LOGICAL DESIGN: CHANGING DIMENSIONS

Slowly changing dimensions

- TYPE 1 (overwriting the history)

Overwrite the value

- TYPE 2 (preserving the history)

Add a dimension row

- TYPE 3 (preserving one or more versions of history)

Not recommended

Add new attributes

Fast changing dimensions

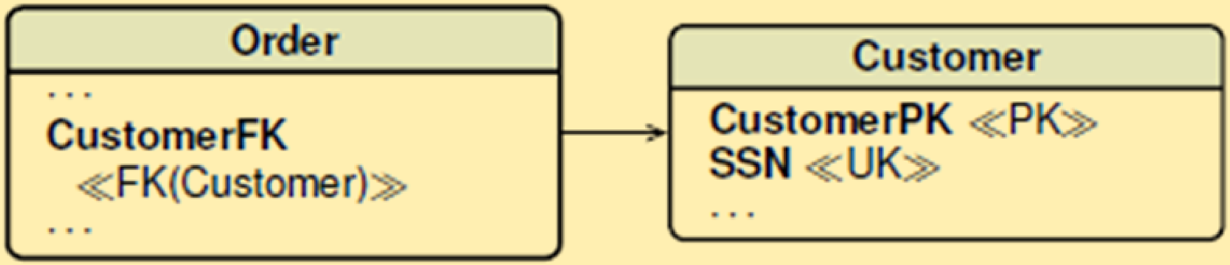
- TYPE 4

Add a new dimension (called **mini** or **profile**)

These aspects are not modelled in the conceptual schema

LOGICAL DESIGN: TYPE 2 SLOWLY CHANGING DIMENSIONS

Dimensions with both a surrogate and a natural key

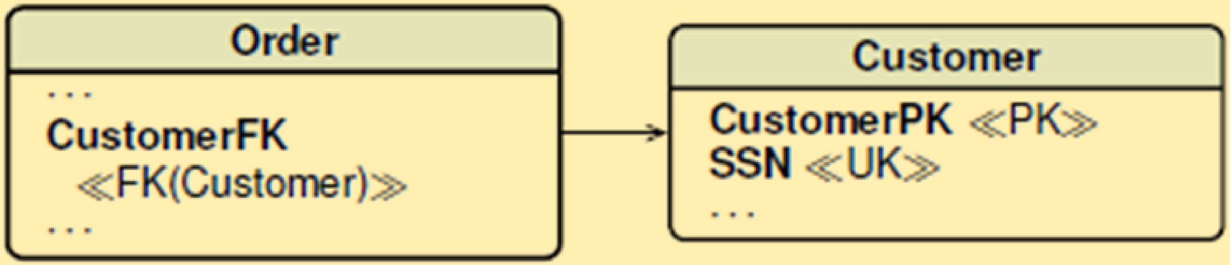


The customer **Jones** moved from zip code of 10019 to 45678 on 1/7/2018.

CustomerPK	SSN	Name	Zip	DateStart	DateEnd
1	31422	Murray	94025	1/1/1900	NULL
2	**12427**	**Jones**	**10019**	1/1/1900	NULL
3	22224	Smith	33120	1/1/1900	NULL

LOGICAL DESIGN: TYPE 2 SLOWLY CHANGING DIMENSIONS

Dimensions with both a surrogate and a natural key

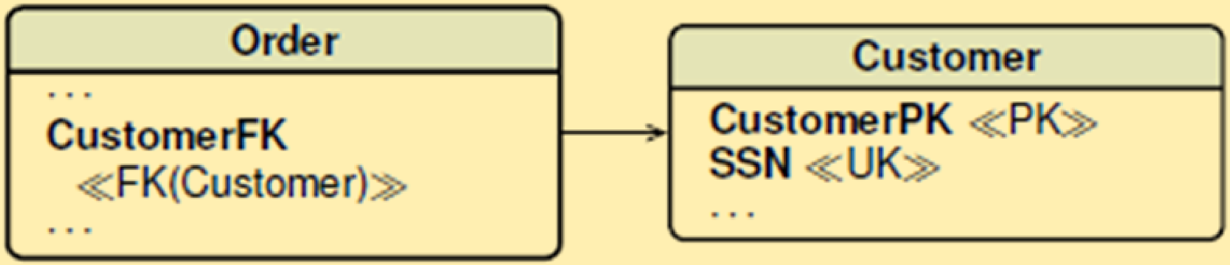


The customer **Jones** moved from zip code of 10019 to 45678 on 1/7/2018.

CustomerPK	SSN	Name	Zip	DateStart	DateEnd
1	31422	Murray	94025	1/1/1900	NULL
2	**12427**	**Jones**	**10019**	**1/1/1900**	**30/6/2018**
3	22224	Smith	33120	1/1/1900	NULL
4	**12427**	**Jones**	**45678**	**1/7/2018**	**NULL**

LOGICAL DESIGN: TYPE 2 SLOWLY CHANGING DIMENSIONS

Dimensions with both a surrogate and a natural key



The customer **Jones** moved from zip code of 10019 to 45678 on 1/7/2018.

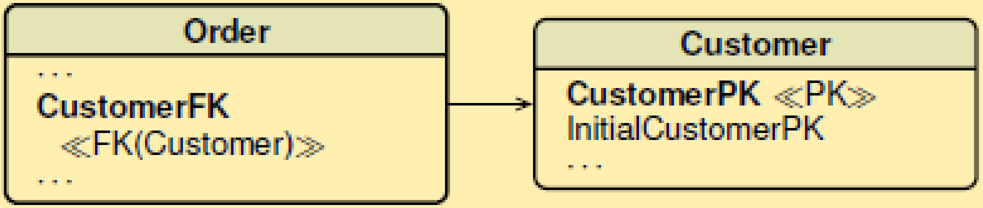
CustomerPK	SSN	Name	Zip	DateStart	DateEnd
1	31422	Murray	94025	1/1/1900	NULL
2	**12427**	**Jones**	**10019**	**1/1/1900**	**30/6/2018**
3	22224	Smith	33120	1/1/1900	NULL
4	**12427**	**Jones**	**45678**	**1/7/2018**	**NULL**

SQL: How many customers have made an Order greater than ... ?

COUNT(*) ? **Or** COUNT(DISTINCT SSN) ?

