Data Warehouse
Logical Modeling and Design
(6)

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

January, 2020

Methodological framework
Logical Modeling: The Multidimensional Model
Logical Design : From Fact schema to ROLAP Logical schema
ROLAP schema in MDX for Mondrian

Outline

Bibliographie

- **Livres**
  - 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, ISBN : 2-7117-4811-1.
  - Franco J-M., « Le Data Warehouse ». Ed. Eyrolles, Paris, 1997. ISBN 2-212-08956-2.
- **Cours**
  - Course of M. Golfarelli M. and S. Rizzi, University of Bologna
  - Course of M. Böhlen and J. Gamper J., Free University of Bolzano
  - …

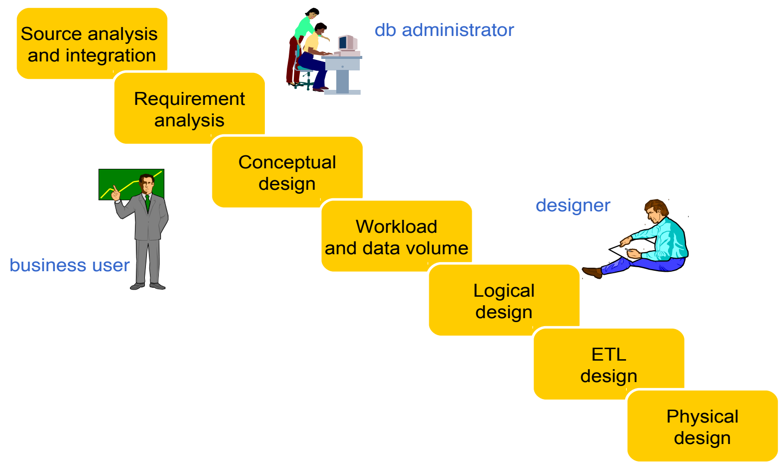
1. Methodological framework

- **Conceptual Design & Logical Design**
- **Life-Cycle**
- **Design Phases**
- **Schemata derivations**

Conceptual Design & Logical Design

- **Entite-Relation models** are not very useful in modeling DWs
- Is now universally recognized that a DW is based on a **multidimensional view of data :**
  - *But there is still no agreement on HOW to implement its conceptual design !*
- **Most of the time, DW design is at the logical level** : a multidimensional model (star/snowflake schema) is directly designed :
  - *But a* ***star schema*** *is nothing but a* ***relational schema****: it contains only the definition of a set of relations and integrity constraints !*
- **A better approach:**
  - 1) **Design** first a **conceptual model** : **Conceptual Design**
  - 2) Then ***translate this conceptual model*** into a **logical model : Logical Design**

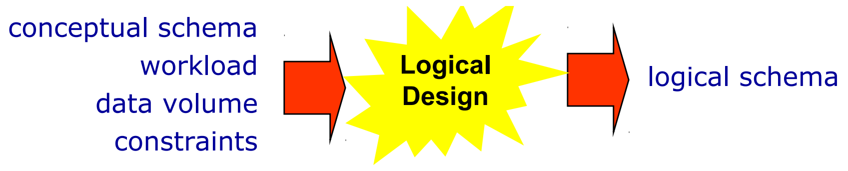
Design Phases



Problematic of the Logical Design

The **Logical Design** transforms the Conceptual Schema for a DM into a **Logical Schema** :

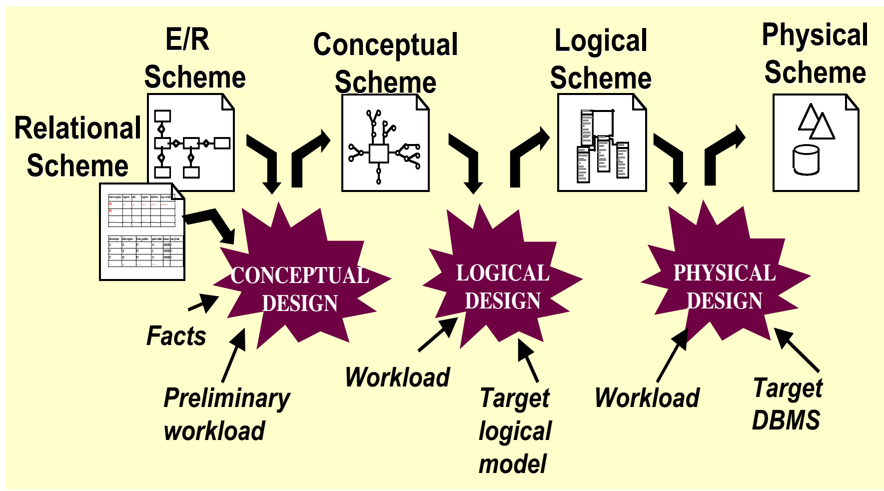
- Choice of the **type of logical schema**
- Translation of conceptual schema to logical schema
- Optimization (view materialization, fragmentation)



Different principles from the one used in operational databases :

- data redundancy
- denormalization of tables

Schemata derivations



2. Logical Modeling: The Multidimensional Model

- **The Multidimensional Model:**
  - **Fact,**
  - **Measures,**
  - **Dimensions**
- **Star and Snowflake Schemata**
- **Aggregates and Views**

The Multidimensional Model

Multidimensional Model :

- is a **logical model**
- has one purpose: **Data analysis**
- is the **most popular data model** for DW
- is more **built in "meaning"** :
  - What **is important**
  - What **describes the important**
  - What we want to **optimize**
  - Automatic aggregations means **easy querying**
- is **recognized by OLAP/BI tools** : Tools offer powerful query facilities based on MD design

The Multidimensional Model: Fact and Dimensions

- Data is divided into **facts** (with **measures**) and **dimensions**
- **Facts**
  - are the important entity, e.g., a sale
  - have measures that can be aggregated, e.g., sales price
- **Dimensions :**
  - describe facts
  - Ex : a sale has the dimensions Product, Store and Time
- **Goal for dimensional modeling:**
  - Surround facts with as much context/dimensions as possible (redundancy may be ok in well-chosen places)
  - But you should **not** try to model **all** relationships in the data (unlike E/R and OO modeling!)

