DECISION SUPPORT SYSTEMS

DataWareHousing, BuSiNeSS InteLLigeNce aNd aLL tHat.

We have been talking mostly about operational systems; let's talk about another important aspect of information systems, decision support systems that are supposed to provide "strategical" and not only "tactical" data.

The topic has looooong been in the air. Perhaps that today you would get snappy charts instead of a multipage report, but the idea is the same.

Dynamic information…
for better management

Within your organization, the need for prompt, accurate management decisions crops up at all times, in all places. Each delayed decision cuts into your overall efficiency and productivity. With System/34, you can respond to situations as they happen.

One line of information—or several—can be called up at any display station, or you can request a multipage report to be printed on demand.

IBM System 34 Brochure, 1977
www.computerhistory.org

On Line Transaction Processing

Traditionally, systems that record orders, production, billing and so forth are called OLTP systems. They are characterized by short transactions that are executed a large number of times.

Getting information out of OLTP databases isn't easy: you have tons of tables, queries are complicated, answering an unusual question is difficult.

> “During the turmoil many banks had to carry out big fact-finding missions to see where they stood. “Answering such questions as ‘What is my exposure to this counterparty?' should take minutes. But it often took hours, if not days,” says Peyman Mestchian, managing partner at Chartis Research, an advisory firm. Insiders at Lehman Brothers say its European arm lacked an integrated picture of its risk position in the days running up to its demise.”

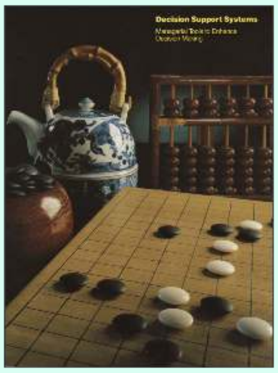
`http://www.economist.com/node/15016132`

Operational issue

what about the long term?

If OLTP systems are perfect for daily management, directors need to know where they should take the business: what is on a downwards trend, which part is picking up, where to invest, and where should the company be in 5 years time. They like to think of themselves as "visionary", they aren't always, but when they aren't having some synthetic ideas about the present state helps (assuming nothing disruptive happens)

DECISION SUPPORT SYSTEMS



Once again, it has long been a promise of IT. Some marketing leaflets that are 40 years old could have been written yesterday.

http://www.computerhistory.org/

1977 Brochure (Tymshare)

Here the needs of the information system are different. To see long-term trends we need long term data. We don't need every detail of what your latest shopping cart contained. Additional data (how is the competition doing?) helps.

Longer Term

Different Grain of Information

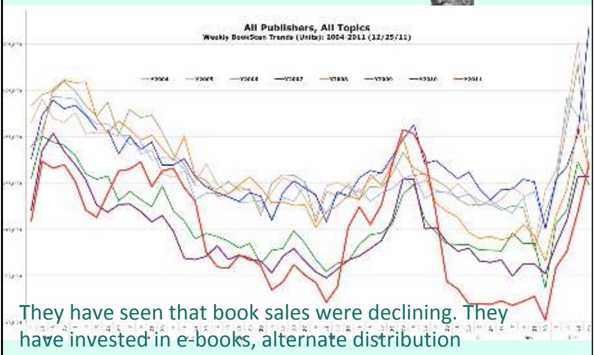
Additional Data

Example

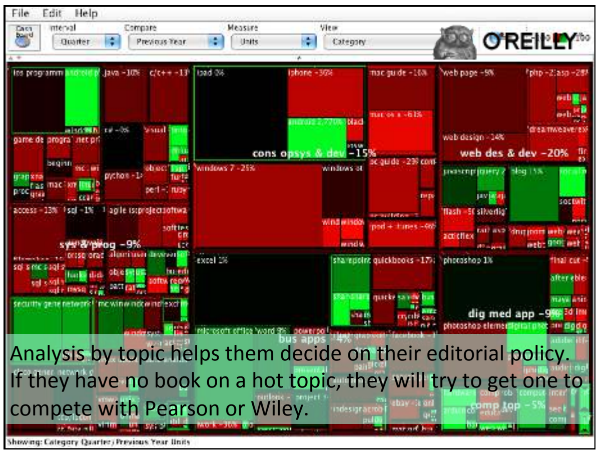
O'REILLY®

nielsen

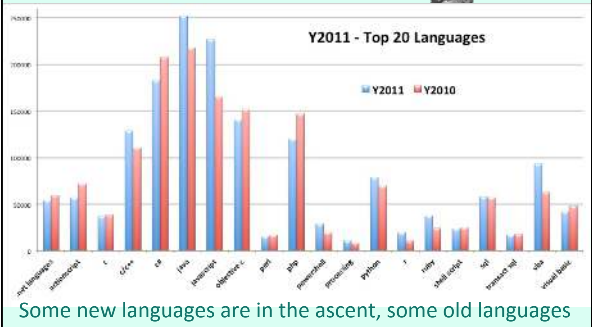
A good illustration is what O'Reilly, the computer book publisher, has been doing for quite a while with data bought from Nielsen, a market research company that compiles book sales (the information here is a few years old, O'Reilly no longer publish it ...)



