Methodological Framework

Conceptual Modelling: the Dimensionnal 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 Dimensionnal 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**

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

**Various strategies**

- **Top-Down Approach**:
  1. Design of DW
  2. Design of DMs

- **Bottom-Up Approach**:
  1. Design of DMs
  2. Integration of DMs in DW
  3. Maybe no physical DW

- **Mixed Approach**:
  1. Design of DW for DM1
  2. Design of DM2 and integration with DW
  3. Design of DM3 and integration with DW
  4. ...

**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

**(-)** Weaknesses:
- 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.
Bernard ESPINASSE - Data Warehouse Conceptual modeling and Design

**Life-Cycle with a Top-Down strategy**

- **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**
  - 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

**Bottom-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 ...

**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 of interest
- 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

**Main Data Warehouse/Mart Design Steps**

Each Data Mart (DM) will be designed according these steps:
### Conceptual Design of a Data Mart (DM)

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

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

### The Dimensional Fact Model (DFM)

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.
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** contain 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 dimensional 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
- Store -> storeCity -> state -> country

Bernard ESPINASSE - Data Warehouse Conceptual modeling and Design
**Advanced Modeling: Shared Hierarchies**

*Shared hierarchies* exist when entire portion of hierarchies are frequently replicated 2 or more time in fact schemata. In particular in time hierarchies, 2 or more date-type dimensions with different meaning can easily exist in 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>Non-temporal 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 than 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.

Overlapping Compatible Fact Schemata (2)

Consider the 2 fact schemata:

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

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:

H can be used, for instance, to calculate the percentage of non-European employees for each city, job and year.

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.
Consider the measure each week. expresses, for each product, the number of copies present within each warehouse during aggregated by using operators such as average, maximum, minimum; Figure 5 shows an.

The step to derive DF schemata from Relational schema is:

1. Finding and defining facts from Relational schema
   - 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 follows:

- Each node of the attribute tree corresponds to one or more Relational schema attributes

- The root of the attribute tree corresponds to the primary key of F

- For each node v, the corresponding attribute functionally determines all the attributes that correspond to the descendants of v (functionnal dependencies)

### 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, fate, 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)

**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:
Attribute Tree 3 (TICKETS):

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

3. Building the Fact Schema

For each fact:

- 3.1. Pruning and grafting the attribute tree:
  - 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

- 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).

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 (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
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

Fact schema “RENTAL”:

- number
- gender
- title
- category
- length
- director
- mainActor
- date
- fact
- dimensions

SQL measure glossaries for fact schema “RENTAL”:

```sql
number = SELECT COUNT (*)
FROM RENTALS R INNER JOIN COPIES C
ON R.positionOnShelf = C.positionOnShelf,
COPIES C INNER JOIN MOVIES F
RENTALS R INNER JOIN CUSTOMERS C
ON R.cardNumber = C.cardNumber
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)

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

Defining fact schema: FLIGHT example (3)

Final attribute tree

Derived fact Schema

Defining fact schema: FLIGHT example (4)

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

SQL measure glossaries for fact schema "TICKET ISSUE":

```sql
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;
```

```sql
SELECT SUM(C.numberofBag)
FROM TICKETS T INNER JOIN FLIGHT_INSTANCES I ON T.flightNumber = I.flightNumber AND T.date = I.date
GROUP BY T.ticketNumber, I.flightNumber, T.tickets;
```

`numberofbags` = `SELECT SUM(C.numberofBag)`

```sql
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.tickets;
```

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

---

Rent a car example (2)

Choosing either RENTALS or PAYMENTS as fact is the same here, because these 2 tables are related by a one-to-one link.

---

Rent a car example (3)

In the edited attribute tree, the drop-off date is pruned and replaced by a Duration attribute computed as the number of days between the drop-off and the pick-up dates.
Rent a car example (4)

« RENTAL » Fact schema:

- Area
- Country
- State
- City
- Office
- dropoff
- pickup
- Car
- Fuel
- Brand
- Model
- Category
- registration
- Date
- Month
- Year
- Amount
- Discount
- Duration
- Miles
- PaymentMode