OLAP

INF 551

Wensheng Wu

Why OLAP?

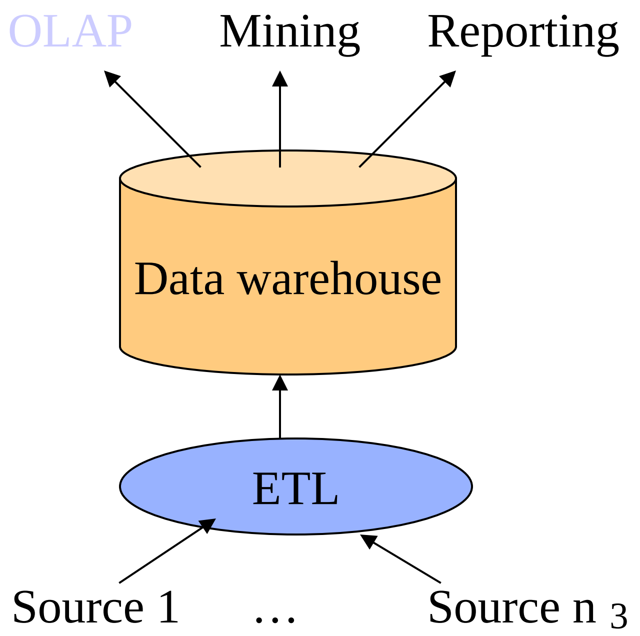
- Traditionally, database was designed for OLTP
  - E.g., banking transaction, order processing, etc.
  - Process operational data
  - Have predictable light workload: read/write only a few records/tuples
  - Require frequent update, insert, & delete operations

- Increasing need to support decision making & data analysis
  - Requires ad hoc complex queries over a large volume of data
  - Need comprehensive exploration: identify trends, create summaries, etc.
  - Analyze current + historical data
  - Involve mostly read operations

OLAP and Data Warehouses

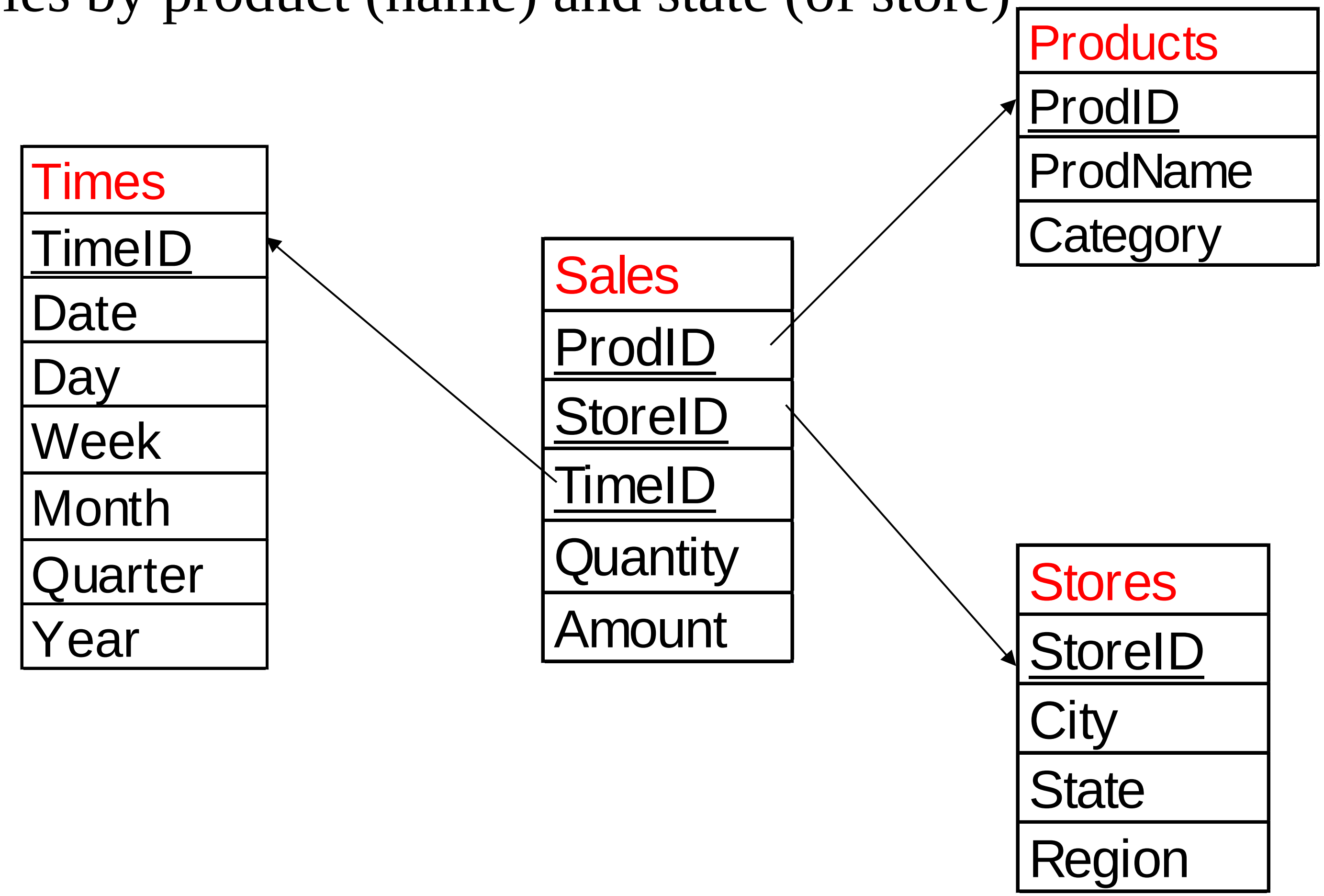
- Data warehouse is a perfect data store for OLAP
  - Having a local copy of data speeds up the analytical queries
  - Avoid competing with OLTP systems for resources
  - Make special organizations possible to support OLAP

- Disadvantages of warehousing:
  - Data might be delayed, e.g., 24 hours
  - May not be suitable for real-time analytics



OLAP on Sales Data

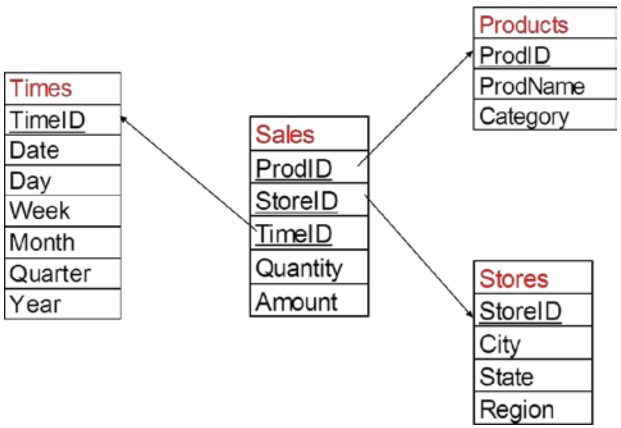
- Examine sales after 2009-1-1 to see how average sales amount varies by product (name) and state (of store)



Example OLAP Query

- Examine sales after 2009-1-1 to see how average sales amount varies by product and state

```
select    prodname, l.state, avg(s.amount)
from      sales s, products p, times t, stores l
where     s.prodid = p.prodid and s.timeid = t.timeid
          and s.storeid = l.storeid
          and t.date >= '2009-01-01'
group by prodname, l.state
order by prodname, l.state, avg(s.amount)
```

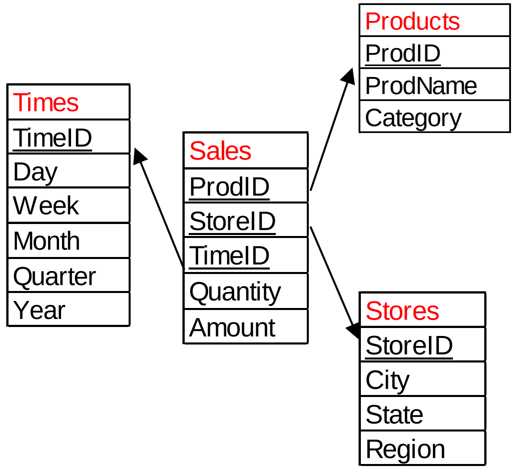


ProdName	State	AvgSales
Droid	NC	9.5
iPhone 3GS	NY	10.66667
Macbook Pro	NY	8
Nexus One	CA	8

OLAP vs. OLTP Queries

- OLAP query requires aggregation over many data in the warehouse
  - E.g., previous query requires all sales data from 1/1/2009
- Contrast to simple lookup in OLTP
  - E.g., find # of units of Nexus One sold on Feb 5, 2009

```
select   s.quantity
from     sales s, products p, times t
where    s.prodid = p.prodid and s.timeid = t.timeid
         and t.date = '2009-02-05'
         and p.prodname = 'Nexus One'
```

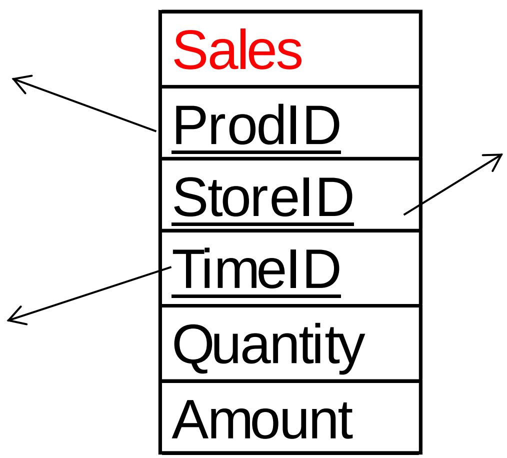


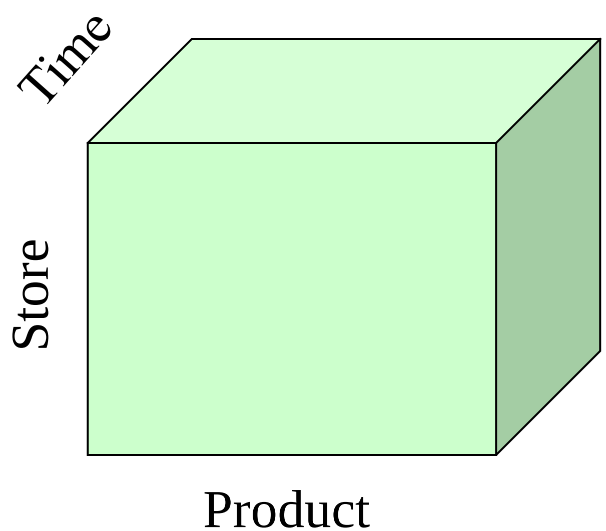
Roadmap

- Multidimensional data model
- Implementation: ROLAP vs. MOLAP
- Language constructs
  - Cube, rollup operators
  - Window functions

