Business intelligence

conjuntos de datos a un determinado objetivo, son integrados, son volatiles, cambian en el tiempo, orientados a dar apoyo a la toma de decisiones

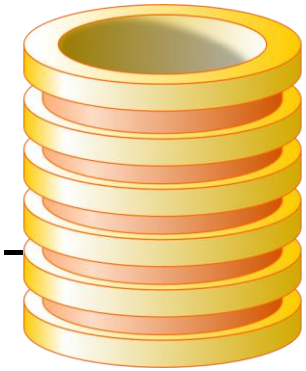
Unit 2 – Datawarehouse and OLAP
S2-1 – Datawarehouse

Introduction to datawarehouse

- **Objective:** to provide infrastructure that support **DSS (Decision Support System)** within an organization
  - Organizations begin **with basic** systems to manage **day-to-day operations**.
  - Powered by multiple **operational databases** (e.g. sales, inventory).
- **Requirements for DSS**
  - New requirements for **complex queries** and **data integration** of historical and operational data to generate knowledge.
  - Purposes:

Analysis of the organization
Make predictions
Define strategies

Introduction to datawarehouse



- You can still use the traditional system:
  - Maintaining **daily transactional work** in the original information systems (known as **OLTP**, On-Line Transactional Processing).
  - **Basic data analysis** is done in real time on the same database (known as **OLAP**, On-Line Analytical Processing).

las operacionales no estan optimizadas para consultas analiticas

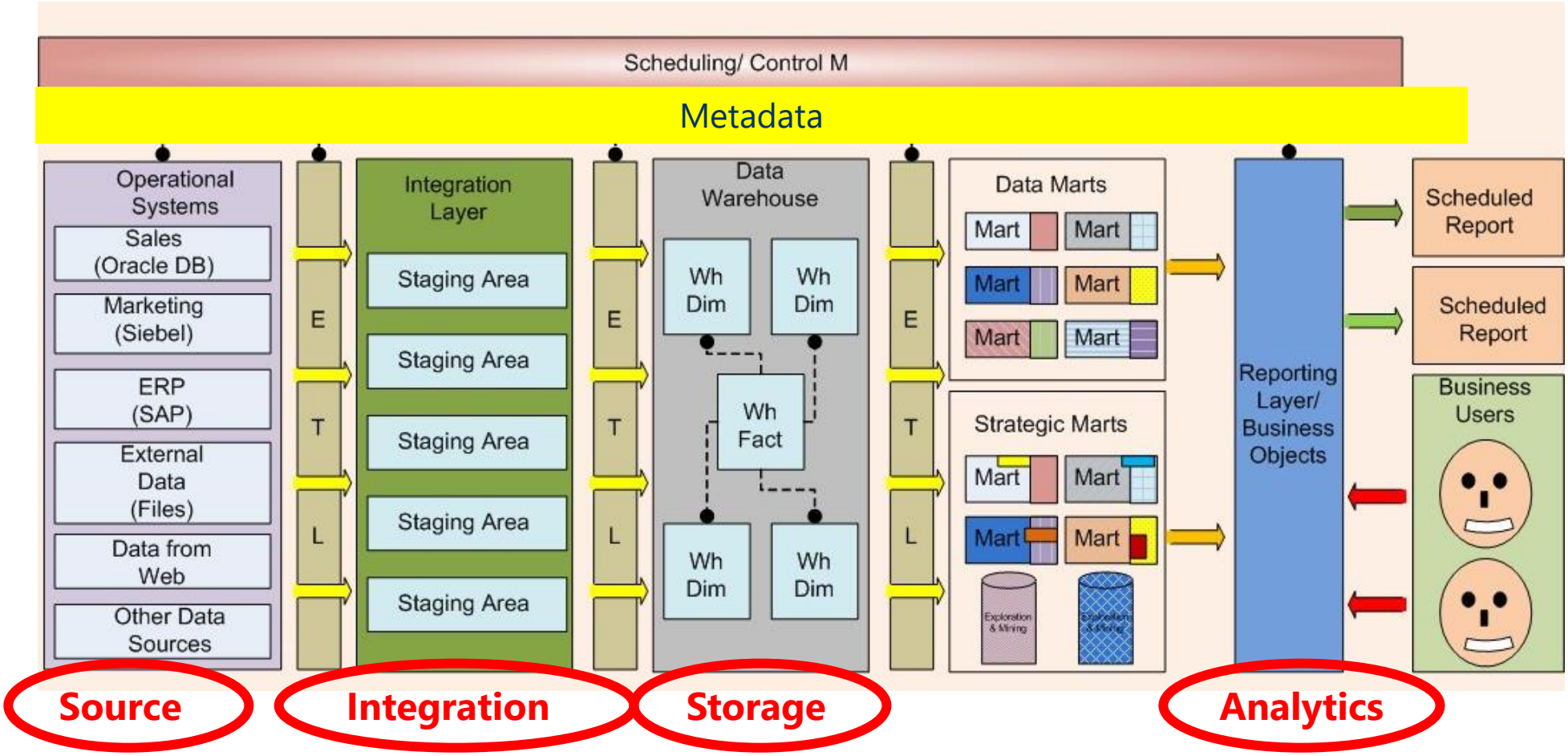
para garantizar la integridad de los datos, hay que tomar la decisiones acerca de las transaccion (lo que hablamos en BDGE)

- However, there are **limitations**:
  - Efficiency problems in daily work due to **complex queries** that are made when there is low load.
  - **No specific design** for analytical workloads.
  - **Performance bottlenecks** for in-depth analysis on operational data.

cuellos de botella

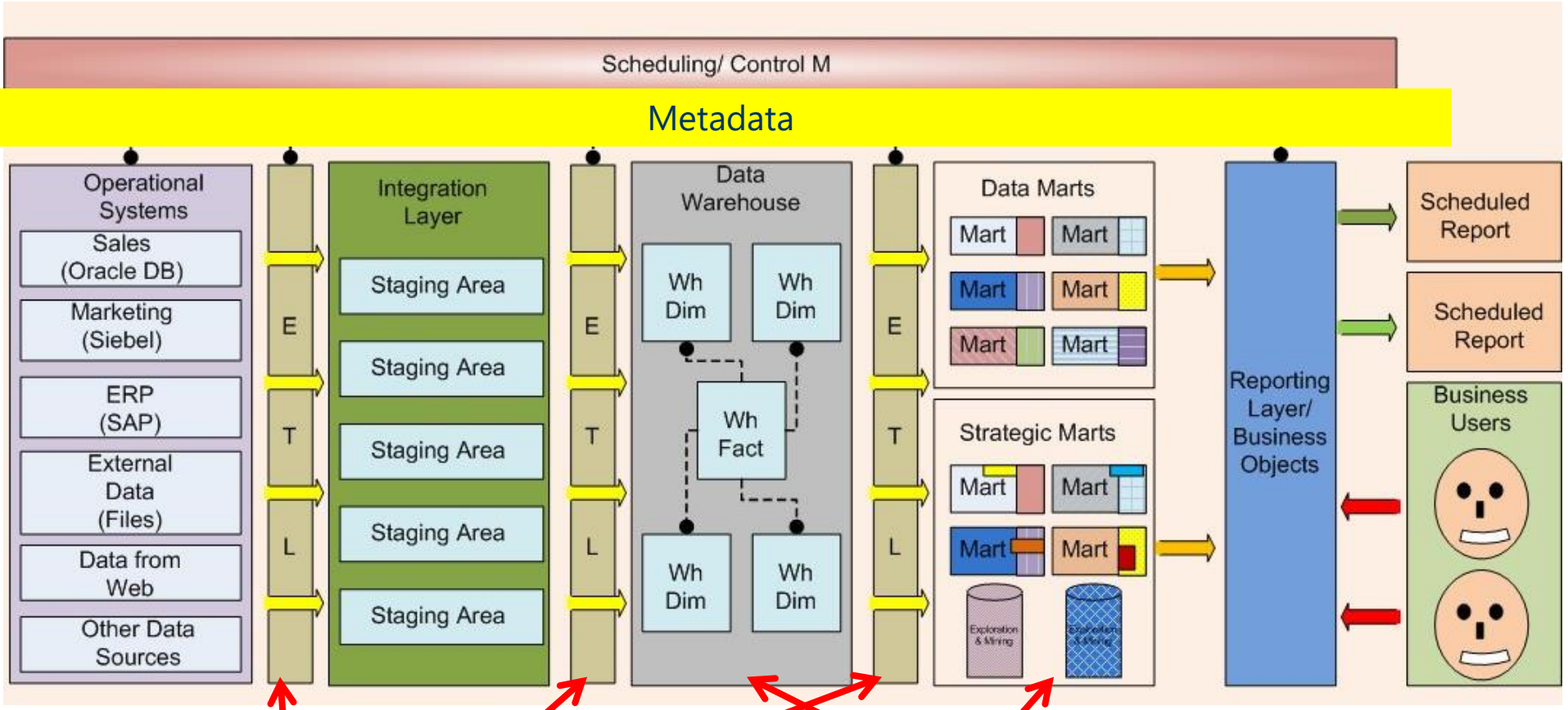
Conceptual Architecture and Components

en un entorno de analisis los requisitos van cambiando !!



el almacen de datos alimenta los data marts

Conceptual Architecture and Components



mas componentes

DWH Components

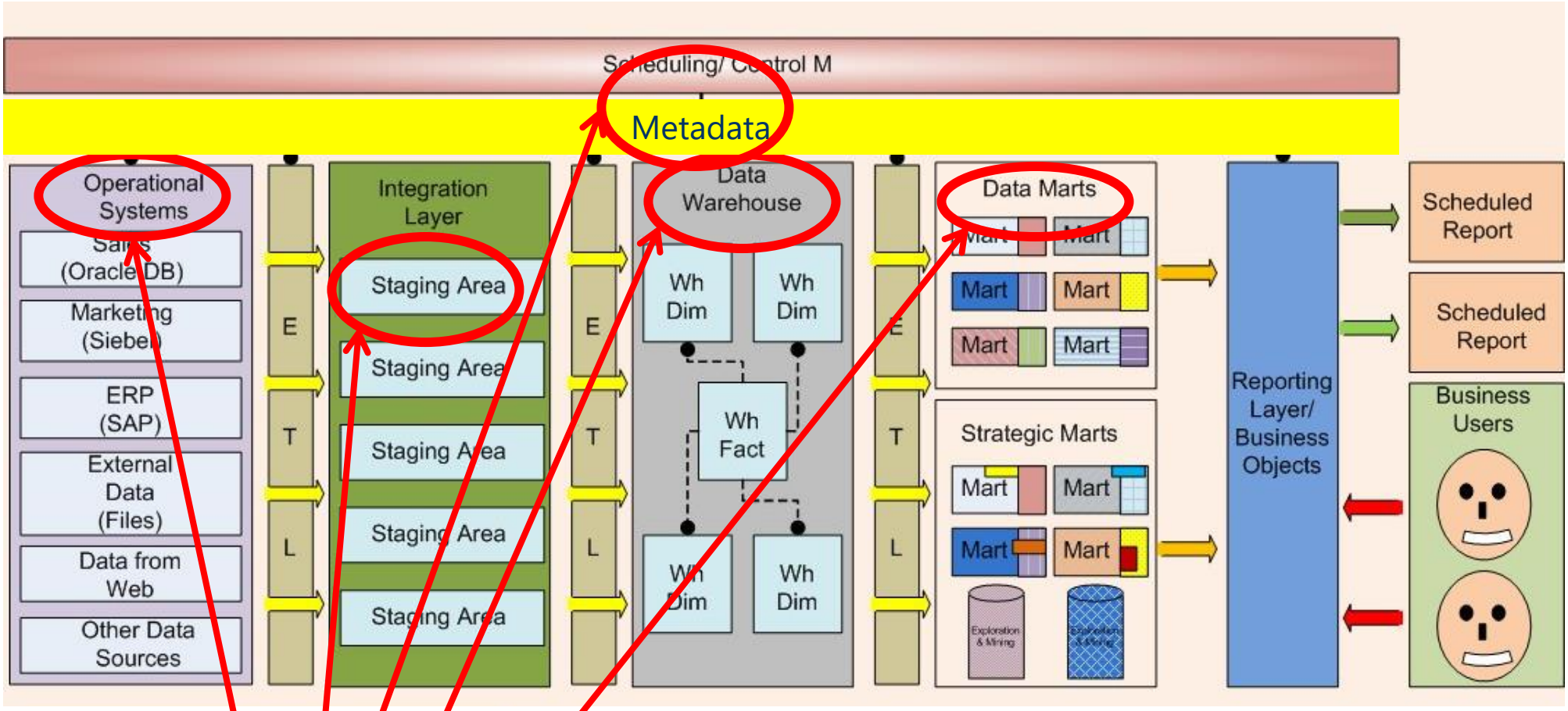
caracteristicas

- **Load manager:** runs ETL tasks
  - **Extract:** Retrieves the data from various source systems.
  - **Transform:** applies business rules, clean data, format it for storage.
  - **Load:** inserts the transformed data into the DWH.

esquema de estrella siguen siendo valido en una base de datos columnar

- **DWH manager (server):** it allows to define and maintain the datawarehouse: data definition, aggregation, views, index, backup, etc..
- **Query manager:** Query execution, monitoring, ad-hoc forms, etc.
- **Access tools:** tools to design queries and reports, tools to develop end-user applications, OLAP tools, data mining tools, enterprise Information Systems (EIS)

Conceptual Architecture and Components



Data in a DWH

- **Data sources:** files, web, xls, databases, …

staging area procesa los datos, landing area es donde se guardan, no son lo mismo auqnue se suelen mezclar

- **Staging area:** temporary storage where data is cleaned and transformed before going into the DWH.
  - E-R, Relational model. Not compulsory to have it implemented, but usually convenient.
- **Datawarehouse:** data collection for **decision making**
  - **Structure:** Multidimensional model
- **Data mart:** departamental DWH

un data mart para cada seccion de la empresa

- **Metadata:** describes business activities, the business objects, and rules that guide those activities.
  - Technical meta data needs to be mapped to the business meta data.
  - Includes documentation about data sources (origen, description, aggregation level, storage, …)

Data Warehouse: Features of a DW

- The core of a BI architecture is the **Data Warehouse**
- It provides a **consistent** and **unified** source of data for decision-making processes.
- Data Warehouse: A **central repository** of data designed to support the decision-making processes.
  - Information Oriented (not processes)
  - Integrated
  - Time-variant
  - Nonvolatile

unica fuente de verdad de la organizacion !!

Features of a DW

- **Information oriented** (not processes):

  DWH is designed to **efficiently view information** on the basic activities (sales, purchasing, production, ...) of the organization.

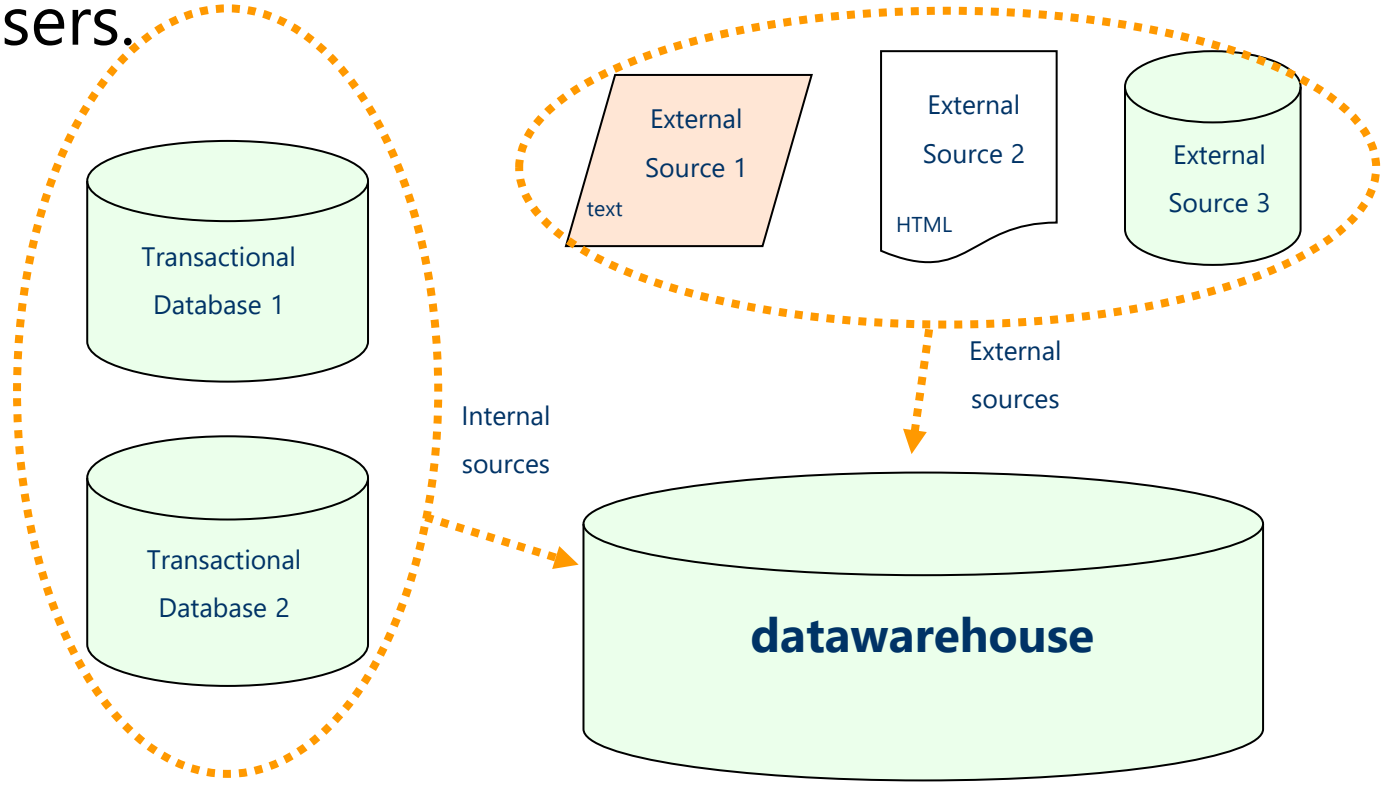
  It is not meant to support **operational processes** like **order management or billing**.

  - Necessary information is extracted **from transactional systems** and efficiently stored for analysis.
  - It leaves out irrelevant information. Only the **necessary data** is extracted and stored for **analysis**.

lagos de datos: almacenes de datos semi, no, o incluso estructurasdos donde almacenamos todo lo que creemos que puede tener valor para la organización. Se linkea el datawarehouse con ese lago

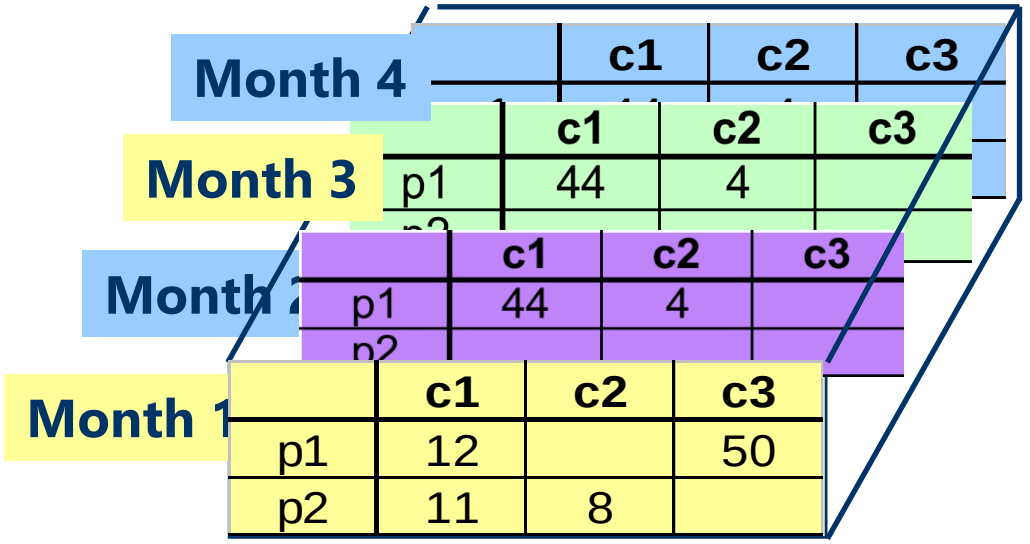
Features of a DW

- **Integrated:** la integracion es muy dificil
  - it collects data not only from the **internal transactional databases**, but it can also include **external sources**.
  - It must be **consistent** to present a **unified view** of data to users.



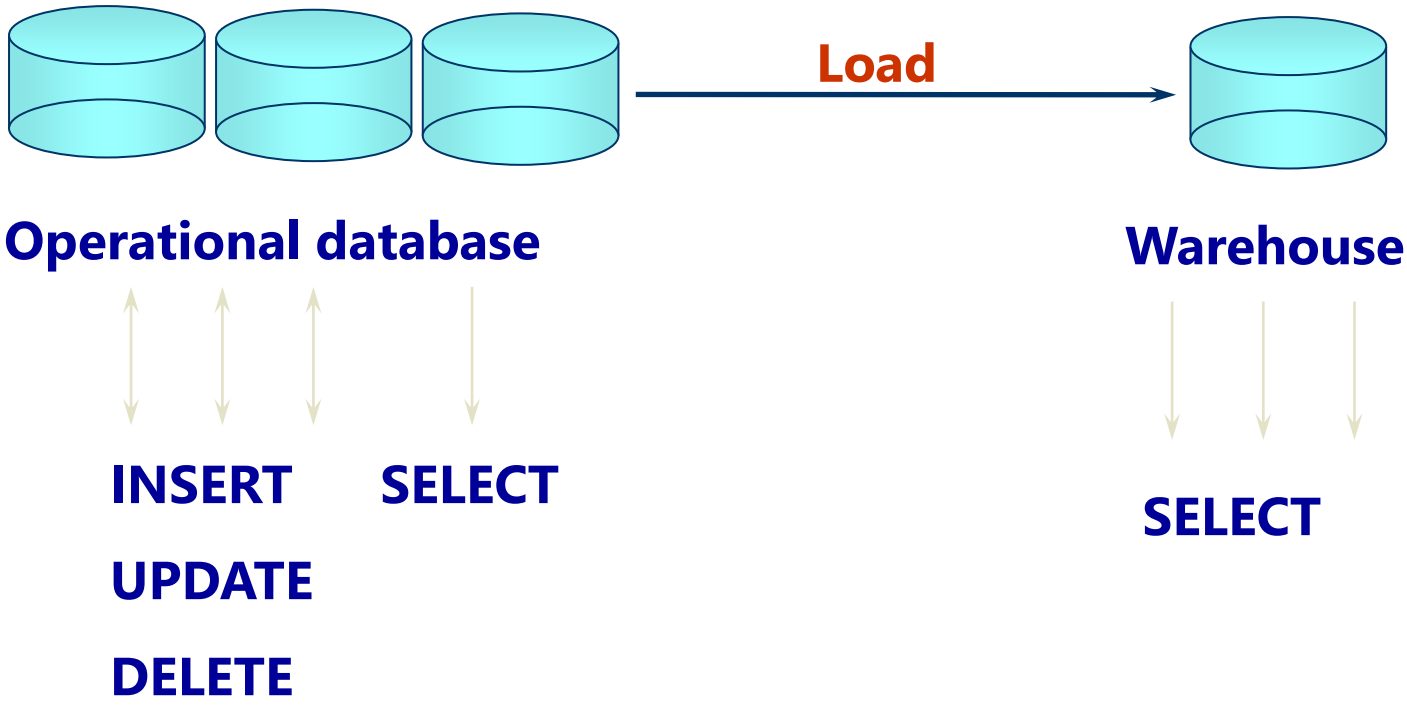
Features of a DW

- **Time-variant:**
  - Data are relative to **a period of time.**
  - It is **periodically updated** to include new information.
  - Detection of historical **changes**, **trends**, **patterns**, …



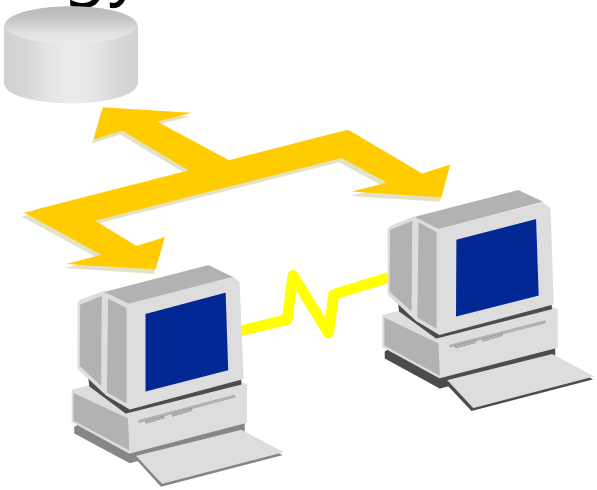
Features of a DW

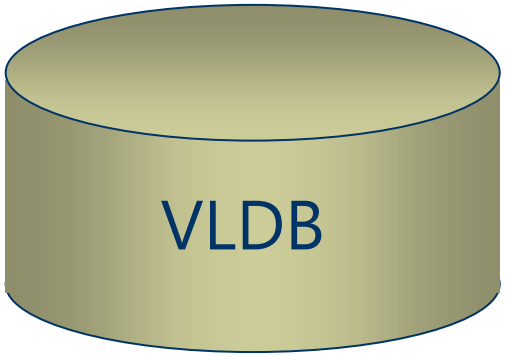
- **Non-volatile:**
  - the stored data are **not** (typically) **updated or deleted**, **only increased.**
  - Data in the DWH remains **stable** over time.



Introduction

- Advances in technology have encouraged the development of data warehouse technology
  - **Big data technology**
    - Parallelism
    - Hardware
    - Distributed operating systems
    - Database
    - Query languages





- VLDB
- Big memory
- Indexing techniques
- Open systems (interoperability)
- Specialized HW and SW for DWH
- Tools for data analysis

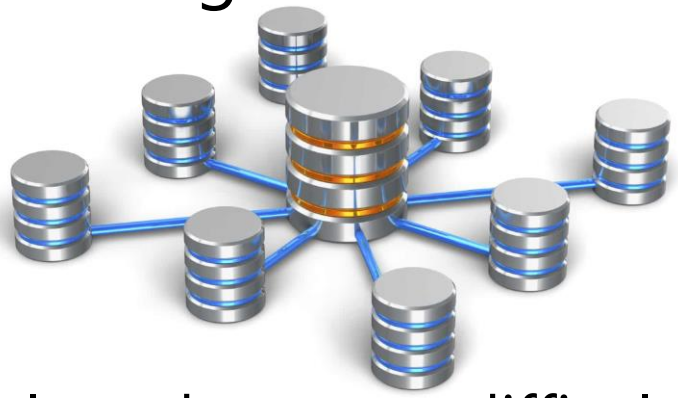
OLTP vs DWH

resumen de diferencias

OLTP System	Data Warehouse
Stores current data	Stores historical data (including current data for trend analysis)
Stores detailed data	Stores summarized or aggregated data
Data are dynamic (updatable)	Data are static (non-volatile)
Repetitive processes	Ad-hoc, unforeseeable processes
Predictable usage pattern	Unpredictable usage pattern
High rate of transactions	Medium or low rate of transactions
Low response time (seconds)	Variable (usually longer) response time
Transaction-oriented	Data analysis-oriented
Application or process-oriented	Information-oriented
Supports daily operational decisions	Supports strategic decisions
High number of users (administrative)	Few users (managers, analysts)
Medium-sized databases	Large-sized databases

Advantages of using DW

- The **advantages** of using DW are, among others:



  - **High ROI**
  - **Competitive advantages**:
    - Information non previously available, unknown or difficult to extract and incorpórate.
  - **Increased productivity**: decisiones más rapidas y mas informadas
    - More integrated and easy-to-access information.
    - Make faster and more informed-decisions.
  - **Better data quality and consistency**

Problems

Resources	Understatement of resources to load
	High demand for resources
	High cost of ownership
Data Integration	Complexity of integration
	Hidden problems of source systems
Data Capture	Required data are not captured
	Homogenization data
Usage	Increased demand from end users
	Data ownership
	Long-term projects

Multidimensional model

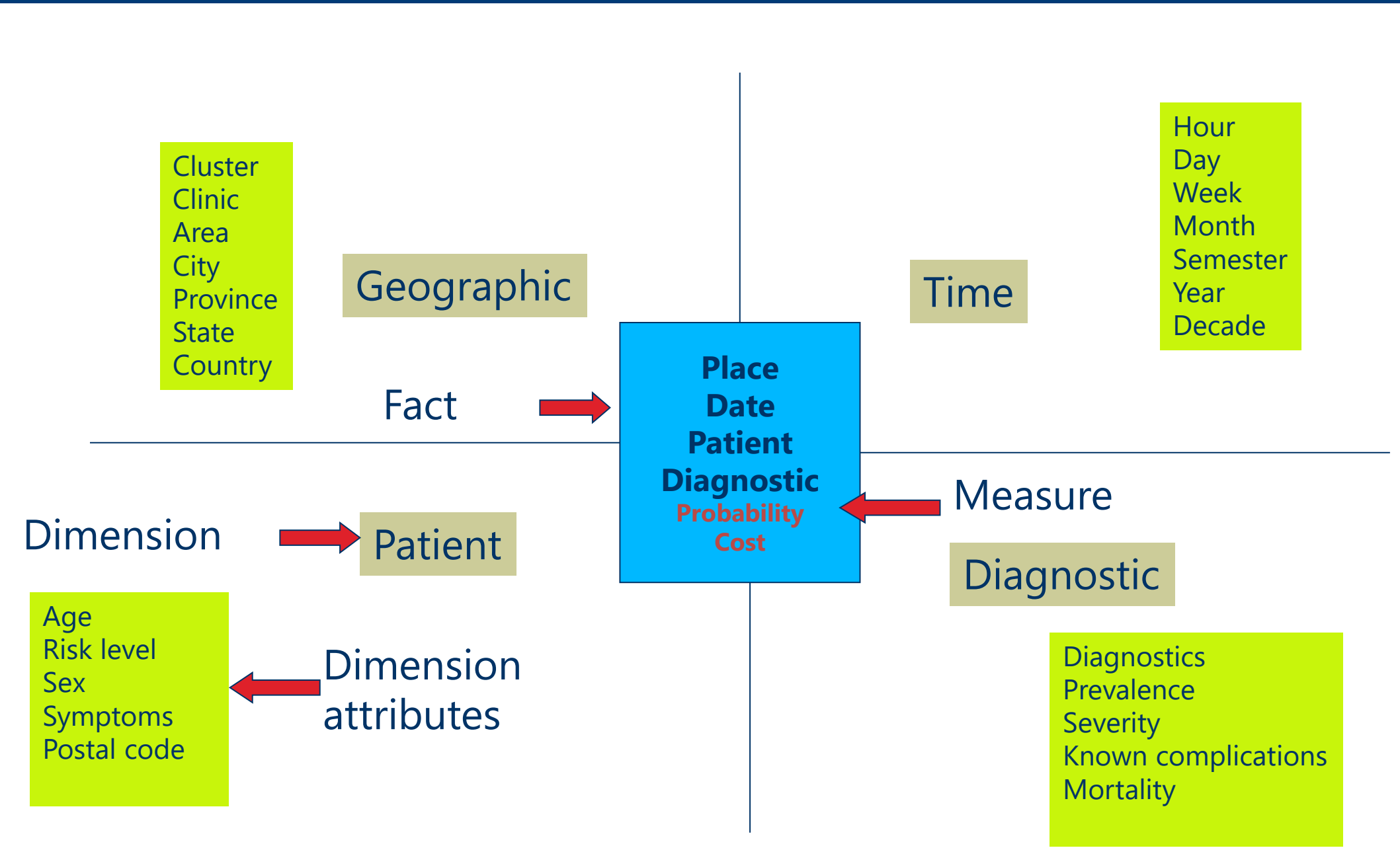
una dimension no deberia crecer al mismo tiempo que la de hechos

- **Multidimensional model:**
  - A **fact** is the activity or event being analyzed.
  - A **fact table** stores the quantitative data or measures, which are usually **aggregated** for analysis.
  - **Dimensions** are descriptive attributes that provide context for the fact (e.g. time, location, customer, …).
  - The **dimension tables** store the **dimension attributes**. They are connected to the fact tables via **foreign keys**.

