt, oft mit zusätzlicher Information angereichert
- Mehrere Gigabytes bis Terabytes
- Interaktive Antwortzeiten für komplexe Anfragen erwartet; ad-hoc Updates nicht üblich



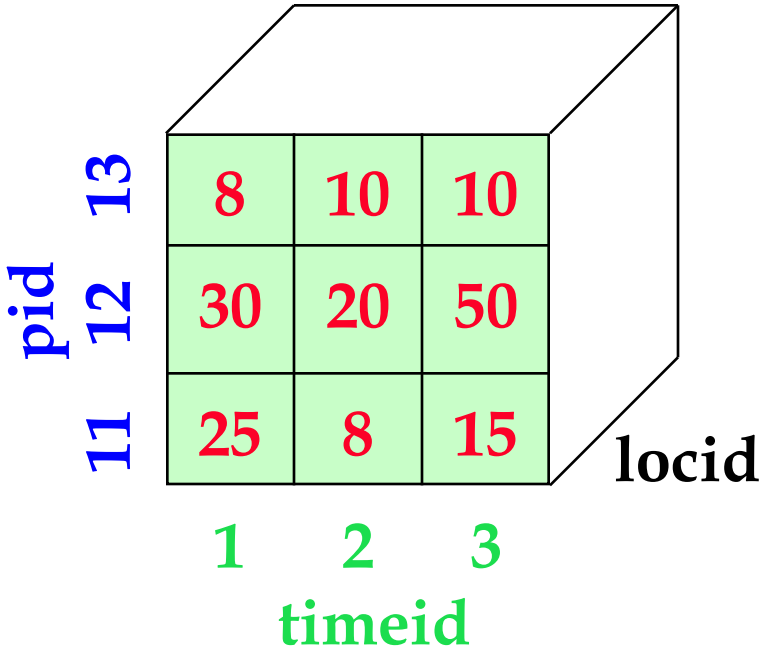
Aufgaben beim Warehousing

- Semantische Integration: Beim Bezug von Daten aus unterschiedlichen Quellen, sind alle Arten von Heterogenitäten zu beseitigen, z.B.
  - Verschiedene Währungen und Maßeinheiten
  - Unterschiede in den Schemas
  - Verschiedene Wertebereiche
- Heterogene Quellen: Zugriff auf Daten in unterschiedlichsten Formaten und Repositories
  - Möglichkeiten der Replikation ausnutzen
- Load, Refresh, Purge:
  - Daten müssen ins Warehouse geladen werden (Load)
  - Daten müssen periodisch aktualisiert werden (Refresh)
  - Veraltete Daten müssen entfernt werden (Purge)
- Metadata-Management: Verwaltung der Informationen über Daten im Warehouse (Quellen, Ladezeit, Konsistenzanforderungen etc.)

Multidimensionales Daten Model

- Sammlung von numerischen Größen, die von einer Menge von Dimensionen abhängen.
  - Z.B. Größe **Verkauf**, mit 3 Dimensionen:
    - **Produkt** (Schlüssel: pid)
    - **Ort** (locid)
    - **Zeit** (timeid).