A Multidimensional View of OLAP Data

- Data examined by OLAP may be stored in
  - A fact table: storing facts/measures, e.g., units sold, sales amount, etc.
  - A set of dimension tables: each is a perspective for examining the facts
- Alternatively, viewed as a multidimensional (base) cuboid
  - Cell in the cuboid represents fact: a multidimensional data point



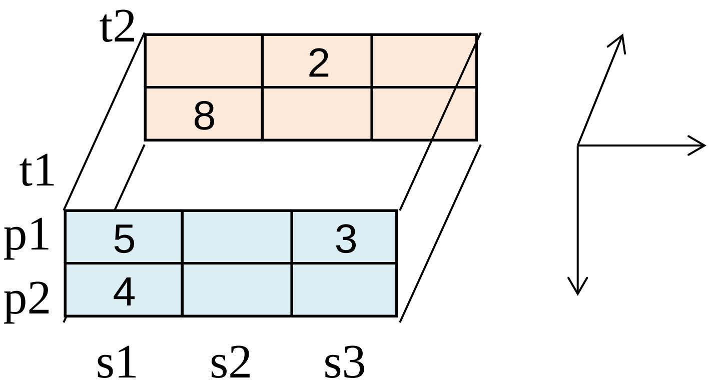


Example of Base Cuboid

- Values of attributes in the fact table become labels for the positions in the cube dimensions

ProdID	StoreID	TimeID	Quantity
p1	s1	t1	5
p1	s3	t1	3
p2	s1	t1	4
p1	s2	t2	2
p2	s1	t2	8

Fact table view



Multidimensional view

From Base Cuboid to Data Cube

- Data cube contains a set of cuboids, each storing the aggregate values in a different subset of dimensions

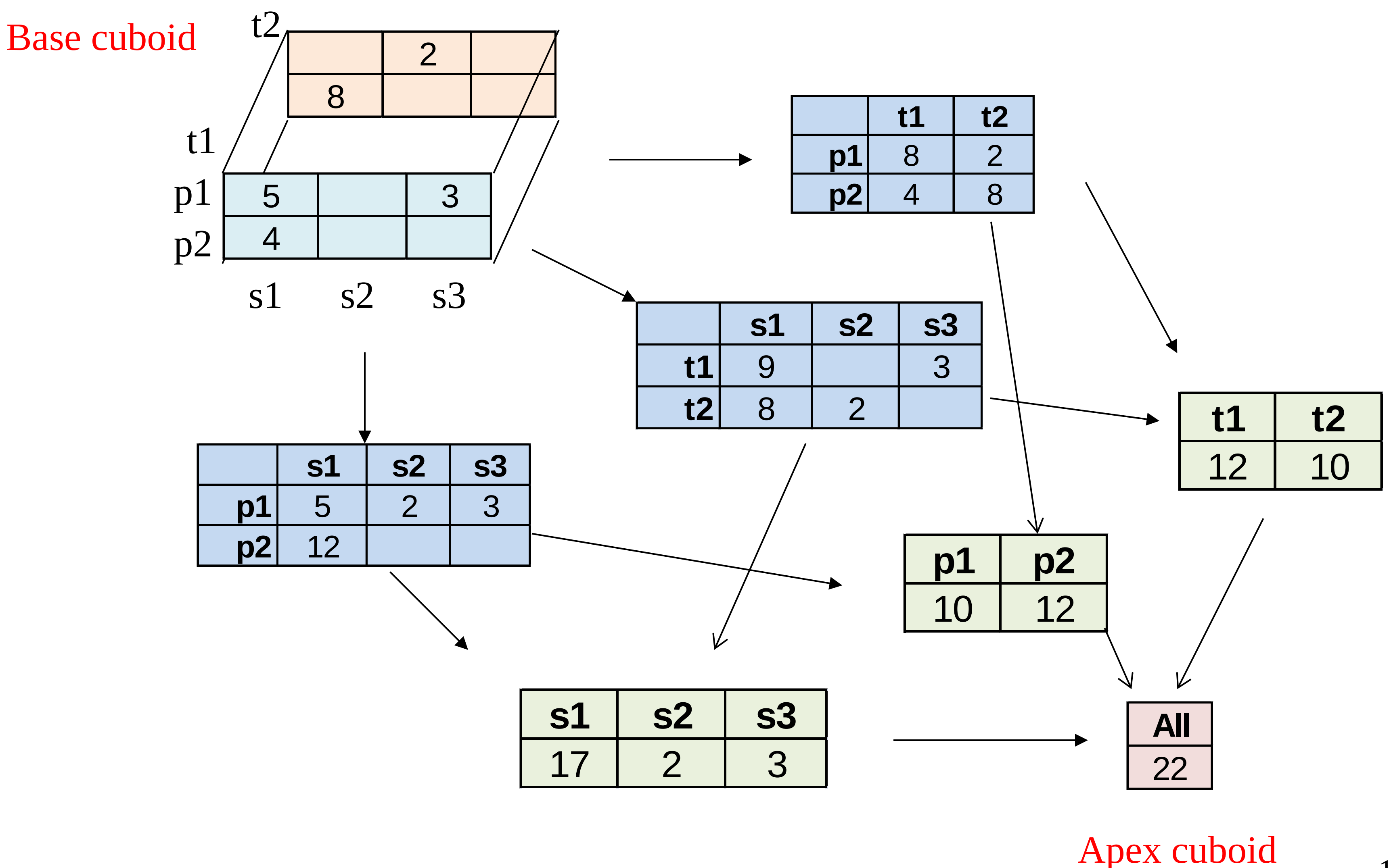
- E.g., a 3-dimensional sales data cube contains 8 cuboids:
  - Product, store, time (base cuboid, 3-dimensional)
  - Product, store
  - Store, time
  - Product, time
  - Product
  - Store
  - Time
  - All (0-d cuboid, apex cuboid)

2-d

1-d

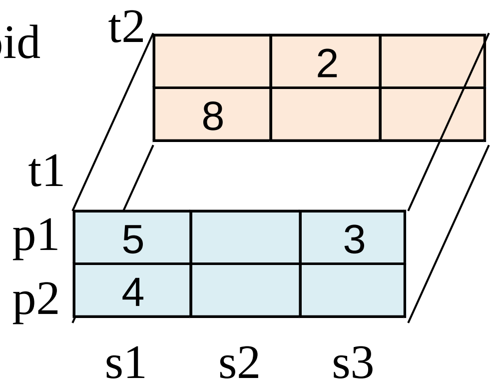
- There are 2^n cuboids in an n-dimensional data cube

Computing a Data Cube



Computing a Data Cube Using SQL

Base cuboid



select prodid, storeid, sum(quantity)

