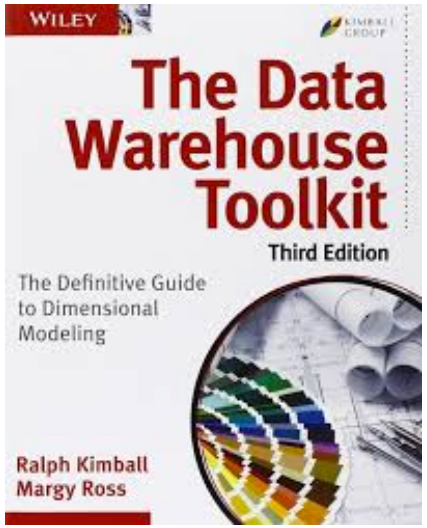
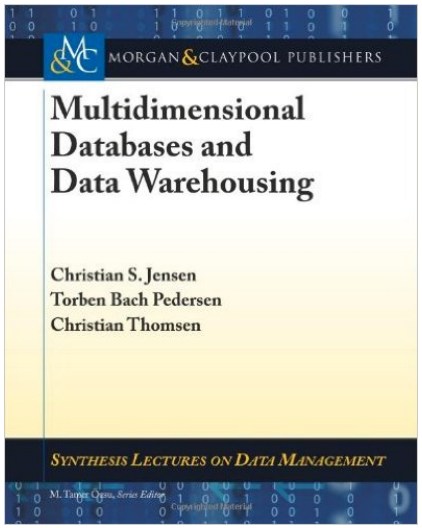
Resources

these slides cannot replace the textbooks by any means !

- Entrepôts de données, guide pratique de modélisation dimensionnelle. R.Kimball, M.Ross





- Multidimensional databases and Data Warehousing C.S. Jensen, T.B.Pedersen and C.Thomsen

Plan

- Modelling Hierarchies and N:M relationships
- Evolving data
  - Techniques for handling updates
- Evolving schema
  - What to do with new data-sources, dimensions, or attributes

HIERARCHIES

Sales

- Find total amount of sales for employee ‘A’

Sales

DimEmployee

EmpID	Name
1	A
2	B
3	C
4	D
5	E
6	F
7	G

FactSales

EmpID	Amount
1	100
1	34
3	821
4	4
2	12
5	5
4	6

Sales

DimEmployee

EmpID	Name
1	A
2	B
3	C
4	D
5	E
6	F
7	G

FactSales

EmpID	Amount
1	100
1	34
3	821
4	4
2	12
5	5
4	6

Sales

EmpID	Sum(Amount)
1	134

Sales

- Find total amount of sales for employee ‘A’

```
SELECT  EmpID, SUM(Amount)
  FROM  Employee E, Sales F
 WHERE  E.EmpID = F.EmpID
   AND  E.Name = 'A'
GROUP BY EmpID
```

Hierarchies

- Find total amount of sales for people “below” **manager** ‘A’