Multidimensional model

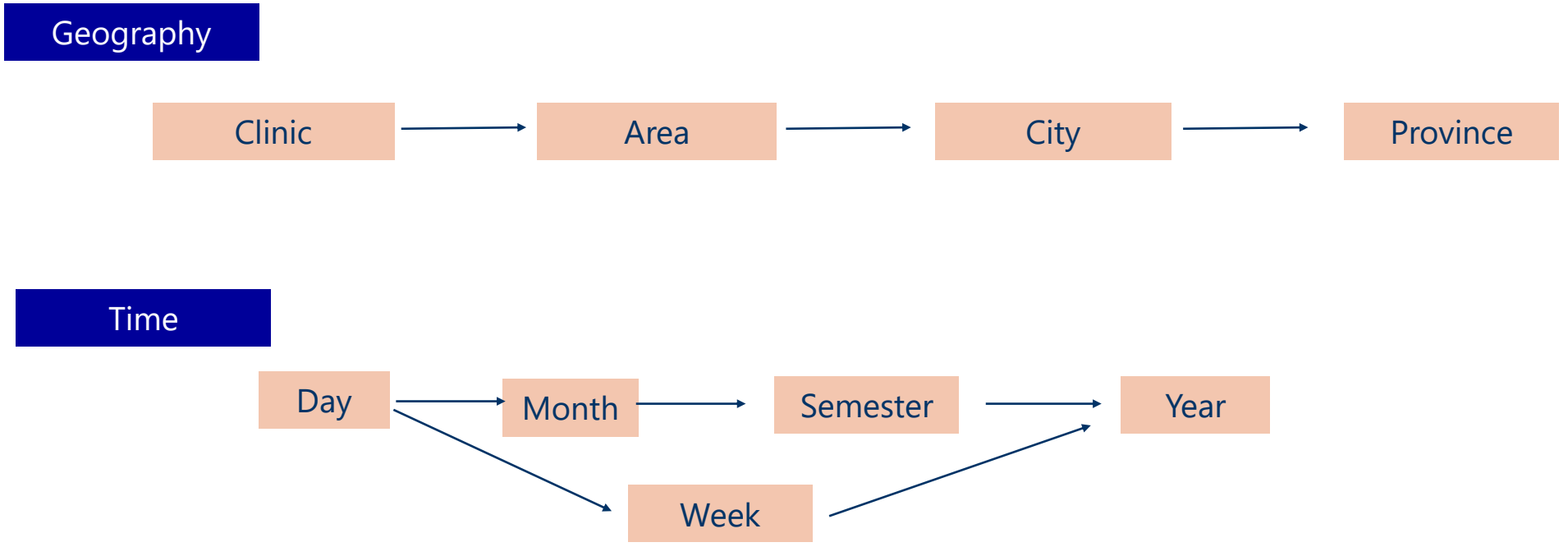
- **Activity analized**: Diagnostics.
- Information recorded about diagnostic: **"Avian influenza diagnostics** has been realized in the **clinic "Morales"** on **Oct, 11th 2012** with a probability of **85%** to the **patient "Joseph"**. las dimensiones las vemos en la siguiente tabla
- The **geographic** and **temporal context** are important, not only the concrete diagnostic to a person.
- **Time** is hierarchical, sequential and many types of aggregations and calculations are made using it.

Multidimensional model



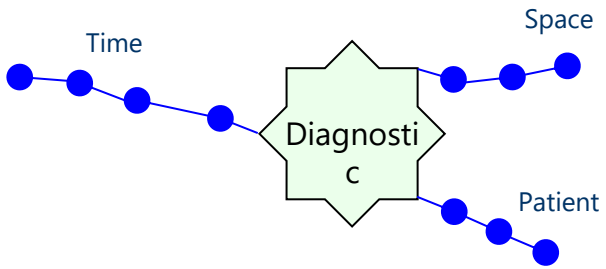
Multidimensional model

- **Hierarchy:** organization of dimensión attributes in levels where data can be analyzed at different levels of detail.

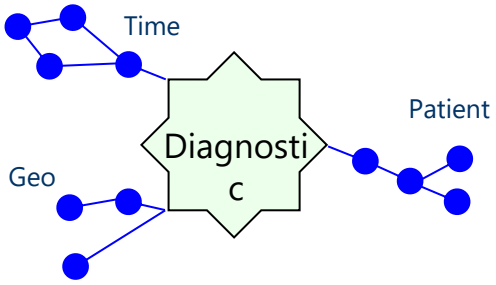


Multidimensional model

- **Basic structures**
  - **Star schema:** Direct relationship between fact table and dimension tables. Dimensions are **denormalized**.



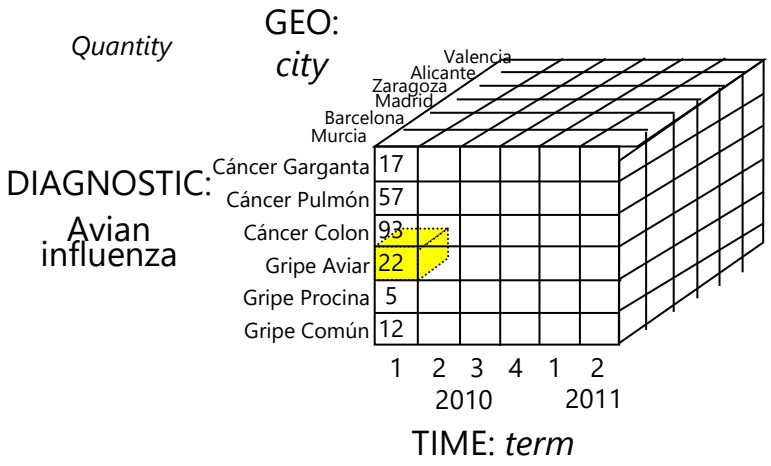
  - **Snowflake schema:** Dimensions are **normalized**. Reduced redundancy but more joins during queries.

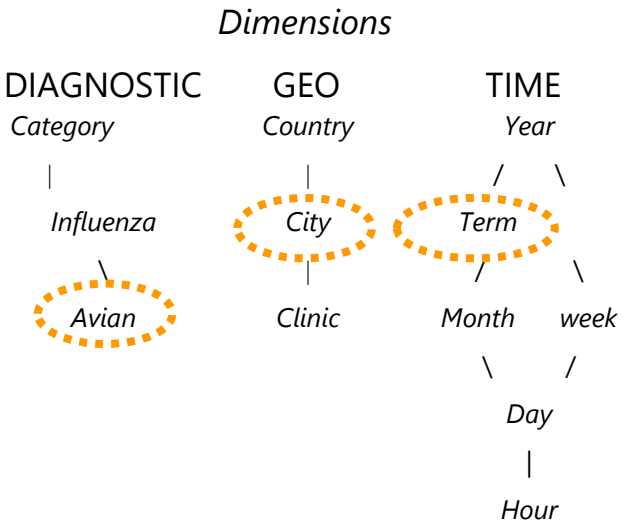


Multidimensional model

- **Facts** can be queried at **different aggregation levels**:
  - Query **measures** about the **facts** defined by **dimensions**' attributes and constrained by values of those **attributes**

FACT: *"The first term of 2010 22 cases of avian influenza were diagnosed in Murcia*

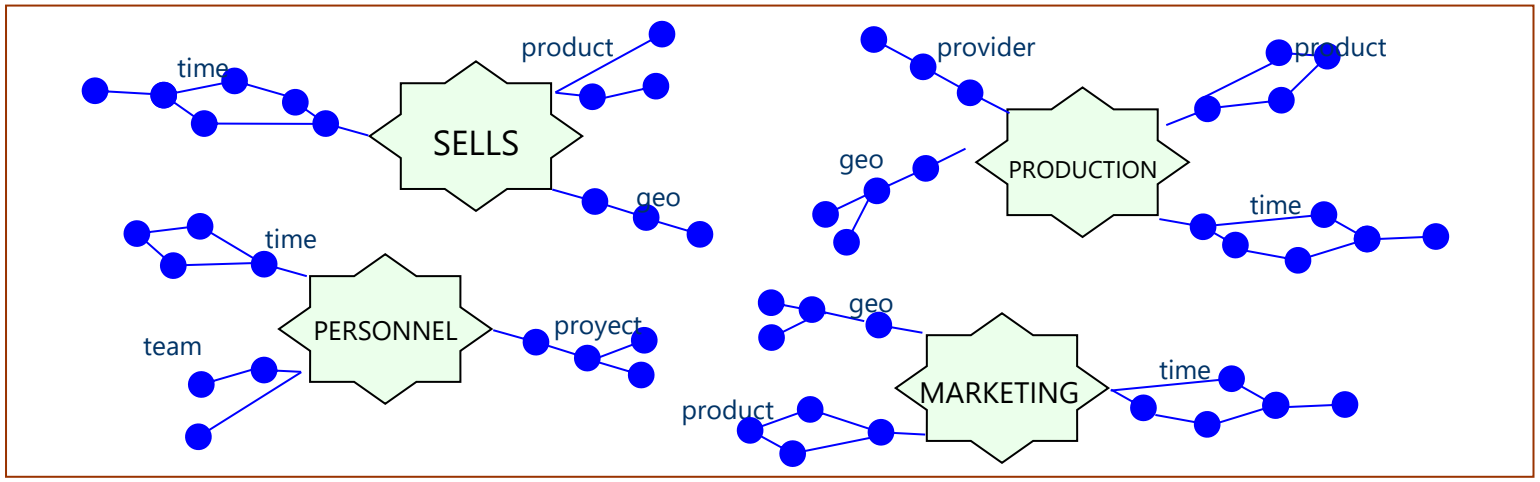




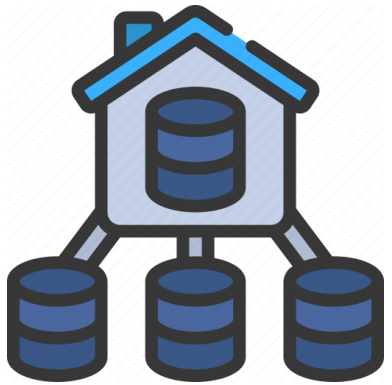
- An aggregation level for a set of dimesions is called cube.

Multidimensional model

- Not all the information can be stored in a single schema
  - Some **departamental DWH** are needed
  - Each of these is called **datamart**. implementarlo si el DW esta sobrecargado
  - A data mart is a **subset** of the data warehouse that serves the **specific** needs of a department or business unit. They can be created from scratch, **independently** from the DWH.



Design strategies



- **Datamart:**
  - defined to meet the needs of a department or subdivision of the organization.
  - contains less detail and more aggregated information.

como el DW es la unica fuente de verdad, tiene mas datos que un datamart, por lo que necesita tambien la maxima granularidad

- **Top-down (Inmon): Data-centric**
  - First define the data warehouse of the whole organization and then define data marts on it.
- **Bottom-up (Kimball): Business-centric**
  - predefine departmental data marts and then integrate them into a data warehouse for the organization

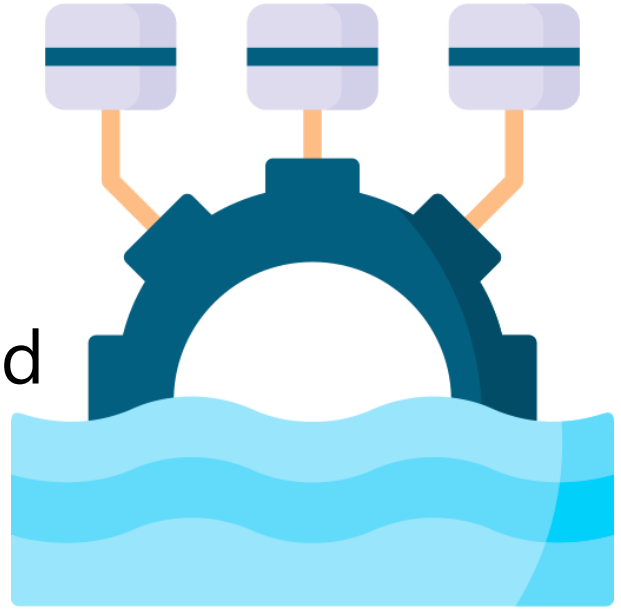
Datamart

- **Datamart:** subset of the DWH.
- The **DWH** can be formed by **several datamarts** and, optionally, **additional tables**.
- Are defined to meet the needs of a department or subdivision of the organization.
- Contains **less detailed information** and **more aggregated** information.
- They are **easier to understand** and use.
- They may be the **intermediate step** between the data warehouse and transactional system

Datamart

- **Why datamarts:**
  - Provide users with access to the **most commonly** analyzed data.
  - Smaller and focused data, with **better response time**.
  - **Offloads** the primary DWH
  - The data will already be **adapted** for OLAP queries or DM.
  - **More control** over their data mart by department users.
  - Being **simpler**, ETL processes are too.
  - **Low cost** and **faster implementation**.
  - Greater involvement of users in the overall data warehouse. **Better data quality, security, relevance**,…

Data lakes



- In DWH data are transformed and structured, and we keep only the data needed for the analysis.
- Are we **losing** data that may be needed in the future?
- **Data lakes** store **ALL** source data in original format (may be not transformed)
- Easy to transform later for new DWH or for data analysis
  - But **schema-on-read**. Is it good?
- Created for, not for business users**data scientist and analysts**

Business intelligence

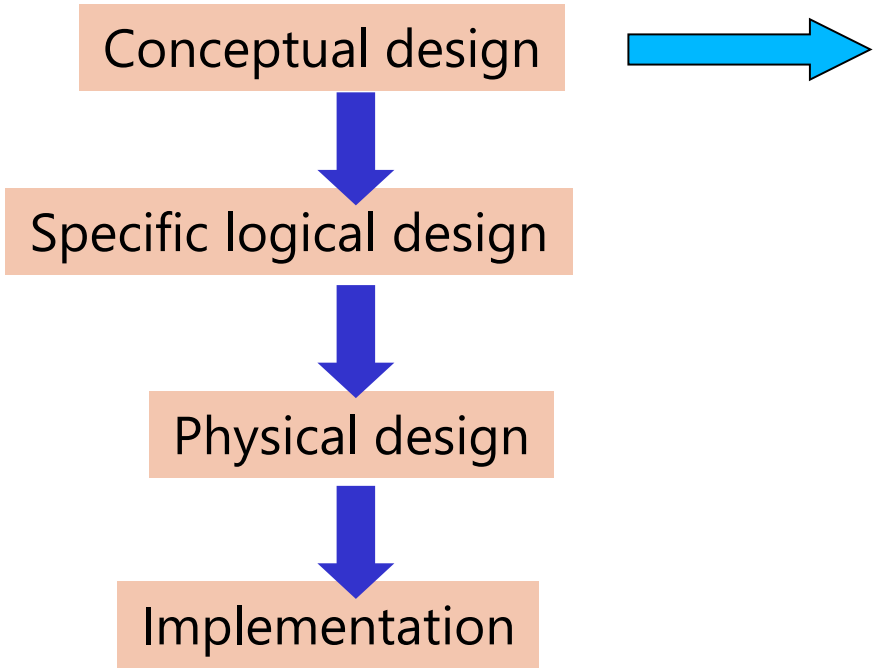
Unit 2 – Datawarehouse and OLAP
S2-2 – Datawarehouse design

Datawarehouse design

Does it ring a bell?

- Conceptual design
- Specific logical design
- Physical design
- Implementation

Datawarehouse design



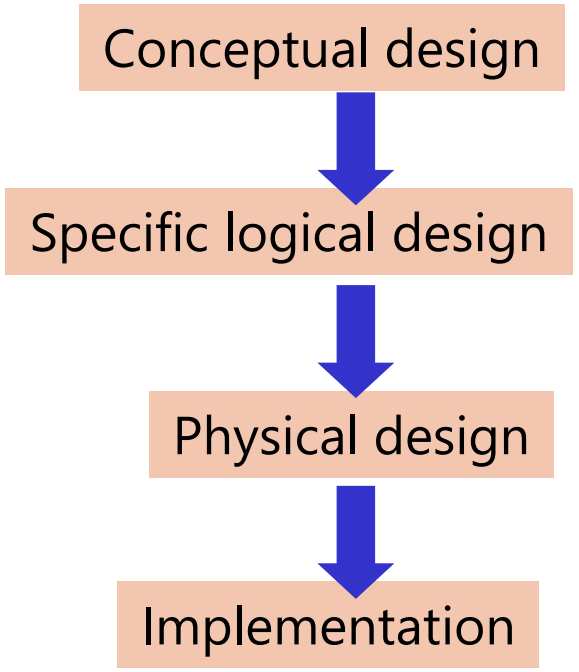
Requirement analysis

- Understand **business requirements.**
- Identify **business processes, KPIs**.
- Identify **data sources.**
- Identify **facts** and **measures.**

Conceptualization

- Eg: Identify multidimensional model

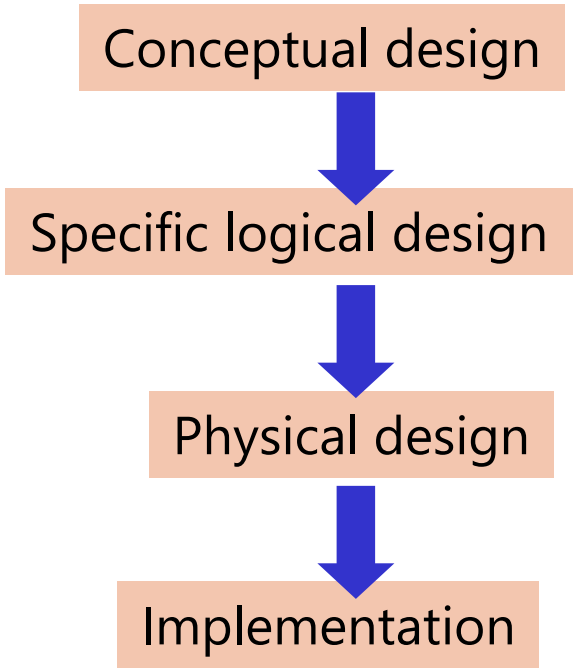
Datawarehouse design



Multidimensional modeling

- Star and snowflake schemas
- Kimball's methodology

Datawarehouse design

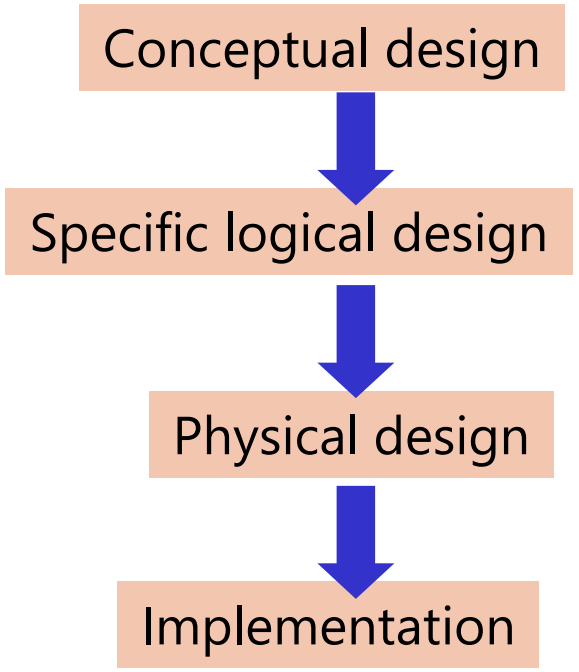


Storage management
(ROLAP, MOLAP, HOLAP)

Big data technologies

ETL Process design

Datawarehouse design



ETL and Data Integration implementation

OLAP tools

Datawarehouse design

- **Multidimensional model:**
  - models an activity (**fact**) and **dimensions** that characterize the activity.
  - relevant information about the event (activity) is represented by **measures.**
  - Usually, the fact table uses a **composite key** created from the dimension's FK.
  - Descriptive information for each dimension is represented by **dimension attributes.**
  - Each dimension table uses a **simple key** to uniquely identify each record.

ER vs Multidimensional model

The Multidimensional Model and Entity Relationship have connections but are different.

- **application**
  - ER is used for transaction systems (OLTP).
  - MM is used for data analysis (OLAP).
- **structure**
  - ER identifies and eliminates redundancy relations.
  - MM usually includes denormalization for more efficient queries.
- **use**
  - ER queries can be complex, with multiple joins.
  - MM queries are simple and efficient.

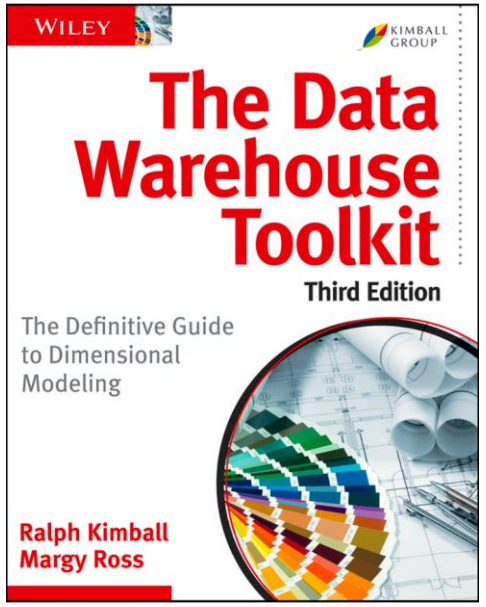
Methodology: multidimensional modeling

- **We start from:**
  - **Knowledge** about the domain (e.g. conceptual model)
  - **Data Sources**
  - **Indicators**: understand user queries
- **Objective:** resolve user queries efficiently.
  - **Methods** focused on **logical** and **physical design**
  - [**Kimball, 96**]: Methodology of **9 steps**

Methodology for multidimensional modeling

9-step methodology [Kimball, 96] :

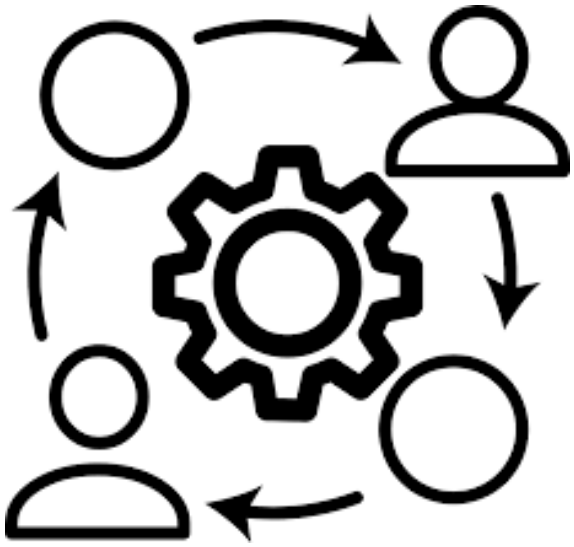
1. Select the **process.**
2. Select the **granularity**.
3. Identification and conformation of the **dimensions.**
4. Selection of the **facts.**
5. Storing **derived** or **aggregated** values in fact tables.
6. **Complete** the dimension tables.
7. Select the **duration** of the database.
8. Manage **slowly changing dimensions**.
9. Select **priorities** and **query modes**.



Methodology. Step 1

Step 1: Select the process

- **Process:** business activity that the DWH will model.
- A process is supported by **OLTP** systems.
- Start with the **most important** to the organization.
- **Examples**: diagnoses, deceased, inventory, billing, …



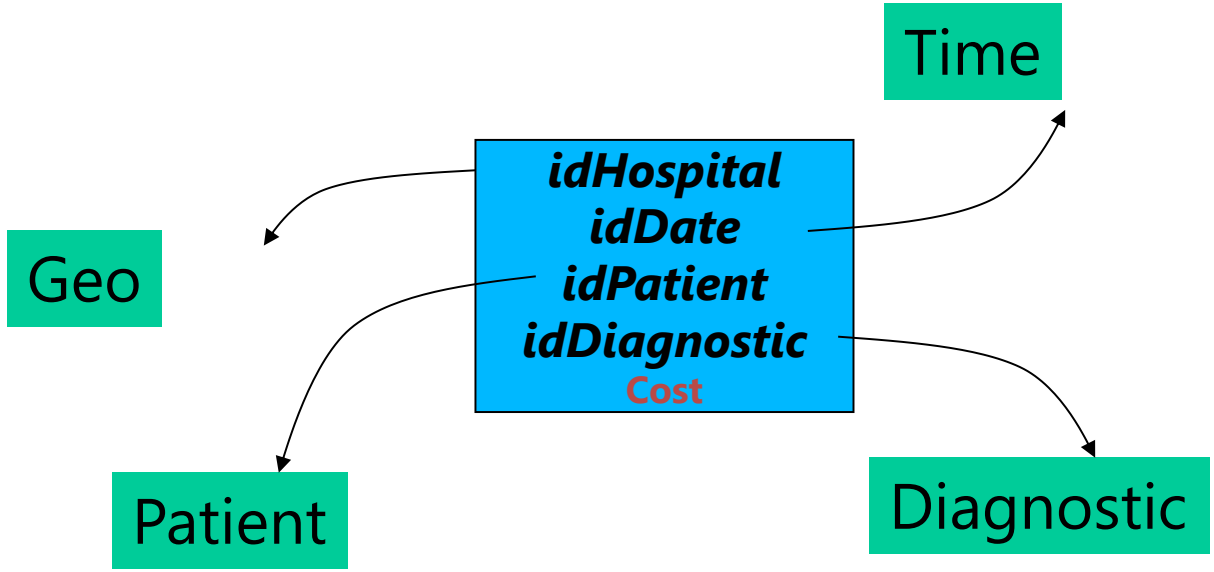
Methodology. Step 2

Step 2: Select the granularity

- **Granularity**: lowest level of detail in which information is stored in the fact table.Tell us how **detailed** our data is.
- It affects the **DWH size**, **analysis flexibility**, **query complexity**,…

- **Example**: weekly cost of diagnoses in health centers

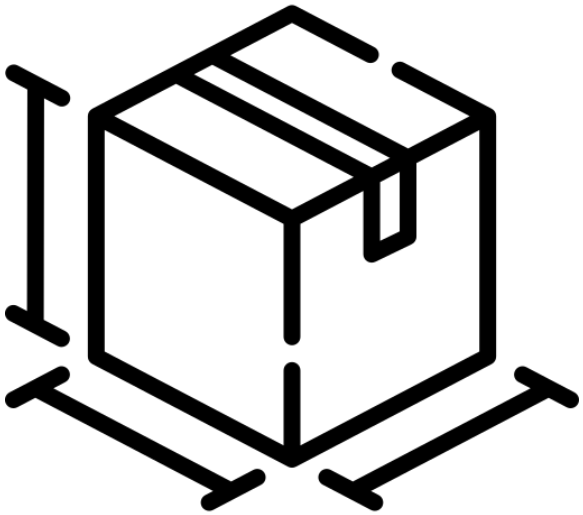
¿Days? ¿Weeks?
The level that allows the better analysis: Fine grain
¿Test performed or cost?



Methodology. Step 3

Step 3: Identification and conformation of the dimensions

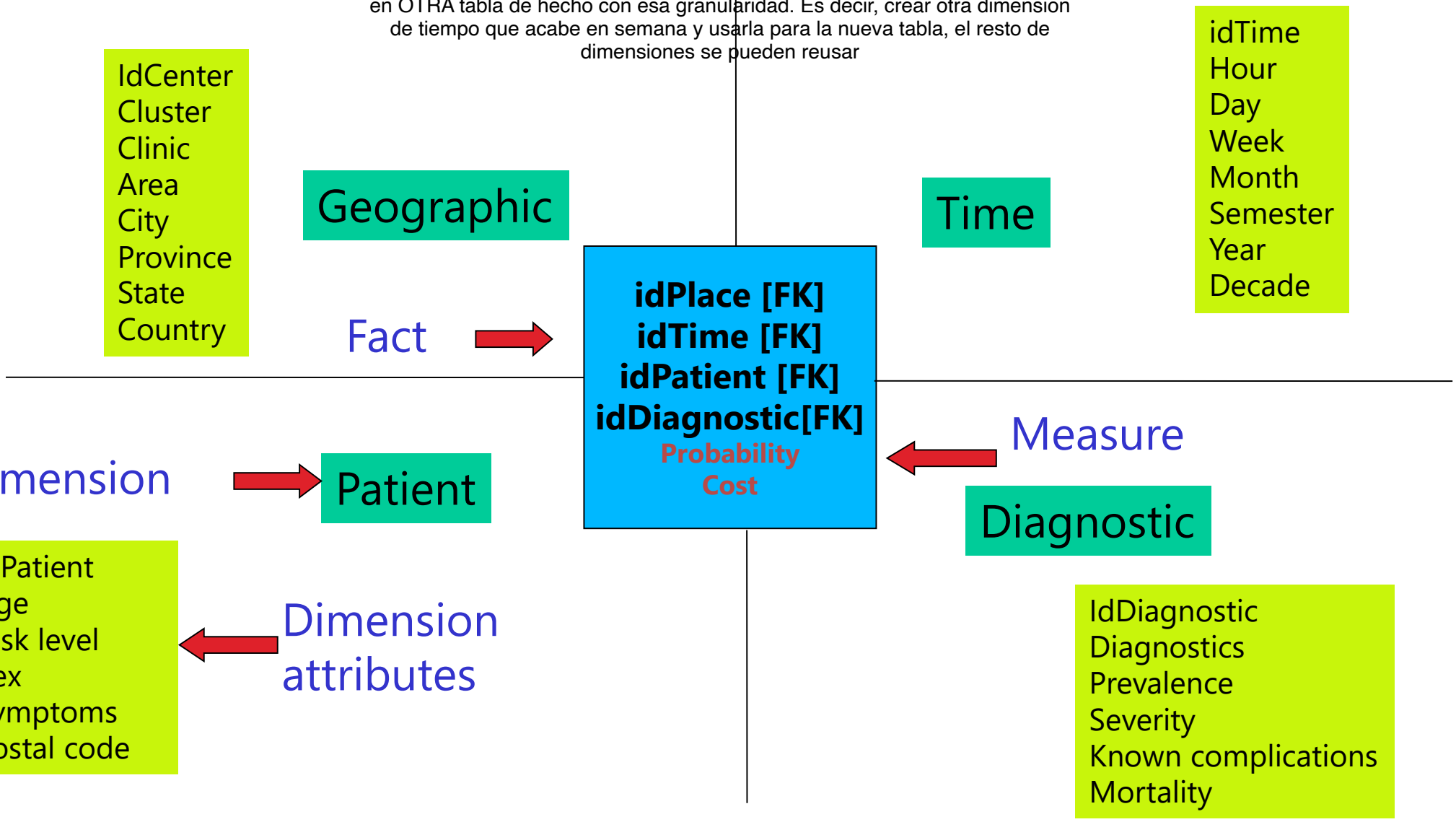
- After identifying the facts, the dimensions and their attributes **are defined**.
- **Dimensions**: characterization of facts at the selected level of detail.
- They are **descriptive** and are **query parameters**.
- **Hierarchies** between dimension attributes.



IdCenter
Cluster
Clinic
Area
City
Province
State
Country

Methodology. Step 3

is queremos granularidad semanal en la tabla de hechos, hay que guardarlos en OTRA tabla de hecho con esa granularidad. Es decir, crear otra dimension de tiempo que acabe en semana y usarla para la nueva tabla, el resto de dimensiones se pueden reusar



Methodology. Step 3

- Dimensions define the **interest areas** for analyzing the facts:
  - Age
  - Age ranges
  - Sex
  - Risk level
  - Nacionality

idPatient
Age
Risk level
Sex
Symptoms
Postal code

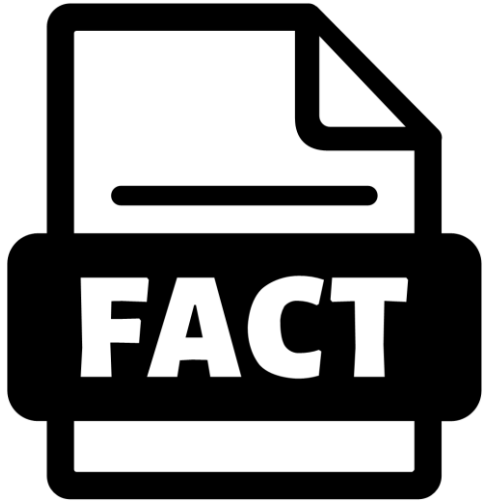
- Special attention is paid to **time** and **space** (they apply to almost every type of business analysis)

Methodology. Step 3

- Common attributes in the **time** dimension:
  - Day number, month number, year number, number of weeks.
  - day of the month (1 .. 31): allows comparisons on the same day across different months (e.g. sales on the 1st).
  - weekday (Monday ... Sunday).
  - month-end or Weekend indicators: comparisons of the last day of the month or weekend in different months.
  - quarter (1 .. 4): analysis of a specific quarter in different years.
  - holiday indicator: allows analysis on days adjacent to a holiday.
  - season (spring, summer, autumn, winter).
  - special event (football, elections, earthquake, ...)

Methodology. Step 4

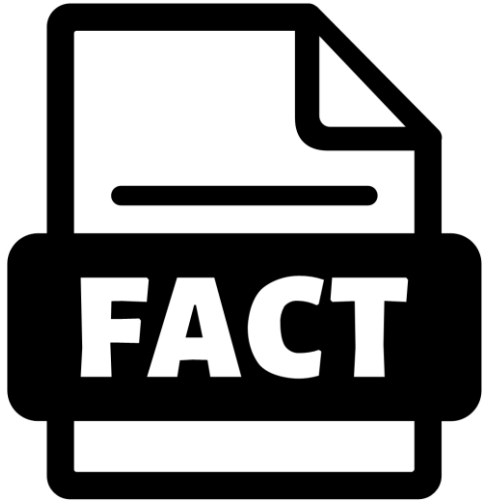
Step 4: Select the facts.