from sales

group by prodid, storeid

	s1	s2	s3
p1	5	2	3
p2	12		

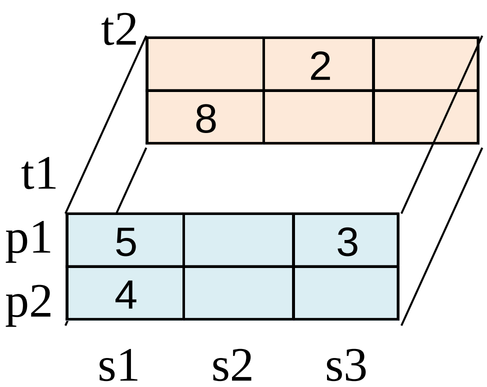
s1	s2	s3
17	2	3

All
22

Apex cuboid

Roll up vs. Drill down

- Rollup (drill down) for fewer (more) details of data



Roll up

Drill down

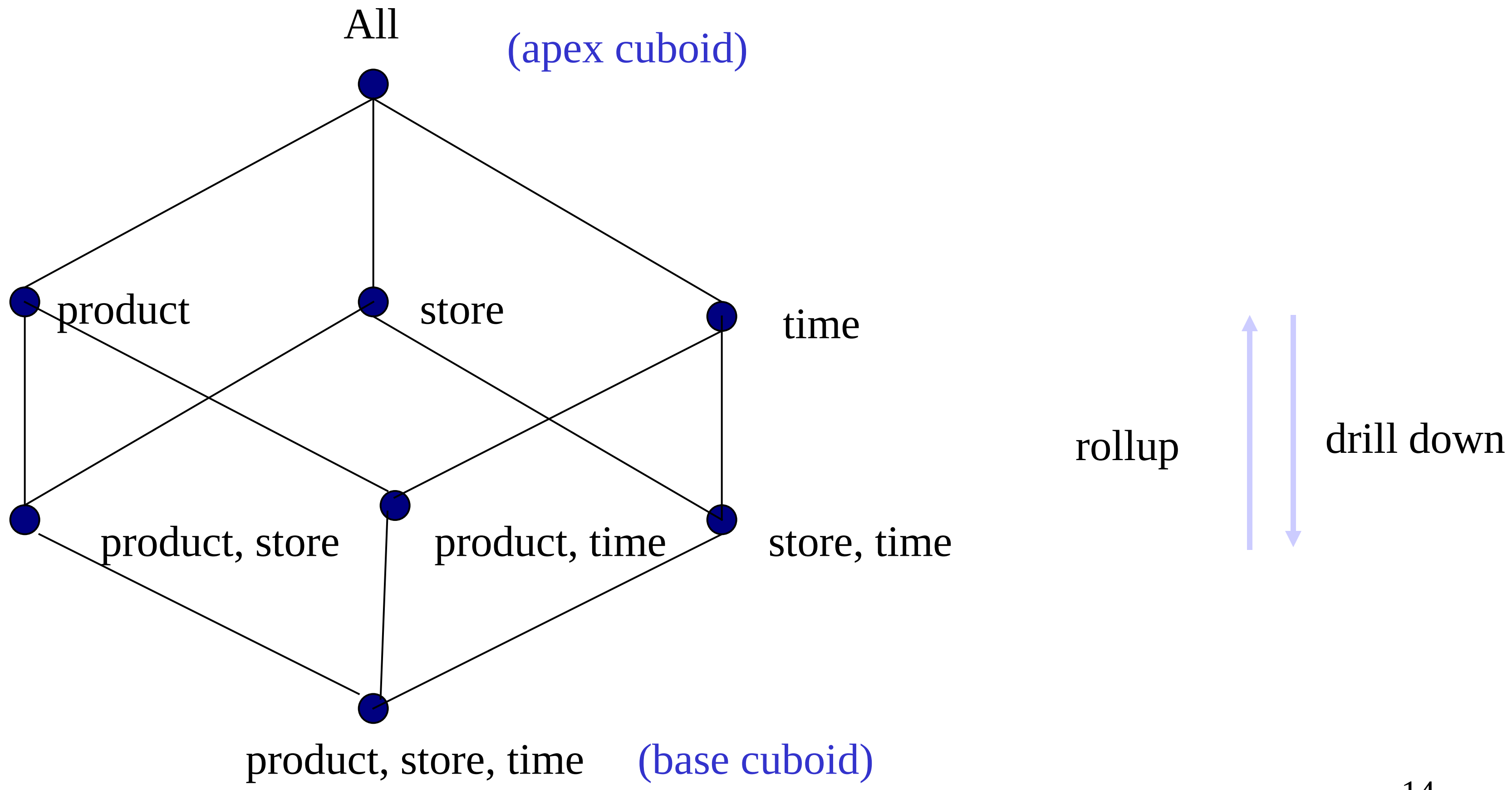
	s1	s2	s3
p1	5	2	3
p2	12		

s1	s2	s3
17	2	3

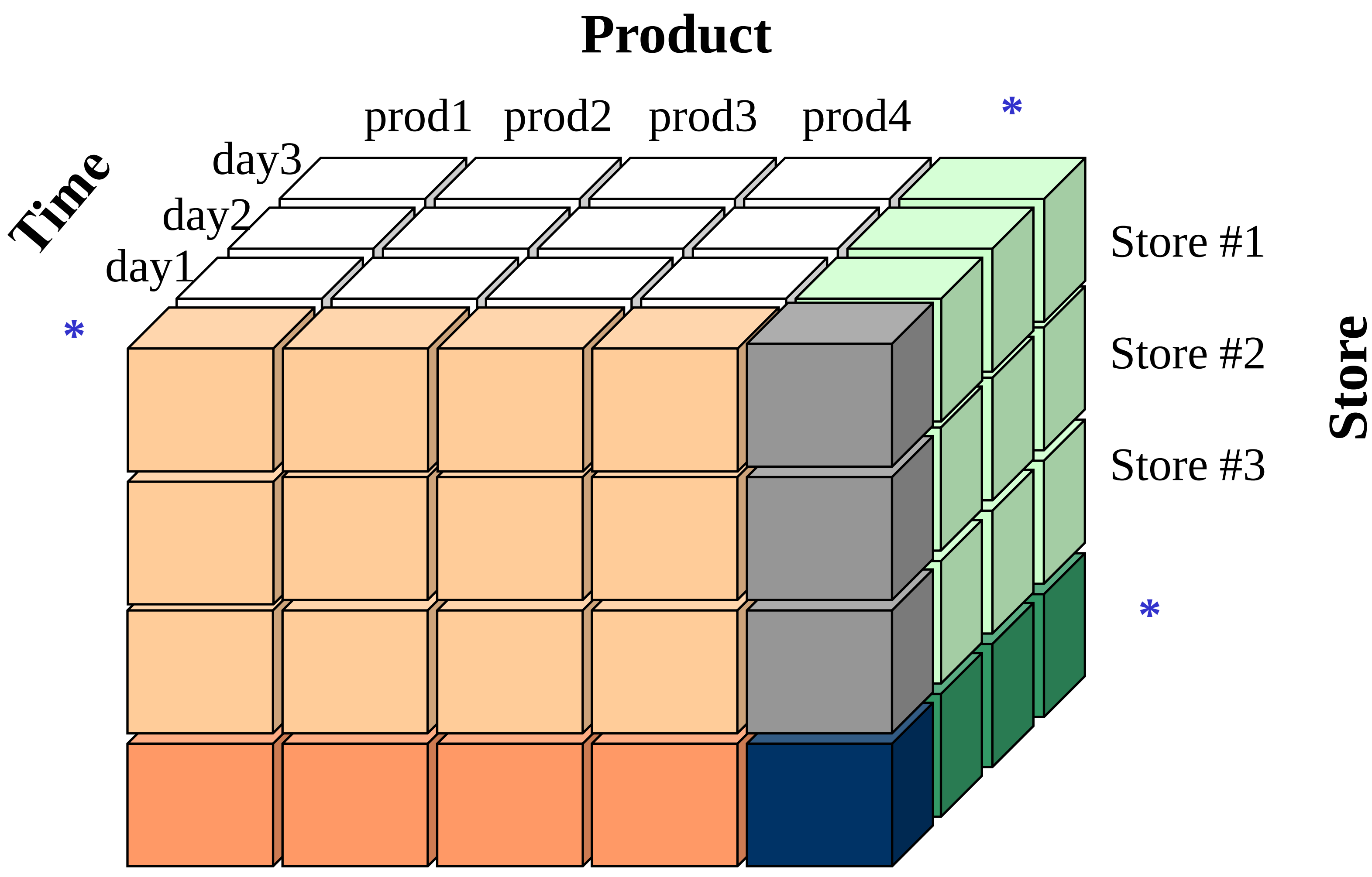
All
22

Representing Cube in a Lattice

- Line indicates the superset/subset relationship of dimensions in the cuboids
  - E.g., {product} is a subset of {product, store} & {product, time}

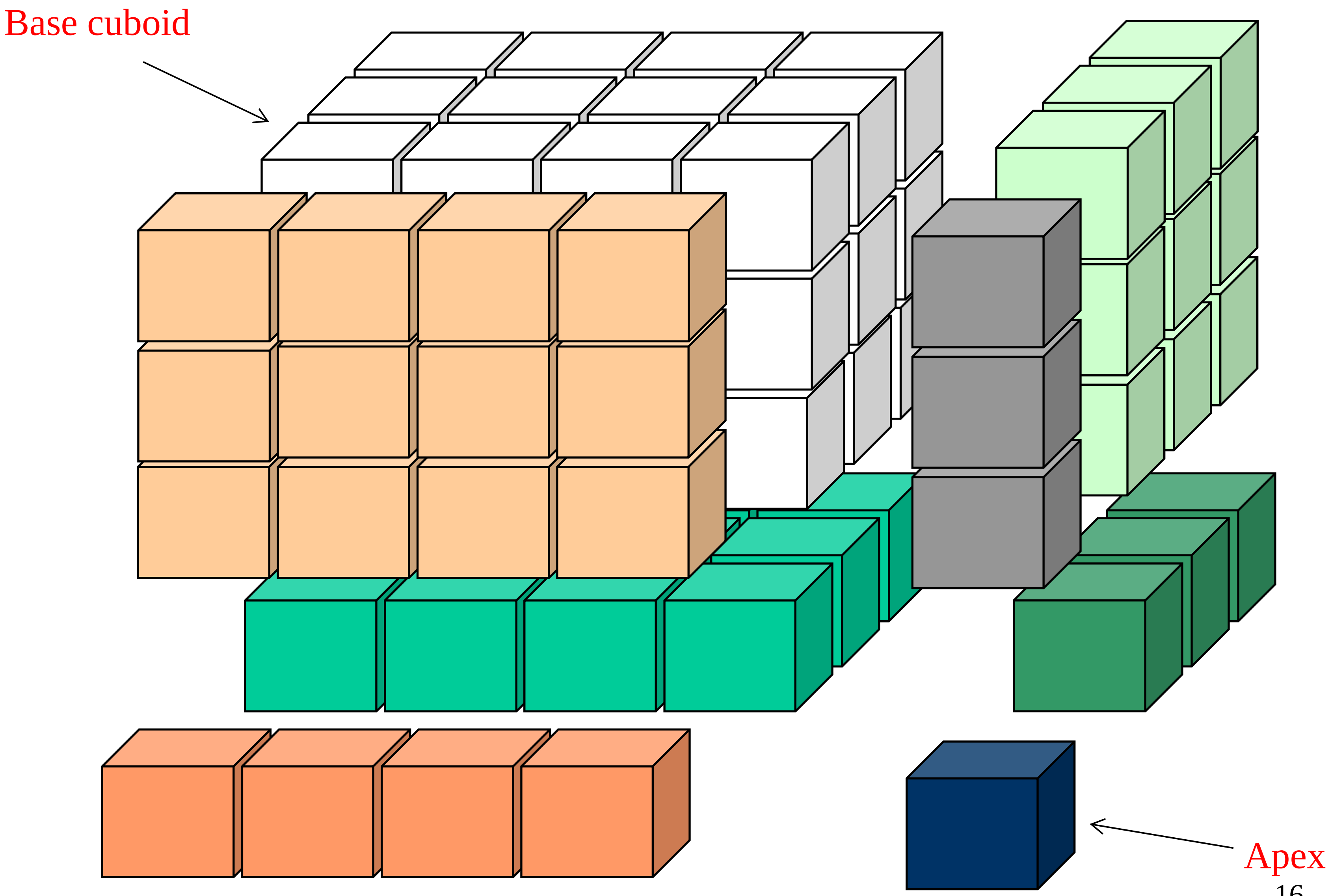


Visualizing Data Cube



*: values in the corresponding dimension are aggregated

Visualizing Cuboids



ROLAP vs. MOLAP

- Two methods to support cube-structured data for OLAP
  - ROLAP (relational OLAP)
  - MOLAP (multidimensional OLAP)

- ROLAP
  - Data stored in relations, organized into special schemas
  - E.g., star & snowflake schemas
  - One of relation in the schema is the fact table (= base cuboid)
  - Other relations give information about the dimensions
  - Fact table may be extended to store summary/aggregate data

- MOLAP: Data & aggregates stored in multi-dimensional structure (e.g., array)

Star Schema

Times
<u>TimeID</u>
Date
Day
Week
Month
Quarter
Year

Dimension table

Sales
<u>ProdID</u>
<u>StoreID</u>
<u>TimeID</u>
Quantity
Amount

Fact table

Products
<u>ProdID</u>
ProdName
Category

Dimension table

Stores
<u>StoreID</u>
City
State
Region

Dimension table

FD (functional dependencies) in Stores table:

storeid → city, state, region

state → region

Is Stores in BCNF?

Functional Dependencies

- A form of constraint (hence, part of the schema)
- Finding them is part of the database design
- Used heavily in schema refinement

Definition:

If two tuples agree on the attributes

$A_1, A_2, \ldots A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots B_m$

Formally: $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Examples

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

- EmpID $\longrightarrow$ Name, Phone, Position
- Position $\longrightarrow$ Phone
- but Phone $\not\longrightarrow$ Position

In General

- To check $A \longrightarrow B$, erase all other columns

…	A	…	B	
	X1		Y1	
	X2		Y2	
	…		…	

- check if the remaining relation is many-one (called ***functional*** in mathematics)

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Keys of Relation

- After defining FDs, we can define keys
- Key of a relation R is a set of attributes that
  - functionally determines all attributes of R, and
  - none of its subsets determines all attributes of R
- Superkey
  - a set of attributes that contains a key
- We will need to know the keys of the relations in a DB schema, so that we can refine the schema

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations.