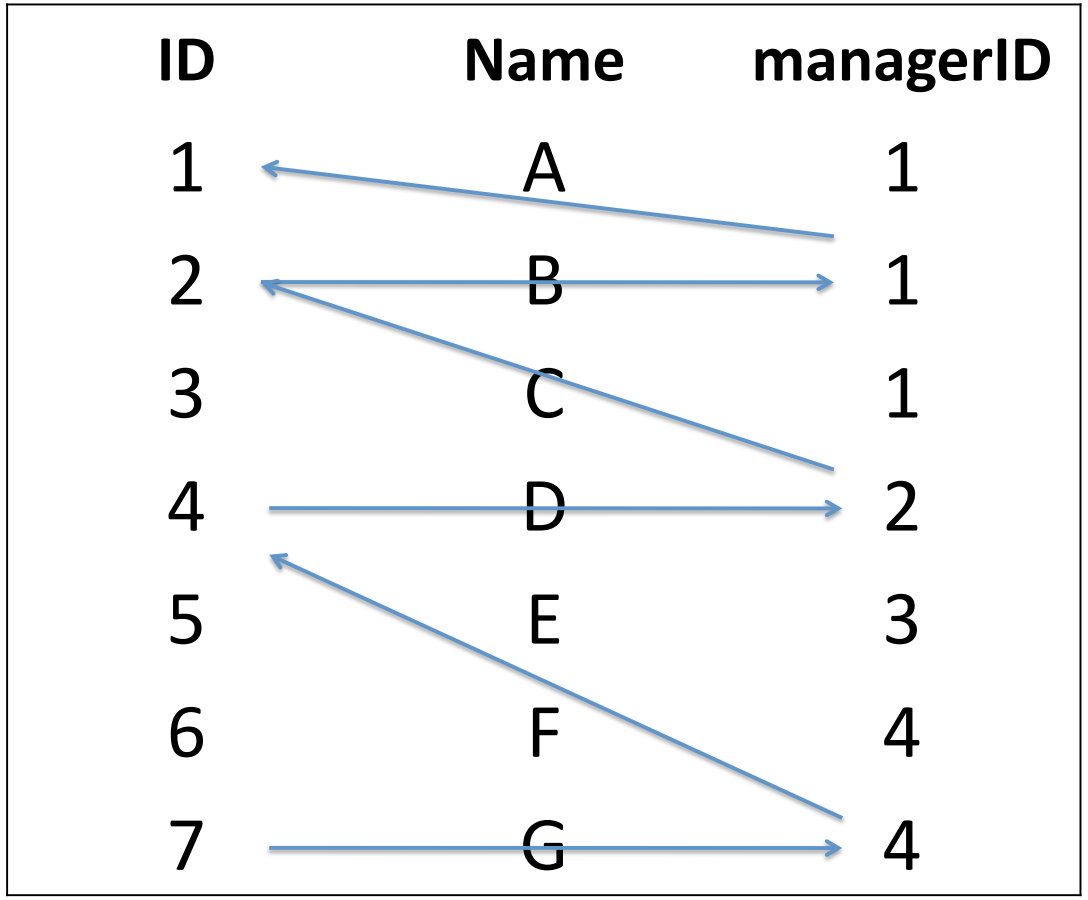
Hierarchies

`EmployeeDim(ID,name,managerID)`

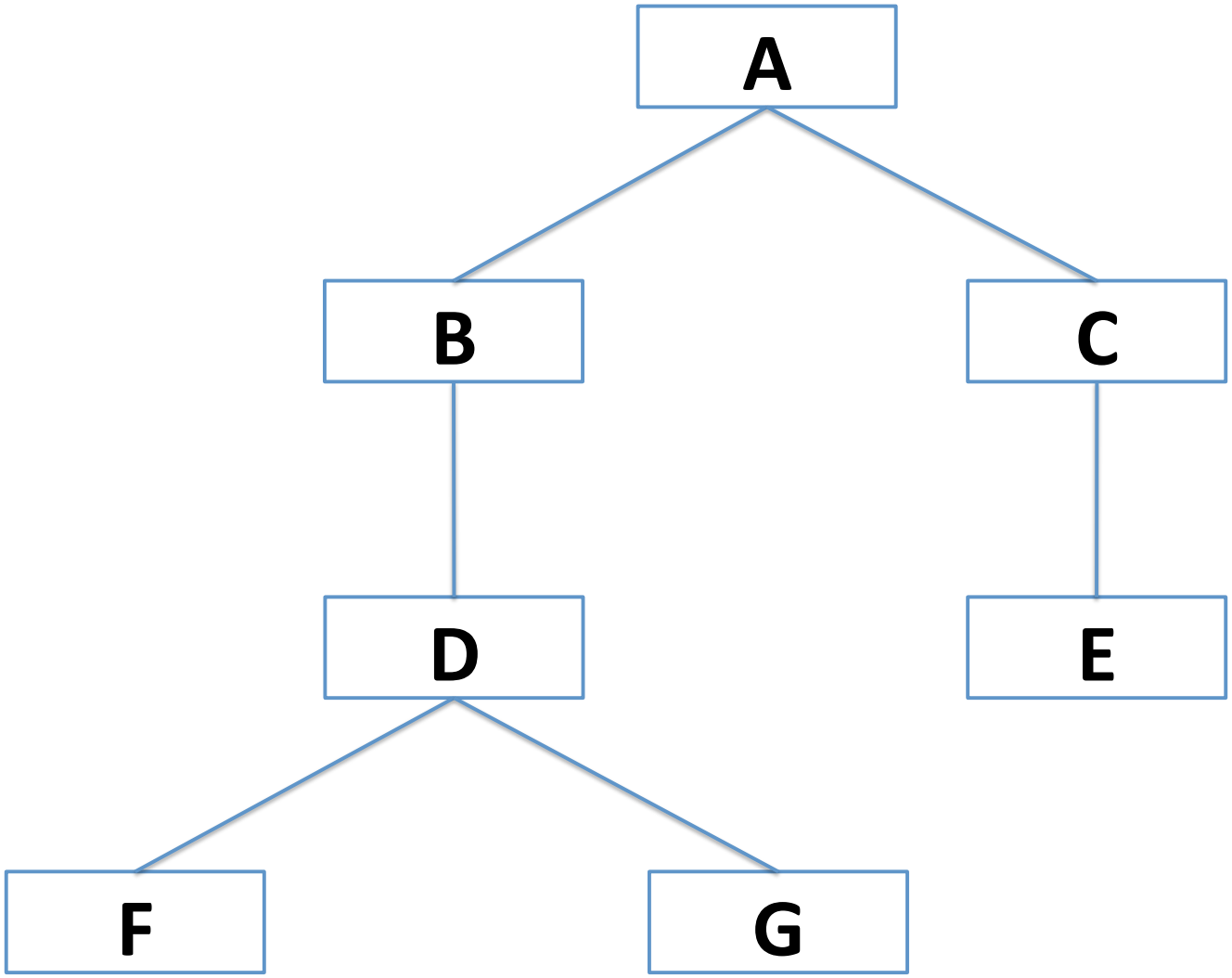
ID	Name	managerID
1	A	1
2	B	1
3	C	1
4	D	2
5	E	3
6	F	4
7	G	4

Hierarchies

`EmployeeDim(ID,name,managerID)`



Hierarchies

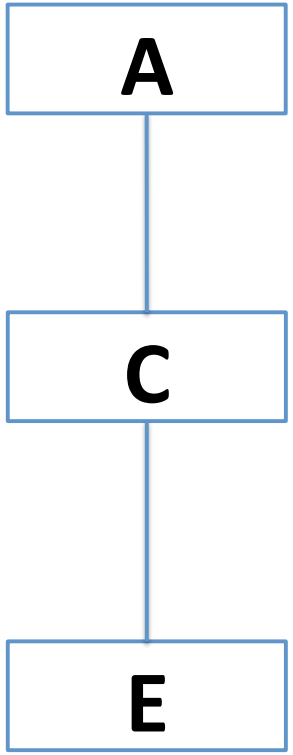


Hierarchies

- Computing the transitive closure on the fly is not a good idea

- Solution : introduce an (intermediate) bridge table

Hierarchies



Bridge Table (for A,C,E only)

Manager	Employee	Distance	Bottom flag	Top flag
A	A	0		***true***
A	C	1		
A	E	2	(false	otherwise)
C	C	0		
C	E	1		
E	E	0	***true***	

- #rows worst-case : quadratic in the #rows of DimEmpl

Bridge Table

Manager	Employee	Distance	Bottom flag	Top flag
A	A	0		***true***
A	C	1		
A	E	2	(false	otherwise)
C	C	0		
C	E	1		
E	E	0	***true***	

Bridge Table : Usage

- Find total amount of sales for manager ‘A’

```
WHERE E.EmpID = Bridge.Manager
AND   Bridge.Employee = F.EmpID
```

Bridge Table : Usage

- Find total amount of sales for manager ‘A’

```
SELECT E.EmpID, SUM(F.Amount)
FROM Employee E, Bridge, Sales F
WHERE E.EmpID = Bridge.Manager
AND   Bridge.Employee = F.EmpID
AND   E.Name = 'A'
GROUP BY E.EmpID
```

