Methodological Framework

Conceptual Modelling: the Dimensional Fact Model (DFM)

Conceptual Design: from Relational schema to DFM

Plan

1. Methodological Framework
   - Conceptual Design & Logical Design
   - Top-Down Versus Botton-Up Approach
   - Design Phases and schemata derivations

2. Conceptual Modelling: The Dimensional Fact Model (DFM)
   - Fact schema
   - Dimension hierarchies
   - Additive, semi-additive and non-additive attributes
   - Overlapping compatible fact schemata
   - Representing query patterns on a fact schema

3. Conceptual Design: From Relational schema to DFM of Data Mart
   - Finding and defining facts from Relational schema
   - Building the Attribute Tree from Relational schema
   - Building the Fact Schema from Attribute Tree

Bibliographie

- Books

- Courses
  - Course of M. Golfarelli M. and S. Rizzi, University of Bologna
  - Courses of M. Böhlen and J. Gamper J., Free University of Bolzano
**Conceptual Design & Logical Design**

- **Entity-Relation models** are not very useful in modeling DWs
- DW is conceptually based on a multidimensional view of data:
  - But there is still **no agreement** on HOW to develop 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/snowflake 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) which is then **translated** into a logical model: Logical Design

---

**Methodological Framework**

- Building a DW is a **very complex task**, which requires an **accurate planning** aimed at devising satisfactory answers to organizational and architectural questions
- A large number of organizations **lack experience and skills** that are required to meet the challenges involved in DW projects
- Major cause of DW failures lies in the **absence of a global view of the design process**, of a **design methodology**
- **Design Methodologies** are necessary to minimizing the risks for failure

- **3 main strategies for DW design:**
  - Top-Down strategy
  - Bottom-Up strategy
  - Mixed strategy

---

**Top-Down Strategy**

**Analyze** global business needs, **plan** how to develop a DW, **design** it, and implement it as a **whole** with its DMs

**(+)^{Strengths:**
- Promising: it is **based on a global picture** of the goal to achieve, and in principle it ensures consistent, well integrated DW

**(-)^{Weakness:**
- High-cost estimates with long-term implementations discourage company managers from embarking on these kind of projects.
- Analyzing and integrating all relevant sources at the same time is a **very difficult task**: they are all available and stable at the same time.
- Extremely difficult to forecast the specific needs of every department involved in a project, which leads to specific DMs
- As no working DW system is going to be delivered in the short term, users cannot check for this project to be useful, so they lose trust and interest in it.
Phase 1: Goal setting and planning of the DW
- set system goals, borders, and size
- select an approach for design and implementation
- estimate costs and benefits
- analyze risks and expectations
- examine the skills of the working team

Phase 2: Infrastructure design of the DW
- analyze and compare the possible architectural solutions
- assess the available technologies and tools
- create a preliminary plan of the whole DW system

Phase 3: Design and development of Data Marts
- Every iteration causes a new DM and new applications to be created and progressively added to the DW system

Mixed Strategy

Top-Down and Bottom-Up strategies should be mixed:
- When planning a DW, a bottom-up strategy should be followed
- One Data Mart (DM) at a time is identified and prototyped according to a top-down strategy by building a conceptual schema for each fact
- The first DM (DM1) to prototype:
  - is the one playing the most strategic role for the enterprise
  - should be a backbone for the whole DW
  - should lean on available and consistent data sources

Botton-Up Strategy
- DW is incrementally built and several DM are iteratively created
- Each DM is based on a set of facts that are linked to a specific department and that can be interesting for a user group

(+) Strengths:
- Leads to concrete results in a short time
- Does not require huge investments
- Enables designers to investigate one area at a time
- Gives managers a quick feedback about the actual benefits of the system being built

(-) Weakness:
- Keeps the interest for the project constantly high may determine a partial vision of the business domain.

=> Mixed strategy ...

Main Data Warehouse/Mart Design Steps

Each Data Mart (DM) will be designed according these steps:
2. Conceptual Design of Data Mart: The Dimensional Fact Model

- Fact schema
- Dimension hierarchies
- Fact schema and fact instances
- Additive attributes
- Semi-additive and non-additive attributes
- Overlapping compatible fact schemata
- Representing query patterns on a fact schema

The Dimensional Fact Model (DFM) has been 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.

