

Bases de Dades Avançades

Carlos Arbonés and Juan P. Zaldivar

GCED, UPC.
Apunts.

Contents

1 Data Warehousing	6
1.1 Operational vs Analytical Data	6
1.2 Single-layer architecture	6
1.3 Data Sources	7
1.4 Definition of Data Warehouse	7
1.5 Data Marts	7
1.6 Two-layer architecture	8
1.7 Three-layer architecture	9
1.8 Metadata	9
2 Multidimensional modeling	10
2.1 FASMI test	10
2.2 Real World	11
2.3 Conceptual World	12
2.3.1 Types of schemas	12
2.4 Representation World	13
2.4.1 MOLAP: Multidimensional Matrix	13
2.4.2 ROLAP	13
2.4.3 Comparison of Desing Steps: OLTP vs OLAP	14
2.4.4 HOLAP	15
3 Advanced Multidimensional Design	16
3.1 Algebraic operations	16
3.2 Advanced Design	16
3.3 Aggregation Problems	18
4 Extract, Transform and Load	19
4.1 Definition	19
4.1.1 Extraction	19
4.1.2 Transformation	19
4.1.3 Load	20
4.2 ETL Steps	21
4.2.1 Data extraction mechanisms	22
4.2.2 Kinds of transformation tasks	23
4.3 ETL process design	24
4.3.1 ETL tools GUI	24
4.3.2 Hand-coded ETL	24
4.3.3 ETL process quality	25
4.4 Architectual setting	25
4.4.1 Data flow	25
4.4.2 Control flow	26
4.4.3 Staging area	26
4.4.4 Metadata managment	26
4.5 ETL operations	27
4.5.1 Non-Blocking Example	28
4.5.2 Blocking Example	28
4.5.3 Blocking optimized Example	29
4.5.4 List of blocking and Non-blocking operations	29
5 Data Quality	30
5.1 Data Conflicts	31

5.2	Quality Measures	31
5.2.1	Completeness	31
5.2.2	Accuracy	32
5.2.3	Timeliness (Freshness)	32
5.2.4	Consistency	32
5.2.5	Trade-offs between dimensions	33
5.3	Quality Rules	33
5.3.1	Restricciones de integridad en RDBMS	33
5.3.2	Problems in the rules	34
5.3.3	Fine-tuning imperfections	34
5.4	Quality improvement	35
5.4.1	Process for Object identification	35
5.4.2	Search space reduction	36
5.4.3	Comparison/Similarity function	36
6	Schema and Data Integration	37
6.1	BD distribuida	38
6.2	Kinds of Heterogeneity	39
6.2.1	Schema and instances	39
6.2.2	Intra-class heterogeneity	39
6.2.3	Inter-class heterogeneity	40
6.2.4	Data-Metadata heterogeneity	40
6.3	Overcoming System Heterogeneity	41
6.4	Schema Integration	41
6.4.1	Alignment Process	41
6.4.2	Mapping	42
6.4.3	Examplre of Global as View (GAV)	43
6.5	Data Integration	44
6.5.1	Entity resolution	44
6.5.2	Algoritmo R-Swoosh	44
7	Distributed Data Management	45
7.1	Dsitributed Systems	45
7.1.1	Escalabilidad	45
7.1.2	Performance/Efficciency	46
7.1.3	Reliability/Availability	46
7.1.4	Concurrency	46
7.1.5	Transparency	46
7.2	Distributed Database Systems	47
7.2.1	Extended ANSI-SPARC Architecture of Schemas	48
7.2.2	Centralized DBMS Functional Architecture	50
7.3	Cloud Databases	51
7.3.1	Parallel database architectures	51
7.4	(Distributed) Data Design	53
7.4.1	Data Fragmentation	53
7.4.2	Data Allocation	54
7.4.3	Data Replication	54
7.5	Distributed Catalog Managment	54
8	Distributed Data Processing	55
8.1	(Distribure) Transaction Management	55
8.1.1	CAP theorem	55

8.1.2	Managing replicas	56
8.1.3	Replication Management Configuration	56
8.2	(Distributed) Query Processing	57
8.2.1	Allocation Selection	58
8.2.2	Phases of distributed query processing	58
8.3	Bulk Synchronous Parallel Model	60
8.3.1	Kinds of parallelism	60
8.3.2	Demand-Driven Pipelining	61
8.3.3	Producer-Driven Pipelining	62
8.4	Measures of Parallelism	62
9	Distributed Processing Engine (MapReduce)	63
9.1	MapReduce as a DDBMS component	63
9.2	Components and Use	64
9.3	Relational algebra in MapReduce	65
9.3.1	Projection	65
9.3.2	Cross Product	65
10	Distributed Processing Engine (Spark)	67
10.1	Background	67
10.2	Dataframes	68
10.2.1	Characteristics of dataframes	68
10.2.2	Relation, DataFrame, and Matrix	68
10.2.3	Spark Dataframe definition	69
10.2.4	DataFrame vs Matrix/Array/Tensor	69
10.2.5	DataFrame vs Relation/Table	69
10.2.6	Dataframe implementations	70
10.2.7	Operations	70
10.2.8	Optimizaciones	71
10.3	Abstractions	72
10.3.1	Spark SQL	72
10.3.2	Spark SQL interface	73
10.3.3	Shared Optimization and Execution	73
11	Big Data Architectures and Model Management	75
11.1	Data Management Backbone	76
11.1.1	Data Management	76
11.1.2	1st Generation: Data Warehouses	77
11.1.3	2nd generation: Data Lakes	77
11.1.4	Heterogeneity of data formats	78
11.1.5	3rd gen: Cloud Data Lakes	78
11.1.6	4th gen: Lakehouses	79
11.2	Data Analysis Backbone	79
11.3	ML Lifecycle Management	79
11.3.1	AutoML Overview	79
A	Theoretical questions	80
B	OLAP and the Multidimensional Model	82
B.1	Theoretical questions	82
B.2	Problems	83
C	Extraction, Transformation and Load	91

C.1	Theoretical questions	91
C.2	Problems	91
D	Data Quality	93
D.1	Problems	93
E	Schema and Data Integration	99
E.1	Theoretical questions	99
E.2	Problems	99

1 Data Warehousing

1.1 Operational vs Analytical Data

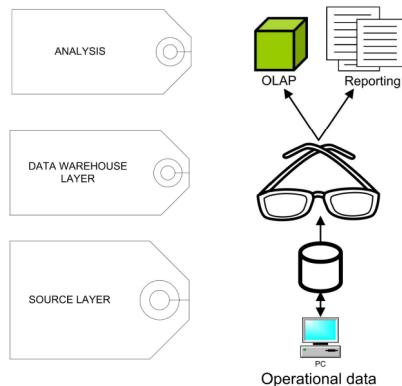
Lo primero que tenemos que darnos cuenta es separar entre datos operacionales y datos analíticos. Los datos operacionales son los del día a día de la empresa, los que utilizamos para gestionar el negocio. Los datos analíticos son los que utilizamos para tomar decisiones respecto nuestro negocio (son de naturaleza temporal).

- **Evolución:** para ver el día a día solo nos importan los datos de hoy. En cambio para tomar decisiones necesitamos datos antiguos y gestionar la evolución que han sufrido en distintos ámbitos de la empresa.
- **Volúmenes de datos:** el hecho de guardar datos históricos para analizar su evolución hace que crezca el volumen de datos.
- **Niveles de agregación:** Por otro lado, cuando tomamos decisiones no miramos los datos de un solo cliente sino que miramos los datos agregados. P.e los datos de nuestros clientes en una cierta región, en un cierto año, etc.
- **Actualización:** Los datos no tienen porqué estar completamente actualizados. Lo que nos interesa es ver la evolución histórica y por un dato que falte no pasa nada grave. En cambio para datos operacionales si que es importante. P.e no podemos sacar dinero de un banco si no tenemos el saldo de ayer actualizado.
- **Tiempo de respuesta:** Tenemos muchos datos, lo que impacta al tiempo de respuesta. Siempre es importante pero a ciertas escalas es más crítico. Menor el número de usuarios.
- **Usuarios:** Normalmente, el tipo de usuarios que tenemos en un sistema decisional/de análisis son los ejecutivos de la empresa.
- **Funcionalidad:** Operacional o de análisis.

1.2 Single-layer architecture

Tenemos datos operacionales y a partir de estos generamos una vista (nivel virtual) que nos permite alimentar nuestras herramientas OLAP o de *query reporting*. Los datos pasan por un filtro que los reorganiza de la forma más adecuada para estas herramientas.

Únicamente tenemos nuestras fuentes de datos operaciones, no generamos una réplica de estos datos. Esto facilita la gestión y reduce costes (menos infraestructura), el problema es que no guardamos esta réplica de los datos (el histórico se puede ver comprometido en mayor medida según los **data sources** que tengamos).



En general ocasiona problemas a largo y corto plazo. Por eso se opta en la mayoría de los casos por un **Data Warehouse**.

1.3 Data Sources

De más comunes/frecuentes a menos tenemos:

1. **Propios sistemas operacionales de la empresa:** aplicaciones propias que tienen los datos sobre nuestro negocio (RRHH, contabilidad, logística), son conocidas por nosotros y tenemos el control sobre los cambios.
2. **Sistemas de información de nuestros socios:** ya sea de nuestros clientes/proveedores. Podemos llegar a acuerdos de cómo gestionar los datos. No tenemos todo el control pero tenemos una cierta capacidad de negociar como mínimo.
3. **Red Informática Mundial y Redes Sociales:** no tenemos ningún control, puede cambiar cualquier cosa (lo cual es normal). Cada vez que tenemos que tomar una decisión tenemos que ir a esa *web*, pero puede ser que el dato que queremos ya no esté disponible. Lo cual puede ser perjudicial en el caso de Single-layer al no realizar réplicas.
4. **Fuente no digitalizada:** o bien los pasamos a mano los datos, con escáner o técnicas especializadas de reconocimiento de caracteres. De nuevo, si solo se mantiene la arquitectura de un solo nivel, la digitalización de estos documentos se puede perder al no realizar réplicas.

1.4 Definition of Data Warehouse

Un Data Warehouse es una base de datos que sirve para **tomar decisiones**. Esta base de datos debería tener **4 características**:

- **Orientada a un tema:** se trata de poner juntos los datos de un mismo *tema* (*clientes, ventas, productos, etc.*) para mejorar el rendimiento (consultas más sencillas). No contiene los datos de todos los ámbitos de la empresa. Es lo contrario a orientado a funcionalidad.
- **Integrada:** tenemos diferentes tipos de datos (data sources) y los metemos en una única BD de manera que los datos se puedan cruzar.
- **Variante en el tiempo:** se refiere a la capacidad de una base de datos para mantener un registro histórico de los datos a lo largo del tiempo. En otras palabras, una base de datos variante en el tiempo permite almacenar y acceder a datos que cambian con el tiempo y mantener un seguimiento de las modificaciones y actualizaciones realizadas en esos datos.
- **No volátil:** los datos almacenados en la base de datos no se pierden ni se eliminan de manera accidental o automática. Los datos son persistentes y se mantienen en la base de datos hasta que se realice una acción específica para eliminarlos.

En este sentido, **Data Warehousing** es más un proceso que un producto. La filosofía de gestionar los datos. No es único sistema, sino es un conjunto que se tienen que interconectar para poder tomar decisiones.

1.5 Data Marts

Son pequeños almacenes de datos (Almacenes de datos departamentales/proyectos).

- Orientados a análisis.
- Normalmente, diseño multidimensional.
- Historial parcial, no hace falta almacenar datos muy antiguos.
- No necesita todas las fuentes de datos.

- No necesitan la granularidad más fina. Uso de datos mensuales, trimestrales, etc.
- Permiten una reducción del coste.

El error que podemos cometer con esto es que en lugar de tener un único proyecto generamos proyectos independientes (*Data Mars* completamente independientes). La manera correcta de hacerlo sería no sustituir el DW corporativo por los *Data Mars*, sino que **además de** tener el DW corporativo generar pequeños *Data Mars* para cada departamento/proyecto, ya que son más fáciles de gestionar de manera individual y tienen menos coste.

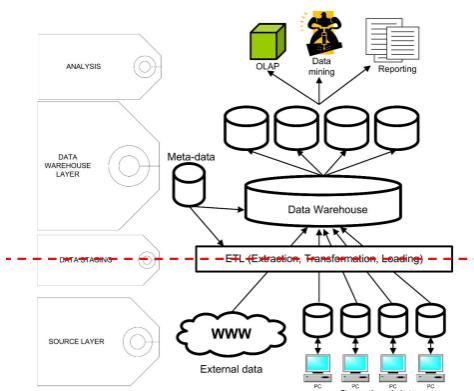
Por lo tanto lo que hacemos es crear un único repositorio donde cargamos todas las fuentes después de haberlas limpiado e integrado **únicamente una vez**. Cuando tenemos esta única fuente de conocimiento para la empresa, a partir de aquí generamos las diferentes **vistas** (*Data Mars*) para los departamentos/proyectos de análisis que tengamos.

Characteristics	Data warehouse	
	Departmental	Corporate
Subjects	Specific	Generic
Data sources	Few (some)	Many (all)
Size	Gigabytes	Terabytes
Development time	Months	Years
Data model	Multidimensional	Relational (file system)

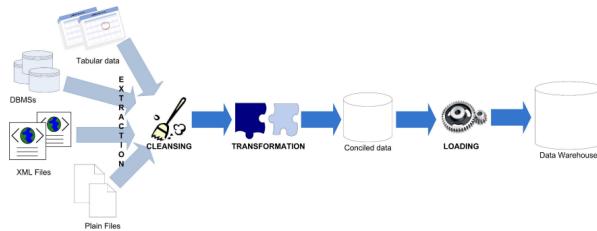
1.6 Two-layer architecture

Tenemos un **primer nivel** que son las **fuentes de datos operacionales y posibles datos externos**. A partir de aquí tendremos un **segundo nivel** que seria el **nivel decisional** donde tenemos nuestro **almacén de datos corporativos** con todos sus **Data Mars** y las herramientas de Análisis.

Añadimos un paso intermedio de **ETL** (Extract Transform Load) donde leemos los datos de estas fuentes operacionales (web, redes), **los limpiamos, los integramos** y los cargamos todos en este almacén de datos **orientado a temas, integrado, no volátil y variable con el tiempo (histórico)**, de manera que si estas fuentes de datos se pierden o cambian nosotros no perderemos estos datos.



El **ETL** tiene que **extraer** todos los datos de las diferentes fuentes, los va a **limpiar** (mirar si hay contradicciones entre las diferentes fuentes y cruzar unas con otros y ver si se pueden complementar), transformar (P.e normalización) y finalmente **conciliar**, ponerlos todos juntos y mirar que tengan sentido. Finalmente **cargaríamos** estos datos en nuestro almacén de datos, en nuestra fuente única de verdad de la empresa, a partir de donde definiríamos los Data Marts.

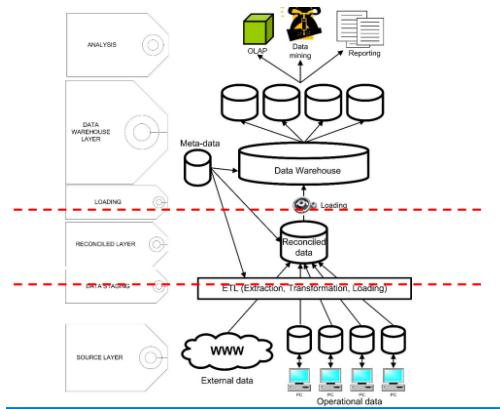


1.7 Three-layer architecture

En este caso tenemos 3 niveles. Mantenemos los dos niveles de antes (operacional y decisional) y introducimos un paso intermedio que son los **Datos Reconciliados**. Aquí, el ETL hace las tareas de antes pero para no molestar (reducir el coste) del DW no cargamos directamente los datos en este porque puede perjudicar la gestión y rendimiento de la BD, lo hacemos en dos pasos.

La carga del ETL la hacemos en una base de datos **temporal** (en el sentido que luego se va a borrar) y entonces una vez tengamos la **extracción, transformación y limpieza** de todas las fuentes de datos ahora en un único paso los volcaríamos en el DW todos de golpe y en un momento más adecuado (P.e por la noche o vacaciones).

Con este ganamos simplicidad de la arquitectura (dividir y vencer). Además estos datos reconciliados están disponibles mucho antes y el usuario puede consultarla también. Si nos esperamos a tener estos datos en el DW puede ser tarde para hacer según que tipo de cosas. Puede ser una ventaja competitiva para el negocio.



1.8 Metadata

Son datos más abstractos, datos que nos permiten **interpretar** otros datos. No hay que confundirlo con *derivados, summarizados* o *agregados*, que son datos que se obtienen **a partir** de otros datos. Los metadatos son datos que **hablan de otros datos**. Algunos ejemplos de metadatos son: *fuentes de datos, documentación general, esquemas, etc.*

2 Multidimensional modeling

Una de las herramientas más utilizadas son las hojas de cálculo. Sirven muy bien para ciertas tareas pero tienen limitaciones.

- **No gestionan meta-datos:** Las filas y columnas se identifican con números y letras pero estos no tienen significado y su interpretación y operación puede resultar complicado. También su consulta.
- **Espacio limitado:** El número de filas y columnas es limitado y por lo tanto el número de celdas también. En la práctica es limitado.
- La posición de los datos limita las operaciones que podemos hacer.
- **Operaciones de agregación es limitada**, como p.e. querer agrupar clientes por distritos, el tiempo en meses y años, etc.

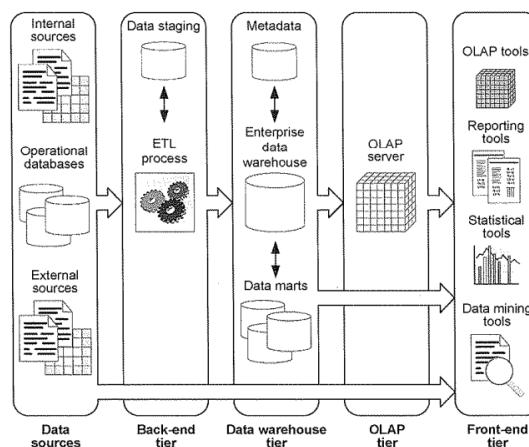
2.1 FASMI test

Cinco características que se tienen que cumplir las herramientas para el análisis de datos.

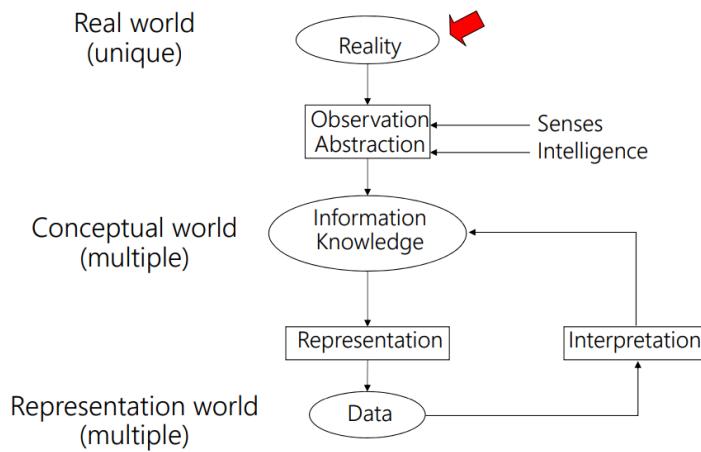
- (F) **Rápidas** independientemente del volumen de los datos.
- (A) Tienen que **permitir** hacer **análisis de los datos**
- (S) Se tiene que permitir interactuar de forma **concurrente** entre diferentes usuarios.
- (M) Multidimensional, las dimensiones no deberían estar limitadas.
- (I) Se debería permitir gestionar información y no solo datos. Para interpretar el contenido y su significado.

Las hojas de cálculo **solo cumplen** las **primeras dos características**. Permiten solo 3 dimensiones: filas, columnas y hojas. No permiten gestionar meta-datos y por lo tanto no permiten gestionar información.

Las bases de datos **operacionales** cumplen la compartición (concurrencia de diferentes usuarios) y la información (meta-datos mediante el uso de catálogos). La rapidez depende de las consultas. No están pensadas para hacer análisis, sino para consultas y modificación. Además no entienden el concepto de multidimensionalidad.



2.2 Real World



El mundo real es único en el modelo multidimensional, **solo hay un mundo que corresponde al mundo de nuestros datos** que queremos analizar. Pero se puede representar de varias maneras. En nuestro caso la representación sera en un **hipercubo**. Se pueden hacer operaciones con los hipercubos, como:

- **Slice:** Fijar un valor en una de las dimensiones. Deshacerse del resto de las dimensiones. Si se generaliza se pueden fijar varios valores en varias dimensiones y sería la operación *dice*.
- **Dice:** Se fijan varios valores en varias dimensiones. El resultado es una especie de dado.
- **Sort:** Aplicar una ordenación en una de las dimensiones. Por ejemplo poner en orden alfabético los atributos de esa dimensión.
- **Pivot:** Reordenar las dimensiones. Los datos del cubo no cambian, solo cambia como las visualizamos.
- **Roll-up:** Movernos en las jerarquías de agregación (*mapeos/correspondencias entre elementos de una misma dimensión p.e. Paris y Lyon los juntamos en una sola etiqueta llamada Francia y agrupamos sus valores*). Agrupación de los valores/datos dependiendo de un mapeo. Llevando esto al extremo podemos juntar los valores de toda una dimensión de manera que habrá una dimensión menos. Pérdida de detalle.
- **Drill-down:** Operación contraria de *Roll-up*. División de elementos en subelementos según jerarquías de agregación de una misma dimensión. Desglose de los datos. (*p.e Q1 se puede desglosar en Enero, Febrero y Marzo*).
- **Drill-across:** Operación binaria. Se necesitan dos cubos a la entrada de la operación. Join de cubos en lugar de tablas relacionales. Las dimensiones de los cubos tienen de coincidir para poder realizar la operación.
- **Union:** Se requiere que las dimensiones sean las mismas de los cubos de entrada. Pero los valores de las dimensiones no tienen porque coincidir. Equivalente a la unión de conjuntos.
- **Difference:** Equivalente a la diferencia de conjuntos. Quitar las celdas que coinciden en los cubos de entrada.