Facts (data) "live" in a multidimensional « **cube** »

Facts and subject

Facts represent the **subject** of the desired analysis : the "important" in the business that should be analyzed :

- A fact is most often identified via its **dimension values** :
  - A fact is a **non-empty cell**
  - Some models give facts an **explicit identity**
- Generally, a fact should :
  - be attached to **exactly one** dimension value in each dimension;
  - only be attached to dimension values in the **bottom levels**
  - *Ex : if the lowest time granularity is day, for each fact the exact day should be specified*
  - some models do not require this.

Types of Facts

Different **types of facts** are distinguished :

- **Event facts (transaction)** : ***a fact for every business event*** *(Ex : sale)*
- **« Fact-less » facts** :
  - A fact per event *(Ex : customer contact)*
  - No numerical measures
  - ***An event happened for a dimension value combination***
- **Snapshot fact** :
  - A fact for every dimension combination at given time interval
  - ***Captures current status*** *(Ex : inventory)*
- **Cumulative snapshot facts** :
  - A fact for every dimension combination at given time interval
  - ***Captures cumulative status up to now*** *(Ex : sales to date)*

Every type of facts answers different questions : often event facts and snapshot facts exist

Dimension

- **Dimensions** are the core of multidimensional databases
- Other types of databases do not support dimensions
- Dimensions are used for :
  - **Selection** of data
  - **Grouping** of data at the right level of detail
- Dimensions consist of **dimension values**

Ex :

- *Product dimension has values "milk", "cream", ...*
- *Time dimension has values "1/1/2001", "2/1/2001",..*

- Dimension values may have an **ordering** :
  - Used for comparing cube data across values
  - Especially used for Time dimension

Ex : percentage of sales increase compared with last month

Dimension and hierarchies

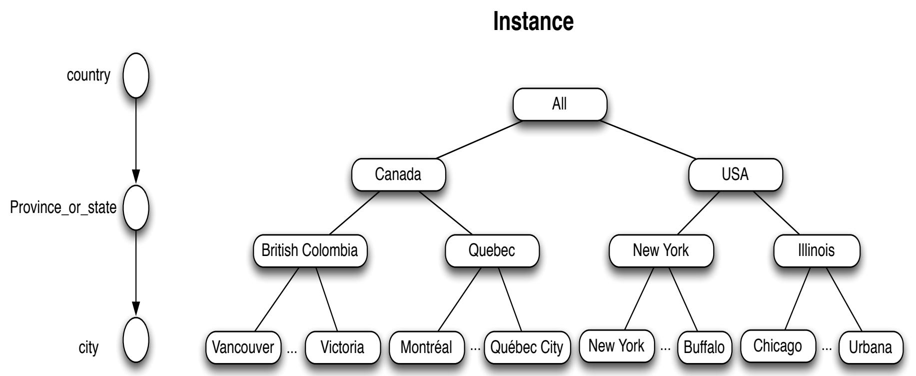
- Dimensions encode **hierarchies** with **levels** : Typically 3-5 levels (of detail)
- Dimension values are organized in a **tree structure** or **lattice** :
  - *Ex :*

    Product: Product -> Type -> Category
    Store: Store -> Area -> City > County
    Time: Day -> Month -> Quarter -> Year
- Dimensions have a **bottom level** and a **top level (ALL)**
- **Levels** may have **attributes** simple, non-hierarchical information
  - *Ex : Day has Workday as attribute*
- **General rule**: **dimensions should contain much information** :
  - *Ex :*

    Time dimensions may contain holiday, season, events,...
    Good dimensions have 50-100 or more attributes/levels

Hierarchy example

- A location dimension with attributes *street, city, province_or_state,* and *country* encodes implicitly the following hierarchy :



The Multidimensional Model: Cube

- A **cube** may have **many dimensions** :
  - More than 3 – the term "hypercube" is sometimes used
  - Theoretically no limit for the number of dimensions
  - Typical cubes have 4-12 dimensions
- But only 2-3 dimensions can be viewed at a time :
  - Dimensionality reduced by queries via projection/aggregation
- A cube consists of **cells** :
  - A given combination of dimension values
  - A cell can be empty (no data for this combination)
  - A **sparse** cube has many empty cells
  - A **dense** cube has few empty cells
  - Cubes become **sparser** for many/large dimensions.

