Data Warehouse/Data Mart Conceptual Modeling and Design

From E/R or Relational schema to Fact schema in DFM method

Aix-Marseille université

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille

January, 2021

From Entity-Relationship schema to DFM
From Relational schema to DFM

Plan

1. Reminder of the DFM Data Warehouse design method
- Two design levels: conceptual and logical

2. From Entity-Relationship (E-R) schema to Fact schema
- Finding and defining facts from E-R schema
- Building and reduce the Attribute Tree from Relational schema
- Building the Fact Schema from Attribute Tree defining dimention and measure fact attributes

3. From Relationnal schema to Fact schema
- Finding and defining facts from E-R schema
- Building and reduce the Attribute Tree from Relational schema
- Building the Fact Schema from Attribute Tree defining dimention and measure fact attributes

Bibliographie

- **Books**
  - Golfarelli M., Rizzi S., “Data Warehouse Design : Modern Principles and Methodologies”, McGrawHill, 2009.
  - Kimball R., Ross, M., “Entrepôts de données : guide pratique de modélisation dimensionnelle”, 2°édition, Ed. Vuibert, 2003.
  - S. Rizzi. “Conceptual modeling solutions for the data warehouse”. In Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications, J. Wang (Ed.), Information Science Reference, pp. 208-227, 2008.
  - M. Golfarelli, D. Maio, S. Rizzi. “Conceptual Design of Data Warehouses from E/R Schemes”. Proceedings 31st Hawaii International Conference on System Sciences (HICSS-31), vol. VII, Kona, Hawaii, pp. 334-343, 1998.
- **Courses**
  - Course of M. Golfarelli M. and S. Rizzi, University of Bologna
  - Courses of M. Böhlen and J. Gamper J., Free University of Bolzano

1. Reminder of the DFM method

- **DFM method**
- **Shemata derivations**
- **Conceptual Design: main steps**

The Dimensionnal Fact Model (DFM)

The **Dimensional Fact Model (DFM)** has be proposed by Golfarelli M., Rizzi S. to support a Conceptual Design of DW

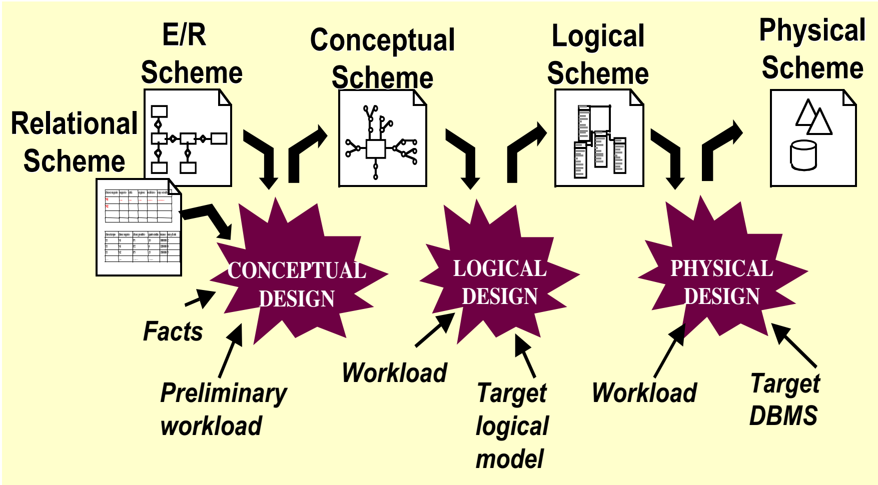
The DFM is a **graphical conceptual model** for **Data Mart design**

The **aim** of the **DFM** is to :

1. Provide an efficient **support to Conceptual Design**
2. Create an environment in which **user queries may be formulated intuitively**
3. Make **communication possible between designers and end users** with the goal of formalizing requirement specifications
4. Build a **stable platform for logical design** (independently of the target logical model)
5. Provide **clear and expressive design documentation**

The conceptual representation generated by the DFM consists of **a set of fact schemata** that basically model *facts, measures, dimensions, and hierarchies.*

Schemata derivations for DMs design



Conceptual Design: main steps

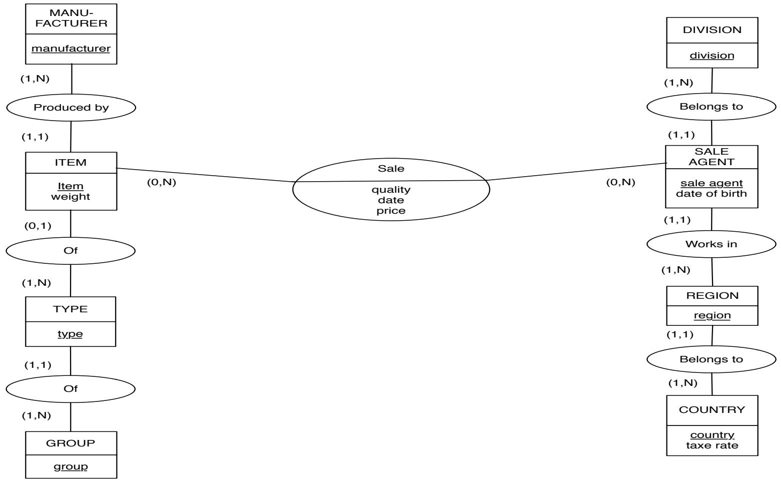
- **Conceptual Design** is based on the **documentation** of the underlying operational information system (IS) which is based on :
  - ***Entity-Relation (E-R) schemata*** *OR*
  - ***Relational schemata***
- **Main steps to derive Fact schema from documentation are:**
  1. Find facts
  2. For each fact:
     a) Navigate functional dependencies
     b) Drop useless attributes
     c) Define dimensions and measures
- **Derivations from E/R schema or from Relational schema** are **very similar** (main difference concerns the algorithm used to build the attribute tree)