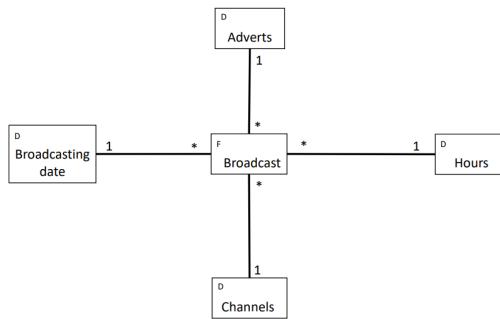
2.3 Conceptual World

Como se conceptualizan los hipercubos de datos? Hay **diferentes maneras de entender los datos conceptualizados**, porque se pueden interpretar de formas diferentes. Una manera de conceptualizar el mundo real es la notación **UML**.

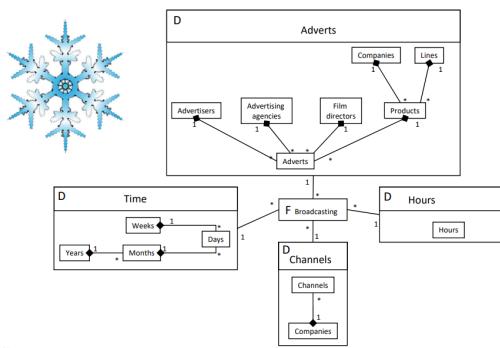
El modelo multidimensional de los cubos se centra en focalizar la información que nos interesa. De todas las interrelaciones, nos centramos en las que nos interesa y hacemos un esquema.

2.3.1 Types of schemas

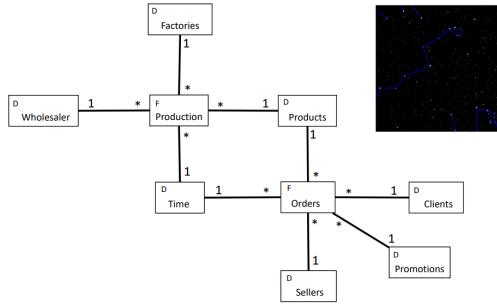
- **Star schema:** En el centro se coloca lo que se desea analizar y a los alrededores ponemos las diferentes dimensiones que tendrá el cubo para analizar el concepto central. Entre el concepto central y los extremos siempre hay una relación de uno a muchos 1 — — *. La ventaja es que selecciona de todo el dominio solo lo que nos interesa y de todos los tipos de relación solo nos fijamos en asociaciones de uno a muchos 1 — — *.



- **Snowflake:** Se ven el hecho central en medio y las dimensiones a los alrededor. En diferencia, es que además de las dimensiones se muestran las jerarquías de agregación. Es posible entonces realizar **Roll-up's**. Simplemente es dar más información al usuario.



- **Galaxy or Constellation:** En lugar de una sola estrella, se tendrían estrellas conectadas entre ellas mismas. Con dimensiones comunes, lo que facilita **Drill-across** de una estrella a otra.



2.4 Representation World

Se puede representar de formas diferentes pero siempre interpretables para obtener información para la toma de decisiones.

2.4.1 MOLAP: Multidimensional Matrix

(*Multi-dimensional Online Analytical Processing*)

La primera opción de representación es directamente agarrar el cubo y serializarlo. Estirarlo y ponerlo en fila. El disco no entiende de dimensiones, solo guarda 1 bit detrás del otro. Que se ajusta a la interpretación de los datos desde la memoria del disco. Se estira mediante hacer *slices*, hasta tener *slices* de dimensión asequible para poder almacenarlas de forma serial una detrás de la otra (2D).

La ventaja es que si se quiere recuperar todas los datos correspondientes a un mismo valor serán de fácil acceso porque están guardadas de forma serial (estarán almacenadas juntas dentro del disco). Pero dependiendo de la búsqueda los datos pueden no quedar a un solo acceso a disco, lo cual implicaría que las queries sean más costosas. Lo mismo pasa con las actualizaciones o inserciones que implicaría en algunas ocasiones regenerar todo el cubo. Desbalance entre queries muy rápidas y queries muy costosas. Si sabemos que tipos de consultas haremos o tenemos pocas dimensiones, puede resultar beneficioso.

2.4.2 ROLAP

(*Relational Online Analytical Process*)

La solución puede ser usar **herramientas relacionales**, donde guardar los datos de los cubos sería en un SGBD relacional y se necesitaría una capa de traducción que produzca la vista en forma de cubos. Funciona con lenguaje SQL y es fácil de obtener. El rendimiento de estas herramientas no es de lo más eficiente porque fueron concebidas para **OLTP** (*Online transactional process*) y lo que requerimos son **OLAP** (*Online Analytical process*). Cuando intentamos hacer esto usamos muchos *joins*, que son una operación muy costosa.

Star-join schema

En la traducción lo que haríamos es tener una tabla de hechos en el centro donde guardaríamos todas las celdas y las tablas de dimensión alrededor, una por cada dimensión. La tabla de hechos tiene *foreign keys* que apuntan a cada una de las dimensiones, estas llaves forman la *primary key* de la tabla de hechos. En las dimensiones tendríamos las jerarquías de agregación pero aplazadas.

Puede haber redundancia en los datos, por ejemplo en una tabla de *Time* los meses, trimestres y años se repiten mucho. Lo que se podría hacer es crear 3 tablas aparte y en

la de *Time* solo guardar el día, para que la información no se repita. Pero en este caso no es una buena idea, ya que la redundancia no es tanto como se puede pensar. Esto sucede porque la tabla de hechos siempre será de dimensiones mucho mayores que las tablas de dimensión y dominará el coste y el espacio, por lo tanto no es relevante por el volumen total que se tiene. Además el número de *joins* se incrementa. Lo cual también implica la simplificación del **Cube Query** (Patrón de consulta).

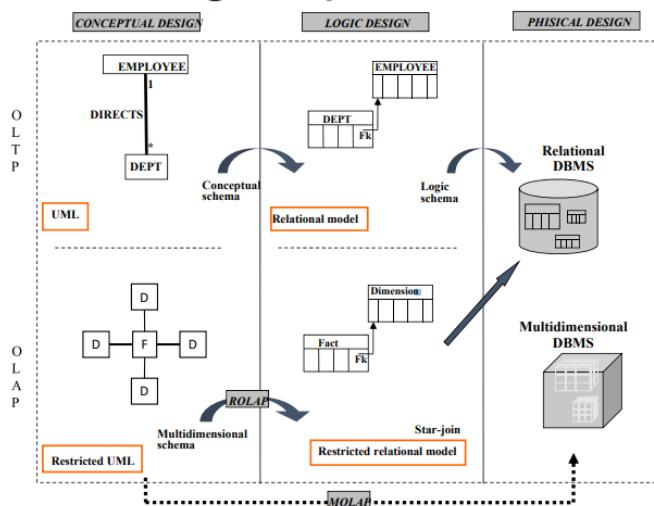
Cube-Query

```
SELECT d1.attr, ..., dn.attr, F(f.Measure1), ...
FROM Fact f, Dimension1 d1, ..., Dimensionn dn
WHERE f.key1 = d1.ID AND ... AND f.keyn = dn.ID AND <slice-dice condition>
GROUP BY d1.attr, ..., dn.attr
ORDER BY d1.attr, ..., dn.attr
```

```
SELECT h.name AS Hospital , t.month AS Month, AVG(cost)
FROM Admissions a, Hospitals h, Time t
WHERE a.HospitalId=h.Id AND a.TimeId=t.id AND t.year=2022
GROUP BY h.name, t.month
ORDER BY h.name, t.month
```

Los atributos del $\text{SELECT } d_i$ son los mismos que los atributos del ORDER BY y GROUP BY . Al saber que todas las consultas multidimensionales siguen este mismo patrón de consulta, esto facilita el trabajo del gestor relacional.

2.4.3 Comparison of Desing Steps: OLTP vs OLAP



En el caso de diseño OLTP creamos un esquema UML sin ningún tipo de restricción, cualquier tipo de relación entre las clases con cualquier tipo de multiplicidad, y esto lo traducimos al modelo relacional. Lo traducimos como ya sabemos: con tablas, llave primaria y foránea, etc. Finalmente esto se va a un SGBD relacional.

Cuando tenemos una tecnología OLAP seguimos un patrón de estrella con la tabla de hechos en medio y las tablas de dimensiones alrededor, solo se admiten relaciones $1 - - - *$. No podemos tener cualquier esquema UML sino sólo esquemas en formato estrella. No podemos traducirlo de cualquier manera sino con una tabla de hechos con una llave foránea para cada una de las dimensiones. Y todo éste se almacena en tablas relacionales y las consultas se realizan con SQL.

La alternativa es saltarnos el paso de traducción y directamente traducir del esquema de estrella, serializarlo, no guardarlo en SGBD relacional i guardarlo directamente en el disco con tecnología específica para serializar matrices y utilizando técnicas de indexación para recuperar fácilmente los datos.

Cuando tenemos pocas dimensiones o sabemos las consultas, serializar es normalmente mucho mejor. Cuando tenemos muchas dimensiones o no podemos saber las consultas que tendremos lo más seguro es apostar por un SGBD relacional. Lo que sucede normalmente es que ni estamos en un caso ni en el otro y tendremos que utilizar es un sistema híbrido.

2.4.4 HOLAP

Lo que se hace es mezclar las dos cosas, serialización de matices por un lado y sistemas relacionales por el otro.

- Los **cubos que son más densos** (con muchas celdas, sin huecos) los ponemos con una herramienta MOLAP porque serializar un cubo denso en mucho mejor porque no deja espacios. Cuando tenemos un **cubo más disperso** con espacios utilizaremos ROLAP que no deja vacíos, cuando no hay información no se guarda la fila.
- Los **datos atómicos** (datos que no se pueden dividir más) mejor guardarlos ROLAP ya que son más dispersos pero cuando comencemos agregar los cubos, estos son mas densos y sale mas a cuenta MOLAP.
- Si tenemos **datos que son accedidos muy frecuentemente** y sabemos las consultas más utilizadas mejor serializar de forma adiente para recuperarlas de forma rápida. Las que no tenemos claro cuándo y como se accederán las ponemos con una herramienta ROLAP.

3 Advanced Multidimensional Design

3.1 Algebraic operations

Conjunto de operaciones algebraicas y como pueden operar en un cubo de datos. Como se transforma la consulta SQL. Hay diversas operaciones:

- **Selection (<cube>, <predicate on dimensions>)**: Nos permite hacer *slices* en *dice* dada una condición en una dimensión.
- **Projection (<cube>, <measures left>)**: -Nos permite eliminar medidas del cubo.
- **Roll-up (<cube>, <destination level>[, <aggregation function>])**: Subir jerarquías de agregación, cuando subimos, como estamos agrupando los datos tenemos que indicar que función (como es opcional, por defecto es la SUM).
- **Drill-Down (<cube>, <destination level>)**: Bajar la jerarquía de agregación.
- **Drill-Across (<cube>, <new fact>[, <aggregation function>])**: Nos permite hacer *joins* entre cubos (como es opcional la función de agregación, por defecto es la SUM).
- **ChangeBase (<cube>, <new base>)**: Nos permite reordenar las dimensiones o los datos dentro de las dimensiones. Hay que indicar con qué dimensiones nos quedamos y el orden de estas. Cuando se usa para eliminar una dimensión esta debe ser una constante.

Mirar [video](#) para ejemplo.

3.2 Advanced Design

Vamos a ver técnicas que nos ayudan a reflectar la realidad y algunos casos especiales que tráctaremos.

1. **Factless Facts**: El primero es el caso en que la **tabla de hechos no tiene ningún atributo**, no tiene medidas. Aun así podemos seguir **contando** hechos. Hay casos donde interesa **registrar algún evento** como *p.e la asistencia de los estudiantes en las clases. Entonces para cada estudiante, para cada clase y franja horaria pondríamos una fila en la tabla de hechos y dejaríamos constancia de esto. Esto nos permitiría contar el número de asistencias a clase.*

Otra situación donde nos encontramos esto le decimos **coverage** (cobertura). Es cuando queremos dejar constancia de cosas que no están ligadas a ningún evento. *P.e las promociones. Si ponemos algún producto en promoción y no hay ninguna venta esta promoción no quedaría registrada en nuestra BD. Entonces para cada tienda, tiempo, producto, etc. dejamos constancia de si esta promoción se ha dado en el algún momento independientemente de si han habido ventas o no. Esto nos permitiría como en el caso de antes contar el número de promociones que había en un cierto instante de tiempo.*

Esto aparece solo en el nivel mas bajo de granularidad, porque en el momento que contamos generamos una medida y podemos seguir contándola, hacer el min/max... El nivel más bajo que guardaríamos en nuestro *datawarehouse* sería sin ninguna medida.

2. **Degenerated Dimensions**: Sucede no cuando el hecho no tiene medidas sino cuando la dimensión no tiene atributos, entonces hablamos de dimensiones degeneradas. Sirve para **dejar constancia** y permitir un **mejor análisis**. En este caso no ponemos llave foránea ya que no sirve de nada porque la dimensión a la que

hace referencia no tiene atributos (no hay dimension a la que apuntar realmente por eso). La dejamos parte de la llave primaria de la tabla de hechos.

3. **Junk Dimensions:** Caso contrario al anterior, lo que tenemos es muchos atributos dimensionales booleanos o de cierto número pequeño de valores. Para guardarlos, la primera opción sería hacer **una dimensión para cada una de ellos** con una llave foránea a cada una de las dimensiones. No es una buena opción ya que estas no tienen atributos y estariamos malgastando espacio y tiempo de hacer *joins*. Otra opción es ponerlos en la llave primaria pero no poner la llave foránea, sin embargo, esto hace que la llave primaria crezca sin sentido. La tercera y mejor opción sería crear una tabla para cada combinación de los atributos en la llave primaria tenemos llave foránea a esa tabla que tiene un identificador que nosotros creamos. De esta manera la tabla de hechos no crece tanto. Esta solución tiene ventajas ya que nos ahorra tiempo en la tabla de hechos y nos permite analizar las combinaciones de los atributos puestos en la tabla y mirar si hay correlación en los elementos de la tabla.
4. **Too large dimensions:** Otro caso que puede suceder es que algunas dimensiones que en general son pequeñas comparadas con la tabla de hechos pueden estallar. *P.e en la tabla de tiempo si ponemos segundos, minutos, horas, durante años. La solución seria todo lo que hace referencia al tiempo ponerlo en una dimensión y todo lo de la fecha en otra dimensión.* Las consultas serán un poco mas difíciles por la combinación de JOIN. La ventaja es que podemos analizar mas fácilmente el tiempo del día en que se hacen ventas porque solo tenemos que consultar una sola tabla.
5. **Slowly Changing Dimensions:** Otro problema que nos podemos encontrar es que aunque normalmente las dimensiones no cambian, pueden acabar cambiando por dos razones. La primera porque la realidad puede cambiar (*p.e una ciudad obtiene la independencia de su país*) y la segunda son errores humanos (*p.e nos equivocamos de Barcelona*). El problema que tenemos es como dejar constancia de estos cambios. Hay 3 opciones:
 - **Sobrescribir el valor:** opción más sencilla pero perdemos constancia del valor anterior.
 - **Añadir una nueva columna** donde dejamos constancia del valor anterior. Si el valor no ha cambiado le ponemos NULL y si ha cambiado le ponemos el valor anterior. Solo permite reflectar un cambio, a no ser que añadamos muchas columnas por cada cambio que se realiza.
 - **Añadir columna de tiempo** que nos dice para cada fila hasta que momento ese valor es válido. Es la opción más complicada pero a la vez la más genérica. En este caso el tiempo también forma parte de la *primary key* porque sino aparecerían repetidos (puede haber dos instancias de la misma primary key pero no en el mismo instante de tiempo).
6. **Aggregation hierarchies:** Tendemos a pensar que las jerarquías de agregación son lineales, pero no es verdad, en general pueden tener cualquier topología. Se tratan de un grafo donde lo único que se tiene que cumplir es que haya un único nivel arriba (ALL) y otro nivel que está abajo de todo que es donde apunta la tabla de hechos. En medio puede ser lineal pero no tiene porque serlo. Esto no afecta a la implementación. Podemos explicitarlo en el esquema *snowflake* o de manera implícita en el *star schema*. A nivel de implementación lo único que pasará es que tendremos mas atributos en la tabla de dimensiones.
7. **Conformed hierarchies/dimensions:** Cuando tenemos diferentes dimensiones compartidas en diferentes esquemas de estrella (esquemas de galaxia). Puede ser que estas estrellas vengan de diferentes fuentes pero deberíamos de conformarlas. Si tenemos que hacer *joins* entre ellas y lo que no puede ser es que las tablas

sean diferentes. Esta información que vienen de diferentes fuentes la tenemos que homogeneizar (tenerla de la misma forma) para que todas tablas de hechos puedan apuntar a la misma dimensión.

Lo que hay que hacer es crear una tabla donde en las filas tenemos las diferentes tablas de hechos y en las columnas tenemos las diferentes tablas de dimensión. Haciendo esto podemos ver las dimensiones que comparten y homogeneizar estas.

3.3 Aggregation Problems

Vamos a ver los problemas de agregaciones del modelo multidimensional y las situaciones que pueden surgir. Esto es importante porque las jerarquías de agregación y las operaciones *roll-up* son las operaciones más importantes de los cubos de datos.

- Lo primero que debemos mirar es que las categorías deben de ser **disjuntas**. Un dato no puede pertenecer a más de un valor de la misma categoría para que se pueda realizar la agregación.
 - En el caso de disyunciones (donde las categorías se solapan) se crea una table *puente* intermedia donde guardaríamos el peso de cada relación. *P.e si un alumno pertenece a dos asignaturas se le asignaría un peso de 1/2 a cada una para que a la hora de sumar solo cuente como 1.*
- Los datos deben estar **completos**, si hacemos la agregación debe ser el total real. Lo que estamos sumando tiene que corresponder al total.
 - Si la jerarquía de agregación es **incompleta** lo que deberíamos hacer es añadir fantasmas (valores ficticios) que sirvan para agregar las consultas¹.
- Tenemos que mirar las 3 características sean **compatibles**: el **tipo de dimensión** (temporal o no), **tipo de medida** (acumulativa, estado o valor por unidad) y el **tipo de función de agregación** que estamos utilizando (*min, max, avg, sum*).
 - Se trata de una relación ternaria, si las miramos por parejas no tenemos suficiente información para saber si lo que hacemos es correcto.

	Cumulative	State	Value per unit
min	No problem	No problem	No problem
max	No problem	No problem	No problem
sum	No problem	Non-temporal	Never
avg	No problem	No problem	No problem

¹El nivel más bajo normalmente corresponde al que tiene más instancias

4 Extract, Transform and Load

Extraer datos desde de una o múltiples fuentes, transformarlas según las necesidades y cargarlas a otro repositorio de datos. Puede ser por varios motivos:

- Simplemente **para guardar los datos**. No es lo mismo que un *backup* porque un *backup* se tiene que restaurar pues es una captura instantánea de los datos y no es habitual acceder ni transformar los datos.
- Por el **uso de herramientas específicas que necesitan una cierta estructura** específica de los datos (e.g aplicaciones para *machine learning*) y se necesitan transformar para el sitio en donde se usarán.
- La **integración de datos**, tenemos múltiples fuentes y se necesitan usarlas todas juntas.
- **Para asegurar que no se pierden/alteran los datos**. Pasa en entornos web, donde se cambia sin avisar o sin pedirnos permiso. Lo mejor es extraerlos y ponerlos en donde si tenemos un control.
- **Eliminar errores de las fuentes de datos y arreglarlos** que sabemos que existen en todas las fuentes de datos. Aplicar algoritmos de limpieza.
- **Corregir/Imputar** los valores faltantes. Como en la mayoría de casos no tenemos control para modificar las fuentes de datos, copiamos los datos en nuestro repositorio y allí las modificamos.

Todo esto permitido hasta un cierto punto, pues se tienen que respetar **aspectos legales y éticos que hay sobre los datos**. Primero nos tenemos que preguntar de quien son los datos y si tenemos permiso para hacer lo que queremos hacer. **Se les ha informado al propietario de los datos para poder hacerlo?**

Especialmente crítico cuando se trata de **datos personales**. En el caso de que se nos de permiso, debemos garantizar la confidencialidad tanto de los datos como del uso que se le dará al resultado de nuestro análisis.

4.1 Definition

4.1.1 Extraction

Tenemos múltiples tipos de datos que además son heterogéneos. Hemos de integrarlos. Dependiendo de las **características temporales** que tengan los datos se nos será más fácil o difícil extraer los datos:

- **Transitorio:** El caso más difícil, es cuando la base de datos no tiene ninguna característica temporal, no guarda ningún valor antiguo. Si se produce un cambio entre 2 extracciones, el cambio entre valores se perderá.
- **Semi-periódico:** guardan 1 de los cambios (siempre guardan el valor anterior). Mientras que no hayan dos cambios o más entre extracciones, estos valores no se perderán.
- **Temporal:** que guarde absolutamente todos los cambios que se han hecho. No hay problema para hacer la extracción, solo se pide todos los cambios desde la última extracción.

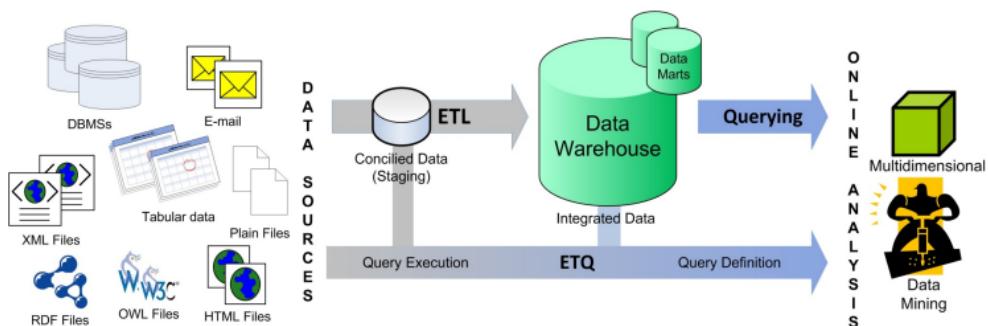
4.1.2 Transformation

- **Cambiar esquema:** por temas de integración, temas de estructura de datos para uso de herramientas de análisis, etc.
- **Convertir el conjunto de caracteres:** Pasa cuando tenemos distintos sistemas operativos o diferentes SGDB's y se utilizan conjuntos de caracteres diferentes. Caracteres extraños (e.g acentos o la ñ en castellano) que pueden salir raros.

- Mejorar la calidad de datos: Limpieza, imputación, completar fuentes, etc.

