Advanced MD Modeling

SDRP Lecture 2

Christian Thomsen
chr@cs.aau.dk

Slides adapted from Christian S. Jensen,
Torben Bach Pedersen, and Man Lung Yiu

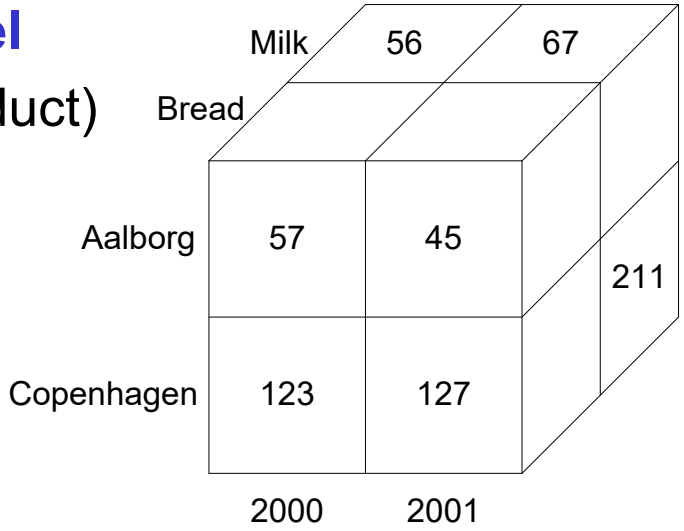
Last time

- Multidimensional databases and data warehousing
  - Why do we need Business Intelligence
  - Data analysis problems
  - Data warehouse introduction
  - Fundamental multidimensional modeling
    - Data is divided into *facts* and *dimensions*
    - Facts have *measures* and ”live” in a *cube*
    - Dimensions describe the facts
  - Relational representations
    - star schemas, snowflake schemas
  - Querying
    - drill-down, roll-up, …
- Exercises

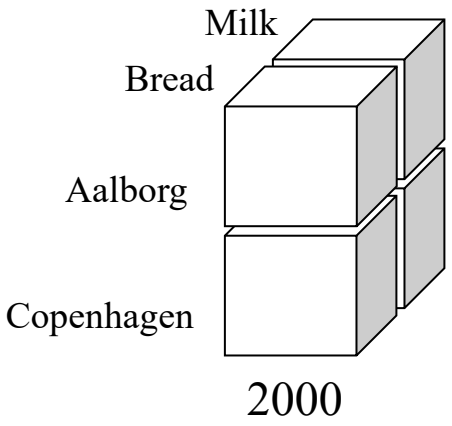
OLAP Queries

Starting level

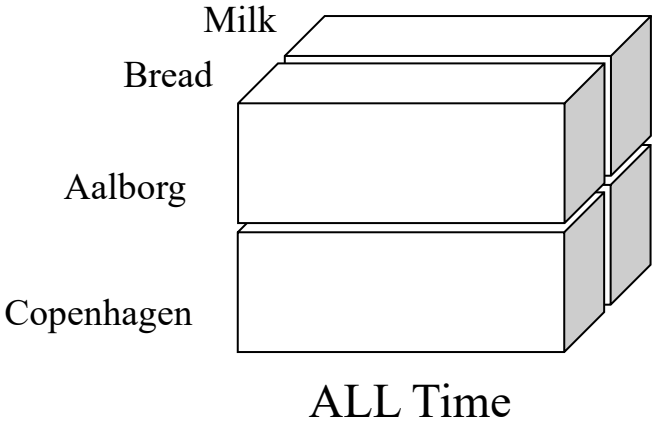
(City, Year, Product)



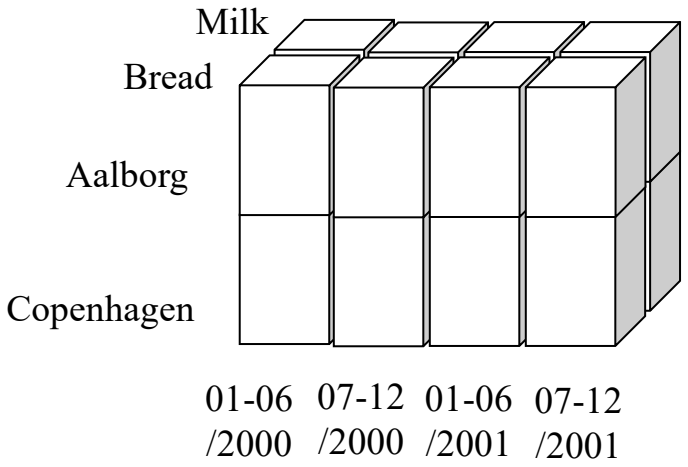
Slice/Dice:



Roll-up: get overview



Drill-down: more detail



Today

Advanced MD modeling

After today's lecture and exercises, you'll be able to

- Handle dimension changes in different ways
- Use special kinds of dimensions
- Use advanced hierarchies

Changing Dimensions

- In the previous lecture, we assumed that dimensions are stable over time:
  - New rows in dimension tables can be inserted
  - Existing rows do not change