- **Fact**: analyzed information stored in the fact table.
- Useful facts are usually **numerical**, **additive**, or in general **facts that can be aggregated.**
- This is because large sets of the fact table are normally queried.
- **Select events** based on the **granularity of the information** chosen:
  - Cost of the treatment
  - Cost of the tests
  - Deceased status
  - Total daily detected

no se meten medidas cualitativas en una tabla de hechos

Methodology. Step 4



- **Facts**
  - **Additives**: can be aggregated in all dimensions.
    - **Activity data** are typically additives.
    - **Examples**: sales, units, money.
  - **Semi-Additive**: they can be aggregated only in some dimensions.
    - **Intensity data** are not usually additives.
    - **Examples**: Stock, total existing stocks (can be added across some dimensions but not in time)
  - **Non-Additives**: they can not be aggregated across any dimension.
    - **Examples**: temperature, unit price, percentage, …
    - Can be aggregated by average values
    - **Dummy variables**: (0 or 1) to indicate occurrence.

Methodology. Step 4

- **Fact Tables:**
  - **Transactional**: represent detailed events in space time.
    - Provides the maximum level of exploration. maximo nivel de detalle
  - **Factless**: contain no measures, only the occurrence of certain events. tablas de hechos sin hechos - marcan la ocurrencia de un evento
    - Use to establish **relations between dimensions**.
    - **Examples**: students to class attendance, products that are not sold (negative analysis)
  - **Snapshot**: Each row is an instant of time.
    - Describe the state of the facts in a particular moment in time.
    - Normally includes semi-additive and non-additive facts.
    - They are often taken at predefined intervals and can be cumulative.

Methodology. Step 5-7

Step 5: Storing derived or aggregated values in fact tables.

- Include **derived attributes** that may be useful (e.g., non-additive attributes).
- **Examples**: differences, decomposed values (eg numerator and denominator) or calculated (amount= the price * units)

Step 6: Complete the dimension tables

- Add textual descriptions to the dimensions.
- Intuitive and understandable for users.

Step 7: Select of the duration of the database

en muchos casos hay datos historicos

- Set date from which to store data.
- It depends on the problem: data validity, business requirments, data availability, storage constraints, ...

Methodology. Step 8-9

por ejemplo guardar las facturas durante X años, etc

Step 8: Manage slowly changing dimensions (SCD)

- When an attribute changes value but **not the key.**
- **Example**: change in marital status, job category, ….
- Some **solutions** to slowly changing dimensions:
  - **Type 1**: overwrite a changed dimension attribute.
  - **Type 2**: create a new dimension record.
    - Current value (active, valid, …) + validity date
    - **Note**: Business key are repeated. *Handle queries with care.*
  - **Type 3**: establish an alternate attribute, so that both the old and the new are accessible.
  - **Type 4:** mini-dimension (also historical table)
  - Also **combinations**: types 4, 5 (4+1), 6 (1+2+3), 7

Step 9: Select priorities and query modes (physical design)

Step 8: Control slowly changing dimensions

TABLA DE HECHOS

MEDIDAS	PROYECTO	INVESTIGADOR
1	P1	I1
2	P2	I1
3	P3	I1
4	P4	I1

TABLA DE HECHOS

MEDIDAS	PROYECTO	INVESTIGADOR
1	P1	I1
2	P2	I2
3	P3	I3
4	P4	I3

TABLA DE HECHOS

MEDIDAS	PROYECTO	INVESTIGADOR
1	P1	I1
2	P2	I1
3	P3	I1
4	P4	I1

TABLA DE HECHOS

MEDIDAS	PROYECTO	INVESTIGADOR	ESTADO
1	P1	I1	E1
2	P2	I1	E2
3	P3	I1	E3
4	P4	I1	E3

DIMENSION INVESTIGADOR

PK	BusinessKey	Nombre	Categoría	Facultad
I1	Manolo	M. Campos	AYD	Informática

TIPO 1: REESCRIBIR CATEGORÍA

TIPO 2: NUEVO REGISTRO. MANTENER BK

DIMENSION INVESTIGADOR

PK	BusinessKey	Nombre	Categoría	Facultad	FECHA_INI	FECHA_FIN	VIGENTE
I1	Manolo	M. Campos	AYD	Informática	2005	2009	N
I2	Manolo	M. Campos	CD	Informática	2010	2016	N
I3	Manolo	M. Campos	TU	Informática	2017	-	Y

TIPO 3: NUEVO ATRIBUTO

DIMENSION INVESTIGADOR

PK	BusinessKey	Nombre	Categoría AC	Categoría	Facultad	FECHA_INI	FECHA_FIN
I1	Manolo	M. Campos	TU	CD	Informática	2005	2009

TIPO 4: MINIDIMENSION

DIMENSION INVESTIGADOR

PK	BusinessKey	Nombre	Facultad	FECHA	ESTADO
I1	Manolo	M. Campos	Informática	2005	E1

MINIDIMENSION ESTADO

PK	Categoría ACTUAL
E1	AYD
E2	CD
E3	TU

TIPO 5: MINIDIMENSION +OUTRIGGER

Methodology: Errors / guidelines (Kimball)

Error 1: Not sharing dimensions between different fact tables.

- Correct: Unify master files. Example: Gender {'M', 'F'}.

Error 2: Not unifying facts from different fact tables

- Correct: Unify facts, even if they come from different departments or systems. Example: retail and enterprise sale.

Error 3: Ignoring aggregate tables

- Correct: instead of adding hardware to solve performance issues, use pre-aggregated tables to improve query performance.

Error 4: Forget the highest level of detail in the data model.

- Correct: Maximum detail in 3 areas: staging, relational and dimensional.

area de procesamineot de ETL

Methodology: Errors / guidelines (Kimball)

Error 5: Mix facts of different granularity in the same fact table.

- Correct: Avoid mixing different granularities in a fact table. It is better to create separate tables for different aggregation levels.

Error 6: Create a dimensional model to solve a particular report instead supporting multiple analytical needs.

Error 7: Adding dimensions to a fact table before setting its granularity.

- Correct: The fact table only contains FK and measures. Do not store dimensional data directly in the fact table.

Methodology: Errors / guidelines (Kimball)

Error 8: Create "smart keys" to relate a dimension table to a fact table.

- Correct: Surrogate keys should be auto-increment (even for the time dimension)

Why?

- Heterogeneous data sources keep their own business key.
- Changes in source applications should not affect the DWH.
- Performance (storage size and comparison speed).

TABLA DE HECHOS

MEDIDAS	PROYECTO	NVESTIGADOR
1	P1	~~Manolo~~
2	P2	~~Manolo~~
3	P3	~~Manolo~~
4	P4	~~Manolo~~

DIMENSION INVESTIGADOR

PK	BusinessKey	Nombre	Categoría	Facultad
I1	Manolo	M. Campos	AYD	Informática

Methodology: Errors / guidelines (Kimball)

Error 9: Not to address slowly changing dimensions.

Error 10: Splitting hierarchy levels into multiple dimensions. "error suave", no hacer normalmente hay que hacerlo de forma justificada

Error 11: Shorten the descriptions in the dimension tables with the intention of reducing the space required.

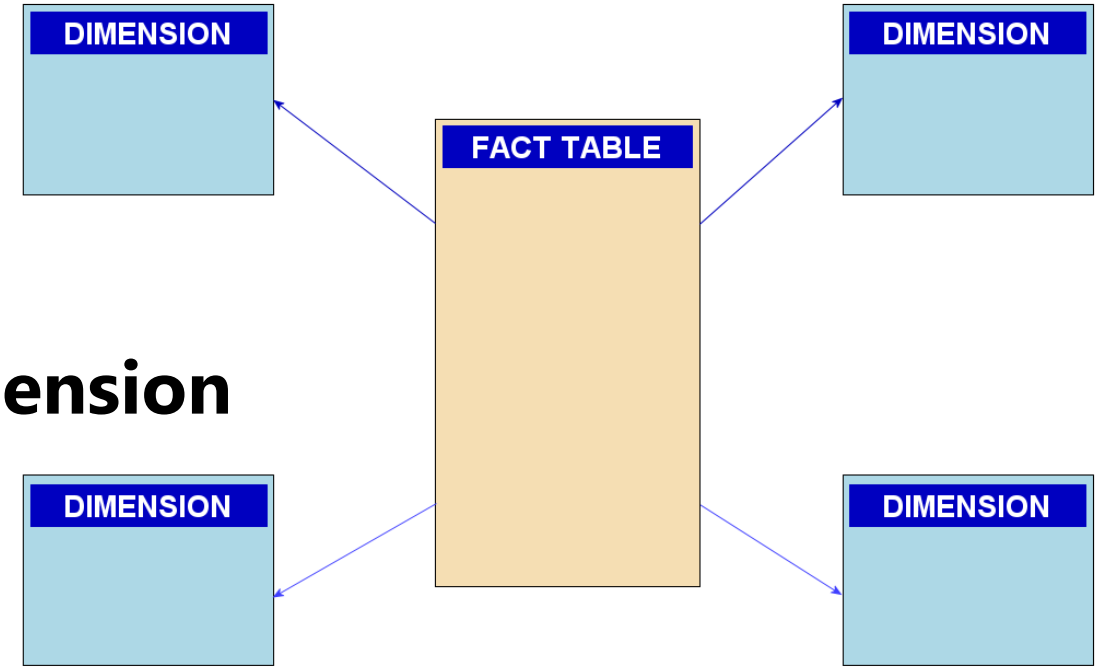
- The dimensions are the interface that users have to browse.
- They take up little space in relation to the facts.

Error 12: Include text attributes in a fact table, if done with the intention of filtering or grouping.

DW Design techniques

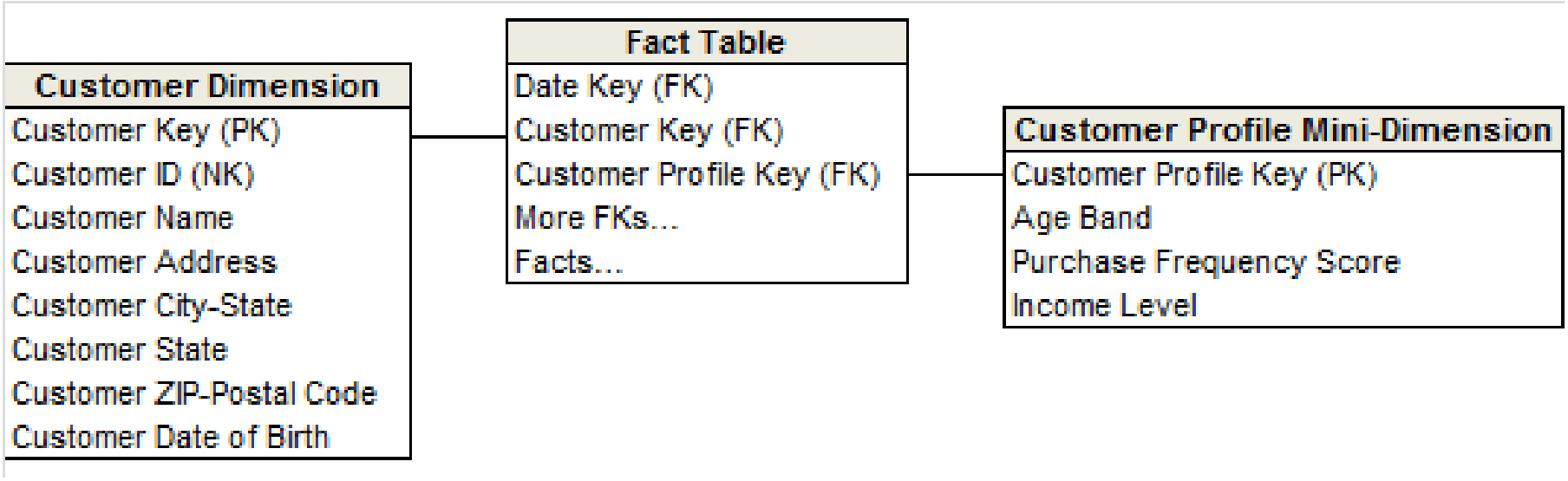
tipos de dimension

- **Date and time**
  - Minidimension
  - Several time zones
- **(Shrunken) Rollup dimension**
- **Junk dimension**
- **Degenerate dimension**
- **Bridge table**
- **Variable depth hierarchies**
- **Outrigger dimension**
- **Snowflake dimension (normalization)**



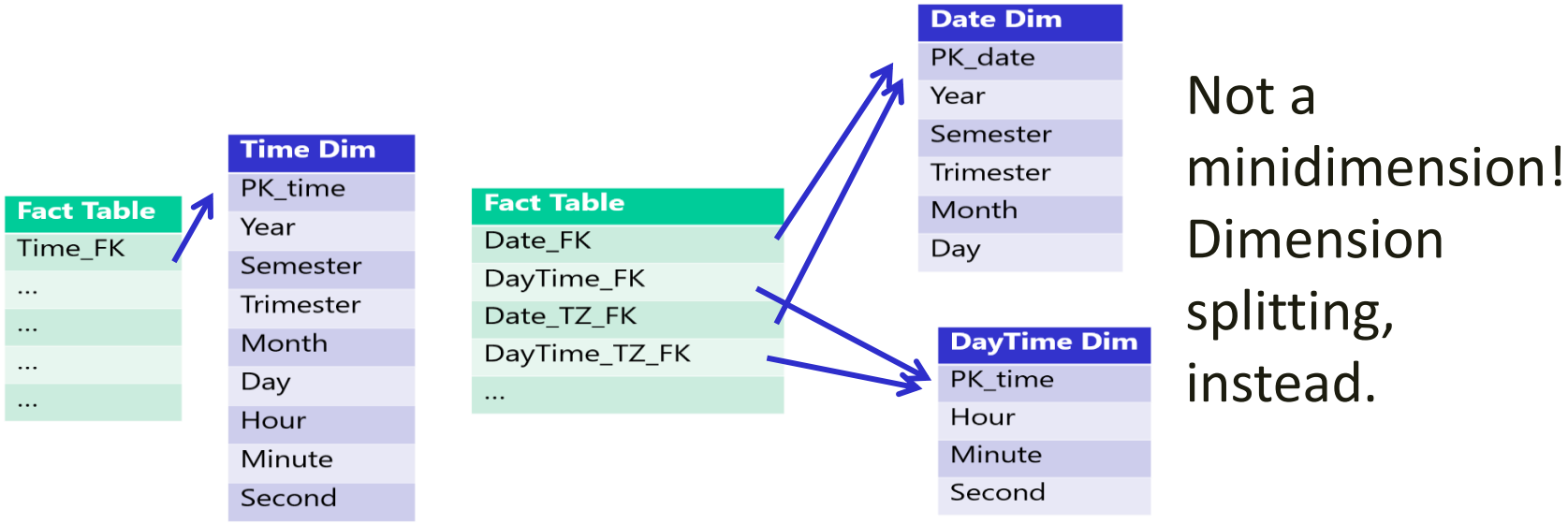
Date and time

- Track time at different granularities (e.g. day, week, hour)
- Minidimension: smaller dimensions to store frequently changed attributes.



Date and time

- Track time at different granularities (e.g. day, week, hour)
- Minidimension: smaller dimensions to store frequently changed attributes.
- Several time zones: another FK to track other TZ.
- "Role playing": several FK to same dim with different roles

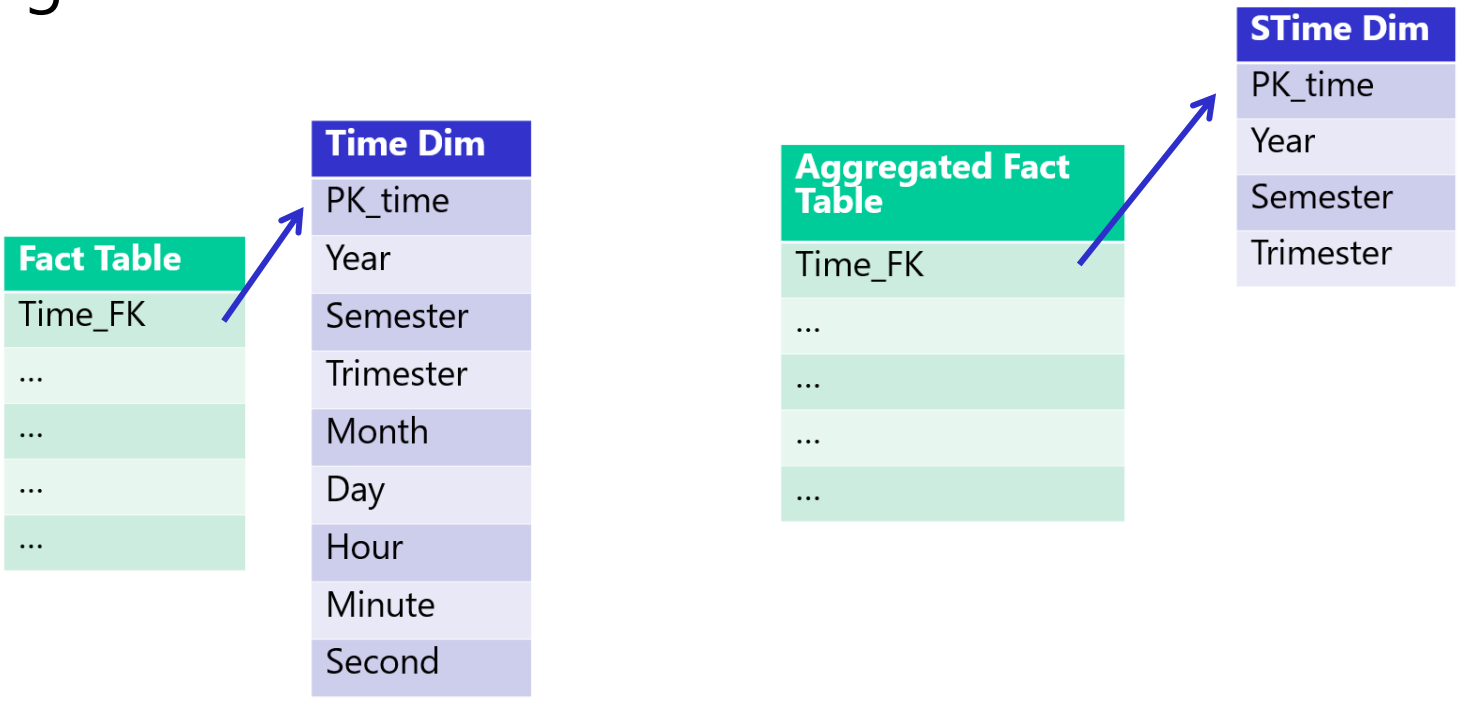


Not a minidimension! Dimension splitting, instead.

dimension agregada

(Shrunken) Rollup dimension

- For aggregatted fact tables. A smaller version of an existing dimension.
- Smaller dimension -> better performance and storage for aggregated facts.

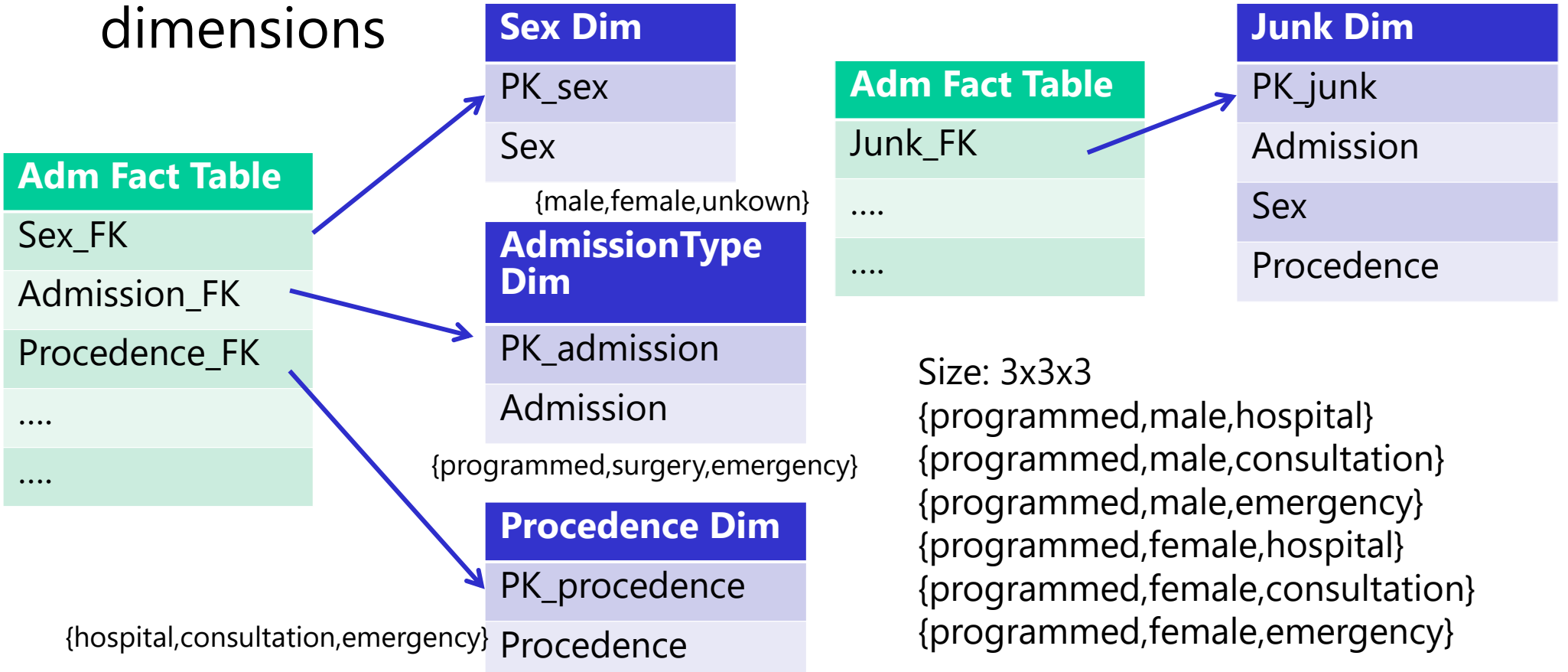


Junk dimension

- Fact tables with multiple dimensions with low cardinality

condensa dimensiones pequeñas en otra

- Junk dimension holds cartesian product of small dimensions



se usa para agrupar, para ver por ejemplo los productos asociados a una factura

Degenerate dimension

- Only contains the PK. Eg: order number, invoice number,… id, …
- Store it in the fact table (number)
- Useful for grouping
- **Not for filtering,** as there are no descriptive attributes

Adm Fact Table
Sex_FK
Admission_FK
Procedence_FK
….
Episode_number:bigint
….

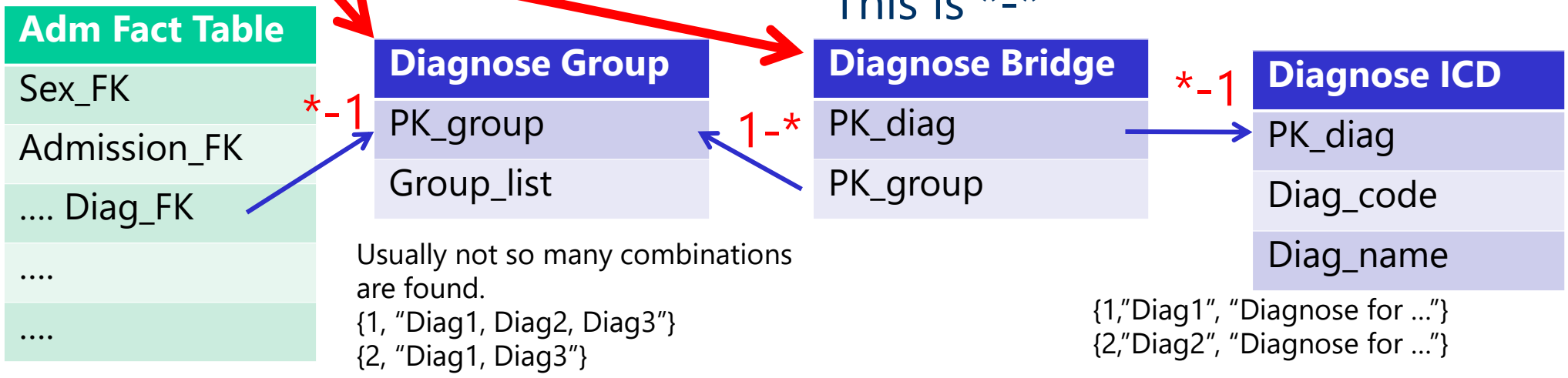
relacion entre tablas M-N

Problem M-N (*-*) associations between fact and dimensions. E.g.: admission has several diagnoses.

- Is this the right granularity? Solved in fact-tables!!!
  - Change granularity to diagnose? Another fact table?
- Standard solution: "**Bridge table**" with 2 additional tables

la mas complicada, se usa en BDs muy grandes

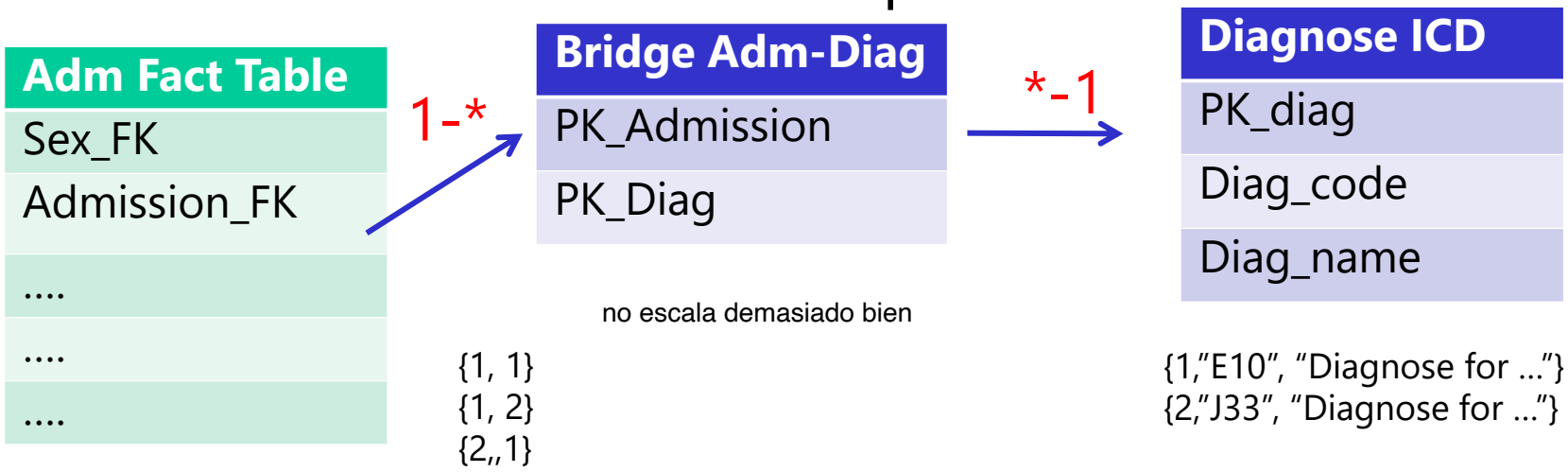
  - **Group table**: to keep association 1-* between fact and dimension**. Bridge table** between fact-table and dimension or dimension and values.



DW Design techniques

Problem M-N (*-*) associations between fact and dimensions. E.g.: admission has several diagnoses.

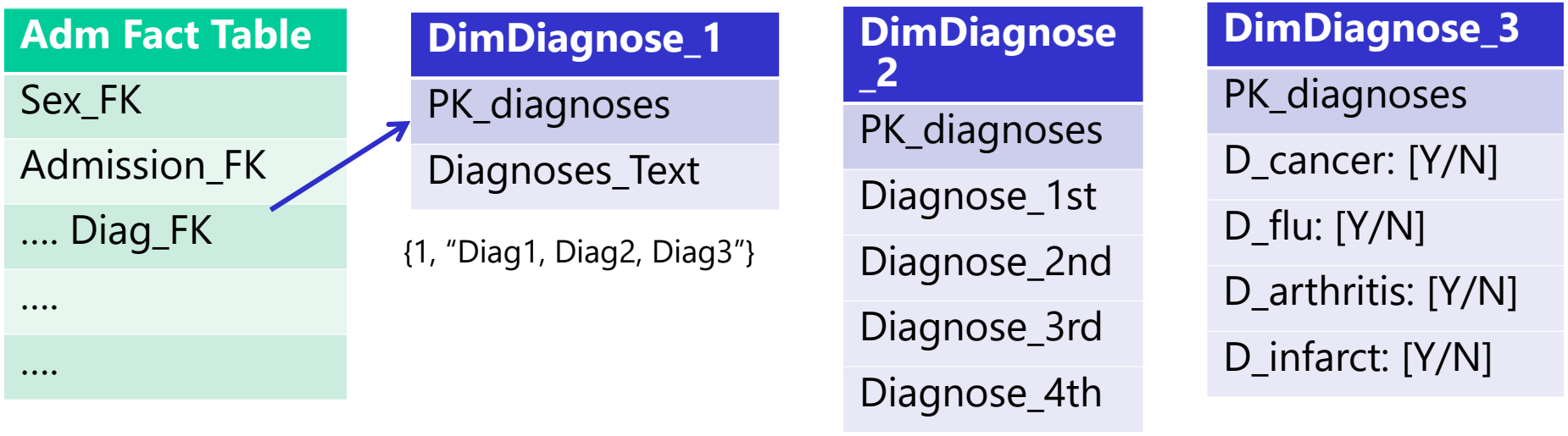
- Other alternatives? This is a simpler solution



- Simple queries, fewer tables, good for small datasets. Difficult to group diagnoses (previous option is better), adds redundancy and doesn't scale well.

Alternatives to store and processing many-to-many relationships

- String concatenation: "diag1 # diag 2" (Pathstring)
- Multiple attributes in the dimension. Eg. Diagnose1: diag1, diagnose2: diag2.
  - Are they sorted? Order is important? First value? Second value? Limited number of attributes? -> Multiple dummy attributes: Diagno



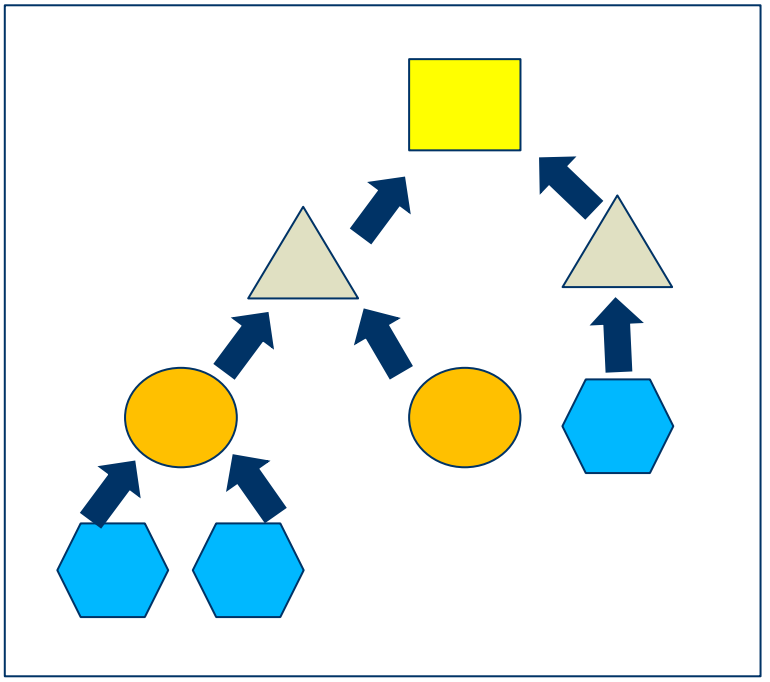
DW Design techniques

Variable depth hierarchies. The number of levels can change between records.

- Recursive queries in SQL and OLAP are limited.