A relation R is in BCNF if and only if:

Whenever there is a nontrivial FD $A_1, A_2, \ldots A_n \longrightarrow B$ for R , it is the case that $\{ A_1, A_2, \ldots A_n \}$ is a super-key for R.

In English (though a bit vague):

Whenever a set of attributes of *R* is determining another attribute, it should determine ***all*** attributes of *R*.

Example

Name	SSN	Phone Number
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(201) 555-1234
Joe	909-438-44	(212) 555-4000

What are the dependencies?

SSN → Name

What are the keys?

(SSN, Phone Number)

Is it in BCNF?

Decompose it into BCNF

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN → Name

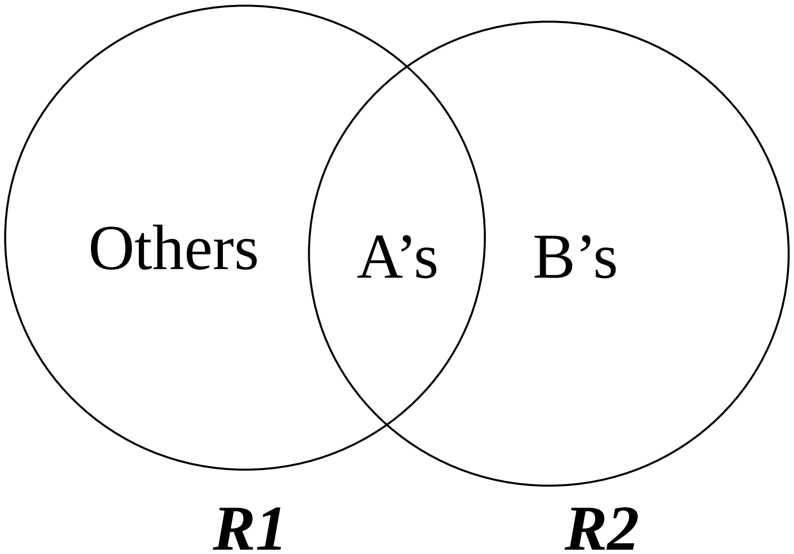
SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

BCNF Decomposition

Find a dependency that violates the BCNF condition:

$$A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$$

Decompose:



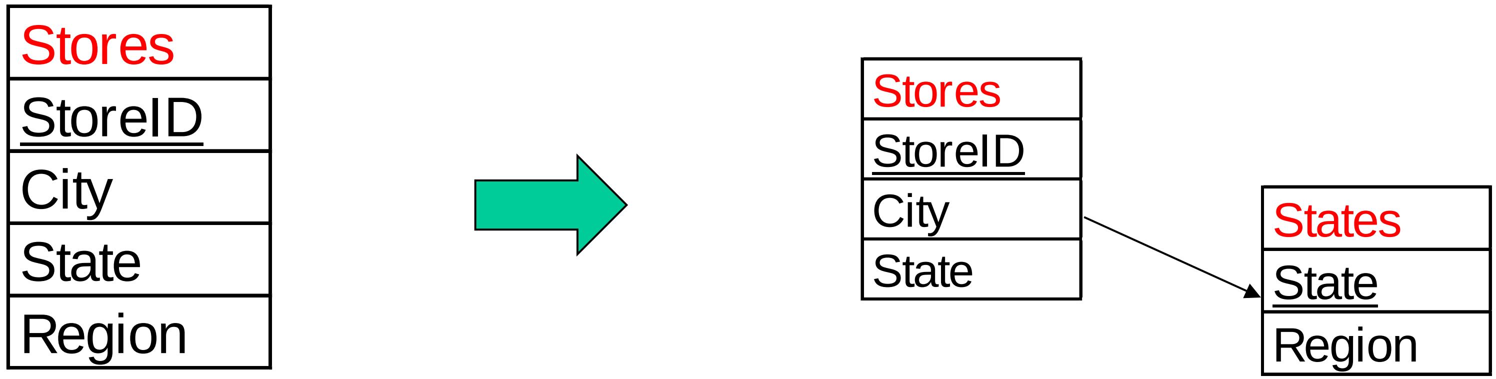
Continue until there are no BCNF violations left.

Normalizing Stores Table

FD in Stores:

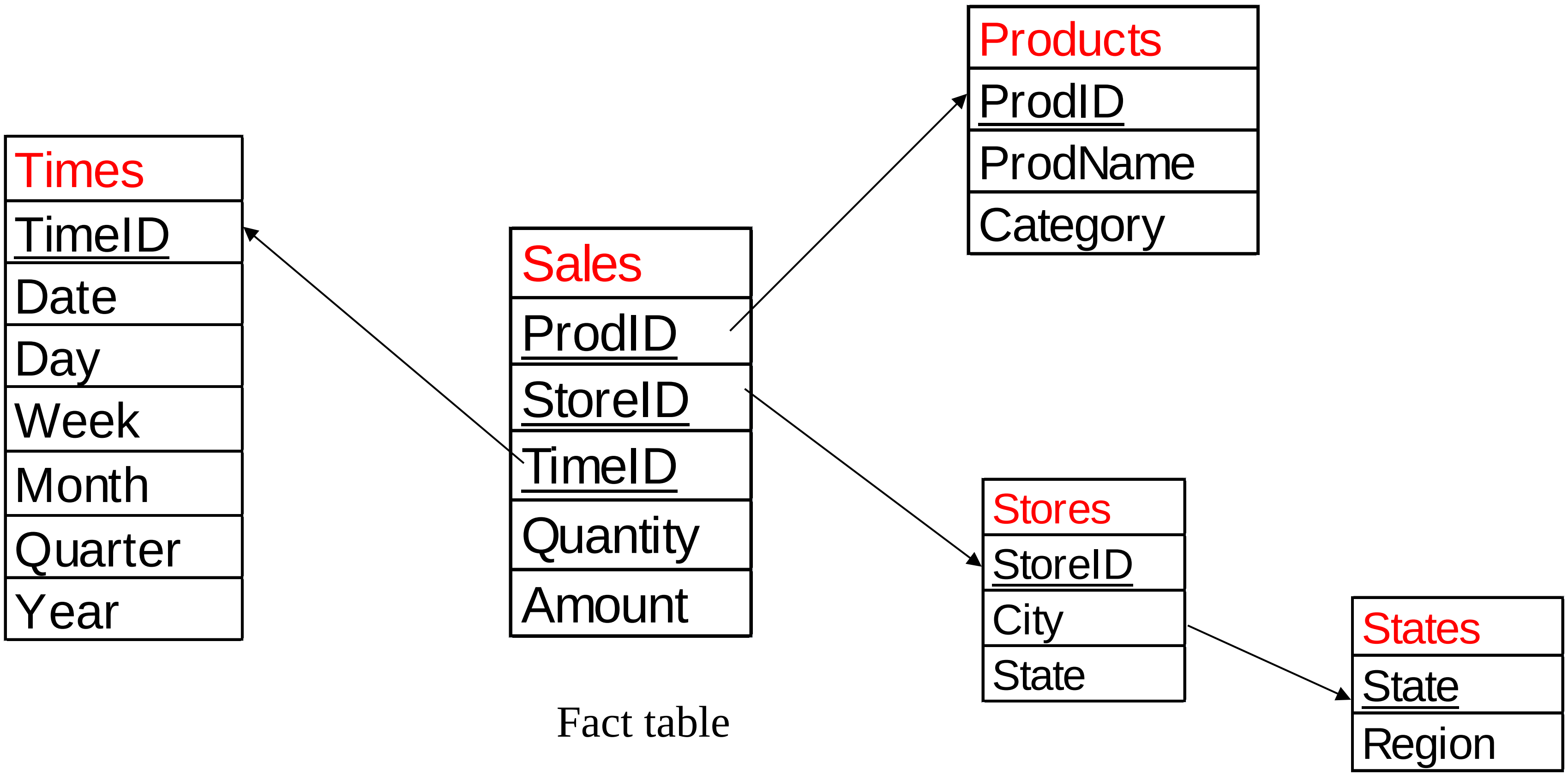
storeid → city, state, region

state → region



Normalize tables in star schema into BCNF

Snowflake Schema



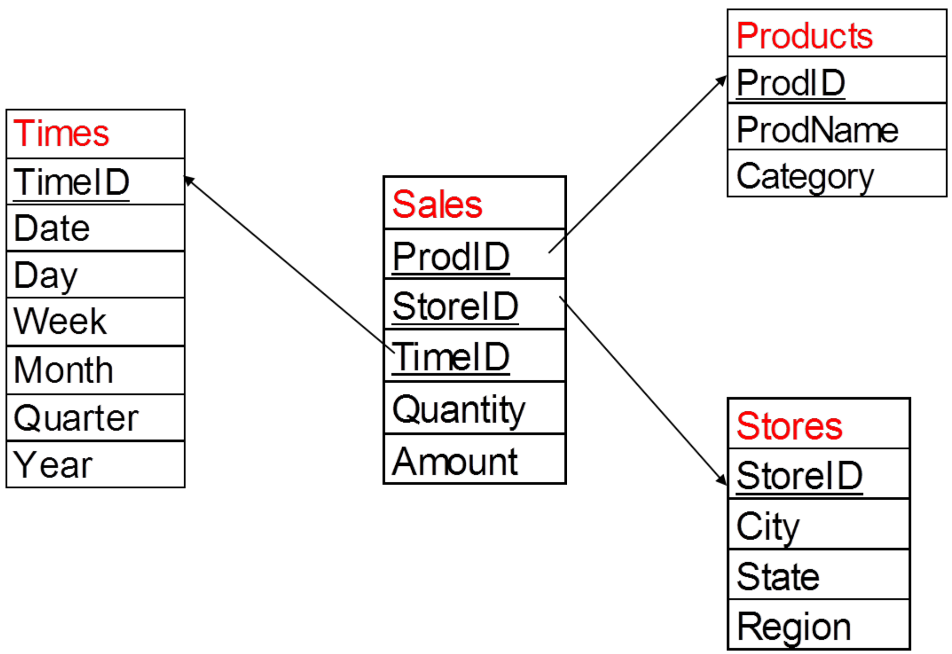
Normalize dimensional tables in star schema into BCNF

Star vs. Snowflake

- Star schema may contain redundancy, but this is less an issue than in OLTP
- Reason 1: Dimension tables rarely change, thus reducing chances of anomaly caused by redundancy (e.g., update anomaly)
- Reason 2: Dimensional tables are typically much smaller than fact table, so space saved is not that significant
- Reason 3: Avoid expressive joins among dimension tables, thus improving query response time