2. From Entity-Relationship schema to Dimensionnal Fact schema

- **Finding and defining facts from E-R schema**
- **Building the Attribute Tree from E-R schema**
- **Building the Fact Schema from Attribute Tree**

Entity-Relationship (E-R) schemata: variants (1)

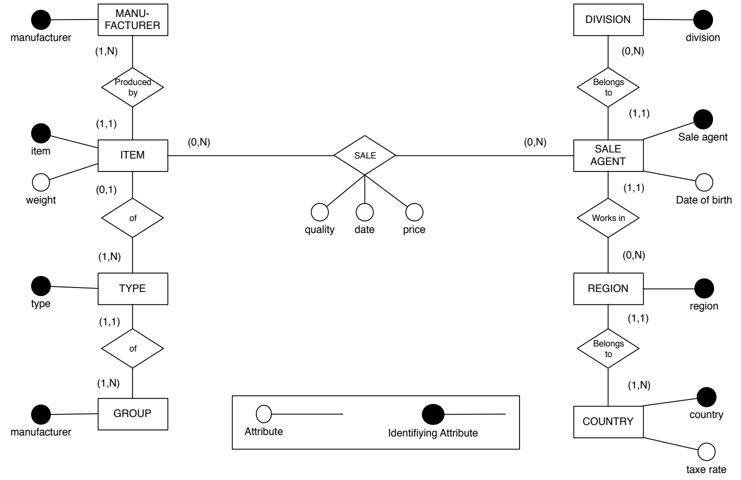
1.E-R schema in French formalism (MERISE) :



Note : identifying attributes are underlined, specific notation and place of cardinalities

Entity-Relationship (E-R) schemata: variants (1)

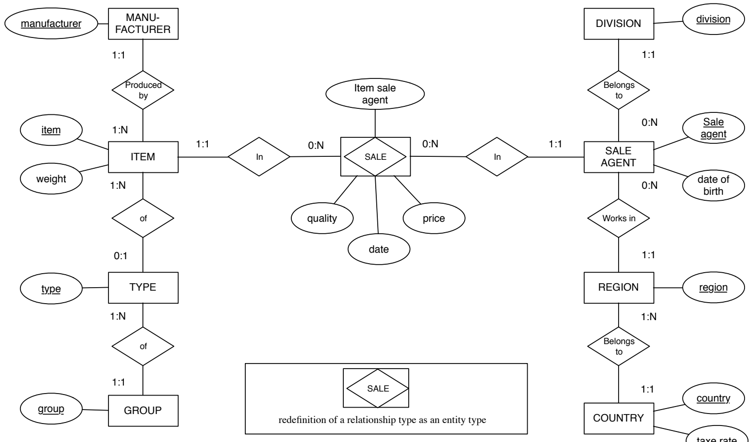
2.E-R schema in Italian formalism (Battini, Golfarelli and Rizzi, ...) :



Note : identifying attributes are distinguished, specific notation and place of cardinalities

Entity-Relationship (E-R) schemata: variants (1)

3.E-R schema in German Formalism



Note : identifying attributes are underlined, cardinalities places are inverted from other formalisms, redifinition of relationship type as entity type

From E-R schema to Dimensionnal Fact schema (DF)

The step to derive DF schema from E-R schema is :

- **1. Finding and defining facts from E-R schema**

For each fact :

- **2. Building the Attribute Tree from E-R schema**
- **3. Building the Fact Schema from Attribute Tree**

Note that the step to derive DF schemata from E/R schema is very similar: the main difference concerns the algorithm used to build the attribute tree

1. Identifying FACT from E-R schema (1)

Facts are concepts of primary interest for the decision-making process. Typically, ***they correspond to events occurring dynamically***

On a E/R scheme, a fact may be represented either by :

- an **entity F**
- or by **an n-ary relationship R** between entities E1,... En.

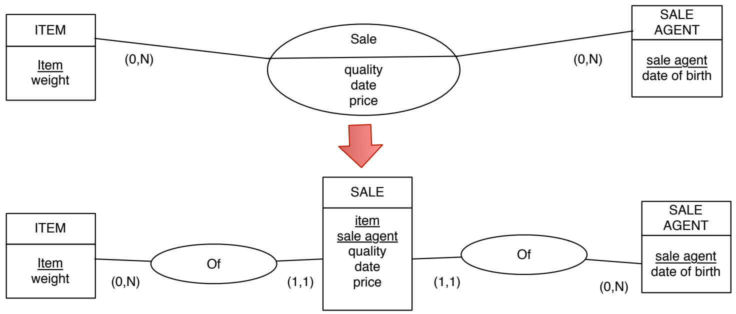
• In the latter case, for the sake of simplicity, it is worth transforming R into an entity **F** by replacing each branch **Ei** with a binary relationship **Ri** between **F** and **Ei**;

• The **attributes of the relationship** become **attributes of F**; the identifier of F is the *combination of the identifiers* of Ei, i=1,...n.