Granularity of facts

- **Granularity of facts** is important :
  - What does a single fact mean?
  - Determines the **level of detail**
  - Given by the combination of bottom levels
  - *Ex : "total sales per store per day per product"*
- Important for number of facts : ***Scalability***
- Often the granularity is a single business transaction :
  - *Ex : sale*
  - Sometimes the data is aggregated (total sales per store per day per product)
  - Aggregation might be necessary for scalability

Measures

- **Measures** represent the fact property that users want to **study** and **analyze :**
  - Ex : the total sales price
- A measure has 2 components :
  - **Numerical value** (ex : sales price)
  - **Aggregation formula** (ex : SUM): used for ***aggregating*** / ***combining*** a number of measure values into one
- **Additivity** is an important property for measures :
  - Single fact table rows are (almost) never retrieved, but aggregations over millions of fact rows
- Measure value determined by the combination of dimension values :
  - Measure value is ***important for all aggregation levels***.

Types of Measures

3 types of measures are distinguished :

- **Additive measures** :
  - ***Can be aggregated over all dimensions*** using **SUM** operator
  - *Ex : sales price, gross profit computed from sales and cost*
  - *Often occur in event facts*
- **Semi-additive measures** :
  - ***Cannot be aggregated over some dimensions*** – typically time
  - *Ex : inventory: additive across time or store, non-additive accross product*
  - *Often occur in snapshot facts*
- **Non-additive measures** :
  - ***Cannot be aggregated over any dimensions***
  - *Ex : unit cost*
  - *Occur in all types of facts*

General DW Design Steps

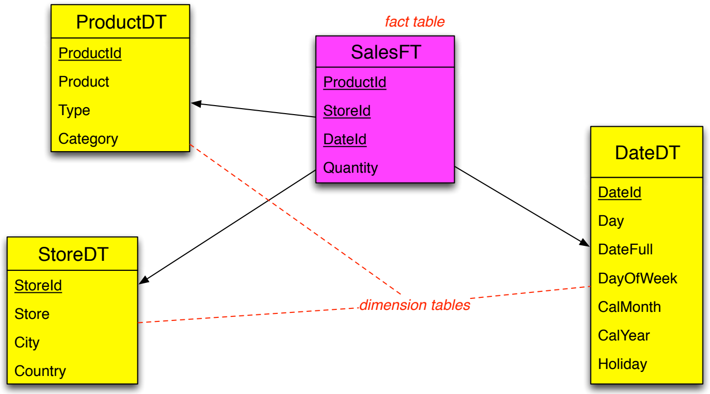
- **1. Choose the business process(es) to model :**
  - *Ex : Sales*
- **2. Choose the granularity of the business process :**
  - *Ex : Items by Store by Promotion by Day*
  - *Low granularity is needed ?*
  - *Are individual transactions necessary/feasible ?*
- **3. Choose the dimensions :**
  - *Ex : Time, Store, ...*
- **4. Choose the measures :**
  - *Ex : Dollar_sales, unit_sales, dollar_cost, customer_count*

3. Implementing a Dimensional Model in ROLAP

- **Star schema**
- **Snowflake schema**
- **Aggregates and views**

Star Schema

- Is a **common approach** to draw a dimensional model
- Consists of : **one fact table** and **many dimension tables :**



Dimension Schema and Instance in Star Schema

Date
DateKey
Date
DateFull
DayOfWeek
CalMonth
CalYear
Holiday

schema of dimension Date

instance of dimension Date

Date

DateKey	Date	DateFull	DayOfWeek	CalMonth	CalYear	Weekday
1	01/01/02	Januar 1, 2002	Tuesday	January	2002	Weekday
2	01/02/02	Januar 2, 2002	Wednesday	January	2002	Weekday
3	01/03/02	Januar 3, 2002	Thursday	January	2002	Weekday
4	01/04/02	Januar 4, 2002	Friday	January	2002	Weekend
5	01/05/02	Januar 5, 2002	Saturday	January	2002	Weekend

Relational “Star Schema” Evaluation

Forces :

- **Simple** -> ease-of-use
- Relatively **flexible**
- **Fact table is normalized** (dimensions tables are not necessary normalized)
- **Dimension tables often relatively small**
- “Recognized” by many RDBMS -> **good performance**

Weakness :

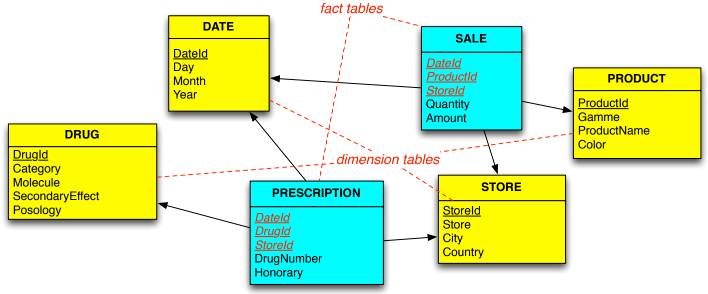
- **Hierarchies are ”hidden” in the columns**
- **Dimension tables are de-normalized**

=> From relational Star schema to relational « Snowflake » schema

Constellation Schema

- **Merge several star schemata, which use common dimensions**
- Has consequently **several facts** and **dimensions common or not**

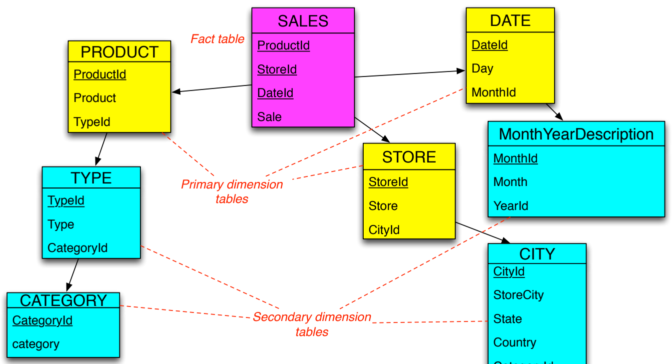
Ex : Drugs sales in pharmacies



- Make up of **2 star-schemas** :
  - first concerning sales in pharmacies (SALE)
  - second concerning medecin prescription (PRESCRIPTION)
- ***DATE*** and ***STORE*** dimensions are shared by PRESCRIPTION and SALE facts.