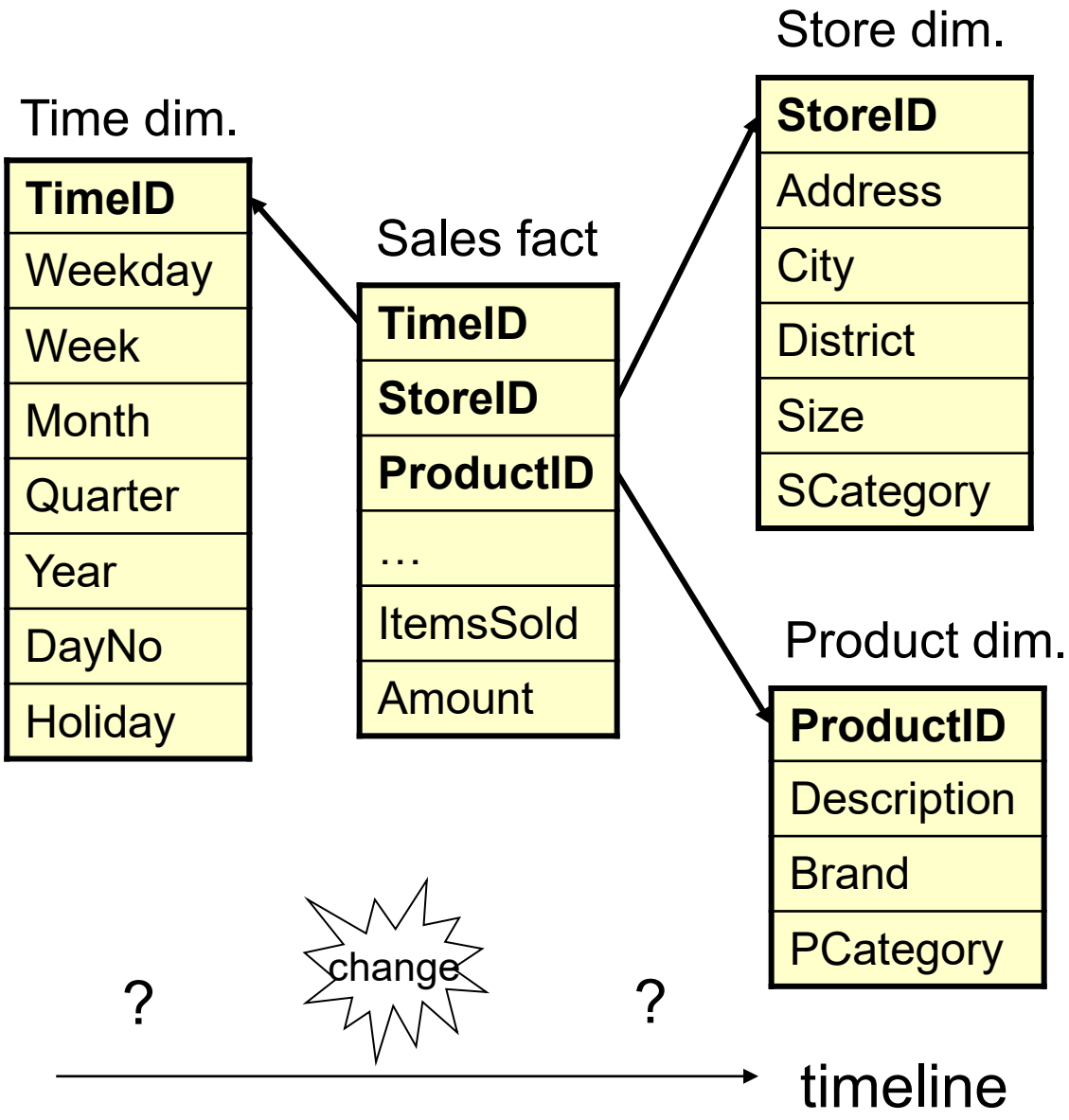
- We now study techniques for handling changes in dimensions
- “Slowly changing dimensions” phenomenon
  - Dimension information can change, but changes are not frequent
  - We (still) assume that the schema is fixed

Handling Changes in Dimensions

Methods to handle dimension changes:

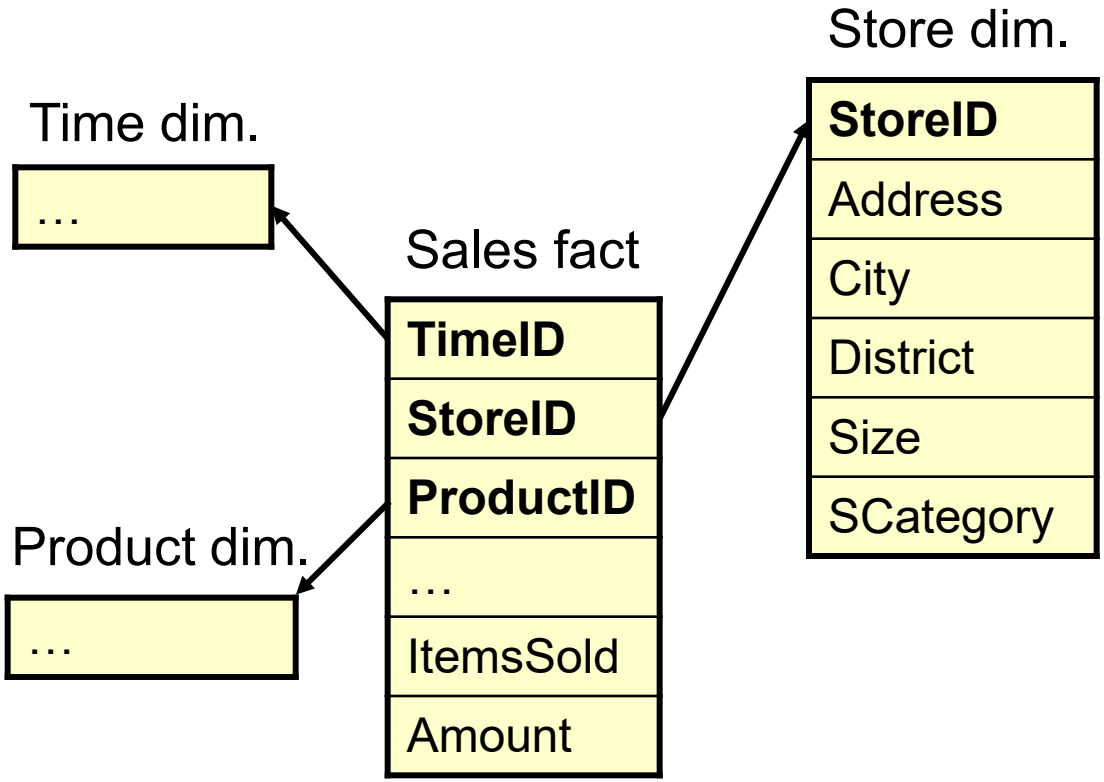
- 1. No special handling
- 2. Versioning of dimension values
  - + Use of Timestamping
- 3. Capturing the previous and the current value
- 4. Split into changing and constant attributes

Example



- Attribute values in dimensions vary over time
  - A store changes Size
  - A product changes Description
  - Districts are changed
- Problems
  - Dimensions not updated ➔ DW is not up-to-date
  - Dimensions updated in a straightforward way ➔ incorrect information in historical data

Example



The store in Aalborg has the size of 250 sq. metres.

On a certain day, customers bought 2000 apples from that store.

Sales fact table

StoreID	…	ItemsSold	…
001		2000	

Store dimension table

StoreID	…	Size	…
001		250	

Solution 1: No Special Handling

Sales fact table

StoreID	…	ItemsSold	…
001		2000	

Store dimension table

StoreID	…	Size	…
001		250	

The size of a store expands

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	Size	…
001		450	

A new fact arrives

StoreID	…	ItemsSold	…
001		2000	
001		3500	

StoreID	…	Size	…
001		450	

What's the problem here?

Solution 1

- **Solution 1**: Overwrite the old values in the dimension tables
- Consequences
  - Old facts point to rows in the dimension tables with incorrect information!
  - New facts point to rows with correct information
- Pros
  - Easy to implement
  - Useful if the updated attribute is not significant, or the old value should be updated for error correction
- Cons
  - Old facts may point to “incorrect” rows in dimensions

Solution 2: Versioning of Rows

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	Size	…
001		250	

different versions of a store

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	Size	…
001		250	
002		450	

A new fact arrives

StoreID	…	ItemsSold	…
001		2000	
002		3500	

StoreID	…	Size	…
001		250	
002		450	

Which store does the new fact refer to?

Solution 2

- **Solution 2**: Versioning of rows with changing attributes
  - The *key* that links the dimension table and the fact table, identifies a *version* of a dimension member, not just a dimension member
  - Surrogate keys make this easier to implement
    - – what if we had used, e.g., the shop's zip code as key?
    - Always use surrogate keys!!!

- Consequences
  - Larger dimension tables
- Pros
  - Correct information captured in DW
  - No problems when formulating queries
- Cons
  - Cannot capture the development over time of the subjects the dimensions describe in the simplest form (but we can fix that)

Solution 2 with Timestamping

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	Size	From	To
001	250	2018	-

attributes: “From”, “To”

StoreID	TimeID	…	ItemsSold	…
001	234		2000	

StoreID	Size	From	To
001	250	2018	2019
002	450	2020	-

StoreID	TimeID	…	ItemsSold	…
001	234		2000	
002	456		3500	

StoreID	Size	From	To
001	250	2018	2019
002	450	2020	-

Solution 2 with Timestamping

- Versioning of rows with changing attributes. Use timestamping of row versions in the SCD with From and To attributes

- Pros
  - Correct information captured in DW
- Cons
  - Larger dimension tables
    - (not really a problem)

Example

- Product descriptions are versioned, when products are changed, e.g., new package sizes
  - Old versions are still in the stores, new facts can refer to both the newest and older versions of products
  - Time value for a fact not necessarily between “From” and “To” values in the fact’s Product dimension row

- Unlike changes in Size for a store, where all facts from a certain point in time will refer to the newest Size value

Solution 3

- **Solution 3**: Create two versions of each changing attribute
  - One attribute contains the current value
  - The other attribute contains the previous value
- Consequences
  - Two values are attached to each dimension row
- Pros
  - Possible to compare across the change in dimension value (which is a problem with Solution 2)
    - Such comparisons are interesting when we need to work simultaneously with two alternative values
    - Example: Categorization of stores and products
- Cons
  - Not possible to see when the old value changed to the new
  - Only possible to capture the two latest values

Solution 3: Two versions of Changing Attribute

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	DistrictOld	DistrictNew	…
001		37	37	

versions of an attribute

StoreID	…	ItemsSold	…
001		2000	

StoreID	…	DistrictOld	DistrictNew	…
001		37	73	

StoreID	…	ItemsSold	…
001		2000	
001		2100	

StoreID	…	DistrictOld	DistrictNew	…
001		37	73	

We cannot find out **when** the district changed.