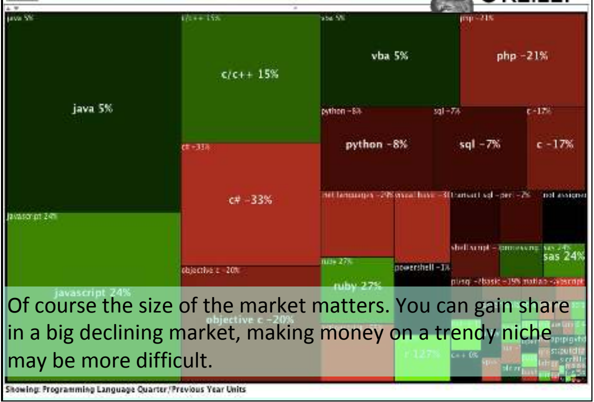
They have seen that book sales were declining. They have invested in e-books, alternate distribution channels, organization of Tech conferences.



Analysis by topic helps them decide on their editorial policy. If they have no book on a hot topic, they will try to get one to compete with Pearson or Wiley.



Some new languages are in the ascent, some old languages are holding very well. They may not be much interested in your COBOL book project.



Of course the size of the market matters. You can gain share in a big declining market, making money on a trendy niche may be more difficult.

Never forget that assuming that the future will follow exactly in the steps of the present and the past is always dangerous.

> **Mutual Funds, Past Performance**
>
> This year's top-performing mutual funds aren't necessarily going to be next year's best performers. It’s not uncommon for a fund to have better-than-average performance one year and mediocre or below-average performance the following year. That's why the SEC requires funds to tell investors that a fund's past performance does not necessarily predict future results.

`http://www.sec.gov/answers/mperf.htm`

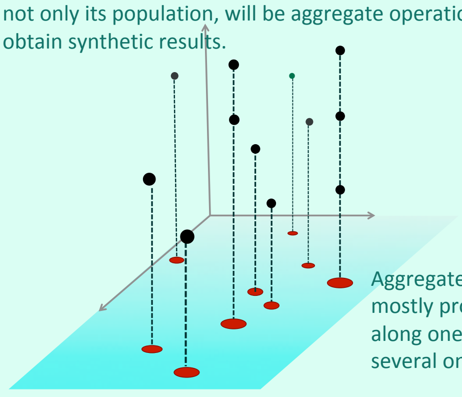
Having longer term data (historical data) and additional data means that we'll have to deal with possibly far more data than operational databases.

Longer Term
Additional Data } BIG

What will be big may not be so much what we STORE, if we aggregate data, but what we PROCESS, because aggregation is a costly operation.

Different Grain of Information
Aggregate

And indeed most operations against a data warehouse, and not only its population, will be aggregate operations to obtain synthetic results.



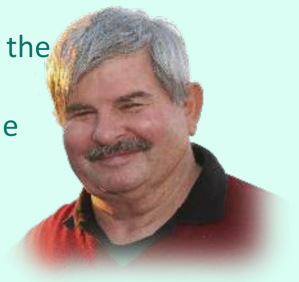
Aggregates are mostly projections along one axis (or several ones)

Aggregates, unfortunately, require by definition processing huge volumes of data, sorts, and computations all along. All operations that require I/Os and CPU. The workload is completely different from an operational database workload and this is why data warehousing and business intelligence are considered a topic apart in information systems.

Aggregating takes time

THE PRINCIPLES

Bill Inmon coined the expression "Data Warehouse" in the early 1990s


Bill Inmon

What Inmon had noticed, which has indeed been a recurrent phenomenon in IT since the introduction of personal computers, is that because databases are huge, schemas are complicated and queries difficult to write, a large number of people, especially in higher circles, were mostly working with figures extracted from various sources (including printed reports), and that a lot of strategic data was living outside "official IT"

Shadow IT

Official IT

This shadow IT mostly lives in spreadsheets, shared by email or (today) in a public cloud. Needless to say, data organization is poor, consistency weak, several versions circulate, and security is low.

data spaghetti

not a single version of the truth

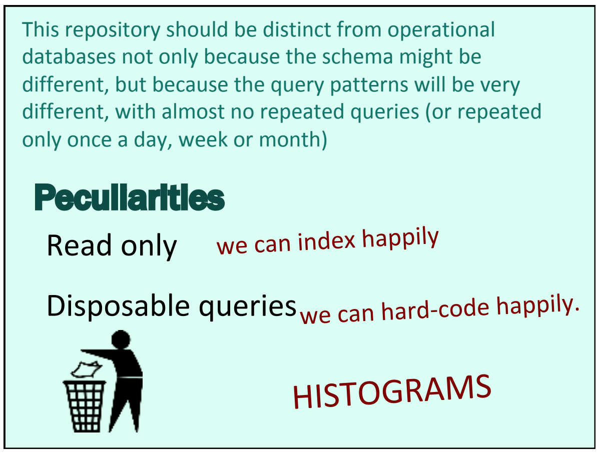
Inmon basically suggested creating a single repository that would act as a reference and could be managed with professional (IT) standards.

Inmon's definition

a subject oriented,	"themes"
nonvolatile,	Read only
integrated,	Clean
time variant	Historical data

collection of data in support of

management's decisions



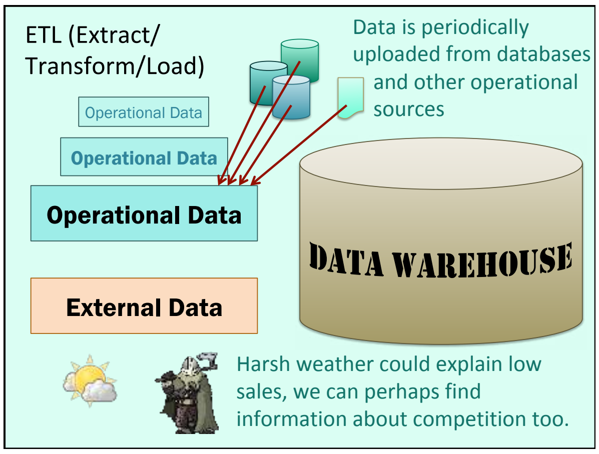
This repository should be distinct from operational databases not only because the schema might be different, but because the query patterns will be very different, with almost no repeated queries (or repeated only once a day, week or month)

Peculiarities

Read only we can index happily

Disposable queries we can hard-code happily.

