On-Line Analytical Processing

Rasmus Pagh

Reading: RG25, [WOS04, sec. 1+2]

Today's lecture

- Multi-dimensional data and OLAP.
- Bitmap indexing.
- Pre-aggregation and materialized views.

Background

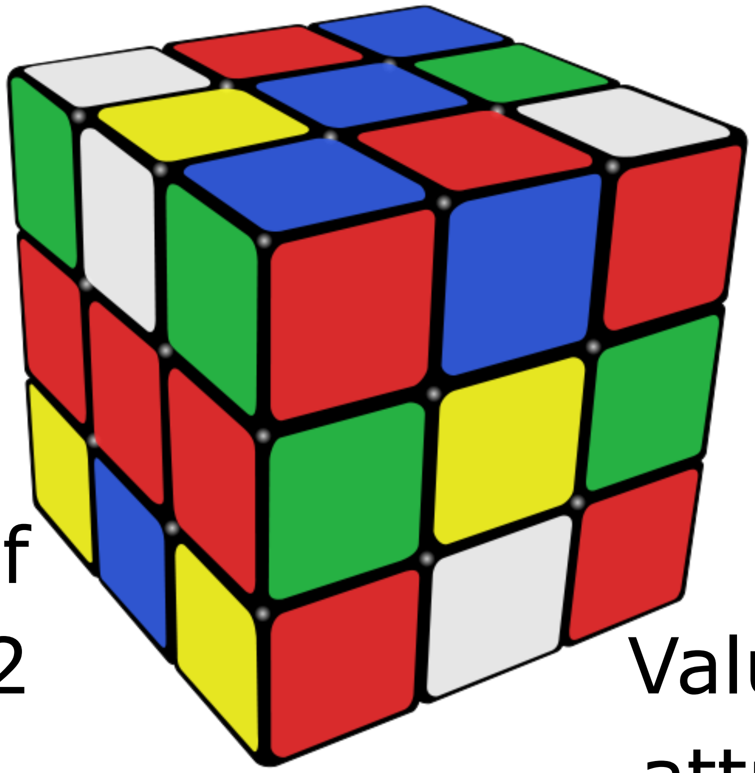
- Data = gold mine!
  - Data can identify useful patterns that can be used to form a proper business strategy.
- Two main approaches:
  - Data mining: Automated ”mining” of patterns. (In two weeks.)
  - OLAP: Interactive investigation of data. (This lecture.)
- OLAP = On-Line Analytic Processing
  OLTP = On-Line Transaction Processing

Data Warehouse

- Data warehouse: Integrated collection of data used in support of decision-making processes.
- Typical characteristics:
  - Records data over time (not just “current”).
  - *Snapshot* of the operational data.
  - Can be organized as collection of ”data marts”.
- Challenges: Data integration, cleaning…
- Much more about this next week!

Multi-dimensional data model

- Tuples with d attributes can be viewed as points in a d-dimensional ”cube”.



Value of attrib. 1

Value of attrib. 2

Value of attrib. 3

Multi-dimensional data model

- **Example**: (BIDD,'John Doe',12) can be viewed as the point with coordinates
  - BIDD
  - John Doe
  - 12

  along the *dimensions*
  - Course
  - Name
  - Grade
- Dimensions often have *granularities* (e.g. study line BSWU, class of 2013)