Conceptual Design of a Data Mart (DM)

- Conceptual Design is based on the documentation of the underlying operational information system (IS):
  - Relational schemata or
  - E/R schemata

- Steps:
  1. Find facts
  2. For each fact:
     a) Navigate functional dependencies
     b) Drop useless attributes
     c) Define dimensions and measures
**Exemple of Fact Schema**

Ex: a simple 3-dimensional fact schema « SALE » for a chain of stores:

- A **fact schema** is structured as a tree whose root is a fact
- A **Conceptual Model** of a DW consists of a set of **fact schemata**

**Fact, Measure and Dimension**

A **fact** is a *concept relevant* to decision-making processes:
- It models a set of **events** (ex: in a company: sales, shipments, purchases, ...)
- It has **dynamic properties** or evolve in some way **over time**
- It has one or more **numeric** and **continuously valued attributes** which "**measure**" the fact from different points of view

- a **measure** is a numerical property of a fact and describes a quantitative fact aspect that is relevant to analysis:
  Ex: every sale is quantified by its quantity, receipts, unitPrice, numberOfCustomer

- a **dimension** is a fact property with a finite domain and describes an analysis axes of the fact:
  Ex: typical dimensions for the sales fact are *product, store, and date*

**Dimension Hierarchies (1)**

- **Hierarchy** determines how fact instances may be aggregated and selected significantly for the decision-making process and determines the **granularity** adopted for representing facts.
- **Hierarchies** are subtrees rooted in **dimensions**:

**Dimension hierarchies (2)**

In dimension hierarchies:

- **nodes** represented by **circles** are **dimension attributes** which may assume a discrete set of values.
  Ex: week, month, product, ...
- **arcs** represent relationships between pairs of attributes: these relationships are functional dependencies:
  Ex: product -> type; type -> category; category -> department …
- **dimension attributes** in the nodes along each sub-path of the hierarchy starting from the dimension define **progressive granularities**.
**Advanced Modeling: Descriptives Attributes**

**non-dimension attributes** contains additional information about an attribute of the hierarchy: it cannot be used for aggregation! Ex: *size*: aggregating sales according to the *size* of the product would not make sense!

**Advanced Modeling: Optional Arcs**

**Optional arcs** (marked by a dash) express optional relationships between pairs of attributes (useful for logical design) Ex: *diet, promotion*. The *diet* attribute takes a value (such as cholesterol-free, gluten-free, or sugar-free) only for food products; for the other products, it is undefined.

**Advanced Modeling: Cross-Dimensional Attributes**

**Cross-dimensional attribute** is a dimensionnal or descriptive attribute whose value is defined by the combination of 2 or more dimensional attributes, possibly belonging to different hierarchies. Ex: if a product Value Added Tax (VAT) depends both on the product category and on the country where the product is sold, you can use a cross-dimensional attribute to represent it:

**Advanced Modeling: Convergence**

A **convergence** takes place when 2 dimensional attributes within a hierarchy are connected by 2 or more alternative paths of many-to-one associations (Graphically, use of arrows). Ex: in store dimension, store are grouped into sales districts and no inclusive relationship exists between districts and states, but each district is part of only one country:

Store -> salesDistrict -> country  
or  
Store -> storeCity -> state -> country
**Advanced Modeling: Shared Hierarchies**

*Shared hierarchies* exist when entire portion of hierarchies are frequently replicated 2 or more times in fact schemata.

In particular, in time hierarchies, 2 or more date-type dimensions with different meaning can easily exist in a same fact, and need to build a month-year hierarchy on each one of them.

=> an abbreviation is introduced

Ex: calling and called phone numbers …

**Advanced Modeling: Multiple Arcs**

*Multiple arc* models a many-to-many association between the 2 dimensional attributes it connects (Graphically, denoted by doubling of the arc).

Ex: in a fact schema modeling the sales of books, whose dimensions are date and book. It would certainly be interesting to aggregate and select sales on the basis of book authors.

However, it would not be accurate to model author as a dimensional child attribute of book because many different authors can write many books. Then, the relationship between books and authors is modeled as a multiple arc:

**Advanced Modeling: Additivity (1)**

3 Types of measure:
- Flow measure: refer to time (ex: number of products sold in a day)
- Level measure: evaluated at particular time (ex: number of products in inventory)
- Unit measure: evaluated at particular time but are expressed in relative terms (ex: product unit price, discount percentage)

Suitable operators for aggregation:

<table>
<thead>
<tr>
<th></th>
<th>Temporal hierarchies</th>
<th>Nontemporal hierarchies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow measures</td>
<td>SUM, AVG, MIN, MAX</td>
<td>SUM, AVG, MIN, MAX</td>
</tr>
<tr>
<td>Level measures</td>
<td>AVG, MIN, MAX</td>
<td>SUM, AVG, MIN, MAX</td>
</tr>
<tr>
<td>Unit measures</td>
<td>AVG, MIN, MAX</td>
<td>AVG, MIN, MAX</td>
</tr>
</tbody>
</table>

3 Natures of measure:
- additive along a dimension when can be used the SUM aggregation operator
- non-additive along a dimension if the aggregation operator is not SUM (ex: inventory level)
- a non-additive measure is non-aggregable if no operator exists (ex: unitPrice product)