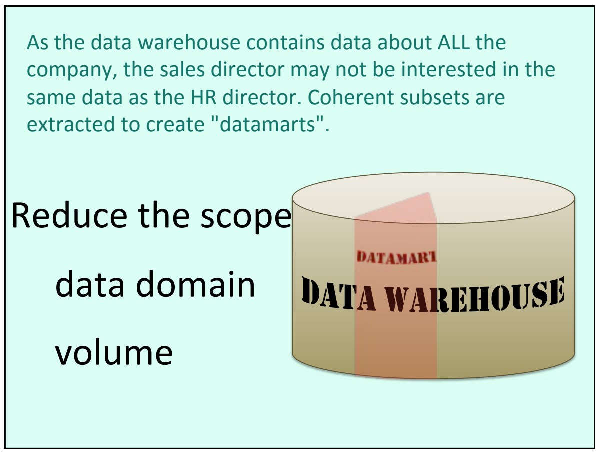
HISTOGRAMS



ETL (Extract/ Transform/Load)

Data is periodically uploaded from databases and other operational sources

Harsh weather could explain low sales, we can perhaps find information about competition too.



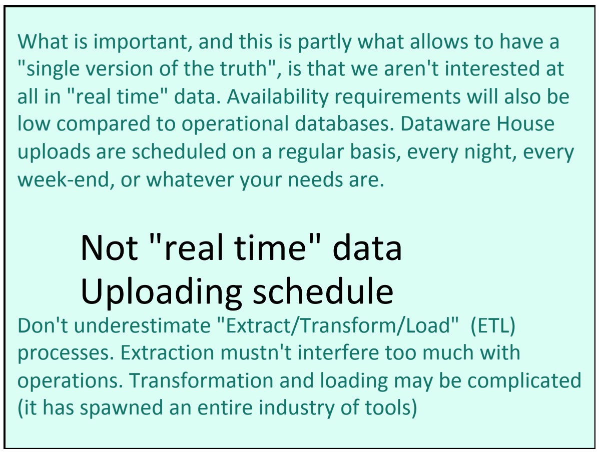
As the data warehouse contains data about ALL the company, the sales director may not be interested in the same data as the HR director. Coherent subsets are extracted to create "datamarts".

Reduce the scope

data domain

volume

DATAMART
DATA WAREHOUSE



What is important, and this is partly what allows to have a "single version of the truth", is that we aren't interested at all in "real time" data. Availability requirements will also be low compared to operational databases. Dataware House uploads are scheduled on a regular basis, every night, every week-end, or whatever your needs are.

Not "real time" data

Uploading schedule

Don't underestimate "Extract/Transform/Load" (ETL) processes. Extraction mustn't interfere too much with operations. Transformation and loading may be complicated (it has spawned an entire industry of tools)

The purpose

To outsmart competition

The promise, of course (and the justification to huge investments in hardware, software licences and consulting) is to give a company that definitive edge over its competitors.

Outsmart with "Best Practices"?

One may wonder about the competitive edge when all actors are building data warehouses with the same tools, but let's refrain from being party-poopers. Taking decisions from sound information isn't bad.

Flickr: Striatic

If SQL was initially designed as a language for non-specialists, this is the area where it has most obviously failed. You need tools (IBM Cognos and SAP Business Objects are among the better known - if you feel that the names look like acquisitions, you are right)

The problem

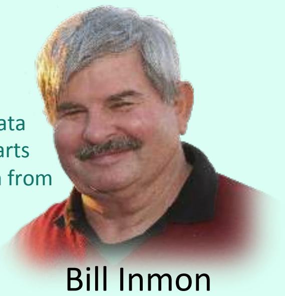
Generating efficient queries for people who cannot spell SQL

For Inmon, a company should start by defining precisely a data warehouse, as a properly normalized database. The problem is that while the database design is going on, you can spend months before the management sees results.

TOP -DOWN

An enterprise has one data warehouse, and data marts source their information from the data warehouse.

3NF



Bill Inmon

Which is why Inmon's ideas were very successfully challenged by Ralph Kimball, who suggested the opposite approach: starting with quickly designed, denormalized datamarts in departments.

Ralph Kimball

BOTTOM-UP

Departmental datamarts, the data warehouse is the enterprise-wide collection of datamarts.

2NF

Dimensions

Kimball boldly suggested to shake the dogma of proper database design for decision support systems.

Every non key attribute must provide a fact about the key, 1NF
the whole key, 2NF
and nothing but the key. 3NF

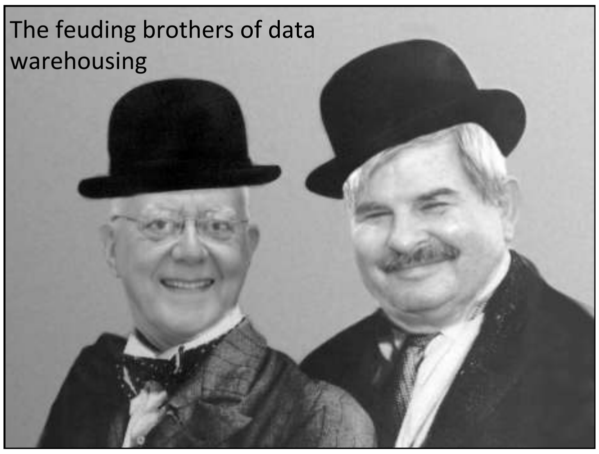
William Kent (1936 – 2005)

2NF Some columns store data derived from other columns

Of course, Kimball knows perfectly what proper normalization means and doesn't discuss the fact that operational systems must be properly normalized. But he points out that the rule that there should be no repeated data in different rows stems from the desire to avoid anomalies when data is updated in some rows and not the others, and that this concern is irrelevant in a read-only database. By contrast, repeating data makes querying easier by removing the need for joins or functions.