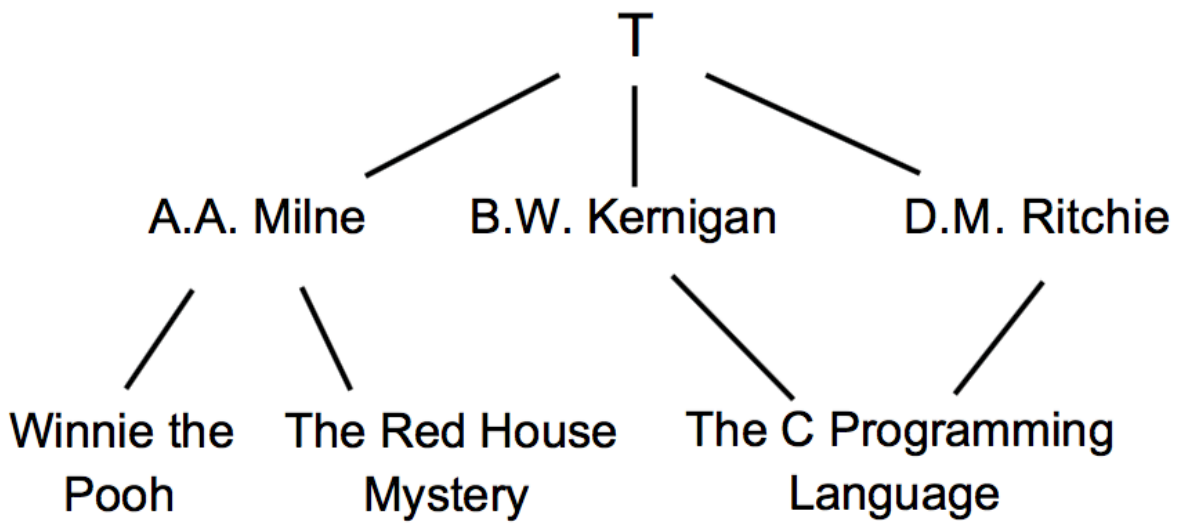
Sales

ID	Sum(F.Amount)
A	960

N:M RELATIONSHIPS

N : M relationships

- authors can write many books
- books can have many authors



How much did a writer sold ?

FactBookSales

book	author	unit_price	copies
Winnie	Milne	15	10.000.000
Red House	Milne	20	50.000
C	Kernigan	30	500.000
C	Ritchie	30	500.000

How much did a writer sold ?

FactBookSales

book	author	unit_price	copies
Winnie	Milne	15	10.000.000
Red House	Milne	20	50.000
C	Kernigan	30	500.000
C	Ritchie	30	500.000

How much did a writer sold ?

author	copies
Milne	10.050.000
Kernigan	500.000
Ritchie	500.000

How much did a book sold ?

FactBookSales

book	author	unit_price	copies
Winnie	Milne	15	10.000.000
Red House	Milne	20	50.000
C	Kernigan	30	500.000
C	Ritchie	30	500.000

How much did a book sold ?

FactBookSales

book	author	unit_price	copies
Winnie	Milne	15	10.000.000
Red House	Milne	20	50.000
C	Kernigan	30	500.000
C	Ritchie	30	500.000

How much did a book sold ?

author	copies
Winnie	10.000.000
Red House	50.000
C	**1.000.000**

Bridge Table

Book	Author	Order	Percentage
Winnie	Milne	1	1
Red House	Milne	1	1
C	Kernigan	1	0.5
C	Ritchie	2	0.5

Bridge table allows to deal also with the **order** of entities as well as their **contribution** (in percentage).

Bridge Table Join with Fact Table

Book	Author	Order	Percentage	unit_price	copies
Winnie	Milne	1	1	15	10.000.000
Red House	Milne	1	1	20	50.000
C	Kernigan	1	0.5	30	500.000
C	Ritchie	2	0.5	30	500.000

How much did a book sold ?

```
SELECT SUM(copies* percentage)
FROM   BookSales, Bridge
WHERE  book = Bridge.book
AND    Bridge.author = author
GROUP  BY book
```

How much did a book sold ?

author	copies
Winnie	10.000.000
Red House	50.000
C	**500.000**

TECHNIQUES FOR HANDLING VERY LARGE TABLES

Fact tables

- are not the only tables that can grow large

Customer (User) Dimension

- Most challenging dimension for any DW

  Can be :

- Extremely **deep** : millions of rows
- Extremely **wide** : hundreds of columns
- **Frequently Updated**

Customer (User) Dimension

DIMENSION ATTRIBUTE	EXAMPLE VALUES
Salutation	Ms.
Informal Greeting Name	Jane
Formal Greeting Name	Ms. Smith
First and Middle Names	R. Jane
Surname	Smith
Suffix	Jr.
Ethnicity	English
Title	Attorney
Street Number	123
Street Name	Main
Street Type	Road
Street Direction	North West
Post Box	2348
Suite	100A
City	Kensington