LOGICAL DESIGN: TYPE 2 SLOWLY CHANGING DIMENSIONS

- **Dimensions with a surrogate key only**



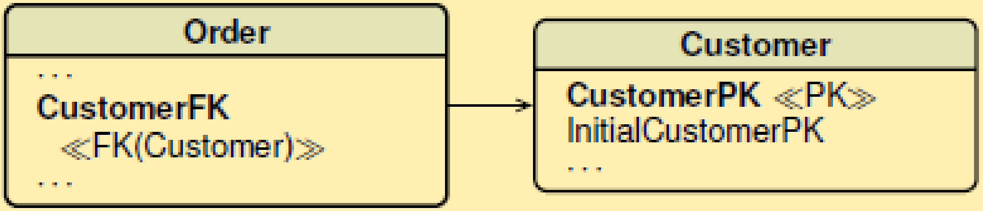
(b) *First surrogate key in the dimension table*

The customer **Jones** moved from zip code of 10019 to 45678 on 1/7/2018.

CustomerPK	InitialCustomerPK	Name	Zip	DateStart	DateEnd
1	1	Murray	94025	1/1/1900	NULL
2	**2**	**Jones**	**10019**	**1/1/1900**	**30/6/2018**
3	3	Smith	33120	1/1/1900	NULL
4				**1/7/2018**	**NULL**

LOGICAL DESIGN: TYPE 2 SLOWLY CHANGING DIMENSIONS

- **Dimensions with a surrogate key only**



Order

. . .

CustomerFK

≪FK(Customer)≫

. . .

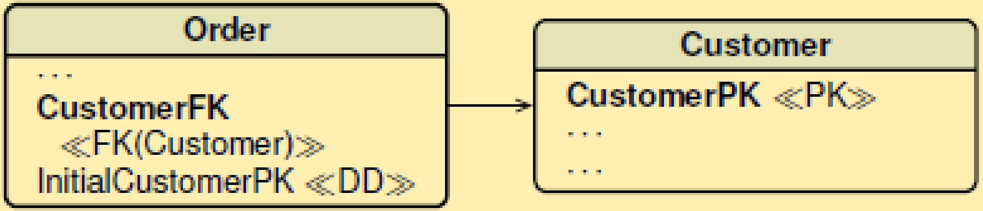
Customer

CustomerPK ≪PK≫

InitialCustomerPK

. . .

(b) *First surrogate key in the dimension table*



Order

. . .

CustomerFK

≪FK(Customer)≫

InitialCustomerPK ≪DD≫

Customer

CustomerPK ≪PK≫

. . .

. . .

(c) *First surrogate key in the fact table*

Advantages of (c) vs (b):

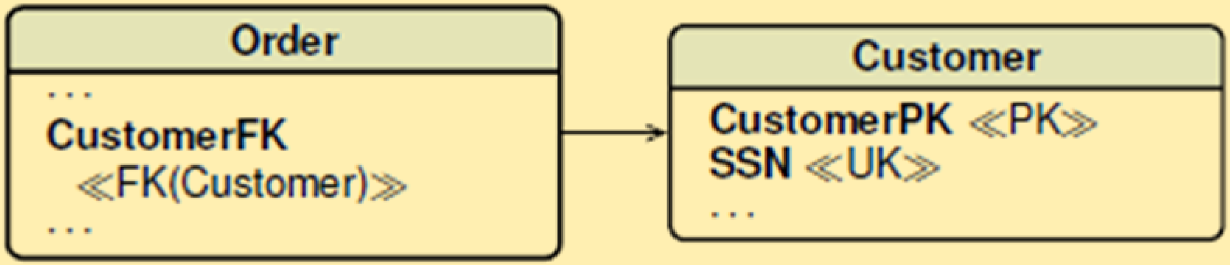
- In some queries, no need to join to calculate COUNT(DISTINCT InitialCustomerPK)
- Interpreting InitialCustomerPK as a measure would require it in the fact table

Advantages of (b) vs (c):

- Less space
- Mapping from CustomerPK to InitialCustomerPK already in the Customer table, while (c) requires a separate mapping table in the staging area to populate the fact table.

LOGICAL DESIGN: TYPE 3 SLOWLY CHANGING DIMENSIONS

Add new attributes to keep track of customer data change

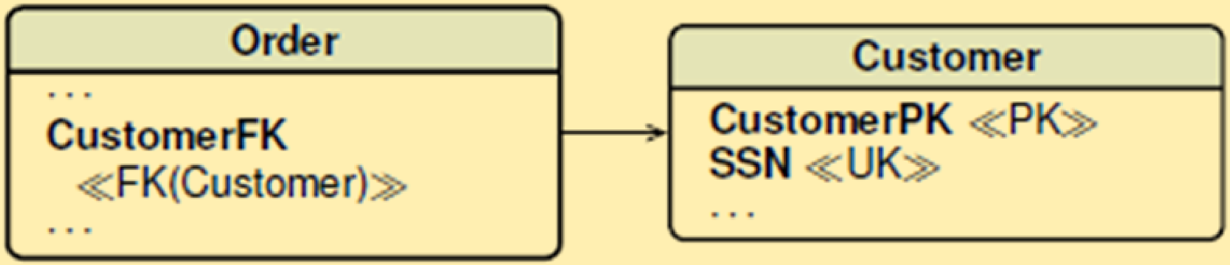


The customer **Jones** moved from zip code of 10019 to 45678 on 1/7/2018.

CustomerPK	SSN	Name	Zip	Old_Zip	DateStart	OldDateStart
1	31422	Murray	94025		1/1/1900	NULL
2	**12427**	**Jones**	**45678**	**10019**	**1/7/2018**	1/1/1900
3	22224	Smith	33120		1/1/1900	NULL

LOGICAL DESIGN: TYPE 3 SLOWLY CHANGING DIMENSIONS

Add new attributes to keep track of customer data change



Order

...

CustomerFK
≪FK(Customer)≫

...