To illustrate Kimball's vision, this is what the MOVIES table could look like in a kimballesque database. The English title avoids a join with ALT_TITLES, country name and continent avoid a join with COUNTRIES, storing the decade avoids deriving it from the year.

```
movieid         movieid
title           title
country         english_title
year_release    country
                country_name
                continent
                year_release
                decade
```


The feuding brothers of data warehousing

THE CONCEPTS

Mostly
Kimball's

To support his ideas, Kimball has introduced a number of new concepts which have become very popular and that you are almost certain to encounter sooner or later.

1

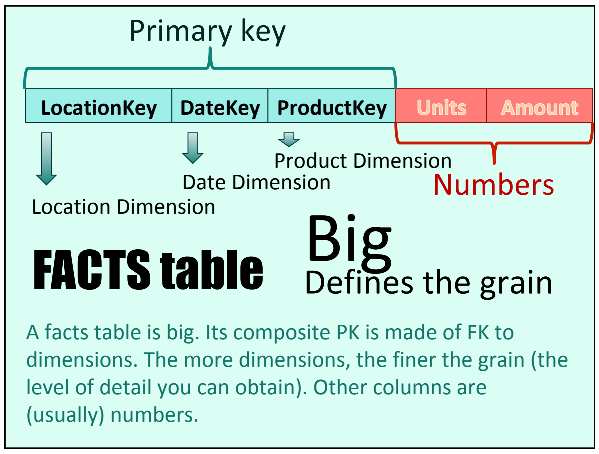
Facts and dimensions

First of all, facts and dimensions. Facts, to make it simple, are principally figures. Dimensions are the axes along which these figures can be aggregated (as we are in a world where we want synthesis).
Let's give an example.

I'd like to see sales in USD per country, month and per line of product.

This is a typical managerial request. Sales are facts. Everything that follows a "per" is a dimension. Simple.

Flickr: Steve Wilson

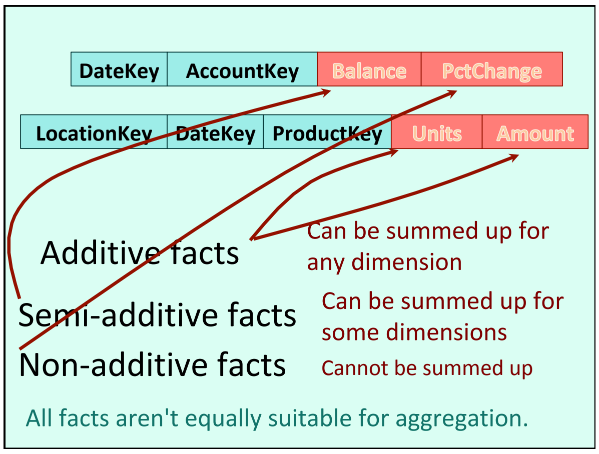


FACTS table

Big

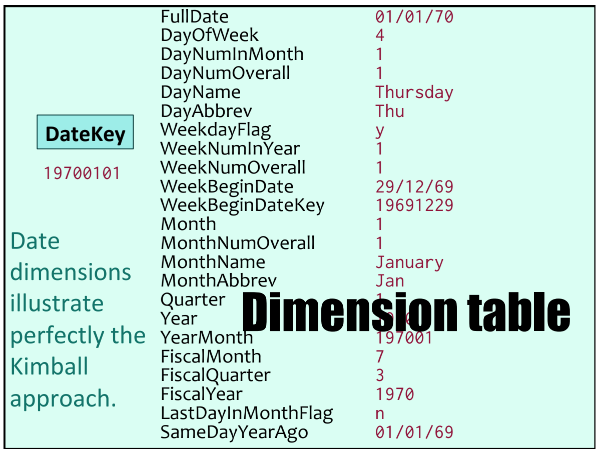
Defines the grain

A facts table is big. Its composite PK is made of FK to dimensions. The more dimensions, the finer the grain (the level of detail you can obtain). Other columns are (usually) numbers.



Additive facts	Can be summed up for any dimension
Semi-additive facts	Can be summed up for some dimensions
Non-additive facts	Cannot be summed up

All facts aren't equally suitable for aggregation.