4.1.3 Load

- **On-line**
- **Off-line:** hacemos fuera a los usuarios del sistema para cargar los datos (*update window*). Durante la ventana de actualización, los usuarios no pueden trabajar con el sistema (el sistema está *offline*). Mientras está fuera de línea, cargamos los datos al DW y cuando acabamos lo ponemos a trabajar. Usualmente en la noche/fin de semana.
- Durante la carga se **desactivan/borran los índices** ya que insertar de uno en uno es mas costoso que no insertar todos los datos de golpe dentro del índice. A veces desactivamos el índice, esperamos que acaben las inserciones y actualizamos el índice con todas las inserciones que se hayan hecho o a veces borramos el índice y lo recreamos de nuevo (si el algoritmo de inserción es el mismo, no tiene mucho sentido recrear el índice). Solo en carga off-line, en online no podemos borrar los índices porque sino los usuarios no podrían trabajar.



En la imagen podemos observar una estructura más de alto nivel de como funciona el ETL. En la entrada tenemos diferentes tipos de fuentes de datos. Lo más importante aquí es que pasamos el **ETL a una área de trabajo (staging area)** donde de forma **temporal** podemos **materializar resultados parciales de estas transformaciones**. Esto nos hace más eficiente estas transformaciones y también puede servir para, si una cosa falla, no tener que volver hasta el principio. Es una práctica habitual tener algún tipo de almacenaje temporal para el ETL.

Una vez acabadas estas transformaciones **cargáramos los datos en nuestro almacén de datos** (DW) y desde este almacén de datos corporativo crearíamos nuestro Data Mars. Una vez hecho esto tenemos herramientas multidimensionales, Data Mining y Machine Learning que accederían a este DW o a los Data Mars y harían estos análisis. Esto sería el ETL tradicional.

Cada vez tenemos más prisa y urgencia por tener las cosas, la creación del DW a veces no nos conviene porque enlentece la fase de consulta (e.g mientras cargamos y no cargamos alomejor ha pasado un día). La alternativa de hoy en día es ejecutar consultas contra las fuentes y contra la área temporal del ETL (staging). Una vez tenemos estos datos extraídos de las fuentes hacemos una transformación mínima (normalmente sin mejora de calidad porque cuesta mucho tiempo) y se la pasamos a los usuarios. En vez de hacer *Load* de los datos hacemos directamente *Query*, es lo que se conoce como ETQ.

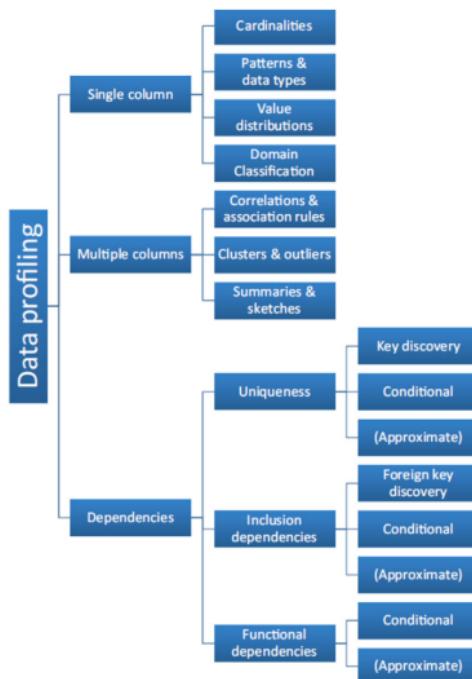
Otra alternativa es ELT (impulsada por vendedores de SGBD). No se hace la transformación en cualquier lenguaje/herramienta, sino que directamente se cargan los datos a la base de datos y la base de datos es capaz de hacer las transformaciones dentro del SGBD. Fusionar el *staging area* con el DW.

4.2 ETL Steps

No se puede poner en el DW todos los datos del mundo. **Definir los objetivos de negocio y el análisis** (target MD schema donde queremos cargar los datos). Lo más importante es que sean **datos relevantes y que no sean redundantes** ni de baja calidad, en términos de completeness, precisión, consistencia (cumples las restricciones de integridad), vigencia. **Garbage in - Garbage out**.

Para saber esto se necesitan entender los datos (**data profiling**). Por temas de calidad, ver las características y ver si encajan con lo que queremos hacer (cuestión relativa a los objetivos).

- **Mirar los datos de uno en una:** Si siguen un patrón, que distribución de valores tienen y posible clasificación dependiendo de su dominio.
- **Múltiples columnas a la vez:** buscar correlación o reglas de asociación, clusters o outliers y crear resúmenes (sketches) de estos datos.
- **Dependencias entre los datos:** ver si realmente tienen repetidos (considerar claves primarias), buscar inclusiones dependientes (claves foráneas) o dependencias funcionales (un valor determina el valor del otro).

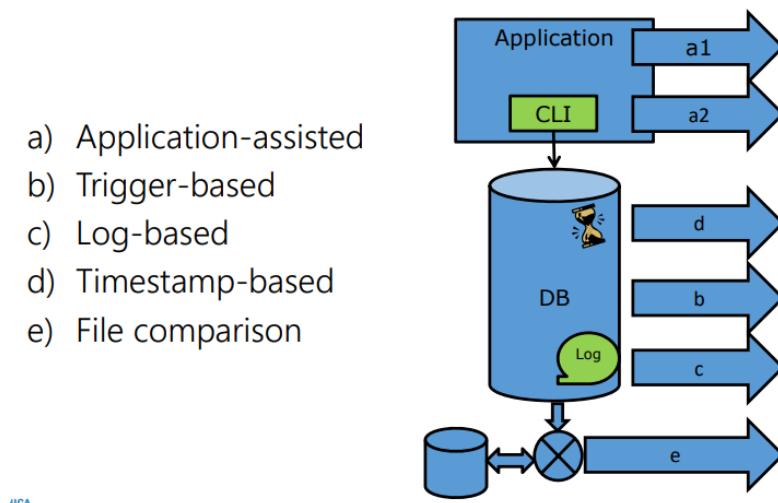


4.2.1 Data extraction mechanisms

Tenemos que ver como extraer los datos.

- **Las interceptamos en la propia aplicación que las está generando.**
 - Modificar el código de la aplicación para que a la vez que se hace la inserción a la base de datos, nos la envíe a nosotros en algún sitio que digamos.
 - Modificamos el driver (*call level interface*) de manera que cuando desde la aplicación se escriba para hacer una inserción a la base de datos, automáticamente el driver replique este dato a nosotros. La modificación del driver puede ser más complicada al inicio pero a largo plazo es mejor. No se debe conocer la aplicación, ni sufrir por modificaciones de la aplicación.
- **Nos esperamos que la aplicación la cargue a la base de datos y las cogemos de ahí.**
 - Mediante **triggers** (mecanismos que cuando se hace cualquier modificación en la BD se hace una acción) para que nos envíen los datos.
 - Podemos mirar los mecanismos del log. El log es el sistema para recuperarse si hay una falla (energética, etc). El log tiene constancia de los cambios que se han hecho en la base de datos. Si es software abierto, si es software propietario el formato del log no se conoce y no se puede hacer.
 - Consultas directas contra la base de datos. Si la BD es temporal, permite hacer consultas contra el histórico. Entonces pedimos todos los cambios que han habido desde la última extracción.
 - Extraer todos los datos de la BD y compararlos con la extracción previa que se haya hecho y ver las modificaciones desde entonces.

- a) Application-assisted
- b) Trigger-based
- c) Log-based
- d) Timestamp-based
- e) File comparison



4.2.2 Kinds of transformation tasks

La primera sería la selección de los datos. Se tiene que mirar que sean relevantes, etc.

Cleaning

En general mejorar la calidad de los datos. Primero generar los perfiles para entender lo que tienen y lo que no.

- **Algunos atributos que seguramente tengan información compuesta.** e.g una columna con la dirección puede tener el nombre de la calle, el zip, etc. que a lo mejor es más efectivo analizar por separado.
- **Estandarización:** de valores numéricos pero también de str.
- **Mejora de la calidad:**
 - *Completeness:*
 - * **Introducir valores manualmente:** normalmente no es buena idea, es demasiado farragoso.
 - * **Poner valor por defecto:** tampoco es buena idea, no da buen resultado en general.
 - * **Average/mediana/moda**
 - * **Average/mediana/moda por clase**
 - * **Aproximación de TEOI:** poner el valor que aporta la mayor información.
 - * **Tablas de lookup** donde tenemos correspondencias para cada valor.
 - *Correctness:*
 - * **Diccionarios (tablas de lookup)** para corregir valores
 - * **Detección de outliers:** detectar outliers sin borrarlos (pueden ser importantes), solo identificarlos.
 - *Analizar las restricciones de integridad (business rules).*

Size reduction

Si pensamos en los datos estructurados en formas de tabla, solo se puede reducir en número de filas o columnas.

- *Por filas (length):*
 - **Agregación:** groupby y nos quedamos con el promedio, mediana, min, max, etc.
 - **Representativo:** clustering y nos quedamos con algún representante (el centro normalmente).
 - **Muestreo**
- *Por columnas (atributos):*
 - **Correlación:** la información ya nos la da otra columna.
 - **Análisis de significación:** para ver si realmente nos da información relevante.
 - **Ganancia de clasificación**

Preparation

Puede pasar que algunos algoritmos tengan algunos requisitos específicos (solo valores numéricos, etc.).

- **Categórico a numérico**
- **Diferentes métodos de discretización,** numérico a categórico
- **Normalizar**
- **Conversión de los datos a metadatos (Onehot encoding):** para algoritmos que no permiten datos multivaluados (*listas*).

- **Derivación de información:** e.g un algoritmo con la data de nacimiento trabajará peor que con la edad (derivar la columna)
- **Enriquecimiento:** Haciendo `join` con otras fuentes de datos, mientras sea relevante.

4.3 ETL process design

El diseño funciona parecido al que hacemos cuando se diseña una BD, es decir, tenemos primero un diseño **conceptual, lógico y físico**.

- **Diseño conceptual:**
 - Puede realizarse en múltiples aplicaciones como UML o incluso lenguajes específicos para diseñar ETL.
 - Con el propósito de mejorar la comprensión y tener la posibilidad de compartirlo.
 - Incluso una posibilidad de automatizar la traducción al diseño lógico si escogemos la herramienta adecuada.
- **Diseño lógico:**
 - Descripción detallada del workflow, las dependencias, el esquema y los planes de restauración.
 - Representado por un grafo dirigido acíclico donde los vértices son las fuentes de datos, operaciones o "targets" y las relaciones indican las dependencias/fluxos de datos que hay de un nodo a otro. (Talend)
 - Se muestran las transformaciones que sufren los datos.
- **Diseño físico:** generado automáticamente a partir del diseño lógico.

4.3.1 ETL tools GUI

Herramientas gráficas (Talend). Recomendado para proyectos grandes, procesamiento relativamente sofisticado y/o cuando no tenemos mucha habilidad de programación. Además de eso proporciona:

- **Un repositorio de metadatos integrados.** Deja constancia de todo lo que se tenga (fuentes de datos, transformaciones, cargas).
- **Conversiones de tipos de datos automática y comprobación de errores.**
- **Trazar/visualizar de donde vienen los datos y de que dependen las cosas cuando se cargan los ficheros.**

4.3.2 Hand-coded ETL

Posibilidad de sustituir la automatización. Recomendable cuando tenemos el conocimiento, hay rutinas ya implementadas o cuando las herramientas ya preestablecidas no hacen exactamente lo que queremos que hagan.

Conjuntamente con esto, si queremos hacer un proceso similar a *Map-Reduce* (librerías de Spark) se necesitarían hacer a mano y eso nos facilitaría el uso de datos sin esquema (e.g imágenes, texto en lenguaje natural). Se preprocesarían estas fuentes con el *Map-Reduce* y a partir de aquí se extraerían una serie de características que se cargarían a un SGBD y ya se podría continuar con todo el procesamiento que se desea.

4.3.3 ETL process quality

En cualquiera de las dos alternativas, se tiene que mantener unos estándares de calidad.

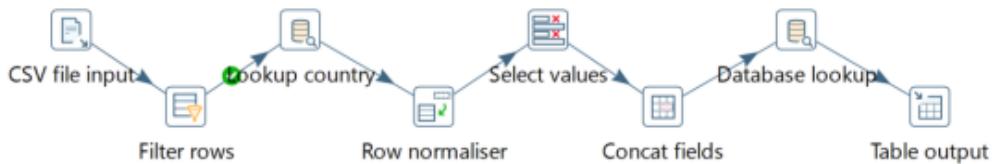
- **Rendimiento:** el más importante. Se ha de ejecutar de la manera mas rápida y eficientemente posible, pensando en la escalabilidad. Con recursos razonables dentro del presupuesto.
- **Confianza:** de solucionar los posibles errores en términos de robustez. Gestión de errores, tolerancia a fallos y recuperabilidad.
- **Testabilidad:** tenemos de ser capaces de testar el código incluso tras hacer cambios. Comprobar que se mantiene la eficiencia. También tienen que ser traceables, saber de donde vienen.
- **Mantenibilidad:** muy importante si se piensa en el futuro. El ETL se ha de ejecutar de manera recurrente a lo largo del tiempo, habrá cambios en las fuentes, etc. y se tiene que mantener/cambiar. Entonces depende de como se defina el coste de mantenimiento en el presente, afectará en un futuro.
 - Se tiene que intentar hacer el código/ETL más comprensible. (etiquetar, anotaciones, comentarios, etc.).
 - Mantenibilidad difícilmente cuantificable. Aunque existen algunas métricas como (#operaciones, #fuentes) pero son bastante evidentes y no se puede hacer mucho.
 - Se puede hacer Modularidad/Encapsulación: definición de métodos, funciones significativos con comentarios que pueden ser reutilizables de la manera mas conveniente para ayudar a la mantenibilidad.

4.4 Architectual setting

Distinción entre **data flow** (flujo de datos) y **control flow** (flujo de control).

4.4.1 Data flow

También conocido como **transformación**, lo que hace es transformar los datos. El orden de ejecución no se puede saber a menos de que haya precedencias claras en el grafo. Las aristas de este grafo dirigido lo que muestran es como los datos van pasando de una transformación a la otra.



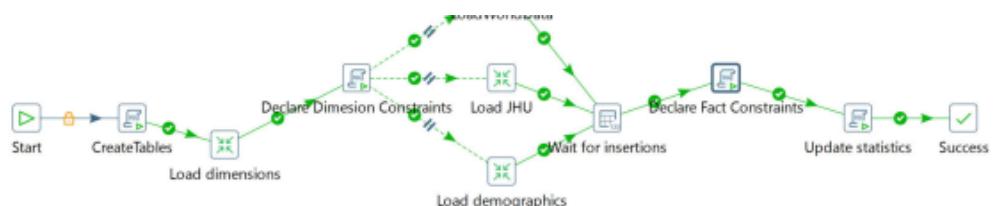
- **Fuente (Source):** Es el punto de origen de los datos, donde se inicia el flujo de datos. Puede ser una base de datos, un archivo, un sensor, una API, u otras fuentes de datos.
- **Transformación (Transformation):** Son las operaciones que se aplican a los datos a medida que se mueven a través del flujo. Las transformaciones pueden incluir filtrado, limpieza, agregación, cálculos y cualquier cambio en la estructura o el formato de los datos.
- **Ruta (Path):** Es la dirección en la que fluyen los datos desde la fuente a través de las transformaciones hasta llegar al destino.

- Destino (Destination): Es el punto final del Data Flow, donde se almacenan los datos procesados o se utilizan para un propósito específico. Puede ser una base de datos de destino, un almacén de datos, una aplicación, un informe, entre otros.

4.4.2 Control flow

Si queremos forzar un orden de ejecución, también se llaman **Jobs**. Las aristas de este grafo ya no indican paso de datos entre tareas, sino que precedencias entre tareas. Comenzamos con un evento inicial, tenemos un evento final y en medio tenemos diferentes transformaciones complejas. Las aristas indican precedencias. Puede ser secuenciales o paralelas.

La dependencia es muy importante, e.g. se deben cargar las tablas de dimensión antes que las tablas de hecho porque sino habría errores de `foreign key`.



Distinción importante entre las aristas de ambos grafos. Las del **data flow** muestran el paso de datos entre una tarea y otra. Mientras que las aristas del **data control** muestran precedencia entre tareas. El flujo de control lo único que indica es precedencia, que se hace antes de que, pero no indica flujo de datos, no se pasan datos entre ellos, simplemente un planificador.

4.4.3 Staging area

Es un sistema de almacenamiento usado para mejorar el funcionamiento o el rendimiento del ETL. Pueden estar guardadas los datos aquí en ficheros planos, *XML*, tablas relacionales o en cualquier otro formato.

Lo importante es que facilita la ejecución del ETL. Puede hacer que vaya más rápido, hacerlo recuperable, puede hacer *backups* (por si alguna cosa no interesa repetir la ejecución) o bien puede servir para hacer auditoría (dejar constancia de las transformaciones que se hacen y las correspondencias que hay entre unos datos y otros).

4.4.4 Metadata management

Muchas veces los metadatos se nos permiten extraer de forma automática (e.g. de las fuentes de datos o de la **Staging area**), en qué formato están, qué tipo de datos son, etc. Lo mismo para el DW. También metadatos del proceso de ejecución como cuantas filas tenemos. Incluso alguna herramienta para la definición de las reglas de negocio.

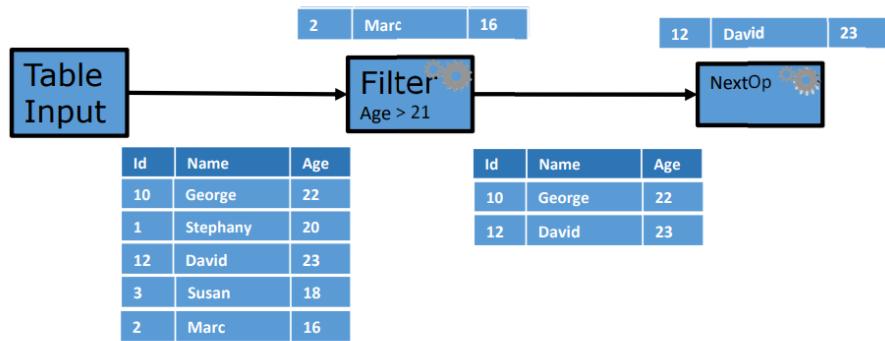
4.5 ETL operations

Operation Level	Operation Type	Pentaho PDI	Talend Data Integration	SSIS	Oracle Warehouse Builder
Field	Field Value Alteration	Add constant Formula Number ranges Add sequence Calculator Add a checksum	tMap tConvertType tReplaceList	Character Map Derived Column Copy Column Data Conversion	Constant Operator Expression Operator Data Generator Transformation Mapping Sequence
Dataset	Duplicate Removal	Unique Rows Unique Rows (HashSet)	tUniqRow	Fuzzy Grouping	Deduplicator
	Sort	Sort Rows	tSortRow	Sort	Sorter
	Sampling	Reservoir Sampling Sample Rows	tSampleRow	Percentage Sampling Row Sampling	
	Aggregation	Group by Memory Group by	tAggregateRow tAggregateSortedRow	Aggregate	Aggregator
	Dataset Copy		tReplicate	Multicast	
Row	Duplicate Row	Clone Row	tRowGenerator		
	Filter	Filter Rows Data Validator	tFilterRow tMap tSchemaComplianceCheck	Conditional Split	Filter
	Join	Merge Join Stream Lookup Database lookup Merge Rows Multiway Merge Join Fuzzy Match	tJoin tFuzzyMatch	Merge Join Fuzzy Lookup	Joiner Key Lookup Operator
	Router	Switch/Case	tMap	Conditional Split	Splitter
	Set Operation - Intersect	Merge Rows (diff)	tMap	Merge Join	Set Operation
	Set Operation - Difference	Merge Rows (diff)	tMap		Set Operation
	Set Operation - Union	Sorted MergeAppend streams	tUnite	Merge Union All	Set Operation

Operation Level	Operation Type	Pentaho PDI	Talend Data Integration	SSIS	Oracle Warehouse Builder
Schema	Field Addition	Set field value Set field value to a constant String operations Strings cut Replace in string Formula Split Fields Concat Fields Add value fields changing sequence Sample rows	tMap tExtractRegexFields tAddCRCRow	Derived Column Character Map Row Count Audit Transformation	Constant Operator Expression Operator Data Generator Mapping Input/Output parameter
	Datatype Conversion	Select Values	tConvertType	Data Conversion	Anydata Cast Operator
	Field Renaming	Select Values	tMap	Derived Column	
	Projection	Select Values	tFilterColumns		
Table	Pivoting	Row Denormalizer	tDenormalize tDenormalizeSortedRow	Pivot	Unpivot
	Unpivoting	Row Normalize Split field to rows	tNormalize tSplitRow	Unpivot	Pivot
Value	Single Value Alteration	If field value is null Null if Modified Java Script Value SQL Execute	tMap tReplace	Derived Column	Constant Operator Expression Operator Match-Merge Operator Mapping Input/Output parameter
Source Operation	Extraction	CSV file input Microsoft Excel Input Table input Text file input XML Input	tFileInputDelimited tDBInput tFileInputExcel	ADO .NET / DataReader Source Excel Source Flat File Source OLE DB Source XML Source	Table Operator Flat File Operator Dimension Operator Cube Operator
Target Operation	Loading	Text file output Microsoft Excel Output Table output Text file output XML Output	tFileOutput tDelimited tDBOutput tFileOutputExcel	Dimension Processing Excel Destination Flat File Destination OLE DB Destination SQL Server Destination	Table Operator Flat File Operator Dimension Operator Cube Operator

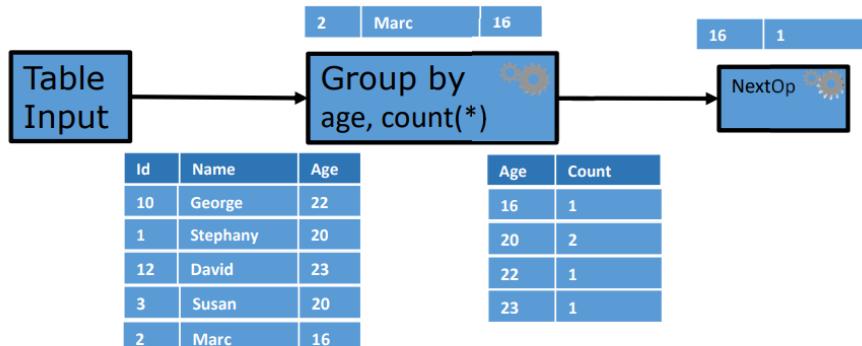
4.5.1 Non-Blocking Example

El filtro cogería fila por fila y comprobaría la condición del filtro. Mientras una fila este en estado de comprobación en el filtro, la fila anterior que pasó por el filtro ya puede haber pasado a la siguiente operación para procesarla. Las dos operaciones adyacentes se están ejecutando en paralelo.