Tratamiento

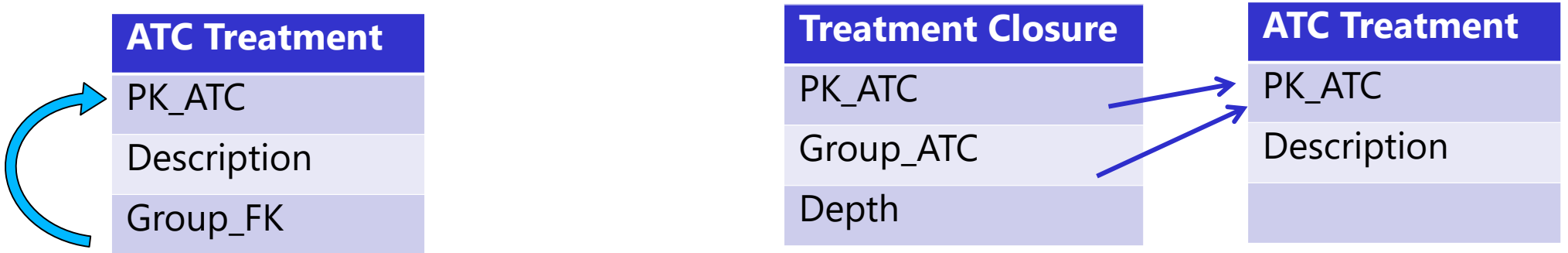
PK	ATC	descripcion	Padre
1	J01AA01	Des J01AA01	J01AA
2	J01AA02	Des J01AA02	J01AA
3	J01AA	Des J01AA	J01A
1	J01AB01	Des J01AB01	J01AB
2	J01AB02	Des J01AB02	J01AB
3	J01AB	Des J01AB	J01A
3	J01A	Des J01A	J01
3	J01B	Des J01B	J01
4	J01	Desc J01	J
5	J	Antibiotic	-



DW Design techniques

Solutions:

- Business decision: Not all levels apply: “Ceuta” and “Melilla” are not “Province”. Use business significative value
- Pathstring with complete path in hierarchy (same as with bridge tables but hierarchy flattened into a string).
- Slightly ragged: if range is small, force fix depth (same as with bridge tables)
- Bridge table with depth level (**closure table**)
  - Foreing key to dimension + depth attribute



DW Design techniques

Closure table (from previous slide)

- Bridge table: additional table.
- Foreing key to dimension + depth attribute
- Combined PK

	Fk_treatm
	1
	2

1-*

Tratamiento

PK	ATC	descripcion
1	J01AA01	Des J01AA01
2	J01AA02	Des J01AA02
4	J01AA	Des J01AA
5	J01AB01	Des J01AB01
6	J01AB02	Des J01AB02
7	J01AB	Des J01AB
8	J01A	Des J01A
9	J01B	Des J01B
10	J01	Desc J01
11	J	Antibiotic

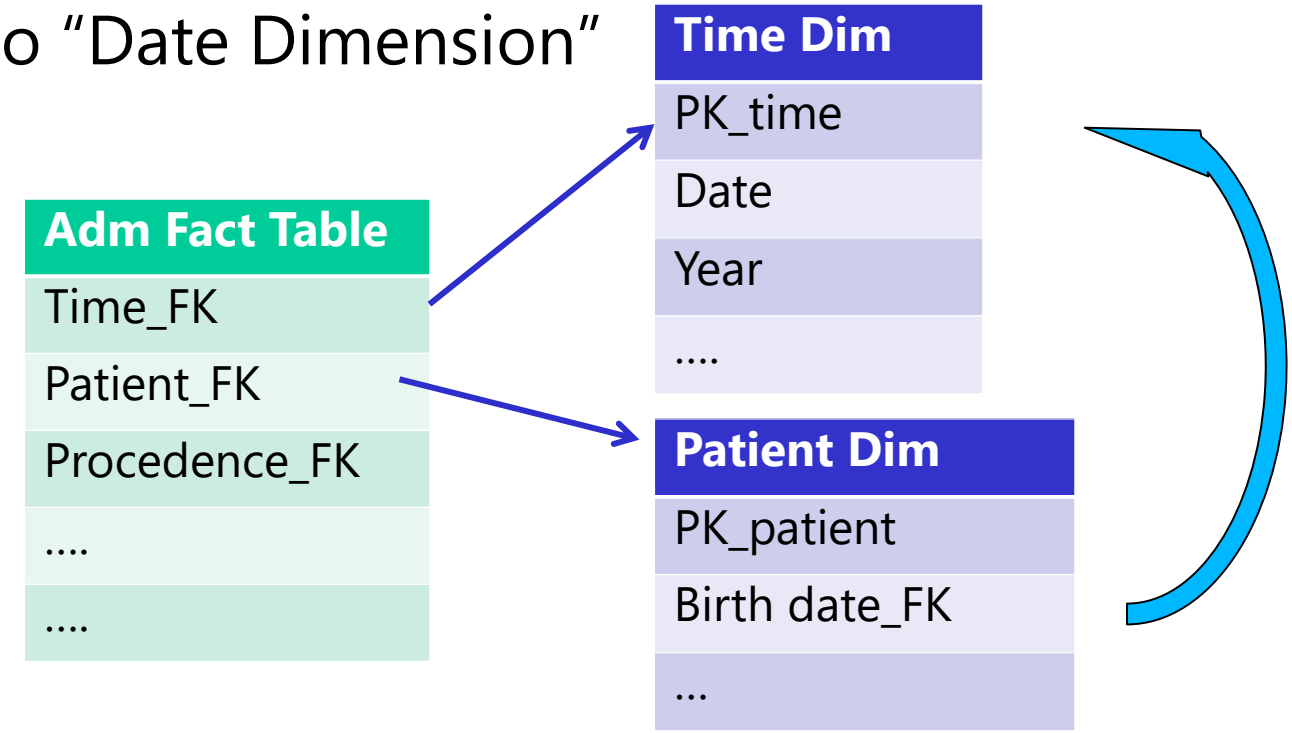
1-*

Tratamiento Closure

Hijo (PK, FK)	Padre (PK)	Profundidad
J01AA01	J01AA	4
J01AA01	J01A	3
J01AA01	J01	2
J01AA01	J	1
J01AA02	J01AA	4
J01AA02	J01A	3
J01AA02	J01	2
J01AA02	J	1
J01AB01	J01AB	4
J01AB01	J01A	3
J01AB01	J01	2
J01AB01	J	1
J01AB02	J01AB	4
J01AB02	J01A	3
J01AB02	J01	2
J01AB02	J	1

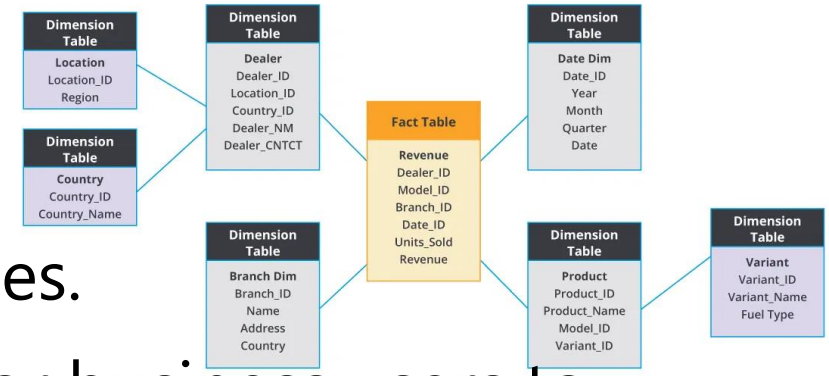
Outrigger dimension

- Exception!!
- When a dimension has a FK to another dimension.
- Eg.: “Registered user” and “Unregistered user”
  - “Birth date” links to “Date Dimension”



Snowflake dimensions

- A hierarchical relationship in a dimension table is normalized, low-cardinality attributes are stored in separate tables connected to the main dimension table through FK.
- Accurately models the hierarchy through normalization, yet a flattened (denormalized) dimension can store the same information in a single table.



- Typically used in big dimension tables.
- Snowflake schemas can be harder for business users to understand and navigate.
  - They can also negatively impact query performance, as they involve more table joins.

Normalization

DIMENSIÓN ESTRUCTURA

PK	BK	Nombre Área	Depto	Departamento	Depto ...	Facultad	Facultad Descripción	Fac ...
A1	LSI	Lenguajes y Sistemas	DIS	Informática y Sistemas	[varios]	FIUM	Facultad de Informática	[varios]
A2	ISA	Informática y Automática	DIS	Informática y Sistemas	[varios]	FIUM	Facultad de Informática	[varios]

Replicated data

DIMENSION ÁREA

PK	BK	Nombre Área	Depto
A1	LSI	Lenguajes y Sistemas	DIS
A2	ISA	Informática y Automática	DIS

FK

DIMENSION DEPTO

Depto	Departamento	Depto ...	Facultad
DIS	Informática y Sistemas	[varios]	FIUM

FK

DIMENSION FACULTAD

Facultad	Facultad Descripción	Fac ...
FIUM	Facultad de Informática	[varios]

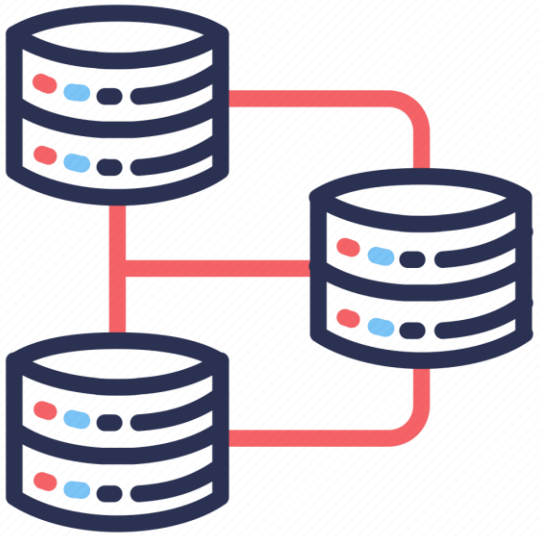
No replicated, but join needed

Normalization

Normalization is a process that organizes database tables by **removing anomalies and improving data consistency**. It helps prevent:

- **Update** anomalies
- **Inconsistent** Data
- **Addition** anomalies
- **Deletion** anomalies

ver ejemplos si eso



All **attributes depend on the key**, the whole key and nothing but the key.

- **1NF**: Keys exist and no repeating groups.
- **2NF**: No partial dependencies.
- **3NF**: No transitive dependencies.

1st Normal Form

distinguimos tres nivles

1NF

Table has a **primary key**

No **repeating groups or multivalued attributes**

A **multivalued attribute** is an attribute that may have several values for one record (e.g. list of courses for a teacher)

ID	Name	Courses
1	Pepito	BI, AMD, BDII

A **repeating group** is a set of one or more multivalued attributes that are related.

ID	Name	Course1	Course2	Course3
1	Pepito	BI	AMD	BDII

2nd Normal Form

2NF

Already in 1NF. No partial dependencies

Every non-key attribute must **depend on the full concatenated** key, not just part of it.

T_ID	C_ID	T_Name	C_Name
T1	C1	Pepito	AMD
T1	C2	Pepito	BDII

T_name depends only on T_ID and C_Name on C_ID. How can we solve it?

no se encuentra en la segunda forma normal

C_ID	C_Name
C1	AMD
C2	BDII

T_ID	C_Name
T1	Pepito

T_ID	C_ID
T1	C1
T1	C2

3rd Normal Form

en relacional se suele usar esta

3NF

No transitive dependencies

A table in in **3NF** is it is **2NF** and all **non-key attributes** depend only on the **PK.**

Course	Area_Dpt	Teacher
AMD	LSI	Pepito
BDII	ARQ	Manolito

The candidate Key is Course, but Teacher depends on Area_Dpt.

Course	Teacher
AMD	Pepito
BDII	Manolito

Teacher	Area_Dpt
Pepito	LSI
Manolito	ARQ

Business intelligence

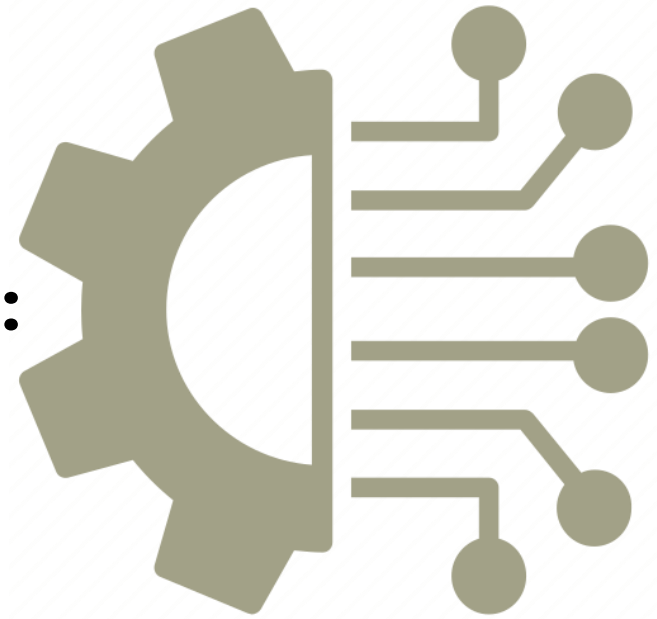
Unit 2 – Datawarehouse

S2-3 – Data integration - ETL

Data integration

Data come from different systems.

- Need for integration.
- Very different source systems



Data stored in at least in two places:

- The source system.
- The datawarehouse.

DWH features

- Exchange, integrate and consolidate data from many systems.
- It provides a new unified database.
- Volume tends to be very large.

Data integration - ETL

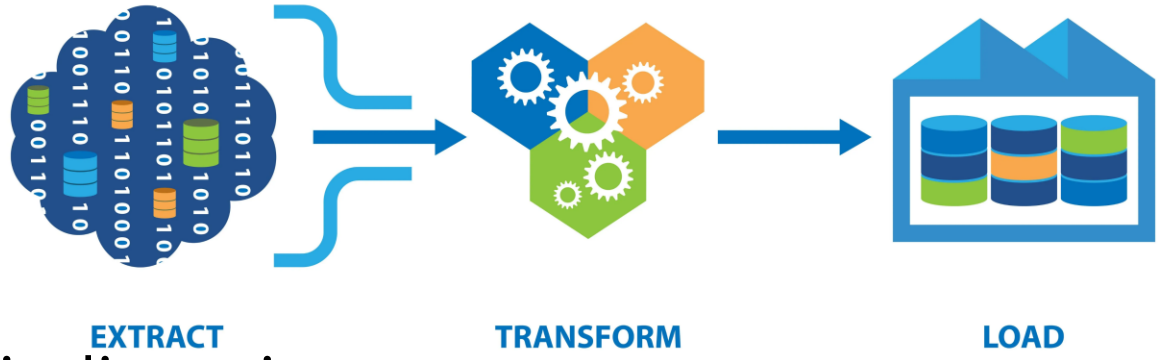
The maintenance of the data warehouse is done by means of the **ETL** (**Extraction** - **Transformation** - **Load**) process

- is **the responsibility of the team** developing the data warehouse.
- is **built specifically** for each DW.
- you can use **market tool**s or **programs designed specifically. Example:** ETL tools: Oracle Data Integrator, Informatica Power Center, Talend, Stitch, Pentaho, AWS Glue, Alooma, Hevo Data, SSIS (SQL Server Integration Services)
- It is very **time consuming** when building our DW/BI solution.

Importance of ETL

General steps:

- Initial load
- Refreshment: daily, weekly, monthly, … periodic maintenance



It **adds value** to data

- **Metadata**: document data quality
- Captures **data flows** (processes)
- Increases **quality**: less mistakes
- **Conformed** data between departments

ETL description

- The data of one or more operating systems **needs to be accessible** in the DWH.
- ETL is the process of **extracting data from source systems** and **bringing them to DWH**
  - Load DWH **regularly**.
- It consists of several steps
  - **Extraction**: Pulling data from the sources.
  - **Cleaning**: Ensuring data is accurate, remove duplicates, …
  - **Transformation**: Converting the data to match DWH.
  - **Load**: Inserting the transformed data into the DWH.

Business Requirements

- Definition: The business needs determine the selection of data sources and their transformation within ETL.
- Objective: Align ETL processes with business goals for optimal decision-making.

Compliance Requirements

- Regulatory Compliance: Ensure that legal and regulatory data requirements are met (e.g., telecommunications).
- Chain of Custody: Maintain data custody and adhere to legal frameworks for all data and reporting.

Data Integration Requirements

- Key Dimensions and Metrics: Identify common dimensions and attributes across processes to develop standardized KPIs and metrics.
- Objective: Ensure all systems can work together seamlessly with consistent data granularity and units.

Latency Requirements

- Data Availability: Determine the speed at which data must be made available to users.
- Impact: Latency directly influences ETL architecture design and performance.

Archiving and Lineage Requirements

- Archiving Policies: Define policies for data archiving, including the number of copies and retention strategy.
- Data Lineage: Use metadata to track the origin, changes, and quality of data, ensuring transparency in data handling.

Available Skills and Licensing Requirements

- Skills Assessment: Implement ETL based on the available expertise in the organization.
- Legacy Systems: Consider any restrictions imposed by legacy systems and their licensing agreements. It is good to differentiate between when the use of legacy systems is mandatory versus when it is merely recommended.

Data Quality Requirements

- Problem Identification: Identify elements with data quality issues and assess if operational systems can be modified for improvement.
- Monitoring: Include a system for ongoing quality tracking.

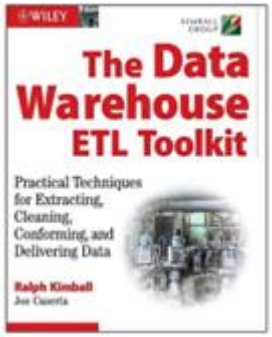
Security Requirements

- Data Sensitivity: Assess the level of sensitivity for all data processed through ETL.
- Security Measures: Security extends to backups and must comply with compliance requirements.

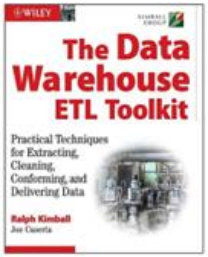
Business Intelligence Interface Requirements

- Simple and Reliable Data Access: Ensure data reaches BI applications quickly, reliably, and simply.
- Collaboration: Identify facts and dimensions to expose to BI tools, working closely with data architects and application developers.

ETL: Subsistemas (Kimball)



- ETL
  - Extraer
    1. Calidad de Datos-Perfilado de Datos
    2. Captura de los cambios en los datos
    3. Sistema de Extracción
  - Limpieza y Conformidad
    4. Limpieza de datos
    5. Esquema de eventos de error
    6. Añadir dimension de auditoría
    7. Sistema de deduplicación
    8. Sistema de conformidad
  - Entrega
    9. Gestor de las dimensiones lentamente cambiantes (SCD Manager)
    10. Generador de claves subrrogadas
    11. Gestor de jerarquías
    12. Gestor de dimensiones especiales
    13. Constructores de tablas de hechos
    14. Línea de proceso de las claves subrrogadas
    15. Constructor de la tabla puente para dimensiones multivaluadas
    16. Gestor de datos retrasados
    17. Sistema de gestión de dimensiones
    18. Sistema de provisión de tablas de hechos
    19. Constructor de agregaciones
    20. Constructor de cubos multidimensionales
    21. Gestor de propagación de datos
  - Gestión
    22. Programador de trabajos (Job Scheduler)
    23. Sistema de backup
    24. Sistema de recuperación y reinicio
    25. Sistema de control de versiones
    26. Sistema de migración de versiones
    27. Monitor del flujo de trabajo
    28. Sistema de ordenación (sorting)
    29. Analizador de linaje y dependencias
    30. Sistema de escalado de problemas
    31. Sistema de paralelización/canalización (pipelining)
    32. Sistema de seguridad
    33. Gestor de conformidad
    34. Gestor del repositorio de metadatos

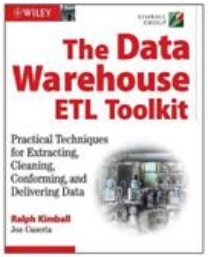


Comp 1: Extraer

- **Subsist 1**: **Perfilado de datos**: Analizamos los datos para conocer su contenido, consistencia y estructura. Nos permite hacernos una idea de los hechos y dimensiones, y del mapeado a implementar.
- **Subsist 2**: **Sist de captura de los cambios en los datos**: Podemos cargar todos los datos de las fuentes, pero con los volúmenes de datos de hoy en día eso en ocasiones no es factible. Debemos tener la capacidad de solo transferir aquellos cambios desde la última extracción. Técnicas comunes implican chequear los logs de transacciones, detectar cambios con CRC/Hash, por el timestamp un record ha sido creado/modificado, etc.

- **Subsist 3**: **Sistema de Extracción**: Relacionado con la obtención de datos de los sistemas fuente. Cada una de ellas podría tener estar en un entorno/BBDD diferente. Podemos obtener los datos en forma de ficheros (XML, csv, txt,…) o a través de un flujo de datos que podemos iniciar a través de una consulta, acceso a un servicio web, etc. Con un fichero no necesitamos re-consultar la fuente en caso de fallo. Aquí podemos decidir comprimir los datos para acelerar la transmisión. Debemos evaluar la necesidad de encriptar la información en tránsito.

ETL: Comp 2: Conformidad

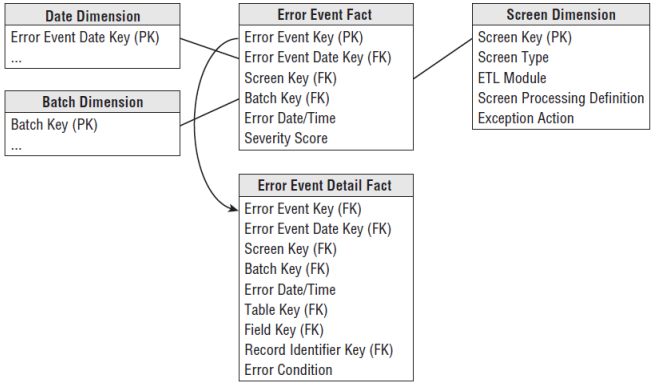


Comp 2: Limpieza y conformidad de datos (la T de ETL)

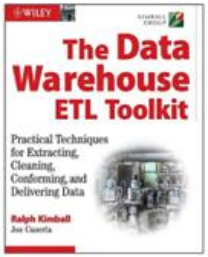
- **Subsist 4**: **Limpieza de datos**: Desarrollar una arquitectura para diagnosticar problemas, determinar métricas sobre la calidad de los datos, y responder a problemas (ej. resolviéndolo, reportándolo, abortando, continuando,...). Los procesos de filtrado se implementan en el pipeline de datos en la forma de chequeos de calidad, que pueden ser: de columnas (datos dentro de una misma columna), estructurales (relaciones entre los datos en distintas columnas) y de las reglas de negocio (ej. un cliente VIP ha realizado unas compras por encima de un umbral).
- **Subsist 5**: **Esquema de eventos de error**: Un esquema dimensional centralizado para almacenar los eventos de los chequeos de calidad. Cada error produce una entrada en Error Event Fact

Los procesos de filtrado son los responsables de añadir records a estas tablas.

Un error que añade una fila e Error Event Fact podría generar muchas entradas en la Error Event Detail Fact



ETL: Comp 2: Conformidad

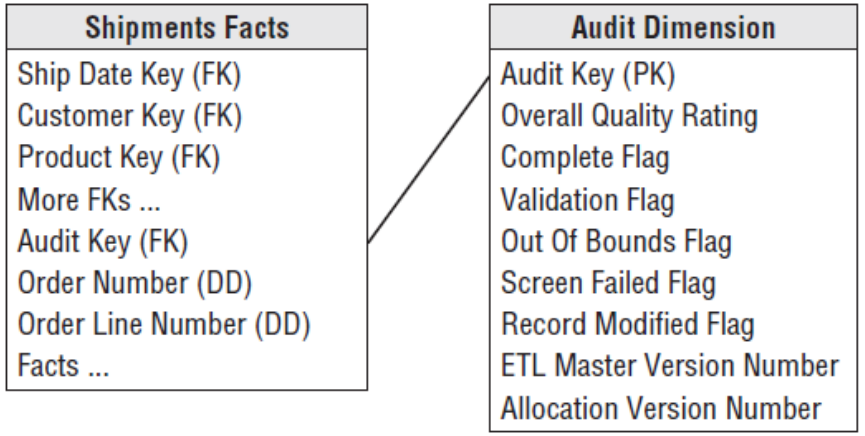


Comp 2: Limpieza y conformidad de datos (la T de ETL)

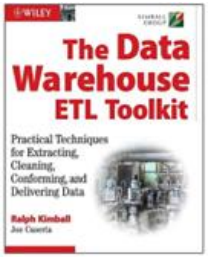
- **Subsist 6**: **Añadir dimension de auditoría**: Dimensión añadida por el sistems ETL con el propósito de evaluar los resultados de los procesos de filtrado. Si todo va bien, solo se genera una fila en la tabla Audit Dimension, y la correspondiente FK será añadida a la tabla de hechos. Pero si tenemos condiciones de error, necesitaríamos más entradas en la tabla de dimensiones.

- **Subsist 7**: **Sistema de deduplicación**

En muchas ocasiones las dimensiones proceden de varias fuentes. Necesitamos combinarlas para crear una imagen que combine las columnas de más alta calidad para cada fila (proceso de supervivencia). Este proceso incluye reglas claras de negocio para decidir la prioridad de las diferentes fuentes a la hora de construir cada fila.

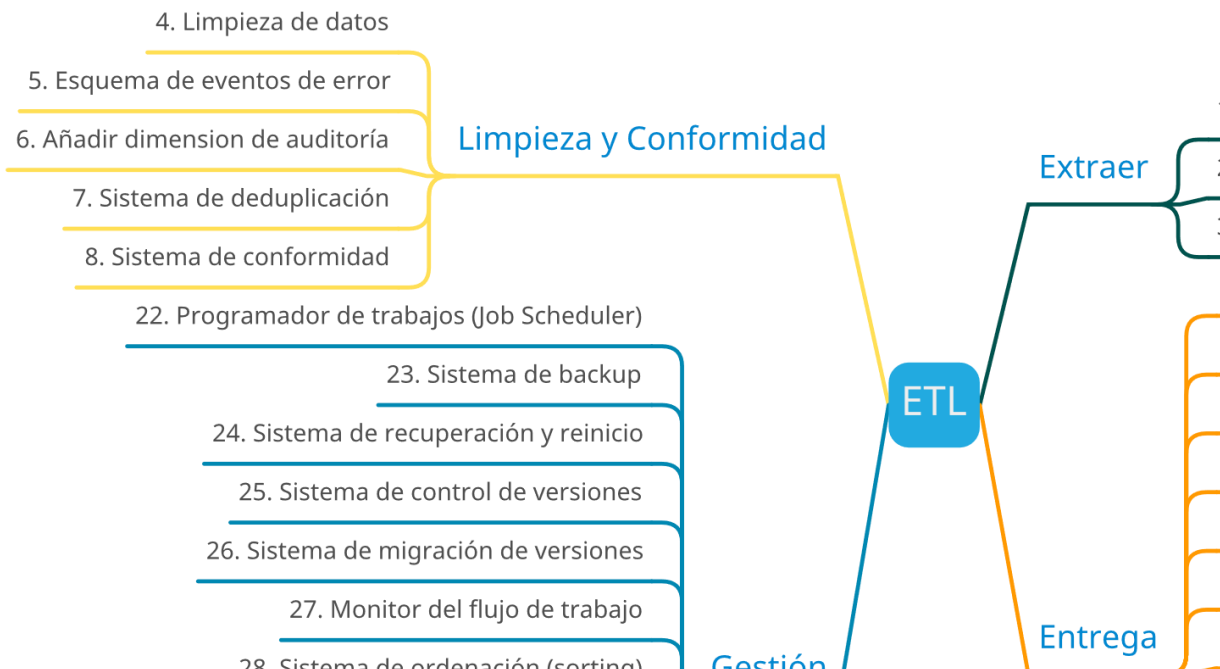


ETL: Comp 2: Conformidad

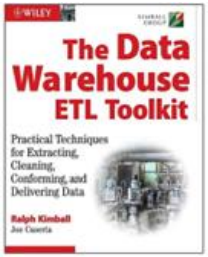


Comp 2: Limpieza y conformidad de datos (la T de ETL)

- **Subsist 8**: **Sistema de conformidad**: Encargado de definir y crear dimensiones y hechos estructuralmente idénticos a partir de la combinación e integración de datos procedentes de diversos sistemas que se han deduplicado, filtrados para eliminar datos inválidos y estandarizados.

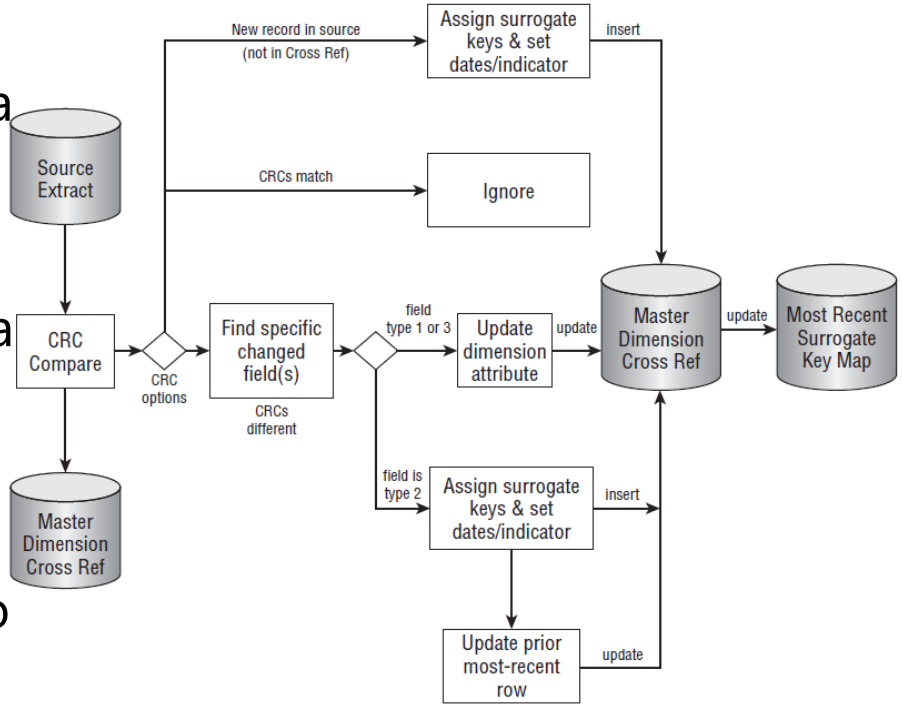


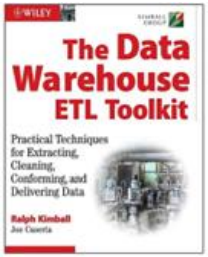
ETL: Comp 3: Entrega



- **Comp 3: Entrega** (la L de ETL)
  - **Subsist 9: Gestor de las dimensiones lentamente cambiantes** (SCD Manager): El sistema ETL tiene que gestionar el valor de un atributo que ha cambiado respecto al valor almacenado en el DW. Las respuestas típicas son: sobreescribir (tipo 1), añadir una nueva fila (tipo 2), y añadir una nueva columna (tipo 3)

- **Tipo 1-** Cambiamos la dirección de un cliente. Solo actualizamos la fila sin tocar las claves de la tabla de dimensión ni la de la tabla de hechos.
- **Tipo 2-** Ahora imaginemos que queremos mantener el historial de cambios y, por tanto, al cambiar la dirección, necesitamos añadir una nueva fila. Seguimos sin tocar la tabla de hechos, pero usaremos ahora una nueva FK para ese cliente.
- **Tipo 3-** Añadimos una columna nueva para almacenar los cambios. Por ejemplo, en el caso anterior, añadimos una nueva columna para incluir la dirección vieja.



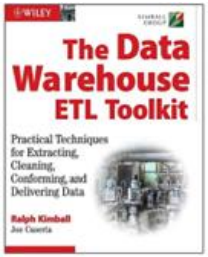


Comp 3: Entrega (la L de ETL)

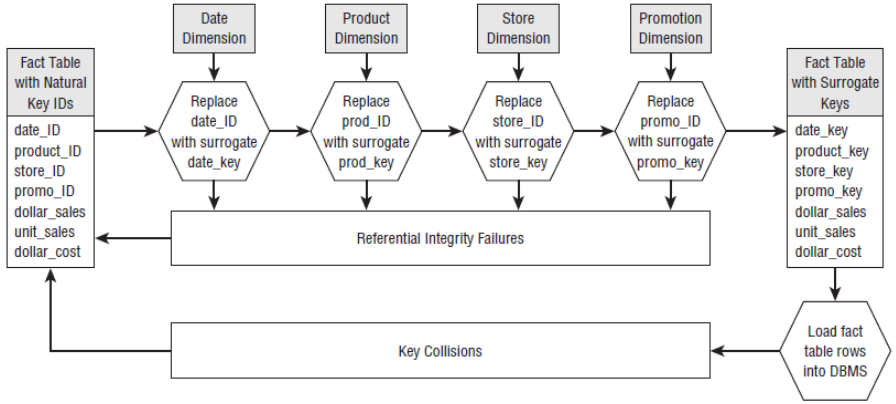
- **Subsist 10**:**Generador de claves subrrogadas**: Se recomienda usar este tipo de claves como PK de las tablas de dimensiones. Necesitamos un mecanismo robusto para generarlas en entornos ETL. En PostgreSQL, una secuencia valdría.
- **Subsist 11**: **Gestor de jerarquías**: Es normal encontrarnos con jerarquías en las dimensiones. Hay 2 tipos:
  - **Fijas**: número estable de niveles que se modelan como atributos de la dimensión. Ej: producto->fabricante
  - **Desiguales (ragged)**: número variable de niveles. Podemos recurrir a un modelo copo de nieve para modelarlas. A veces , las direcciones postales son desiguales

- **Subsist 12**: **Gestor de dimensiones especiales**: Apoya las características de diseño de las dimensiones para la organización, pudiendo abarcar todas o algunas de:
- Dimensiones de tiempo y fecha
- Dimensiones basura (junk)
- Dimensiones "encogidas" (shrunken)- subconjuntos de dimensiones mayores
- Dimensiones pequeñas estáticas- normalmente tablas lookup sin fuente de referencia

ETL: Comp 3: Entrega



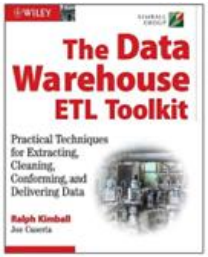
- **Comp 3: Entrega** (la L de ETL)
  - **Subsist 13:Constructores de tablas de hechos**: Enfocado a la construcción de los diferentes tipos de tablas de hechos:
    - **Transaccionales:** Se cargan cuando la transacción ocurre o en intervalos a la máxima granuralidad . Se cargan los datos y, a veces, si se requiere actualizar datos ya almacenados, primero se insertan y después de borran los viejos.
    - **Instantáneas (snapshot) periódicas:** Se cargan a intervalos regulares y representan períodos. Ej: balances mensuales de un banco.
    - **Instantáneas acumuladas:** para representar eventos finitos que evolucionan con un inicio y final bien definidos. Ej: procesar un pedido: tenemos primero la fecha de pedido, pero no sabemos la de envío, recogida,... A medida que las sabemos tenemos que ir actualizándolas.
  - **Subsist 14: Línea de proceso de las claves subrrogadas**: Reemplazar las claves operacionales naturales del registro de la tabla de hechos con las subrrogadas de la dimensión.