Customer (User) Dimension

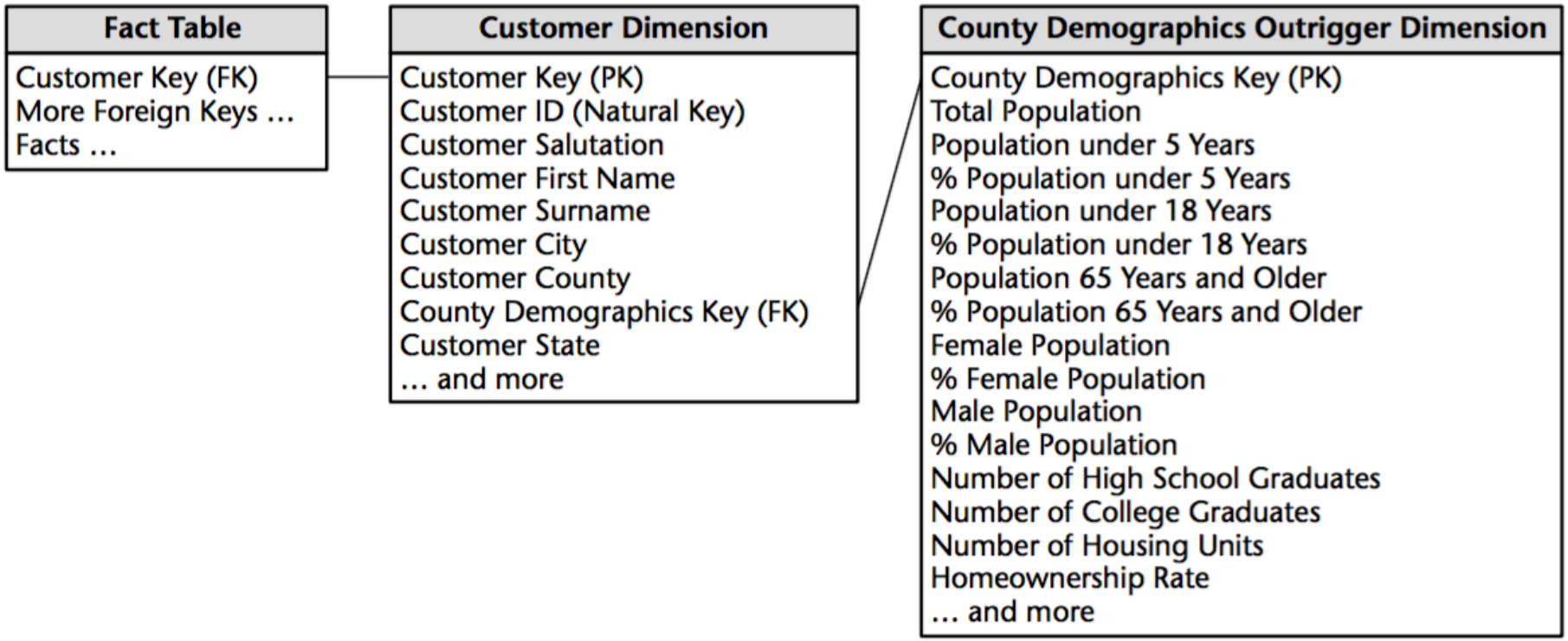
- Marketing agency maintains **over ~3,000 attributes** about its customers
- Retailers, credit card companies, and government agencies have **~100 million customers**
- Telecommunication company Bouygues **~10 million** customers in France
- Very difficult structures to maintain and to query

Dealing with Very Large Tables

We have no choice :

- Normalization for dimensions (Snowflaking)
- Partitioning for dimensions
  (by rows, by columns, hybrid)
- Partitioning for facts (by rows)

Normalization (Snowflaking)



Snowflaking is legal, but it should be exceptionally used to handle very large dimensions.

Here, the demographics of the customer's country

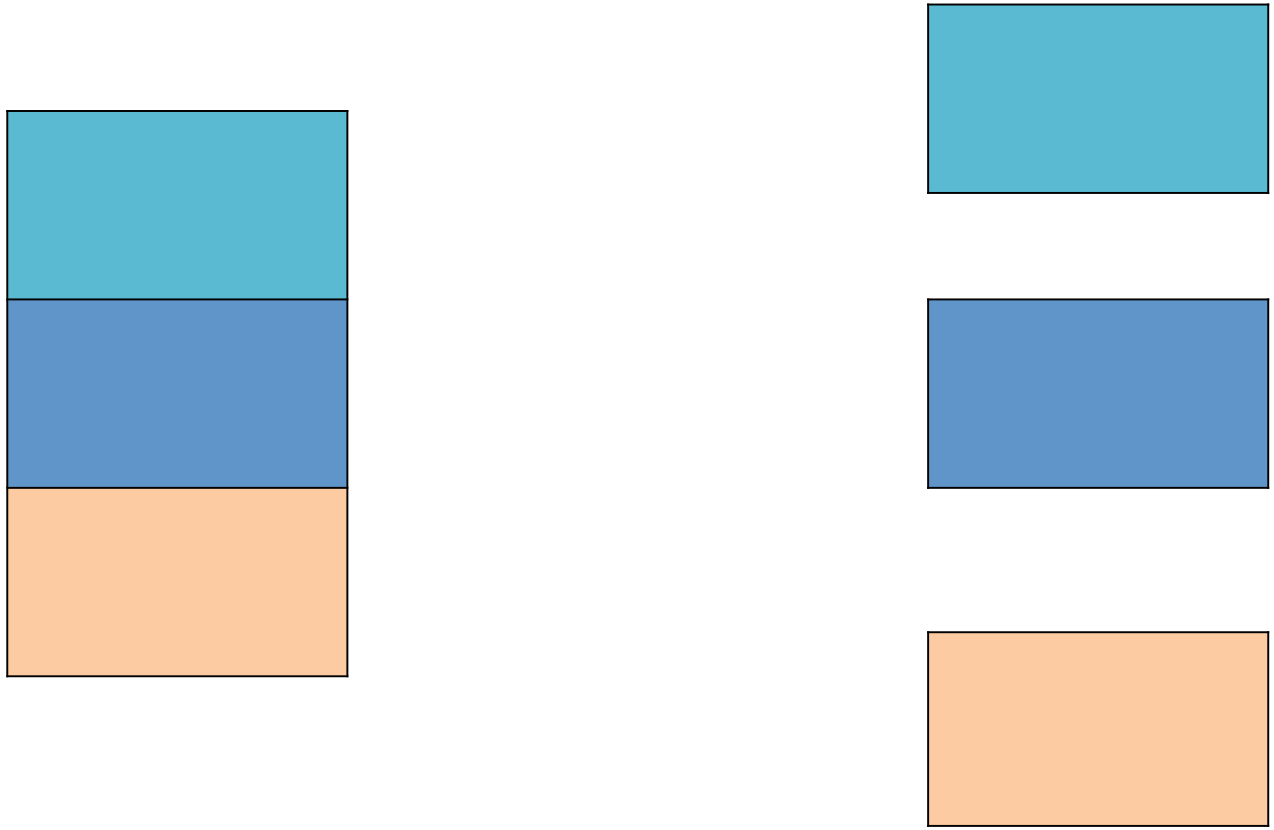
Partitioning

- Splitting one large table into several small

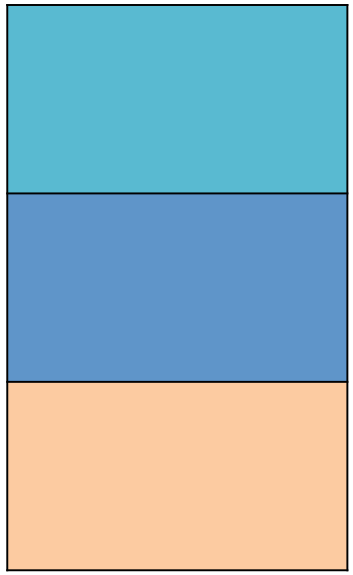
  Partitioning **!=** Snowflacking
  (does not use normalization)