Snowflake Schema

- Is **obtained from a star schema** by breaking down one or more dimension tables into smaller tables to remove transitive functional dependencies
- Dimension table referenced in the fact table are « primary dimension tables » and other are « secondary dimension tables »



Snowflake Schema Evaluation

Forces :

- **Hierarchies are made explicit/visible**
- **Very flexible**
- **Dimension tables use less space :**
  - However this is a minor saving
  - Disk space of dimensions is typically less than 5 percent of disk for DW

Weakness :

- **Harder to use due to many joins**
- **Worse performance :**

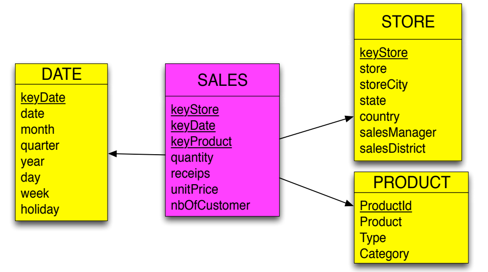
Ex : efficient bitmap indexes are not applicable

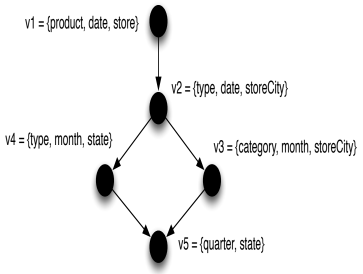
Aggregates and Views

View = Fact table that include Aggregate Data

- **Primary view : defined in the fact schema dimensions** (primary group-by sets), **populated by operational data**
- **Secondary views : new fact tables including aggregate**(s) (secondary group-by sets), **populated by others views** (not by operational data)

star schema | some sales schema views





- v1 = *primary view* and v2 … v5 = *secondary views*
- if vi -> vj then *vj can be calculated by aggregating vi data*

Relational Schema with Aggregate Data (1)

Solution 1 : the easier solution in a *star schema*, is to *store both primary view data and secondary view data in the same fact table* :

SALES (fact table)

keyStore	keyDate	keyProduct	quantity	receipts	…
1	1	1	170	85	…
2	1	1	300	150	…
3	1	1	1700	850	…
…	…	…	…	…	…

STORE (dimension table)

(- = NULL)

keyStore	store	storeCity	state	…
1	Coop1	Colombus	Ohio	…
2	-	Austin	Texas	…
3	-	-	Texas	…
…	…	…	…	…

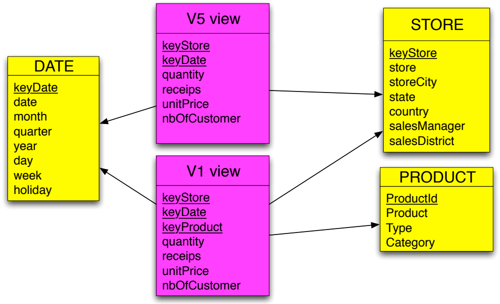
Aggregation level of fact table tuples specified by corresponding tuples in dimension table :

- the dimension table STORE related to aggregate data will have NULL values in all attributes whose aggregation level is finer than the current one
- in the fact table SALES :
  - *first tuple is related to one sale,*
  - *second tuple store aggregate data for every sale in Austin,*
  - *third tuple sums up all the sales in Texas.*

only fact table is used for request, **but performance are bad because table is too big**

Relational Schema with Aggregate Data (2)

Solution 2: a common solution is to store *different group-by sets into separate fact tables* in a *constellation schema* :

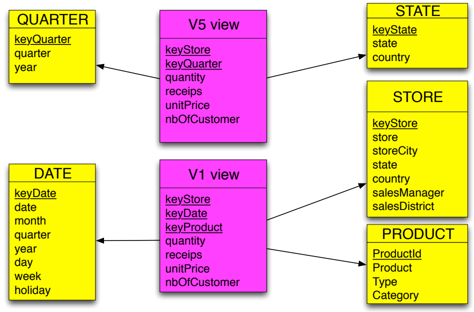


- here a fact table concerne **primary view V1** and an other **V5 secondary view**
- dimension tables can be merged as in solution 1, or replicate for each aggregate view
- the fact tables that correspond to the group-by sets where one or more dimensions are completely aggregated do not have foreign key referencing these dimensions

the size of fact table size is far larger than size of the dimension tables, **then performance depends of the fact table optimisation**

Relational Schema with Aggregate Data (3)

Solution 3: dimension tables are *replicated* for every view including only the set of attributes that are valide for the aggregation level at which that dimension table is created :