ETL: Comp 3: Entrega

- **Comp 3: Entrega** (la L de ETL)
  - **Subsist 15: Constructor de la tabla puente para dimensiones multivaluadas**: Soporta relaciones Many-to-Many entre hechos y dimensiones. Ej: hecho: compra (cantidad, valor); dimensión: razones de la compra. Otras: pacientes/diagnósticos, clases/profesores,…
  - **Subsist 16: Gestor de datos retrasados**: En realidad, no todos los datos llegan al mismo tiempo. Ej: podemos tener datos de ventas diarios y datos de personal mensuales. El sistema ETL debe solucionar estas situaciones en las que hechos y dimensiones pueden llegar tarde para mantener la integridad referencial. Una posible solución podría ser asignar un valor por defecto para la dimensión hasta que se conoce ésta. Ej: Nombre cliente: TBD
  - **Subsist 17: Sistema de gestión de dimensiones**: Autoridad centralizada que prepara y publica las dimensiones conformadas (dimensiones cuya semántica, estructura y uso conviene a toda la empresa) a la comunidad del DW. Entre sus responsabilidades:
    - Implementar etiquetas descriptivas para los atributos
    - Gestionar los cambios en los atributos
    - Añadir nuevas filas a la dimensión conformada, generando nuevas claves subrrogadas
    - Distribuir las actualizaciones dimensionales

ETL: Comp 3: Entrega

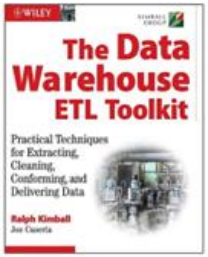


Comp 3: Entrega (la L de ETL)

- **Subsist 18: Sistema de provisión de tablas de hechos**: Administra una o varias tablas de hechos, siendo responsable de su creación, mantenimiento y uso. Responsabilidades:
- Recibe las dimensiones replicadas del gestor de dimensiones.
- Añade y actualiza las filas de las tablas de hechos
- Asegura la calidad de los hechos
- Recalcula los agregados
- Notifica a los usuarios de cambios, actualizaciones y problemas.

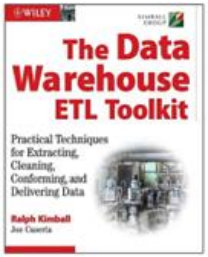
- **Subsist 19: Constructor de agregaciones**: Creación y mantenimiento de estructuras físicas agregadas en la BBDD que ayudan a mejorar el rendimiento. Debemos controlar cuidadosamente cuales necesitamos, sin quedarnos cortos ni pasarnos. Los hechos/dimensiones sumario se crean a partir de los hechos/dimensiones base.
- **Subsist 20: Constructor de cubos multidimensionales**: Crear y mantener los esquemas ROLAP estrella que permiten crear los cubos. Los cubos MOLAP presentan datos multidimensionales fáciles de explorar. Los cubos deben actualizarse si los hechos o dimensiones cambian.

ETL: Comp 3: Entrega



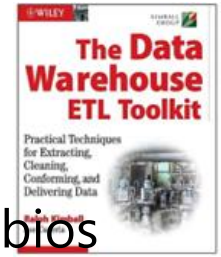
- **Comp 3: Entrega** (la L de ETL)
  - **Subsist 21: Gestor de propagación de datos**: Se encarga de mover datos del DW a otros entornos, como sistemas de minería de datos, o de auditoría.

ETL: Comp 4: Gestión Entorno



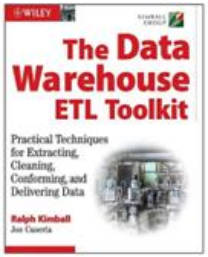
Comp 4: Gestionar el entorno ETL

- El entorno ETL debe ser fiable, disponible según los SLA, y adaptable según la evolución del negocio. Para ellos necesitamos los siguientes subsistemas:
  - **Subsist 22: Programador de trabajos** (Job Scheduler): Sistema para programar y lanzar los trabajos ETL. Debe conocer y controlar las dependencias entre trabajos ETL. Responsable de definir trabajos, programarlos, capturar metadatos y volcarlos en ficheros log, así como notificar problemas.
  - **Subsist 23: Sistema de backup**: Almacenar, archivar y extraer elementos del sistema ETL.
  - **Subsist 24: Sistema de recuperación y reinicio**: Los trabajos deben recuperarse en caso de errores y tener la capacidad de reiniciarse.
  - **Subsist 25: Sistema de control de versiones**: Almacenamos información sobre las diferentes versiones (scripts, SQL, Jobs,…) en el flujo de procesos de ETL: Git, SVN, …



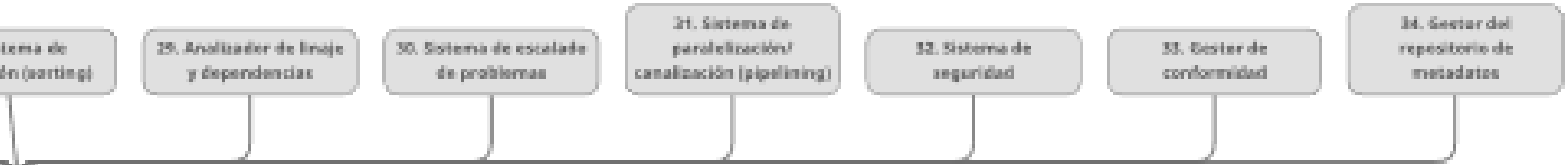
Comp 4: Gestionar el entorno ETL

- **Subsist 26: Sistema de migración de versiones**: Transferir los cambios desde la etapa de desarrollo, a testeo, y después a producción
- **Subsist 27: Monitor del flujo de trabajo**: Panel de control y sistema de reporte de todos los trabajos iniciados por el programador de trabajos. Incluye un sistema de auditoría, los logs del ETL y la monitorización de la BBDD.
- **Subsist 28: Sistema de ordenación** (sorting): Ordenar es una capacidad muy importante en ETL, usada muy a menudo en agregaciones y joins.Es importante que se pueda aplicar de forma rápida y eficiente.
- **Subsist 29: Analizador de linaje y dependencias**: (linaje) ->Representa las fuentes físicas y las transformaciones de cualquier elemento de datos, en cualquier etapa del proceso ETL. (dependencia) ->Representar todos los elementos de datos "downstream" y reportes finales que se verían afectados por un cambio en un elemento "upstream". Ej dependencia: identificar los cubos y esquemas estrella que usan un elemento de la fuente.
- **Subsist 30: Sistema de escalado de problemas**: Sistema automático/manual donde una condición de error se eleva al nivel correspondiente para analizarlo y resolverlo. Va desde simples entradas en los logs, hasta a notificar a los operadores, supervisor o desarrollador.



Comp 4: Gestionar el entorno ETL

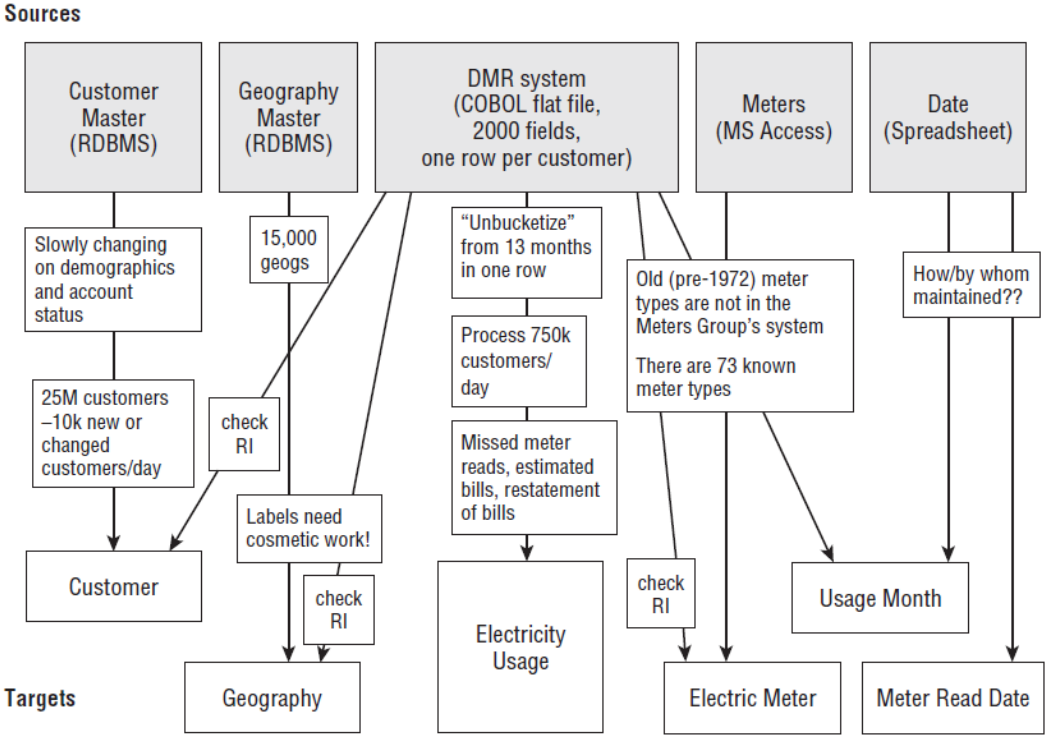
- **Subsist 31: Sistema de paralelización/canalización** (pipelining): Normalmente las restricciones temporales nos obligan a aprovecharnos de la disponibilidad de recursos computacionales para aquellas tareas que se pueden aprovechar de ello.
- **Subsist 32: Sistema de seguridad**: Gestionar la seguridad a nivel individual y rol de todos los datos y metadatos en ETL. Esto también incluye el sistema de backup.
- **Subsist 33: Gestor de conformidad**: Mantener la cadena de custodia de los datos en entornos regulados (ej: bancos, salud,…) y registrar quién ha accedido de forma autorizada a los datos. Los cambios en los datos deben de reportarse.
- **Subsist 34: Gestor del repositorio de metadatos**: Administra todos los metadatos de ETL. Esto incluye la captura y mantenimiento, incluyendo toda la lógica de las transformaciones. Incluye metadatos de procesos, técnicos y de negocio.



Planificar e implementar ETL

- **Pasos preliminares (Kimball):**
  - Diseño lógico
  - Comprender la arquitectura DW/BI

- **Etapas planificación ETL:**
  1. **Esbozar un plan de alto nivel**: Identificar fuentes y destinos (dimensiones y tablas hecho)



ETL PROCESS STEP	ETL SUBSYSTEM: EXTRACTING DATA	CLEANING AND CONFORMING	DELIVERING FOR PRESENTATION	MANAGING THE ETL ENVIRONMENT
Plan				
Create a high level, one-page schematic of the source-to-target flow.	1			
Test, choose, and implement an ETL tool (Chapter 5).				
Develop default strategies for dimension management, error handling, and other processes.	3	4, 5, 6	10	
Drill down by target table, graphically sketching any complex data restructuring or transformations, and develop preliminary job sequencing.		4, 5, 6	11	22
Develop One-Time Historic Load Process				
Build and test the historic dimension table loads.	3	4, 7, 8	9, 10, 11, 12, 15	
Build and test the historic fact table loads, including surrogate key lookup and substitution.	3	4, 5, 8	13, 14	
Develop Incremental Load Process				
Build and test the dimension table incremental load processes.	2, 3	4, 7, 8	9, 10, 11, 12, 15, 16, 17	
Build and test the fact table incremental load processes	2, 3	4, 5, 8	13, 14, 16, 18	
Build and test aggregate table loads and/or OLAP processing.			19, 20	
Design, build, and test the ETL system automation.		6	17, 18, 21	22, 23, 24, 30

Planificar e implementar ETL

Etapas planificación ETL:

2. **Seleccionar una herramienta ETL**: Existen muchas opciones que pueden leer de diversas fuentes. Podemos desarrollarla o bien emplear herramientas GUI.
3. **Desarrollar estrategias** para las actividades más comunes en el sistema ETL. Incluye:
   - Extracción de datos y archivado
   - Evaluar calidad de dimensiones y hechos
   - Administrar cambios en atributos de dimensiones
   - Asegurar cumplimiento requisitos de disponibilidad del DW y ETL
   - Diseñar sistema de auditoría de datos
   - Organizar el almacenamiento de datos intermedio (staging area) para almacenamiento temporal.

Planificar e implementar ETL

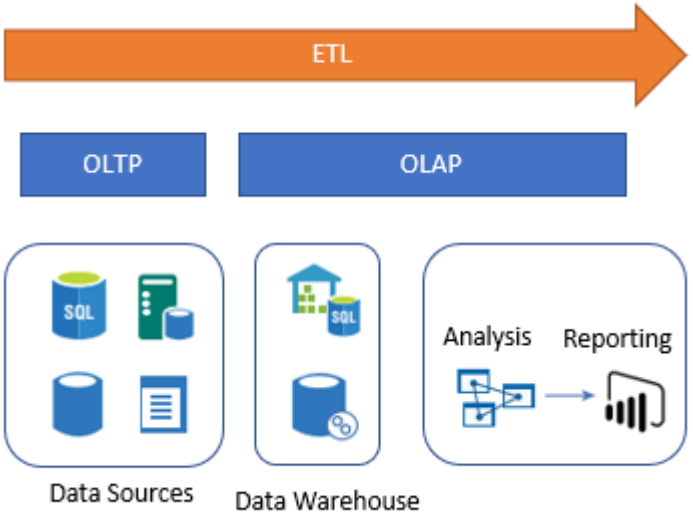
- **Etapas planificación ETL:**

4. **Mapear de forma detallada** el flujo de datos desde las fuentes a las tablas de dimensiones y hechos. Podemos hacer uso de diagramas de flujo o esquemas que nos ayuden a realizar esta labor.
5. **Cargar las tablas de dimensiones con datos históricos**: Este es un proceso que hacemos una vez y que difiere de la carga incremental. Normalmente comenzamos con las tablas más simples (ej: tipo 1). Después podemos transformar los datos : convertir tipos, combinarlos de fuentes diferentes, asignarles claves subrrogadas, etc.
6. **Cargar la tabla de hechos con datos históricos**: Podemos realizar transformaciones: sustituir valores numéricos "especiales" por NULL en hechos aditivos o semiaditivos, calcular hechos complejos, añadir una clave de auditoría, etc.
7. **Procesado incremental de la tabla de dimensiones**: A veces podemos usar la misma lógica que en la carga de datos históricos. Un reto es identificar los datos nuevos/actualizados.

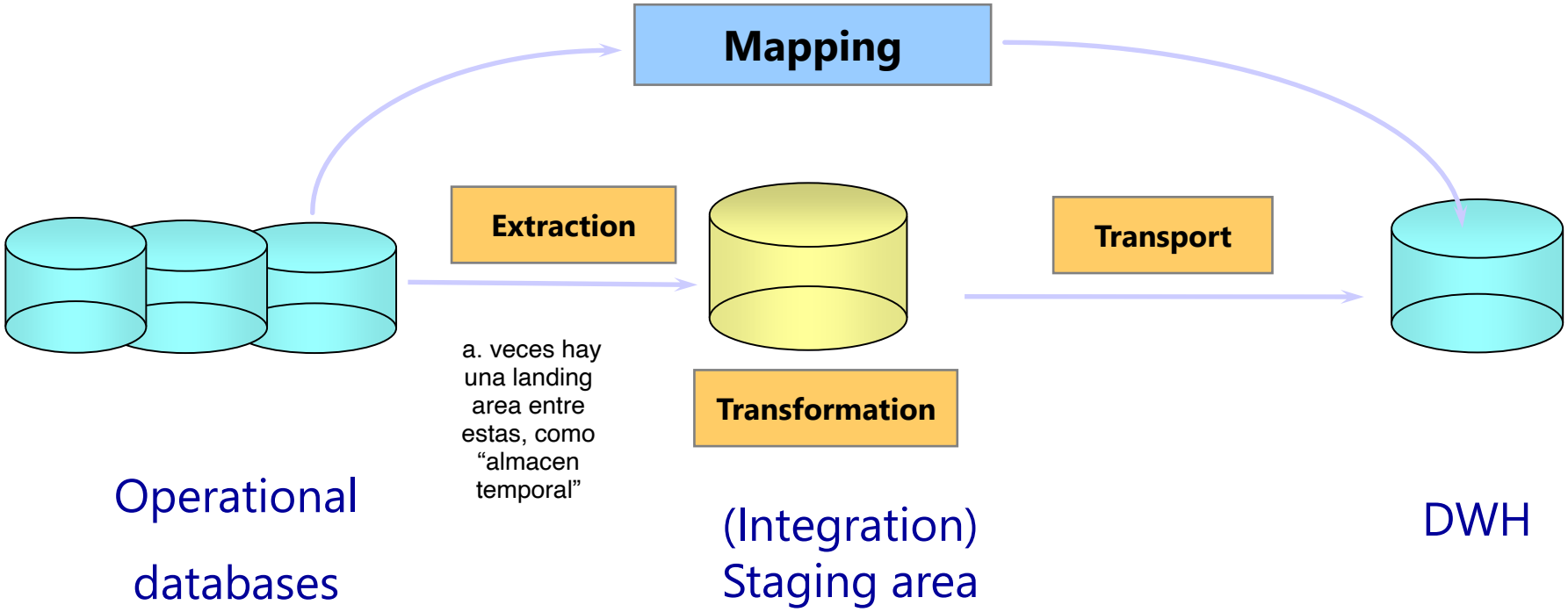
Planificar e implementar ETL

- **Etapas planificación ETL:**

8. **Procesado incremental de la tabla de hechos**: Esta fase normalmente requiere automatizar el proceso y puede requerir procesado en paralelo y chequeo de la calidad de datos.
9. **Agregar carga de tablas y OLAP**: A veces necesitamos procesado adicionar más allá del esquema estrella. Necesitamos actualizar MOLAP, las vistas (incluyendo las materializadas) y las tablas sumario con agregaciones.
10. **Operación del sistema ETL y automatización**: en esta etapa necesitamos programar los trabajos, gestionar los errores/excepciones, operaciones sobre las BBDD(actualizar las estadísticas, recrear los índices, limpiar datos,…)



ETL



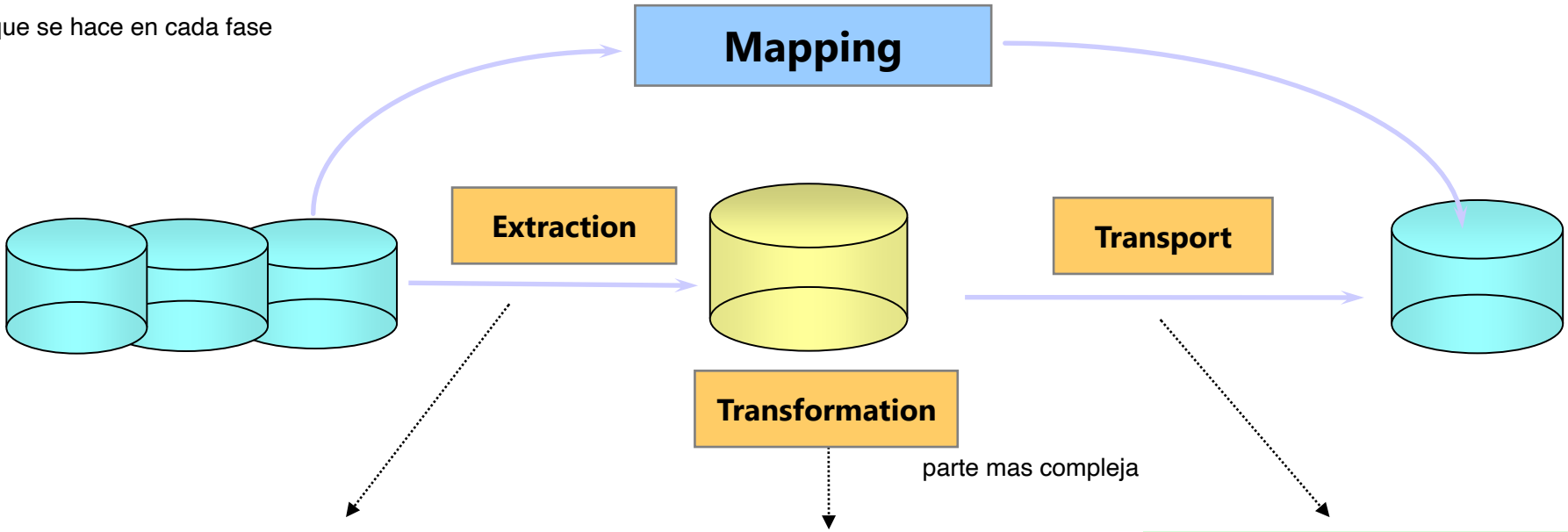
The **Staging** area:

- allows transformations without paralyzing the operational databases or the datawarehouse.
- Allows storing metadata.
- Facilitates the integration of external sources.
- Not necessarily stores data in Relational model

ETL

quality test, ver por ejemplo que la edad esta entre 0 y 130

que se hace en cada fase



parte mas compleja

- ✓Identifying data that has changed
- ✓Extraction (read) data.
- ✓Aggregates definition
- ✓Maintenance of metadata

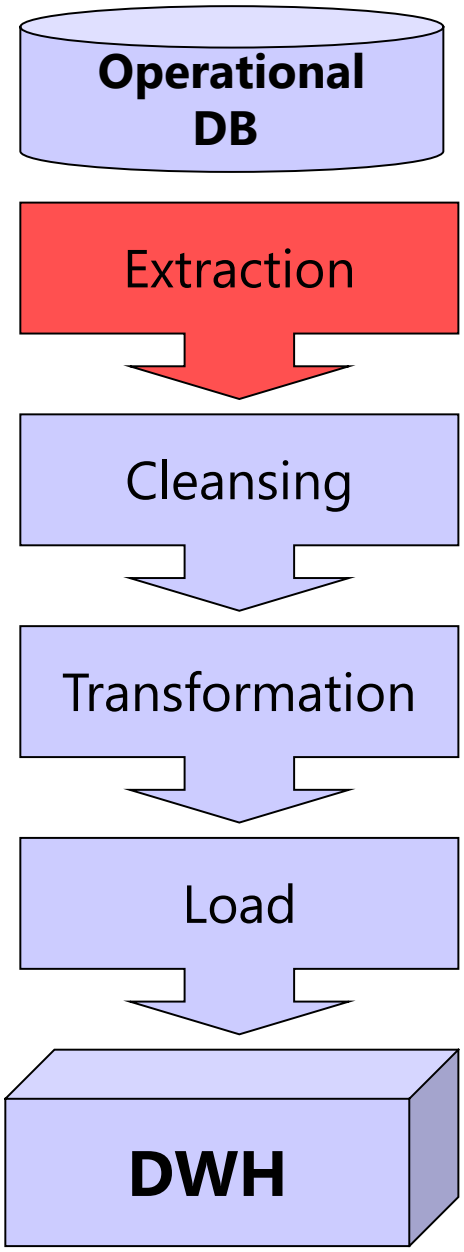
en tiempo real hay que garantizar que se procesan en nada; en stream pues cuando los proceses da igual, pero lo recibes

- ✓Data cleansing and Transformation
- ✓Data integration (calculation of derived data)
- ✓Key creation
- ✓Obtaining aggregates
- ✓Maintenance of metadata

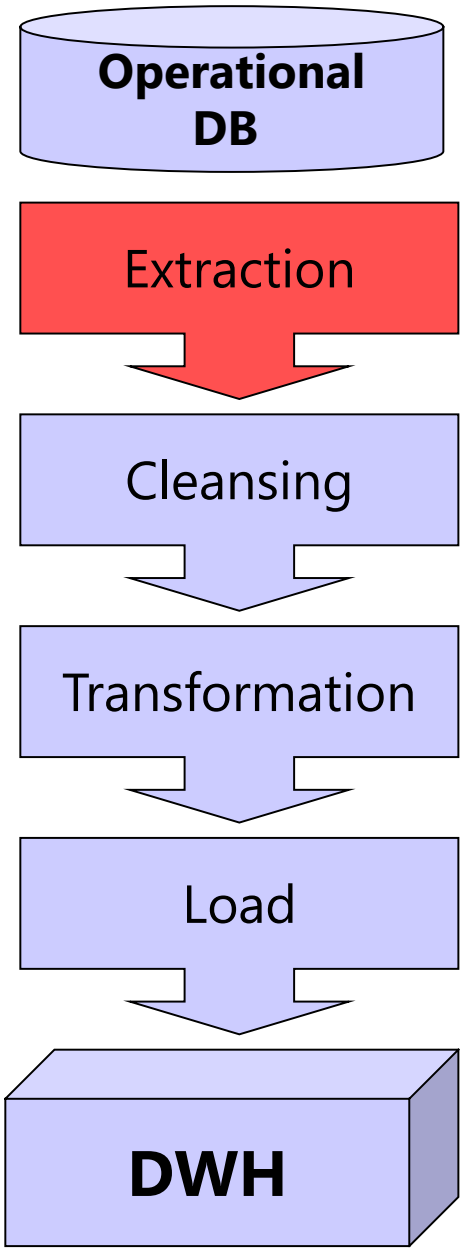
- ✓ Load data
- ✓ Indexing
- ✓ Loading aggregated data
- ✓ Quality test
- ✓ Exception management
- ✓Maintenance of metadata

los indices son muy importantes para el rendimiento (no pasarse !!). Un indice no es mas que otra tabla que ocupa espacio, asi que hay que ser cuidadoso

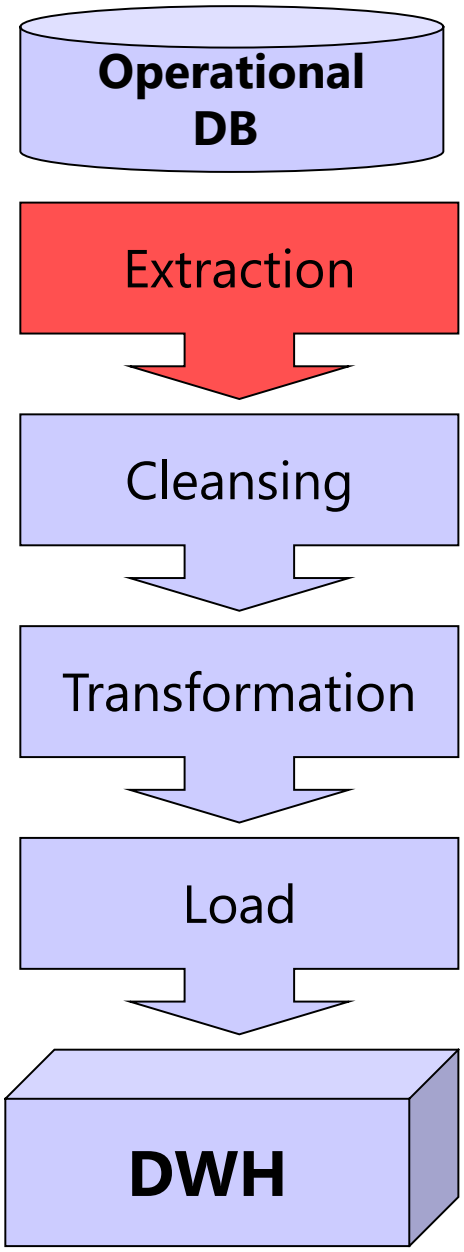
ETL



- **Extraction** = obtaining an image of a given subset of the source data to be loaded into the DWH
- **Logical** and **physical** extraction.
  - Logical: Complete vs incremental
  - Physical: Online vs Offline
- Also "**Ingestion**" in BigData



- **Complete extraction**
  - Capture **all the source** data in an instant of time
  - The extraction reflects **all available data** in the source system, it is not necessary to look at the source changes from the last extraction.
  - The source data are taken "**as is**", and it does not require additional logic (e.g. timestamps).
  - Example: dumping one table, or SQL that retrieves the entire table.
  - Moving from **hadoop to relational and viceversa**: Apache Sqoop, Spark, NiFi, Flume, Kafka; Talend, …

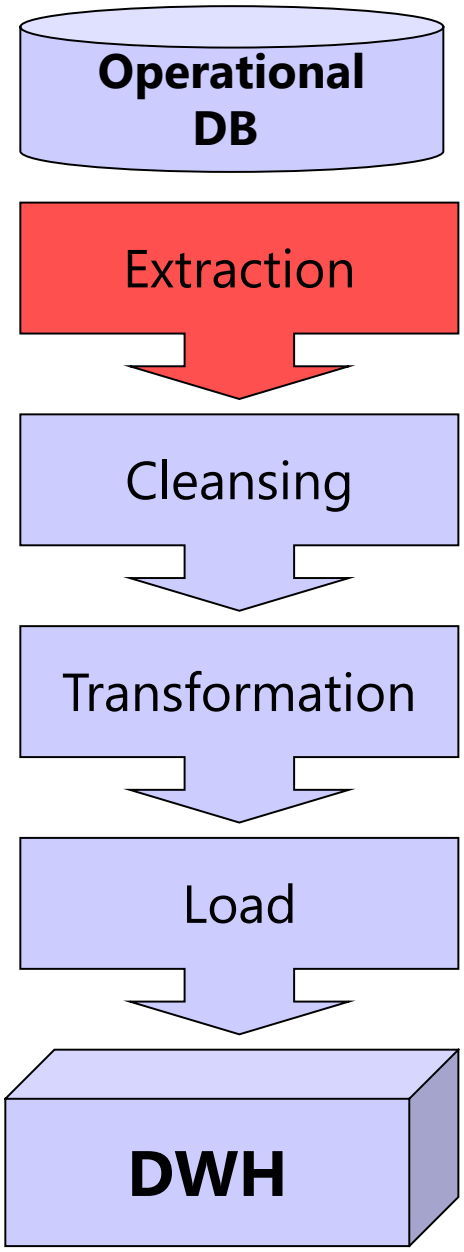


- **Incremental extraction**
  - At a **specific point in time**. One event triggers it.
  - Only **changed data** will be extracted.
  - Identify the information from one point changing in time **CDC (Change Data Capture):**
    - **Timestamps** on row
    - **Version number** on row
    - **Status** on row
    - **Triggers** (on database or application)
    - **Logs** (on database or application)
      - Powerful tools for this choice
  - Related to **Slowly Changing Dimension**
  - Pull Vs Push
  - Real time or event oriented processing: eg. Flume

no necesitamos /queremos extraer todos los datos

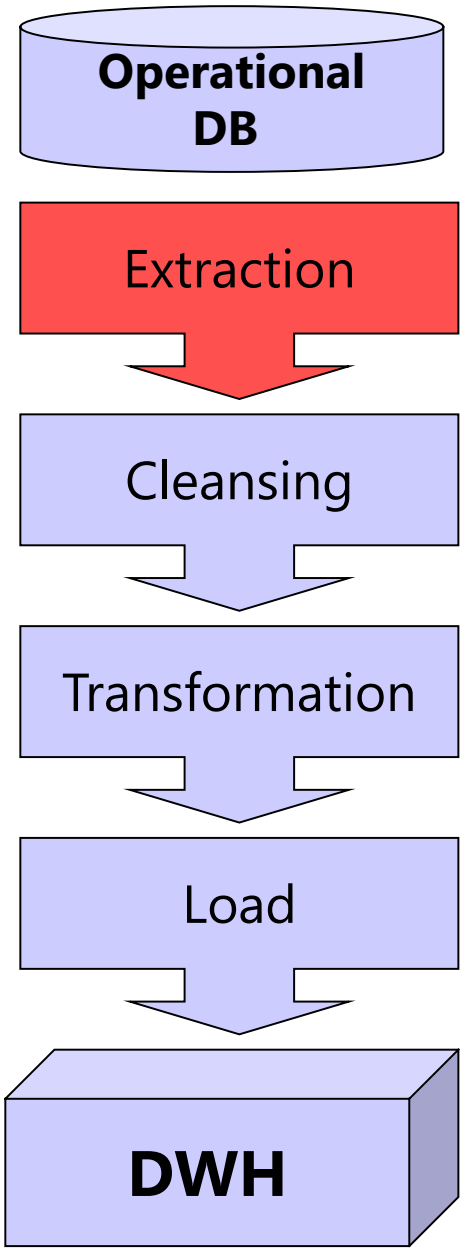
posibles soluciones

absicamente quien toma la inicitativa para coger los datos: pull hace peticion a la fuente. Push la fuente pone los datos a disposicion