1. Identifying FACT from E-R schema (1)

In our example:

- The fact is the **n-ary relationship "Sale"** between the entities "**ITEM**" and "**SALE AGENT**"
- The n-ary relationship « **Sale** » is transformed into an **entity « SALE »** as follow:



2. Building the Attribute Tree (1)

1. **Root of the Attribute Tree**

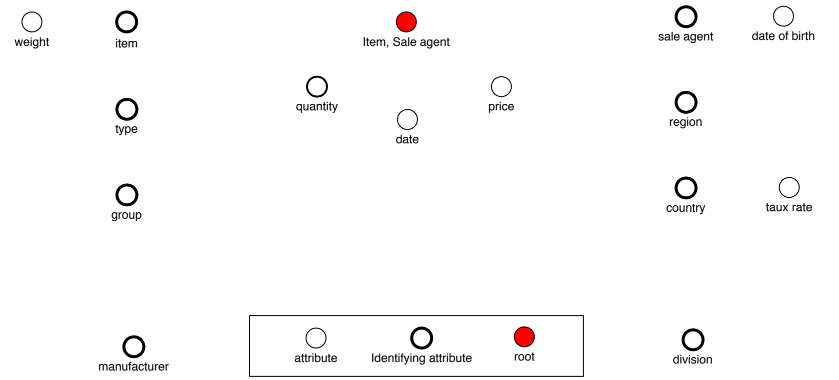
After identifying entity F as a fact the tree is built in the following way:

• Each **attribute** and **identifying attribute** within the E/R schema **becomes a node** (a node may represent a fact, a fact measure, a dimension, a dimension attribute or a non-dimension attribute of the resulting fact schema)

• The **identifying attributes of the fact entity F** becomes the **ROOT** (red/grey)

2 - Building the Attribute Tree : example (1)

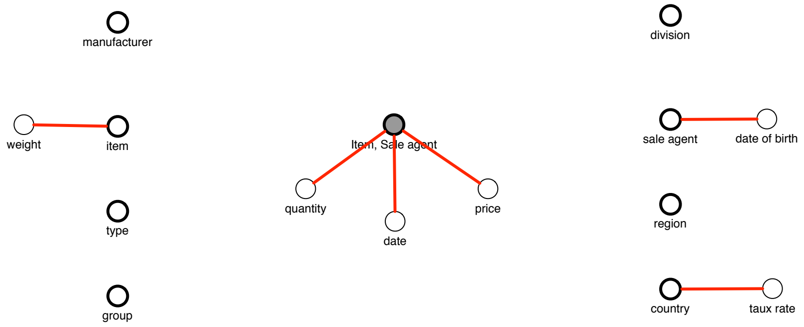
2. **Placing all attributes according their topological position in the E-R schema:**



2 - Building the Attribute Tree : example (2)

3. Installing Entity dependencies

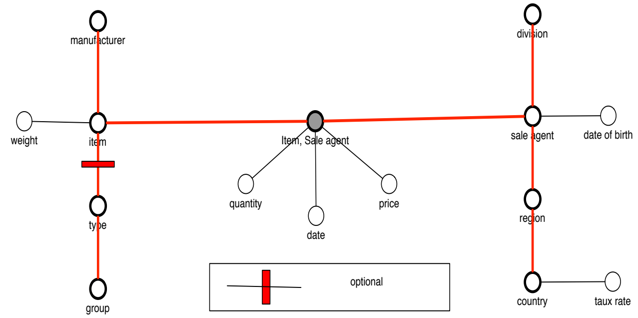
If a node corresponds to a *identifying attribute* of an entity in the E/R schema, all other nodes representing attributes of this entity are attached to it.



2 - Building the Attribute Tree : example (3)

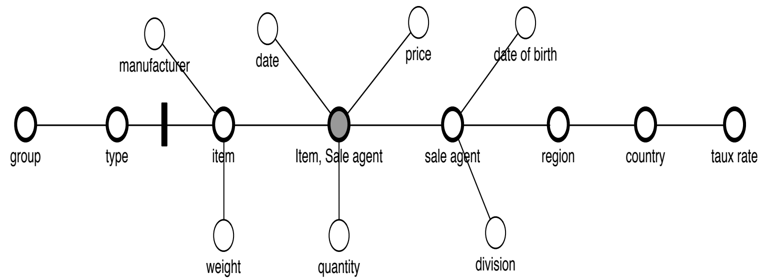
4. Relationship links and optional relationships

- Nodes representing identifying attributes are CONNECTED to each other according to their corresponding entity's relationships in the E/R schema.
- *Optional relationships,* i.e. relationships with a cardinality of (0,1) on the "1-side", are tagged by a dash (▬).



3 - Rearranging

In our example, after rearranging the nodes our attribute tree looks like this:



4 - Pruning and grafting the Attribute Tree (1)

- Not all of the tree's attributes (nodes) may be of interest for the data warehouse, so the next step is eliminating unwanted or unnecessary information by pruning and grafting the attribute tree.
- « **Pruning** » means to drop nodes or entire sub-trees. All attributes belonging to dropped sub-trees will not be included in the resulting fact schema and can therefore not be used as aggregation levels.

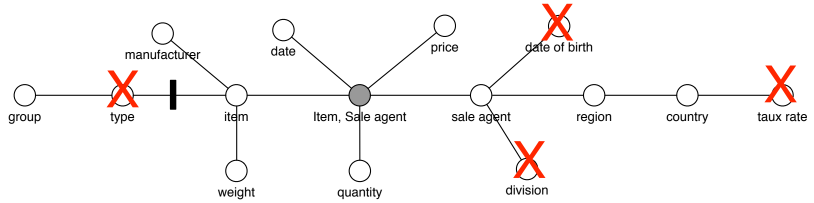
In our example:

- we could prune the entire sub-tree including "**region**", "**country**" and "**tax rate**" leaving us with only "division" as a possible higher aggregation level for "sales agent" ("date of birth" represents a typical candidate for a non-dimension attribute).
- however, we decide to prune only "**division**", "**date of birth**" and "**tax rate**".