Beispiel mit Slice locid=1



pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35

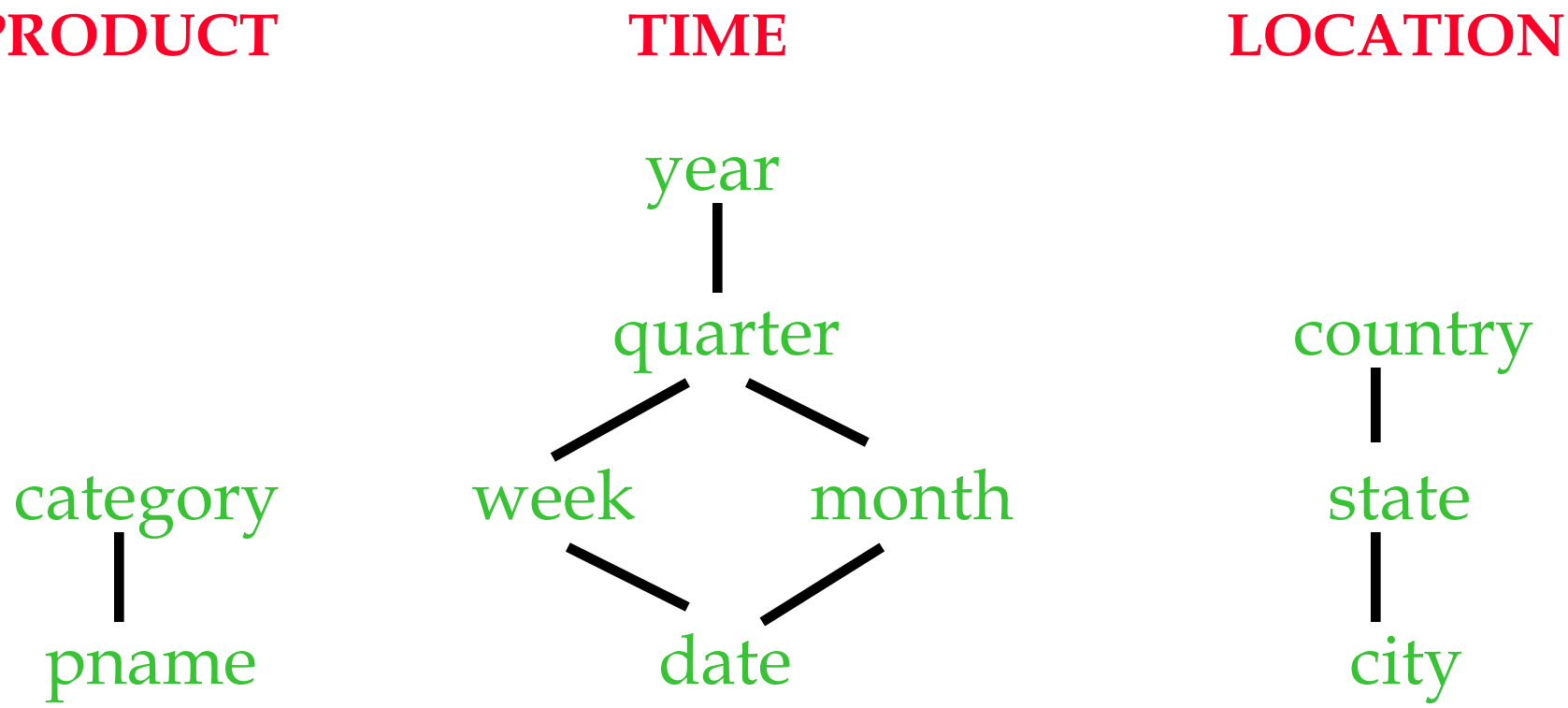
MOLAP vs. ROLAP

- MOLAP
  Physische Speicherung multidimensionaler Daten in einem (disk-residenten, persistenten) Array gespeichert
- ROLAP
  Physische Speicherung multidimensionaler Daten in Relationen
- Fakten-Tabelle
  Hauptrelation, die Dimensionen mit einer Größe verbindet
  Beispiel:
  Sales (pid, timeid, locid, sales)
- Dimensionen-Tabelle
  Assoziiert mit einer Dimension, enthält zusätzliche Attribute
  Beispiel:
  Products (pid, pname, category, price)
  Locations (locid, city, state, country)
  Times (timeid, date, week, month, quarter, year, holiday_flag)

Fakten-Tabellen sind viel kleiner als Dimensionen-Tabellen

Hierarchien in Dimensionen

- In jeder Dimension kann die Menge der Werte in Hierarchien organisiert sein



OLAP-Queries

- Beeinflußt durch SQL und durch Spreadsheets
- Häufige Operation: Aggregation einer Größe über eine oder mehrere Dimensionen
  - Bestimme den Gesamtverkauf.
  - Bestimme den Gesamtverkauf für jede Stadt oder für jedes Bundesland.
  - Finde die Top-5 Produkte, gemessen am Gesamtverkauf.
- Roll-Up: Aggregation auf verschiedenen Stufen in einer Hierarchie einer Dimension
  - Beispiel:
    Gegeben sei der Gesamtverkauf pro Stadt
    Möglicher Roll-Up: Ermittle Gesamtverkauf pro Bundesland
- Drill-Down: Umgekehrte Operation zum Roll-Up
  - z.B.: Gegeben sei Gesamtverkauf pro Bundesland, Drill-Down möglich zur Ermittlung Gesamtverkauf pro Stadt
  - Drill-Down auch in einer anderen Dimension möglich, z.B. um den Gesamtverkauf pro Produkt für jedes Bundesland zu ermitteln