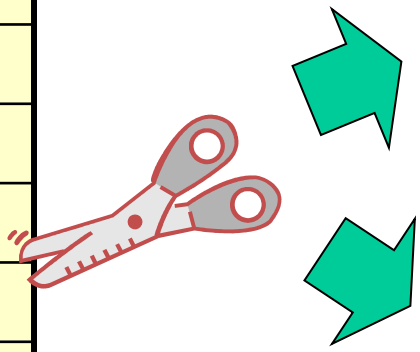
Rapidly Changing Dimensions

- Difference between “slowly” and “rapidly” is subjective
  - Solution 2 is often still feasible
  - The problem is the size of the dimension
- Example
  - Assume an Employee dimension with 100,000 employees, each using 2K bytes and many changes every year
  - Solution 2 is recommended
- Examples of (large) dimensions with many changes: Product and Customer
- The more attributes in a dimension table, the more changes per row are expected
- Example
  - A Customer dimension with 100M customers and many attributes
  - Solution 2 yields a dimension that is too large

Type 4: Dimension Splitting

Customer dimension (original)

CustomerID
Name
PostalAddress
DateOfBirth
…
Gender
NumberOfKids
MaritalStatus
CreditScore
BuyingStatus
Income
Age
…



CustomerID
Name
PostalAddress
DateOfBirth
…

(New) Customer dimension (SCD): relatively static attributes

ProfileID
Gender
NumberOfKids
MaritalStatus
CreditScoreBand
BuyingStatusBand
IncomeBand
AgeBand
…

Profile dimension (not an SCD): often-changing attributes; describing different profiles;

Solution 4

- Solution 4
  - Make a “minidimension” with the often-changing attributes
  - Convert (numeric) attributes with many possible values into attributes with few discrete or banded values
    - E.g., Income group: [0,10K), [10,20K), [20,30K), [30,40K)
    - Why? Any Information Loss?
  - Insert rows for all combinations of values from these new domains
    - What do we do, if there are too many (theoretical) combinations?
  - If the minidimension is too large, it can be further split into more minidimensions
    - Synchronous/correlated attributes should be in the same minidim.

Solution 4 (Changing Dimensions)

- Pros
  - Size of dimension tables is kept down
  - Changes in a customer's profile values do not result in changes in dimensions
- Cons
  - More dimensions and more keys in the star schema
  - Navigation of customer attributes is more cumbersome as these are in more than one dimension
  - Using value groups gives less detail
  - The construction of groups is irreversible

Changing Dimensions - Summary

- Why are there changes in dimensions?
  - Applications change
  - The modeled reality changes
- Multidimensional models realized as star schemas support change over time to a large extent
- Different techniques to handle changes over time at the instance level exist
  - Solution 2 is the most useful

Quiz

Q2.1 When handling a type-1 SCD, we represent a change by

- A) overwriting the changed value with the new value
- B) adding a new row
- C) adding a new column
- D) adding a new table

Q2.2 When handing a type-2 SCD, we represent a change by

- A) overwriting the changed value with the new value
- B) adding a new row
- C) adding a new column
- D) adding a new table

Q2.3 When handling a type-3 SCD, we

- A) add a column whenever there is a change
- B) add a row whenever there is a change
- C) have two columns to represent the changing attribute

Outriggers

- Consider again the Customer and Profile dimensions from Solution 4
- If we want to keep track of the *current* profile, we could include the current Profile attributes in Customer – but this is not a good solution … *Why?*
- It makes better sense to keep the Profile attributes in their own dimension and reference it from the Customer dim.
- A dimension which is referenced by another dimension is called an outrigger
- A dimension may act both as a "normal" dimension and an outrigger
  - For example Date used for Order Date and referenced from Customer to track birthday/first order/…

Degenerate Dimensions

- Consider a DW for analyzing orders of products
  - Granularity: The individual line items
  - Dimensions: Customer, Product, and Date
- How do we see what line items belong together?
- We need the dimension Order
  - Its only attribute is the order number
  - (and we do not need a surrogate key in this case)
- We call a dimension consisting of a single numerical identifier for a degenerate dimension
- How do we represent this in ROLAP?
- In ROLAP, we do not need a dimension table for a degenerate dimension
  - Pointless – we would have a foreign key in the fact table pointing to a single identifier
  - Instead, we add the order number directly to the fact table

Junk Dimensions

- After obvious dimensions are identified, a small amount of unrelated attributes are often left over
- For the previous example with orders, we might have Ship Mode, Promoted, and Wrapping left over
- We don't want to drop them or make a separate dimension for each of them
  - *Why not?*