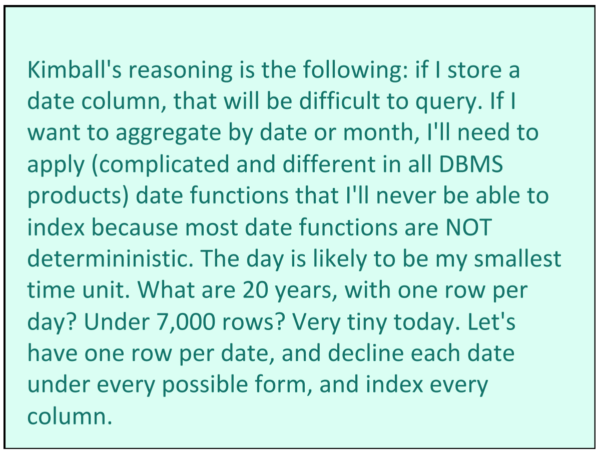
DateKey

19700101

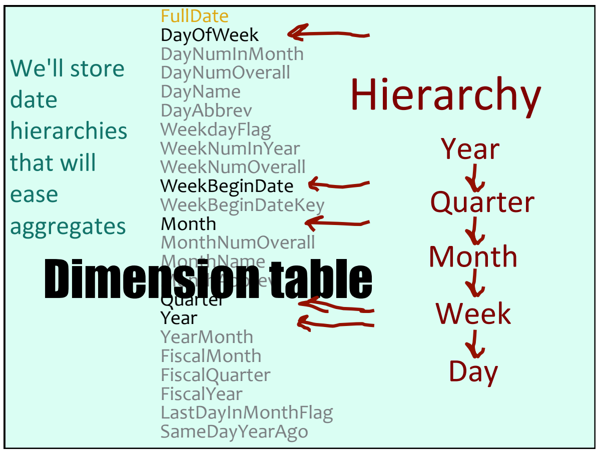
FullDate	01/01/70
DayOfWeek	4
DayNumInMonth	1
DayNumOverall	1
DayName	Thursday
DayAbbrev	Thu
WeekdayFlag	y
WeekNumInYear	1
WeekNumOverall	1
WeekBeginDate	29/12/69
WeekBeginDateKey	19691229
Month	1
MonthNumOverall	1
MonthName	January
MonthAbbrev	Jan
Quarter	1
Year	1970
YearMonth	197001
FiscalMonth	7
FiscalQuarter	3
FiscalYear	1970
LastDayInMonthFlag	n
SameDayYearAgo	01/01/69

Dimension table

Date dimensions illustrate perfectly the Kimball approach.



Kimball's reasoning is the following: if I store a date column, that will be difficult to query. If I want to aggregate by date or month, I'll need to apply (complicated and different in all DBMS products) date functions that I'll never be able to index because most date functions are NOT determininistic. The day is likely to be my smallest time unit. What are 20 years, with one row per day? Under 7,000 rows? Very tiny today. Let's have one row per date, and decline each date under every possible form, and index every column.



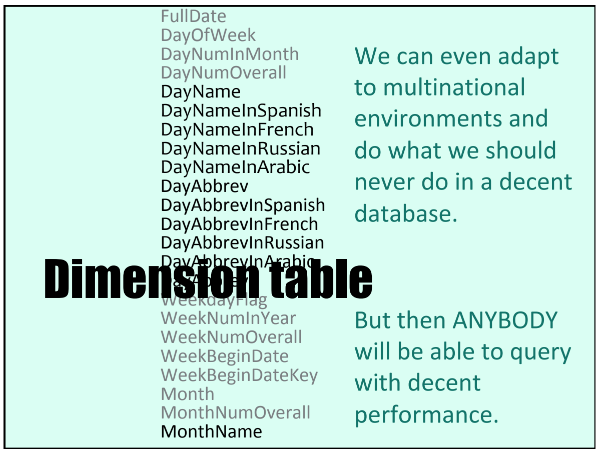
Dimension table

We'll store date hierarchies that will ease aggregates

FullDate
DayOfWeek
DayNumInMonth
DayNumOverall
DayName
DayAbbrev
WeekdayFlag
WeekNumInYear
WeekNumOverall
WeekBeginDate
WeekBeginDateKey
Month
MonthNumOverall
MonthName
MonthAbbrev
Quarter
Year
YearMonth
FiscalMonth
FiscalQuarter
FiscalYear
LastDayInMonthFlag
SameDayYearAgo

Hierarchy

$$\begin{tikzcd}[row sep=small] \text{Year} \arrow[d] \\ \text{Quarter} \arrow[d] \\ \text{Month} \arrow[d] \\ \text{Week} \arrow[d] \\ \text{Day} \end{tikzcd}$$

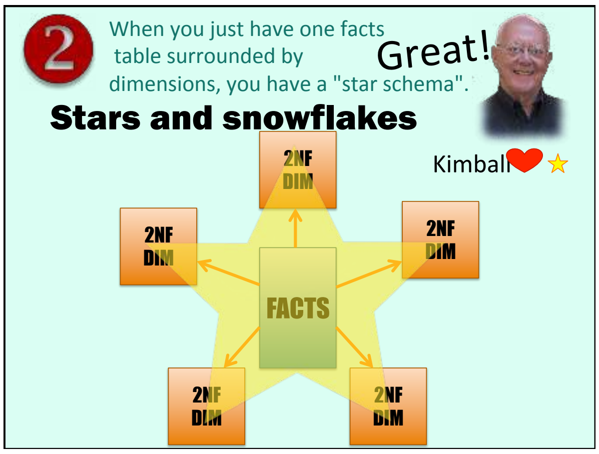


Dimension table

FullDate
DayOfWeek
DayNumInMonth
DayNumOverall
DayName
DayNameInSpanish
DayNameInFrench
DayNameInRussian
DayNameInArabic
DayAbbrev
DayAbbrevInSpanish
DayAbbrevInFrench
DayAbbrevInRussian
DayAbbrevInArabic
DayAbbrev
WeekdayFlag
WeekNumInYear
WeekNumOverall
WeekBeginDate
WeekBeginDateKey
Month
MonthNumOverall
MonthName

We can even adapt to multinational environments and do what we should never do in a decent database.

But then ANYBODY will be able to query with decent performance.