OLAP-Queries

- Pivotierung: Aggregation in ausgewählten Dimensionen
  - z.B.: Pivotierung auf Ort und Zeit resultiert in einer Kreuzung (siehe Beispieltabelle)
- Slicing und Dicing: Einzelwert- und Wertbereichs-Anfragen in einer oder mehreren Dimensionen
- Zeitdimension in OLAP sehr wichtig
- Beispiele:
  - Ermittle Gesamtverkauf pro Jahr
  - Ermittle Gesamtverkauf pro Jahr für jedes Land
  - Bestimme prozentuale Veränderung des jährlichen Verkaufs für jedes Produkt

	WI	CA	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	399

Vergleich mit SQL-Queries

- Die Kreuzung von Tabellen, die durch Pivotierung entsteht, kann auch durch eine Menge von SQL-Anfragen berechnet werden:

Einträge

```
SELECT SUM(S.sales)
FROM   Sales S, Times T, Locations L
WHERE  S.timeid=T.timeid AND S.locid=L.locid
GROUP BY T.year, L.state
```

```
SELECT SUM(S.sales)
FROM   Sales S, Times T
WHERE  S.timeid=T.timeid
GROUP BY T.year
```

Untere Zeile

```
SELECT SUM(S.sales)
FROM   Sales S, Location L
WHERE  S.locid=L.locid
GROUP BY L.state
```

Rechte Spalte

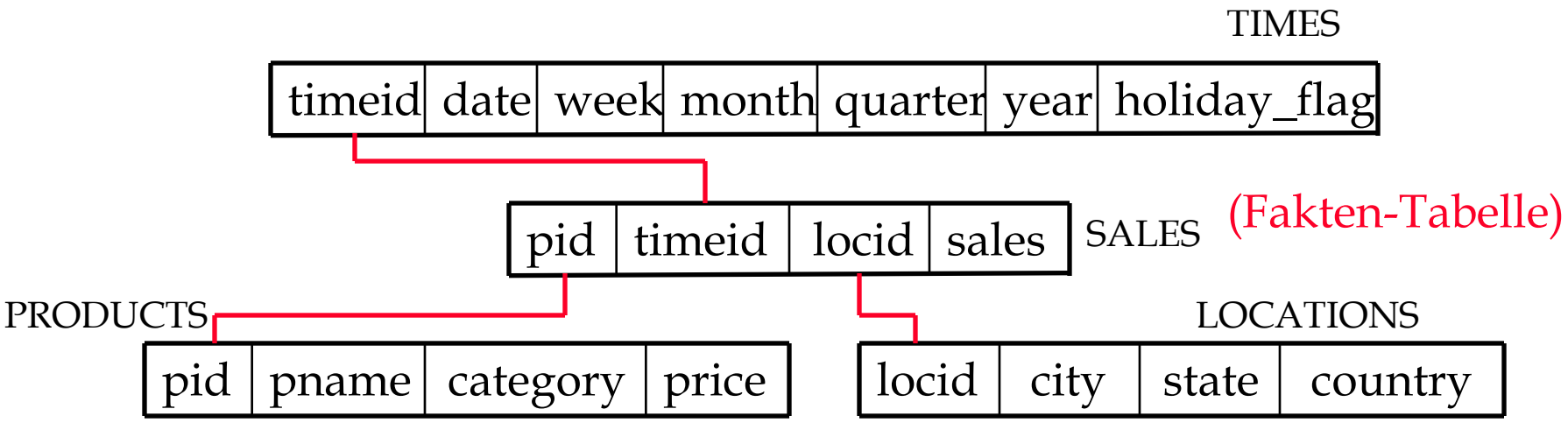
CUBE-Operator

- Verallgemeinerung des gezeigten Beispiels:
  - Bei k Dimensionen gibt es 2^k mögliche SQL GROUP BY Queries, die durch Pivotierung auf einer Teilmenge der Dimensionen erzeugt werden können
- CUBE pid, locid, timeid BY SUM Sales
  - Äquivalent zum Roll-Up von Sales auf allen 8 Teilmengen der Menge {pid, locid, timeid}
  - Jeder Roll-Up korrespondiert mit einer SQL-Query der Form:

Gegenwärtig viel Anstrengungen zur Optimierung des CUBE-Operators

```
SELECT SUM(S.sales)
FROM   Sales S
GROUP BY grouping-list
```

Datenbankentwurf für OLAP



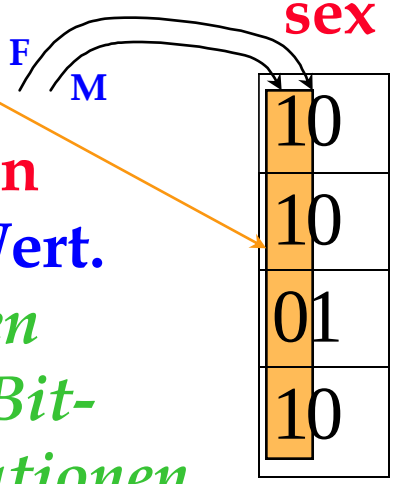
- Fakten-Tabelle in BCNF; Dimensionen-Tabelle unnormalisiert
  - Dimensionen-Tabellen sind klein
  - Updates/Inserts/Deletes in Dimensionen-Tabelle selten
  - Deshalb Anomalien weniger bedeutsam als gute Performance
- Diese Art von Schema in OLAP-Anwendungen sehr gebräuchlich, genannt *Star Schema*
- Berechnung des Joins auf diesen Relationen: *Star Join*

Implementationstechniken für OLAP

- Neue Indexierungstechniken:
  - Bitmap-Index
  - Join-Index
  - Array-Repräsentationen
  - Kompression
  - Vorberechnung von Aggregationen

Beispiel:

Bit-Vektor:
1 Bit für jeden möglichen Wert.
Viele Anfragen können über Bit-Vektor Operationen realisiert werden!



sex
10
10
01
10

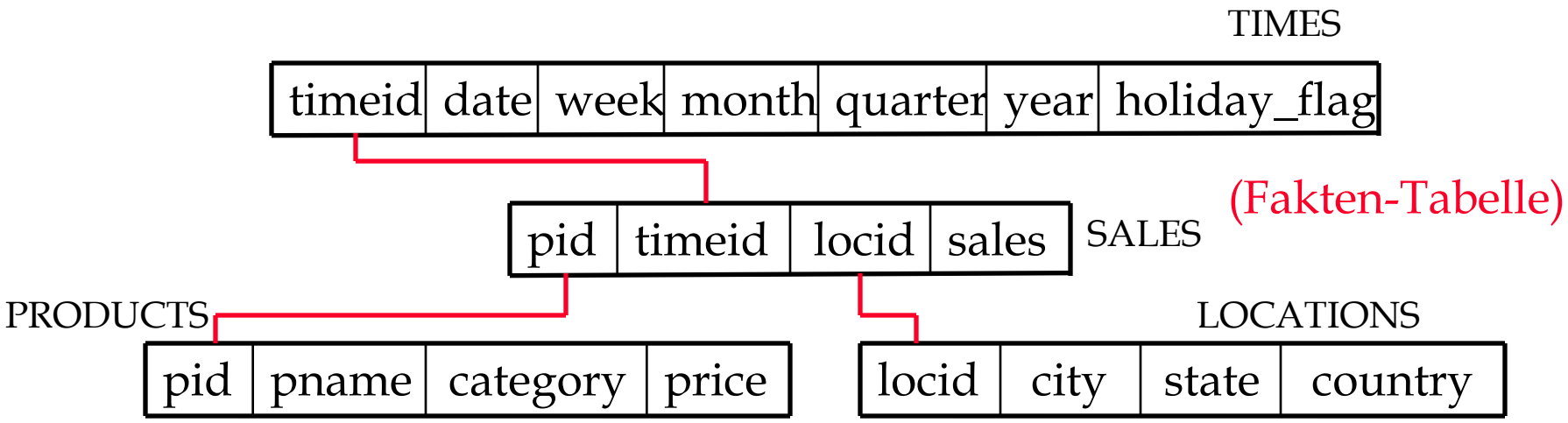
custid	name	sex	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
112	Woo	M	4