- Instead, we can combine them into a single dimension
- We call such a dimension with combinations of unrelated attributes a junk dimension
- Junk dimensions should only be used to group flags and low-cardinality attributes

Role-playing Dimensions

- A dimension can be used multiple times in a cube with different roles
- For example:
  - A Date dimension to represent Order Date and Ship Date
  - A Customer dimension to represent Bill To and Send To

- We do not create two (or more) physical dimensions
- Instead, we create the dimension once and use it as a role-playing dimension

- In a ROLAP setting, we create the dimension table once and a view for each role

Quiz

Q2.4 An outrigger is

- A) A special kind of fact table
- B) A dimension which is referenced by another dimension
- C) A dimension which is represented by several tables

Q2.5 A degenerate dimension is

- A) a dimension that is not represented at all
- B) a dimension that only has a single attribute of any type
- C) a dimension that only has some integer attributes
- D) a dimension that only has a single attribute which is a number

Q2.6 The dimension "Ticket" only has a single attribute which takes values such as "EZ123456", "123XX234", and "N/A". It would be best to

- A) consider this as a degenerate dimension and represent the values in the fact table
- B) represent the dimension by means of an ordinary dimension table with a surrogate key
- C) try to avoid representing the values at all

Q2.7 A junk dimension

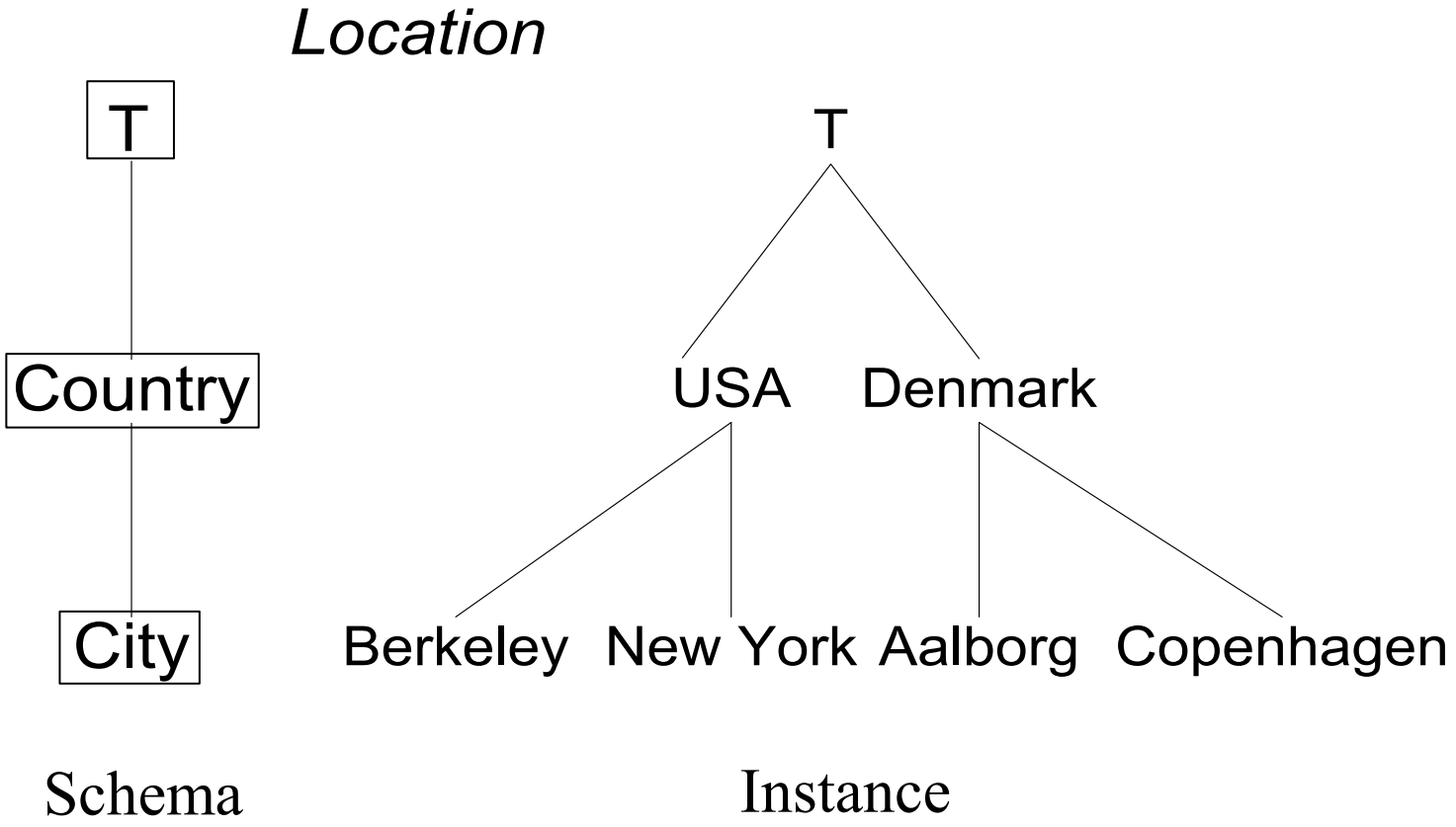
- A) does not have a primary key
- B) should use one of its grouped attributes as its primary key
- C) should have a surrogate key