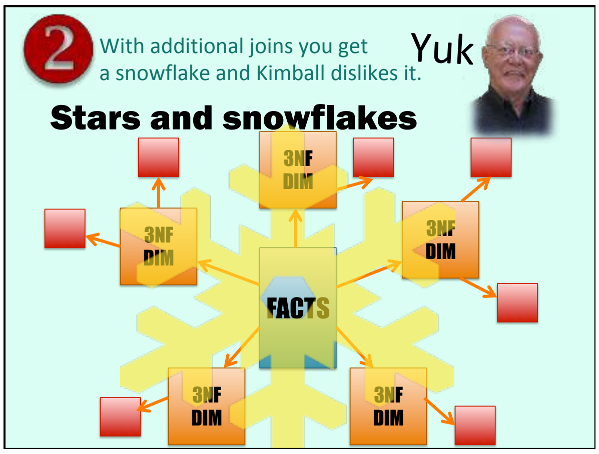
Stars and snowflakes

2

When you just have one facts table surrounded by dimensions, you have a "star schema".

Great!

Kimball

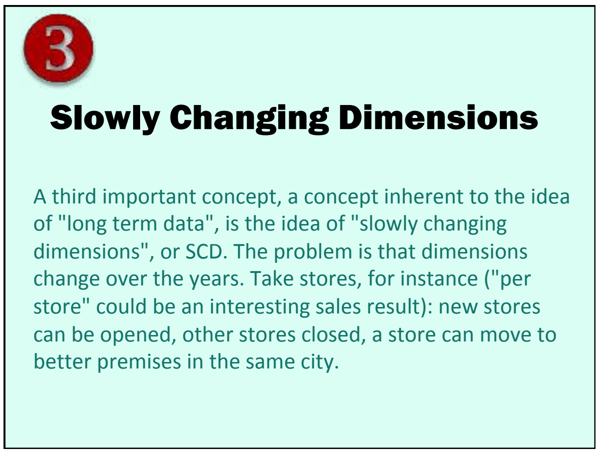


Stars and snowflakes

2

With additional joins you get a snowflake and Kimball dislikes it.

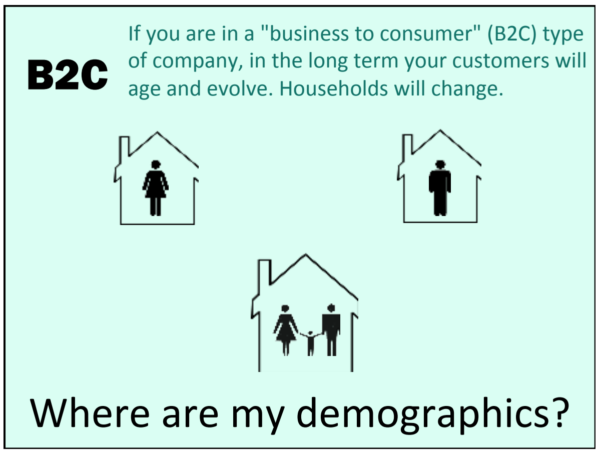
Yuk



3

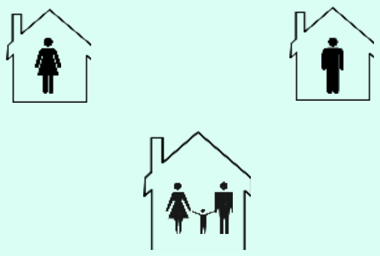
Slowly Changing Dimensions

A third important concept, a concept inherent to the idea of "long term data", is the idea of "slowly changing dimensions", or SCD. The problem is that dimensions change over the years. Take stores, for instance ("per store" could be an interesting sales result): new stores can be opened, other stores closed, a store can move to better premises in the same city.

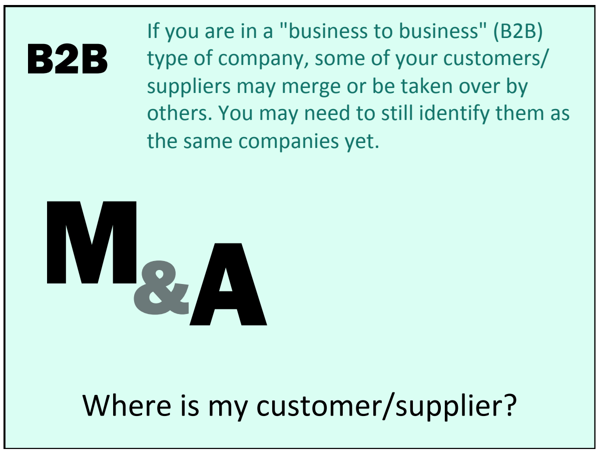


B2C

If you are in a "business to consumer" (B2C) type of company, in the long term your customers will age and evolve. Households will change.



Where are my demographics?

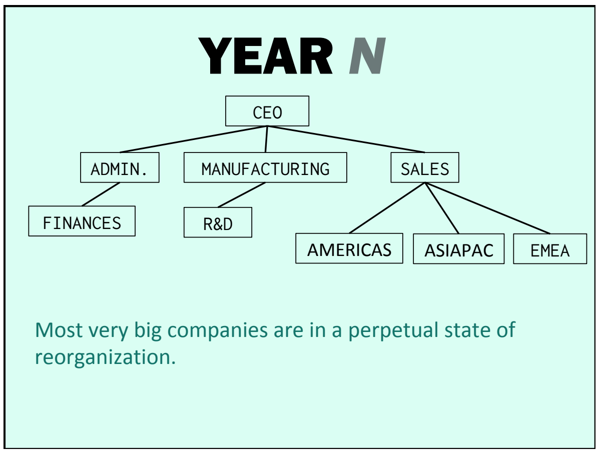


B2B

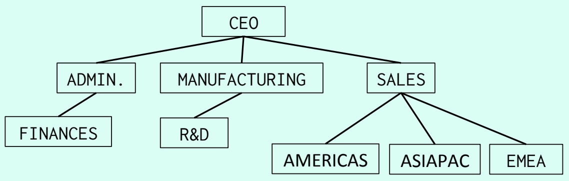
If you are in a "business to business" (B2B) type of company, some of your customers/ suppliers may merge or be taken over by others. You may need to still identify them as the same companies yet.