- Still have to maintain the minimum number of joins

1) Partitioning by-Row



1) Partitioning by-Row (Dimensions)

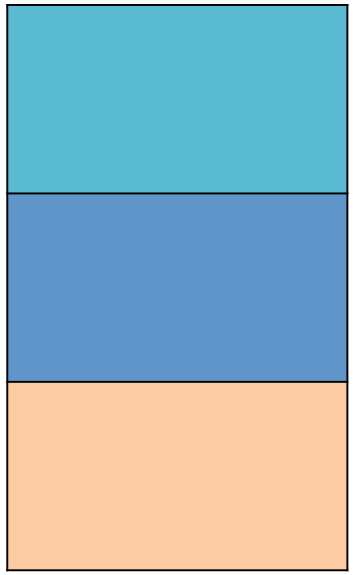


client type 1

client type 2

client type 2

1) Partitioning by-Row (Facts)



time window 1

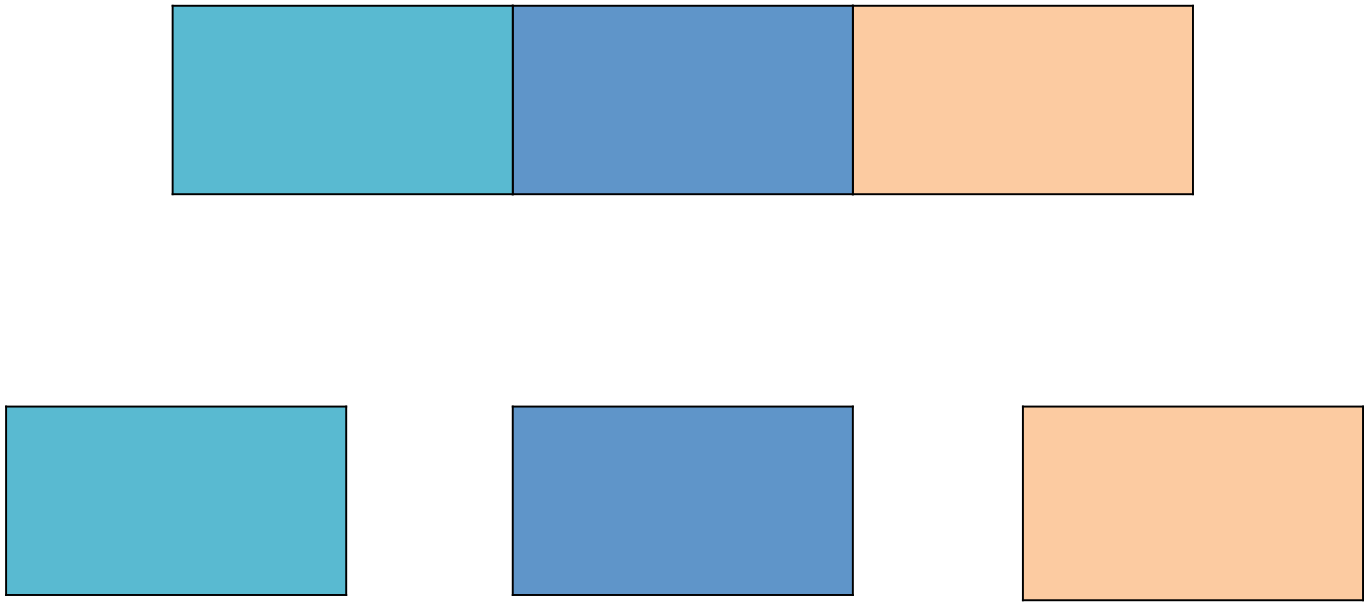
time window 2

time window 3

1) Partitioning by Row

- The large table is split in two or more tables having fewer rows but the same number of columns
- Splitting is based on values in one or more columns
  - Time dimension
  - Type of customer profile, product
  - Table size (when is not possible to use dimensions)

2) Partitioning by Column



2) Partitioning by Column

columns
never updated

columns
rarely updated

columns
frequently updated

2) Partitioning by Column

- The large table is split in two or more tables having fewer columns but keeping the same number of rows