- Advanced Hierarchies

Advanced Hierarchies

- So far, we assumed that a hierarchy was
  - balanced
    i.e., in an instance, all leaves belong the schema's lowest level
  - covering
    i.e., in an instance, every dimension value from a non-T level L belongs to a dimension value at the level immediately above L in the schema. In other words, we don't skip a level
  - strict
    i.e., in an instance, no dimension value has more than one parent
- This means that an instance forms a balanced tree
- We now consider hierarchies where this is not the case…

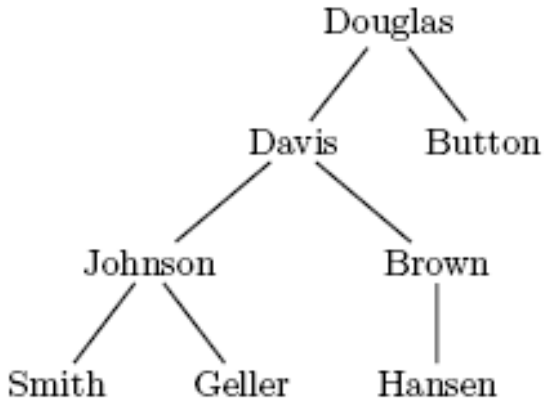
Hierarchy Example



Parent-Child Hierarchies

- Consider an Employee dimension.
  - For each employee, we keep track of his/her manager
  - Managers do also have managers …
  - It is not practical to have a fixed number of levels in the hierarchy
- In a parent-child hierarchy, there is a single level in the schema ….
- … but in instances, we allow a dimension value (from the single level) to have a parent (from the same level)
- We do not allow cycles
- In effect, an instance can have any number of levels

Parent-Child Hierarchies



How do we represent this in ROLAP?

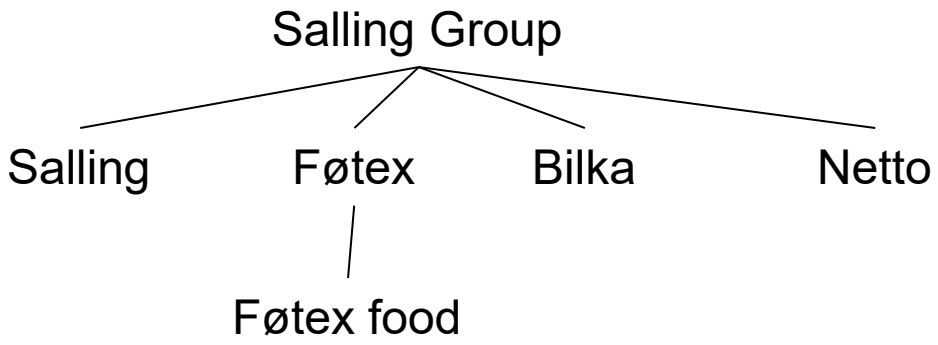
We can use a table with a FK to itself:

EmployeeID	**Employee**	**ManagerID**
1	Douglas	NULL
2	Davis	1
3	Johnson	2
4	Smith	3
5	Geller	3
6	Brown	2
7	Hansen	6
8	Button	1

This is “understood” by some OLAP engines.
But we cannot traverse the hierarchy with *simple* SQL…

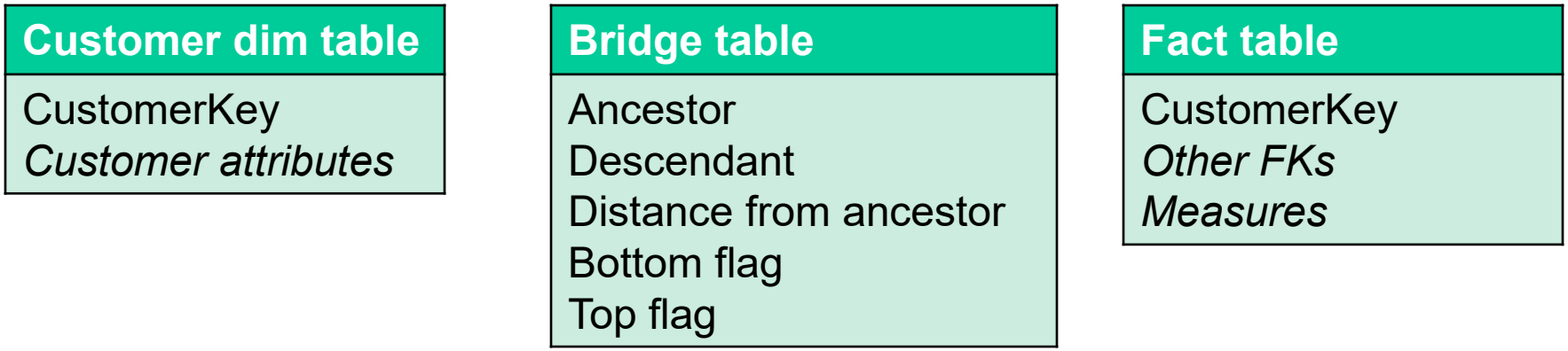
Another ROLAP Solution to Variable Depth Hierarchies