Terminology

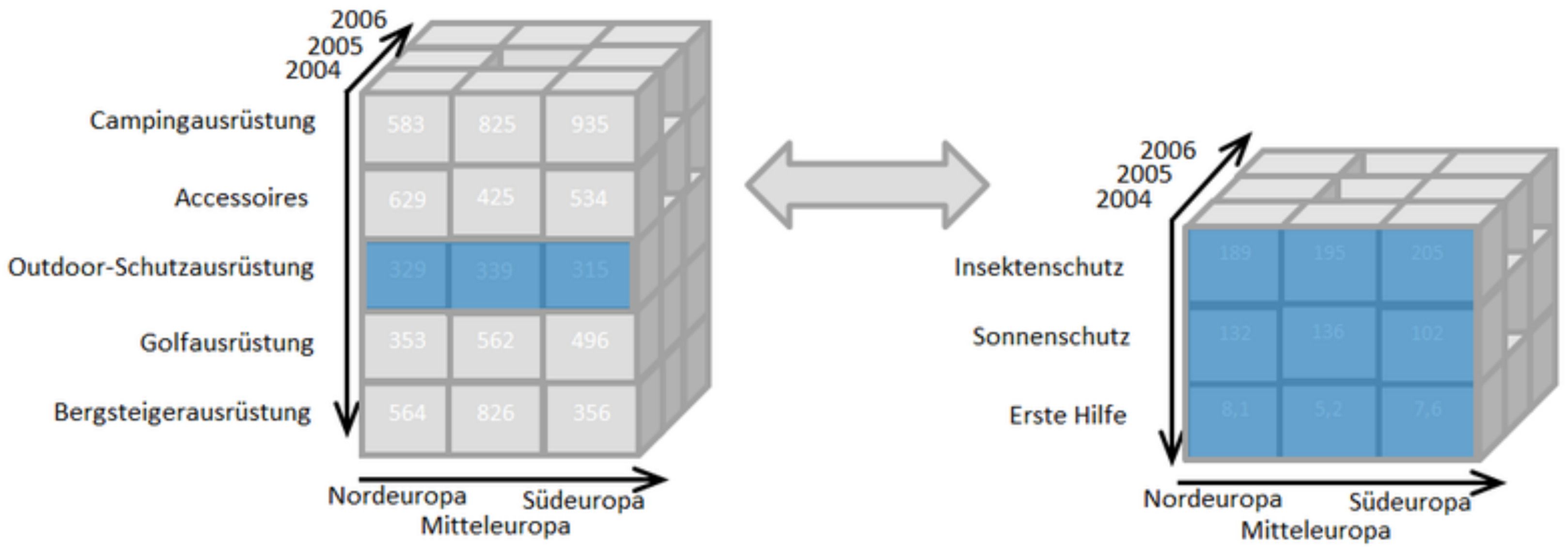
- **Fact** = point in the cube
- **Measure** = number associated with facts, can be aggregated
  - Examples: Sales price, customer debit, time to next click.
- **Dimension** = feature that can be used to select subset of facts.
  - Examples: ID of sales person, name of shop, state of shop,…

Typical OLAP query

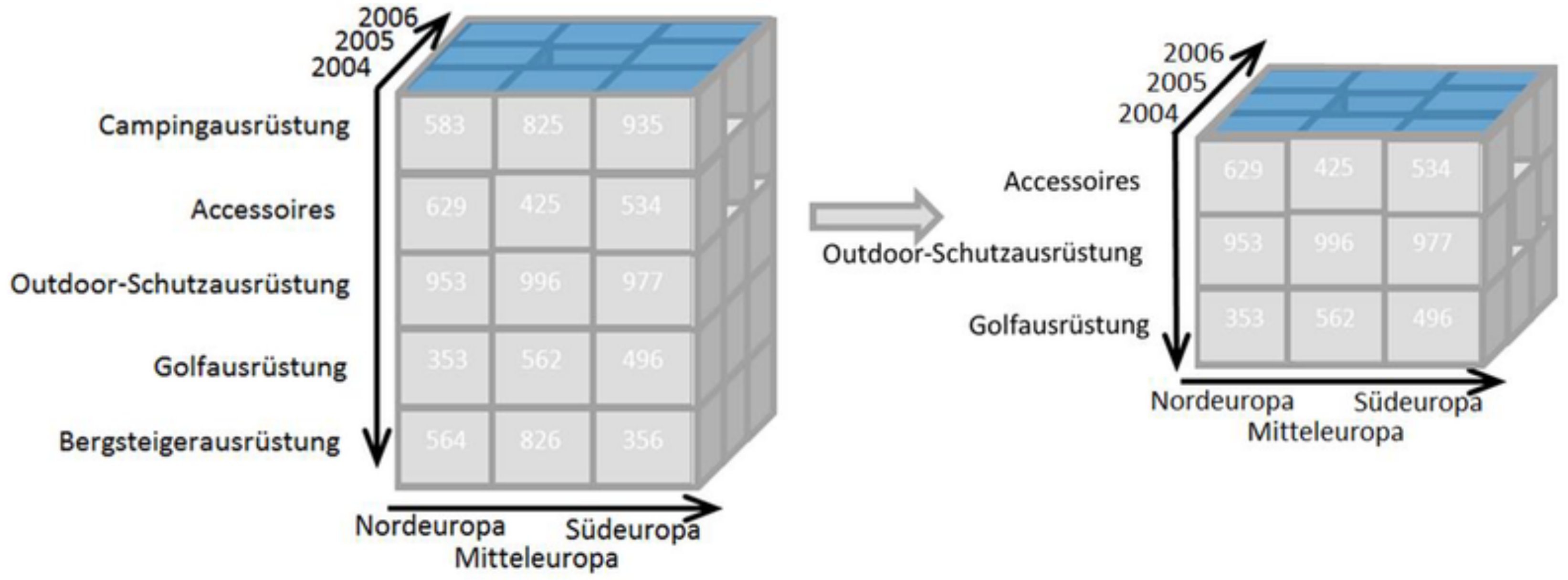
- Selection on one or more dimensions
  - e.g. select sales of one customer group.
- Grouping by one or more dimensions
  - e.g., group sales by quarter.
- Aggregation over each group
  - e.g. total sales revenue.