4 - Pruning and grafting the Attribute Tree (2)

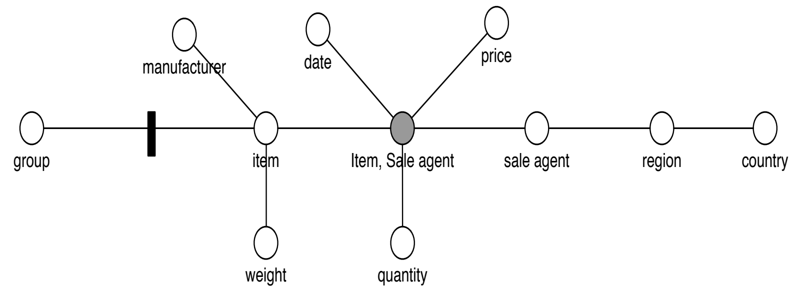
- « **Grafting** » is used if a node or sub-tree contains unnecessary information but its descendants have to be preserved.
- In our example :
  - we decide we only need a classification of the "**items**" by "**group**" and therefore **graft "type"**. All descendants of a grafted attribute ***inherit*** its optionality.

In our example:



4 - Pruning and grafting the Attribute Tree (3)

We obtain the reduced folowing Attribute tree:



Note: It should be noted that grafting a child of the root ***decreases the granularity*** of data but ***increases the number of dimensions*** in the resulting fact schema if the grafted node ***has more than one descendant.***

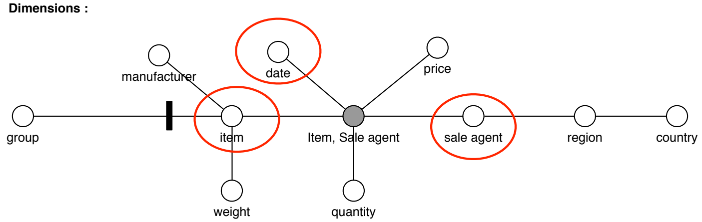
5 - Defining Dimensions (1)

- **Defining dimensions** is a key step, as they determine how the fact instances may be **aggregated**.
- They must be selected from the **descendants of the root** and may either be **discrete** attributes or **ranges of discrete** or **continuous** attributes.

In our example:

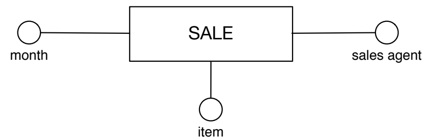
The attributes "**item**" and "**sales agent**" are good candidates for dimensions: both are ***discrete attributes*** and the usage of a range is not necessary.

Dimensions :



5 - Defining Dimensions (2)

After choosing the dimensions we are able to partially outline our **fact schema**:

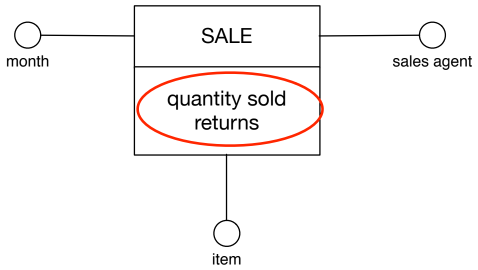


Notes on *Time* dimension:

- **Time** is usually a key dimension in the data warehouse, so we choose "**months**" (ranges of the "date" attribute) as a third dimension.
- If the time-attribute appeared as a descendant of a node that is not the root, it would be necessary to consider **grafting** the attribute tree even more to make time become a dimension.
- It is **worth** mentioning that there may be E/R schemas without any time-attributes at all. Those are snapshot schemas which just describe a current state. In this case old data is replaced continuously by new data in the OLTP. In the data warehouse however some kind of time-attribute may be added to store historical data and allow analysis over time.

7 - Defining Fact Measures (1)

- Fact attributes or **measures** are usually the **SUM** (or **AVERAGE**, **MINIMUM** or **MAXIMUM**) of expressions involving **numerical attributes** of the attribute tree that have NOT BEEN CHOSEN as **dimensions**.
- They may also be **COUNTS of the number of fact instances**.
- Our Fact schema with dimensions and measures:



7 - Defining Fact Measures (1)

Note 1:

- The existence of fact measures is NOT MANDATORY. As stated in the description of the model a fact schema without any key figures may be meaningful, providing the ***occurrences of the fact are counted***.
- If fact measures are defined, it is helpful to create a ***glossary*** that describes ***how to calculate each*** key figure from the attributes of the E/R schema.

Note 2:

- In the previous example the SUM operator in the glossary means that all fact instances are summed up for the same "month", "item" and "sales agent".
- If the E/R schema had "month" as its time attribute instead of "date", it would have been possible to translate the entity's attributes directly into the key figures of the fact schema without using any operators.

8 - Defining Hierarchies (1)

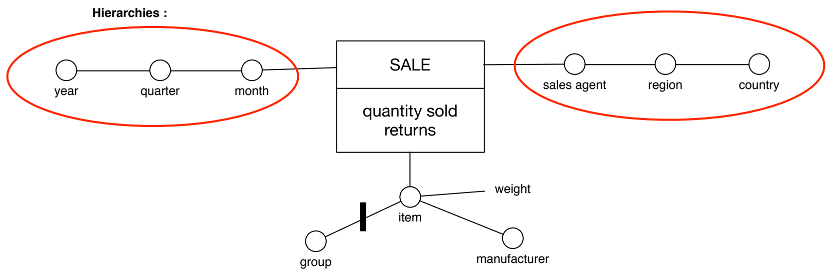
- In the last step we define the ***hierarchies*** for the ***dimensions*** previously identified.
- Using the attribute tree we already have a meaningful basis for the organization of the hierarchies, but it is still possible to further ***prune*** and ***graft*** the tree or to ***add additional aggregation levels***, which is usually done for the **time dimension**.