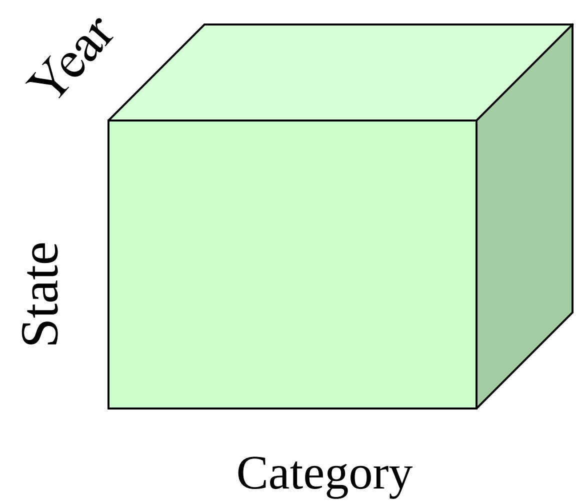
Dimension Tables

- Describe properties about the dimensions
  - May be used to construct additional base cuboids & data cubes



Category	State	Year	Quantity
cell phone	CA	1998	12
cell phone	NC	1998	20
laptop	CA	1998	51
laptop	NC	1999	5
laptop	NC	2000	18

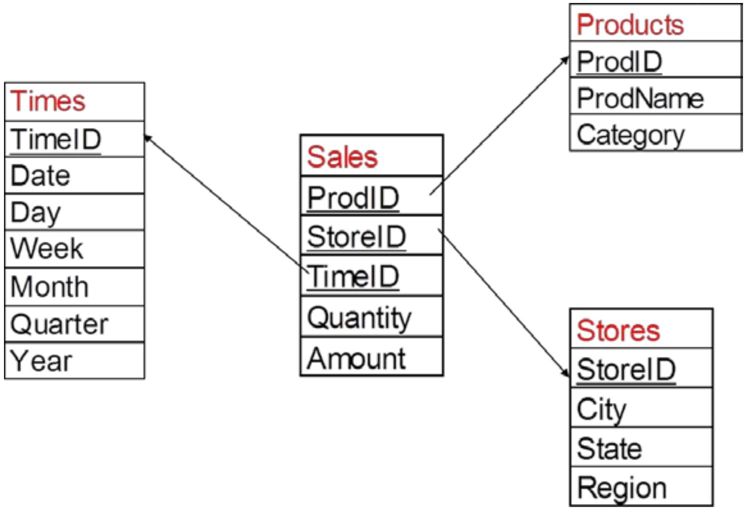
Fact table (base cuboid)



SQL for Generating New Base Cuboid

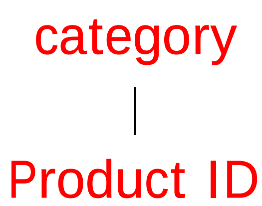
Category	State	Year	Quantity
cell phone	CA	1998	12
cell phone	NC	1998	20
laptop	CA	1998	51
laptop	NC	1999	5
laptop	NC	2000	18

```
select      category, state, year, sum(quantity)
from        sales natural join products
            natural join stores natural join times
group by category, state, year
```

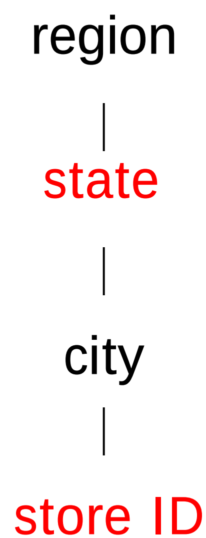


Dimension Hierarchy

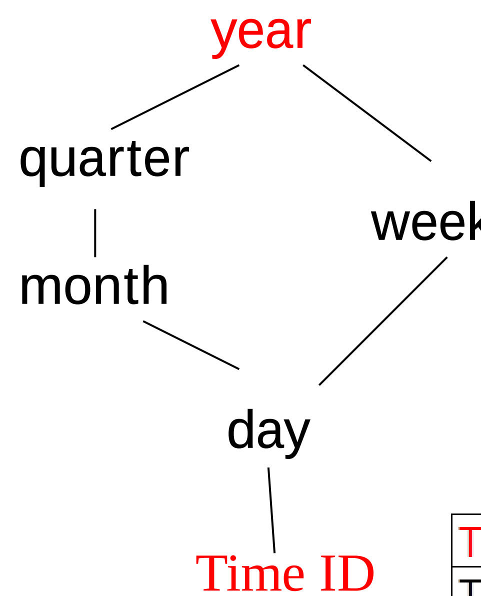
- Attributes of a dimension often form a hierarchy
  - Representing different levels of aggregation



Products
<u>ProdID</u>
ProdName
Category

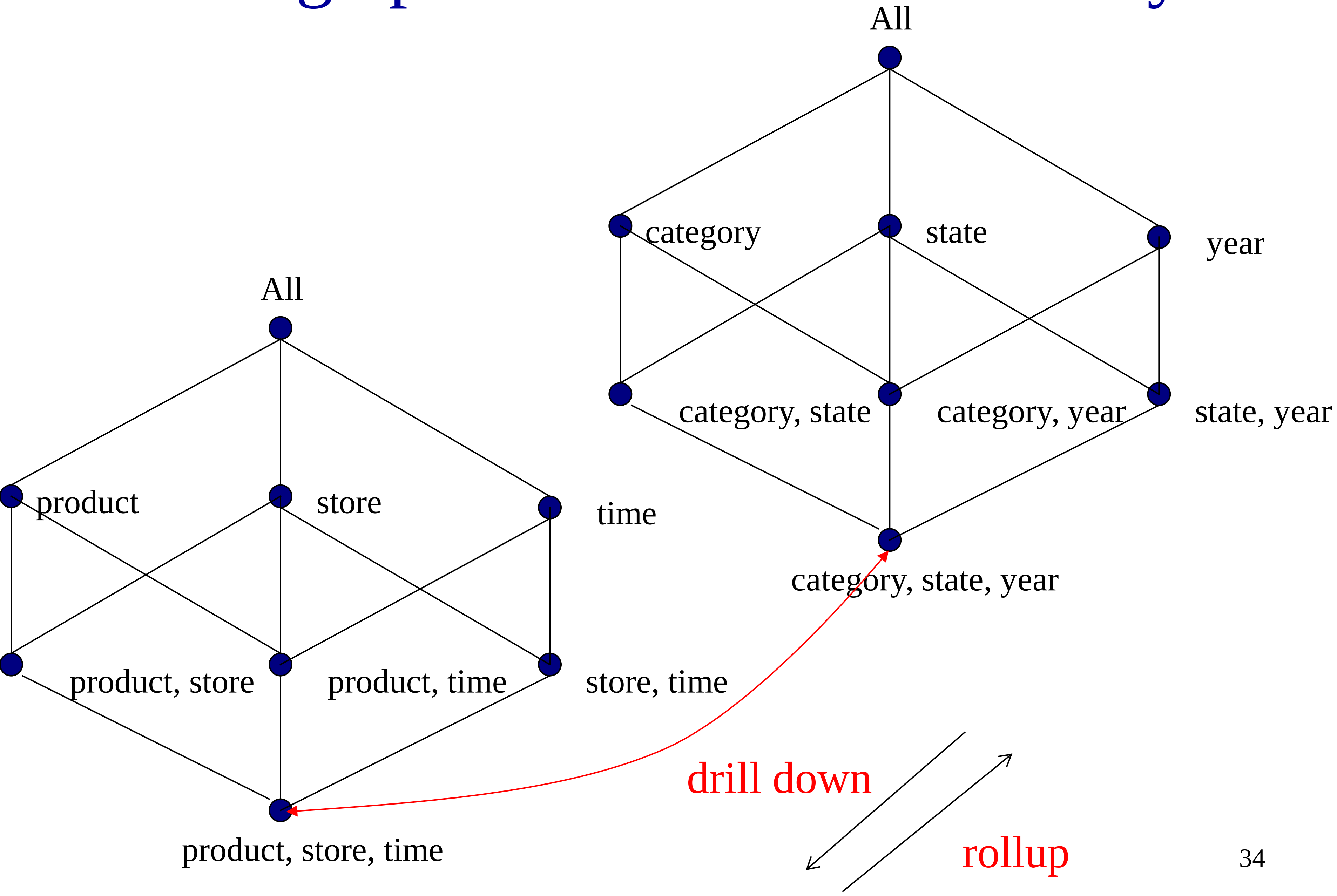


Stores
<u>StoreID</u>
City
State
Region



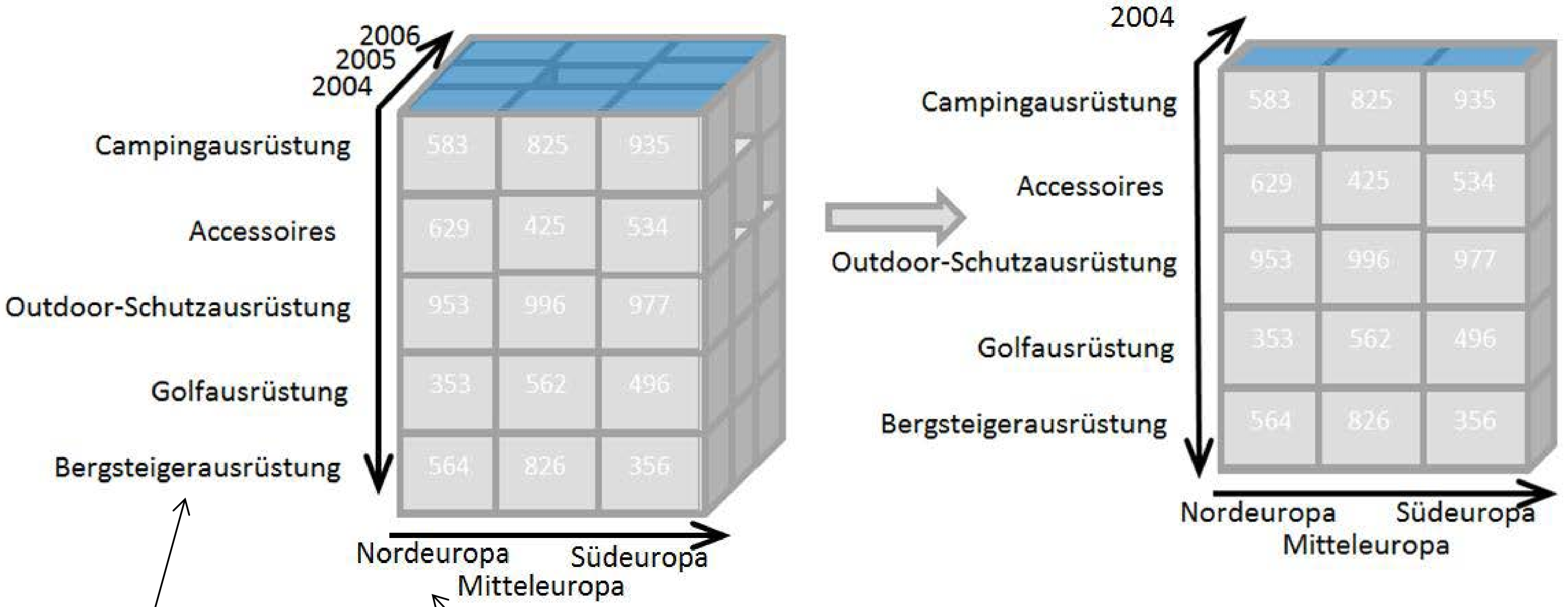
Times
<u>TimeID</u>
Date
Day
Week
Month
Quarter
Year