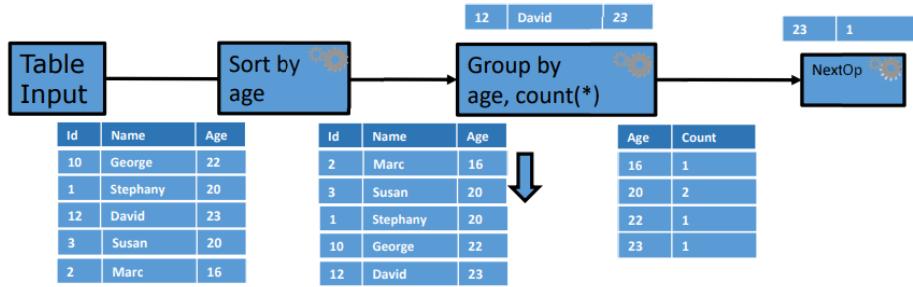
4.5.2 Blocking Example

Group by. No ha generado nada a la salida hasta que no se acaben de preprocesar todas las filas, porque no puede saber si para una edad determinada cuantas de esta habrá en total hasta que no se acabe de leer todas las entradas. Solo cuando ha terminado de leer la última fila, la próxima operación puede comenzar a procesar.



4.5.3 Blocking optimized Example

Una manera de optimizar este caso es (para el ejemplo dado) es realizar una operación de `sort by` para ordenar los datos de manera que se sabe cuando se acaban las filas con el mismo valor para el `group by`. Ahora la operación de bloqueo es el `sort by`.

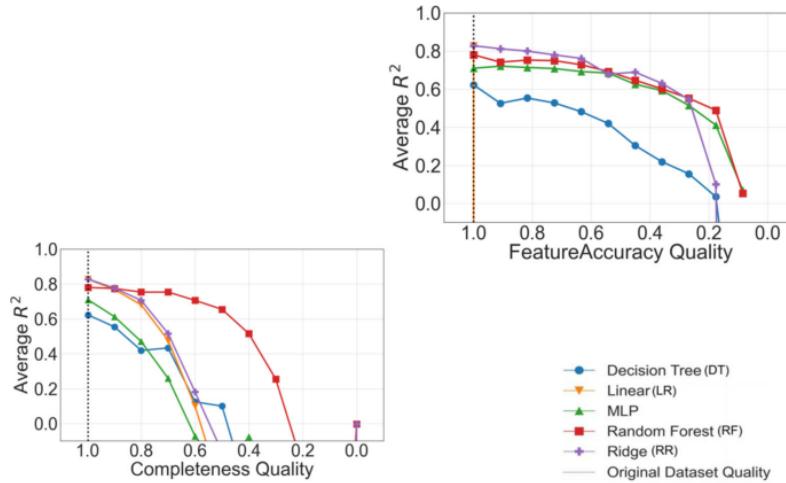


4.5.4 List of blocking and Non-blocking operations

- Blocking
 - Duplicate Removal
 - Sort
 - Aggregation
 - Join
 - Intersect
 - Difference
 - Unpivot (transform rows into columns)
- Non-blocking
 - Field Value Alteration
 - Single Value Alteration
 - Sampling
 - Dataset Copy
 - Duplicate Row
 - Router
 - Union
 - Field Addition
 - Datatype Conversion
 - Field Renaming
 - Projection
 - Pivot (transform columns into rows)
 - Extraction
 - Loading

5 Data Quality

La calidad de los datos tiene un **impacto económico**. Es importante darse cuenta que tiene un impacto en el coste del día a día de la empresa y también desde el punto de vista positivo perdemos ingresos y ventas. Si aumentamos la calidad reduciremos costes y aumentaremos los ingresos.



Desde un punto de vista más técnico podemos ver las gráficas. Tienen el R^2 de los diferentes modelos en el eje y. Como podemos ver dependiendo de la complejidad de los datos nos afecta al R^2 del modelo. Como menos completos sean los datos menor es el R^2 . Esto también sucede con la *accuracy* de los atributos. A veces perdemos mucho tiempo iterando los hiperparámetros o probando modelos y en verdad **lo que tendríamos que hacer es mejorar la calidad de los datos** porque tendrá un impacto mucho mayor.

Tenemos que **determinar para qué uso queremos usar los datos**. La calidad de los datos es un proceso costoso y tenemos que considerar qué cantidad de recursos destinamos ello. En general tener unos datos completos y de calidad mejora en todos los sentidos nuestro modelo.

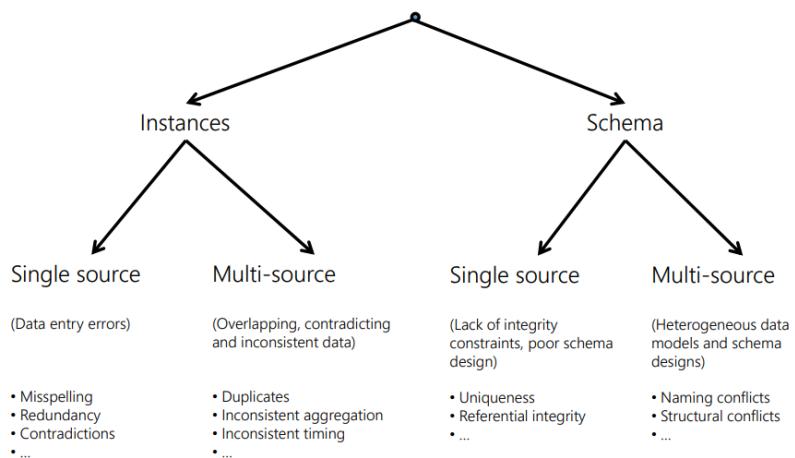
Los **fallos más habituales** son que faltan datos, o que estos datos estén obsoletos (ya no corresponden a la realidad actual). También que los datos no sean lo suficientemente acurados. El **efecto** que esto tiene es que se ahorra costes si aumenta la calidad de datos, se incrementa la eficiencia y la reputación de la empresa mejora (nuestros clientes/socios deben creer que tenemos datos correctos). Podemos perder oportunidades de negocio. Además las usamos para tomar decisiones, si los datos son incorrectos nuestras decisiones probablemente también.

Los problemas pueden venir de cualquier sitio:

- **Carga inicial de los datos:** si hacemos una conversión de datos inicial o si los ponemos a mano nos podemos equivocar. Haciéndolo con *batch* (paquetes de datos que cargamos de manera automática), con interfaces en tiempo real o con diferentes sistemas puede llevar a errores.
- **Preproceso:** si el proceso es manual o automático introducirá errores. Si intentamos limpiar los datos podemos introducir otros errores (por una parte mejoramos pero por la otra no), si purgamos también podemos estar perdiendo datos importantes (errores por borrar demasiado).

- **Inacción:** si no hacemos nada la calidad de los datos se ve repercutida. Hay cambios en el mundo, en la base de datos, en el entorno, etc. que no se capturan si no hacemos nada.

5.1 Data Conflicts



5.2 Quality Measures

Vamos a ver como podemos medir la calidad de nuestros datos, hay muchas perspectivas/dimensiones de calidad de datos y dentro de estas dimensiones hay muchas métricas. Las 3 más relevantes son la **completitud**, la **precisión** y la **consistencia** de los datos. Dentro de la precisión podemos incluir la precisión *temporal*, que es cómo de actualizados están nuestros datos en la BD.

Hay otras dimensiones como:

- **Redundancia**
- **Legibilidad**
- **Accesibilidad**
- **Utilidad**
- **Confianza**

5.2.1 Completeness

Primero, es importante considerar si existe la posibilidad de que falten entidades del mundo real que no estén representadas en nuestra base de datos. Al abordar este escenario, entramos en lo que se conoce como la "**Open World Assumption**" (**OWA**), lo que implica que nuestra base de datos no es cerrada. En otras palabras, estamos abiertos a la posibilidad de que existan otros objetos en el mundo real que no estén reflejados en nuestra base de datos. En este contexto, es fundamental examinar cuidadosamente los objetos del mundo real y determinar cuáles de ellos no están presentes en nuestra base de datos.

En contraste, la "**Closed World Assumption**" (**CWA**) representa el escenario opuesto. Aquí, asumimos que nuestro mundo es cerrado, lo que significa que los únicos objetos que existen en el mundo real son aquellos que están registrados en nuestra base de datos. Bajo esta premisa, no deberíamos tener entidades faltantes en términos de

filas (aunque podríamos enfrentar la ausencia de ciertas columnas o valores dentro de los atributos). Es importante comprender y diferenciar entre estos dos enfoques, ya que pueden tener implicaciones significativas en el tratamiento y la interpretación de los datos.

$$Q_{C_m}(A_i) = \frac{|R(\text{NotNull}(A_i))|}{|R|}$$

Porcentaje de valores que tenemos del atributo. Podemos hacer lo mismo para la tabla entera, cogemos la conjunción para todos los atributos de la tabla. Nos dirá el porcentaje de filas que no tienen ningún atributo nulo.

$$Q_{C_m}(R) = \frac{|R(\wedge_{A_i \in R} \text{NotNull}(A_i))|}{|R|}$$

5.2.2 Accuracy

Nos indica el grado de error en nuestros datos.

- $e_A = |\mathbf{v}_A - \mathbf{v}_{\text{RealWorld}}|$, para llevar a cabo esta medida, es necesario conocer el valor del mundo real, lo que implica realizar un muestreo de estos objetos o atributos. Luego, se compararía el valor real con el valor que tenemos en la base de datos y se calcularía la diferencia.
- $Q_A(A_i) = |\mathbf{R}(e_{A_i} \leq \epsilon)|/|\mathbf{R}|$, miramos si el error es menor que un cierto ϵ , si lo es, seleccionamos estas filas de la tabla y contamos cuantas hay que tienen un valor aceptable para este atributo y entonces dividimos entre el total de filas de la tabla.
- $Q_A(R) = |\mathbf{R}(\wedge_{A_i \in R} e_{A_i} \leq \epsilon)|/|\mathbf{R}|$, podemos hacer lo mismo para toda la tabla, debemos hacer la conjunción para todos los atributos de la tabla, miramos si su error está dentro de este intervalo que consideramos aceptable. Si una fila tiene todos los atributos dentro de esta precisión contamos la fila.

5.2.3 Timeliness (Freshness)

Temporalidad o frescura de los datos. Necesitamos dos datos:

- $age(v) = \text{now} - \text{TransactionTime}$, edad del dato, momento actual menos el tiempo de transacción (cuando se ejecutó la última modificación sobre este dato en nuestra BD). Aunque tenga una edad alta no tiene porqué estar obsoleta, depende de cada cuando deber ser modificado, cada cuanto varia este dato.
- $f_u(v) = \text{updates per time unit}$, frecuencia con la cual esperemos que este valor cambie (e.g Fecha de Nacimiento no esperamos que cambie). Si la edad hace 5 años que no cambia, este dato esta obsoleto, ya que esperamos que cambie cada año.

Teniendo en cuenta estas dos medidas podemos calcular lo siguiente:

- $Q_T(v) = (1 + f_u(v) \cdot age(v))^{-1}$, multiplicamos la frecuencia esperada de cambio multiplicado por la edad, le sumamos 1 para evitar divisiones por 0 y después hacemos la inversa. Si esperamos que un valor cambie 3 veces por año y la edad de esto es 1 año entonces daría $1/4$.
- $Q_T(A_i) = \text{Avg}_{v \in \mathbf{R}[A_i]} Q_T(v)$ calidad temporal de un atributo entero, hacemos el promedio de cada uno de los valores
- $Q_T(R) = \text{Avg}_{A_i \in R} Q_T(A_i)$, hacemos lo mismo para toda la relación

5.2.4 Consistency

La consistencia se basa en medir una serie de **restricciones de integridad** (business rules):

- **Entidad:** ver si conocemos el identificador de la fila.
- **Dominio:** ver si los enteros son enteros, los reales son reales, ...
- **Referenciales:** llaves foráneas.
- **User-defined:** *checks* que podemos poner en la tabla.

También hay que tener en cuenta la **coincidencia de las copias**.

- **Temporal:** si una copia no se ha actualizado y la otra sí.
- **Permanente:** si una copia no se actualiza nunca y la otra si se va actualizando.

$$Q_{C_n}(R, B) = \frac{|R(\wedge_{rule \in B} rule(A_1, \dots, A_n))|}{|R|}$$

R es la tabla y B el conjunto de reglas (es lo que medimos). Hacemos la conjunción para todas las reglas que hay y miramos si estas reglas se cumplen para todos los atributos. Cuantas tuplas hay que cumplan todas las reglas. Porcentaje de tuplas que cumplen todas las restricciones de integridad que tenemos.

5.2.5 Trade-offs between dimensions

Darse cuenta de que **hasta cierto punto, estas metas son contradictorias**. No podemos esperar maximizar todas ellas simultáneamente. Si intentamos maximizar la frescura de los datos, es decir, actualizarlos lo más rápido posible una vez que tenemos información nueva, corremos el riesgo de cometer errores de precisión. Existe la posibilidad de que insertemos valores demasiado apresuradamente, lo que podría generar problemas de consistencia en la base de datos.

Por otro lado, si nos enfocamos en mejorar la completitud y la precisión de los datos, necesitaremos más tiempo, ya que los valores se actualizarán de manera más lenta. Cuando intentamos mejorar la completitud de los datos mediante la imputación de valores faltantes, es probable que surjan errores de precisión y consistencia, y viceversa.

Timeliness \Leftrightarrow Accuracy, Consistency and Completeness

Completeness \Leftrightarrow Accuracy and Consistency

5.3 Quality Rules

Relacionado con la consistencia de la BD tenemos un conjunto de reglas de calidad y por lo tanto es importante analizar que características tienen o pueden tener.

5.3.1 Restricciones de integridad en RDBMS

- **Intra-Attribute:** afectan solo a un atributo. Control de dominio o outliers y presencia de valores nulos.
- **Intra-Tuple:** afectan a una tupla entera como podría ser los checks.
- **Intra-Relation:** afectan a todas las tuplas de una relación, seria el caso de llaves primarias y unicidad dentro de los atributos (sin valores repetidos). También los *triggers* temporales o de tipo estado (e.g definir un trigger que no permita que bajen los sueldos).
- **Inter-Relation:** afectan diferentes relaciones como las llaves foráneas y las assertions.

En algunos casos se puede formalizar mediante dependencias:

- **Dependencias multivaluadas:** que un atributo tenga un cierto valor hacer que otro atributo pueda tener solo algunos valores (determina muchas atributos e.g sabiendo el DNI sabemos qué idiomas habla una persona). Como caso degenerado tenemos las **dependencias funcionales**, que si un atributo tiene un valor, otro atributo solo puede tener un valor. Esto implementa llaves foráneas y *unique*.

$$t1.X = t2.X \implies t1.Y = t2.Y$$

- **Dependencias de inclusión:** para modelar de manera matemática las llaves foráneas. Si tenemos una llave foránea en la relación R, los valores de esa llave foránea serán un subconjunto de los valores de la *primary key* de la otra tabla.

$$R.X(fk) \subset S.Y(pk)$$

5.3.2 Problems in the rules

Pueden dar problemas de:

- **Contradicciones:** hace que no podamos insertar algunas filas

$$CHECK(a < 10 \text{ AND } a > 20)$$

- **Redundantes:** hacen que el rendimiento empeore
 1. $CHECK(a > 20)$
 2. $CHECK(a > 10)$
- **Imperfecciones:** Puede ocurrir que estemos intentando evitar ciertas filas, pero nuestras reglas y restricciones de integridad no logren evitar los estados que deseamos prevenir o, en su lugar, impidan estados que no teníamos la intención de evitar.

Respecto las dos primeras podemos definir una serie de propiedad lógicas de las reglas o restricciones de integridad.

- **Satisfacción del esquema:** un esquema es satisfactible si podemos poner una fila en una tabla (en cualquiera de ellas).
- **Lifeliness:** referencia a tablas o vistas, hay que poder insertar una fila en una tabla concreta.
- **Redundancia:** una restricción de integridad es redundante si la consistencia de la BD no depende de esta, porque las demás restricciones ya hacen su trabajo.
- **State-reachability:** definido por un conjunto de tuplas. Quiere decir que podemos encontrar un estado consistente de la BD en la cual estén estas tuplas (y puede que más).
- **Contenimiento de la query:** el resultado de una consulta siempre un subconjunto del resultado de otra consulta.

Es importante darse cuenta que en cualquier caso hablamos siempre de BD consistentes, si la BD no es consistente (se violan las restricciones de integridad) no tiene sentido mirar las propiedad lógicas.

5.3.3 Fine-tuning imperfections

Puede ser que algunas tuplas que estén correctas hacen que están reglas de integridad se disparen (falso positivo) o al revés, filas que son incorrectas no se detectan

(false negative). Como las reglas de integridad no se violan muy a menudo, no debería haber demasiados falsos positivos. Los falsos negativos si hacemos un *sampling* y no encontramos ninguno probablemente no hay demasiadas violaciones.

Lo que debemos hacer es:

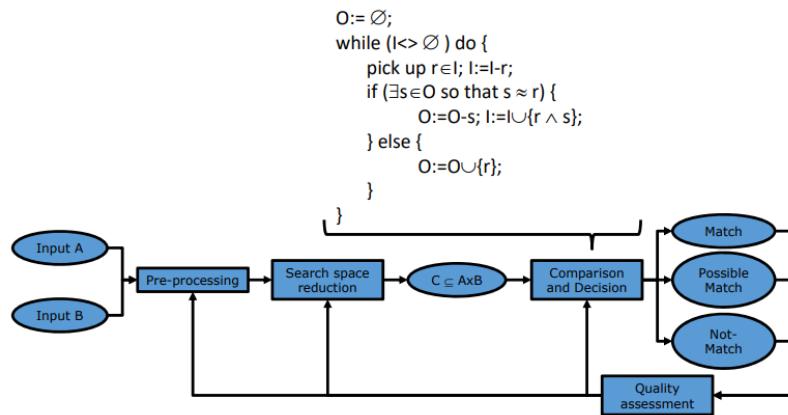
1. Identificar imperfecciones
2. Analizar si hay algún patrón en las imperfecciones
3. Mejorar las reglas y volver a 1

5.4 Quality improvement

Veremos que podemos hacer para mejorar la calidad de nuestros datos. Una de las cosas que necesitaremos es identificar objetos en BD diferentes. Si queremos mejorar la calidad tenemos 2 maneras de hacerlo:

- **Imputar valores** sin ninguna información adicional, simplemente buscando el cambio mínimo o buscando una cierta distribución de los datos de un mismo atributo o de diferentes atributos a la vez.
- **Buscar fuente de información** que esperemos que nos de más información de la que tenemos y cruzar o completar con esta fuente. Nuestra fuente original tiene problemas en algunos atributos y buscamos una fuente de datos que tenga mejor información de estos atributos en concreto.
 1. Encontrar donde está el problema (atributo)
 2. Buscar fuente que sea más confiable por la razón que sea y el coste que tenga (no tiene por qué ser gratuita)
 3. Identificar que dos tuplas en los dos conjuntos de datos realmente coinciden con el mismo objeto del mundo real. Primero deberemos hacer alguna normalización/estandarización de los identificadores porque no tienen porque ser el mismo.
 4. Juntar los registros.

5.4.1 Process for Object identification



- **Input A:** fuente original
- **Input B:** fuente de datos complementaria que nos da la información adicional que tiene mejor calidad
- **Pre-processing:** Normalización/estandarización de los identificadores u otros atributos.

- **Search space reduction:** si queremos comparar todas las tuplas con todas las tuplas de la otra fuente el coste computacional es muy elevado.
- $C \subset A \times B$: analizaremos un subconjunto del producto cartesiano. Este subconjunto lo deberíamos de formar de manera que no perdamos ninguna pareja que nos interese.
- **Comparison and Decision:** tenemos que comparar de estas parejas de A y B, se hace con el algoritmo *R-shush* (tenemos una función de comparación). Esto nos puede dar 3 resultados:
 1. **Match:** las dos tuplas se refieren al mismo objeto.
 2. **Possible Match:** hay una posibilidad de que las dos tuplas se refieran al mismo objeto.
 3. **Not-match:** las dos tuplas **no** se refieren al mismo objeto.
- **Quality assessment:** manualmente miraremos si esto nos está dando un resultado aceptable, si no nos lo está dando deberemos reconsiderar el preproceso, que el espacio no sea demasiado reducido o que la función de comparación funcione bien.

5.4.2 Search space reduction

Hay 3 técnicas conocidas y estudiadas:

- **Blocking:** participamos los dos conjuntos según una clave (atributo) por ejemplo equipos de fútbol.
- **Sorted neighbourhood:** ordenar de alguna manera, en orden alfabético por ejemplo.
- **Pruning:** hacer algún tipo de heurística, por ejemplo no comparar los porteros con los jugadores.

5.4.3 Comparison/Similarity function

Necesitamos una función de comparación, hay muchas opciones.

- **Distance-based**
 - Decide that two records represent the same entity if their distance is below a threshold (after removing affixes)
 - Hamming distance
 - Edit distance (insert, delete and replace)
 - Smith-Waterman algorithm (assign weights)
 - n-Grams comparison
 - Jaro algorithm (insert, delete and transpose)
 - Compare lists of items
 - Jaccard distance
 - Token Frequency-Inverse Document Frequency (TF-IDF)
 - May use phonetics
 - Soundex code
- **Rule-based**
 - Depending on the concrete attributes (e.g., ignoring middle name)
- **Probabilistic techniques** (which require a threshold)
 - Based on conditional probabilities
- **Classification-based**
 - Provide positive and negative training pairs of instances (to train a classifier)

6 Schema and Data Integration

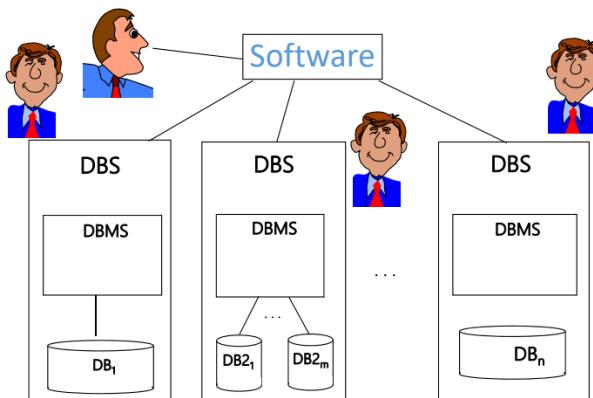
Problema de la integración de los datos: Lo que queremos hacer es una consulta y para resolverla necesitamos acceder a diferentes fuentes de datos.