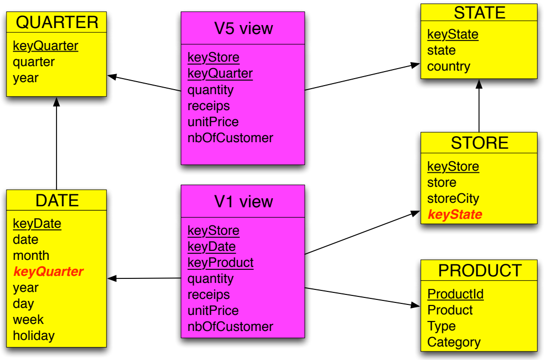


- here are **V1 primary view** and **V5 secondary view** with their specific dimension tables
- note that the key of the fact table for secondary view V5 has not attribute related to store hierarchy, which is completely aggregated

this solution is the **best in performance** because the table acces are optimized, however replicate dimension tables need disk space

Relational Schema with Aggregate Data (4)

Solution 4: a compromise, where aggregate views are materialized in a *snowflake schema* :



- take advantage of the optimization achieved with aggregate data by aggregation level without replicating dimension table

the sales fact is modeled by a snowfake schema

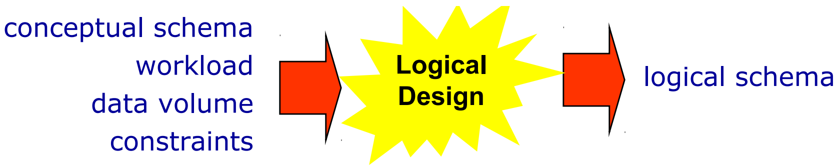
4. Logical Design: From Fact Schema to Logical Schema in ROLAP context

- **From fact schema to relational star-schema: basic rules**
- **Examples towards Relational Star Schema**
- **Examples towards Relational Snowflake Schema**
- **Advanced logical modelling**

Problematic of the Logical Design

The **Logical Design** transforms the Conceptual Schema for a DM into a **Logical Schema** :

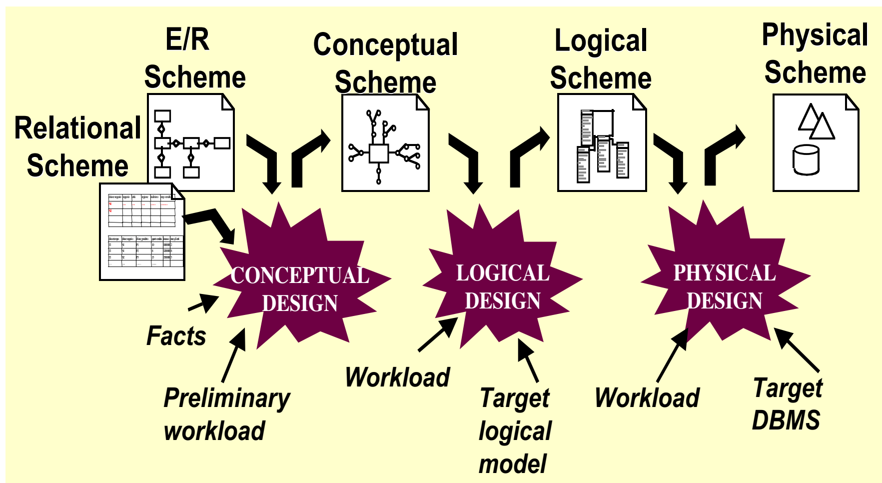
- Choice of the **type of logical schema**
- Translation of conceptual schema to logical schema
- Optimization (view materialization, fragmentation)



Different principles from the one used in operational databases :

- data redundancy
- denormalization of tables

Schemata derivations (1)



Schemata derivations (1)

Conceptual Design :

Step to derive Dimensional Fact (DF) schemata from Relational schema of operational sources (DB) :

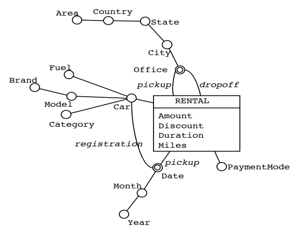
- *1. Finding and defining FACT from Relational/E-R schema*
- *2. Building the Attribute Tree from Relational/E-R schema*
- *3. Building the Fact Schemata from Attribute Tree*

Logical Design :

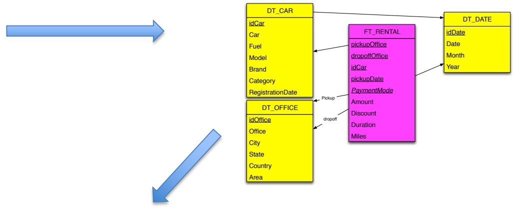
- ***transforms the Fact Schemata into a Logical Schema, a Relational Schema (in ROLAP strategy)***

Schemata derivations (3)

Fact Schema



Relational Schema



snowflake schema :

- DT_CAR (idCar, Car, Fuel, Model, Brand, Category, registrationDate:DT_DATE)
- DT_OFFICE (idOffice, Office, City, State, Country, Area)
- DT_DATE (idDate, Date, Month, Year)
- FT_RENTAL (pickupOffice:DT_OFFICE, dropoffOffice:DT_OFFICE, idCar:DT_CAR, pickupDate:DT_DATE, PaymentMode, Amount, Discount, Duration, Miles)

From Fact Schema (DFM) to Relational Schema: Basic Rules

R1. From « Fact schema » to « Tables » :

- ***Fact box*** *of the fact schema leads to create a* ***Fact Table (FT)***
- ***dimension*** *leads to create a* ***Dimension Table (DT)***

R2. Dimension table (DT) attributes :