M&A

Where is my customer/supplier?

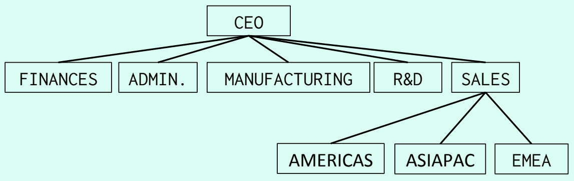


YEAR *N*



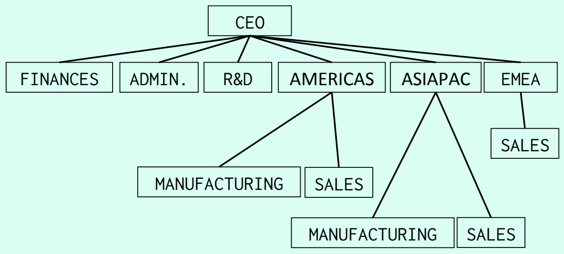
Most very big companies are in a perpetual state of reorganization.

YEAR *N+1*



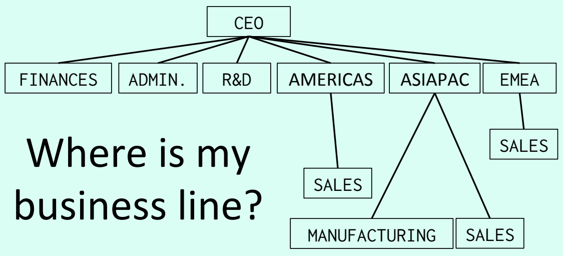
It may be linked to the recognition of the importance of one particular department, or the development of new markets.

YEAR *N+3*



It may be linked to changing business conditions, to new management fashions.

YEAR *N+5*



Where is my business line?

A part of the business may be sold off too. Finally, sometimes it's linked to the arrival of new managers or personal rivalries.

Kimball has tried to classify how to handle slowly changing dimensions. One way to handle them is to ignore them (iPhone 25 same product as iPhone 1)

Type 0 SCD

Keep as it was first inserted

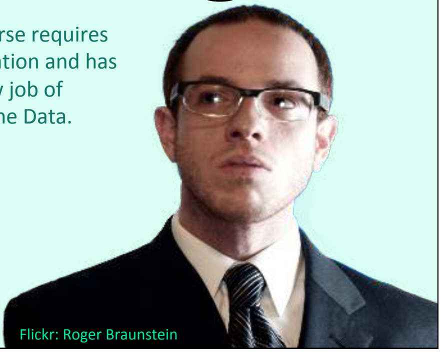
Or you can update, and consider that there is Russia and there never was such thing as Soviet Union. Or consider that there is BDR and DDR, and there is Germany. Or that BDR and DDR became Germany in 1990.

Type 1 SCD	Update dimension - lose old record
Type 2 SCD	Add a new record –- Disconnect old facts/new facts
Type 3 SCD	Add old/new/date information to the dimension

In practice most SCDs are type 1 or 2

Data Manager

All this of course requires due consideration and has created a new job of Guardian of the Data.



THE PLUMBING

Now, we have seen that in all likelihood most queries on a data warehouse will be massive aggregates, that take time. Worst, they will be run by people high up in the hierarchy, whose time is precious (especially when you consider their salaries) and who, having signed the checks and knowing how much all this did cost, expect bang for their bucks.

How can we have blazingly fast aggregates? With a traditional DBMS (there are other options, remember the columnar database presented by Guy Harrison) there are basically two solutions

Speeding up aggregates

Pre-computing aggregates

NOT mutually exclusive ...

When we want a query that says, for instance, "what was the sum of sales per state and month last year", from an SQL standpoint getting good performance is tough. We'll return at most 50x12 rows, which is peanuts. We'll scan a small table of states, and a small table of dates. Neither "state" nor "month" will be very selective.

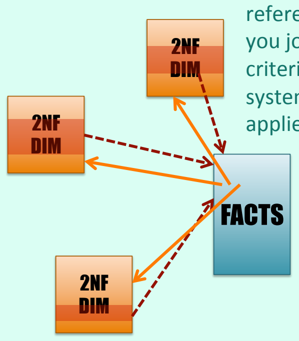
Because of 2NF, cardinality is often low

The only restriction "last year", is rather weak

Only broad criteria

And we may aggregate MANY rows.

Dimensions to facts



Very often, you go from referencing table to referenced table when you join. Here, all criteria will systematically be applied to reference tables that DON'T point to the facts table (it's the opposite)

For low cardinality columns, advanced products such as Oracle implements "bitmap indexes". Their entries are chunk of bits that cover many rows and contain a bit that says whether the row contains the key or not.