- El problema no es como nos conectamos a las bases de datos (se asume ya desde un inicio).
- El problema no es el envío de los datos entre sistemas.
- El problema no es el acceso remoto a las BD. Se puede hacer con cualquier tipo de driver (e.g JDBC)
- El problema no es tener múltiples clientes/servidores.
- El problema no es que tengamos un SGBD distribuido.

El problema es que queremos **hacer una consulta, queremos tener una respuesta y para tenerla tenemos que acceder a múltiples BD.** e.g. los atributos de las diferentes BD pueden referirse a lo mismo pero con valores diferentes en algunos casos.

Tres soluciones posibles a este escenario:

- **Hacer a mano las queries por separado en las diferentes BD.** Para hacerlo que la persona tiene que saber un poco de todo (las fuentes, los modelos, el lenguaje, etc.). Tiene que dividir la consulta en las diferentes consultas para cada una de las BD y el resultado integrarlo en un único resultado. *Inviável para gran número de fuentes de datos.*
- **Crear una nueva BD** contenido toda la información necesaria (**proyecto de migración** de los datos, ETL p.e.). Se tiene que modificar las aplicaciones para acceder a esta nueva BD en lugar de las fuentes de datos. Se tiene que hacer un *testing* de todo y pondríamos en marcha esta fuente de datos única donde todo el mundo trabajaría con esa.
- **BD distribuida:** Capa de **software** que (auto o semiautomáticamente) es capaz de dividir la consulta e integrar las respuestas. Posible hasta cierto punto. La capa de software define dos niveles de acceso:
 - Los usuarios que acceden a través de la capa de software.
 - Los usuarios que ya utilizaban las BD desde antes. Estos se mantienen.



Hay usuarios pre-existentes que continúan trabajando como lo hacían antes. Lo que añadimos es una capa de software donde el usuario quiere hacer una consulta integrada (una consulta una respuesta). Este software divide la consulta en consultas más pequeñas sobre cada fuente. Cada fuente devuelve la respuesta a la capa de software y esta al usuario.

6.1 BD distribuida

Son diferentes **BD interrelacionadas, distribuidas en una red** y que se quiere tan **transparente** como sea posible. Se asume que todo es homogéneo y un sistema único aunque no se especifica en la definición formal.

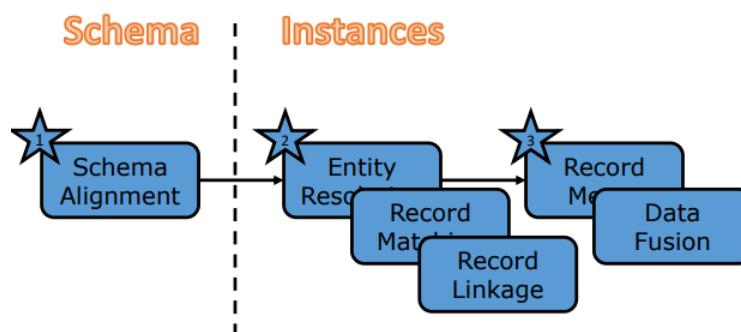
Se pueden clasificar las diferentes BD distribuidas según su autonomía (de más a menos). La autonomía nos lleva a que se puedan hacer las cosas diferentes, por lo tanto tenemos heterogeneidad.

- **heterogenidad de sistema:** En un sistema, vemos que tiene varias BD dentro y son diferentes. Cada uno con sus características y se tiene que solventar de alguna manera. *Esta es la heterogenidad fácil.*



- **heterogenidad de semántica:** Independencia para funcionar utilizando formatos, esquemas, etc. diferentes de los datos. Esto nos lleva a que cuando se busque interpretar, cruzar estos datos y acceder de forma conjunta, se tiene que resolver esta heterogenidad (del significado de los datos). Para hacerlo se tiene que primero integrar la parte del esquema y luego la parte de las instancias:

- **Esquema:** se tienen que encontrar las tablas que contienen información común (que hacen referencia al mismo tipo de objeto/identidad del mundo real) y también los atributos (mirar si son los mismos o no). *No es fácil de encontrar.*
- **Instancia:** Una vez encontrado que dos tablas contienen información del mismo tipo de entidad. Se tiene que ver si las entidades concretas que hay dentro (tuplas) realmente son las mismas o no (**Entity resolution = Record matching/linkage**).
- **Record Merge:** Coger todos los atributos de las filas que coinciden y ponerlos juntos. La idea es una vez tenemos fusionados los datos de todas las entidades que realmente coinciden ya seremos capaces de devolverle al usuario una respuesta única para su consulta.

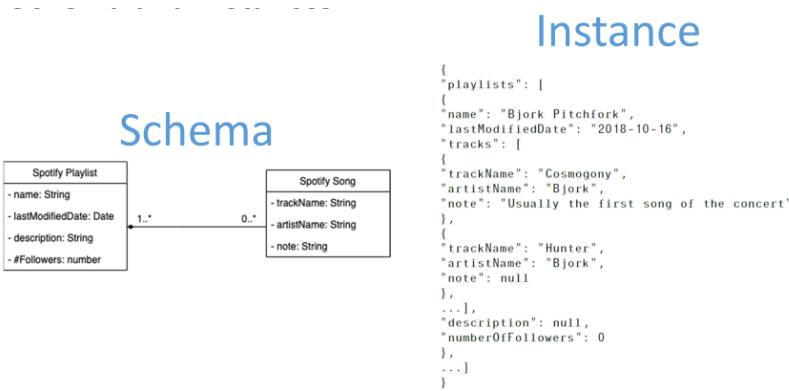


6.2 Kinds of Heterogeneity

El problema real que tenemos es la heterogeneidad de semántica (la del significado de los datos, que quieren decir), no de sistemas.

6.2.1 Schema and instances

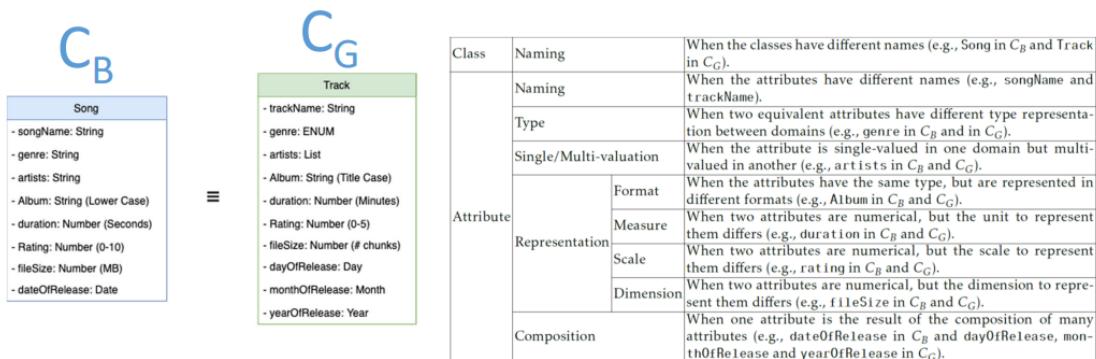
Los problemas de heterogenidad aparecen independientemente de la representación de los datos (esquema, instancia)



6.2.2 Intra-class heterogeneity

Entre las diferentes tablas que se refieren al mismo concepto.

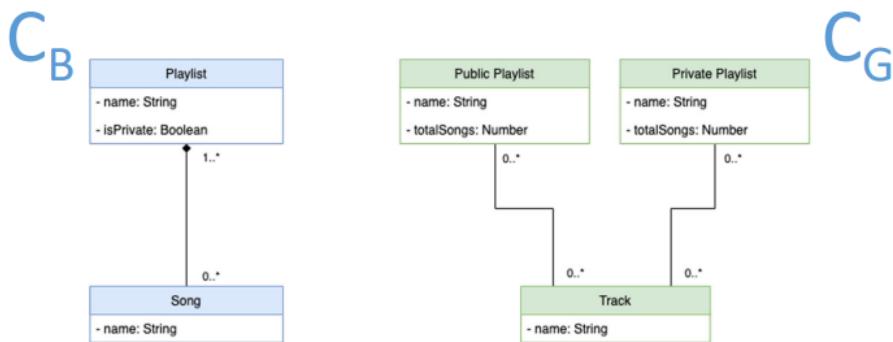
- El nombre de ambas tablas tienen un nombre diferente.
- Entre atributos:
 - El **nombre** entre los atributos es distinto (*songName* - *trackName*).
 - El **tipo de dato** de los atributos es distinto (*String* - *ENUM*).
 - El atributo puede ser **multi o univaluado** (*String* - *List*).
 - La **representación** del atributo: (formato (*Upper, Lower*), medida (*sec, min*), escala (*0-5, 0-10*), dimensión (*MB, GB*))
 - De **composición**. P.e que la date de formato "dd-MM-yyyy" este separado en tres atributos para cada componente.



6.2.3 Inter-class heterogeneity

Relación entre las clases.

- **División de la clase en subclase.** Se tendría que ver si la unión de las subclases corresponde a la clase. (*Playlist* = *Public Playlist* + *Private Playlist*)
- **De agregación:**
 - **Composición:** 1 playlist guarda dentro toda la información de las canciones (por el rombo negro de la flecha). En el de la derecha como no tienen el rombo negro lo que guardarían las playlist serían punteros hacia los diferentes tracks.
 - **Datos derivados:** en C_G el *totalSongs* depende de la tabla Track, pero nos podemos guardar ese valor. En el caso de C_B no perdemos ese valor, cuando queremos el número de canciones contamos la tabla Song.



Super/Subclass of		When a Class in a schema is a generalization or specification of a class in another (e.g., <i>Playlist</i> in C_B and <i>Public/Private Playlist</i> in C_G).
Aggregation	Composition	When a class is composed by an aggregation of another class in one domain, but is an independent class in another (e.g., <i>Song</i> is aggregated inside <i>Playlist</i> in C_B while <i>Track</i> has simple associations with <i>Public/Private Playlist</i> in C_G).
	Derivation	When the characterization of a class is derived from the aggregation of another (e.g., <i>Public/Private Playlist</i> in C_G have <i>totalSongs</i> , an attribute defined by the aggregation of <i>Track</i> in C_G . While in C_B such dependency is absent).

6.2.4 Data-Metadata heterogeneity

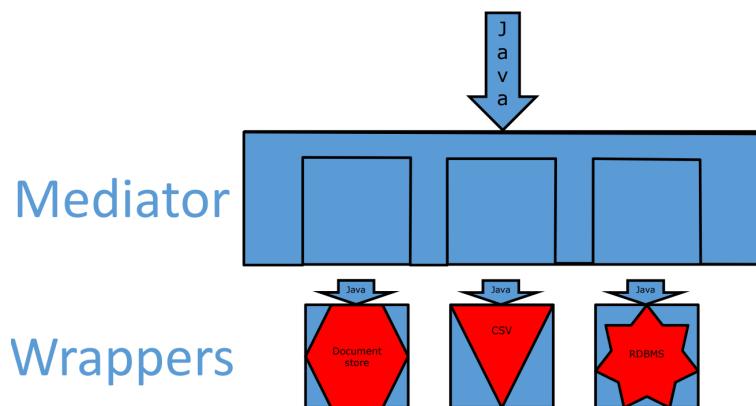
Ocurre cuando en un lado se guardan datos y en el otro meta-datos. P.e. *Rock*, *Pop*, *Rap* son metadatos (ya que se refieren al nombre de la tabla) y en la otra representación serían instancia (datos).



La figura de la izquierda (Rock, Pop, Rap) son metadatos.

6.3 Overcoming System Heterogeneity

Se hace mediante una arquitectura de tipo **Wrappers-Mediator**. Lo que queremos es crear una capa de software (*Mediator*) que lo que hace es ofrecer al usuario (que quiere acceder al sistema de forma integrada) una interficie tipo JAVA en la cual puede resolver sus consultas que necesiten acceder a los sistemas (fuentes de datos). El **Mediator** tiene que acceder a los tres sistemas, pero como los sistemas no tienen la misma estructura, se necesita el **Wrapper**.



Wrapper es una capa de software que se encarga de esconder la complejidad de cada uno de los sistemas al **Mediator**. El **Mediator** vería los tres sistemas como si fueran absolutamente idénticos (aunque los sistemas tengan características diferentes). Unifica la interficie hacia las fuentes.

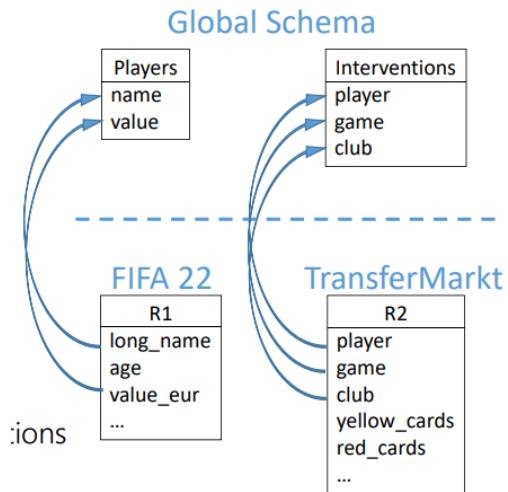
Por lo tanto el mediador es el encargado de deshacer la consulta del usuario en las consultas para cada una de las fuentes y después coger los resultados de cada una de las fuentes y ponerlos juntos para conformar la respuesta única. Los wrappers serían los encargados de simplificar la vida al mediador y que este no tuviera que distinguir el sistema de cada una de las fuentes.

6.4 Schema Integration

La integración del esquema es el primer paso para integrar los datos. Pero antes se deben alinear los esquemas y crear los **mappings** (las correspondencias entre las fuentes y el esquema global que se le ofrece al usuario).

6.4.1 Alignment Process

Como entrada, tiene los esquemas de las diferentes fuentes. El resultado obtendríamos sería el esquema global/integrado/mediador (que utiliza el usuario en las consultas). Después se buscarían las correspondencias entre los atributos de las fuentes de datos y el esquema global definido. Finalmente se tendrían que ver que transformaciones padecen los atributos (cambio de moneda, normalización, etc.).



6.4.2 Mapping

Tipos de mappings se definen y que mecanismo utilizamos para implementarlos:

- **Consistente/coherente (sound):** En el sentido de que la consulta que nos da todos los datos de la fuente es un subconjunto de la consulta que quiere el usuario (hay datos de otras fuentes que no se incluyen en la consulta).

$$q_{\text{obtained}} \subset q_{\text{desired}}$$

- **Completa:** En el sentido que lo que quiere el usuario final es un subconjunto de todo lo que tenemos en la fuente. Hay cosas en la fuente que el usuario no está interesado.

$$q_{\text{desired}} \subset q_{\text{obtained}}$$

- **Exacta:** Cumple las dos anteriores.

$$q_{\text{desired}} = q_{\text{obtained}}$$

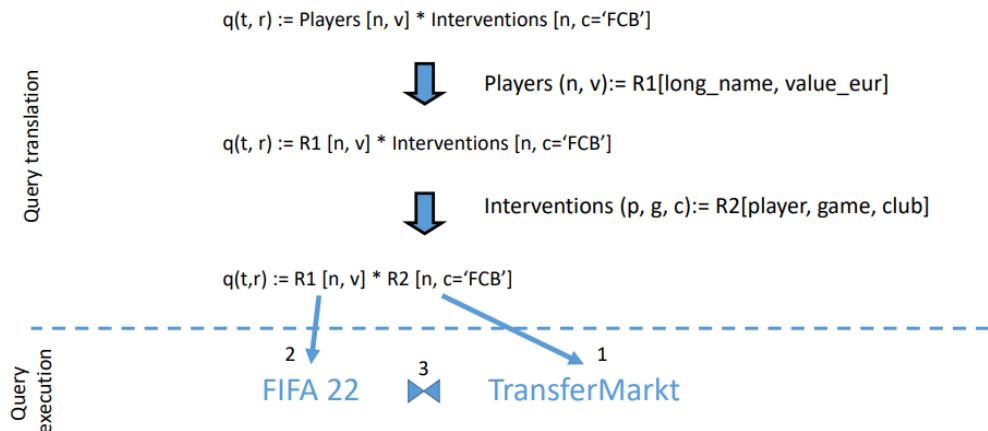
Como se implementan estas correspondencias en el mediador? La solución más simple es la **Global As View**, donde el esquema global se expresa como una vista sobre las fuentes de datos.

- Se define como una vista sobre las fuentes.
- Relativamente sencillo pero se tiene que ir definiendo uno por uno cada elemento del esquema global con su consulta correspondiente (mapping).
- Lo bueno es que el mecanismo de consulta para implementar el mediator es muy sencillo (lo mismo que los SGBD para traducir las vistas).
- Rígido a los cambios de las fuentes.

6.4.3 Example of Global as View (GAV)

- Global schema
 - Players (name, value)
 - Interventions (player, game, club)
- Local schemas
 - FIFA 22
 - R1 (long_name, age, value_eur, ...)
 - TransferMarkt
 - R2 (player, game, club, yellow_cards, red_cards, ...)
- Mappings
 - Players (n, v):= R1[long_name, value_eur]
 - Interventions (p, g, c):= R2[player, game, club]

Tenemos en forma de tablas nuestros esquemas globales (jugadores y intervenciones) y las fuentes de datos (FIFA 22: R1 y TransferMarket R2). El mapping es simplemente una consulta que implementamos con álgebra relacional. Estamos haciendo una proyección de la fuente cogiendo los atributos que nos interesan.



Debemos traducir la consulta que esta expresada en términos del esquema global a las fuentes de datos. El usuario final expresaría la consulta con esa terminología.

El hecho que la variable n sea la misma significa que es el atributo de join. El $c = 'FCB'$ indica una selección. Una vez que el usuario nos ha indicado su consulta utilizamos los mappings para traducir las tablas al formato correcto sobre las fuentes de datos. El esquema global es una vista, que no es más que una consulta.

Una vez hecho esto pasamos la siguiente fase (todo lo anterior lo hace el mediator). En este caso tendríamos que decidir a que fuente accedemos primero (R1 o R2), como hay selección en R2 seguramente se empezaría por esa (ya que esperamos que tenga menos filas). Entonces haríamos una consulta al wrapper de TransferMarket. Una vez tengamos el resultado hacemos la consulta sobre el wrapper de FIFA22 y una vez tengamos ambos resultados haríamos el *join* entre ellos para obtener el resultado único para el usuario.

6.5 Data Integration

Mirar las instancias de las tablas para ver si coinciden y representan el mismo objeto del mundo real. Para hacer esto hay que seguir dos pasos:

1. **Entity Resolution (Record matching)**: Ver cuando dos filas representan el mismo objeto. Necesitamos una función que identifique los objetos, cuando dos filas correspondan al mismo objeto nos lo diga.
2. **Fusionar**: Única fila que represente las múltiples filas que coinciden. Deberíamos homogenizar los esquemas de estas tuplas.

6.5.1 Entity resolution

Lo que nos interesa es ver si dos filas corresponden al mismo objeto. Varios problemas dependiendo del tipo de datos (error de escritura, cambios temporales, abreviaciones, variantes y más mamadas). Se busca simplificar y tener una normalización de los datos.

Función de similitud \approx :

- Basadas en distancias (*Hamming, Levenshtein, Jaccard* $\frac{|A \cap B|}{|A \cup B|}$)
- Basadas en heurísticas
- Probabilísticas
- Modelos de clasificación

Función de fusión \wedge : Si coinciden, FANTÁSTICO! Pero si no hay varias posibilidades:

- Generar un valor null.
- Quedarme los múltiples valores.
- Quedarme con el de la fuente más confiable.
- Usar *crowdsourcing*. Preguntarle al proletariado.

6.5.2 Algoritmo R-Swoosh

Para un único DB con repetidos dentro: O es el output, I input.

```
O := {}
while I ≠ {} do
    escoger r ∈ I
    if (exists s ∈ O tal que s ≈ r) then
        I := I - r; O := O - s; I := I ∪ {r ∧ s}
    else
        I := I - r; O := O ∪ {r}
    end if
end while
```

El resultado del *if* se vuelve a meter al input la fusión de *r* y *s*.

7 Distributed Data Management

7.1 Dsitrubuted Systems

Un conjunto de componentes que interactúan para cumplir un objetivo común y estos componentes que interactúan, lo hacen a través de una red de ordenadores mediante el paso de mensajes.

Los componentes son completamente independientes. Trabajan de forma totalmente concurrente y pueden hacer cosas en paralelo. Si uno falla, el resto continua trabajando igualmente (siempre y cuando el sistema este preparado para cumplir el objetivo común sin el componente).

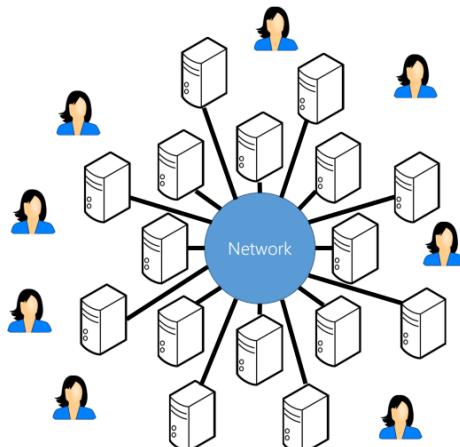
El problema que tenemos es la sincronización, pues no tenemos un "reloj" común. Como están en maquinas diferentes, diferentes puntos de la red puede tardar diferente tiempo para que llegue un mensaje. Se pueden sincronizar con un cierto error y con ciertas dificultades.

- Escalabilidad
- Calidad del servicio: temas de rendimiento, confiabilidad, disponibilidad,
- Conurrencia
- Transparencia

7.1.1 Escalabilidad

Tenemos más usuarios que antes y tenemos dos soluciones para mitigar esta dificultad:

- **Scale up:** Agregar nuevos componentes en una única maquina. *No es una buena opción, porque no es un sistema distribuido, sino que agregariámos ya sea memoria, nuevos procesadores a una única maquina.*
- **Scale out:** Agregar nuevos componentes independientes conectados a la red. Los componentes tienen que hacer propia una proporción de la carga de trabajo. Al añadir un componente, de manera automática, este tiene que hacerse cargo de cierta carga de trabajo. Se tiene que hacer un balance de la carga de trabajo. Se tiene que evitar el tema de **cuellos de botella**, no puede ser que una única maquina de todas las nuevas conectadas, este haciéndose cargo de todas las tareas. Todo esto evitando la comunicación innecesaria.