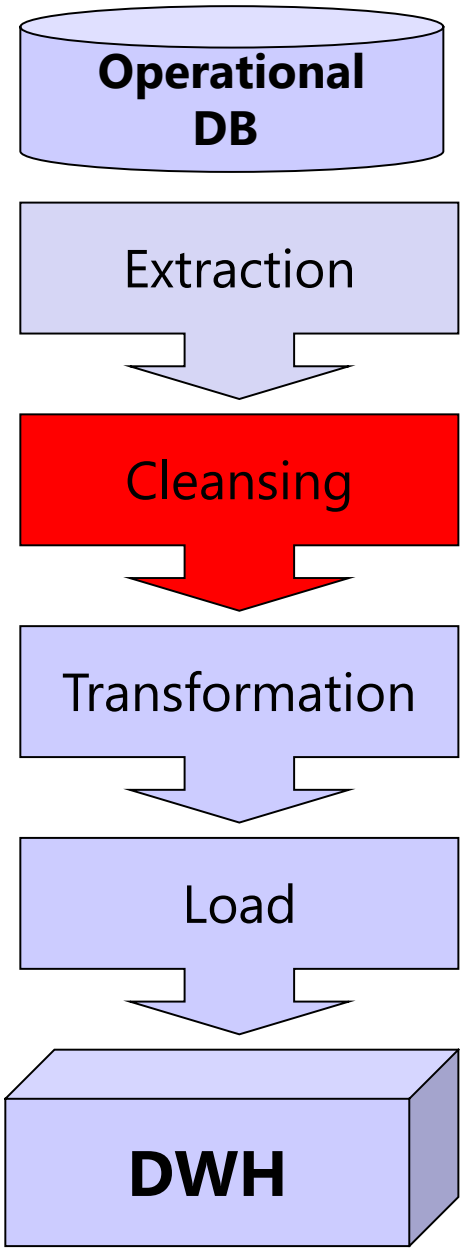
- **Online extraction**
  - **Connect directly** to the source or an intermediate system that stores the required data (eg, snapshots or change tables).
  - The **intermediate system** is not necessarily different from the source system. It can be a separate system designed for handling data extraction.
  - In some cases, incremental extraction is handled by this intermediate system (e.g., CDC mechanisms, logs, snapshots), making it more efficient than directly querying the primary source.
  - (See previous Incremental extraction)

ETL



- **Offline extraction**
  - Data not extracted directly from the source system. It runs outside the original source system. It is taken from external copies or backups of the data, which exist outside the original source system.
  - The data have an existing structure or are created by a dumping routine.
    - **Flat files**. Data in a generic format, eg CSV. They may include extra info on format.
    - **Dump files**. DBMS specific format. Extra info on SQL + format.
    - **Redo logs and archive logs. Transportable tablespaces.**

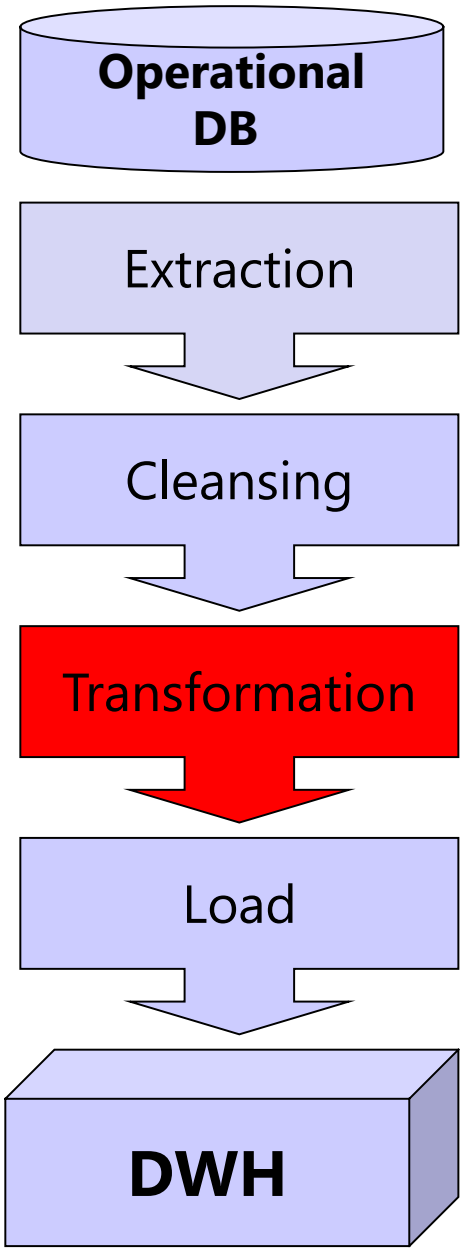
ETL



Cleansing = ensuring data quality

- **Correct mistakes**: format errors, wrong dates, misuse of fields, duplicate data, inconsistencies, restore referential integrity
- Also: decoding, reformatting, add timestamps, converting data, key generation, etc.
- **Alternative terms**: Wrangling, munging, tidying

ETL

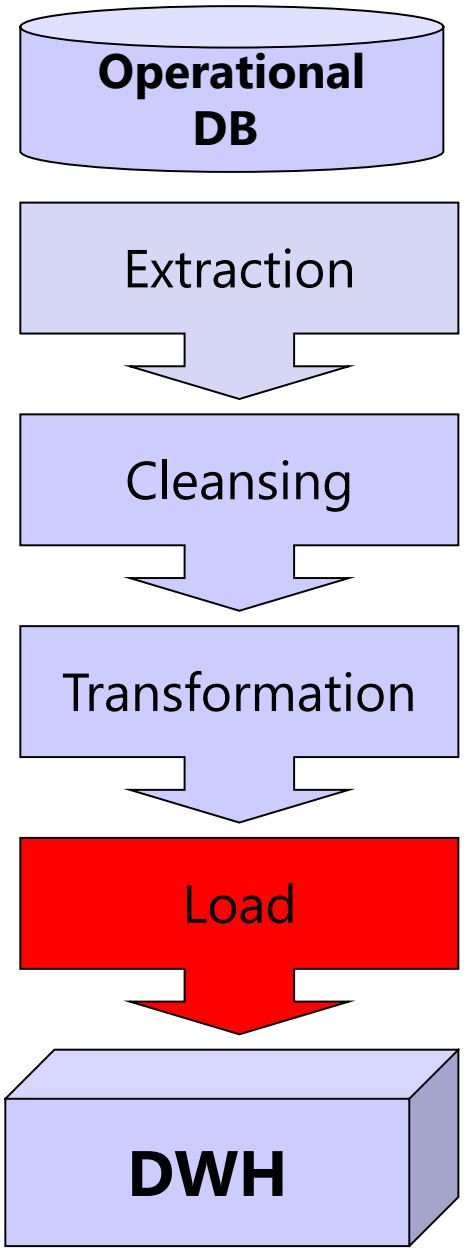


Transform = convert data from the format of an operational system to the DWH format

- **Record Level**:
  - Selection - partitioning
  - Joining - combination
  - Aggregation - precalculation
- **Field level**:
  - Single field - from field to field
  - Multiple field - from multiple fields to a single field or vice versa

- **Types of data transformation**
  - Gradual: step by step transformation process.
  - Pipelined
    - Multiple steps grouped.
    - It is not necessary to use intermediate tables.

ETL



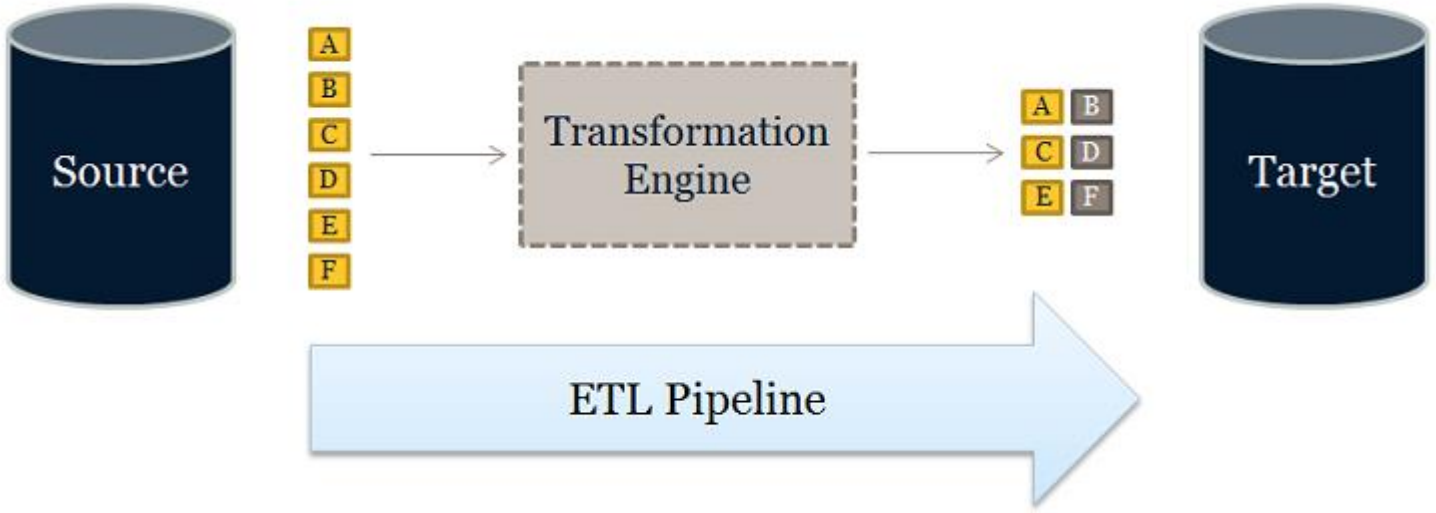
Load = enter data into the DW and create indexes

- Refresh method: determination of refresh intervals
  - Remove unused old data
- Update method: only changes in the sources are taken into account
- Loading mechanisms
  - SQL * Loader (pgload and COPY in Postgres)
  - External tables
  - OCI API and Direct-Path
  - Export / Import (pgdump in Postgres)
  - Sqoop

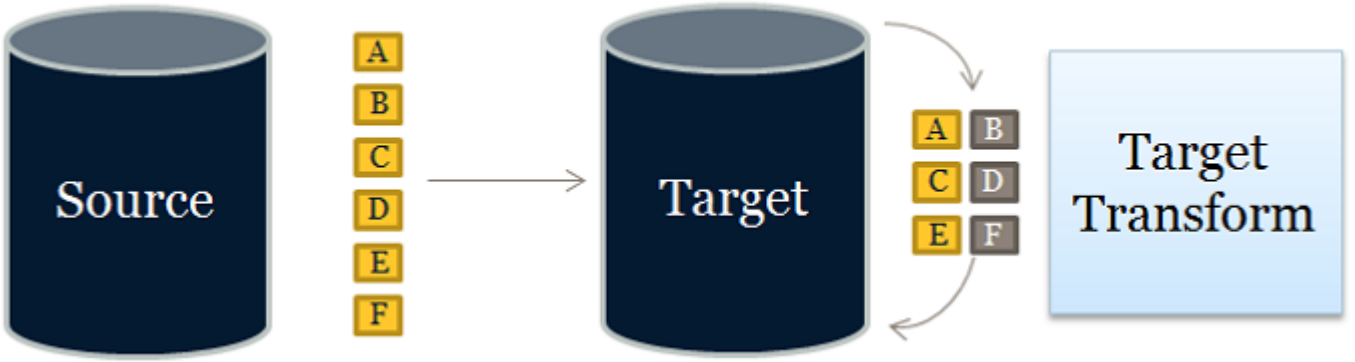
ETL vs ELT (note the order of letters)

- ETL Classic vs ELT "Big Data" approach

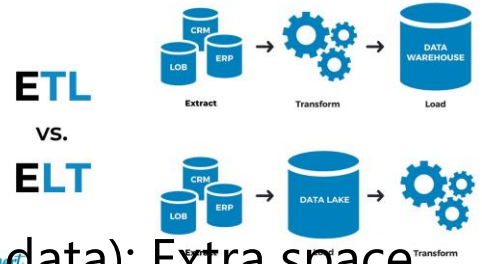
en elt los procesos de transformacion tienen lugar dentro del almacen.
Los datos no tienen que estar ajustados a un esquema determinado



etl es mas costosa y se usa mejor en entornos regulares. Si tengo necesidad de ver datos crudos mejor elt o ingerir datos mas rapidos



ETL vs ELT (note the order of letters)



ELT:

- **Data lakes**
  - **Raw data** in staging in DWH (minimal filter unneeded data): Extra space
  - **Faster**
  - For future needs, there are more data (not only the datamart)
- **Schemaless** / schema on read
- **Less processing on the source** since no transforming is being done
- **More processing when consuming** (need to transform)
- Languages with limited processing: hive, pig, …
- **More scalability and better I/O** than ETL if classic ddbb is used
- Supports both batch or event oriented

ETL vs ELT (note the order of letters)

Classic ETL

- Data marts: pre-aggregated data repositories optimized for queries. Cubes / **Schema on write**
- Only **clean, validated and quality** data
- Staging not needed: Less space
- More processing on extraction: a **pipeline**
  - Much slower
- GUI tools: abstraction independent from language and platform, very rich and powerful transformations

ETL tools

An ETL integration tool is essential for handling data movement, transformation, and quality assurance in data warehousing processes.

- An integration tool needs:
  - Many interfaces
  - Transformations
  - Quality: validation, filters, …
  - Open Source:
    - Apache Airflow, Apache Kafka, Apache Nifi, Pentaho Data Integration, Talend OS, …
  - Commercial:
    - Hevo Data, Informatica Power Center, SnapLogic, Jitterbit, Oracle Data Integrator, Mulesoft, IBM Datastage, Microsoft SQL Server Integration Services, …

Business intelligence

Unit 2 – Datawarehouse and OLAP
S2-4 – Physical design

DW design

Conceptual design

↓

Logical design

↓

Physical design

↓

Implementation

→

Efficiency in storage

Efficiency in queries

Storage model (ROLAP, MOLAP, HOLAP)

Physical design

Objective: improve performance

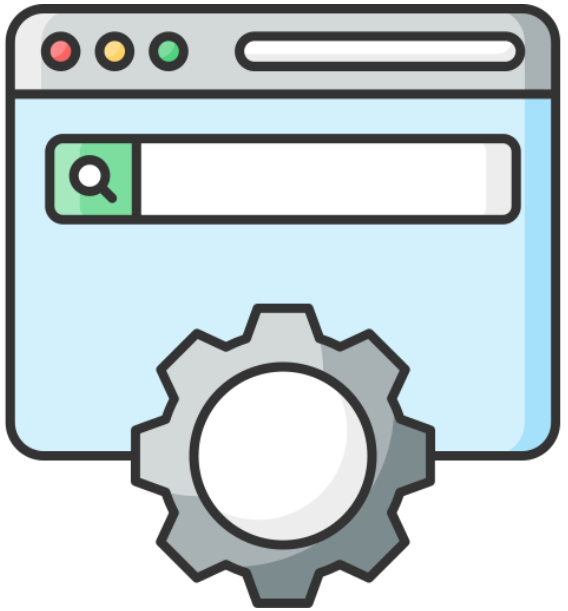
Focus on **improving query performance and scalability**.

Balance between storage efficiency and fast retrieval.

Architecture: **ROLAP, MOLAP, HOLAP**

Storage strategy and query optimization

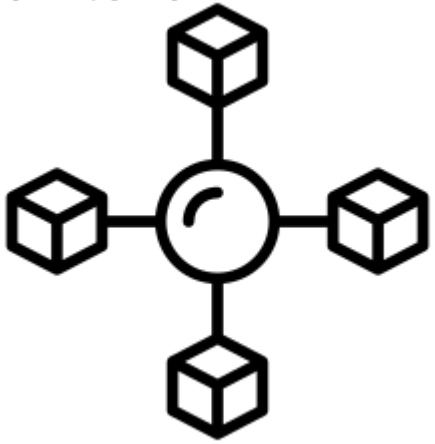
- Denormalization (ROLAP, HOLAP)
- Partitioning (ALL)
- Index (ALL)
- Compression (ALL)
- Materialized Views (ROLAP, HOLAP)



Big data challenges

Different physical storage and query structures

- Multidimensional OLAP (**MOLAP**): DBMS built specifically for data analysis. Data is stored in cubes to improve multidimensional queries.
- Relational OLAP (**ROLAP**) Relational DBMS, which can include both traditional row-based and modern columnar databases.
- Hybrid systems: **HOLAP**. Combines benefits of ROLAP and MOLAP for optimized performance.



ROLAP:

- The DWH is built **on top of relational DBMS**, often using columnar storage to improve performance with big datasets.
- RDBMS vendors provide OLAP-specific extensions, such as **window functions** (e.g., RANK(), SUM() OVER()), and **grouping sets** (GROUPING SETS, CUBE, ROLLUP) or integrated data mining tools: Oracle OLAP, Microsoft SSAS ROLAP, PostgreSQL with OLAP extensions.
- Typically uses a **star schema** or **snowflake** schema with fact and dimension tables.

ROLAP

ROLAP:

Advantages:

- Scalability: There is no limit on the amount of data.
- Proven technology: Functionality already available.
- Separation storage from OLAP processing.
- Cost-effective

Disadvantages:

- Performance overhead: queries on normalized db, usually with multiple joins. Complex multidimensional queries usually slower than in MOLAP systems.
- Limited functionality for SQL. For example, to perform complex calculations or hierarchical queries.
- No pre-aggregated data, slowing response times.
- Potential locking and concurrency problems in busy systems.

MOLAP

- Consist of physically storing data in multidimensional structures (**cubes**) so that the external representation *matches* the internal representation.
- It allows **fast retrieval** of pre-aggregated data.
- BD complexity is **hidden** from the users.
- Analysis is done on **aggregated data** and precalculated metrics or indicators.
- OLAP Engine: The **MOLAP engine** processes and responds to multidimensional queries with very low latency due to pre-aggregated data.
- Specific functionality: Array data structures (proprietary formats), query optimization , data compaction.

MOLAP

MOLAP:

Advantages:

- Fast Performance for small and medium size datasets: slide & dice specific operations.
- Hierarchical analysis: easy to navigate through dimensions.
- Complex calculations are pre-generated.
- Easier to use, even for inexperienced users.

MOLAP

MOLAP:

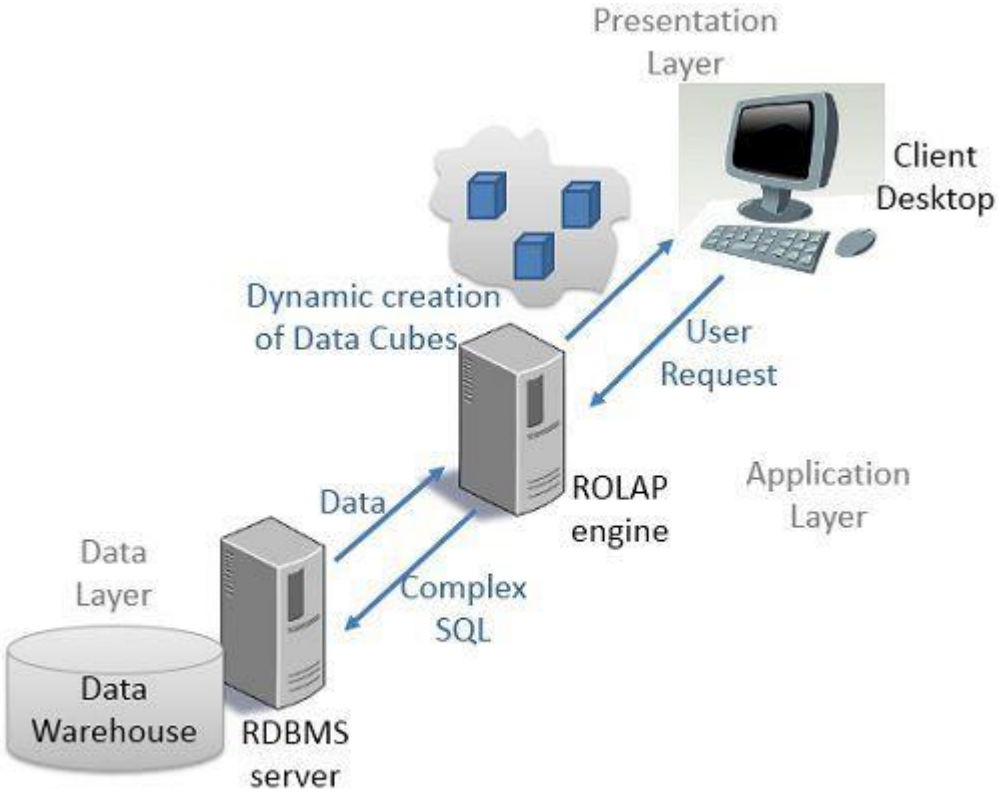
Disadvantages:

- Limited size: Problems with large datasets and many dimensions.
- Latency when loading and pre-aggregate: it can take time.
- Storage overhead: pre-aggregated data can take a lot of storage, specially for large datasets.
- New investment: this technology is not usually present in enterprises. Also usually proprietary.
- Limited Flexibility: Less flexible when new dimensions or hierarchies are introduced, requiring a cube reprocessing.

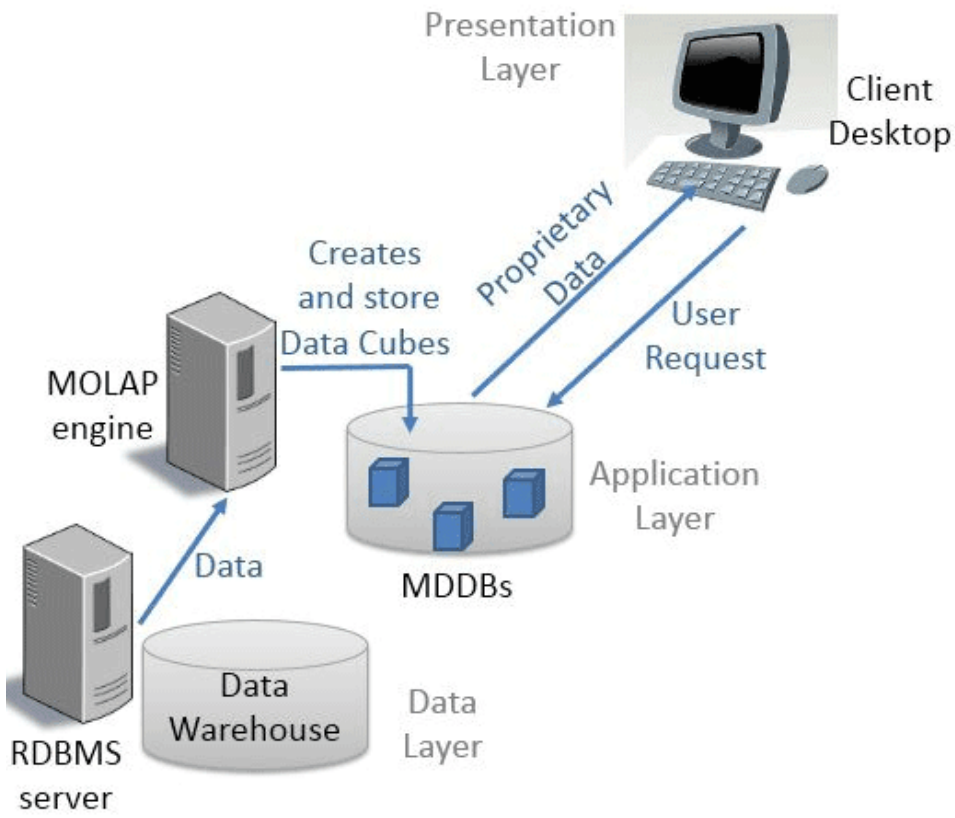
MOLAP: Improving performance

- **Partial Cube Aggregation**: Selectively pre-aggregate dimensions or measures that are queried most frequently, reducing cube processing time.
- **Incremental** Cube Refresh: Use incremental refresh to update only the new data, minimizing downtime and latency.
- **Sparse Data** Handling: Manage sparse data more efficiently by compressing and storing only non-null values.
- **Advanced Partitioning**: Partitioning cubes by key dimensions (e.g., time) allows for faster access and easier management of data over time.

ROLAP and MOLAP



ROLAP Model



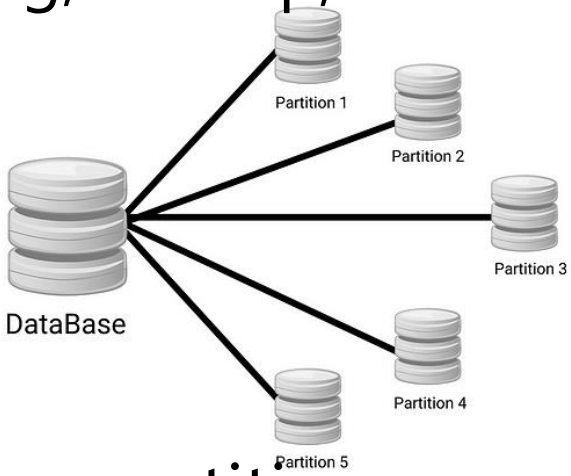
MOLAP Model

Image Source:EDUCBA

Database partitioning

Replace large tables with **more manageable** smaller subtables

- Allows **parallel and distributed** execution
- Partition **prunning** and **partition wise** joins
- **Partition data processin**g: ETL, refreshing, backup, indexing.



Horizontal

- Rows of a table are separated

Vertical

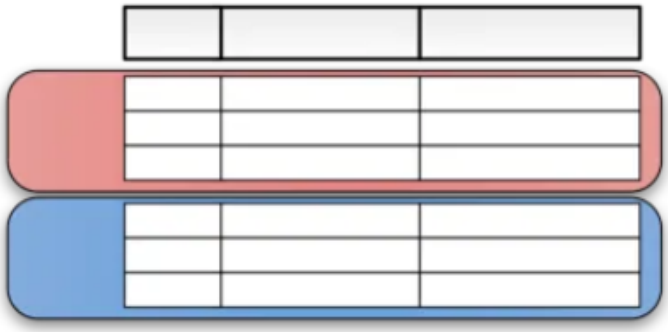
- Columns of a table are separated

Dynamic Partitioning: automatically creates partitions based on incoming data, without the need for predefined ranges or categories.

Time-based Partitioning: Ideal for time series data. A flavor of horizontal partitioning.

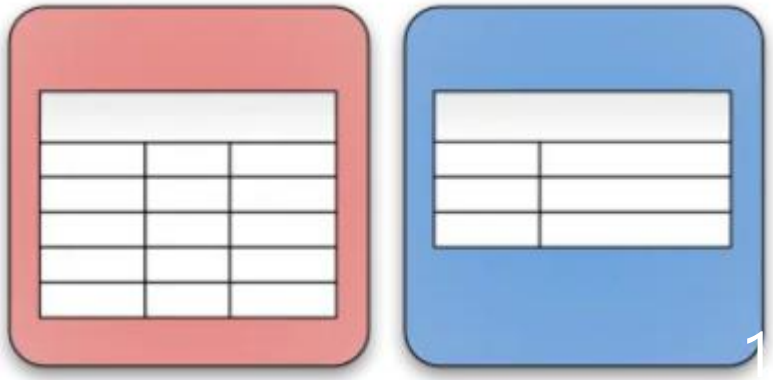
Horizontal partitioning

- **Rows of the same table** are stored in different locations (e.g. disks, nodes) according to some criteria.
- All partitions have the **same columns**.
- **Improves** query performance by scanning fewer rows.
- Remember that in DW the load is incremental (usually there are not updates).
  - New inserts do not produce table movements, can be just as simple are adding a partition.
  - Easy to delete by “time”.
- Criteria
  - Range, List, Hash, Partitioned Index
  - Composed: range + hash, list+hash

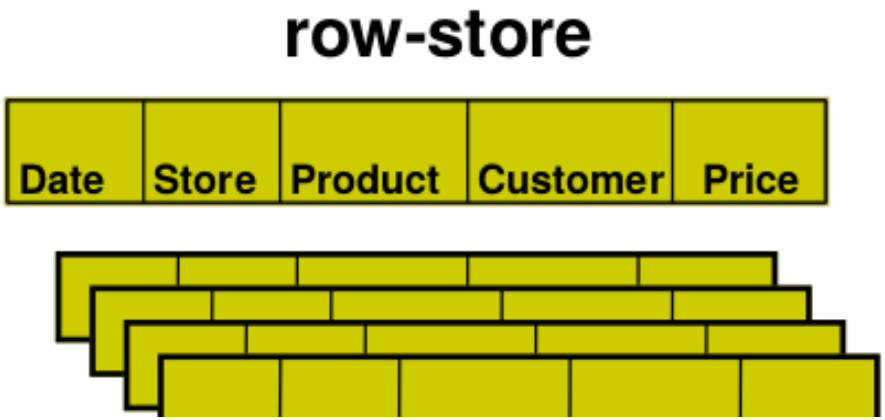


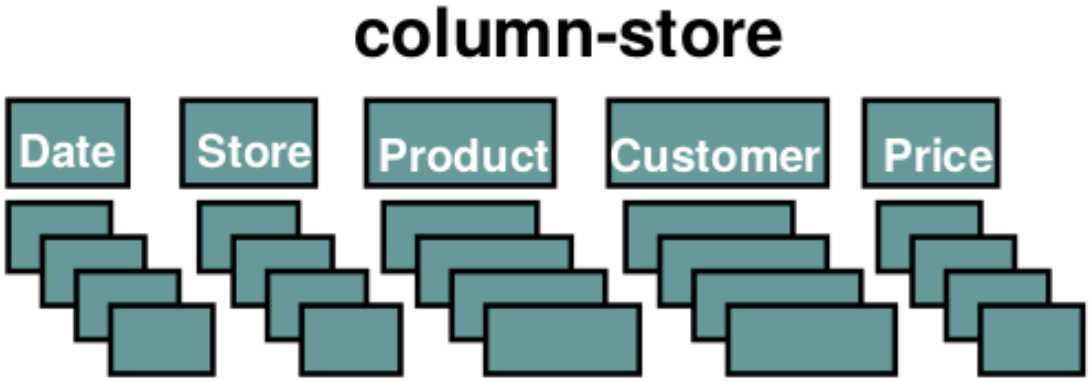
Vertical partitioning

- Divides a table into smaller tables with the **same rows** but a **subset of columns**.
- Not only for multidimensional db
- For **DWH**:
  - A number of operations are **aggregations by columns**
  - **Only columns affected** by operation are retrieved
  - **Compression** of columns is very efficient
    - Text columns without predefined size
  - Some columns may change more than others
  - **Improves performance** for OLAP queries by isolating frequently accessed columns..