Customer

CustomerPK ≪PK≫
SSN ≪UK≫

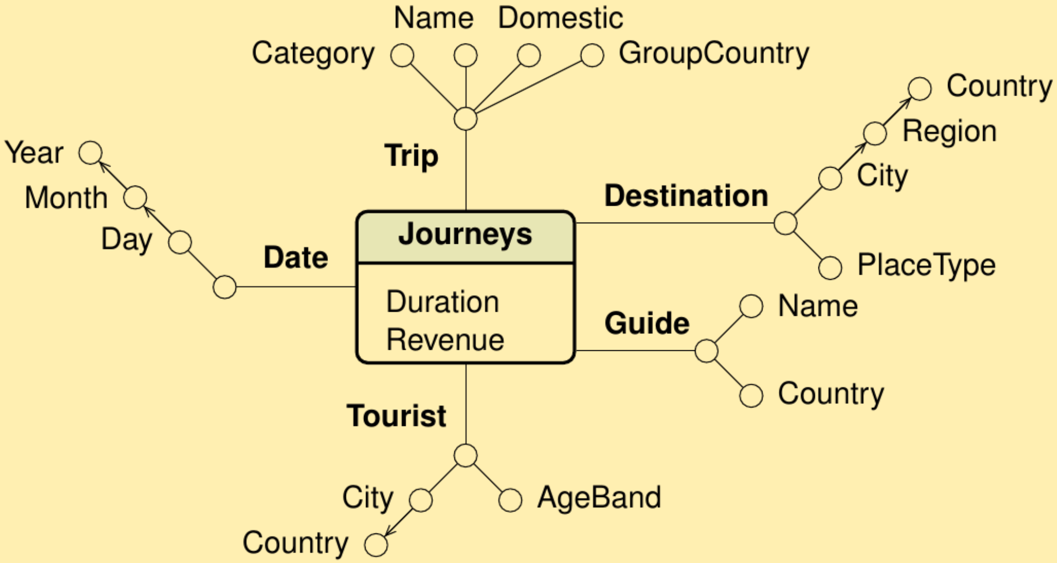
...

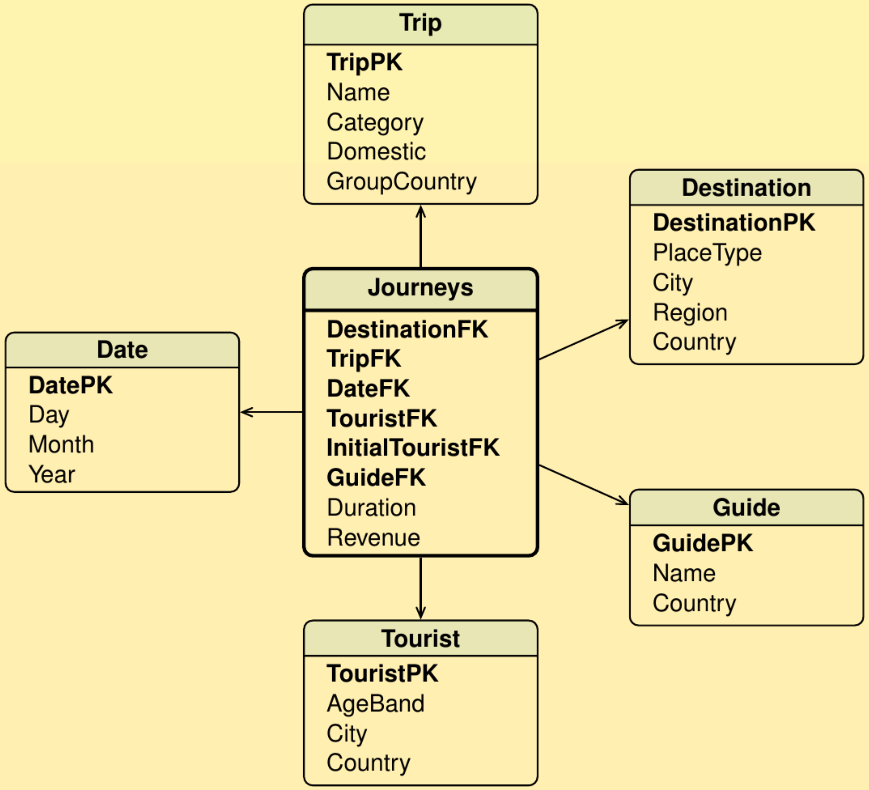
Total revenue by ZIP

Complicates SQL query format

```
WITH tmp AS (
   SELECT *, CASE WHEN DateOrder < DateStart THEN Zip
                          ELSE Old_Zip END As ActualZip
   FROM Order, Customer
   WHERE CustomerFK = CustomerPK
)
SELECT ActualZip, SUM(Revenue)
FROM tmp
GROUP BY ActualZip
```

TRAVEL AGENCY



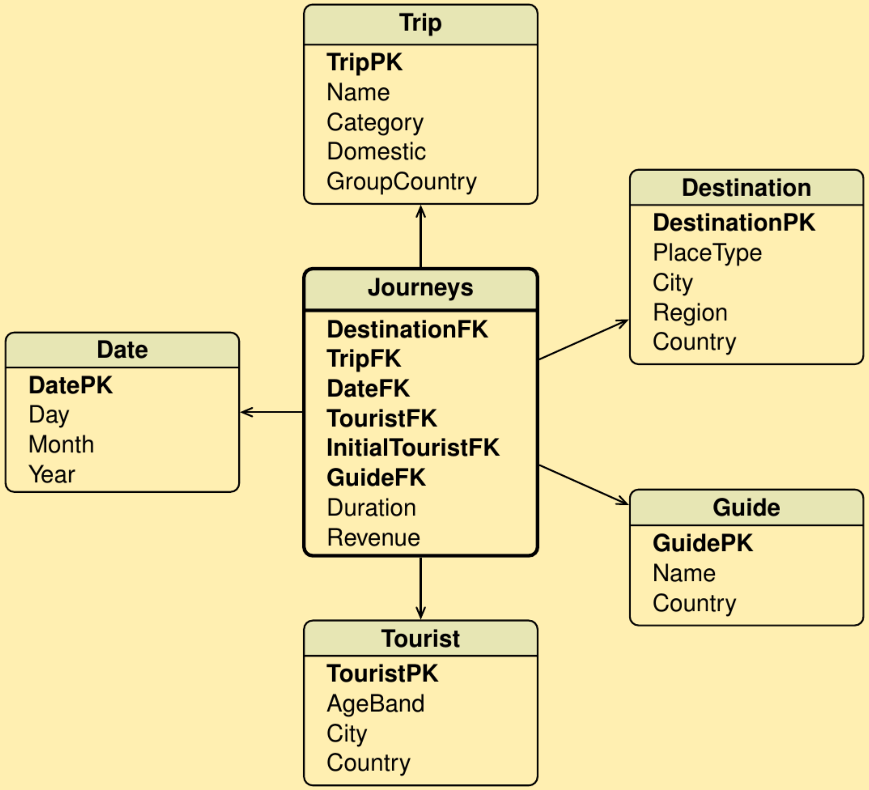


DFM SCHEMA

STAR SCHEMA

ADDITIONAL REQUIREMENTS

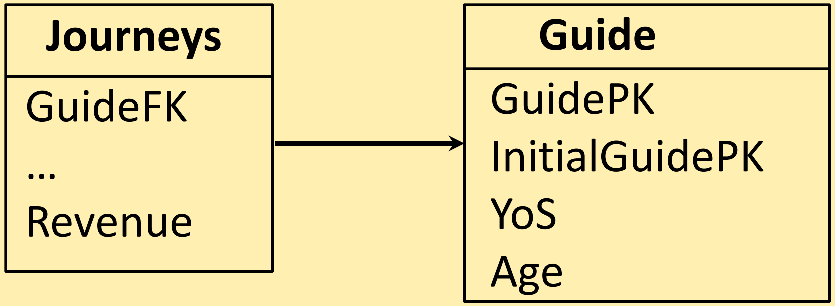
8. Total revenue **by** guide's years of service