Bitmap indexes

MON	1000000100000001000000010000000100000000
TUE	0100000010000000100000001000000010000000
WED	0010000000100000001000000010000000100000
THU	0001000000010000000100000001000000010000
FRI	0000100000000100000000100000000100000000
SAT	0000010000000010000000010000000010000000
SUN	0000001000000001000000001000000001000000

The advantage of bitmap indexes is that by applying AND and OR operations to the bitmaps you can quickly identify the rows that satisfy several conditions (Shenzhen and November) and fetch them.

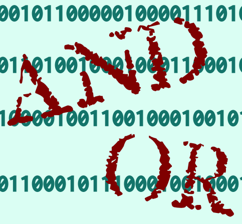
Bitmap indexes

```
111100101100000010000111010001000111

010001[illegible]01001[illegible]00[illegible]011[illegible]0010011110000

00111[illegible]0010011001000100101100010010

10010110001011[illegible]000[illegible]00[illegible]0001110000100
```



Facts

Dimensions

Star transformation ORACLE

```
SELECT c.cust_city, t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id
  AND s.cust_id = c.cust_id
  AND s.channel_id = ch.channel_id
  AND c.cust_state_province = 'CA'
  AND ch.channel_desc = 'Internet'
  AND t.calendar_quarter_desc IN ('1999-01','1999-02')
GROUP BY c.cust_city, t.calendar_quarter_desc
```

Oracle also implements a parameter that enables the "star schema transformation"

This transformation is just adding redundant conditions that make the choice of the suitable plan irresistible to the optimizer.

Star transformation ORACLE

```
SELECT c.cust_city, t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id
  AND s.cust_id = c.cust_id
  AND s.channel_id = ch.channel_id
  AND c.cust_state_province = 'CA'
  AND s.cust_id IN (SELECT cust_id
                    FROM customers
                    WHERE cust_state_province='CA')
  AND ch.channel_desc = 'Internet'
  AND t.calendar_quarter_desc IN ('1999-01','1999-02')
GROUP BY c.cust_city, t.calendar_quarter_desc
```

Star transformation ORACLE

These redundant conditions are added for every dimension in the query

Similar idea, hand-crafted

```
SELECT s.rowid
FROM sales s, customers c
WHERE s.cust_id = c.cust_id
  AND c.cust_state_province = 'CA'
INTERSECT
SELECT s.rowid
FROM sales s, times t
WHERE s.time_id = t.time_id
  AND t.calendar_quarter_desc IN ('1999-01','1999-02')
INTERSECT
SELECT s.rowid
FROM sales s, channels ch
WHERE s.channel_id = ch.channel_id
  AND ch.channel_desc = 'Internet'
```

We only need to hit the index on sales(cust_id) (it contains cust_id and rowid)

"rowid" is Oracle-speak for "row address". You can try to identify the rows you need first.

We are assuming one separate index per FK

Similar idea, hand-crafted

Join to sales, and aggregate

Once you know exactly which rows you want, you can prey on them in the big table.

Database Machines/Appliances

TERADATA

NETEZZA an IBM Company

Greenplum

Oracle ExadataX3

and others ...

Several companies have created dedicated hardware that is more specially geared towards the data warehouse type of queries. Some databases, called columnar databases, store data by columns rather than rows to speed up aggregates.

WELCOME TO ANOTHER ROUND OF "IF WE HAD THE MONEY". I'LL GO FIRST.
DILBERT
by SCOTT ADAMS

Do I need to say it doesn't come cheap?

Precomputing aggregates

Another, simple way to speed up aggregates is NOT to compute them on the fly but, if we have an idea about the aggregates that will be computed most often, to precompute them every time we upload the database, at least partially.

"Materialized views", which are actually tables, are saved query results (very often aggregates) that are prescheduled and automatically refreshed. You can query the materialized view directly, but some optimizers are smart enough to recognize when they can use a pre-computed result in a query on the original table. It's OK on data that isn't "real time data".

Materialized Views

NOT views

View = saved query text

Materialized view = saved query result

You specify in the creation statement the refresh schedule. It can be a full cancel and replace or, sometimes, an incremental refresh that requires a trigger-populated log on the original table(s).

Materialized Views

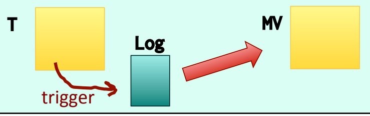
Of course, it's all plain replicated data.

Refresh schedule

Full refresh

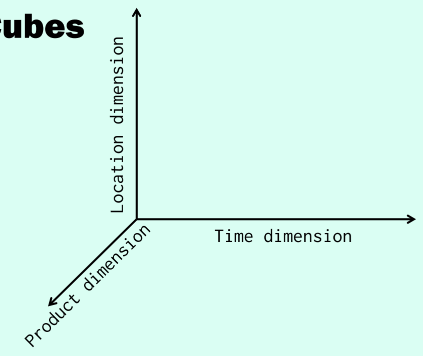
```
insert into ...
select ...
```

Fast refresh



Some products go further and try to compute every possible aggregate possible, detail AND all possible sums.

Cubes



If you don't ask for anything that strays from the predefined path, you get immediate results.

Cubes

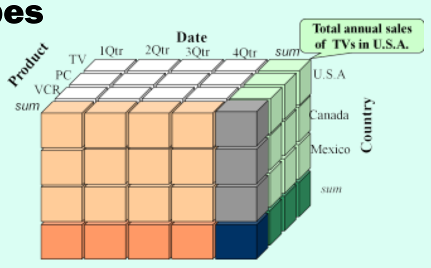


Image found on slidewiki.org

Cubes

Things become funny when you have MANY dimensions – the number of combinations explodes ...