- Assume you sell stuff to commercial customers that have subsidiary relations
- You want to be able to find revenues both for each individual customer and for families
- We assume the following structure for Salling Group:



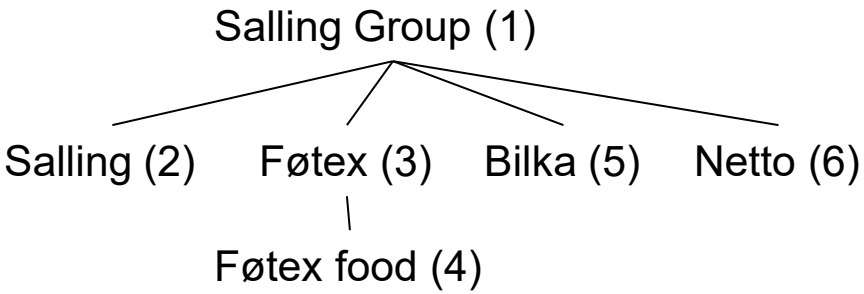
- We can model this by using a bridge table

Bridge Table



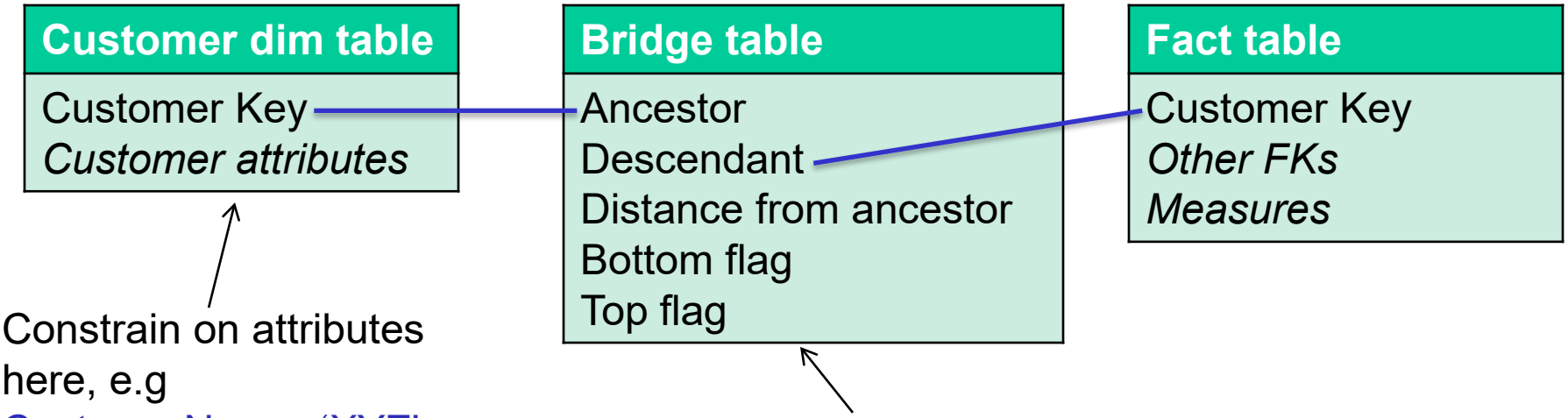
- When querying, we can ignore the bridge table and just join the customer dimension table and the fact table
  - In this way, we can see what we sold to each individual customer
- To be able to navigate in the organizational hierarchy, we use the bridge table
  - The bridge table holds a row for each path between an ancestor and a descendant below it (including the paths of length 0)

Content of the Bridge Table



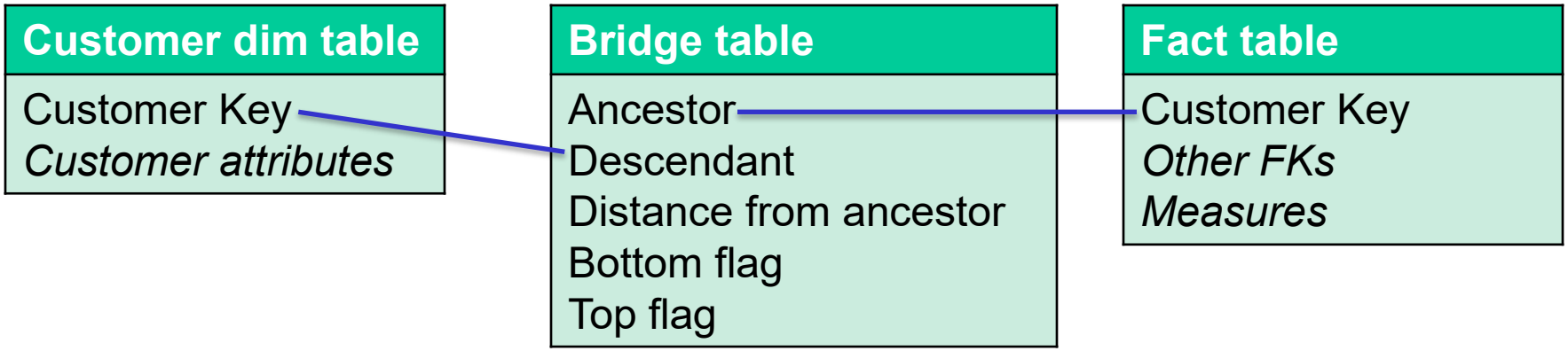
Ancestor	Descendant	Distance	Top	Bottom
1	1	0	Yes	No
1	2	1	No	Yes
1	3	1	No	No
1	4	2	No	Yes
1	5	1	No	Yes
1	6	1	No	Yes
2	2	0	No	Yes
3	3	0	No	No
3	4	1	No	Yes
4	4	0	No	Yes
5	5	0	No	Yes
6	6	0	No	Yes