SQL QUERIES ON (MODIFIED) STAR SCHEMA

LOGICAL DESIGN: TYPE 4 FAST CHANGING DIMENSIONS

SMALL DIMENSIONS: Type 2 technique is still recommended



Journeys

GuideFK
…
Revenue

Guide

GuidePK
InitialGuidePK
YoS
Age

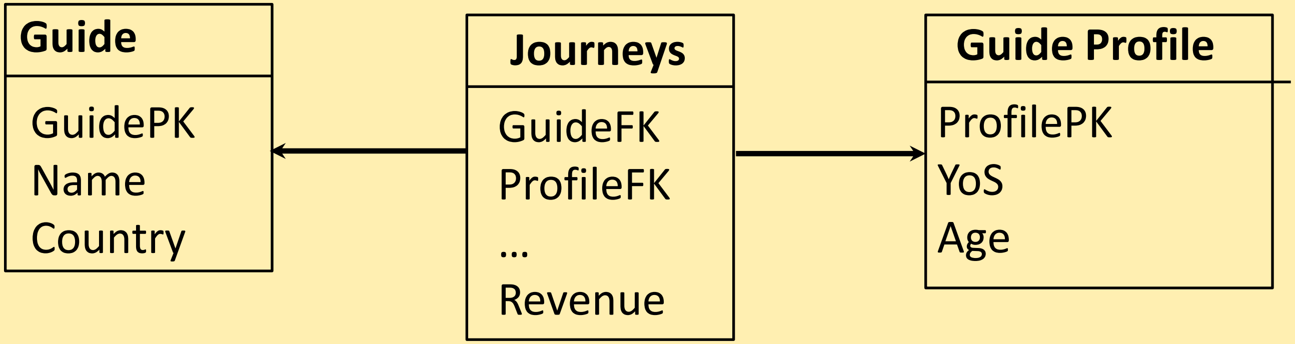
8. Total revenue **by** guide's years of service

```
SELECT YoS, SUM(Revenue) As TR
FROM Journeys, Guide
WHERE GuideFK = GuidePK
GROUP BY YoS
```

LOGICAL DESIGN: TYPE 4 FAST CHANGING DIMENSIONS

LARGE DIMENSIONS: Type 4

Create a separate junk/mini dimension table with frequently changing attributes



Numerical data may be converted into banded values, or into a hierarchy (eg., age -> age-decades -> young-middle-elder)

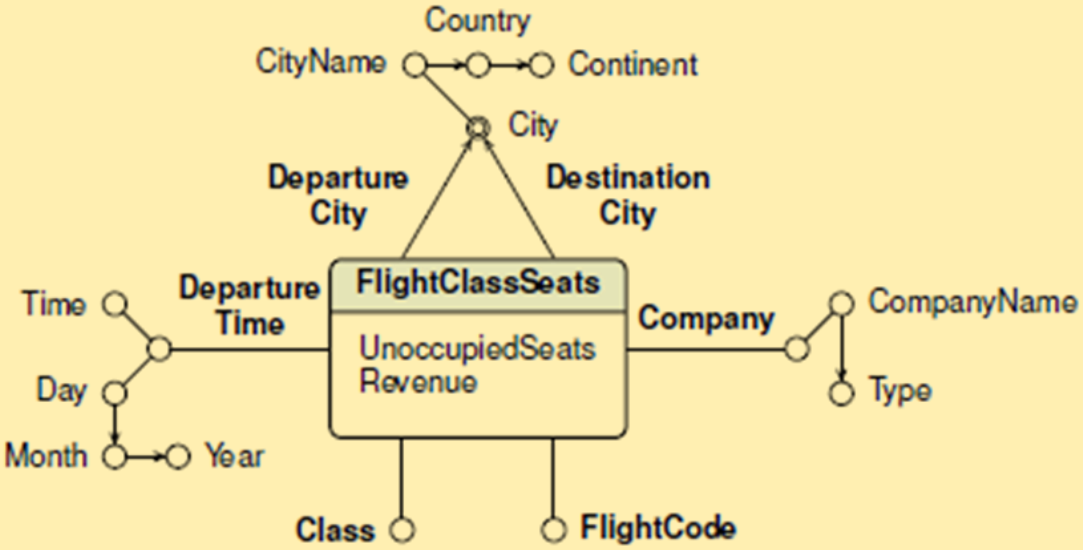
GuideProfile

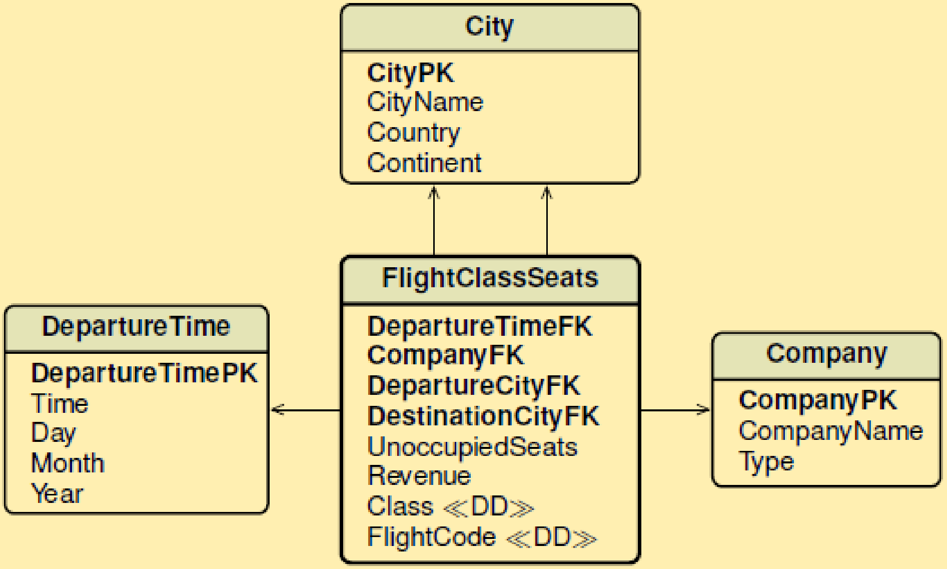
ProfilePK	YoS	Age
1	1	18-30
2	2	18-30
3	…	…
4	1	31-40
…	…	…