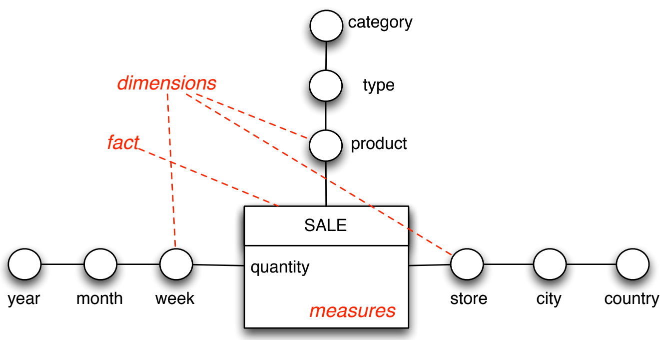
- ***dimension attribute*** *of fact schema become* ***DT attribute***
- *the* ***first dimension attribute*** *(first level) become* ***DT key attribute***

R3. Fact table (FT) attributes :

- ***DT key attribute*** *of each dimension become* ***FT foreign key***
- ***measure attributes*** *of fact schema become* ***FT measure***

Relational Star-Schema "Sale" example (1)

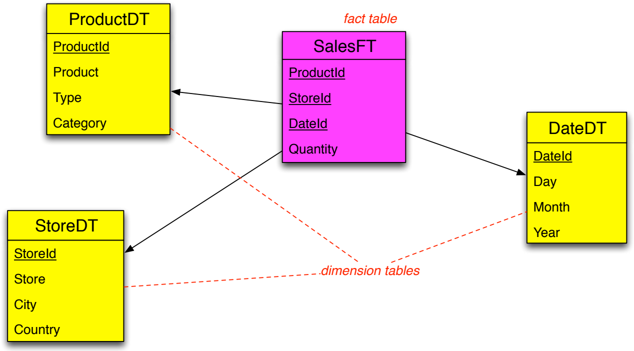
Fact schema :



Relational Star-Schema “Sale” example (2)

Relational schema :

- SaleFT (<u>ProductId, StoreId, DateId</u>, Quantity)
- WeekDT (<u>ProductId</u>, Product, Type, Category)
- StoreDT (<u>StoreId</u>, Store, City, State, Country)
- DateDT (<u>DateId</u>, Day, Month, Year)



Relational Star-Schema “Sale” example (3)

Instances :

ProductDT

ProductId	Product	Type	Category
1	Bud	Beer	Beverage
2	Forst	Beer	Beverage
3	Warst	Beer	Beverage

DateDT

DateId	Day	Month	Year
1	25	May	1997

SalesFT

ProductId	StoreId	DateId	Quantity
1	1	1	575

StoreDT

StoreId	Store	City	Country
1	Bilka	Aalborg	Denmark
2	Spar	Bolzano	Italy

Relational Star-Schema “Sale” example (4)

OLAP Query on Star schema :

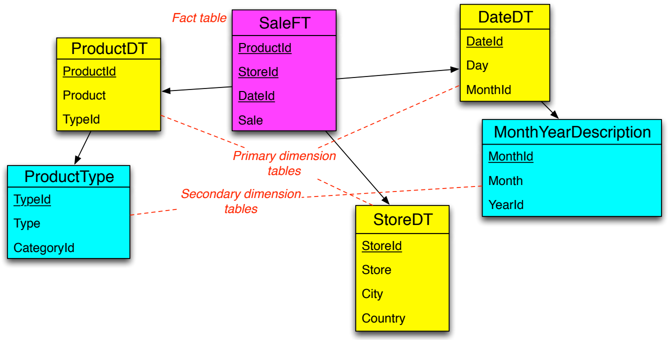
Total quantity sold for each product type, week, and city, only for food products :

```
SELECT City, Week, Type, SUM(Quantity)
 FROM WeekDT, StoreDT, ProductDT, SaleFT
  WHERE WeekDT.WeekID = SaleFT.WeekID AND
    StoreDT.StoreID = SaleFT.StoreID AND
    ProductDT.ProductID = SaleFT.ProductID AND
    ProductDT.Category = ‘Food’
GROUP BY City, Week, Type;
```


Relational Snowflake Schema “Sale” example (1)

Relational schema :

- SaleFT (<u>ProductId, StoreId, DateId</u>, Sale)
- ProductDT (<u>ProductId</u>, Product, TypeId)
- ProductType (<u>TypeId</u>, Type, CategoryId)
- StoreDT (<u>StoreId</u>, Store, City, Country)
- DateDT (<u>DateId</u>, Day, MonthId)
- MonthYearDescription (<u>MonthId</u>, Month, YearId)



Relational Snowflake Schema "Sale" example (2)

Instances :

Product Type

TypeId	Type	CategoryId
1	Beer	1

MonthYear Description

MonthId	Month	YearId
1	May	1997

ProductDT

ProductId	Product	TypeId
1	Bud	1
2	Forst	1
3	Bit	1

DateDT

DateId	Day	MonthId
1	25	1

SalesFT

ProductId	StoreId	DateId	Sale
1	1	1	5.75

StoreDT

StoreId	Store	City	Country
1	Bilka	Aalborg	Denmark
2	Spar	Bolzano	Italy

Relational Snowflake Schema "Sale" example (3)

OLAP Query on Snowflake schema :