- Terminology for ”navigating” data by modifying the query:
  - Roll-up, drill-down, slicing, dicing, pivoting.

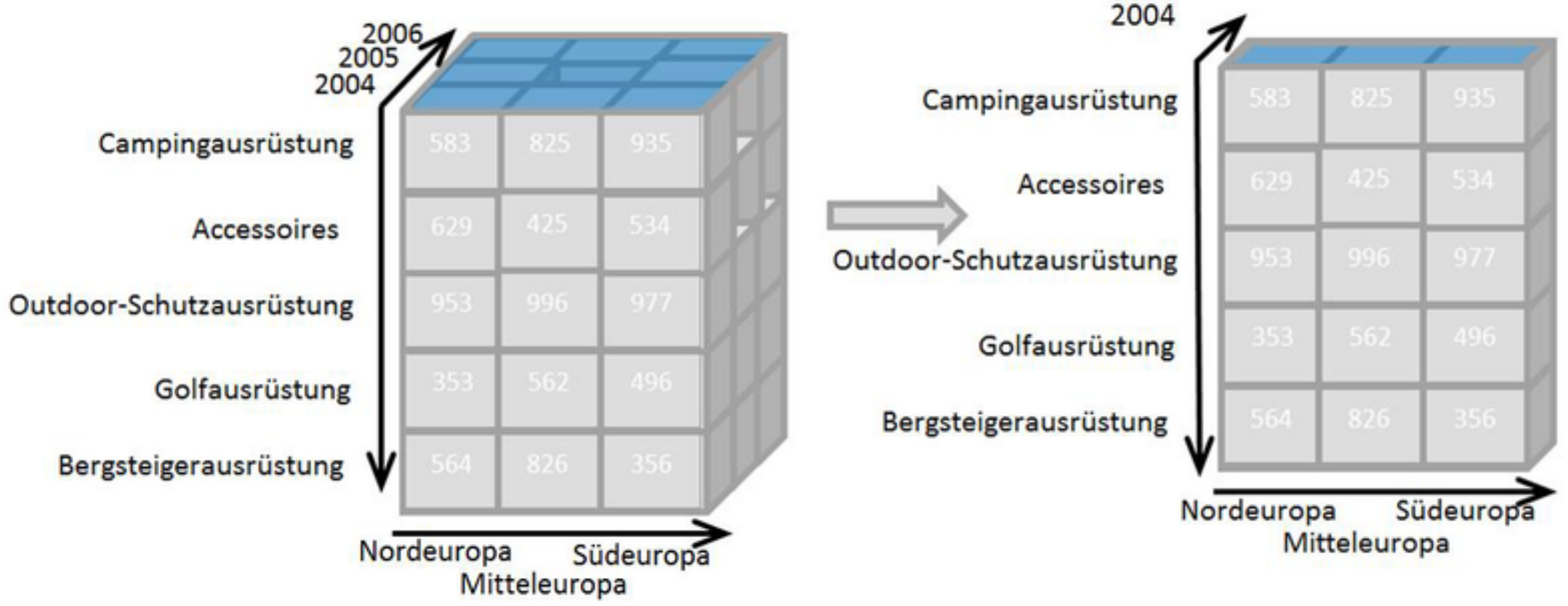
Drill-down / roll-up



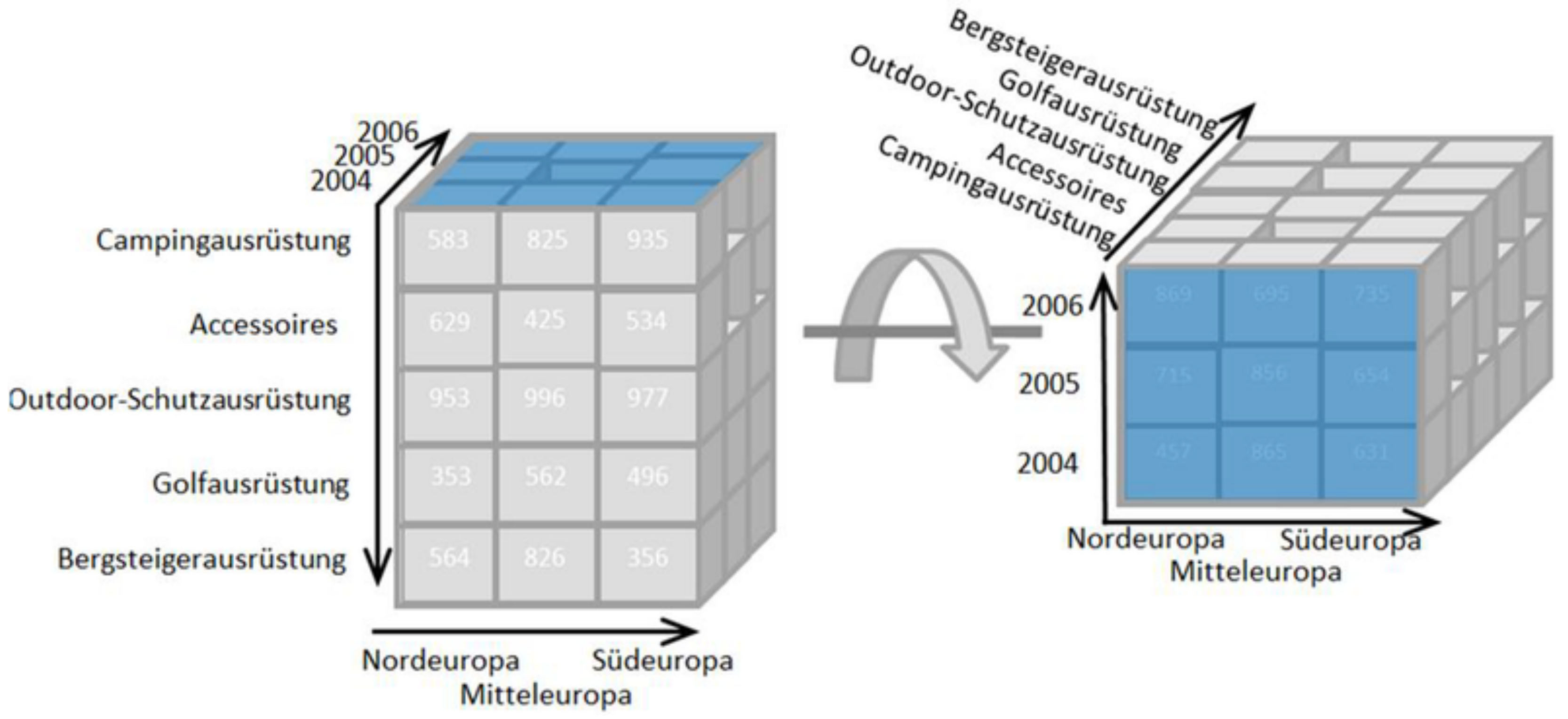
Dicing


2006
2005
2004
Campingausrüstung
Accessoires
Outdoor-Schutzausrüstung
Golfausrüstung
Bergsteigerausrüstung
583
825
935
629
425
534
953
996
977
353
562
496
564
826
356
Nordeuropa
Mitteleuropa
Südeuropa
2006
2005
2004
Accessoires
Outdoor-Schutzausrüstung
Golfausrüstung
629
425
534
953
996
977
353
562
496
Nordeuropa
Mitteleuropa
Südeuropa