8. Total revenue **by** guide's years of service

```
SELECT YoS, SUM(Revenue) As TR
FROM Journeys, GuideProfile
WHERE ProfileFK = ProfilePK
GROUP BY YoS
```

LOGICAL DESIGN: SHARED DIMENSIONS





City
CityPK
CityName
Country
Continent

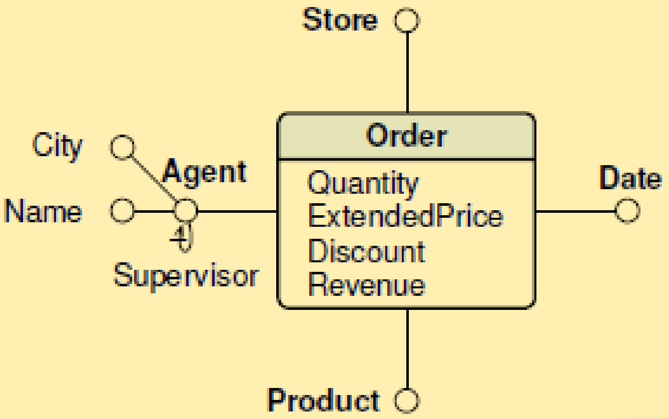
DepartureTime
DepartureTimePK
Time
Day
Month
Year

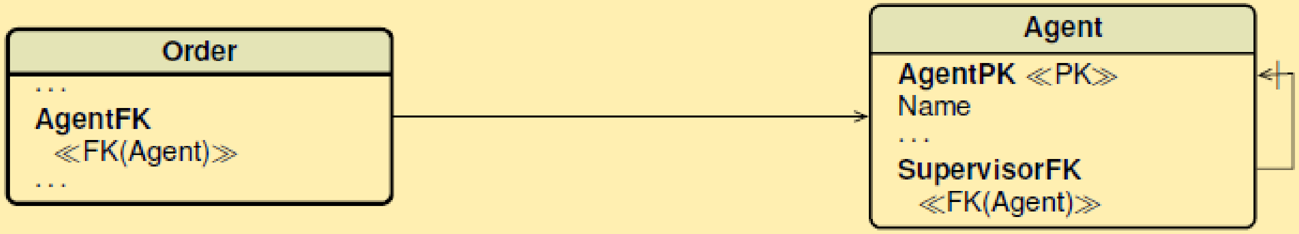
FlightClassSeats
DepartureTimeFK
CompanyFK
DepartureCityFK
DestinationCityFK
UnoccupiedSeats
Revenue
Class ≪DD≫
FlightCode ≪DD≫

Company
CompanyPK
CompanyName
Type

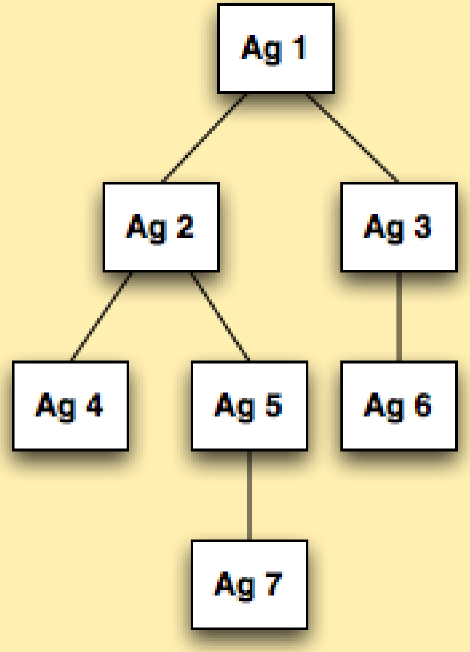
Different Hierarchies	Different tables
Shared Hierarchies	One table
Similar Hierarchies	Views of one table

LOGICAL DESIGN: RECURSIVE HIERARCHIES AND SQL





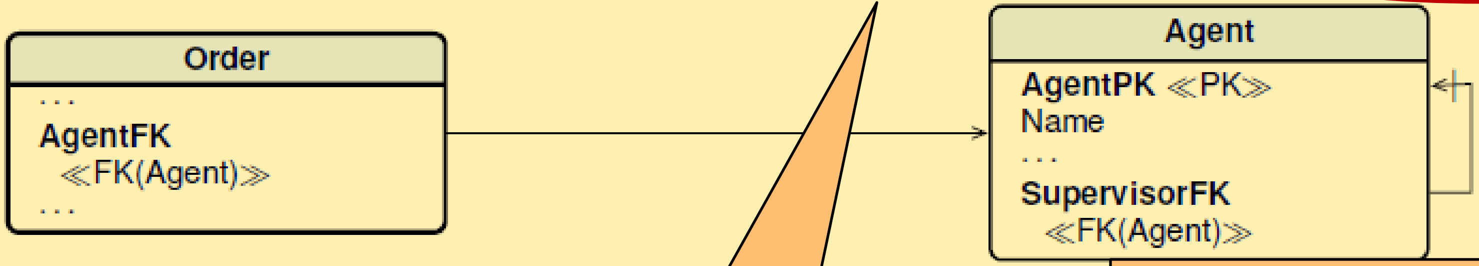
(a) *Without a bridge table*



AgentPK	Name	SupervisorFK
1	Ag1	NULL
2	Ag2	1
3	Ag3	1
4	Ag4	2
5	Ag5	2
6	Ag6	3
7	Ag7	5

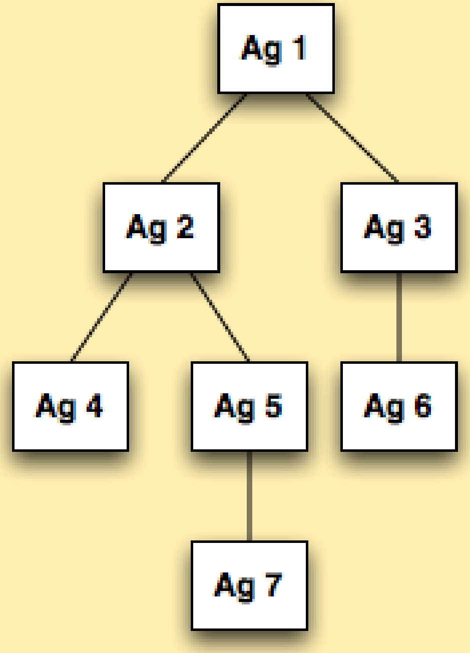
WRITE THE RELATION AGENT

Total revenue for **Agent 2** and for all his subordinates?