  In our example : we add "***quarters***" and "***years***" as ranges of "***months***".
- Attributes that should NOT BE USED for aggregation but nevertheless provide important information may be tagged as ***non-dimensional***.

  In our example : this is the case for attribute "***weight***" which is a non-dimensional attribute
- As we can see now the ***"item"-dimension*** features a parallel hierarchy:
  - the single "***items***" may be aggregated according to their "***manufacturer***"
  - or (optionary) to the "***group***"

8 - Defining Hierarchies (2)

Final Fact schema is:



3. From Relational schema to Dimensionnal Fact Schema

- **Finding and defining facts from Relational schema**
- **Building the Attribute Tree from Relational schema**
- **Building the Fact Schema from Attribute Tree**

From Relational schemata to Dimensionnal Fact schema

The step to derive DF schema from Relational schema is :

- **1. Finding and defining facts from Relational schema**

For each fact :

- **2. Building the Attribute Tree from Relational schema**
- **3. Building the Fact Schema from Attribute Tree**

Note that the step to derive DF schemata from E/R schema is very similar: the main difference concerns the algorithm used to build the attribute tree

1. Finding and defining facts

- Facts correspond to events occurring dynamically
- Within an ***Relational schema***, a fact is represented by a **table:**
  - Tables representing *frequently updated archives* are ***good candidates*** *to define facts*
  - Tables representing *nearly-static archives* or *representing structural properties of the domain* (such as STORE and CITY), *are **not candidates** to define facts*
- Each fact identified on the *Relational schema* becomes the **root** of **an attribute tree**, that become a **fact schema**.

Ex : the more important fact is a product sale, and it is represented by the SALES table

2. Building Attribute Trees

For each **fact** defined from F table, the **attribute tree** is built as follow :

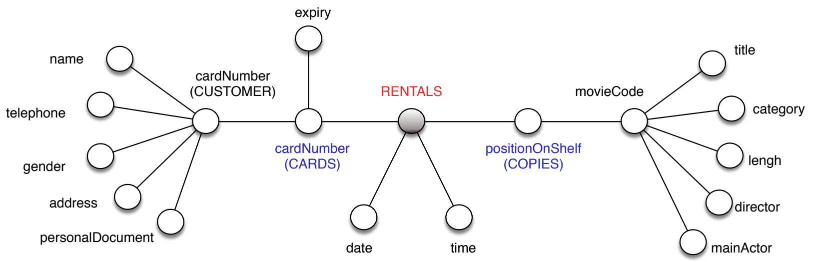
1. Each **node** of the attribute tree corresponds to one or more ***Relational schema attributes***
2. The **root** of the attribute tree corresponds to the ***primary key of F***
3. For each **node** ***v***, the corresponding attribute **functionally determines all the attributes** that correspond to the descendants of ***v*** *(**functionnal dependencies**)*

Building Attribute Trees: DVD example

Relational schema of the DVD rental BD:

- CARDS (cardNumber, expiry)
- CUSTOMERS (cardNumber:CARDS, name, gender, address, telephone, personalDocument)
- MOVIES (moviesCode, title, category, director, lengh, mainActor)
- COPIES (positionOnShelf, movieCode:MOVIES)
- RENTALS (positionOnShelf:COPIES, cardNumber:CARDS, date, time)

The table RENTALS is the only candidate for expressing facts, the **attribute tree** associated is:



Building Attribute Trees : Flight example (1)

Relational schema of the Flight BD:

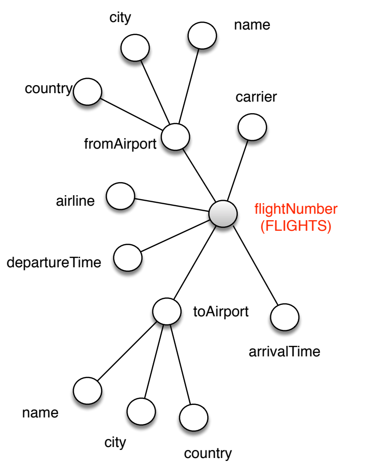
- FLIGHTS (flightNumber, airline, fromAirport:AIRPORTS, toAirport:AIRPORTS, departureTime, arrivalTime, carrier)
- FLIGHT_INSTANCES (FlightNumber:FLIGHTS, date)
- AIRPORTS (IATAcode, name, city, country)
- TICKETS (ticketNumber, flightNumber:FLIGHT_INSTANCES), seat, fare, passengersFirstName, passengersSurname, passengersGender)
- CHECK-IN (ticketNumber:TICKETS, CheckInTime, numberOfBags)

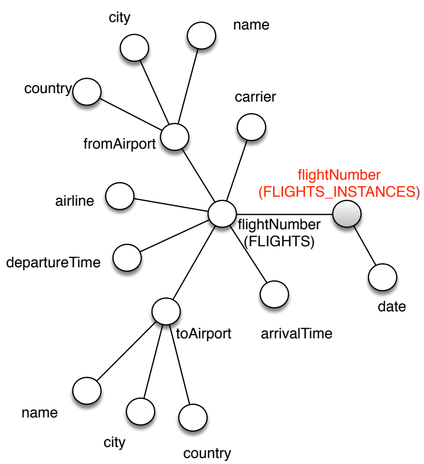
The tables that are candidates for expressing facts are :

- FLIGHTS
- FLIGHT_INSTANCES
- TICKETS
- CHECK_IN

Building Attribute Trees: Flight example (2)