Slicing

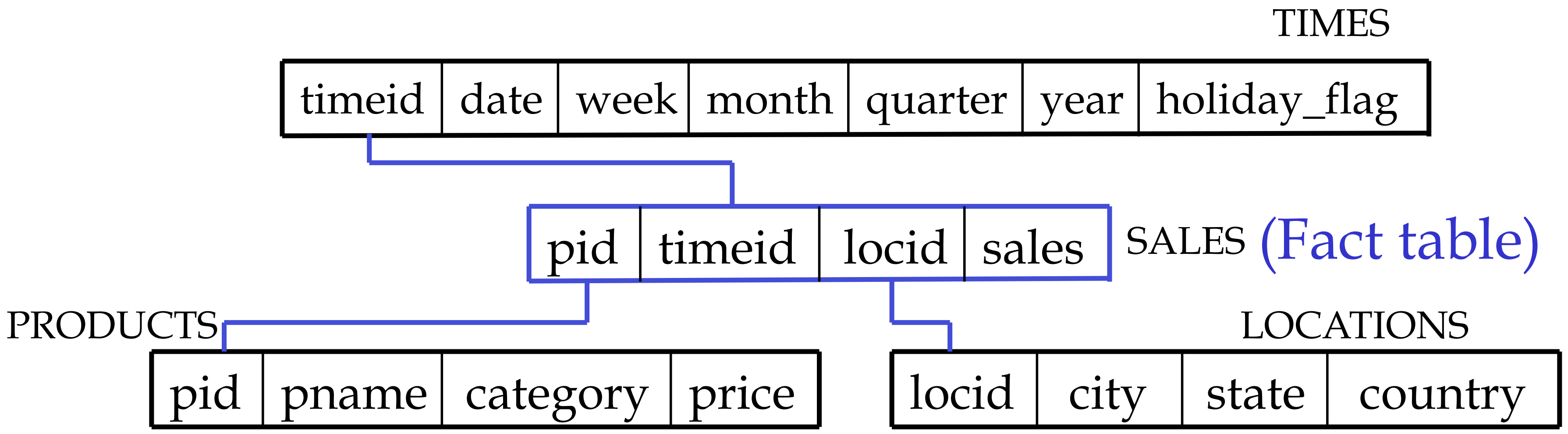

2006
2005
2004
Campingausrüstung
Accessoires
Outdoor-Schutzausrüstung
Golfausrüstung
Bergsteigerausrüstung
583 825 935
629 425 534
953 996 977
353 562 496
564 826 356
Nordeuropa
Mitteleuropa
Südeuropa
2004
Campingausrüstung
Accessoires
Outdoor-Schutzausrüstung
Golfausrüstung
Bergsteigerausrüstung
583 825 935
629 425 534
953 996 977
353 562 496
564 826 356
Nordeuropa
Mitteleuropa
Südeuropa

Pivoting


2006
2005
2004
Campingausrüstung
Accessoires
Outdoor-Schutzausrüstung
Golfausrüstung
Bergsteigerausrüstung
583
825
935
629
425
534
953
996
977
353
562
496
564
826
356
Nordeuropa
Südeuropa
Mitteleuropa
Bergsteigerausrüstung
Golfausrüstung
Outdoor-Schutzausrüstung
Accessoires
Campingausrüstung
2006
2005
2004
869
695
735
715
856
654
457
865
631
Nordeuropa
Südeuropa
Mitteleuropa

Relational set-up of OLAP

- **Star schema:** A ”fact table”, plus a number of ”dimension tables” whose keys are foreign keys of the fact table.
- Example from RG’s slides:


TIMES
timeid | date | week | month | quarter | year | holiday_flag
pid | timeid | locid | sales
SALES (Fact table)
PRODUCTS
pid | pname | category | price
LOCATIONS
locid | city | state | country

Normalization in OLAP

- To limit redundancy, the fact table should not have any avoidable FDs, e.g. `salespersonID → shop state`
- The information corresponding to FDs are stored in dimension tables.
- Normalize dimension tables?
  Usually not.
  - Unnormalized dim. tables: *star schema*.
  - Normalized dim. tables: *snowflake schema*.