rating
00100
00001
00001
00010

Join-Index

- Betrachte den Join von Sales, Products, Times, and Locations, vielleicht noch mit zusätzlicher Selektionsbedingung (z.B. country = “USA”)
  - Ein Join-Index kann zur Beschleunigung solcher Joins konstruiert werden.
  - Der Index enthält [s,p,t,l] , wenn es Tupel gibt (mit sid) s in Sales, p in Products, t in Times and l in Locations, die die Join- (und evtl. auch Selektions-) Bedingung erfüllen.
- Problem: Anzahl der Join-Indexe kann schnell wachsen
  - Passiert, wenn mehrere Spalten in jeder Dimensions-Tabelle an Selektionen und Joins mit der Fakten-Tabelle beteiligt
  - Dieses Problem wird durch eine Variante des Join-Index adressiert:
  - Für jede Spalte mit einer zusätzlichen Selektionsbedingung (z.B. Land), baue einen Index mit [c,s], wenn ein Tupel der Dimensions-Tabelle mit Wert c in der Selektions-Spalte mit einem Tupel aus Sales mit sid s joint
  - Wenn Indexe Bitmaps sind: Bitmapped Join Index.

Bitmapped Join-Index



- Betrachte eine Query mit Bedingung price=10 and country="USA":
  - Angenommen, Tupel (mit sid) s in Sales joint mit einem Tupel p mit Preis =10 und einem Tupel l mit country ="USA". Es gibt zwei Join-Indexe: 1.) Enthält [10,s]. 2.) Enthält [USA,s].
- Durchschnitt dieser Indexe verrät, welche Sales-Tupel zum Join gehören und die gegebene Selektion erfüllen

Views und Decision Support

- OLAP-Anfragen sind typischerweise Aggregationen
  - Vorberechnung (*Precomputation*) ist wesentlich für interaktive Antwortzeit
  - Der CUBE ist praktisch eine Sammlung von Aggregat-Anfragen, mit Vorberechnung als wichtigem Lösungsansatz
  - Zu lösendes Problem: Was läßt sich am besten vorberechnen mit einem begrenzten Speicherplatzumfang, um vorberechnete Ergebnisse zu speichern?
- Data Warehouse kann als eine Sammlung von asynchron replizierten Tabellen und periodisch aktualisierten Views angesehen werden
  - Führt zu einem großen Interesse am Problem der View Maintenance
  - View Maintenance: Konsistenzkontrolle zwischen Sichten und den zugrunde liegenden Basistabellen

View Modification (Berechnung On Demand)

View

```
CREATE VIEW RegionalSales(category,sales,state)
        AS SELECT P.category, S.sales, L.state
           FROM Products P, Sales S, Locations L
           WHERE P.pid=S.pid AND S.locid=L.locid
```

Query

```
SELECT R.category, R.state, SUM(R.sales)
FROM RegionalSales AS R GROUP BY R.category, R.state
```

Modifizierte Query

```
SELECT R.category, R.state, SUM(R.sales)
FROM (SELECT P.category, S.sales, L.state
        FROM Products P, Sales S, Locations L
        WHERE P.pid=S.pid AND S.locid=L.locid) AS R
GROUP BY R.category, R.state
```

Schachtelung von FROM in SQL:1999 möglich

View Materialization (Vorberechnung)

- Angenommen, wir berechnen RegionalSales und speichern dies mit einem geclusterten B+ Baum-Index auf [category,state,sales].
  - Somit kann die eben gestellte Anfrage durch einen Scan nur auf dem Index bearbeitet werden

```
SELECT R.state, SUM(R.sales)
FROM RegionalSales R
WHERE R.category="Laptop"
GROUP BY R.state
```

Index auf vorberechneter View sehr hilfreich!

```
SELECT R.state, SUM(R.sales)
FROM RegionalSales R
WHERE R. state="Wisconsin"
GROUP BY R.category
```