Total quantity sold for each product type, week, and city, only for food products :

```
SELECT City, Week, Type, SUM(Quantity)
FROM WeekDT, StoreDT, ProductDT, CityDT, TypeDT, SaleFT
WHERE WeekDT.WeekID = SaleFT.WeekID AND
   StoreDT.StoreID = SaleFT.StoreID AND
   ProductDT.ProductID = SaleFT.ProductID AND
   StoreDT.CityID = CityDT.CityID AND
   ProductDT.TypeID = TypeDT.TypeID AND
   ProductDT.Category = 'Food'
GROUP BY City, Week, Type;
```


Advanced Logical Modeling: Descriptives Attributes (1)

Conceptual Modelling:

- A *descriptive attribute* contains additional information about an attribute of the hierarchy
- It cannot be used for aggregation

Logical Modelling:

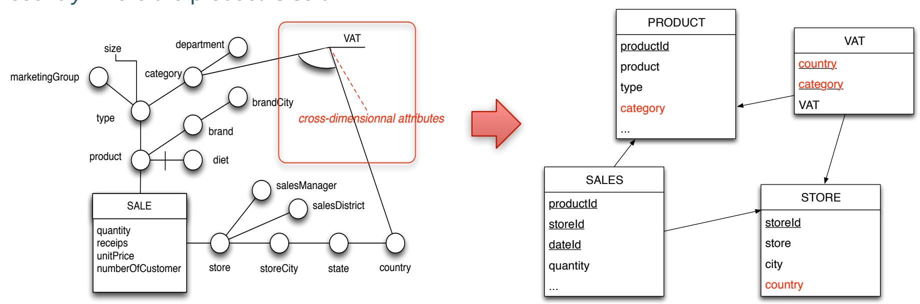
- If a *descriptive attribute* is linked to a *dimensional attribute*, it have to be included in the dimension table for the hierarchy that contains it

  *Ex : the **size** of the product have to be included in PRODUCT table*

- If a *descriptive attribute* is linked to a *fact attribute*, it have to be included in the fact table with measures

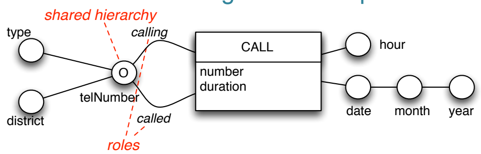
Advanced Logical Modeling: Descriptives Attributes (2)

Conceptual Modelling: A *cross-dimensional attribute b* is a *dimensionnal* or *descriptive attribute* whose value is defined by the combination of 2 or more dimensional attributes a1, …am, possibly belonging to different hierarchies.
Logical Modelling: translation leads to a new table that includes the *b cross-dimensional attribute* and has the *a1, …am attributes as primary key*
Ex : a product Value Added Tax (VAT) depends both on the product *category* and on the *country* where the product is sold

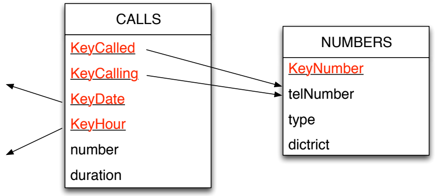


Advanced Logical Modeling: Shared Hierarchies

Conceptual Modelling: Entire *portion of hierarchies are frequently replicated* 2 or more time in fact schemata. Ex: calling and called phone numbers …



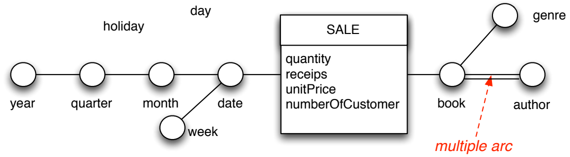
Logical Modelling: we have to avoid multiples dimension tables, which contain all or part of the same data. Ex: the hierarchies contain all the same data, we choose to insert 2 foreign keys referencing the one table that models the telephone number in the fact table:



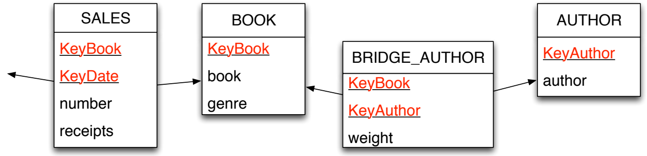
Advanced Logical Modelling: Multiple Arcs

Conceptual Modelling: A ***multiple arc*** models a **many-to-many association between the 2 dimensional attributes** it connects

Ex : in a fact schema modeling the sales of books, book is a dimension, but it would not be relevant to model author as a dimensional child attribute of book because many different authors can write many books => the relationship between books and authors is modeled as a *multiple arc*:



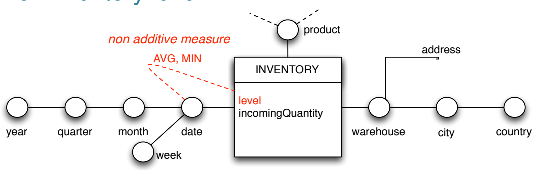
Logical Modelling: solution is to insert a bridge-table to model multiple arcs:



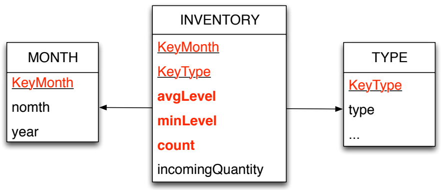
Advanced Logical Modelling: Additivity Issues

Conceptual Modelling: Non-additive measure can be explicitely specify with its operator(s) used for aggregation – other that SUM

Ex: AVG and MIN for inventory level:



Logical Modelling: set a new mesure for each aggregation operator.