7.1.2 Performance/Efficiency

La **latencia** es tiempo que tarda a empezar a recibirse la respuesta a nuestra petición y el **throughput** es el número de peticiones que se pueden responder por unidad de tiempo (min, ss, etc.). Se busca minimizar las **latencias** del sistema y maximizar el **throughput**.

Para mejorar el rendimiento podemos hacer uso de:

- **Paralelismo:** los diferentes componentes responden de manera paralela.
- **Optimizar el uso de la red:** Intentando no colapsar la comunicación (minimizando los mensajes).
- **Técnicas específicas** dependiendo del tipo de sistema distribuido que estemos tratando. (En nuestro caso nos interesan los sistemas distribuidos que gestionan datos).

7.1.3 Reliability/Availability

Nos interesa que mantenga los datos consistentes y nos interesa que el sistema se mantenga en todo momento en funcionamiento. Como los componentes son independientes, si falla un componente nos la chupa, podemos usar los otros. Pero el sistema como tal debe poder reaccionar a las fallas de los componentes.

- Lo que esperamos es que el sistema siga trabajando, pero eso significa que la información que tiene el componente que ha fallado tiene que estar en otro lugar (**replicación**).
- El **rotamiento** de los mensajes tiene que ser flexible. Si, cuando se hace una petición, se destina a un componente en concreto y este falla, entonces el mensaje fallará. Por lo tanto, se tiene que enviar el mensaje/petición de forma genérica y los componentes tienen que hacer suya esta petición dependiendo de cuál esté disponible en ese momento.
- Para saber qué ha fallado, se utiliza normalmente un mecanismo de **heartbeats**, lo que significa que los diferentes componentes simplemente van avisando al resto de que están vivos.
- Si la máquina falla, se debe poder ponerla en marcha otra vez. Cuando hay muchos componentes, no se puede hacer esta recuperación manual, sino que se debe recuperar de forma automática.

7.1.4 Concurrency

La idea es que varios usuarios puedan acceder al sistema al mismo tiempo y hacer peticiones de forma concurrente.

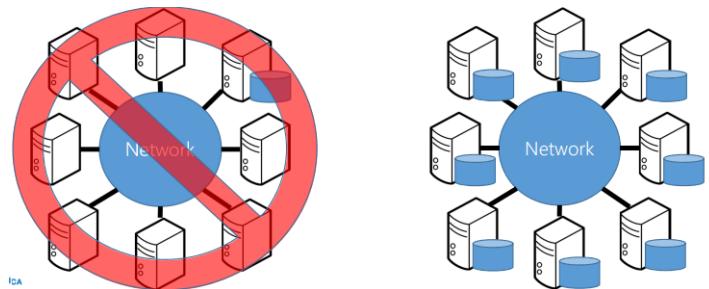
Sistemas de consenso: los componentes reaccionan a las peticiones/fallas de forma independiente y autónoma. Pero el hecho de hacer todo esto tiene que evitar que interfieran entre ellos (bloqueos, *deadlocks*, realización múltiple de una misma tarea, ignorar tareas, etc).

7.1.5 Transparency

Hacer todo lo que hemos dicho antes sin que el usuario se dé cuenta. Tenemos un sistema distribuido, pero desde el punto de vista del usuario es una caja negra. El usuario interactúa con la interfaz/caja que esconde todas las dificultades de implementación. Lo que hace es ocultar el resto de dificultades. Esta es la característica más complicada de todas.

7.2 Distributed Database Systems

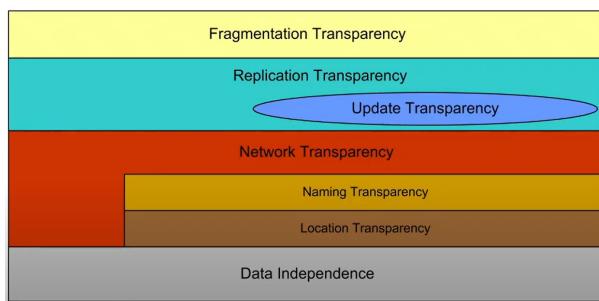
Diferentes bases de datos que están integradas, físicamente distribuidas y la distribución tendría que ser transparente para los usuarios.



La de la izquierda no es un sistema de bases de datos distribuida porque está en un único punto físico aunque este disponible para acceder mediante la red. En el segundo caso, si que se ve que tenemos datos en diferentes puntos de la red y todos ellos son accesibles.

El hecho de que los datos estén distribuidos a través de la red, quiere decir que tenemos una serie de dificultades para accederlas. Trataremos que estas dificultades no afecten el trabajo de nuestros usuarios. Para hacer esto se consideran diferentes niveles de independencia.

1. **Independencia lógica y física:** No sabemos realmente en qué ficheros están guardadas los datos y ofrecemos a los usuarios diferentes mecanismos para que puedan transformar mínimamente estos datos mediante vistas. El primer nivel de independencia no tiene que ver con la distribución de los datos, sino que también aparece en sistemas de bases de datos centralizadas.
2. **Transparencia de red:** No queremos que los usuarios sean conscientes de que los datos estén en diferentes puntos de la red (diferentes máquinas) ni que las diferentes máquinas asignen diferentes nombres a los elementos equivalentes.
3. **Transparencia de replicación:** La replicación de los datos favorece desde un punto de vista de la eficiencia. El usuario no debe ser consciente de la replicación de los datos. El hecho de que haya diferentes réplicas no significa que el usuario modifique cada una.
4. **Transparencia de fragmentación:** Nos interesa que los datos no estén todas de forma monolítica en un único lugar. Sino que se dividen y se accede solo al trozo de datos que nos interesa. Poner solo en una máquina los datos que son relevantes para ese sitio. El usuario no debe ser consciente de esta partición de los datos.



Los niveles de transparencia implican diferentes niveles de autonomía. Si consideramos que cada repositorio/máquina puede funcionar de forma autónoma, cuanta más autonomía, menos transparencia y viceversa.

Esto da lugar a la siguiente clasificación:

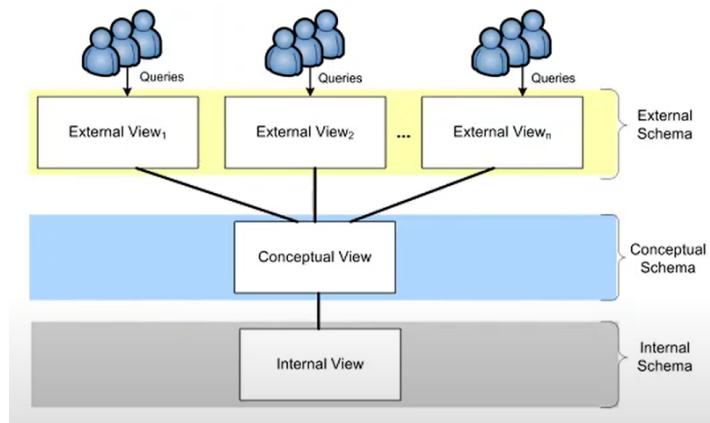
	Autonomy	Central schema	Query transparency	Update transparency
DDBMS	No	Yes	Yes	Yes
T.C. Federated	Low	Yes	Yes	Limited
L.C. Federated	Medium	No	Yes	Limited
Multi-database	High	No	No	No

En *multi-database* vemos que no hay transparencia de ningún tipo. No tiene transparencia de esquema, no hay un esquema único en caso de tener múltiples bases de datos autónomas. Las consultas y modificaciones se tienen que dirigir a la base de datos concreta.

Se tratarán únicamente del primer caso, en el que las diferentes lugares de la red no tiene ningún tipo de autonomía y por lo tanto nos dan una cierta transparencia a la hora de gestionar el esquema de los datos y para la gestión de consultas y modificaciones en el sistema.

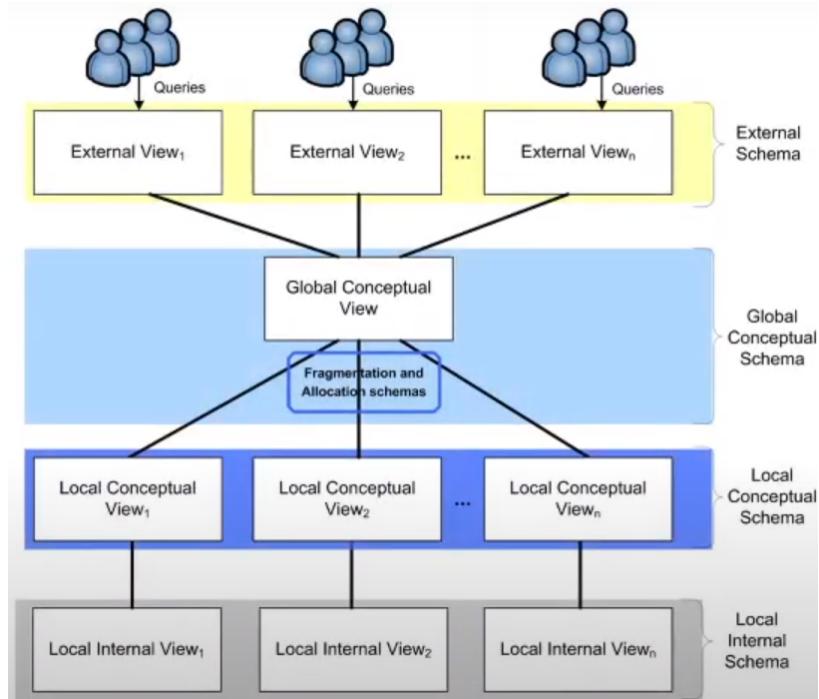
7.2.1 Extended ANSI-SPARC Architecture of Schemas

Tres niveles de esquemas, para el caso en el que tengamos una sola máquina.



1. El **esquema interno** serían los ficheros del sistema operativo. Pueden estar ordenados, en diferentes lugares, etc.
2. El **esquema conceptual** del SGBD serían las tablas, aunque se le llame esquema conceptual.
3. El **esquema externo** serían las diferentes vistas que se le pueden ofrecer a los diferentes usuarios que tengamos. A cada usuario se le pueden definir diferentes vistas.

En el caso que tengamos más de una máquina (caso de un sistema distribuido pero sin autonomía). Tenemos prácticamente la misma situación.



Tanto el esquema interno como externo se mantienen tal y como estaban. El esquema conceptual se divide en dos partes.

- Tendremos un **coordinador** que gestionará un esquema global (*global conceptual esquema*).
- Tendremos los diferentes **workers** (*local conceptual esquema*) que gestionarán diferentes esquemas locales. Cada esquema local tendrá asociado a su máquina su esquema interno, pero este esquema es exactamente igual al esquema interno en el caso centralizado, lo que pasa es que ahora lo tenemos muchas veces.

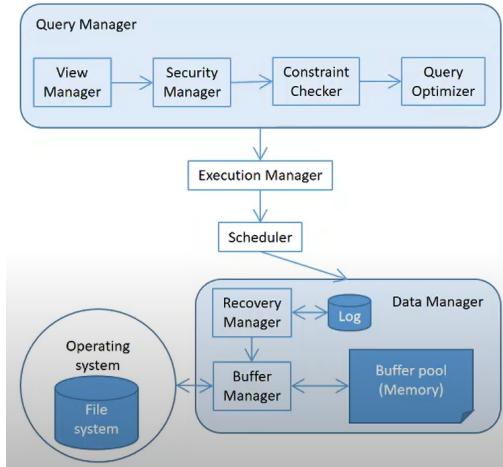
Lo que nos falta es el ligamiento que hay entre el esquema global y los esquemas locales. Vemos que tenemos un catálogo global (donde guardamos los metadatos del sistema) que guarda los *mappings* entre los esquemas externos y el esquema global. Por otro lado, tenemos en cada uno de los nodos un catálogo local que guarda los esquemas conceptuales locales y como se relacionan con los esquemas internos.

Estos mapping ya existían anteriormente en la arquitectura centralizada. Lo que es nuevo son los mappings entre esquema conceptual global (coordinador del sistema) con las correspondencias de lo que guardan cada uno de los **workers** (esquemas conceptuales locales). Lo que guardan son primero que fragmentos hay, como se ha trozado cada uno de los datasets y donde están localizados cada uno de estos datasets.

Se consigue esta transparencia de los esquemas. El usuario solo tendrá que acceder sobre los esquemas externos (vistas).

7.2.2 Centralized DBMS Functional Architecture

Se centra en la cuestión de la transparencia de las consultas. Igual que antes, miramos primero lo que era una arquitectura funcional de un sistema centralizado.

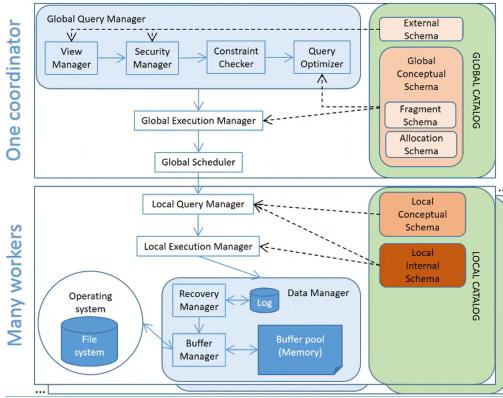


- El **Manager query** se encargaría de diferentes fases:
 - **Security Manager**: Tendríamos GRANT y REVOKE para controlar el acceso a la base de datos. Para controlar la seguridad.
 - **Constarin check**: si se trata de inserts, updates o deletes. Tenemos que mirar que las restricciones de seguridad se cumplan.
 - **Query optimizer**: gestor de los planes de procesos.
- **Execution Manager**: recibe como entrada el plan de procesos del Query Optimizer. Tiene algoritmo predefinidos que implementan las acciones (seleccion, join).
- **Scheduler**: sería como una especie de semáforo (lock) para evitar que diferentes usuarios se interfieran entre ellos. Da aislamiento entre los diferentes usuarios.
- Finalmente los datos llegarían al **Data Manager**. Este se encarga de poder trabajar en la memoria (mas eficiente que el disco).
 - **Buffer Manager**: cuando el usuario/consulta requiere unos ciertos datos, se traen del disco a la memoria y cuando los datos se modifican y se necesita liberar memoria, lo que hace es llevarlas al sistema de ficheros. Garantiza que las modificaciones se hagan en memoria (pues resultan más eficiente) pero al final vayan a parar al disco para preservarlas.
 - En caso de que falle cualquier cosa (se vaya la luz p.e.), tenemos un sistema de Log y un **Recovery Manager** que gestiona el Log para garantizar que no se pierda nada de lo que hayamos hecho.

En el caso de tener un sistema distribuido, lo único que pasa es que tenemos una parte del sistema que es global. Este sería nuestro **coordinador** que es único. tenemos una parte que es local que serían los **workers**, cada uno con las misma estructura porque hemos dicho que no tienen ninguna autonomía.

En este caso el **Query Manager** es global, que está en el nodo coordinador. Por dentro tenemos la misma estructura de antes. La diferencia es que el **Query optimizer** y el **Global Executor Manager** tienen que tener en cuenta los mappings de fragmentos y localización de los datos.

Al igual que el **Execution Manager** se divide en global y local, también es el caso del **Query Manager**.



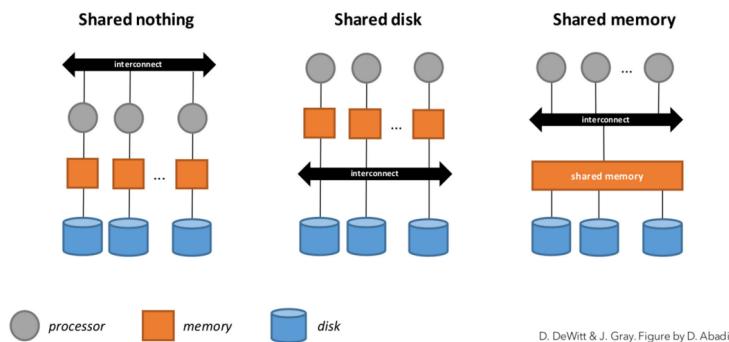
Los **Local Query Manager** en este caso ya no sabe nada sobre la localización ni fragmentación de los datos. Lo único que sabe es lo que tiene el, para eso es el esquema local, para trabajar a un nivel de abstracción (tablas, vistas, p.e.).

7.3 Cloud Databases

Son un tipo de bases de datos distribuidas. A su vez, las bases de datos distribuidas son un tipo de sistemas distribuidos. Por lo tanto todo lo que se aplica a sistemas distribuidos se aplica a las otras dos.

7.3.1 Parallel database architectures

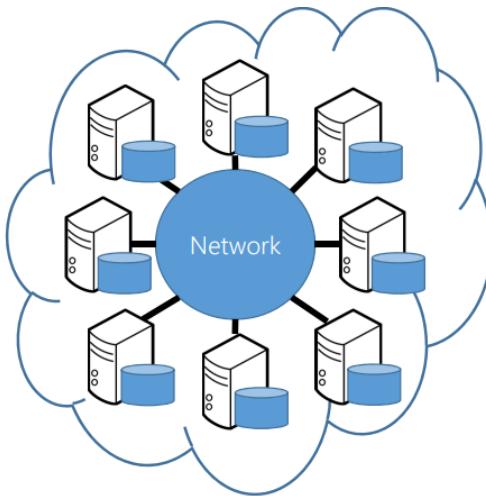
Nesitamos entender en qué entorno estamos. Cuando se habla de sistemas paralelos, tenemos tres opciones.



1. **Comparten memoria:** Tenemos diferentes procesadores que están interconectados pero todos ellos acceden a la misma zona de memoria. Todos pueden acceder a las mismas zonas de memoria y al disco. Podría haber conflictos que requieren coordinación para no *machacar* uno al otro dentro de la memoria. Relativamente complejo y escalabilidad limitada.
2. **Comparten disco:** Interconectamos los discos. Para eso haría falta un hardware específico que nos ayude a conectar los discos y los procesadores podrían acceder a los discos. Arquitectura que puede funcionar en paralelo pero tampoco tenemos mucha escalabilidad. Limitados por el hardware.
3. **Comparten nada:** Maquinas diferentes (independientes), cada una de ellas con su procesador, memoria y disco. Todas están interconectadas a través de la red. Si

se quiere acceder al disco de una otra, lo tiene que hacer a través de la red.

Esto genera una serie de características especiales, que son similares a las de un sistema distribuido comentadas en secciones anteriores.



Una nueva característica importante es que, como los SGBD funcionan en el cloud, normalmente se ofrecen como servicio y no como software. para mantener el servicio rentable se necesitan muchos usuarios que accedan continuamente. Esto es lo que se llama **Multi-tenancy**. Esto trae unas consecuencias desde el punto de vista del proveedor (somos el señor amazon).

- La popularidad del servicio puede cambiar y no se tiene ningún control sobre esto. Puede haber ya sea un aumento de comandas, de usuarios (*flash crowds*) o una disminución repentina y se tiene que soportar.
- Los usuarios son diferentes, con lo que el aumento lineal de usuarios no implica un aumento lineal de recursos. Estos requerimientos son variables dependiendo de los requisitos de los usuarios.
- Se necesita automatizar (a mano no) las cosas para soportar el gran numero de usuarios. Se necesitan metadatos para conocer las características, requisitos de cada uno de los usuarios.
- Los fallos se deben detectar y solventar automáticamente por parte del sistema. No puede ser que si una falle, todo el sistema falle.
- **Scale-out** es inevitable pero esto no debe para el servicio cada vez que se aumente maquinas. Incluyendo parches o updates obviamente.
- El balance de carga se tiene que hacer de forma **elástica**. Si se necesitan más maquinas, algunas pueden asumir una carga mayor para liberar el resto y al revés (poder desconectar algunas maquinas cuando no sean usadas y no se cobre su uso).

Desde el punto de vista del cliente tenemos 4 problemas principales.

1. **Diseño de los datos:** Se tendrán que analizar tres aspectos para ello: como fragmentamos los datos, como colocamos los fragmentos y cuantas copias ponemos (entre mas copias mejor para el rendimiento pero mas problemas de update).
2. **Gestión del catálogo:** El mismo análisis de los tres aspectos pero en lugar de hacerlo sobre datos se hace sobre metadatos.

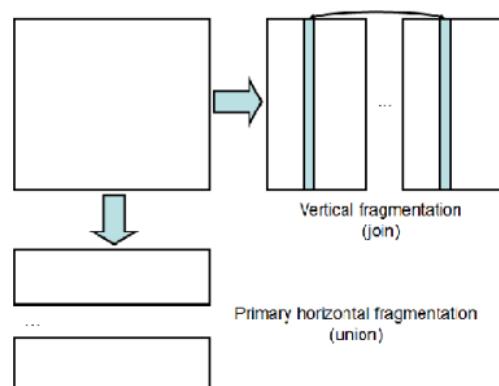
3. **Transacciones:** Se tienen que respetar las propiedades ACID en la medida de lo posible. Necesitamos un mínimo de sistema de recuperación, de control de concurrencia y consistencia.
4. **Procesado de queries:** Como optimizamos las consultas. Los gestores de consultas en el cloud son todavía limitados hasta cierto sentido, no porque sean malos, sino porque su espacio de búsqueda es más grande (porque incluyen distribución y/o paralelismo y replicaciones) que el de un SGBD relacional. No siempre se encuentra el mejor plan de acceso de forma tan eficiente.

7.4 (Distributed) Data Design

Diseño de las bases de datos de cloud (NoSQL). El primer problema es como se resuelve la fragmentación y alocación de los datos.

7.4.1 Data Fragmentation

Dada una tabla/dataset se pueden hacer fragmentaciones horizontales o verticales (Se definen subconjuntos de filas/columnas).



Para deshacer la fragmentación horizontal simplemente se tiene que hacer una UNION. La vertical se tiene que hacer un JOIN (es decir que se necesita la clave primaria en los múltiples fragmentos para poder hacer la operación).

Ventajas:

- Para acceder solamente a un subconjunto de filas/columnas que nos interesa en la consulta.
- Normalmente son necesarios diferentes conjuntos en diferentes lugares (maquinas) por motivos específicos.
- Facilita el paralelismo, pueden procesar cada fragmento a la vez.

Dificultades:

- La gestión del catálogo. Se ha de dejar constancia que fragmentos tiene cada tabla, donde estan y el número de copias.
- El uso de JOIN es costoso. Puede no valer la pena, especialmente si se requiere comunicación a través de la red.
- Si se tiene que modificar una cosa y esta en diferentes fragmentos/replicas se vuelve más costoso.

Entonces se tiene que tener en cuenta una serie de características que tiene que cumplir la estrategia de fragmentación para que sea correcta.