**Advanced Modeling: Additivity (2)**

- Along all the dimensions by default measures are additive (operator SUM)
- Non-additive measure can be explicitly specified with its operator(s) used for aggregation – other that SUM (Ex: AVG and MIN for inventory level measure for time dimension)
Overlapping Compatible Fact Schemata (1)

- Different facts are represented in different fact schemata.
- Queries the user formulates on the DW may require comparing fact attributes taken from distinct, though related, schemata (drill across in OLAP).
- 2 fact schemata are said compatible if they share at least one dimension attribute.
- 2 compatible schemata $F$ and $G$ may be overlapped to create a resulting schema $H$.
- Without conflict between attribute dependencies in the 2 schemata:
  - the set of the fact attributes in $H$ is the union of the sets in $F$ and $G$.
  - the dimensions in $H$ are the intersection of those in $F$ and $G$, assuming that a given dimension is common to $F$ and $G$ if at least one dimension attribute is shared.
  - each hierarchy in $H$ includes all and only the dimension attributes included in the corresponding hierarchies of both $F$ and $G$.

Consider the 2 fact schemata:

- $F$ represents all employees of an enterprise.
- $G$ only the non-European employees.

Overlapping Compatible Fact Schemata (2)

$F$ and $G$ are compatible, they share the time, job and store dimensions.

Overlapping Compatible Fact Schemata (3)

- Schema resulting from overlapping $F$ and $G$ is $H$.

Overlapping Compatible Fact Schemata (4)

- In some cases, aggregation along a dimension can be carried out at different abstraction levels even if the corresponding dimension attributes were not explicitly shown.
  - Ex: a month attribute within a time hierarchy, fact instances can be aggregated by quarter, semester and year by performing a simple calculation.
  - Thus, given the $F$ and $G$ fact schemata, attribute quarter could in principle be added to the time dimension in the resulting schema $H$.

On the other hand, the designer must keep in mind that, by adopting this solution, the time for extracting data by quarter will increase significantly.

Thus, the best solution would probably be to add explicitly the quarter attribute to the time hierarchy in the employee fact schema.
3. Conceptual Design: From
Relational schema to Dimensional
Fact schema of Data Mart

- Finding and defining facts from Relational schema
- Building the Attribute Tree from Relational schema
- Building the Fact Schema from 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 (functional 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 (movieCode: title, category, director, length, 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)
3. Building 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)

Pruning

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 ...
Defining fact schema: DVD example (1)

Relational schema of the DVD rental BD:

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

Defining 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

Defining fact schema: DVD example (3)

Fact schema “RENTAL”:

Defining fact schema: DVD example (4)

SQL measure glossaries for fact schema “RENTAL”:

\[
\text{number} = \text{SELECT COUNT (*)}
\]

\[
\text{FROM RENTALS R INNER JOIN COPIES C}
\]

\[
\text{ON R.positionOnShelf = C.positionOnShelf,}
\]

\[
\text{COPIES C INNER JOIN MOVIES F}
\]

\[
\text{RENTALS R INNER JOIN CUSTOMERS C}
\]

\[
\text{ON R.cardNumber = C.cardNumber}
\]

\[
\text{GROUP BY F.title, R.date, C.name;}
\]
Defining fact schema: Flight example (1)

Relational logical schema describes an operational DB for Flights:

- 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”

Defining fact schema: FLIGHT example (2)

Pruning and grafting the attribute tree:

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

Defining fact schema: FLIGHT example (3)

Final attribute tree

Defining fact schema: FLIGHT example (4)

Fact schema “TICKET ISSUE”:
Defining fact schema: Flight example (5)

SQL measure glossaries for fact schema "TICKET_ISSUE":

\[
\text{numberOfFlight} = \text{SELECT COUNT (*) FROM TICKETS T INNER JOIN FLIGHT_INSTANCES I ON T.flightNumber = I.flightNumber AND T.date = I.date GROUP BY T.passengerNumber, I.date, T.flightNumber;}
\]

\[
\text{numberOfBags} = \text{SELECT SUM (C.numberOfBag) FROM TICKETS T INNER JOIN FLIGHT_INSTANCES I ON T.flightNumber = I.flightNumber AND T.date = I.date JOIN TICKETS T INNER JOIN CHECK_IN C ON T.ticketNumber = C.ticketNumber GROUP BY T.ticketNumber, I.date, T.flightNumber;}
\]

\[
\text{receipts} = \text{SELECT SUM (T.fare) FROM TICKETS T INNER JOIN FLIGHT_INSTANCES I ON T.flightNumber = I.flightNumber AND T.date = I.date GROUP BY T.passengerGender, I.date, T.flightNumber;}
\]

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