- ***Count*** is a support measure necessary to calculate average level
- ***minLevel*** is required to calculate the minimum level for each month and for each product type.

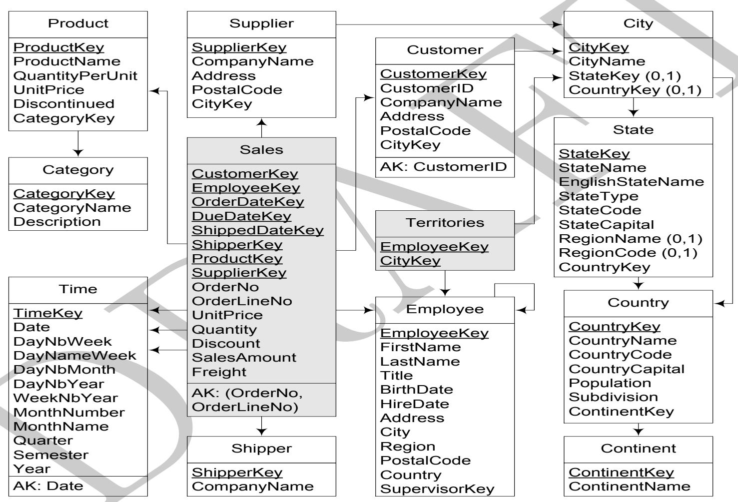
4. ROLAP schema in XML for Mondrian

- **Exampl**
- **Examples towards Relational Star Schema**
- **Examples towards Relational Snowflake Schema**
- **Advanced logical modelling**

The Mondrian ROLAP server

- **Mondrian** is an **open-source relational online analytical processing (ROLAP) server**
- It is also known as **Pentaho Analysis Services** and is a component of the **Pentaho Business Analytics suite**
- In Mondrian, a **cube schema,** written in an **XML syntax,** defines a mapping between the physical structure of the relational data warehouse and the multidimensional cube
- A **cube schema** contains the declaration of cubes, dimensions, hierarchies, levels, measures, and calculated members
- A **cube schema** does **not define** the **data source**, this is done using a JDBC connection string.

Example of a ROLAP DW star schema



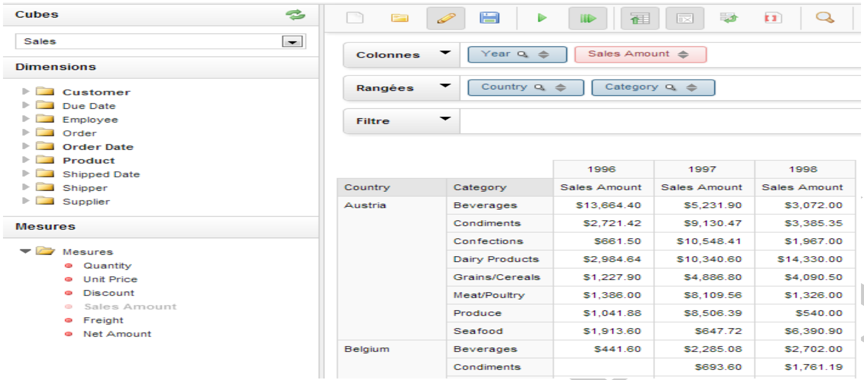
A cube schema definition in Mondrian in XML (1)

```
1.  <Schema name="MySocietyDW"
2.   description="Sales cube of MySociety company">
3.     <PhysicalSchema>
4.     ...
5.     </PhysicalSchema>
6.     <Dimension name="Time" table="Time" ... >
7.     ...
8.     </Dimension>
9.     <Cube name="Sales">
10.          <Dimensions>
11.          ...
12.          </Dimensions>
13.          <MeasureGroups>
14.              <MeasureGroup name="Sales" table="Sales">
15.                  <Measures>
16.                  ...
17.                  </Measures>
18.                  <DimensionLinks>
19.                  ...
20.                  </DimensionLinks>
21.              </MeasureGroup>
22.          </MeasureGroups>
23.       </Cube>
24.  </Schema>
```


The Mondrian ROLAP server (2)

- The Schema element (starting in line 1) defines all other elements in the schema
- The **PhysicalSchema element** (lines 3–5) defines the **tables that are the source data** for the dimensions and cubes, and the foreign key links between these tables
- The **Dimension element** (lines 6–8) defines the **shared dimension** *Time*, which is used several times in the **MySociety cube** for the role-playing dimensions *OrderDate, DueDate*, and *ShippingDate*. Shared dimensions can also be used in several cubes
- The **Cube element** (lines 9–23) define the **Sales cube.** A cube schema contains *dimensions* and *measures*, the latter organized in *measure groups*
- The **Dimensions element** (lines 10–12) defines the *dimensions* of the **cube**
- The **measure groups** are defined using the element *MeasureGroups* (lines 13–22)
- The **measure group Sales** (lines 14–21) defines the *measures* using the *Measures element* (lines 15–17)
- The **DimensionLinks element** (lines 18–20) defines how measures relate to dimensions.

Browsing the cube with Saiku



Saiku is an **open-source analytics client server**. In the figure :

- the *countries* of *customers* and the *categories* of *products* are displayed on *rows*, the *years* are displayed on *columns*, and
- the *sales amount measure* is displayed on *cells*.

Saiku also allows the user to **write OLAP queries in MDX** directly on an editor.