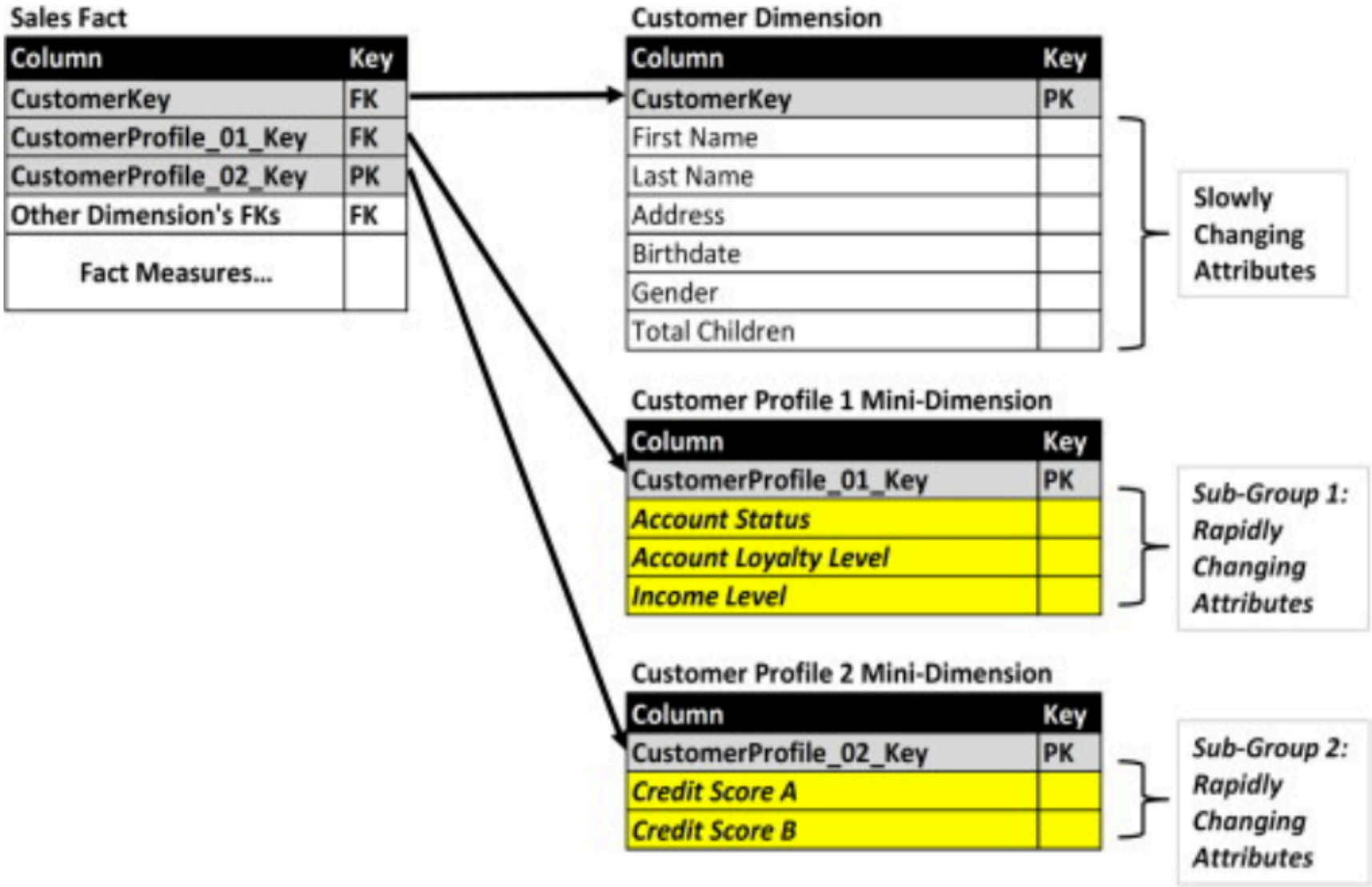
2) Partitioning by-Column

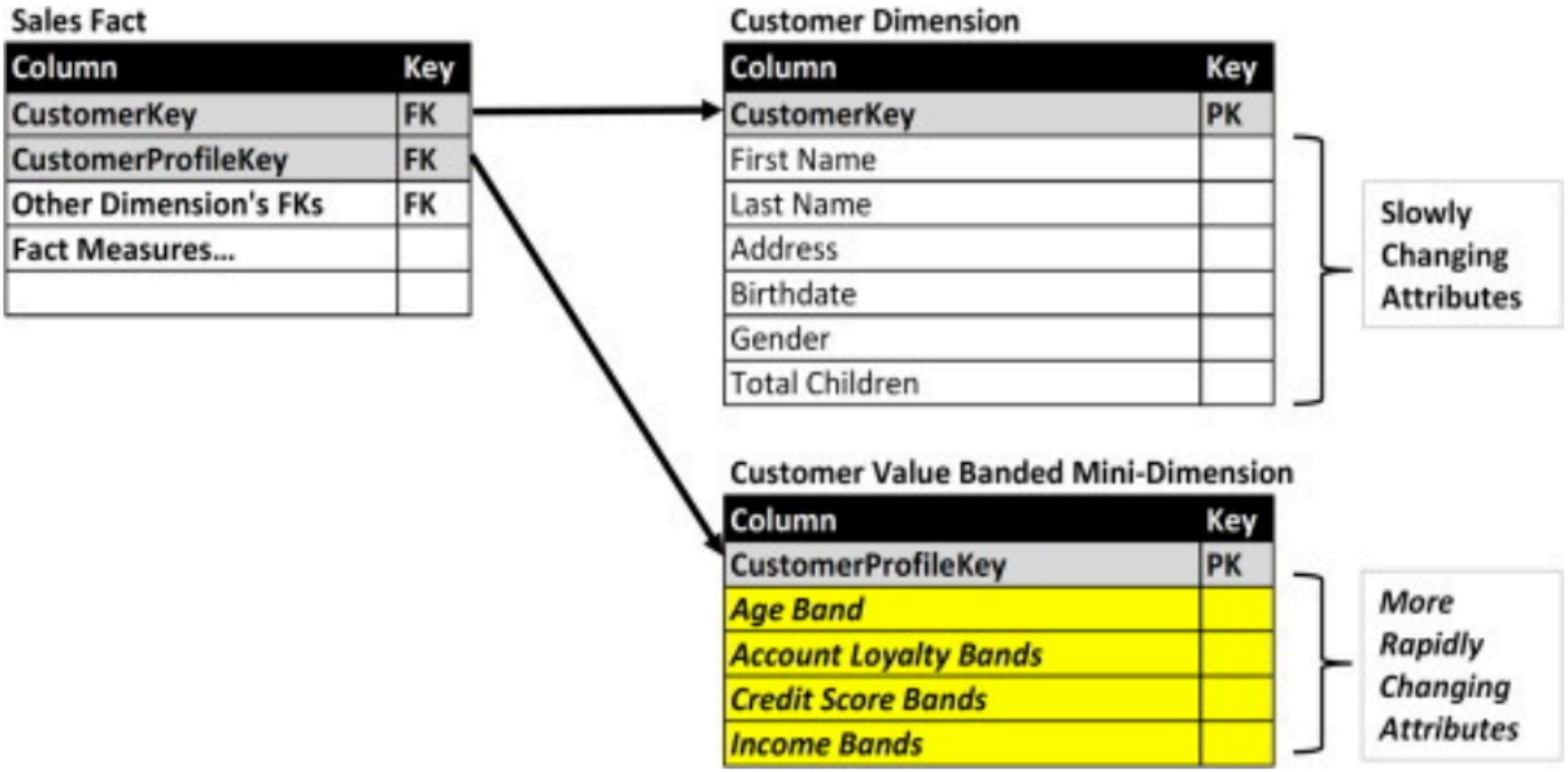
- Divide columns in three categories and try to pack them together (if possible) :

never updated (birthdate)

rarely updated (marital status)

frequently updated (client-age, -income)