(a) *Without* *dge table*

- Requires additional joins
- The number not known apriori

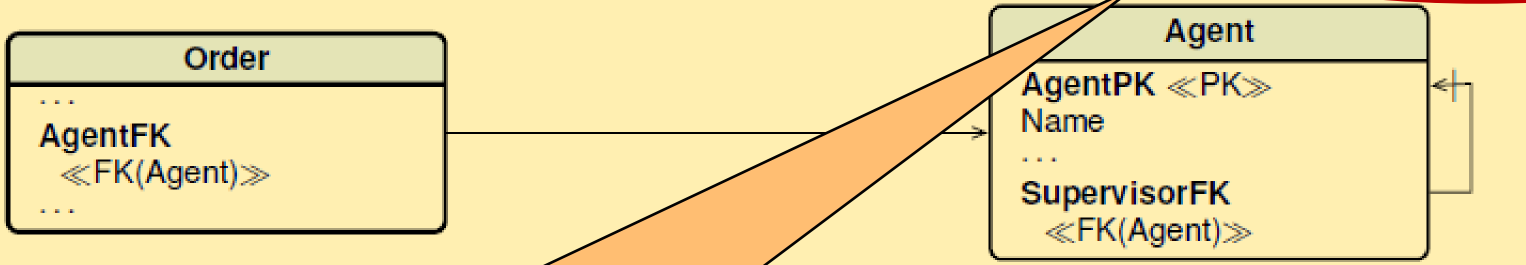


```
SELECT SUM(Revenue) As TR
FROM Order, Agent
WHERE AgentFK=AgentPK
AND Name='Ag2'
```

AgentPK	Name	SupervisorFK
1	Ag1	NULL
2	Ag2	1
3	Ag3	1
4	Ag4	2
5	Ag5	2
6	Ag6	3
7	Ag7	5

WRITE THE RELATION AGENT

Total revenue for **Agent 2** and for all his subordinates?

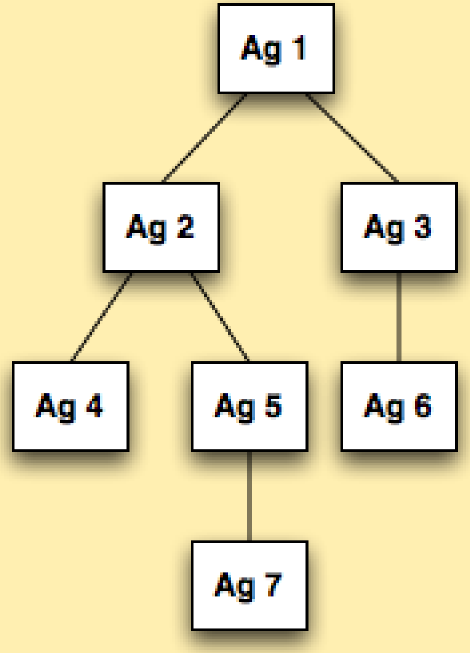


a bridge table

```
WITH RECURSIVE Ag2andSubordinates AS (
 SELECT  AgentPK
 FROM Agent
 WHERE Name='Ag2'
     UNION
 SELECTA.AgentPK
 FROM Agent A JOIN Ag2andSubordinates S ON
A.SupervisorFK=S.AgentPK )
SELECT 'Ag2' AS Name,SUM(Revenue)
FROM Ag2andSubordinates S JOIN Order O ON
S.AgentPK=O.AgentFK;
```

AgentPK	Name	SupervisorFK
1	Ag1	NULL
2	Ag2	1
3	Ag3	1
4	Ag4	2
5	Ag5	2
6	Ag6	3
7	Ag7	5

LOGICAL DESIGN: RECURSIVE HIERARCHIES

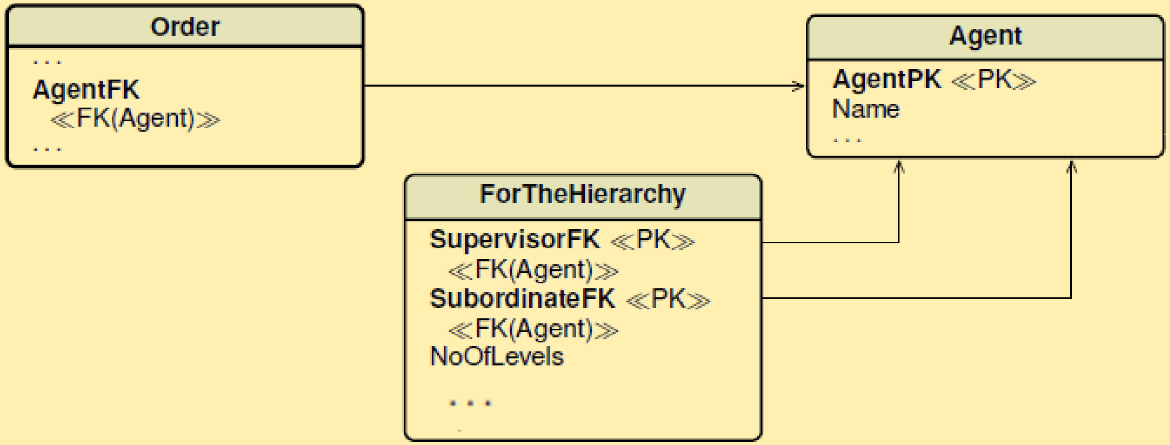


The table **ForTheHierarchy** is defined with **a record for each element of the hierarchy** plus **one for each pair (Supervisor, Subordinate)**

SupervisorFK	SubordinateFK	NoOfLevels
1	1	0
1	2	1
1	3	1
1	4	2
1	5	2
1	6	2
1	7	3
2	2	0
2	4	1
2	5	1
2	7	2
3	3	0
3	6	0
4	4	0
5	5	0
5	7	1
6	6	0
7	7	0