- **Completitud:** cada dato tiene que estar como mínimo en un lugar.
- **Disjunción:** los fragmentos tienen que ser disjuntos. No puede haber redundancia. Excepción al atributo de JOIN (aunque no es redundancia porque no es evitable).
- **Reconstruible:** la tabla original se tiene que poder reconstruir a partir de los fragmentos.

Encontrar la mejor estrategia es un problema computacionalmente complejo NP-hard. Porque depende de la frecuencia de cada consulta, de su plan de acceso, entre otras cosas.

7.4.2 Data Allocation

Nuevamente es un problema NP-hard. Igual que antes depende de diferentes factores: donde se hace la consulta, las estrategias de procesado de query y sobre todo el workload que tengamos (normalmente no es predecible). Una simplificación del problema es intentar minimizar únicamente el coste de comunicación de enviar datos por los datos, el cual suele ser el principal cuello de botella.

7.4.3 Data Replication

Se puede ver como una generalización del problema de guardar los datos. En este caso de localizar los datos en un único lugar, se puede decidir localizarlas en múltiples.

Proporciona mas alternativas de ejecución, aunque la búsqueda es mayor. Mejora la disponibilidad en caso de que una maquina falle, tenemos replicas de los datos en otras ubicaciones. En cambio suele haber problemas de consistencia en las actualizaciones (en entornos read-only esto no sucede).

7.5 Distributed Catalog Management

Se tienen los mismo problemas que en el diseño de los datos. Fragmentación, Localización y Replicación. Son los mismo problemas y se pueden hacer las mismas consideraciones. La diferencia principal es que en los primeros dos puntos, hay diferencia en metadatos locales y globales.

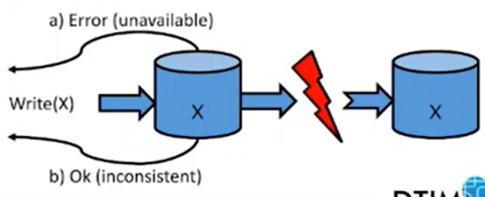
8 Distributed Data Processing

8.1 (Distributed) Transaction Management

8.1.1 CAP theorem

El sistema en cuestión presenta tres características fundamentales: **consistencia** (asegura la consistencia de los datos, especialmente entre réplicas), **disponibilidad** (los usuarios pueden llevar a cabo operaciones) y **tolerancia** (capacidad de recuperarse de fallos de red que desconectan componentes del sistema).

El teorema establece que de estas tres características, solo es posible cumplir simultáneamente con dos.



En la Figura se ilustra un caso en el que el sistema cuenta con dos réplicas de x (donde x puede representar cualquier cosa, como un conjunto de datos o una fila). En esta situación el sistema está caído, asumimos la presencia de la tercera característica. Es crucial notar que de las otras dos características, solo se puede tener una a la vez.

Cuando se intenta escribir en X durante la caída de la red, se produce un error indicando que el sistema no está disponible. La segunda opción sería aceptar la escritura, pero resultaría en una inconsistencia en el sistema. Esto se debe a que solo se puede escribir en una parte de la red (ya que la otra parte no es accesible para el usuario ni para la operación). Como resultado, la X en el lado izquierdo tendría un valor, mientras que la X en el lado derecho tendría otro, generando una inconsistencia.

Configuration alternatives

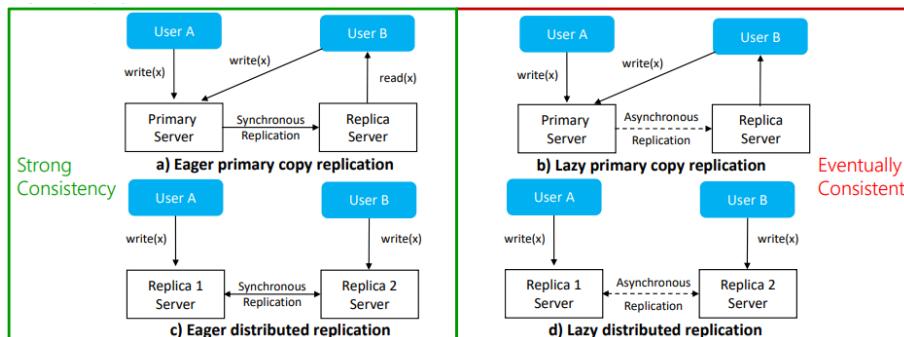
- **Consistencia total:** Renunciamos a la disponibilidad; en este caso, las réplicas se sincronizan de forma síncrona. En el momento en que recibo una modificación, modifco todas las réplicas de manera que todo el sistema sea consistente. Esto se logra siempre y cuando la red esté disponible. Si la red está caída, esta configuración no puede llevarse a cabo.
- **Eventualmente consistente:** Renunciamos a la consistencia. Aquí, los cambios se propagan a las réplicas de manera asíncrona. Recibo una modificación y poco a poco vamos propagando los cambios. Si la red está caída, no ocurre nada; simplemente esperamos a que vuelva a estar disponible. Esto significa que el sistema siempre estará disponible. Siempre recibiré operaciones y las aceptaré perfectamente. En caso de caída de la red, simplemente tomará más tiempo sincronizar, y durante este tiempo el sistema no será consistente.
- **Datos no distribuidos:** Renunciamos al particionamiento, ya que no es realista en un entorno de red. La única manera de que la red no caiga es que no exista. Si existe, es susceptible a caídas. Renunciamos a estar en la nube; estamos en un servidor con una única máquina, por lo que podemos tener disponibilidad y consistencia.

No hay una configuración que sea universalmente buena; todo depende de nuestro contexto.

8.1.2 Managing replicas

La replicación de los fragmentos mejora la latencia y la disponibilidad pero crea un problema de sincronización. Entonces tenemos 4 posibilidades, dependiendo de si modificamos la copia primaria (si existe), que es la única que podemos modificar, o permitimos modificar cualquier copia. La segunda decisión que tenemos que tomar es si la sincronización es inmediata o diferida (*sin prisa*).

- **Eager primary copy replication:** Podemos modificar solo la copia primaria; obviamente, podemos leer de cualquier copia e inmediatamente, de forma síncrona (inmediata), cuando guardamos la modificación en la copia primaria, la guardamos en cualquier otra copia que tengamos.
- **Lazy primary copy replication:** Igual que antes, todas las modificaciones van solo a la copia primaria. Podemos leer de cualquier copia, pero la propagación es discontinua, lo que indica que es asíncrona. En el momento que recibimos la modificación en la copia primaria, sin prisa vamos propagando estos cambios a las demás copias.
- **Eager distributed replication:** Aceptamos escrituras tanto en una copia como en la otra. Si tenemos 50, aceptaríamos escrituras en las 50 y luego las sincronizaríamos de manera inmediata. Quien reciba la modificación inmediatamente se propaga a las demás replicas.
- **Lazy distributed replication:** Recibimos modificaciones en cualquier réplica y propagamos sin prisa de una a la otra. Quien reciba primero lo confirma al usuario que ha recibido la modificación correcta y luego propaga sin prisa a las demás copias que haya.



8.1.3 Replication Management Configuration

Si definimos el comportamiento del sistema en términos de 3 variables, N, W y R, podemos analizar las configuraciones que tenemos:

- **N:** número de replicas (máquinas).
- **W:** número de replicas que tienen que estar escritas antes de hacer commit, antes de confirmarle al usuario.
- **R:** número de replicas que necesitamos leer antes de darle al usuario lo que nos está pidiendo. Si sabemos que el sistema es totalmente síncrono, no tiene sentido pedir más de una lectura, ya que sabremos que serán todas iguales. Tiene sentido elegir una $R > 1$ cuando sabemos que el sistema es asíncrono.

Named Situations

- **Ventana de inconsistencia** $W < N$: escribimos en menos replicas de las que tenemos y confirmamos al usuario que ya hemos escrito. Hay una ventana desde que hemos confirmado al usuario hasta que realmente conseguimos escribir en todas las N.
- **Strong consistency** $R + W > N$: aunque haya una diferencia grande entre W y N y tarde en llegar a todas las replicas, como cuando leemos miramos suficientes copias, nos aseguramos de que todo lo que leemos es correcto. Si no hemos escrito suficientes, al menos leemos las suficientes.
- **Eventually consistent** $R + W \leq N$: como son menores que N, podría ser que no se solaparan. Que escribamos en unas ciertas máquinas y leamos de otras, devolviendo al usuario datos incorrectos. En el caso anterior no puede pasar porque como mínimo habrá una intersección entre lo que leo y lo que escribo.
- **Potential conflict** $W < (N + 1)/2$: podría ser que dos usuarios estén escribiendo en dos lugares diferentes de la red, generando un conflicto. Uno modificaría una cosa y el otro modificaría el mismo dato y pondría otra cosa, y no sabemos con cuál quedarnos, cuál es la que ha llegado antes.

Typical Configurations

- **Fault-tolerant system** $N = 3, W = 2, R = 2$: tenemos 3 máquinas; cuando escribimos lo hacemos en 2 máquinas y cuando leemos lo hacemos en 2. Es un sistema totalmente consistente ya que, aunque no escribimos en todas las copias, cuando leemos leemos 2 y, por lo tanto, seguro que una de ellas es una que hemos escrito antes. Si una de las máquinas falla, podemos seguir trabajando, la que falla se ignorará siempre.
- **Massive replication for read scaling** $R = 1$: podemos tener las máquinas que sean; la W es irrelevante. Hacemos réplicas e simplemente pedimos una lectura y ya está.
- **Read one-Write All** $R = 1, W = N$: prioriza las lecturas, lee solo 1, pero el problema es que penaliza mucho las escrituras. Las hacemos síncronas y nos esperamos a haberlas escrito todas, y esto en un entorno cloud puede ser un desastre.

8.2 (Distributed) Query Processing

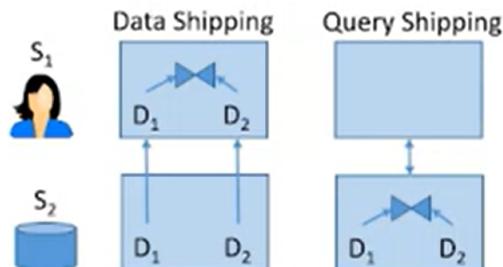
- **Coste de comunicación:** Estamos en un entorno distribuido, por lo tanto, los componentes se tienen que comunicar por la red, lo que tiene un costo mayor. Además, si tenemos que enviar datos **data shipping** (muchos), esto tiene un costo muy elevado. Si hablamos de una red local (LAN), este costo será más reducido que en una red online. Dependiendo del costo de entrada/salida, esto puede variar.
- **Fragmentación/Replicación:** Gestión de los fragmentos y las replicas. Necesitamos metadatos y estadísticas para saber dónde están, cuáles existen, cuántas replicas. Necesitamos gestionar el catálogo para mejorar el procesamiento de las consultas.
- **Join optimization:** Si tenemos fragmentos, los tenemos que juntar. Si son horizontal union, si son verticales con joins. Estaremos generando más joins en este caso, en qué orden los hacemos es relevante. No solo esto, sino que tenemos algoritmos o estrategias especiales que en entornos centralizados no se consideran (tenemos mas opciones de algoritmos en el sistema distribuido).
- **Paralelismo:** Es lo más importante. Queremos paralelizar, aprovechar esta potencia de cálculo que tenemos de las diferentes máquinas a la vez para mejorar el tiempo de respuesta. Decidir quién ejecuta qué. En un entorno centralizado, lo más importante es reducir el número de accesos al disco. En el momento que hablamos

de sistemas distribuidos, en general, lo que hay que hacer es minimizar el envío de datos a través de la red y también hay que tener en cuenta en algunas cosas el costo de entrada/salida.

8.2.1 Allocation Selection

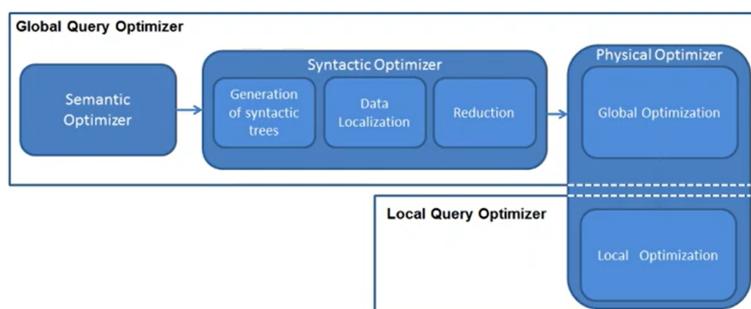
¿Qué podemos hacer para minimizar el envío de datos? Hay que prestar atención a dónde ponemos los datos y qué estrategia de ejecución hacemos.

- **Data shipping:** Enviar los datos donde sea, a una cierta máquina decidimos que ejecutaremos la consulta. Enviamos todos los datos hacia allí, tal vez no es la mejor opción. Esto genera mucha transferencia de datos, que es lo que queremos evitar.
- **Query shipping:** En vez de mover los datos, movemos la operación. La consulta que ha pedido el usuario no la hacemos en la máquina donde está el usuario, sino que le enviamos la consulta a través de la red. El tamaño de la consulta será muy pequeño comparado con el tamaño de los datos. Si los datos son muy pedidos, la máquina se colapsa, por eso puede acabar dando problemas.
- **Hybrid strategy:** Dinámicamente viendo cuál es la carga de cada una de los datos, el uso de la red y la disponibilidad de esta en este momento. Entonces podemos coger una estrategia híbrida donde algunas consultas las enviamos desde donde están los datos, pero si estas máquinas están ocupadas y la red está disponible, tal vez es mejor mover los datos a otro sitio/maquina que no esté tan colapsada en este momento.



8.2.2 Phases of distributed query processing

Tenemos un optimizador de consultas que sigue las mismas 3 fases que sigue cualquier optimizador de consultas relacional tradicional.



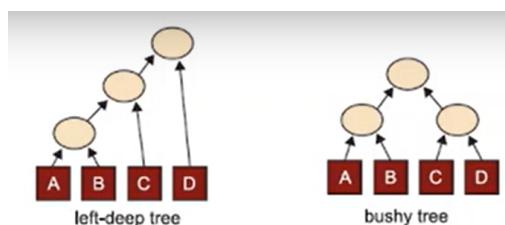
- Optimizador semantico

- **Optimizador sintáctico:** se añade una fase de localización de datos, donde están los fragmentos. También hay que ver si todos estos fragmentos nos hacen falta (**reduction**). Si el dataset tiene 50 atributos pero nuestra consulta solo pide 5 a lo mejor estos no están en todos los fragmentos. recuperando 1 de los fragmentos verticales o 2 a lo mejor ya tenemos estos atributos i los demás no necesitamos recuperarlos. Lo mismo pasa con una fragmentación horizontal. Fase de reducción, ciertos fragmentos, todo i pertenecer al dataset que ‘pide el usuario a lo mejor no hacen falta.
- **Optimizador físico:** tiene una fase global i una local. Lo que hace es dividir la consulta en trozos que envía a los diferentes workers, las maquinas que harán la ejecución. Cada una de estas maquinas ejecutara el trozo que le toque i el optimizador global decidirá como los datos que han generado cada worker como se juntan para darle al usuario un resultado único.

En cada maquina tendríamos una optimización local, que depende del que tenga cada uno de las maquinas en el hardware. El coordinador envía un trozo de la consulta y ya es cuestión de cada worker decidir con sus características propias cual es la manera mas diente de ejecutar ese trozo de la consulta. Cada uno hará lo que crea que sea mejor para sus recurso disponibles y la situación de carga de trabajo-

Optimizador sintáctico

Tiene que generar nuestro árbol sintáctico donde en las hojas están las tablas o los fragmentos y después tenemos nodos que representamos con círculos o elipses que representan los diferentes operadores.



Normalmente en un sistema centralizado lo que se coge es un **left-deep tree**. Esta demostrado que estos arboles de proceso en general si no dan el el óptimo no quedan demasiado lejos, es una heurística para reducir el espacio de búsqueda. Estos arboles son difíciles de paralelizar o no esta claro que se puedan paralelizar.

Mientras que si hacemos un **bushy tree**, no lineal, que tiene diferentes operadores que no tienen nada que ver el uno con el otro. Todos son se pueden paralelizar porque son independientes, uno trabaja con la A i la B mientras que el otro trabaja con la C y la D. En el caso del **left-deep tree** habían dependencias. En este caso el espacio de búsqueda es mayor.

Otras dificultados son que en vez de considerar joins binarias consideramos joins de N entradas, de 3-4 tablas a la vez. Por lo tanto esto añade otras posibilidades a esta generación de arboles de proceso.

El otro problema que tenemos es que estimar el tamaño de los resultados intermedios es mucho más relevante en este caso, ya que nuestro cuello de botella es la red, por lo tanto tenemos que saber que enviamos por ella.

Optimizador físico

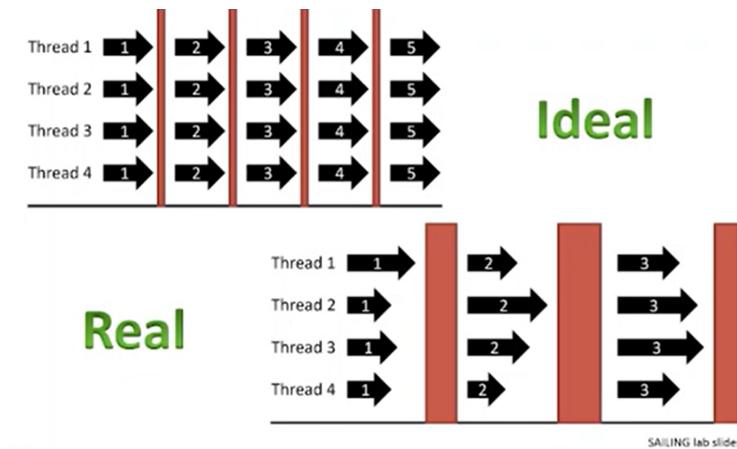
El optimizador físico transforma el árbol sintáctico en un plan más eficiente. Decide el algoritmo que se utiliza (por ejemplo algoritmos de hacer joins) y cual es el mas adiente para cada caso y el método de acceso a los datos (por ejemplo uso de índices). Sobre todo el paralelismo y buscar como explotar la localización de los datos. Tenemos que saber de donde cogemos los datos, si tenemos datos replicados qué réplica nos conviene más coger por ejemplo.

Para escoger qué plan de ejecución primero tenemos que tener en cuenta que la query que quiere el usuario no puede variar, los planes deben de ser equivalentes. Estimaremos los costes de estos y escogeremos la mejor solución.

Normalmente se busca reducir el tiempo de respuesta de la consulta o a latencia. Se busca beneficiarse del paralelismo. Hay dos partes para determinar el coste de la consulta. El coste de los workers/trabajadores/máquinas (ciclos de cpu y operaciones de entrada/salida). El otro coste es el de comunicación, el envío de mensajes de sincronización y de envío de datos.

8.3 Bulk Synchronous Parallel Model

Queremos que todos los threads duren y terminen al mismo tiempo. La realidad es que por la razón que sea, algo tarda más que el resto (hay más datos, una máquina es más rápida que la otra, etc). Esto hace que resulte difícil determinar qué se ejecuta de forma secuencial y qué no.



8.3.1 Kinds of parallelism

- **Inter-query:** diferentes consultas siempre se han podido ejecutar en paralelo. Mientras una consulta lee del disco la otra está utilizando la cpu, ya que estos trabajan de forma independiente.
- **Intra-query:** paralelismo dentro de la propia consulta.
 - **Intra-operator**

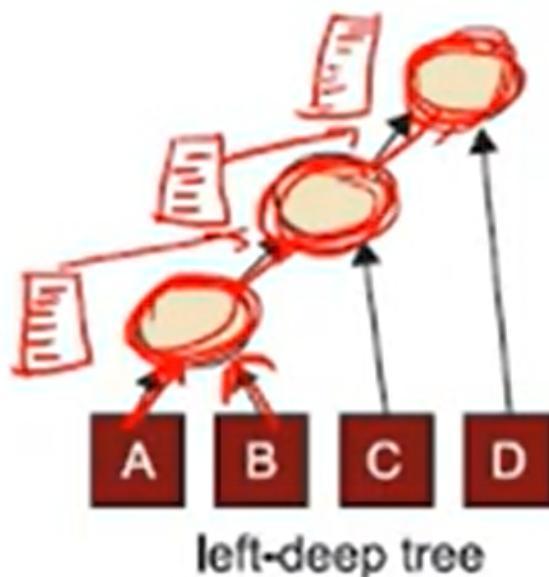
- * *Unario*: por ejemplo una selección. si hemos particionada la tabla o dataset anterior mente las diferentes particiones podrán hacer la selección por separado, normalmente no sale a cuenta particionar de forma dinámica durante la consulta si no se ha hecho antes.
- * *Binario*: si la operación es binaria (join) el coste puede llegar a justificar (no siempre) que invitamos tiempo particionando un dataset que no lo estaba, simplemente para que la consulta (join) vaya mas rápida.
- **Inter-operator**: depende de que arbol tengamos
 - * *Independent*: si teníamos un bushy tree podemos paralelizar operadores que sean independientes, que no tienen dependencias entre ellos.
 - * *Pipelined*: en el caso de tener un left-deep tree que parece que los operadores no se pueden paralela. También se puede parallelizar, simplemente cogemos un granularidad mas pequeño. Lo que parallelizamos no es la ejecución de todo el resultado intermedio (como en el bushy tree), en este caso se paralleliza fila a fila. Si reducimos lo que estamos parallelizando, no parallelizar dataset entero sino tupla a tupla aquí también se puede parallelizar.

Hay dos tipos de pipelining.

8.3.2 Demand-Driven Pipelining

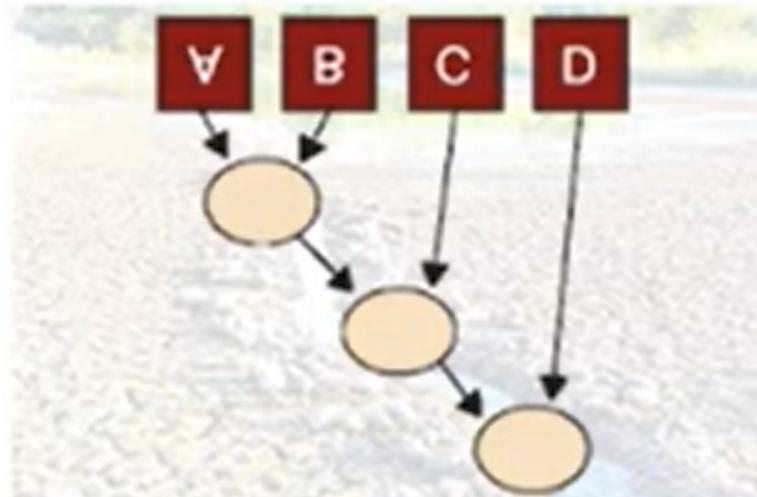
La idea es que aqui estiramos. Cada uno de los operadores funcionan como un tipo de cursor. Cuanto el usuario hace open del primer cursor, este operador hace open del segundo y este del tercero.