Sales Fact
Column
Key
CustomerKey
FK
CustomerProfile_01_Key
FK
CustomerProfile_02_Key
PK
Other Dimension's FKs
FK
Fact Measures...
Customer Dimension
Column
Key
CustomerKey
PK
First Name
Last Name
Address
Birthdate
Gender
Total Children
Slowly Changing Attributes
Customer Profile 1 Mini-Dimension
Column
Key
CustomerProfile_01_Key
PK
Account Status
Account Loyalty Level
Income Level
Sub-Group 1: Rapidly Changing Attributes
Customer Profile 2 Mini-Dimension
Column
Key
CustomerProfile_02_Key
PK
Credit Score A
Credit Score B
Sub-Group 2: Rapidly Changing Attributes


Sales Fact
Column
Key
CustomerKey
FK
CustomerProfileKey
FK
Other Dimension's FKs
FK
Fact Measures…
Customer Dimension
Column
Key
CustomerKey
PK
First Name
Last Name
Address
Birthdate
Gender
Total Children
Slowly Changing Attributes
Customer Value Banded Mini-Dimension
Column
Key
CustomerProfileKey
PK
Age Band
Account Loyalty Bands
Credit Score Bands
Income Bands
More Rapidly Changing Attributes

3) Hybrid Partitioning : byRow+ byColumn

- Example

- Split all the attributes that don’t change over time (e.g birthday of a customer)

  from all attributes that may change (like a phone number)

- Then split vertically choosing other criteria

UPDATING DIMENSIONS

Updates

“due to dynamic nature of reality,
the modeled reality, as well as the uses of the data warehouse, change over time’’

Naïve answer

- Well, no problem.
- Almost no deletion is made on the DW.
- In 90% of the cases we merely do insert-updates.

- for record-updating, replace old with new values.

- In reality : we have frequent and unfrequent updates on dimensional tables

(Unfrequent) Updates

Dimensional Tables

Unfrequent Dimension Update

- The rate of a movie can change during time
- The department of a product can change over time
- The status of a client may change over time

Unfrequent Dimension Update

BookID	Book	Rating	Genre
7493	Tropical Food	4 stars	Children's books
9436	Winnie the Pooh	5 stars	Children's books
9948	Gone With the Wind	4 stars	Fiction
9967	Italian Food	4 stars	Cooking

Book (dimension)

- Rating of "Gone With the Wind" may drop to "3 stars"
- Genre of "Italian Food" becomes "Mediterran Cooking"
- So what ? These rows are already being referred to by fact table rows with old sales !

Unfrequent Dimension Update

- Up to now, we pretended dimensions are independent
- In particular, independent of time
- Unfortunately, this is not the case in the real world

Unfrequent Dimension Update

- Make every dimension time dependent : space waste

Rather,

- take advantage of the fact that most dimensions are nearly constant over time

- make small changes when needed

Strategies for Updating Dimensions

1. Overwrite the Value
2. Row Versioning
3. Duplicate Attributes

Type 1: Overwrite the Value

- Brutal forcing : overwrite the old value with new one

BookID	Book	Rating	Genre
7493	Tropical Food	4 stars	Children's books
9436	Winnie the Pooh	5 stars	Children's books
9948	Gone With the Wind	3 stars	Fiction
9967	Italian Food	4 stars	Cooking

Book (dimension)

(+) Fact table rows can now refer to updated dimensions

(-) The datawarehouse probably has now incorrect data

Type 1: Overwrite the Value PRO

- Easy to implement (even too easy…)

- There may be cases where the changes are corrections or inaccuracies introduced are uninportant
  - *genre of “Italian Food”*

    becomes “Mediterran Cooking”

Type 1: Overwrite the Value CONS

- this approach basically ignores fundamental problem

- we loose the history of changes

Type 2 : Row Versioning

BookID	Book	Rating	Genre	ValidFrom	ValidTo	Newest	Version
7493	Tropical Food	4 stars	Children's books	2006-03-01	2008-12-31	No	1
9436	Winnie the Pooh	5 stars	Children's books	2000-01-01	9999-12-31	Yes	1
9948	Gone With the Wind	4 stars	Fiction	1999-06-01	2008-10-15	No	1
9967	Italian Food	4 stars	Cooking	2003-04-05	2009-05-01	No	1
9995	Gone With the Wind	3 stars	Fiction	2008-10-16	9999-12-31	Yes	2
10100	Tropical Food	4 stars	Cooking	2009-01-01	9999-12-31	Yes	2
11319	Italian Food	4 stars	Mediterranean cooking	2009-05-02	9999-12-31	Yes	2

Book (dimension)

- Primary technique for accurately tracking changes
- Partitioning by column to minimize overhead

Type 3: Duplicate Attributes

- Keep two values (current,previous) for each attribute

BookID	Book	Rating	OldRating	Genre	OldGenre
7493	Tropical Food	4 stars	4 stars	Cooking	Children's books
9436	Winnie the Pooh	5 stars	5 stars	Children's books	Children's books
9948	Gone With the Wind	3 stars	4 stars	Fiction	Fiction
9967	Italian Food	4 stars	4 stars	Mediterranean cooking	Cooking

Book (dimension)

- Utilized to analyze data across changes.

- Example : difference in the distribution of sales across two classifications / internal-organizations / districts

Type 3: Duplicate Attributes

- Keep two values (current,previous) for each attribute

BookID	Book	Rating	OldRating	Genre	OldGenre
7493	Tropical Food	4 stars	4 stars	Cooking	Children's books
9436	Winnie the Pooh	5 stars	5 stars	Children's books	Children's books
9948	Gone With the Wind	3 stars	4 stars	Fiction	Fiction
9967	Italian Food	4 stars	4 stars	Mediterranean cooking	Cooking

Book (dimension)

- Can reconstruct only a **partial** history

- Usually limited by only 2 values, but can grow to N

Strategies for Updating Dimensions

1. Overwrite the Value
2. Add a Version Column
3. Duplicate Attributes

Can be also combined, depending on the needs.