Sample OLAP query (in SQL)

```
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
```

Get fine-grained aggregate data

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

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

Get dimension ”coordinates” +aggregates

Special SQL OLAP syntax

- Some DBMSs support special syntax for computing many aggregates.
- `SELECT a,b,… GROUP BY CUBE (a,b)` produces a table that contains the aggregate grouped by all subets of `{a,b}`.
- Essentially the union of
  - `SELECT a,b … GROUP BY a,b`
  - `SELECT a,null,… GROUP BY a`
  - `SELECT null,b … GROUP BY b`
  - `SELECT null,null, …`

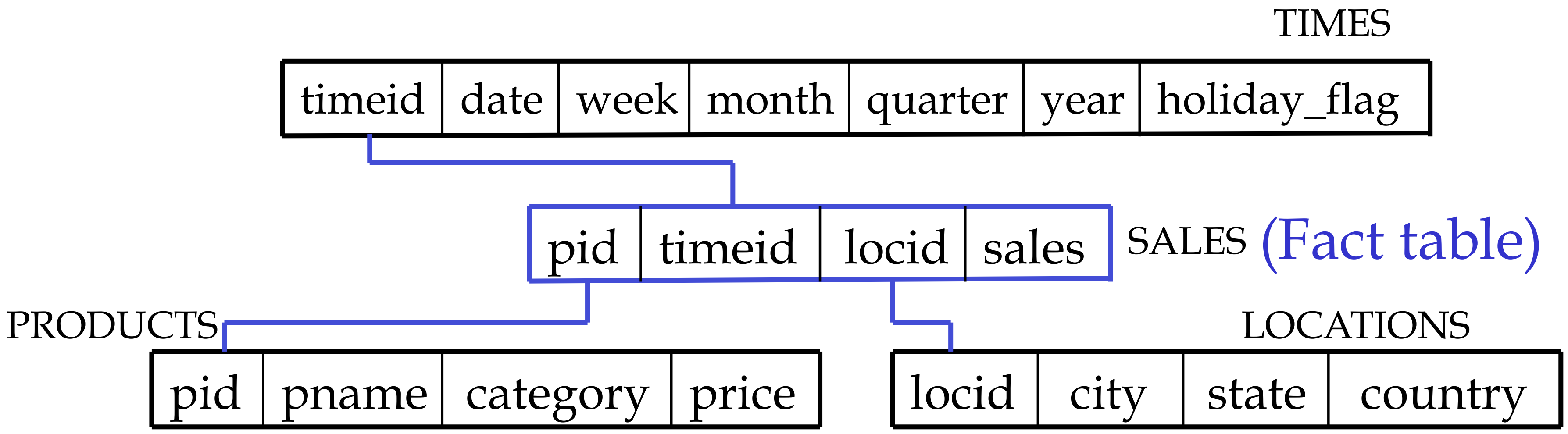
OLAP challenge

- Support queries fast enough to allow interactive “on-line” analytics.

- Two main techniques for achieving this:
  - Indexing
  - Pre-aggregation

Problem session

- How would you efficiently answer the query: *"Find the total sales of raincoats for each location last year."*
- You may assume suitable indexes, and that *timeids* are ordered by time.



TIMES
timeid | date | week | month | quarter | year | holiday_flag
pid | timeid | locid | sales
SALES (Fact table)
PRODUCTS
pid | pname | category | price
LOCATIONS
locid | city | state | country

Indexing low cardinality attributes

- Suppose there are only 4 different years in our previous example.
- Then we may represent the locations of N fact table tuples using only 2N bits.
- However, an (unclustered) index on location seems to require at least N log N bits.
- Can we get by reading less data?

Basic bitmap index

- For each possible value (of ”year”) and each tuple, store a bit that is 1 iff the tuple contains the given value.
- Store these bits **ordered by column** (with no RID).
- The bitmap index works as a **join index** that can be used to compute joins using RID intersection.

Y1?	Y2?	Y3?	Y4?	RID
0	0	1	0	1
1	0	0	0	2
1	0	0	0	3
0	1	0	0	4
0	0	0	1	5

Gain of bitmap indexes

- Assume bottleneck is reading data.
- How much can **at most** be gained by using bitmap indexes to do a star join (with a selection on each dimension table), compared to using a B-tree?
  - Theoretically 1 bit/tuple vs log N bits/tuple.
  - Typically 1 bit/tuple vs 32 bits/tuple.
- Main case where there is no gain:
  - A single dimension is very selective.
  - (Usually only the case for high cardinality attributes.)

Compressed bitmap indexes

- If there are many possible values for an attribute (it has ”high cardinality”), basic bitmap indexing is not space efficient (nor time efficient).
- **Observation:** A column will have few 1s, on average. It should be possible to ”compress” long sequences of 0s.
- **How to compress?** Usual compression algorithms consume too much computation time. Need simpler approach.