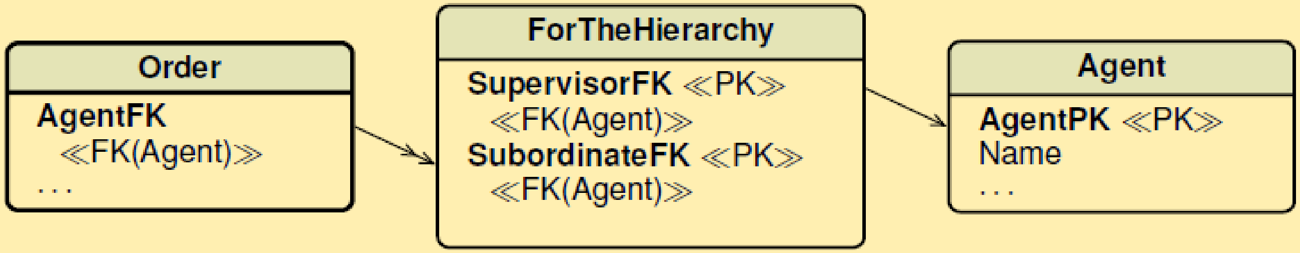
(SupervisorFK, SubordinateFK) is the Primary Key.

LOGICAL DESIGN: RECURSIVE HIERARCHIES



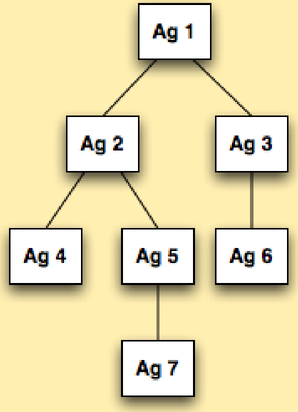
(b) *With a bridge table*

Total revenue for ***Agent 2*** and for all her subordinates



(a) *Descending the hierarchy*

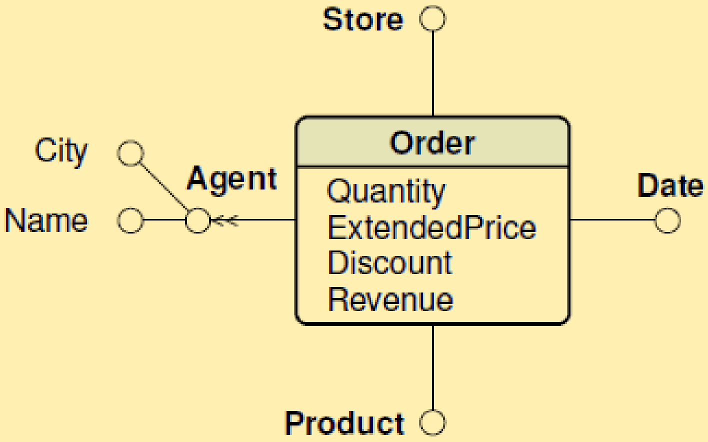
⇓

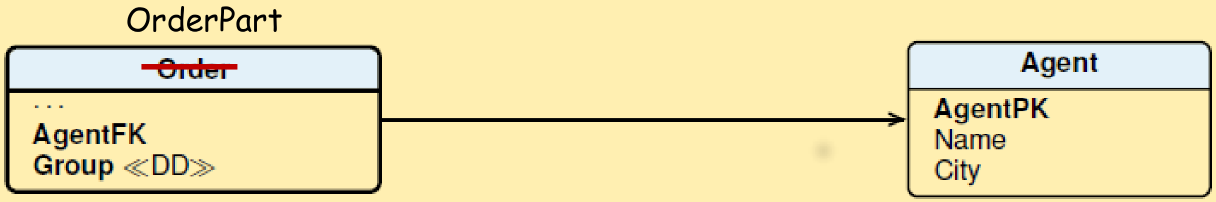


```
SELECT     A.Name, SUM(Revenue)
FROM       Order O, ForTheHierarchy H, Agent A
WHERE      O.AgentFK = H.SubordinateFK AND H.SupervisorFK = A.AgentPK
           AND A.Name = 'Ag2' AND NoOFLevels <= 2
GROUP BY   A.Name;
```

In case a maximum distance is required

LOGICAL DESIGN: MULTIVALUED DIMENSIONS





(d) *New fact granularity*

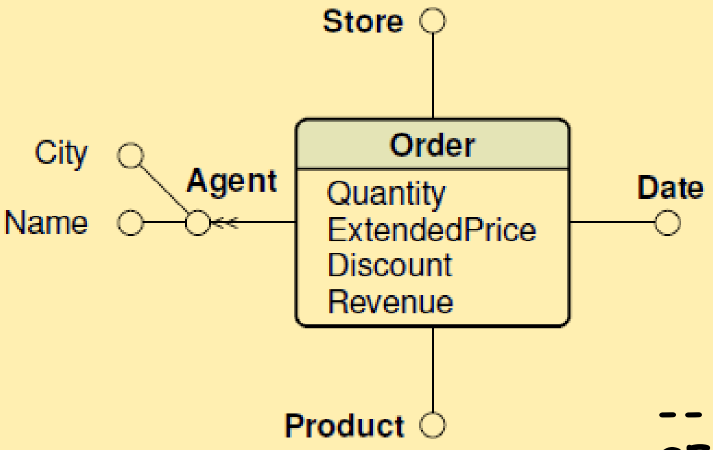
OrderPart

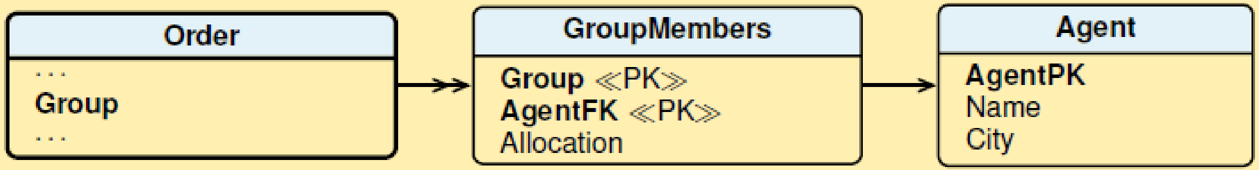
AgentFK	Group	Revenue
...	...	...
3	57	70 €
8	57	30 €
...	...	...