Rolling up on Dimension Hierarchy



Slicing

- Specify specific values for one or more dimensions, e.g., year = 2004



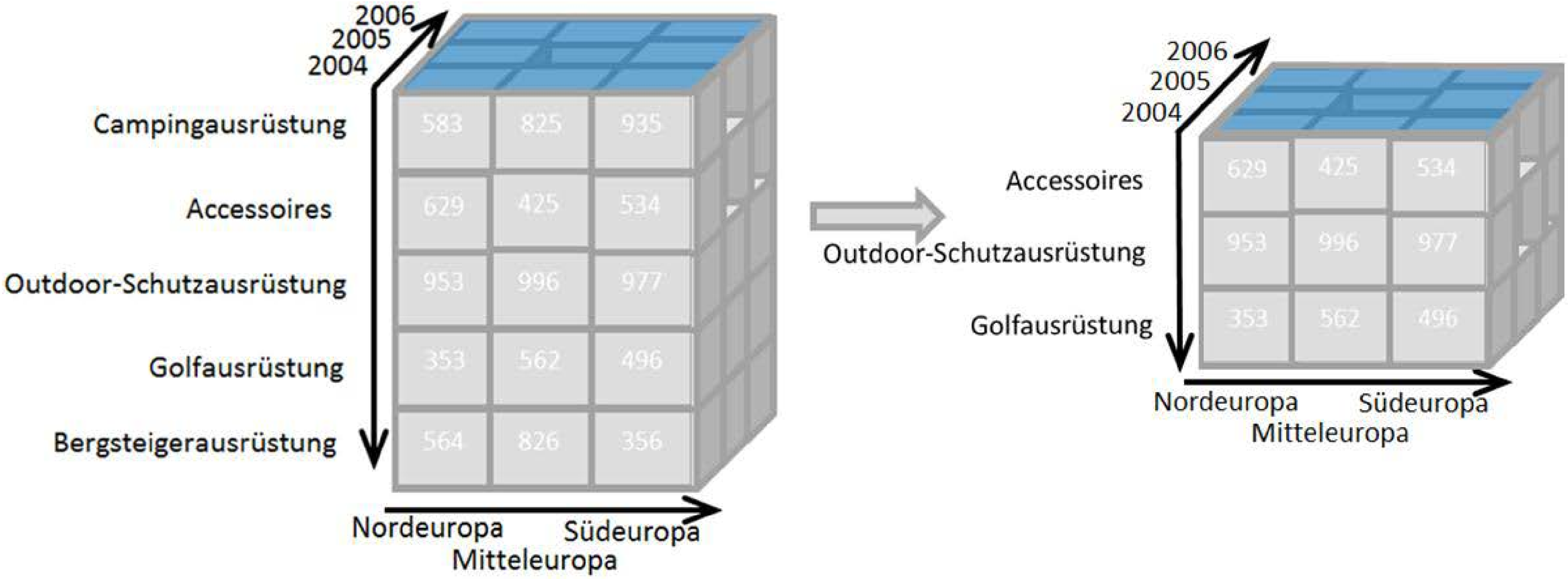
Climbing equipment

Northern Europe

From Wikipedia

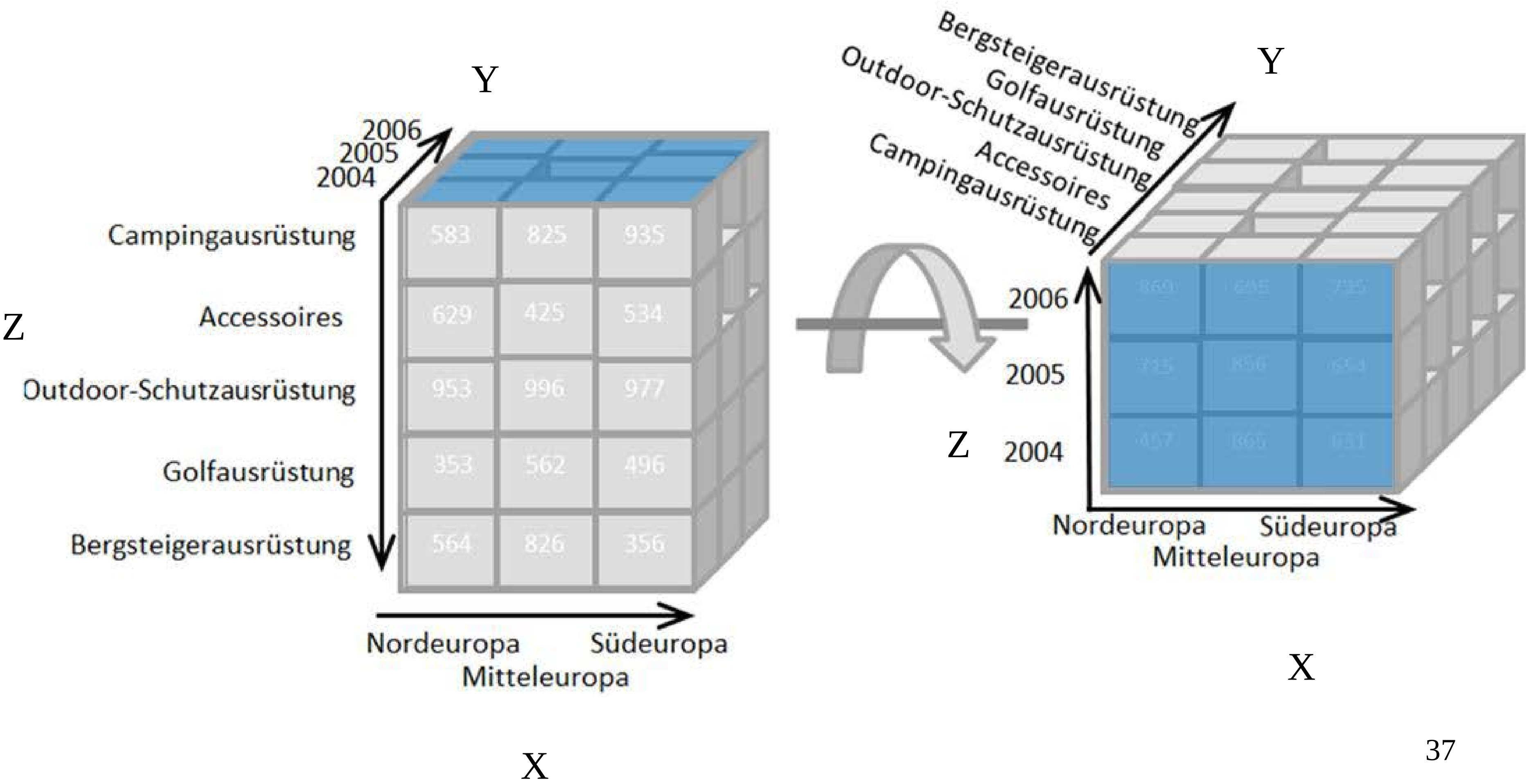
Dicing

- Specify a range of values for one or more dimensions



Pivoting (a data cube)

- Rotating the data cube



Pivoting (a table) in Excel

- From table view to multidimensional view

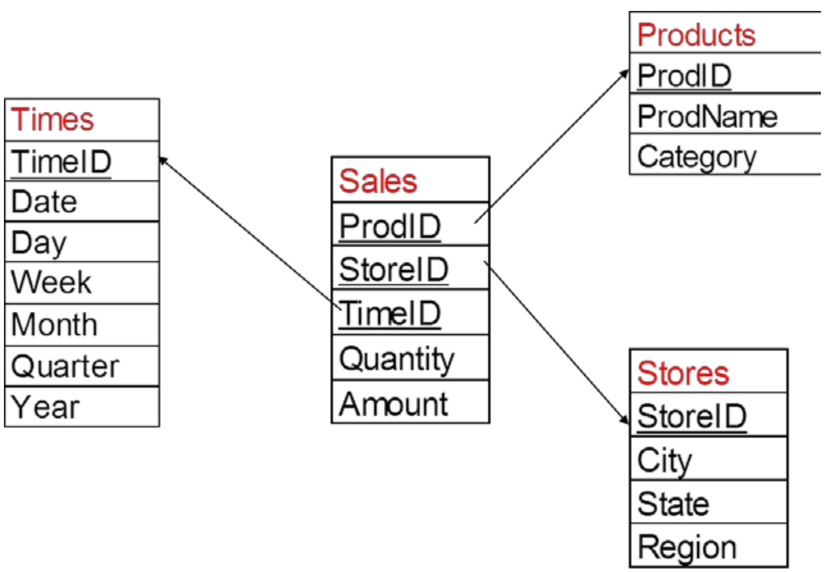
	A	B	C	D	E	F	G
1	**Region**	**Gender**	**Style**	**Ship Date**	**Units**	**Price**	**Cost**
2	East	Boy	Tee	1/31/2005	12	11.04	10.42
3	East	Boy	Golf	1/31/2005	12	13	12.6
4	East	Boy	Fancy	1/31/2005	12	11.96	11.74
5	East	Girl	Tee	1/31/2005	10	11.27	10.56
6	East	Girl	Golf	1/31/2005	10	12.12	11.95
7	East	Girl	Fancy	1/31/2005	10	13.74	13.33
8	West	Boy	Tee	1/31/2005	11	11.44	10.94
9	West	Boy	Golf	1/31/2005	11	12.63	11.73
10	West	Boy	Fancy	1/31/2005	11	12.06	11.51
11	West	Girl	Tee	1/31/2005	15	13.42	13.29
12	West	Girl	Golf	1/31/2005	15	11.48	10.67

Sum of Units	Ship Date					
Region	1/31/2005	2/28/2005	3/31/2005	4/30/2005	5/31/2005	6/30/2005
East	66	80	102	116	127	125
North	96	117	138	151	154	156
South	123	141	157	178	191	202
West	78	97	117	136	150	157
(blank)						
Grand Total	363	435	514	581	622	640