2) Partitioning by Column

columns
never updated

columns
rarely updated

columns
frequently updated

Frequently Updated Dimensions

- Break off rapidly changing attributes into one or more separate dimensions

- Then use two foreign keys
  —one for the primary dimension table
  —another for the rapidly changing attributes

- This is called a **mini-dimension**

Datawarehouse

Schema Evolution

What can happen if

- We want to integrate a new data-source ?
- We want to add new dimensions and new facts ?
- We want to add new dimensional attributes ?
- We want to change the values of dimensional attributes

What can happen if

- We want to integrate a new data-source ?
  - May have a granularity problem
- We want to add new dimensions and new facts ?
  - This is ok, add « default » values for existing row-facts
- We want to add new dimensional attributes ?
  - Introduce « default » values for existing rows
- We want to change the values of dimensional attributes
  - Techniques seen before

New data source

- Addition of a completely new data source involving existing dimensions as well as **unexpected new dimensions**.
- Almost always, a new source of data has its own granularity and dimensionality, so we create a new fact table.
- **We should avoid force-fitting new measurements into an existing fact table of consistent measurements.**
- The existing applications will still work because the existing fact and dimension tables are untouched.

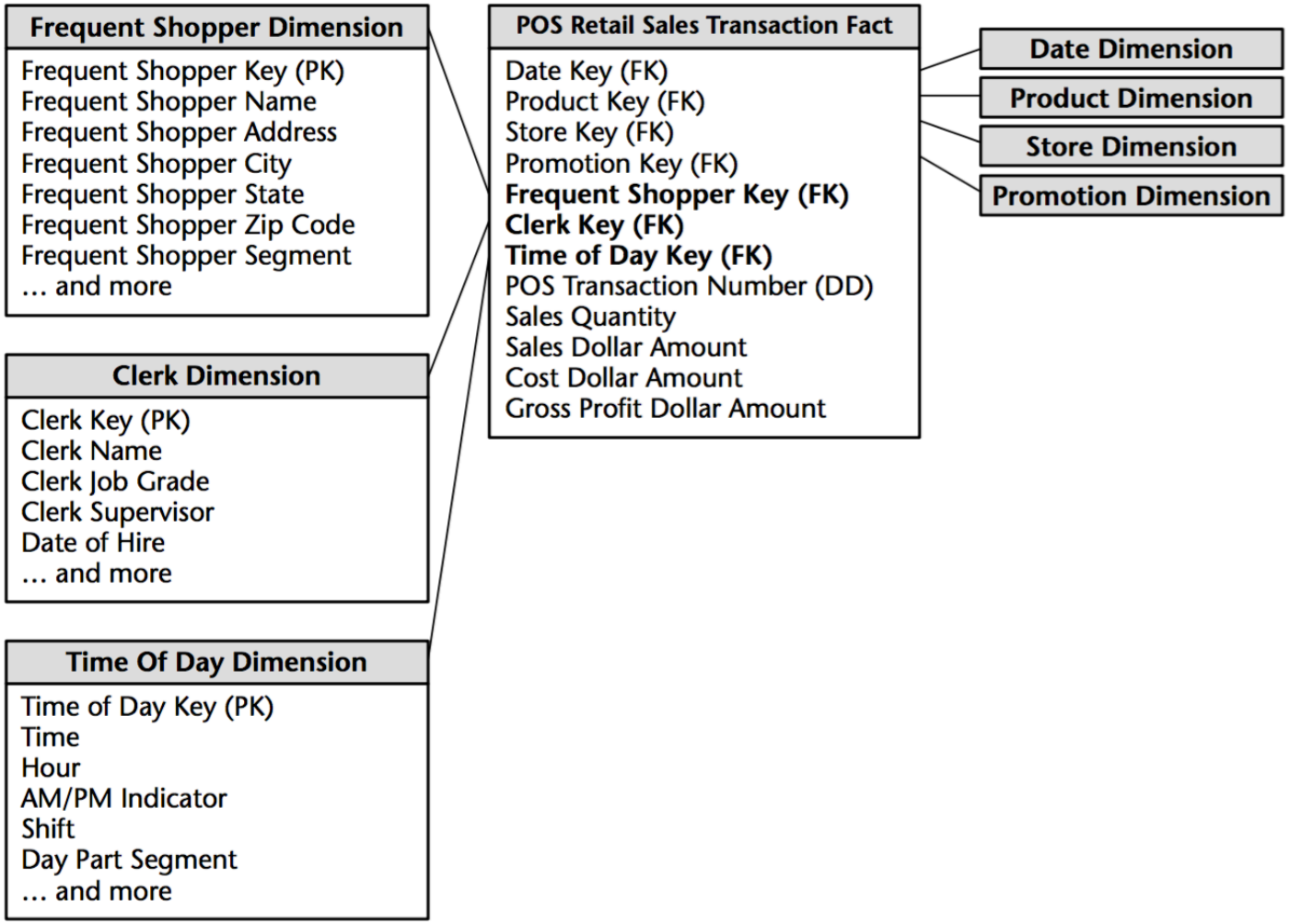
Adding new dimensions is easy

Study case : how to extend the model to include a **frequent shopper** program ?

1. Create a *frequent shopper* (= customer) dimension table and add another foreign key in the fact table.

2. substitute a shopper key corresponding to a “Prior to Frequent Shopper Program” description on our historical fact table rows

Adding new dimensions is easy



Schema Extensibility

- Schema gracefully extends because data is modeled at its most granular level
- If the grain would be daily retail sales (transactions summarized by day, store, product, and promotion) difficult to incorporate the frequent-shopper, time-of-day, or employee dimensions.
- **Premature summarization inherently limits extensibility** because the additional dimensions often don’t apply at the higher grain.

New dimension attributes

- Can add new textual descriptors of a product

- If the new attributes are available only after a specific point in time, then “Not Available” or its equivalent should be populated in the old dimension records.

New dimensions

- As we just illustrated we can add a dimension to an existing fact table by adding a new foreign key field and populating it correctly with values of the primary key from the new dimension.

New measured facts : Two Cases

- Easy case : new facts available in the same measurement event and at the same grain as the existing facts. (ALTER TABLE)

- Otherwise, if facts occur naturally at a different grain, **define a new table**.

- **It is almost always a mistake to mix grains in the same fact table.**