Columnar databases





Pros:
-easy to add/modify a record
Cons
-might read in unnecessary data

Pros:
-only need to read in relevant data
- Column compression easy and efficient (light algorithms)
- Aggregated queries very fast.
Cons:
- tuple writes require multiple accesses

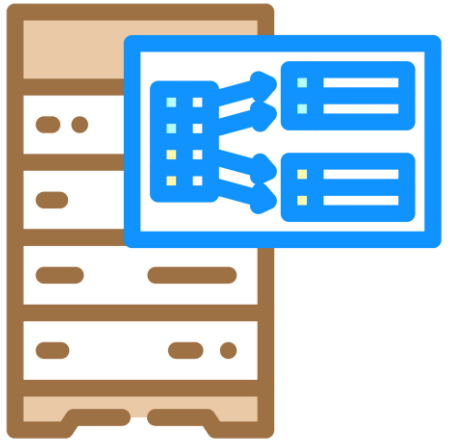
Note: We don't elaborate more here because you see them in other course.

http://cs-www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf

Indexing

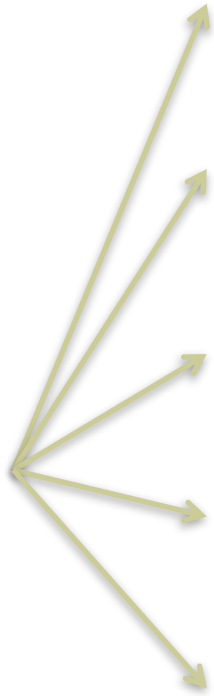
Index: secondary data structure for arbitrary access and efficient

- **Characteristics**:
  - **Content**:
    - Value of the attribute indexed
    - Address to the storage
  - **Sorted entries** according to the value for efficient searches
  - **Small size**: usually fits in one data block
- Some types: B-Trees, Hash
  - Specific for OLAP: Bitmap, Join Index
- Used only for retrieving **few values**



Index: General concept

cod A	Bloque
00	B4
11	B1
22	B2
33	B2
44	B4
55	B3
77	B3
88	B1
99	B5



Bloque	cod A	nombre	agencia	fechaNacim
B1	88	Fele Martínez	Glam	22/02/75
	11	Najwa Nimri	Actors	14/02/72
B2	33	Nancho Novo	Rol	17/09/78
	22	Santiago Segura	Amigueta s	17/07/65
B3	77	Luis Tosar	BCN	13/10/71
	55	Candela Peña	Actors	14/07/73
B4	00	Maribel Verdú	Glam	02/10/70
	44	Penélope Cruz	BCN	28/04/74
B5	99	Javier Bardem	BCN	01/03/69

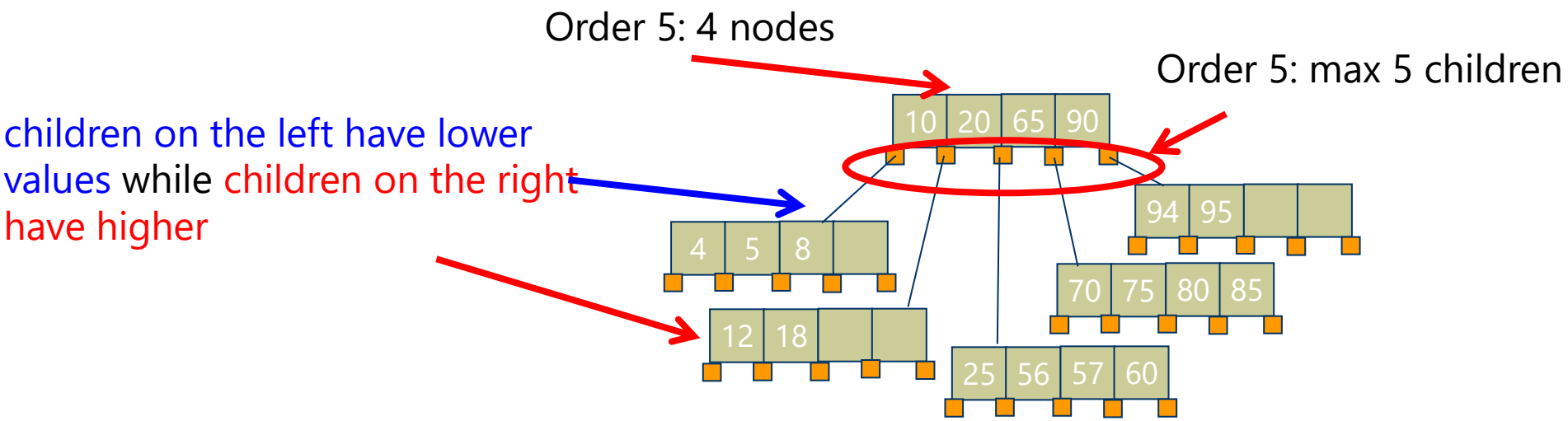
Just one block

B-Tree Index

Deep balanced tree: all leaf nodes at the same depth. Each path from the root to a leaf has the same length.

N-order:

Each node can have up to n children and store n-1 keys.



Bitmap Index

Index on a particular column
Each value in the column has a bit vector: bit-op is fast
The **length** of the bit vector: **# of records** in the base table
The i-th bit is set if the i-th row of the base table has the value for the indexed column

Cust	Region	Type
C1	Asia	Retail
C2	Europe	Dealer
C3	Asia	Dealer
C4	America	Retail
C5	Europe	Dealer

RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Bitmap Index

Reduced response time for large classes of ad hoc queries: Flexible WHERE.

Reduced storage requirements compared to other indexing.

Fully indexing a large table with a traditional B-tree index can lead to structures several times larger than the data in the table.

Bitmap indexes are typically **only a fraction of the size** of the indexed data in the table.

Efficient maintenance during parallel DML and loads.

The advantages of using bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small (1:100).

A table with one million rows, a column with 10,000 distinct values is a candidate

Comparison, union and aggregation use bit arithmetic.

Not suitable for high cardinality domains: too many columns (better B-Tree)

Often support compression

[Bitmap] Join Index

It materializes relational join in JI file and speeds up relational join

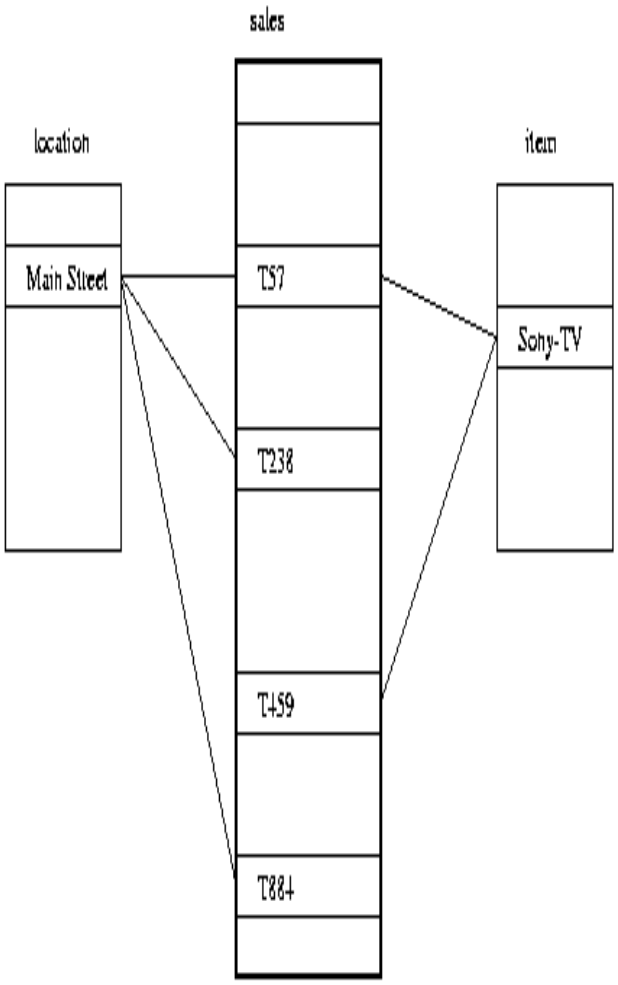
- Join index: JI(R-id, S-id) where R (R-id, …) JOIN S (S-id, …). R-fact table, S-dimension.

Better performance as it precomputes join results and stores them into bitmap format.

In DWH: join between facts and dimensions.

- Equi-inner join between PK and FK
- E.g. fact table: Sales and two dimensions city and product
  - A join index on city maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city
- Join indices can span multiple dimensions

NOTE: Related to denormalization. They precompute join results and reduce the need for real-time joins.



Compression

Used in **multidimensional** DBMS.
Saving **disk space**, increasing **memory efficiency**, and improving **query performa**nce by reducing the amount of data read from disk (multidimensional databases are usually massive).
Overhead for updating, deleting, and processing in general. Data needs to be uncompressed, modified and compressed, leading to **overload during write operations**. In DWH environments, write operations are **less frequent**.
Hybrid Columnar Compression: groups of rows are stored in columnar format, with the values for a given column stored and compressed together.
Storing column data together, with the same data type and similar characteristics, drastically **increases the storage savings** achieved from compression.

Materialized Views

Materialized views are query results that have been **stored in advance** so long-running calculations are not necessary when you actually execute your SQL statements.
They act like tables and improve performance in complex queries, especially those involving joins and aggregations.
Useful for aggregates.
Problem:

- **Resources** consumption
- Refreshing policies: **on demand, on commit**.

Efficient Data Cube Computation

Data cube can be viewed as a **lattice of cuboids**

- The bottom-most cuboid, with the most detailed data , is the **base cuboid**
- The top-most cuboid (**apex**) contains only one cell
- How many cuboids in an n-dimensional cube with L levels?

$$T = \prod_{i=1}^{n} (L_i + 1)$$

Materialization of data cube

- Materialize every cuboid (**full materialization**), none (**no materialization**, every query is computed from the base cuboid), or some (**partial materialization**)
- Selection of which cuboids to materialize
  - Based on size, sharing of cuboids between queries, access frequency, etc.

The "Compute Cube" Operator

Cube definition and computation in DMQL

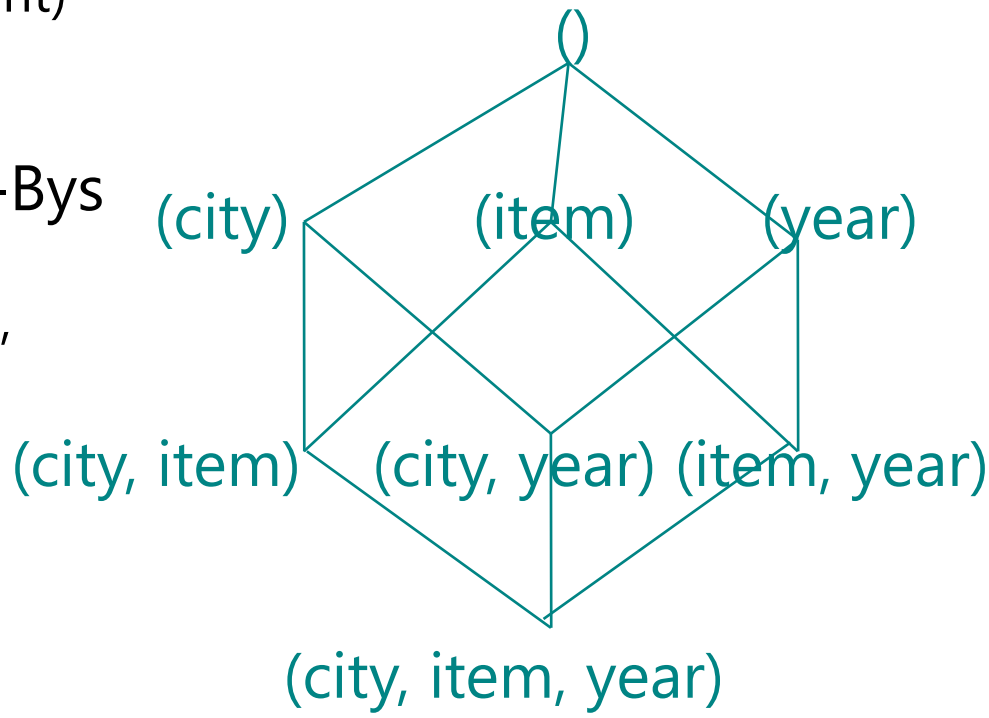
- define cube sales [item, city, year]: sum (sales_in_dollars)
- compute cube sales

Transform it into a SQL-like language (with a new operator cube by, introduced by Gray et al.'96)

```
SELECT item, city, year, SUM (amount)
FROM SALES
CUBE BY item, city, year
```

Need compute the following Group-Bys

- (item, city, year),
- (item,city),(item, year), (city, year),
- (item), (city), (year)
- ()



Efficient Processing OLAP Queries

Determine which operations should be performed on the available cuboids

- Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection

Determine which materialized cuboid(s) should be selected for OLAP op.

- Let the query to be processed be on {brand, province_or_state} with the condition "year = 2004", and there are 4 materialized cuboids available:
  1) {year, item_name, city}
  2) {year, brand, country}
  3) {year, brand, province_or_state}
  4) {item_name, province_or_state} where year = 2004

  Which should be selected to process the query?

con country no puedes llegar a provincia o estado. Surven todos menos el 2

ROLAP MOLAP y OLAP SABER EN QUE CONSISTEN PREGUNTA BREVE
Particionamiento
Indexado importante
join bitmap y tal saber en consiste
cuboides

Explore **indexing structures** and **sparse vs. dense** array structs in MOLAP

tipo de examen:
alguna de tipo test
alguna de conceptos (uno o dos parrafos)
alguna un poco mas largo (2 o 3 parrafos)
Suele poner un alternativa en alguna pregunta.

Data Lakehouses

A data lakehouse represents the synergy between a data warehouse and a data lake, offering the scalability and flexibility of data lakes with the ACID transactions and structure of data warehouses.

- **Storage Layers**:
  - **Raw Data Layer**: Stores unstructured or semi-structured data as it is ingested (e.g., log files, JSON, or Parquet).
  - **Curated Data Layer**: Stores cleaned and transformed data, often partitioned by time or domain to improve queries.
- **Query Optimization**: Indexing and data skipping techniques (e.g., Apache Parquet's predicate pushdown).
- **Columnar Formats**: storage formats like **Parquet** and **ORC** for efficient query performance.

Data Lakehouses

- **Transaction Support**: Ensures ACID compliance for structured queries and updates, using technologies like **Delta Lake** (Databricks) and **Apache Iceberg**.
- **Partitioning and Compaction**: Strategies for partitioning data across file systems and regular data compaction to improve performance.
- **Serverless architectures** (scalable, on-demand processing of large datasets without infrastructure management). E.g. Google BigQuery, Amazon Redshift Spectrum.
- **Real-time analytics** (very fast query responses for low latency applications). E.g. Apache Pinot and ClickHouse.
- Combines **Data Lakes** (+ scalability for raw, unstructured data) and **Warehouses** (+ structure + ACID) for hybrid data management.

Modern Software Alternatives

ROLAP:

- ClickHouse, Apache Pinot, Google BigQuery, Snowflake, Amazon Redshift

HOLAP:

- Apache Druid, Apache Kylin, Atscale,

MOLAP:

Microsoft Azure Analysis Services, Kyvos Insights, Oracle Essbase,

Business intelligence

Unit 2 – Datawarehouse and OLAP
S2-4 – Physical design

DW design

Conceptual design

↓

Logical design

↓

Physical design

↓

Implementation

→

Efficiency in storage

Efficiency in queries

Storage model (ROLAP, MOLAP, HOLAP)

Physical design

Objective: improve performance

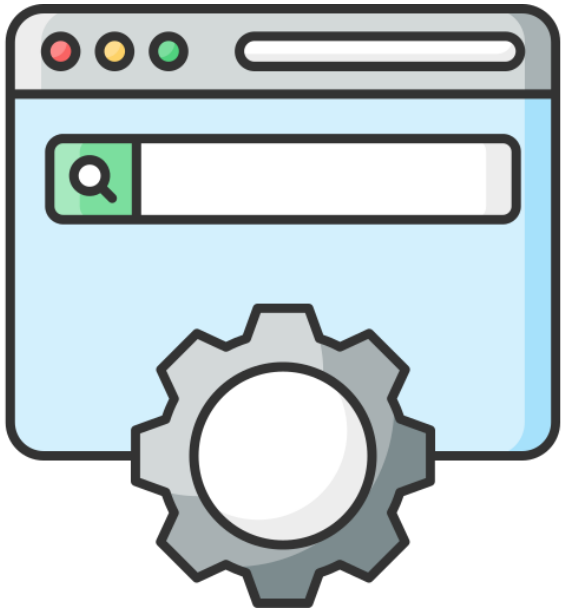
Focus on **improving query performance and scalability**.

Balance between storage efficiency and fast retrieval.

Architecture: **ROLAP, MOLAP, HOLAP**

Storage strategy and query optimization

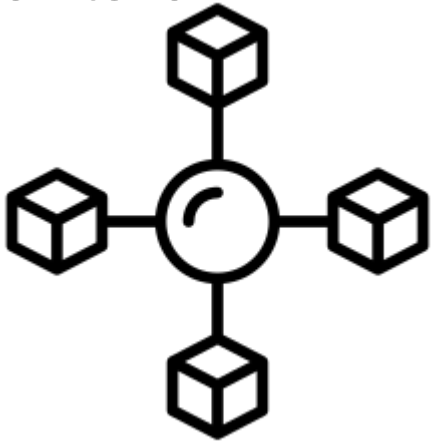
- Denormalization (ROLAP, HOLAP)
- Partitioning (ALL)
- Index (ALL)
- Compression (ALL)
- Materialized Views (ROLAP, HOLAP)



Big data challenges

Different physical storage and query structures

- Multidimensional OLAP (**MOLAP**): DBMS built specifically for data analysis. Data is stored in cubes to improve multidimensional queries.
- Relational OLAP (**ROLAP**) Relational DBMS, which can include both traditional row-based and modern columnar databases.
- Hybrid systems: **HOLAP**. Combines benefits of ROLAP and MOLAP for optimized performance.



ROLAP:

- The DWH is built **on top of relational DBMS**, often using columnar storage to improve performance with big datasets.
- RDBMS vendors provide OLAP-specific extensions, such as **window functions** (e.g., RANK(), SUM() OVER()), and **grouping sets** (GROUPING SETS, CUBE, ROLLUP) or integrated data mining tools: Oracle OLAP, Microsoft SSAS ROLAP, PostgreSQL with OLAP extensions.
- Typically uses a **star schema** or **snowflake** schema with fact and dimension tables.

ROLAP

ROLAP:

Advantages:

- Scalability: There is no limit on the amount of data.
- Proven technology: Functionality already available.
- Separation storage from OLAP processing.
- Cost-effective

Disadvantages:

- Performance overhead: queries on normalized db, usually with multiple joins. Complex multidimensional queries usually slower than in MOLAP systems.
- Limited functionality for SQL. For example, to perform complex calculations or hierarchical queries.
- No pre-aggregated data, slowing response times.
- Potential locking and concurrency problems in busy systems.

MOLAP

- Consist of physically storing data in multidimensional structures (**cubes**) so that the external representation *matches* the internal representation.
- It allows **fast retrieval** of pre-aggregated data.
- BD complexity is **hidden** from the users.
- Analysis is done on **aggregated data** and precalculated metrics or indicators.
- OLAP Engine: The **MOLAP engine** processes and responds to multidimensional queries with very low latency due to pre-aggregated data.
- Specific functionality: Array data structures (proprietary formats), query optimization , data compaction.

MOLAP:

Advantages:

- Fast Performance for small and medium size datasets: slide & dice specific operations.
- Hierarchical analysis: easy to navigate through dimensions.
- Complex calculations are pre-generated.
- Easier to use, even for inexperienced users.

MOLAP

MOLAP:

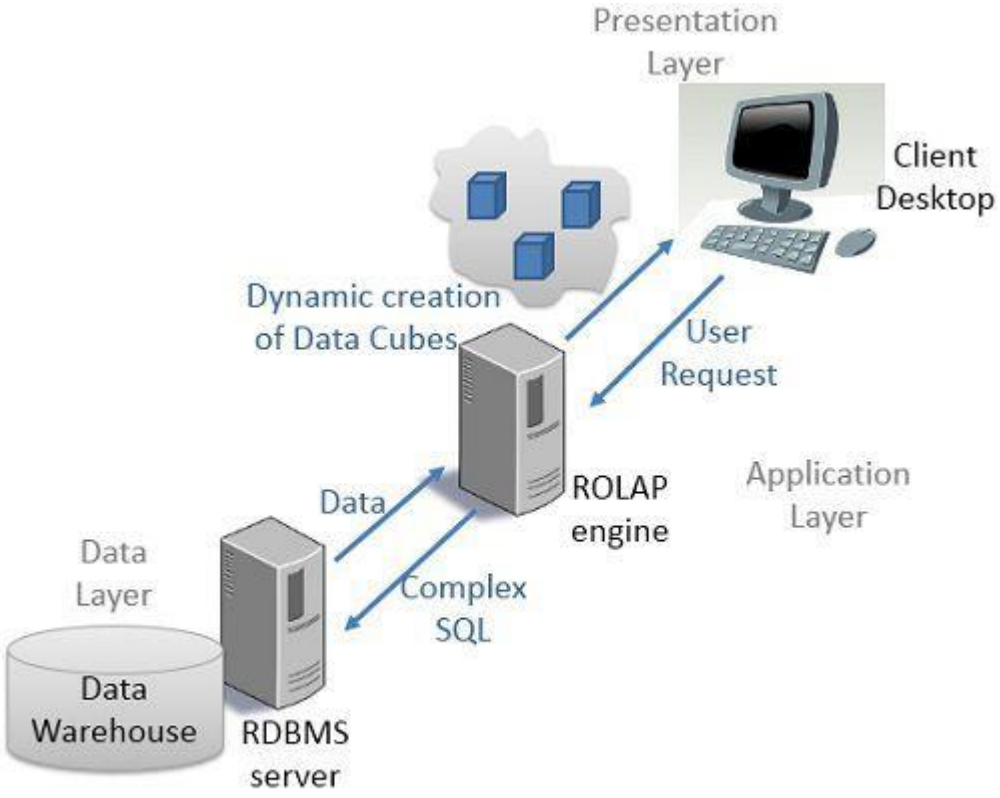
Disadvantages:

- Limited size: Problems with large datasets and many dimensions.
- Latency when loading and pre-aggregate: it can take time.
- Storage overhead: pre-aggregated data can take a lot of storage, specially for large datasets.
- New investment: this technology is not usually present in enterprises. Also usually proprietary.
- Limited Flexibility: Less flexible when new dimensions or hierarchies are introduced, requiring a cube reprocessing.

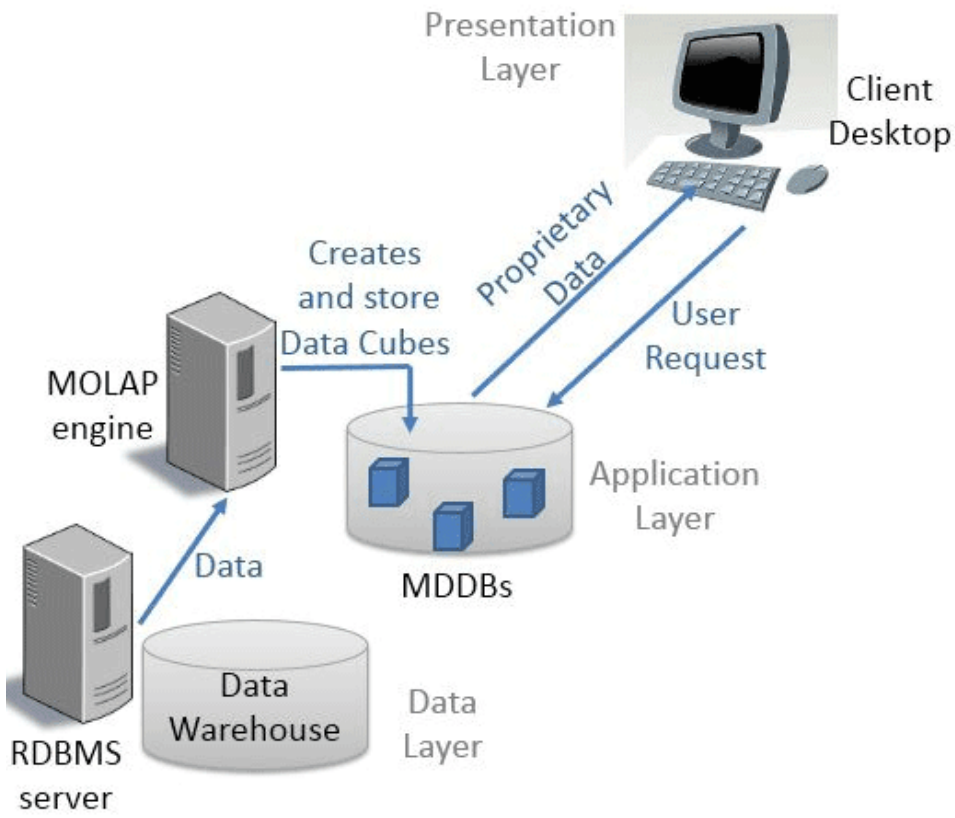
MOLAP: Improving performance

- **Partial Cube Aggregation**: Selectively pre-aggregate dimensions or measures that are queried most frequently, reducing cube processing time.
- **Incremental** Cube Refresh: Use incremental refresh to update only the new data, minimizing downtime and latency.
- **Sparse Data** Handling: Manage sparse data more efficiently by compressing and storing only non-null values.
- **Advanced Partitioning**: Partitioning cubes by key dimensions (e.g., time) allows for faster access and easier management of data over time.

ROLAP and MOLAP



ROLAP Model



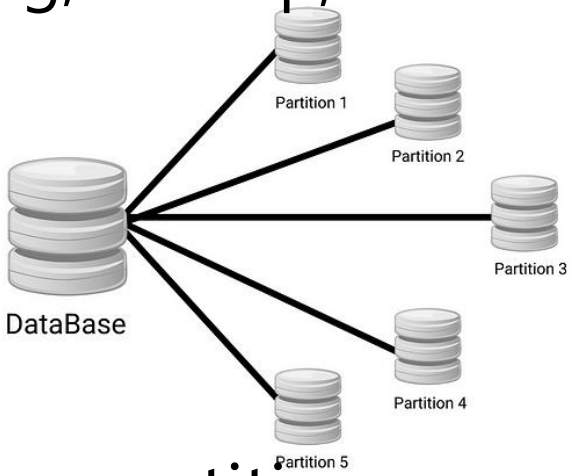
MOLAP Model

Image Source:EDUCBA

Database partitioning

Replace large tables with **more manageable** smaller subtables

- Allows **parallel and distributed** execution
- Partition **prunning** and **partition wise** joins
- **Partition data processin**g: ETL, refreshing, backup, indexing.



Horizontal

- Rows of a table are separated

Vertical

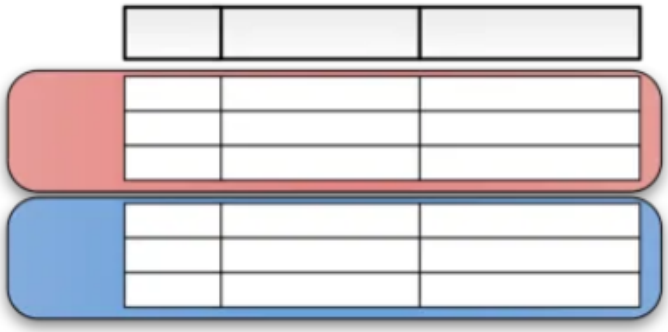
- Columns of a table are separated

Dynamic Partitioning: automatically creates partitions based on incoming data, without the need for predefined ranges or categories.

Time-based Partitioning: Ideal for time series data. A flavor of horizontal partitioning.

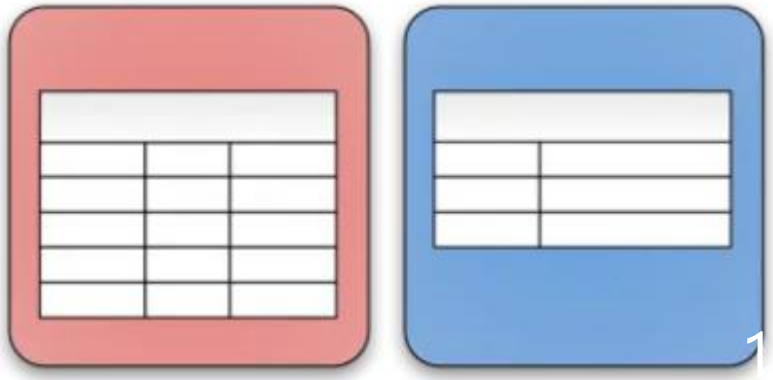
Horizontal partitioning

- **Rows of the same table** are stored in different locations (e.g. disks, nodes) according to some criteria.
- All partitions have the **same columns**.
- **Improves** query performance by scanning fewer rows.
- Remember that in DW the load is incremental (usually there are not updates).
  - New inserts do not produce table movements, can be just as simple are adding a partition.
  - Easy to delete by “time”.
- Criteria
  - Range, List, Hash, Partitioned Index
  - Composed: range + hash, list+hash

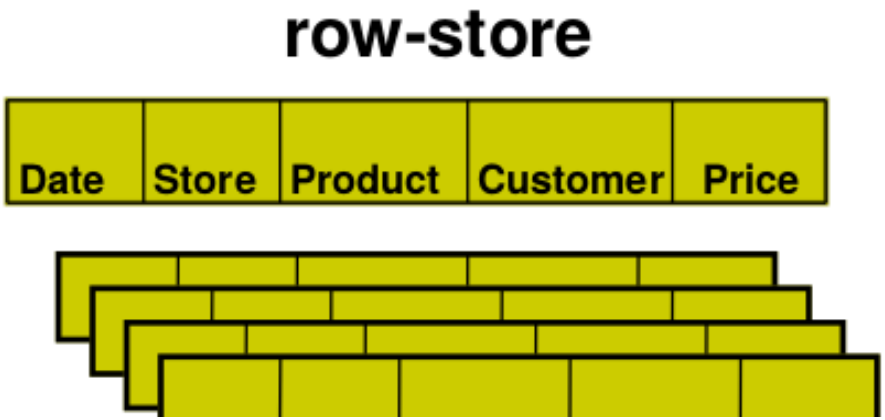


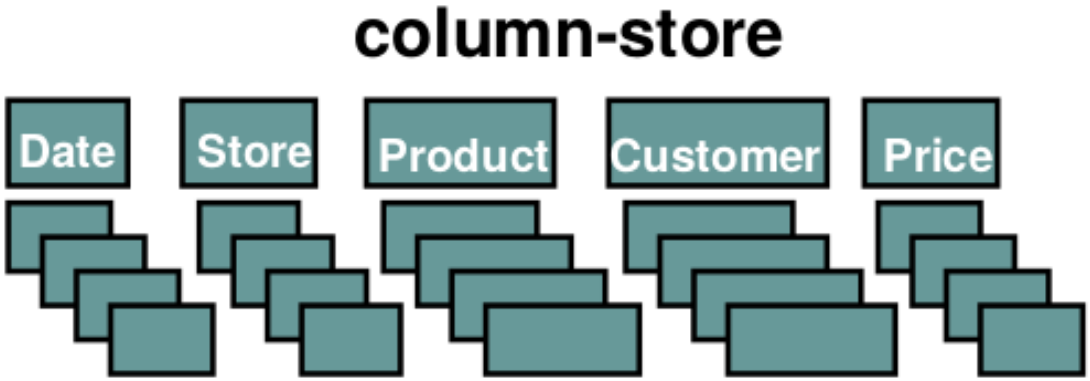
Vertical partitioning

- Divides a table into smaller tables with the **same rows** but a **subset of columns**.
- Not only for multidimensional db
- For **DWH**:
  - A number of operations are **aggregations by columns**
  - **Only columns affected** by operation are retrieved
  - **Compression** of columns is very efficient
    - Text columns without predefined size
  - Some columns may change more than others
  - **Improves performance** for OLAP queries by isolating frequently accessed columns..



Columnar databases





Pros:
-easy to add/modify a record
Cons
-might read in unnecessary data

Pros:
-only need to read in relevant data
- Column compression easy and efficient (light algorithms)
- Aggregated queries very fast.
Cons:
- tuple writes require multiple accesses

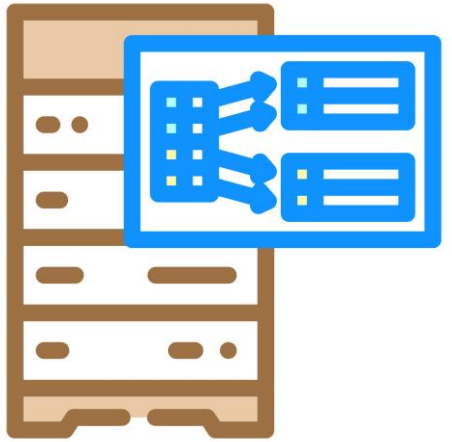
Note: We don't elaborate more here because you see them in other course.

http://cs-www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf

Indexing

Index: secondary data structure for arbitrary access and efficient

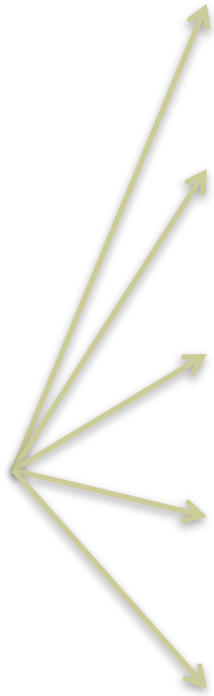
- **Characteristics**:
  - **Content**:
    - Value of the attribute indexed
    - Address to the storage
  - **Sorted entries** according to the value for efficient searches
  - **Small size**: usually fits in one data block
- Some types: B-Trees, Hash
  - Specific for OLAP: Bitmap, Join Index
- Used only for retrieving **few values**



Index: General concept

cod A	Bloque
00	B4
11	B1
22	B2
33	B2
44	B4
55	B3
77	B3
88	B1
99	B5

Just one block



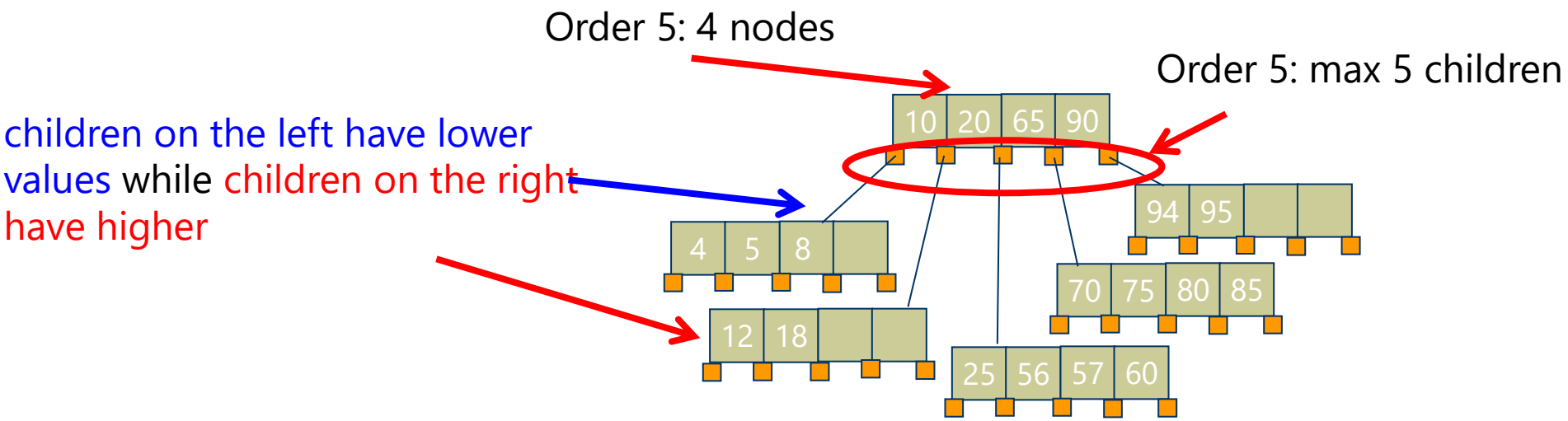
Bloque	cod A	nombre	agencia	fechaNacim
B1	88	Fele Martínez	Glam	22/02/75
	11	Najwa Nimri	Actors	14/02/72
B2	33	Nancho Novo	Rol	17/09/78
	22	Santiago Segura	Amigueta s	17/07/65
B3	77	Luis Tosar	BCN	13/10/71
	55	Candela Peña	Actors	14/07/73
B4	00	Maribel Verdú	Glam	02/10/70
	44	Penélope Cruz	BCN	28/04/74
B5	99	Javier Bardem	BCN	01/03/69

B-Tree Index

Deep balanced tree: all leaf nodes at the same depth. Each path from the root to a leaf has the same length.