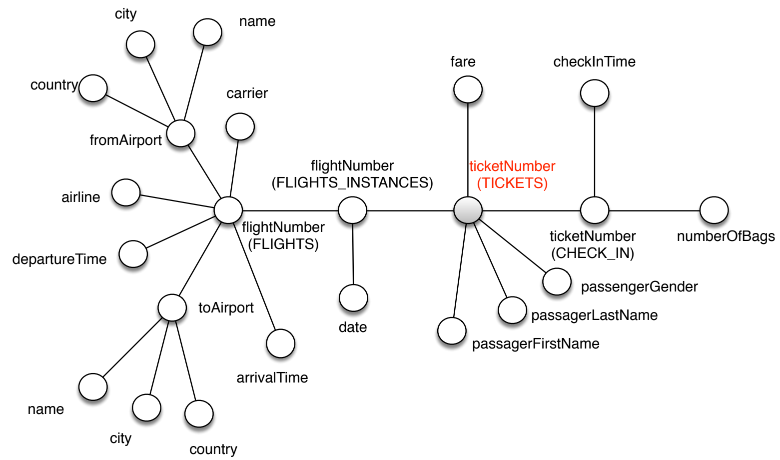
Attribute Tree 1 (FLIGHTS)	Attribute Tree 2 (FLIGHTS_INSTANCES)





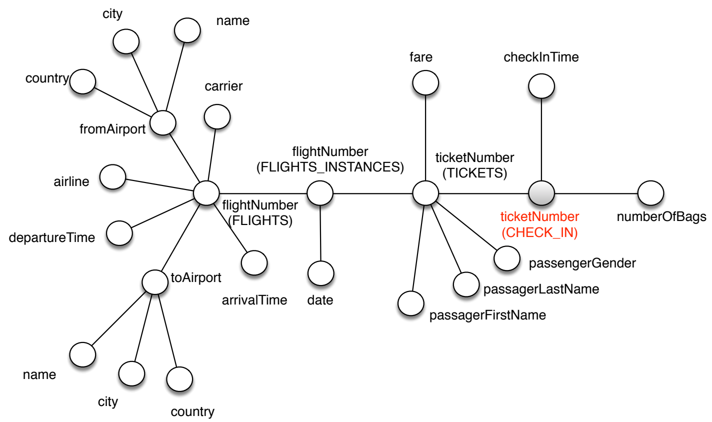
Building Attribute Trees: Flight example (3)

Attribute Tree 3 (TICKETS):



Building attribute Trees: Flight example (4)

Attribute Tree 4 (CHECK_IN):



Facts TICKETS and CHECK_IN are the best choices because existing functional dependencies permit to include a maximum of attributs in trees 3 and 4.

3. Buiding the Fact Schema

For each fact:

- **3.1. Pruning and grafting the attribute tree:**
- **3.2. Defining Fact Schema with its dimensions (fact dimensions)**
- **3.3. Defining Fact Schema measures (fact attributes)**
- **3.4. Defining Fact Schema granularity of data (dimension hierarchies)**
- **3.5. Draw the Fact Schema**

The step to derive DF schemata from E/R schema is very similar: the main difference concerns the algorithm used to build the attribute tree

3.1. Pruning and grafting the attribute tree (1)

For each fact:

Some attributes in the tree maybe uninteresting for the DW :

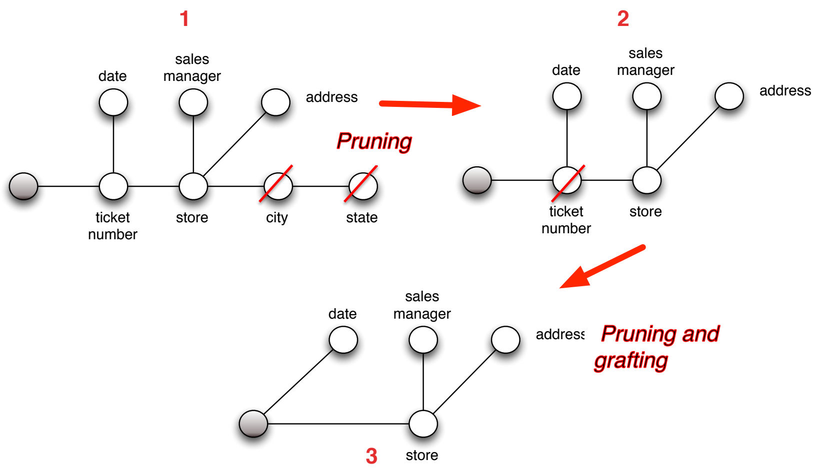
- We can *retain or graft any nodes corresponding to composite keys*
- We can *modify, add, or delete a functional dependency*
- We can add one or more functional dependencies if a non-normalized table exists in the relational schema

In order to drop useless levels of detail, it is possible to apply the following operators:

- ***Pruning**: delete a node and its subtree.*
- ***Grafting**: delete a node and move its subtree. It is useful when an attribute is not interesting but the attributes it determines must be preserved.*

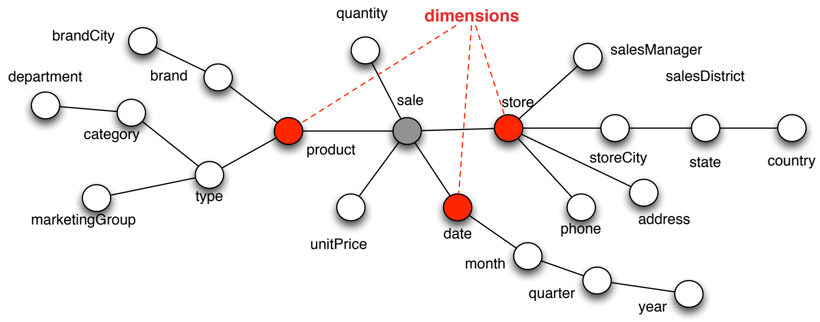
3.1. Pruning and grafting the attribute tree (2)

.