Single order of 100 € {

LOGICAL DESIGN: MULTIVALUED DIMENSIONS

Metrics: SUM(Revenue*Allocation)





(b) *Another type of an auxiliary table*

```
-- Total Revenue by Agent 3
SELECT SUM(Revenue*Allocation)
FROM Order AS O, GroupMembers AS G
WHERE O.Group = G.Group AND G.AgentFK=3
```

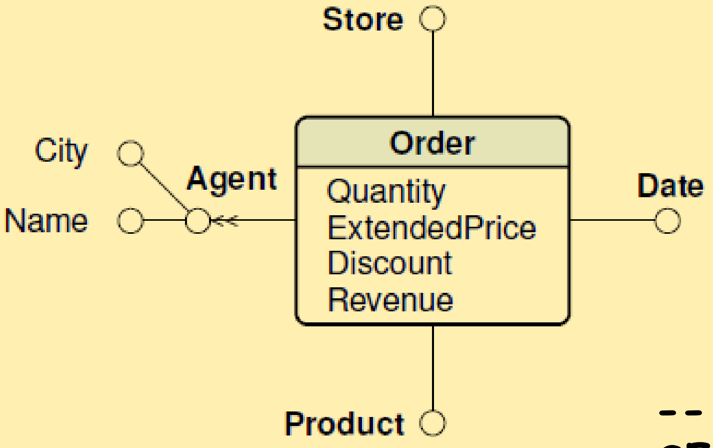
Order

Group	Revenue
…	…
57	100 €
…	…

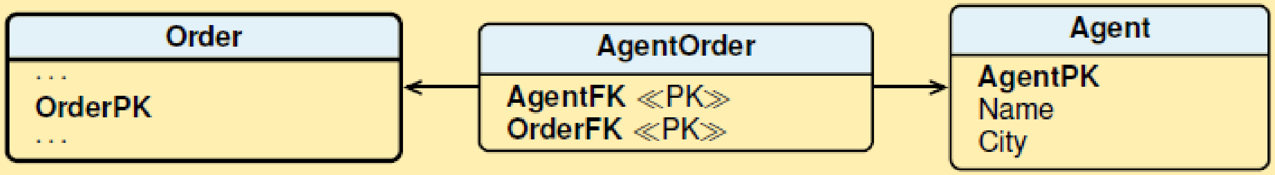
GroupMembers

Group	AgentFK	Allocation
…	…	…
57	3	70%
57	8	30%
…	…	…

LOGICAL DESIGN: MULTIVALUED DIMENSIONS



Metrics: SUM(Revenue*Allocation)



(a) *A traditional representation of a many-to-many relationship*

```
-- Total Revenue by Agent 3
SELECT SUM(Revenue*Allocation)
FROM Order AS O, AgentOrder AS A
WHERE O.OrderPK = A.OrderFK AND A.AgentFK=3
```

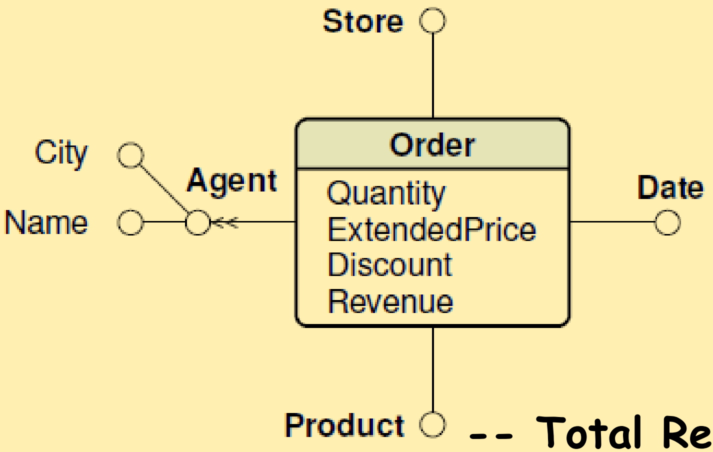
Order

OrderPK	Revenue
...	...
1234	100 €
...	...

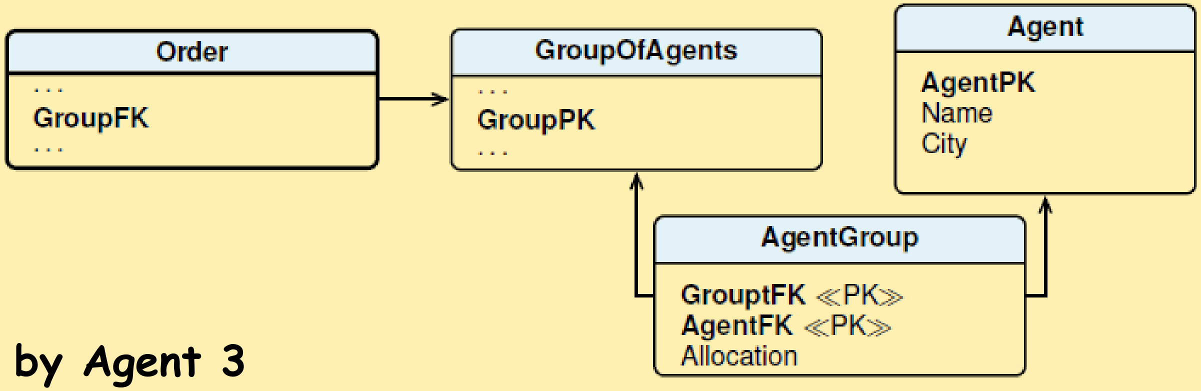
AgentOrder

OrderFK	AgentFK	Allocation
...	...	...
1234	3	70%
1234	8	30%
...	...	...

LOGICAL DESIGN: MULTIVALUED DIMENSIONS



Metrics: SUM(Revenue*Allocation)



(c) *A bridge table*

```
-- Total Revenue by Agent 3
SELECT SUM(Revenue*Allocation)
FROM Order AS O, GroupOfAgents AS G, AgentGroup A
WHERE O.GroupFK = G.GroupPK AND A.GroupFK=G.GroupPK AND A.AgentFK=3
```

Order

OrderFK	Revenue
…	…
57	100 €
…	…

GroupOfAgents

GroupPK
…
57
…

AgentGroup

GroupFK	AgentFK	Allocation
…	…	…
57	3	70%
57	8	30%
…	…	…

SUMMARY

Building a DW (conceptual and logical design, and data loading) is a **complex task** that requires **business skills**, **technology skills**, and **program management skills**.

The logical design of a conceptual schema is not trivial, especially for storage optimization, and for treating **dimensions that change over time** and **multivalued dimensions / dimensional attributes**.

Finally, several controls are needed for the review of a project to improve the quality of the conceptual and logical design, see the lecture notes.

Next, another complex task is **using** a DW to translate the business requirements into SQL queries that can be satisfied by the DW.