Descending the Hierarchy




```
SELECT C.CustomerName, SUM(F.measure)
FROM Customer C, Bridge B, Fact F
WHERE C.CustomerKey = B.Ancestor AND F.CustomerKey = B.Descendant
AND C.CustomerName = 'XYZ'  AND B.Distance > 0
GROUP BY C.CustomerName
```

Ascending the Hierarchy



Unbalanced Hierarchies

- In an unbalanced hierarchy, a dimension value belonging to a non-bottom level can have no children
  - Dimension values for the lowest level are missing
- Recall the exercise fra Ch. 2 about the magazine shops
  - Assume that big shops are subdivided into departments
  - Small shops do not have departments

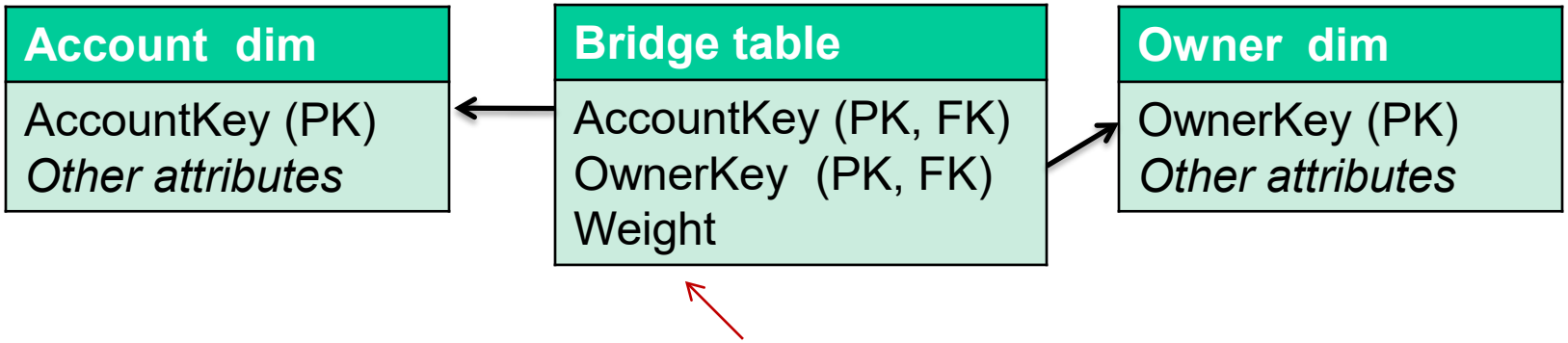
- We can make the unbalanced hierarchy balanced by adding placeholder values

Non-Covering Hierarchies

- In a non-covering hierarchy, instances can skip a level between the leaves and the root
- In the schema City → State → Country, we have a problem with Danish cities, but not with German cities
- We can insert a *placeholder value* for Danish cities
  - Represents a “fake” state in Denmark
  - The placeholder can be without a name or can be named as its parent (“Denmark” in this example)

Non-Strict Hierarchies

- In a non-strict hierarchy, a dimension value can have more than one parent
- For example, an account may have more than one owner
- In ROLAP, we cannot use a single column to represent the level (or the FK to the level) then
- Instead, we can use a bridge table



The bridge table can have properties.
Why do we need the weight property?
What should the weights add up to for a given account?

Multiple Hierarchies

- A dimension can have more than one hierarchy
  - Hierarchies may share some levels
  - Date: Normal calendar and fiscal calendar
- If so, we say that the dimension has *multiple hierarchies*
- Often, it does not make sense to use multiple hierarchies from one dimension in a single analysis
  - Calendar quarters on the x axis and fiscal quarters on the y axis would not make sense…
  - The hierarchies have the same analytical purpose: Grouping of days into coarser time units

Parallel Hierarchies

- What about the Genre and Publisher hierarchies from the book store example?
- Sometimes, it *does* make sense to use two hierarchies from the same dimension simultaneously
- When they have different analytical purposes…
- We then refer to them as *parallel hierarchies*

Summary

- Slowly Chainging Dimensions (SCDs)
  - Type 1
  - Type 2
  - Type 3
  - Type 4
- Special dimensions:
  - Outriggers
  - Degenerate dimensions
  - Junk dimensions
  - Role-playing dimensions
- Hierarchies
  - Variable-depth hierarchies
  - Unbalanced, non-covering, non-strict hierarchies
  - Multiple and parallel hierarchies

Exercises

- **JPT Section 3.5**

- (And F-klub case session tomorrow)