N-order:

Each node can have up to n children and store n-1 keys.



Bitmap Index

Index on a particular column
Each value in the column has a bit vector: bit-op is fast
The **length** of the bit vector: **# of records** in the base table
The i-th bit is set if the i-th row of the base table has the value for the indexed column

Cust	Region	Type
C1	Asia	Retail
C2	Europe	Dealer
C3	Asia	Dealer
C4	America	Retail
C5	Europe	Dealer

RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Bitmap Index

Reduced response time for large classes of ad hoc queries: Flexible WHERE.

Reduced storage requirements compared to other indexing.

Fully indexing a large table with a traditional B-tree index can lead to structures several times larger than the data in the table. Bitmap indexes are typically **only a fraction of the size** of the indexed data in the table.

Efficient maintenance during parallel DML and loads.

The advantages of using bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small (1:100).

A table with one million rows, a column with 10,000 distinct values is a candidate

Comparison, union and aggregation use bit arithmetic.

Not suitable for high cardinality domains: too many columns (better B-Tree)

Often support compression

[Bitmap] Join Index

It materializes relational join in JI file and speeds up relational join

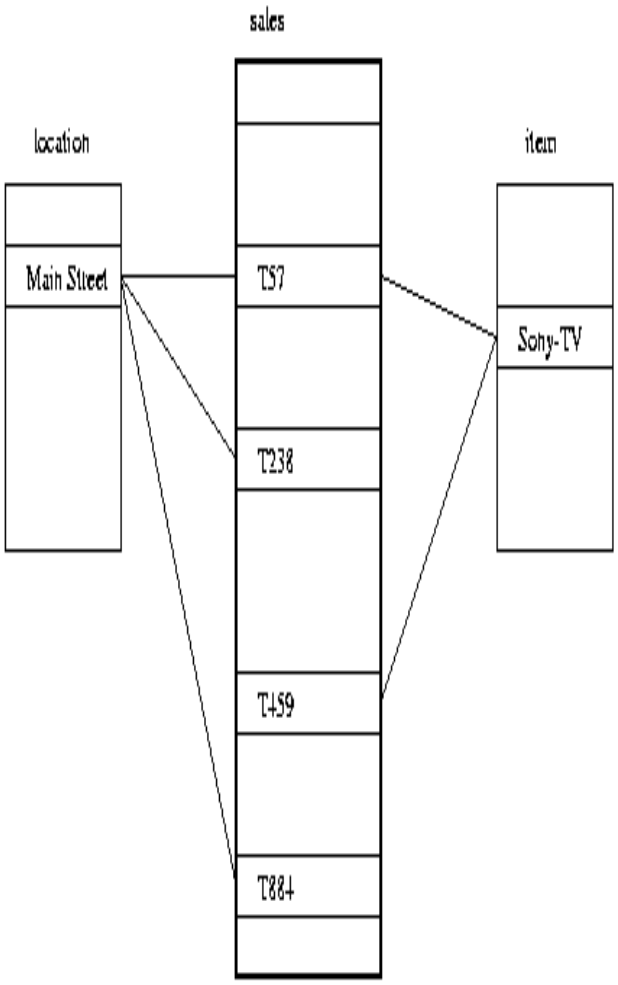
- Join index: **JI(R-id, S-id)** where R (R-id, …) JOIN S (S-id, …). R-fact table, S-dimension.

Better performance as it **precomputes join results** and stores them into bitmap format.

In DWH: join between facts and dimensions.

- Equi-inner join between PK and FK
- E.g. fact table: Sales and two dimensions city and product
  - A join index on city maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city
- Join indices can span multiple dimensions

NOTE: Related to denormalization. They precompute join results and reduce the need for real-time joins.



Compression

Used in **multidimensional** DBMS.
Saving **disk space**, increasing **memory efficiency**, and improving **query performa**nce by reducing the amount of data read from disk (multidimensional databases are usually massive).
Overhead for updating, deleting, and processing in general. Data needs to be uncompressed, modified and compressed, leading to **overload during write operations**. In DWH environments, write operations are **less frequent**.
Hybrid Columnar Compression: groups of rows are stored in columnar format, with the values for a given column stored and compressed together.
Storing column data together, with the same data type and similar characteristics, drastically **increases the storage savings** achieved from compression.

Materialized Views

Materialized views are query results that have been **stored in advance** so long-running calculations are not necessary when you actually execute your SQL statements.
They act like tables and improve performance in complex queries, especially those involving joins and aggregations.
Useful for aggregates.
Problem:

- **Resources** consumption
- Refreshing policies: **on demand, on commit**.

Efficient Data Cube Computation

Data cube can be viewed as a **lattice of cuboids**

- The bottom-most cuboid, with the most detailed data , is the **base cuboid**
- The top-most cuboid (**apex**) contains only one cell
- How many cuboids in an n-dimensional cube with L levels?

$$T = \prod_{i=1}^{n} (L_i + 1)$$

Materialization of data cube

- Materialize every cuboid (**full materialization**), none (**no materialization**, every query is computed from the base cuboid), or some (**partial materialization**)
- Selection of which cuboids to materialize
  - Based on size, sharing of cuboids between queries, access frequency, etc.

The "Compute Cube" Operator

Cube definition and computation in DMQL

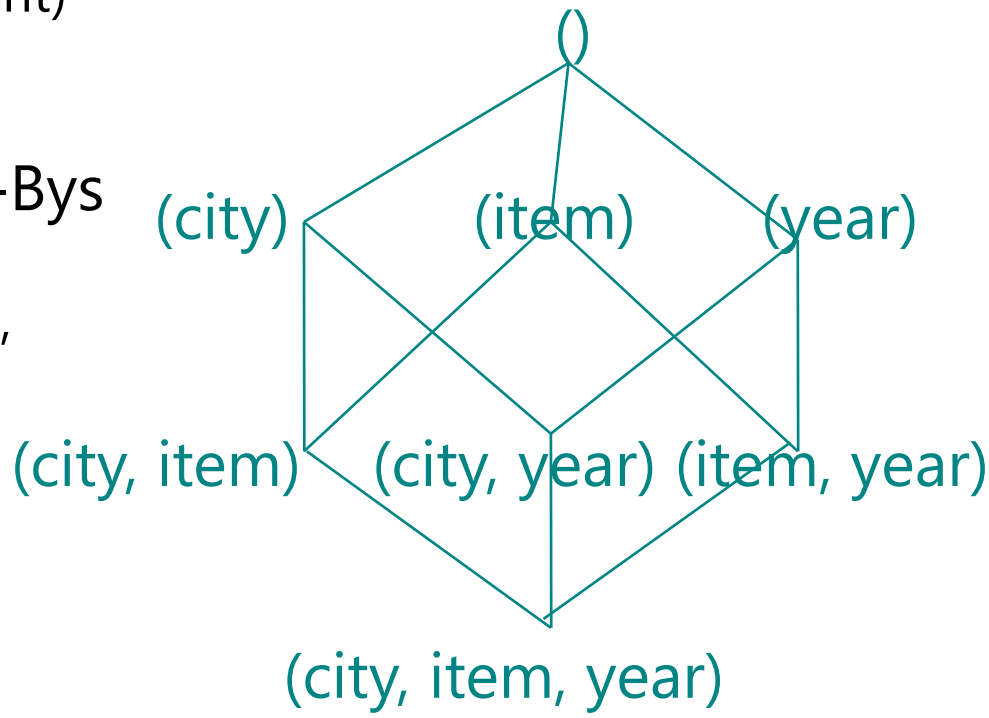
- define cube sales [item, city, year]: sum (sales_in_dollars)
- compute cube sales

Transform it into a SQL-like language (with a new operator cube by, introduced by Gray et al.'96)

```
SELECT item, city, year, SUM (amount)
FROM SALES
CUBE BY item, city, year
```

Need compute the following Group-Bys

- (item, city, year),
- (item,city),(item, year), (city, year),
- (item), (city), (year)
- ()



Efficient Processing OLAP Queries

Determine which operations should be performed on the available cuboids

- Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection

Determine which materialized cuboid(s) should be selected for OLAP op.

- Let the query to be processed be on {brand, province_or_state} with the condition "year = 2004", and there are 4 materialized cuboids available:
  1) {year, item_name, city}
  2) {year, brand, country}
  3) {year, brand, province_or_state}
  4) {item_name, province_or_state} where year = 2004

  Which should be selected to process the query?

Explore **indexing structures** and **sparse vs. dense** array structs in MOLAP

DWH-Data Lakes

DWH- Model for the flow of data from operational systems into decision support.

DWH+Big Data-> High maintenance costs, poor flexibility and difficult scalability.
DWH support **BI** and **Reporting**, but struggle with **ML**.

Data Lakes offer raw data repositories in multiple formats. Data processing frameworks: Hadoop (initial) -> Apache Spark (much more popular today).

DWH+Data Lakes: Multipurpose applications, such as SQL analytics, real-time monitoring, and ML. Cons: Additional complexity and multiple copies of data.

Data Lakes

Scalable raw data repository fitted for Big Data (structured, semi- and unstructured data).

Raw data ingestion. All data types.

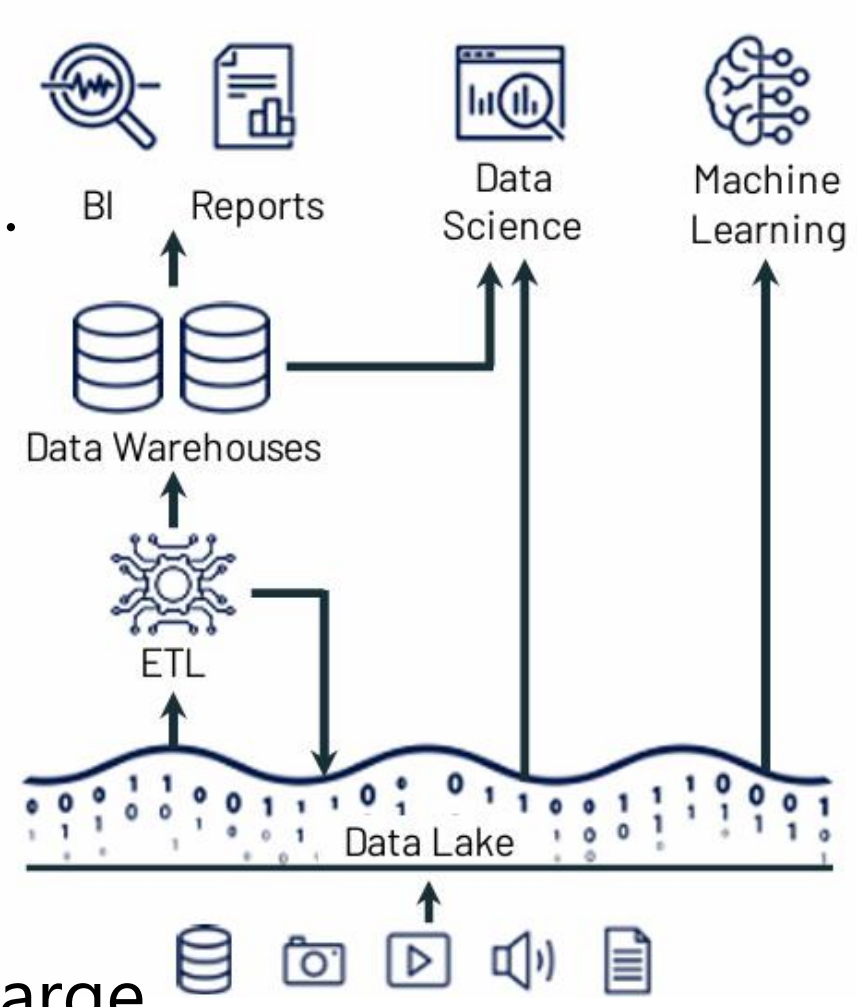
No ACID, **BASE** instead.

Data quality and governance:

Ideally-> Clean data repositories

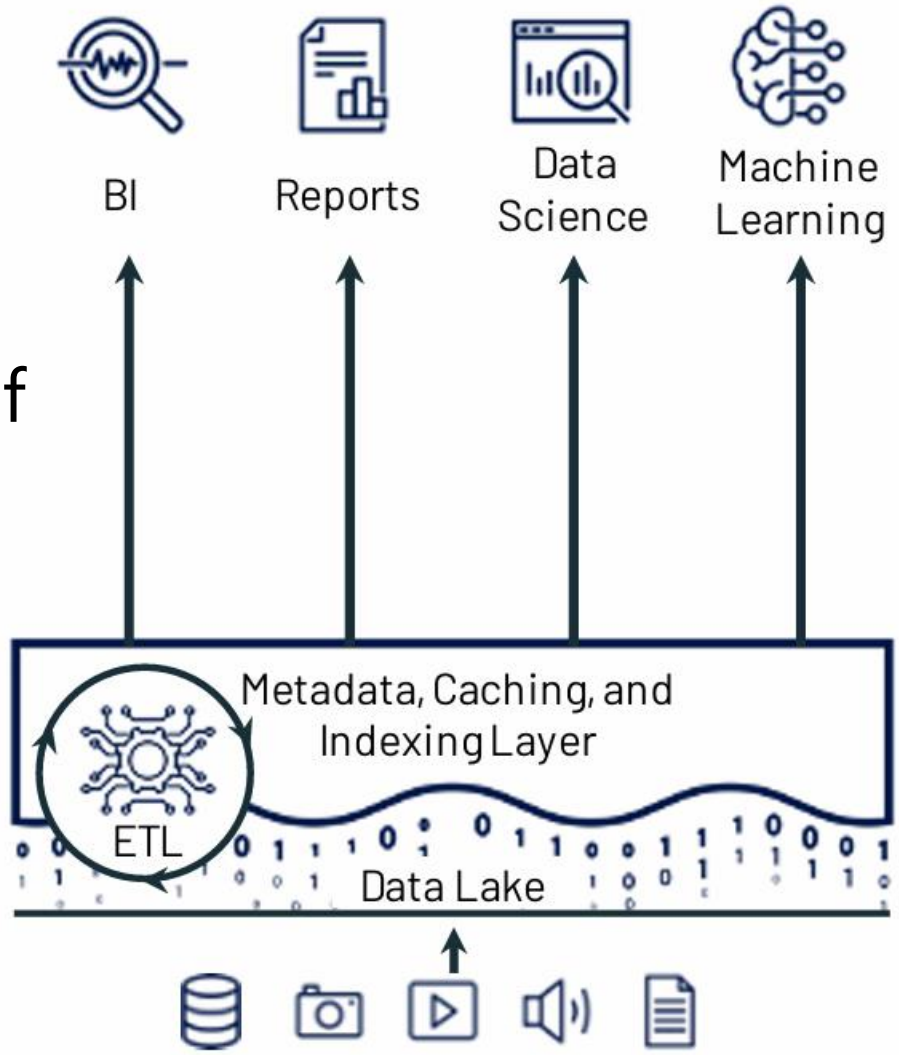
But, often: **data swamps**.

Challenges: High cost, data redundancy, complex architecture, security, difficult managements of large metadata, …



Data Lakehouses

A **data lakehouse** represents the synergy between a data warehouse and a data lake, offering the scalability and flexibility of data lakes with the ACID transactions and structure of data warehouses.

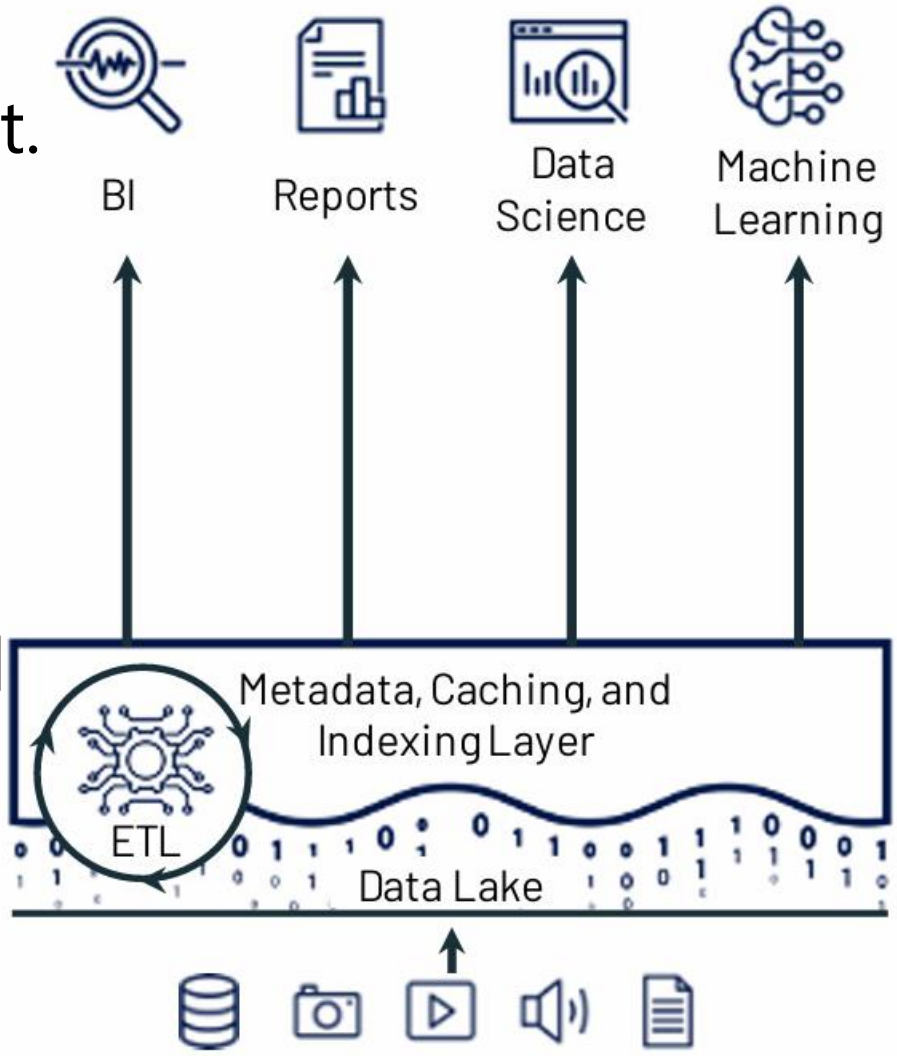


Simplifies data management. Provides a **single platform** for all forms of data analysis.

Core OS technologies: Delta Lake, Apache Iceberg, Apache Hudi.

Data Lakehouses

ACID transactions.
Access to **early versions** of data
Improved **metadata** management.
An **unique architecture** for all applications.
Lower costs.
Reduced data redundancy.
Direct connection by BI Tools.
Uses mostly **open standards and open formats**, suitable for batch and streaming data.
Enables **BI in all the data**.
The query engine is **connected directly** to the data lake (reduced ETL).



DEMO

Lakehouse

Delta-Lake

https://github.com/delta-io/delta-docker
git clone https://github.com/delta-io/delta-docker.git
cd delta-docker
Start Docker desktop
In powershell, run bash.
docker build -t delta_quickstart -f Dockerfile_delta_quickstart .
docker run --name delta_quickstart --rm -it --entrypoint bash delta_quickstart

Run a container from the image with a JupyterLab entrypoint:
docker run --name delta_quickstart --rm -it -p 8888-8889:8888-8889 delta_quickstart
Use the http to connect to Jupyter notebook:
http://127.0.0.1:8888/lab
Using python3 pykernel, select quickstart.ipynb

Delta-Lake

Write a Spark DataFrame to a Delta Lake table

```
data = spark.range(0, 5)

(data
  .write
  .format("delta")
  .save("/tmp/delta-table")
)
```

Read the above Delta Lake table to a Spark DataFrame and display the DataFrame

```
df = (spark
        .read
        .format("delta")
        .load("/tmp/delta-table")
        .orderBy("id")
      )

df.show()
```

```
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
+---+
```

```
part-00001-d39859ef-7163-47ac-b011-bacc9ccf04d0-c000.snappy.parquet
part-00001-d748a1c1-9a27-480b-92cb-ac7e9b1d5468-c000.snappy.parquet
part-00001-d78e6bcc-ce8f-4b0e-9d82-b690ae34c2cd-c000.snappy.parquet
part-00001-d797bb15-3260-4346-9ba4-6cb050ecdd94-c000.snappy.parquet
part-00001-d9e4a44f-8a6d-4658-bf10-80488c768f0f-c000.snappy.parquet
part-00001-dac72292-a701-424a-8a8a-81592206c08e-c000.snappy.parquet
part-00001-db87103b-50df-4f73-a0cd-a7e616f03e99-c000.snappy.parquet
part-00001-dc625a90-7e1b-4736-84e6-cc069e1b3415-c000.snappy.parquet
part-00001-df79a539-8255-404c-bc86-4778b3b57558-c000.snappy.parquet
part-00001-e56019b5-22f0-40b6-8523-6584d6094773-c000.snappy.parquet
part-00001-e87c5dfd-7821-45ab-b9d7-fe6fc369d0ff-c000.snappy.parquet
part-00001-ebc17974-a298-4e66-ae86-5221a0197b2f-c000.snappy.parquet
part-00001-ed0a69db-e7ea-451d-b635-3cfccb8c2be6-c000.snappy.parquet
part-00001-f08dd55d-264e-42cc-95d4-56109857bf55-c000.snappy.parquet
part-00001-f13ea236-0cd2-4ccd-adc8-37212977b40b-c000.snappy.parquet
part-00001-f477c049-933f-4e03-9dd4-1fdc0ead3318-c000.snappy.parquet
part-00001-f52ed9aa-8de5-44ce-95c8-7f75ddaa357e-c000.snappy.parquet
part-00001-f5a33748-f0d5-4673-bfc8-50b1760f81bb-c000.snappy.parquet
part-00001-fb9e43e2-afa9-405d-9b9f-d5c76963593f-c000.snappy.parquet
part-00002-29570c39-6caa-48c6-88e6-bdb111e1629c-c000.snappy.parquet
part-00002-6ad342e9-cfb5-4769-af72-b33a2013412c-c000.snappy.parquet
part-00002-85144acf-782a-44b2-bde8-0d18462afeff-c000.snappy.parquet
part-00002-8ea9cc0b-6c60-465f-b6cf-c468208dc14f-c000.snappy.parquet
part-00002-ba36c714-528c-4c66-904c-2143b4ad6441-c000.snappy.parquet
part-00002-c77e47d2-8f7d-4afb-bd2a-6539a1a78d2b-c000.snappy.parquet
part-00002-c89e9197-1da9-4b58-84ef-4ae62266e264-c000.snappy.parquet
part-00002-cbf84e9b-810a-421c-b36b-a1bf2a216e24-c000.snappy.parquet
```

Overwrite a Delta Lake table

Overwrite the Delta Lake table written in the above step

```
data = spark.range(5, 10)

(data
  .write
  .format("delta")
  .mode("overwrite")
  .save("/tmp/delta-table")
)
```

Delta-Lake

Showcase `update` feature of Delta Lake and display the resulting DataFrame

```
from delta.tables import *
from pyspark.sql.functions import *

delta_table = DeltaTable.forPath(spark, "/tmp/delta-table")

# Update every even value by adding 100 to it
(delta_table
  .update(
    condition = expr("id % 2 == 0"),
    set = { "id": expr("id + 100") }
  )
)

(delta_table
  .toDF()
  .orderBy("id")
  .show()
)
```

Showcase `merge` feature of Delta Lake and display the resulting DataFrame

```
# Upsert (merge) new data
new_data = spark.range(0, 20)

(delta_table.alias("old_data")
  .merge(
      new_data.alias("new_data"),
      "old_data.id = new_data.id"
      )
  .whenMatchedUpdate(set = { "id": col("new_data.id") })
  .whenNotMatchedInsert(values = { "id": col("new_data.id") })
  .execute()
)

(delta_table
  .toDF()
  .orderBy("id")
  .show()
)
```

Showcase `delete` feature of Delta Lake and display the resulting DataFrame

```
# Delete every even value
(delta_table
  .delete(
    condition = expr("id % 2 == 0")
  )
)

(delta_table
  .toDF()
  .orderBy("id")
  .show()
)
```

Delta-Lake

Display the entire history of the above Delta Lake table

```
# get the full history of the table
delta_table_history = (DeltaTable
                        .forPath(spark, "/tmp/delta-table")
                        .history()
                      )

(delta_table_history
   .select("version", "timestamp", "operation", "operationParameters", "operationMetrics", "engineInfo")
   .show()
)
```

```
+-------+--------------------+---------+--------------------+--------------------+--------------------+
|version|           timestamp|operation| operationParameters|    operationMetrics|          engineInfo|
+-------+--------------------+---------+--------------------+--------------------+--------------------+
|      4|2024-11-18 09:06:...|    MERGE|{predicate -> ["(...|{numTargetRowsCop...|Apache-Spark/3.5....|
|      3|2024-11-18 09:06:...|   DELETE|{predicate -> ["(...|{numRemovedFiles ...|Apache-Spark/3.5....|
|      2|2024-11-18 09:06:...|   UPDATE|{predicate -> ["(...|{numRemovedFiles ...|Apache-Spark/3.5....|
|      1|2024-11-18 09:06:...|    WRITE|{mode -> Overwrit...|{numFiles -> 6, n...|Apache-Spark/3.5....|
|      0|2024-11-18 09:05:...|    WRITE|{mode -> ErrorIfE...|{numFiles -> 6, n...|Apache-Spark/3.5....|
+-------+--------------------+---------+--------------------+--------------------+--------------------+
```

Time travel to the version `0` of the Delta Lake table using Delta Lake's history feature

```
df = (spark
        .read
        .format("delta")
        .option("versionAsOf", 0) # we pass an option `versionAsOf` with the required version number we are interested in
        .load("/tmp/delta-table")
        .orderBy("id")
      )

df.show()
```

```
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
+---+
```

Latest version of the Delta Lake table

```
df = (spark
        .read
        .format("delta")
        .load("/tmp/delta-table")
        .orderBy("id")
      )

df.show()
```

```
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
|  5|
|  6|
|  7|
|  8|
|  9|
| 10|
| 11|
| 12|
| 13|
| 14|
| 15|
| 16|
| 17|
| 18|
| 19|
+---+
```

Delta-Lake

A little bit of Streaming

```
streaming_df = (spark
                .readStream
                .format("rate")
                .load()
               )

stream = (streaming_df
            .selectExpr("value as id")
            .writeStream
            .format("delta")
            .option("checkpointLocation", "/tmp/checkpoint")
            .start("/tmp/delta-table")
          )
```

```
stream2 = (spark
            .readStream
            .format("delta")
            .load("/tmp/delta-table")
            .writeStream
            .format("console")
            .start()
          )
```

```
-------------------------------------------
Batch: 3
-------------------------------------------
+---+
| id|
+---+
| 11|
+---+

-------------------------------------------
Batch: 4
-------------------------------------------
+---+
| id|
+---+
| 12|
+---+
```

```
00000000000000001249.json                00000000000000001673.json
00000000000000001250.checkpoint.parquet  00000000000000001674.json
00000000000000001250.json                00000000000000001675.json
00000000000000001251.json                00000000000000001676.json
00000000000000001252.json                00000000000000001677.json
00000000000000001253.json                00000000000000001678.json
00000000000000001254.json                00000000000000001679.json
00000000000000001255.json                00000000000000001680.checkpoint.parquet
00000000000000001256.json                00000000000000001680.json
00000000000000001257.json                00000000000000001681.json
00000000000000001258.json                00000000000000001682.json
00000000000000001259.json                00000000000000001683.json
00000000000000001260.checkpoint.parquet  00000000000000001684.json
00000000000000001260.json                00000000000000001685.json
00000000000000001261.json                00000000000000001686.json
00000000000000001262.json                00000000000000001687.json
00000000000000001263.json                00000000000000001688.json
00000000000000001264.json                00000000000000001689.json
00000000000000001265.json                00000000000000001690.checkpoint.parquet
00000000000000001266.json                00000000000000001690.json
00000000000000001267.json                00000000000000001691.json
```

```
00000000000000000001.json
::::::::::::::
{"commitInfo":{"timestamp":1731920773177,"operation":"WRITE","operationParameters":{"mode":"Overwrite","partitionBy":"[]"},"readVersion":0,"isolationLevel":"Serializable","isBlindAppend":false,"operationMetrics":{"numFiles":"6","numOutputRows":"5","numOutputBytes":"2686"},"engineInfo":"Apache-Spark/3.5.1 Delta-Lake/3.1.0","txnId":"be8dc362-775c-4baf-9d6e-a5bac8e4b7ed"}}
{"add":{"path":"part-00001-d78e6bcc-ce8f-4b0e-9d82-b690ae34c2cd-c000.snappy.parquet","partitionValues":{},"size":478,"modificationTime":1731920772801,"dataChange":true,"stats":"{\"numRecords\":1,\"minValues\":{\"id\":5},\"maxValues\":{\"id\":5},\"nullCount\":{\"id\":0}}"}}
{"add":{"path":"part-00003-624fc40d-cb57-4595-b3ff-ac8f38a2be3d-c000.snappy.parquet","partitionValues":{},"size":478,"modificationTime":1731920772791,"dataChange":true,"stats":"{\"numRecords\":1,\"minValues\":{\"id\":6},\"maxValues\":{\"id\":6},\"nullCount\":{\"id\":0}}"}}
{"add":{"path":"part-00004-86c3b683-3276-4e76-b567-1b9a86c1e4ff-c000.snappy.parquet","partitionValues":{},"size":478,"modificationTime":1731920772791,"dataChange":true,"stats":"{\"numRecords\":1,\"minValues\":{\"id\":7},\"maxValues\":{\"id\":7},\"nullCount\":{\"id\":0}}"}}
{"add":{"path":"part-00006-c7763aa8-3d16-43dc-a8ca-95726c05acb6-c000.snappy.parquet","partitionValues":{},"size":478,"modificationTime":1731920772791,"dataChange":true,"stats":"{\"numRecords\":1,\"minValues\":{\"id\":8},\"maxValues\":{\"id\":8},\"nullCount\":{\"id\":0}}"}}
{"add":{"path":"part-00007-48b0cbd9-aae9-4b64-a0b5-e285d9e60b20-c000.snappy.parquet","partitionValues":{},"size":478,"modificationTime":1731920772801,"dataChange":true,"stats":"{\"numRecords\":1,\"minValues\":{\"id\":9},\"maxValues\":{\"id\":9},\"nullCount\":{\"id\":0}}"}}
{"remove":{"path":"part-00004-7ebe48bc-2c83-4228-985f-ae768968b371-c000.snappy.parquet","deletionTimestamp":1731920773176,"dataChange":true,"extendedFileMetadata":tr
```

Import CSV

[3]:
```
df = spark.read.format("csv").option("header", True).option("sep",";").load("/opt/spark/examples/src/main/resources/people.csv")
df.show()
```

```
+-----+---+---------+
| name|age|      job|
+-----+---+---------+
|Jorge| 30|Developer|
|  Bob| 32|Developer|
+-----+---+---------+
```

[4]:
```
df.write.format("delta").save("tmp/people_delta")
!tree tmp/people_delta
```

```
24/11/18 11:54:57 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
[Stage 8:=============================================>          (41 + 8) / 50]
tmp/people_delta
├── _delta_log
│   └── 00000000000000000000.json
└── part-00000-ad0ac088-e4af-41f2-9013-83d18490580d-c000.snappy.parquet

1 directory, 2 files
```

[5]:
```
spark.read.format("delta").load("tmp/people_delta").show()
```

```
+-----+---+---------+
| name|age|      job|
+-----+---+---------+
|Jorge| 30|Developer|
|  Bob| 32|Developer|
+-----+---+---------+
```