Pivot table

An Example OLAP Scenario

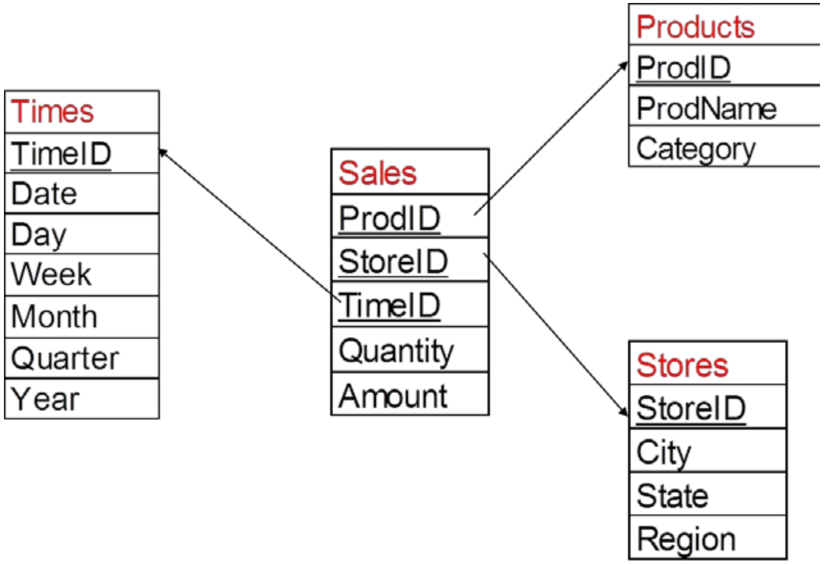
- Sales in 2009 was not good
- First, let's examine yearly sales
  - Rollup from the base cuboid to (1-d cuboid) year




```
select   year, sum(amount)
from     sales s, times t
where    s.timeid = t.timeid
group by year;
```

An Example OLAP Scenario

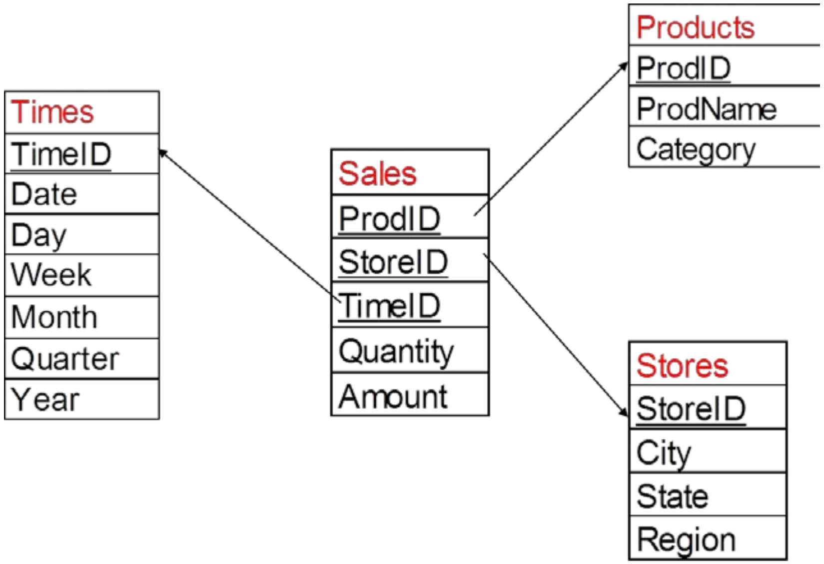
- Take a slice on year = 2009
- Drill down to month




```
select   month, sum(amount)
from     sales s, times t
where    s.timeid = t.timeid
         and t.year = 2009
group by month;
```

An Example OLAP Scenario

- Sales amounts at 3rd month were relatively low
- Further drill down to product category to find out why

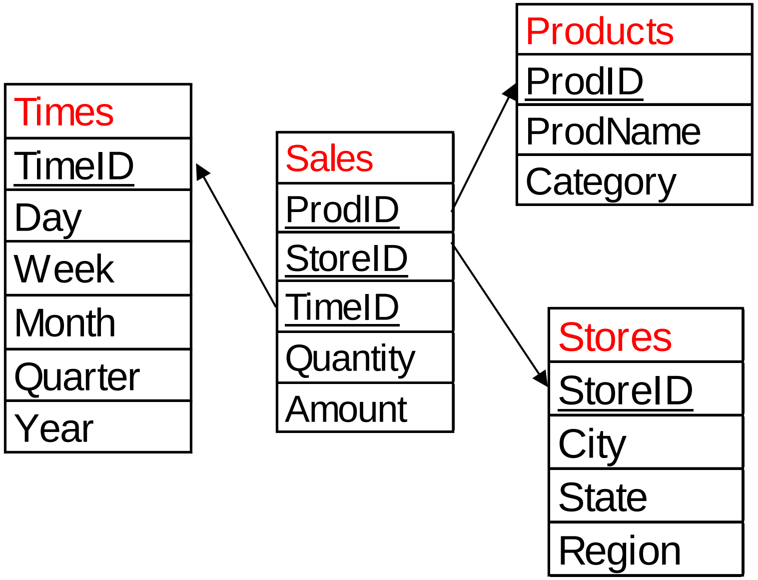



```
select   month, category,
         sum(amount)
from     sales s, times t, products p
where    s.timeid = t.timeid
         and s.prodid = p.prodid
         and t.year = 2009
group by month, category;
```

An Example OLAP Scenario

- Category ‘laptop’ was not selling well in the 3rd month
- Slice on Laptop
- Drill down to store region to find out why

=> Laptops were not selling well in east region in the 3rd month of 2009!




```
select    l.region, sum(quantity)
from      sales s, times t, products p, stores l
where     s.timeid = t.timeid
          and s.prodid = p.prodid
          and t.year = 2009
          and p.category = 'Laptop'
          and s.storeid = l.storeid
          and t.month = 3
group by l.region;
```

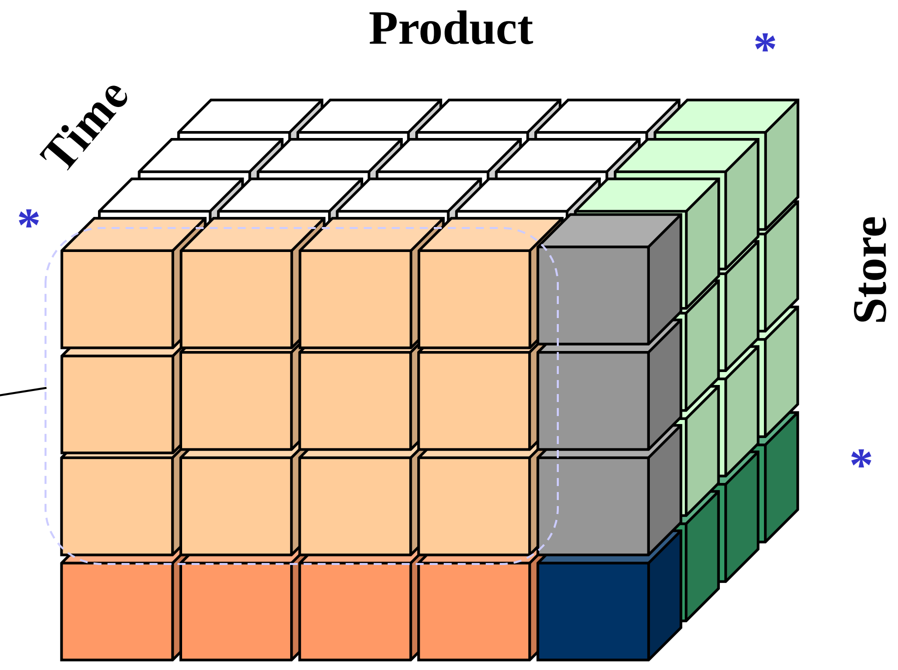
Roadmap

- Multidimensional data model
- Implementation: ROLAP vs. MOLAP
- Language constructs
  - Cube, rollup operators
  - Window functions

The Cube Operator

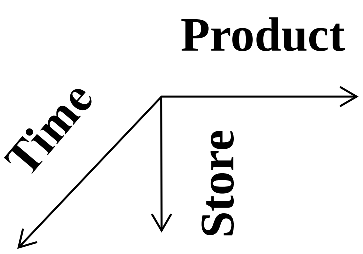
- Given fact table F, create augmented table Cube(F) for data cube
  - Adding tuples with *'s in some dimensions
  - * in dimension D_i: tuple contains all corresponding values the dimension D_i

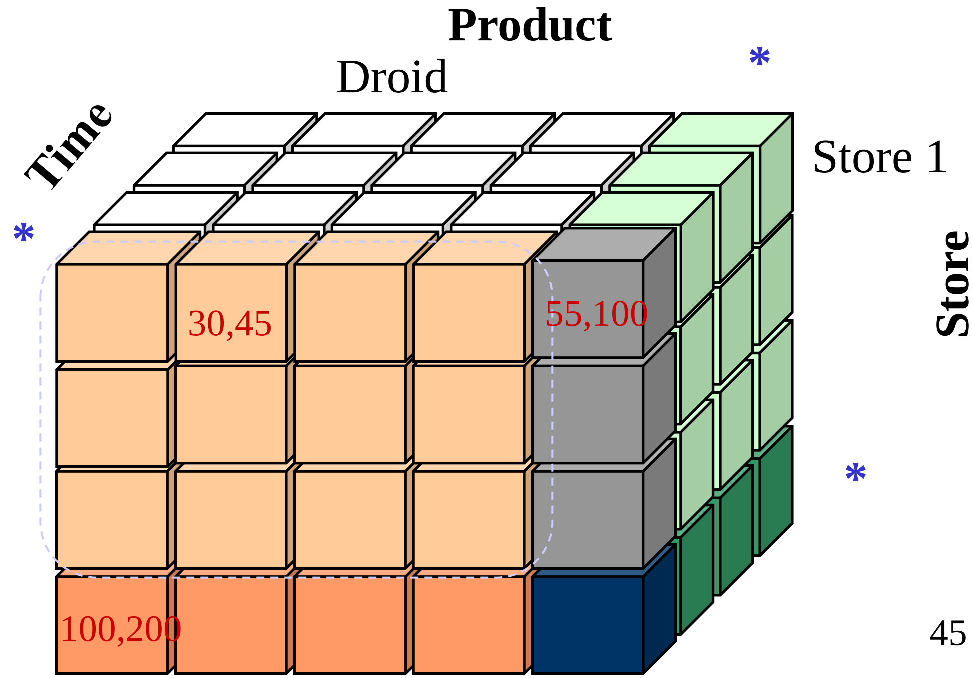
* at time dimension



Tuple with ‘*’ Values

- Tuple: (product, store, time, quantity, amount)
  - E.g.,(‘Droid’, ‘store #1’, ‘2009-3-1’, 5, 12), tuple in base cuboid
- (‘Droid’, ‘store #1’, *, 30, 45)
  - 30 Droids sold at store #1 at all times
- (*, ‘store #1’, *, 55, 100)
  - Total sales at store #1 (of all products and at all times)
- (‘Droid’, *, *, 100, 200)
  - Total sales of Droid