Word-aligned hybrid (WAH) coding

[WOS04]

- In a nutshell:
  - Split the bitmap B into pieces of 31 bits.
  - A 32-bit word in the encoding contains one of the following, depending on the value of its first bit:
    - A number specifying the length of an interval of bits where all bits of B are zeros.
    - A piece of B (31 bits).
  - The conjunction (”AND”) or disjunction (”OR”) of two compressed bitmaps can be computed in linear time (O(1) ops/word).

WAH analysis

- Let N be the number of rows of the indexed relation, and c the cardinality of the indexed attribute.
- At most N WAH words will encode a piece of the bitmap.
- Reasonable assumption:
  - All (or most) gaps between consecutive 1s can be encoded using 31 bits.
  - Thus, at most N+c gaps.
- Total space usage: 2N+c words.
- Compares favorably to B-trees.

Bitmap index support

- MySQL: Not supported – analytics is not an important market segment.
  - However, supports index intersection.
- Oracle:

  ```
  CREATE BITMAP INDEX ON sales(pid)
  ```

  - Internal representation is another compressed bitmap format (BBC).
- Some non-relational systems like FastBit work *entirely* with compressed bitmap representations of sets.

Pre-aggregation

- **Idea**: Precompute certain aggregates that can be used to answer queries quickly.
- **Example**:
  - Precompute the number of sales and average sales price for each dealer.
  - Allows us to quickly find e.g. the number of sales for each country.
- **Wanted**:
  Database support for keeping track of suitable preaggregated values.

Materialized views

- Recall: A *view* is similar to a macro.

```
CREATE VIEW MyView AS
SELECT *
FROM     Sales S, Times T, Locations L
WHERE    S.timeid=T.timeid AND
         S.locid=L.locid
```

- In contrast, a **materialized view** *physically* stores the query result.
  - Can be updated automatically
  - Can be indexed!

Preaggregation with materialized view

1. Materialized view is created:

```
CREATE MATERIALIZED VIEW SalaryByLocation AS
SELECT location_id, country_id, SUM(salary) AS s
FROM Employees NATURAL JOIN Departments
   NATURAL JOIN Locations
GROUP BY location_id, country_id
```

2. Materialized view is used:

```
SELECT country_id, SUM(salary) AS salary
FROM SalaryByLocation
GROUP BY country_id
```

Factor 10 faster than direct query on Oracle XEs example DB.

Automatically using mat. views

- Suppose a user does not know about the materialized view and writes directly

```
SELECT location_id, country_id, SUM(salary) AS s
FROM Employees NATURAL JOIN Departments
    NATURAL JOIN Locations
GROUP BY country_id
```

- A smart (expensive) DBMS will realize that this can be rewritten to a query on the materialized view.
- Rewrite capability is a key technique in relational OLAP systems.

Materialized views in MySQL

- Not supported...
- However, can be emulated by creating tables, i.e., `CREATE MATERIALIZED VIEW` becomes `CREATE TABLE`.
- Main differences:
  - No automatic updates.
  - No automatic query rewrite.

”Refreshing” a materialized view

- Any change to the underlying tables may give rise to a change in the materialized view. There are at least three options:
  - Update for every change (”`ON COMMIT`”)
  - Update only on request (”`ON DEMAND`”)
  - Update when the view is accessed (”lazy”)
- RG describes a way of refreshing where recomputing the defining query is often not necessary (”`FAST`”).

On-line aggregation

- For aggregates like sums and averages, the result on a sample can be used to estimate the result on all data.
  - Same principle as used in opinion polls!
- Can give statistical guarantees on an answer, e.g. ”Answer is 3200±180 with 95% probability”.
  - The longer the query runs, the smaller the uncertainty gets.
  - Possibly ok to terminate before precise answer is known.
- Research area: Do on **stream** of data.

Next steps

- Did you fill out the course evaluation?
- Exercises:
  - Work on hand-in 3 (individual), and/or
  - Work on hand-in 4 (groups)
- Next week: Guest lecture by Kennie Nybo Pontoppidan from Rehfeld.
  - **Abstract**: How do you design a data warehouse for the handling of management information in seven subject areas for all 98 municipalities in Denmark? Especially when there exists up to five different IT systems per field, and that the municipalities can use the systems differently. Add to this a requirement to change the business rules for each municipality, and the ability to report on one municipality, while data for another municipality is updated. The size of the data warehouse will be in the order of 8TB in production.