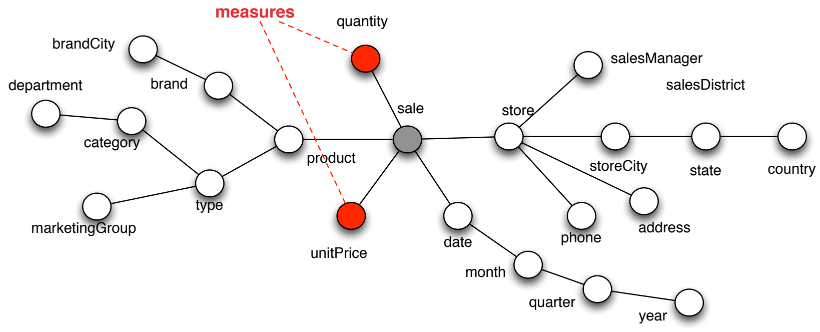
3.2. Defining dimensions

- The choice of dimensions determines the fact ***granularity***
- Dimensions must be ***chosen among the root children*** in the attribute tree.
- ***Time*** should **always** be a ***dimension***



3.3. Defining Measures

- Measures must be ***chosen among the children of the root***
- Measures are typically ***computed*** either by ***counting the number of instances of F***, or by ***summing (averaging, ...) expressions*** which involve ***numerical attributes***
- An attribute ***cannot be both a measure and a dimension***
- A **fact** may **have no measures**



3.4. Defining Granularity

Granularity of data :

- Primary issue in determining ***performance***

- ***depends on the queries users are interested in***
- represents a ***trade-off*** between query ***response time*** and ***detail of information to be stored*** :
  - It may be worth adopting a finer granularity than that required by users, provided that this does not slow down the system too much
  - Constrained by the maximum time frame for loading
- **Choosing granularity** includes defining the ***refresh interval*** that needs to consider :
  - Availability of operational data
  - Workload characteristics
  - The total time period to be analysed

3.5. Draw the Fact Schema ...

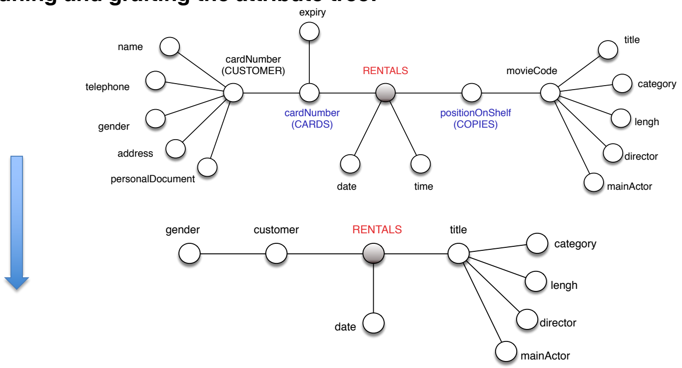
Definig fact schema: DVD example (1)

Relational schema of the DVD rental BD:

- CARDS (<u>cardNumber</u>, expiry)
- CUSTOMERS (<u>cardNumber</u>:CARDS, name, gender, address, telephone, personalDocument)
- MOVIES (<u>moviesCode</u>, title, category, director, lengh, mainActor)
- COPIES (<u>positionOnShelf</u>, movieCode:MOVIES)
- RENTALS (<u>positionOnShelf</u>:COPIES, <u>cardNumber</u>:CARDS, date, time)

Definig fact schema: DVD example (2)

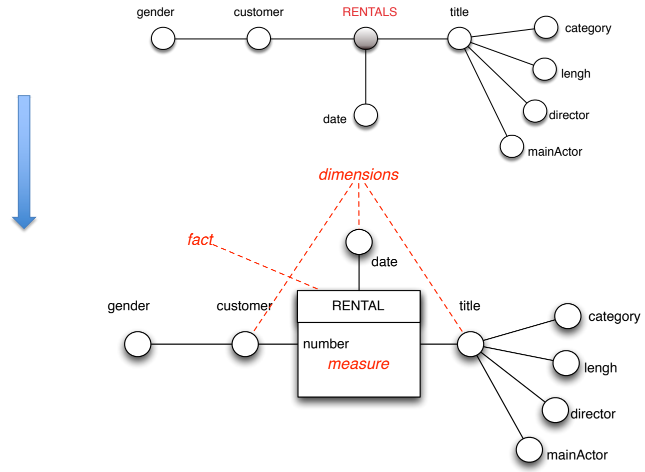
3.1: Pruning and grafting the attribute tree:



- *movieCode* and *Title* are inverted
- *cardNumber(CARDS)* and *name* (renamed customer) are inverted
- *positionOnShelf(COPIES)* and *cardNumber(CARDS)* are grafted
- *time, expiry, telephone, address, personalDocument, movieCode and cardNumber(CUSTOMERS)* are pruned

Definig fact schema: DVD example (3)

Fact schema "RENTAL":



Definig fact schema: DVD example (4)

SQL measure glossaries for fact schema "RENTAL":

```
number = SELECT COUNT (*)
          FROM RENTALS R INNER JOINT COPIES C
            ON R.positionOnShelf = C.positionOnShelf,
            COPIES C INNER JOINT MOVIES F
            RENTALS R INNER JOINT CUSTOMERS C
            ON R.cardNumber = C.cardNumber
         GROUP BY F.title, R.date, C.name;
```