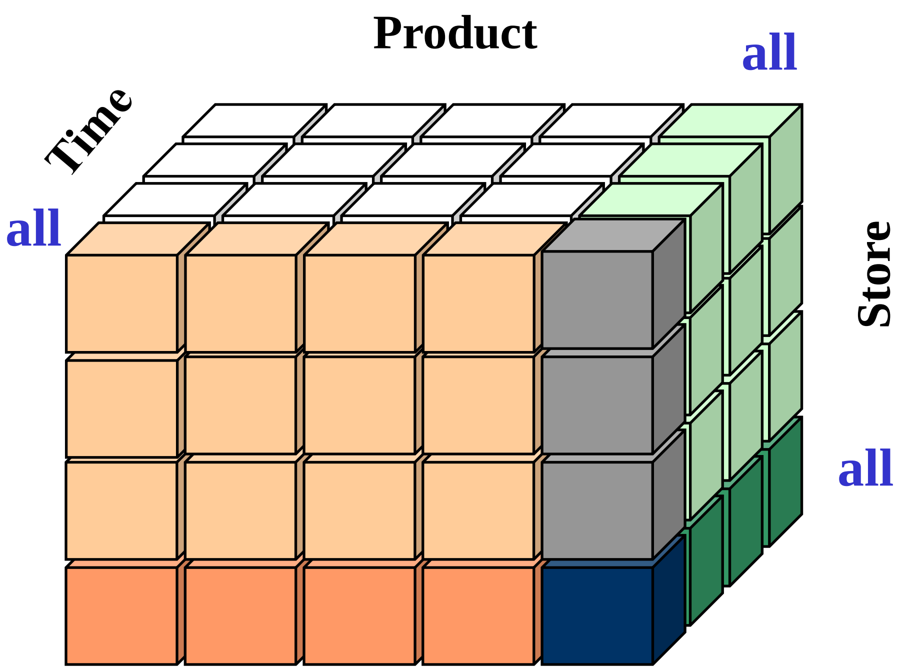


Size of Data Cube

- k dimensions: $D_1, D_2, \ldots, D_k$
  - $|D_i|$: # of values on the dimension D_i

- Size (# of cells) of base cuboid
  - $|D_1| * |D_2| * \ldots * |D_k|$

- Size of cube
  - $(|D_1| + 1) * \ldots * (|D_k| + 1)$



Cube Operator in SQL (SQL Server)

```
select      category, state, year, sum(quantity)
from        sales s
group by    category, state, year with cube;
```

Where is the value for (cell, *, 2000)?

Category	State	Year	Total
cell	CA	1998	3
cell	NC	1998	5
laptop	CA	1998	7
laptop	NC	2000	6

Sales fact table F

Category	State	Year	Total
cell	CA	1998	3
cell	NC	1998	5
laptop	CA	1998	7
laptop	NC	2000	6
cell	CA	NULL	3
cell	NC	NULL	5
laptop	CA	NULL	7
laptop	NC	NULL	6
cell	NULL	1998	8
laptop	NULL	1998	7
laptop	NULL	2000	6
NULL	CA	1998	10
NULL	NC	1998	5
NULL	NC	2000	6
cell	NULL	NULL	8
laptop	NULL	NULL	13
NULL	CA	NULL	10
NULL	NC	NULL	11
NULL	NULL	1998	15
NULL	NULL	2000	6
NULL	NULL	NULL	21

2d

1d

Cube(F)

Rollup Operator in SQL (MariaDB)

- Aggregates on the tails of the sequence of grouping attributes
  - (category, state, year) (category, state) (category) ()
  - From 3-d, 2-d, 1-d, to 0-d

```
select    category, state, year, sum(quantity)
from      sales natural join stores
          natural join products
          natural join times
group by  category, state, year with rollup;
```

Category	State	Year	Total
cell	CA	1998	3
cell	NC	1998	5
laptop	CA	1998	7
laptop	NC	2000	6

Sales fact table F

Category	State	Year	Total
cell	CA	1998	3
cell	NC	1998	5
laptop	CA	1998	7
laptop	NC	2000	6
cell	CA	NULL	3
cell	NC	NULL	5
laptop	CA	NULL	7
laptop	NC	NULL	6
cell	NULL	NULL	8
laptop	NULL	NULL	13
NULL	NULL	NULL	21

Rollup(F)

Window Function

- Allow calculations to be performed across a set of rows related to the current row

```
function ([expression]) OVER (
  [ PARTITION BY expression_list ]
  [ ORDER BY order_list [ frame_clause ] ] )

function: a valid window function

expression_list: expression | column_name [, expr_list ]

order_list: expression | column_name [ ASC | DESC ]
  [, ... ]

frame_clause: <rows and range-type frames>
```

Window Functions

- Rank()
- Dense_rank(): no gaps between rank values
- Sum()
- Avg()
- Count()
- …

Window Examples

- Over()
  - Window is the entire relation

- Over (partition by ProdID)
  - Window is the partition with the same ProdID

- Over (order by ProdID rows unbounded proceding)
  - Window includes all preceding rows in the order given by ProdID

Rank vs. Dense_Rank

- Dense_rank(): no gap between ranks, while ranks by rank() may have skipped rank values: 1, 3, 5, … (ranks 2, 4, … are skipped)

```
select    ProdID, TimeID, StoreID, Amount,
          rank() over (order by Amount desc) r,
          dense_rank() over (order by Amount desc) dr
from      sales
where     ProdID = 'p1'
order by r;
```

ProdID	TimeID	StoreID	Amount	Rank	Dense
p1	t4	s4	12	1	1
p1	t6	s4	12	1	1
p1	t5	s4	8	3	2
p1	t3	s2	8	3	2
p1	t1	s2	7	5	3
p1	t2	s1	6	6	4
p1	t1	s3	5	7	5
p1	t2	s4	3	8	6
p1	t1	s1	2	9	7

Top k by Ranks

- Use subquery to obtain ranks first

```
select *
from (      select ProdID, TimeID, StoreID, Amount,
            rank() over (order by Amount desc) as rank
            from sales
            where ProdID = 'p1')  r
where       r.rank <= 3
order by rank;
```

ProdID	TimeID	StoreID	Amount	Rank
p1	t4	s4	12	1
p1	t6	s4	12	1
p1	t5	s4	8	3
p1	t3	s2	8	3

Ranking Using Multiple Expressions

- Resolve ties by the 1st expression, then by 2nd, …

```
select      TimeID, Quantity, Amount,
            rank() over (order by quantity desc, amount desc)
            as rank
from        sales
where       ProdID = 'p2'
order by rank;
```

TimeID	Quantity	Amount	Rank
t4	4	8	1
t3	4	7	2

Ranking over Aggregates

- "Order by" followed by an aggregate function

```
select      ProdID, sum(Amount) as Total,
            rank() over (order by sum(Amount) desc) r
from        sales
group by ProdID
Order by r;
```

Accept other aggregation
e.g., avg(Amount)
avg(Quantity)

ProdID	Total	Rank
p1	63	1
p4	37	2
p3	19	3
p5	16	4
p2	15	5

Within-Partition Ranking

- Ranking values within each group, via "partition by"

```
select    storeid, timeid, Amount,
          rank() over (partition by storeid order by Amount desc) as GroupRank
from sales
where     prodid = 'p1'
order by storeid, grouprank;
```

StoreID	TimeID	Amount	GroupRank
s1	t2	6	1
s1	t1	2	2
s2	t3	8	1
s2	t1	7	2
s3	t1	5	1
s4	t4	12	1
s4	t6	12	1
s4	t5	8	3
s4	t2	3	4

Frame Clause

- Examples:
  - rows unbounded preceding
  - rows between 1 preceding and 1 following

- Further limit the window to frame

Accumulative Sum

```
select ProdID, StoreID, amount,
        sum(amount) over (order by prodid, storeid rows
unbounded preceding) as AccAmt
from sales order by prodid, storeid;
```

ProdID	StoreID	amount	AccAmt
p1	s1	8	8
p1	s1	6	14
p1	s2	7	21
p1	s2	8	29
p1	s3	5	34
p1	s4	12	46
p1	s4	18	64
p1	s4	12	76
p1	s4	13	89
p2	s2	7	96
p2	s2	8	104
p3	s1	28	132
p3	s1	11	143
p4	s1	36	179
p4	s3	54	233
p4	s3	20	253
p4	s3	68	321
p5	s1	2	323
p5	s2	10	333

Within-Partition Accumulative Sum

```
select ProdID, StoreID, amount,
        sum(amount) over (partition by ProdID order by
StoreID rows unbounded preceding) as AccAmt
from sales order by prodid, storeid;
```

ProdID	StoreID	amount	AccAmt
p1	s1	8	8
p1	s1	6	14
p1	s2	7	21
p1	s2	8	29
p1	s3	5	34
p1	s4	12	46
p1	s4	18	64
p1	s4	12	76
p1	s4	13	89
p2	s2	7	7
p2	s2	8	15
p3	s1	28	28
p3	s1	11	39
p4	s1	36	36
p4	s3	54	90
p4	s3	20	110
p4	s3	68	178
p5	s1	2	2
p5	s2	10	12

Running Average

```
select ProdID, StoreID, amount,
        avg(amount) over (partition by ProdID order by StoreID rows between 1 preceding and 1 following) as RunAvg
from sales order by prodid, storeid;
```

Window size = 3 (one before, one after)

Running Average

ProdID	StoreID	amount	RunAvg
p1	s1	8	7
p1	s1	6	7
p1	s2	7	7
p1	s2	8	6.666666666666667
p1	s3	5	8.333333333333334
p1	s4	12	11.666666666666666
p1	s4	18	14
p1	s4	12	14.333333333333334
p1	s4	13	12.5
p2	s2	7	7.5
p2	s2	8	7.5
p3	s1	28	19.5
p3	s1	11	19.5
p4	s1	36	45
p4	s3	54	36.666666666666664
p4	s3	20	47.333333333333336
p4	s3	68	44
p5	s1	2	6
p5	s2	10	6

Resources

- Window functions in MariaDB
  - https://mariadb.com/kb/en/mariadb/window-functions/