Index weniger sinnvoll (Scan auf der gesamten Blatt-Ebene erforderlich)

Fragen bei Materialisierung von Sichten

- Welche Sichten sollten materialisiert werden, welche Indexe sollten auf den vorberechneten Ergebnissen gebaut werden?
- Mit einer Anfrage und einer Menge materialisierter Sichten: Können wir die materialisierten Sichten benutzen, um die Anfrage zu beantworten?
- Wie häufig sollten wir materialisierte Sichten aktualisieren (Refresh), um sie mit den zugrundeliegenden Tabellen konsistent zu machen?
  - Probleme beim inkrementellen Refresh?
    - Refresh einfach bei neu hinzugekommenen Tupeln in der Basisrelation, problematisch bei gelöschten Tupeln in der BR
- Unterschiedliche View Maintenance Policies möglich:
  - *Lazy*: Sicht wird aktualisiert, wenn zugehörige Anfrage aufgerufen wird (falls nicht schon Konsistenz vorhanden ist)
  - *Periodisch*: Materialisierte Sichten in festen Zeitabständen aktualisiert (*Snapshots*)
  - *Forced*: Aktualisierung nach einer bestimmten Zahl von Änderungen in der Basistabelle

Interaktive Queries: Alternative zu View Materialization

- Top N Queries: Finde die ersten N Tupel des Anfrage-Ergebnisses

  Beispiel: Finden die 10 billigsten Autos!
  Wäre gut, wenn die DB die Kostenberechnung für alle Autos vermeiden könnte vor dem Sortieren, um die billigsten 10 herauszufinden

  – Idee: Schätze einen Kosten-Grenzwert c, so daß die 10 billigsten Autos allesamt weniger als c kosten, aber auch nicht viel mehr weniger. Füge dann die Selektionsbedingung cost<c hinzu und führe die Anfrage aus.
    - Falls Schätzwert richtig, kann die Berechnung für Autos, die mehr als c kosten, vermieden werden
    - Bei falscher Schätzung muß die Selektion zurückgesetzt und die Original-Anfrage erneut berechnet werden

Top N Queries

```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
ORDER BY S.sales DESC
OPTIMIZE FOR 10 ROWS
```

```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
        AND S.sales > c
ORDER BY S.sales DESC
```

- OPTIMIZE FOR Konstrukt ist nicht Bestandteil von SQL:1999, wird aber in kommerziellen DBMS angeboten (DB2, Oracle)
- Cut-off Wert c wird vom Optimierer gewählt

Interaktive Queries: Online-Aggregation

- Online-Aggregation: Betrachte eine Aggregat-Query wie z.B. “Bestimme den Durchschnittsverkauf pro Bundesland“:

```
SELECT    L.state, AVG(S.sales)
FROM      Sales S, Location L
WHERE     S.locid=L.locid
GROUP BY  L.state
```

- Können wir dem Benutzer einige Informationen liefern vor der genauen Berechnung des Durchschnitts für alle Bundesländer?
  - Wir können den aktuellen “laufenden Durchschnitt” für jedes Bundesland zeigen bei Voranschreiten der Berechnung
  - Noch besser: Nutzung von statistischen Techniken und Beispiel-Tupeln zur Aggregation anstelle eines einfachen Durchscannens der aggregierten Tabelle
  - Definition von Grenzen wie z.B. “Durchschnitt für Wisconsin ist 2000±102 mit 95% Wahrscheinlichkeit
    - Wir sollten auch nicht-blockierende Algorithmen verwenden (also z.B. keinen Merge Sort).
    - Blockieren: Keine Ausgabe von Tupeln, bevor nicht alle Eingabe-Tupel verarbeitet sind!

Zusammenfassung

- Decision Support ist schnell wachsendes Teilgebiet von Datenbanken
- Beinhaltet die Erzeugung von Data Warehouses = große konsolidierte Data Repositories
- Warehouses verwenden komplizierte Analyse-Techniken:
  - komplexe SQL-Anfragen
  - “multidimensionale” OLAP-Anfragen (beeinflußt durch SQL und Spreadsheets)
- Neue Techniken erforderlich für:
  - Datenbank-Entwurf
  - Indexierung
  - View Maintenance
  - Interaktive Queries