Definig fact schema: Flight example (1)

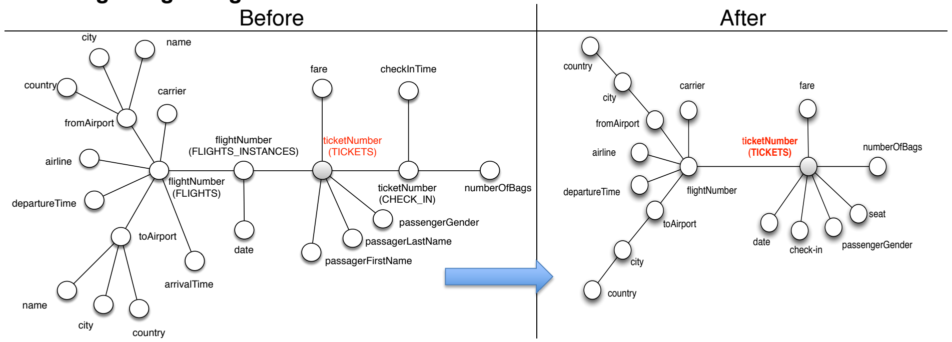
Relationnal logical schema describes an **operational DB** for Fights:

- FLIGHTS (flightNumber, airline, fromAirport:AIRPORTS)
- FLIGHT_INSTANCES (FlightNumber:FLIGHTS, date)
- AIRPORTS (IATAcode, name, city, country)
- TICKETS (ticketNumber, flightNumber:FLIGHT_INSTANCES), seat, fate, passengersFirstName, passengersSurname, passengersGender)
- CHECK-IN (ticketNumber:TICKETS, CheckInTime, numberOfBags)

Fact "TICKET ISSUE"

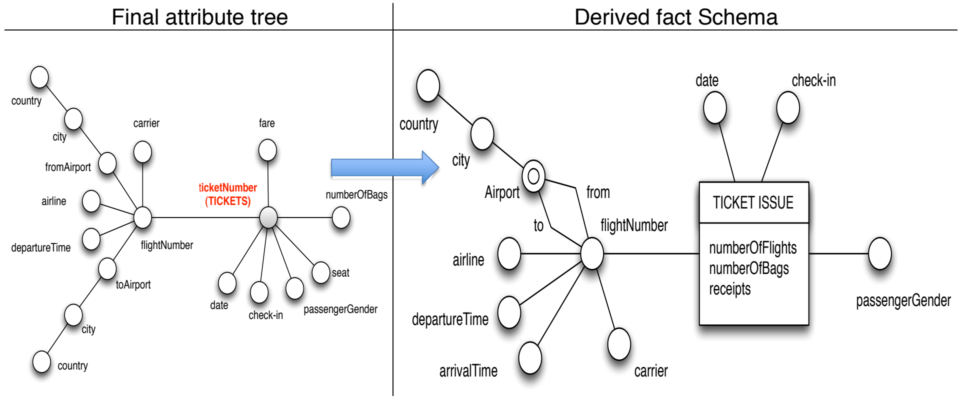
Definig fact schema: FLIGHT example (2)

Pruning and grafting the attribute tree:



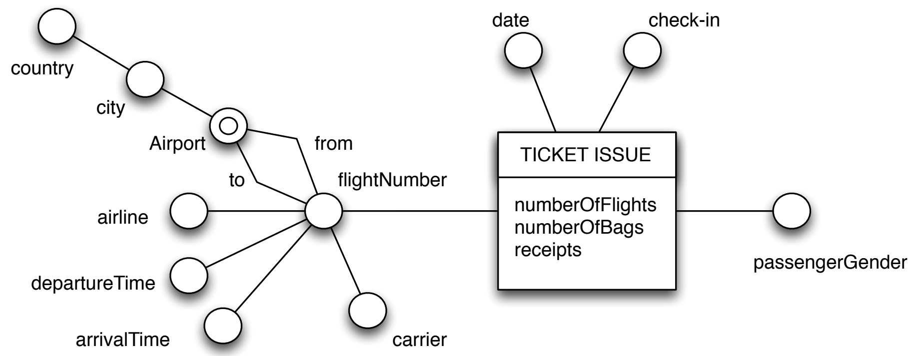
- *country* is now the child of *city*
- *checkIn* is now a *bolean added* on the tree when *number* node was grafted: ist value is TRUE only for tickets whose passengers have checked in.

Definig fact schema: FLIGHT example (3)



Definig fact schema: FLIGHT example (4)

Fact schema "TICKET ISSUE":



Defining fact schema: Flight example (5)

SQL measure glossaries for fact schema "TICKET ISSUE":

```
numberOfFlight = SELECT COUNT (*)
           FROM TICKETS T INNER JOINT FLIGHT_INSTANCES I
             ON T.flightNumber = I.flightNumber AND T.date = I.date
           GROUP BY T.passengerNumber, I.date, T.flightNumber;

numberOfBags = SELECT SUM (C.numberOfBag)
           FROM TICKETS T INNER JOINT FLIGHT_INSTANCES I
             ON T.flightNumber = I.flightNumber AND T.date = I.date
             TICKETS T INNER JOINT CHECK_IN C
             ON T.ticketNumber = C.ticketNumber
           GROUP BY T.ticketNumber, I.date, T.flightNumber;

receipts = SELECT SUM (T.fare)
           FROM TICKETS T INNER JOINT FLIGHT_INSTANCES I
             ON T.flightNumber = I.flightNumber AND T.date = I.date
           GROUP BY T.passengerGender, I.date, T.flightNumber;
```

The check-in dimension was left out to avoid making the query too complex.