Esto no seria parallelizable a menos que se ponga un buffer (zona temporal, de memoria) a cada uno de ellos. Cuando el usuario abre el primer cursor, este crea un buffer y ya empieza a pedir filas. De manera que todos pueden ejecutar en paralelo las filas.



8.3.3 Producer-Driven Pipelining

Aquí empujamos. No tenemos estos iteradores sino que las tablas fluyen hacia los siguientes operadores. Se comporta de forma similar al anterior. Cuando el primer operador acaba de leer de las tablas, las filas van bajando por el resto de operadores.



El problema de hacerlo de esta manera es que en el momento que uno es mas lento que el resto, todo se colapsa. Si un operador puede estar ejecutando sus datos pero el anterior no le envía porque es mas lento, este tampoco le envía nada al siguiente. Y al revés, si un operador es muy rápido pero el siguiente no lo es tanto, los buffers se van llenando y van deteniendo la propagación.

Se pueden hacer de unas asunciones de cual es la ocupación del sistema y su tiempo de ejecución. La ocupación de sistema es el tiempo que tarda un sistema en tener un procesador libre de todos los que estaba utilizando hasta ahora (en los ejemplos anteriores seria el primer operador que lee los datos). El tiempo de ejecución es el tiempo de sistema que requiere para que acaben todos los operadores.

		<u>Occupancy</u>	<u>Execution time</u>	
<u>Serial</u>		T	T	
Inter-operator Parallelism	Independent	No stalls	T/N	$h \cdot T/N$
Pipelining	Pipelining	Stalls	$T/N+k$	$T/N+k+(h-1)(T/N)/ R $

T = time units required for the whole query
 N = operators in the process tree
 $|R|$ = tuples to be processed
 Execution time = time to process the query
 Occupancy = time until it can accept more work
 h = height of the process tree
 k = delay imposed by imbalance and communication overhead

En un entorno centralizado, el tiempo de ocupación y de ejecución es el mismo.

8.4 Measures of Parallelism

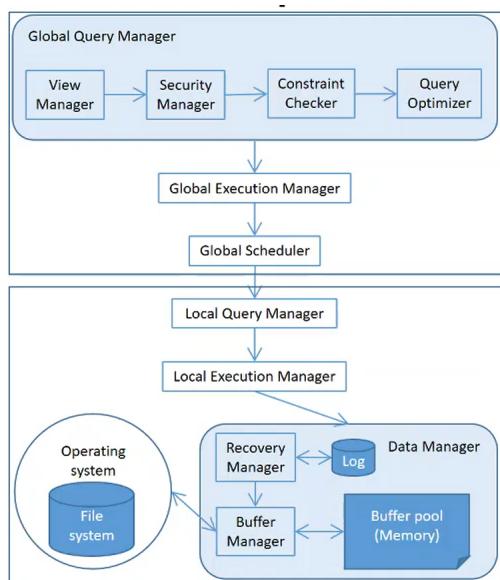
...

9 Distributed Processing Engine (MapReduce)

- Se necesita que escale a la magnitud de la red
- No se quiere preocupar del paralelismo. Que el paralelismo sea por así decirlo transparente a los usuarios.
- Todo lo que se pueda se debería ejecutar en local para evitar movimiento de los datos que siempre es costoso. La distribución de los datos debe ser transparente a los usuarios.
- Balancear la carga de computacion.
- Ser tolerante a los fallos, lo cual es muy probable con las miles de maquinas. En caso de que una se estrope, no se debería repetir todo el computo, sino el computo de esa maquina. **Fine grained fault tolerance**.

9.1 MapReduce as a DDBMS component

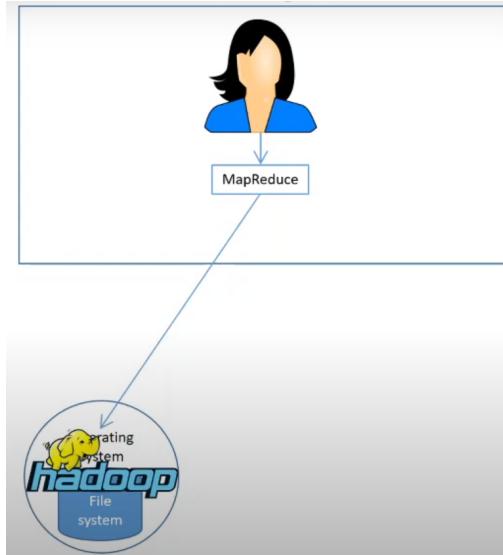
Que es MapReduce se preguntaran ustedes?



En el caso de análisis de gran cantidad de datos, no se necesita el planificador. Simplificación porque solo hay interferencias si alguien modifica los datos y cuando se hace análisis de datos básicamente no se modifica el dataset.

Segunda simplificación es no utilizar un query optimizer. La persona es suficientemente buena para hacer una consulta de forma correcta y su implementación para como acceder los datos. Pasamos del **que?** al **como?**. De declarativo a procedural.

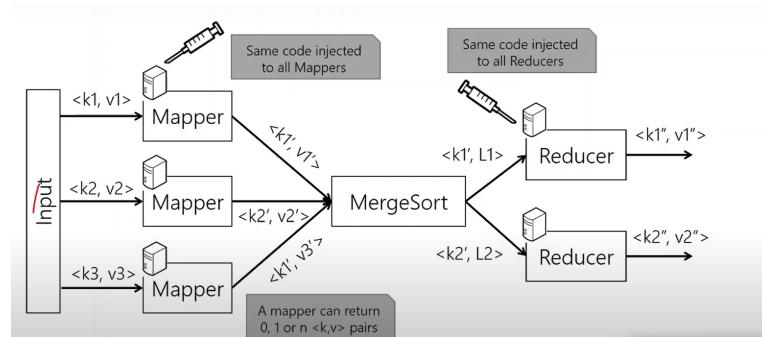
El MapReduce jugaría con el papel del global execution manager.



Toda la parte de optimización local también se simplifica y se traslada al DFS. La estructura es muy simple. Esto se hace tan sencillo porque lo que permite MapReduce también es muy sencillo.

9.2 Components and Use

La entrada son parejas (clave, valor) y que enviamos a una serie de maquinas. Después se inyecta una función a las maquinas (query-shipping) para enviar las transformaciones de los datos que han programado los desarrolladores. Estas funciones se ejecutarán de forma distribuida y paralela en todas las diferentes maquinas y darán como salida parejas (clave, valor).



Cada llamada a esta función de **map**, regresa de 0 a muchas parejas. Lo más importante es que la clave de entrada y de salida no tienen nada que ver (hasta con diferentes tipos de datos).

En la entrada del **MergeSort** ordena y hace un merge de todos los valores diferentes de las claves y los pone dentro de la misma lista. Se provee otra función que se envía a todos los workers para ejecutar la parte de **Reduce**. Ahora los parámetros de entrada de la función son las parejas (clave, lista) y en la salida da una clave diferente a la de entrada.

9.3 Relational algebra in MapReduce

Como se puede hacer una operación en un numero limitado de maquinas de manera paralela?

9.3.1 Projection

$$\pi_{a_1, \dots, a_n}(T) \Rightarrow \begin{cases} \text{map(key } k, \text{ value } v) \mapsto [(\text{prj}_{a_1, \dots, a_n}(k \oplus v), 1)] \\ \text{reduce(key } ik, \text{ vset } ivs) \mapsto [(ik)] \end{cases}$$

π representa la proyección (operación porque tiene una única relación T como parámetro) pero después tiene un segundo parámetro (a_i, \dots, a_i) que indica que atributos de la relación nos queremos quedar.

Dentro de la función **map** tenemos las parejas, que la key podría ser la **PK** y los valores el resto de atributos. Aquí se haría una operación que concatena los **PK** con los otros atributos para recuperar la tupla original y se implementa la operación de proyección de forma local para quedarnos los atributos que nos queremos quedar. La clave resultado del map sería la proyección y el valor una constante unitaria porque no nos interesa.

Reduce recibe la proyección y un conjunto de 1's como valor, que se ignora. La tupla se pasa a la salida.

9.3.2 Cross Product

De una forma similar se puede hacer el producto cartesiano.

$$T \times S \Rightarrow \begin{cases} \text{map(key } k, \text{ value } v) \mapsto \\ \quad \begin{cases} [(h_T(k) \bmod D, k \oplus v)] & \text{if } \text{input}(k \oplus v) = T, \\ [(0, k \oplus v), \dots, (D - 1, k \oplus v)] & \text{if } \text{input}(k \oplus v) = S. \end{cases} \\ \text{reduce(key } ik, \text{ vset } ivs) \mapsto \\ \quad \begin{cases} \text{crossproduct}(T_{ik}, S) | \\ \quad T_{ik} = \{iv \mid iv \in ivs \wedge \text{input}(iv) = T\}, \\ \quad S = \{iv \mid iv \in ivs \wedge \text{input}(iv) = S\} \end{cases} \end{cases}$$

Operación binaria al ser dos relaciones. En el **map**, la clave y valor de entrada son los mismos que para la operación anterior. Si la fila viene de una relación se hace una cosa, sino se hace otra en la función del **map**. Si viene de T se aplica una función de hash y luego se aplica de modulo para la clave y el valor es la tupla completa como en el caso de la operación anterior.

Si viene de la tabla S , emitimos como resultado del map D tuplas clave valor, las claves va de $0, \dots, D-1$, el valor es el mismo de la operación anterior para todas las parejas. D copias de la fila de la tabla S .

El intervalo $0, \dots, D - 1$ serán las claves como entrada del **Reduce**. Cada clave recibe un conjunto de filas que son las que vienen de T y de S .

El parámetro D determina el nivel de paralelismo y distribución.

10 Distributed Processing Engine (Spark)

10.1 Background

Primero tenemos que entender cuál es el problema que intentamos resolver. Por eso tenemos que ver cuáles son las limitaciones de **MapReduce**.

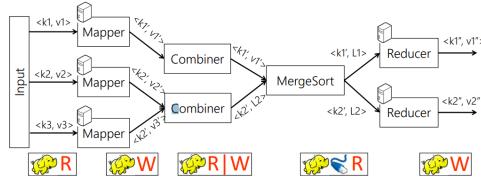


Fig. 10.1: Flujo de MapReduce

En la imagen vemos el flujo que sigue **MapReduce**. Primero tenemos las entradas, los mappers, reducers, el algoritmo de mergesort y los combiners que hay entre medio para reducir los datos y optimizar el acceso.

Lo importante es darse cuenta de que entre cualquiera de estas fases, leemos e escribimos datos HDFS. Estamos constantemente leyendo e escribiendo datos del sistema de ficheros.

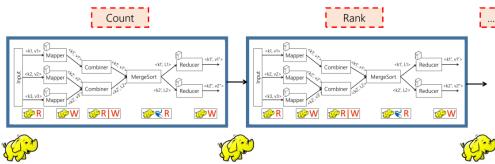
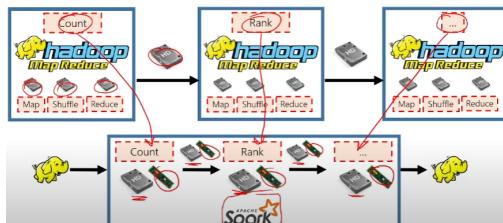


Fig. 10.2: Comunicación entre Fases de MapReduce

No solo esto, sino que si encadenamos dos o más operaciones de **MapReduce**, por ejemplo, para hacer un contador, otro para un ranking, y vamos haciendo, la manera de comunicar los datos entre las diferentes fases vuelve a ser mediante el sistema de ficheros. Esto no es la mejor manera de hacerlo. La mayor crítica que se le hacía a **MapReduce** es que constantemente escribía.

Lo que intentamos resolver con Spark es si tenemos un *count* (job de **MapReduce**), otro que hace un ranking y otro job de **MapReduce** que hace lo que sea, lo que hacemos con Spark es meter estas N transformaciones que tenemos de los datos dentro de la misma *caja* de Spark, dentro de un mismo **job de Spark**.



Entonces, cualquier comunicación que haya entre estas transformaciones o dentro de estas transformaciones puede aún pasar al disco, o que la comunicación vaya en el disco; esto aún es posible. Es un falso mito que Spark solo corre en memoria. Pero la diferencia es que ahora tenemos la opción también de que en la comunicación podemos hacerlo a través de la **memoria**, cosa que no se podría hacer con **MapReduce**, donde la única opción era en disco. Ahora tenemos ambas opciones (que nos proporciona el sistema) según lo que sea más eficiente dependiendo de la memoria que tengamos disponible.

10.2 Dataframes

La principal herramienta para trabajar con Spark son los **dataframes**. La alternativa para trabajar con datos serían tablas relacionales (SGBD) donde podemos usar SQL. El primer problema que esto presenta es que tenemos que definir claramente cuál es el esquema antes de hacer nada, y esto no siempre es evidente. Además, trabajar con datos semi-estructurados no siempre es posible, complica a la hora de hacer consultas.

Por otro lado, también complica mucho el *debug* de las consultas. O falla la consulta entera o funciona entera, pero no falla una parte pequeña de la consulta, cuesta ver lo que está pasando y lo que está fallando. Esto es normal ya que SQL fue concebido para OLTP, no para análisis de datos.

10.2.1 Characteristics of dataframes

Lo que se vio que funcionaba mejor eran los dataframes. Lo que permitía era el tratamiento totalmente simétrico de filas y columnas, no como en SGBD que tenemos columnas fijas y añadimos filas. De esta manera, no solo las columnas se pueden reverenciar por posición y nombre sino también las filas por posición y nombre.

No obstante, los datos aún se tenían que corresponder a un cierto esquema, pero este esquema se puede definir en tiempo de ejecución y no en tiempo de creación de la tabla como un SGBD. Esto es muy útil para cuando hay ciertas filas que no tienen los tipos de datos o precisión que corresponda.

Estos sistemas tenían 3 tipos de operaciones.

- **Relacionales:** filtro, join
- **Hoja de cálculo**
- **Álgebra lineal:** multiplicar matrices, etc.

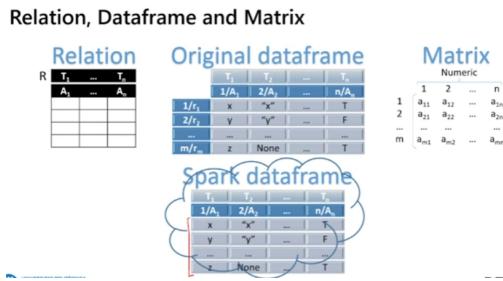
Lo que daba todo esto son operaciones más pequeñas y que se pueden componer e crear cadenas o *pipes* (tuberías) de las filas o datos que van sufriendo diferentes transformaciones una detrás de la otra. Esto encaja bastante con lenguajes imperativos como Java o Python.

10.2.2 Relation, DataFrame, and Matrix

Un dataframe se puede ver como un intermedio entre una relación y una matriz. En las relaciones, el problema que tenemos es que el tipo de datos y atributos están completamente fijados y las filas no tienen identificador, mientras que en las matrices podemos añadir columnas y filas indistintamente.

Entonces, un dataframe sería como un entremedio de las dos. Por un lado, podemos hacer operaciones relacionales, pero por otro, podemos hacer lo mismo que con las matrices. Los tipos de datos no están fijados en tiempos de definición (se pueden cambiar en tiempos de ejecución), y tanto las filas como columnas se pueden referenciar por nombre o posición. Esta era la idea inicial, como por ejemplo funcionan en Python.

Pero con Spark, tenemos casi lo mismo, pero no exactamente. La principal diferencia es que las columnas sí que tienen identificadores, pero las filas no. Lo que tenemos es un conjunto sin orden y, por lo tanto, no podemos tener las filas por orden. Lo más importante es que esto reside en la nube. En general, un dataframe de Spark está distribuido (o puede estarlo) en tantas máquinas como queramos, y podemos usar el paralelismo en nuestras consultas y transformaciones. El otro dataframe reside en memoria y, por lo tanto, solo se puede trabajar en la máquina en la que estamos.



10.2.3 Spark Dataframe definition

Un **Dataset** es una colección de objetos del dominio que se pueden transformar en paralelo utilizando tanto operaciones funcionales como relacionales.

Un **DataFrame** es una colección de datos inmutable organizada en columnas con nombre y que se pueden distribuir en diferentes nodos de un clúster (cloud), y que cada una de estas filas del dataset corresponde a un objeto que es del tipo fila.

Es importante la característica de **inmutable**, un dataframe no se puede cambiar. Lo que quiere decir es que cuando le aplicamos una transformación, lo que obtenemos es un nuevo dataframe; el anterior se queda como estaba y obtenemos uno nuevo. La transformación no se aplica sobre el dataframe donde aplicamos la transformación, sino que se crea uno nuevo.

Además, se puede particionar y distribuir, por eso usa una abstracción de más bajo nivel (RDD) *Resilient distributed datasets*, que es lo que hay por debajo de los dataframes. En la mayoría de casos, no hace falta acceder a los RDD's aunque se puede hacer.

10.2.4 DataFrame vs Matrix/Array/Tensor

Dataframe	Matrix
Heterogeneously typed	Homogeneously typed
Both numeric and non-numeric types	Only numeric types
Explicit column names (also row in Pandas)	No names at all
Supports relational algebra	Does not support relational algebra

10.2.5 DataFrame vs Relation/Table

El dataframe de Spark esta digamos entremedio de los dos extremos que son los dataframes como los de pandas y las tablas relacionales. En las relaciones, los esquemas son prefijos y en los dataframes se puede agregar el esquema a posteriori.

Pandas Dataframe	Spark Dataframe	Relation
Ordered	Unordered	
Named rows	Unnamed rows	
Lazily-induced schema		Rigid schema
Column-row symmetry	Columns and rows are different	
Support linear algebra	Does not support linear algebra	

10.2.6 Dataframe implementations

Pandas	Spark
Eager evaluation	Lazy evaluation
Resides in memory	Requires a SparkSession
Not scalable (multithread operators exist, but manual split is required)	Transparently scalable in the Cloud
Transposable	Non-transposable (problems with too many rows)

Los dataframes de pandas se evalúan en el momento que hacemos cualquier transformación, mientras que en el caso de Spark se declaran las transformaciones que aplicamos a un dataframe. Estas transformaciones se acumulan en memoria y solo cuando llamamos a otro tipo de transformaciones que se llaman **acciones** se ejecutan, no antes. Solo cuando se hace la acción esto dispara todo lo declarado antes.

Los de pandas están en memoria, Spark en la nube (pueden estar también en una sola máquina), pero es importante usar una sesión de Spark que es donde están los dataframes y esta ya se configura si queremos en local, en un clúster o en la nube.

Los dataframes de pandas no escalan ya que están en memoria, cabe lo que cabe en memoria. En el caso de Spark, todo esto es transparente; simplemente declararemos si la sesión se ejecuta local o en la nube, y esto automáticamente se distribuye como se crea más oportuno según el optimizador de consultas que tengamos.

Los dataframes de pandas pueden transponerse porque filas y columnas se tratan simétricamente, mientras que las de Spark no es posible. El numero de columnas no escala tan bien como el numero de filas. Su comportamiento no es simétrico.

10.2.7 Operations

Los dataframes de Spark tienen diferentes tipos de operaciones.

- **Entrada/salida:** obtener datos e convertirlos en un dataframe de diferentes fuentes y se puede generar una salida de CSV, JSON.
- **Transformaciones:** no modifican el dataframe; lo que hacen es generar uno nuevo con los mismos datos que el anterior pero con la transformación determinada.
 - **RDDs:** abstracción por debajo.
 - **SQL:** abstracción más alta.
- **Acciones:** lo que hacen es sacar los datos del cloud. Las acciones cogen un dataframe y generan una estructura de datos en memoria que ya no es un dataframe. Es cuando sacamos los datos en la nube y se ejecutan en cadena las transformaciones.
- **Schema management:** operaciones que permiten trabajar con el esquema de dataframe, ver qué columnas tiene, tipos, etc.

Entrada/Salida

Se puede crear un dataframa a partir de ... o se puede transformar un dataframe a un ...

- Matrix
- Pandas dataframe
- CSV
- JSON
- RDBMS
- HDFS file formats: (ORC, Parquet)

Transformaciones

Cualquiera de estas transformaciones tiene un dataframe como entrada y uno de salida.

El dataframa de salida es siempre diferente del de entrada.

- select
- filter/where
- sample
- distinct/dropDuplicates
- sort
- replace
- groupBy+agg
- union/unionAll/unionByName
- subtract
- join
- ...

Acciones

Reciben como entrada un dataset y resuelven un resultado que se almacenan en memoria.

Materializan el resultado dl cloud.

- count
- first
- collect
- take/head/tail (*cuidado con take porque se transfieren todas las filas del Cloud y podrían no caber en memoria.*)
- show
- write
- toPandas
- ...

Schema Operations

- summary/describe
- printSchema
- columns

10.2.8 Optimizaciones

Lazy evaluation

Una transformación no hace nada; es una declaración de lo que quieras hacer. No pasa nada hasta que usamos una acción que dispara todo esto. Nos apuntamos todas las operaciones y buscamos de optimizarlas cuando se llama a una acción. No se puede optimizar nada si no se tiene una secuencia de transformaciones.

- **Cache/Persist:** Si hay un dataframe que se usa dos veces, las transformaciones que llevan a ese dataframe se ejecutarán 2 veces, lo que es ineficiente. Tenemos la

posibilidad de guardar el resultado de la primera ejecución en caché (en memoria) para que la segunda vez que se vuelva a disparar una acción que involucre ese mismo dataframe no se vuelva a ejecutar absolutamente todo. Consume memoria; si lo haces muchas veces, te quedas sin. Hay que ir desactivando este mecanismo. Si lo usamos 2 veces y no 3, después de la segunda podemos usar *unpersist*.

- Una alternativa muy buena son los **checkpoints**. Son parecidos al **cache/persist**, pero la diferencia es que en este caso se borra toda la historia previa del dataframe, se ejecuta y guarda el resultado en el disco, y esto seguro que no se vuelve a ejecutar nunca más. Si volvemos a llegar a ese dataframe, se cogerá del disco y no se ejecutará nada. El checkpoint es más caro, gasta más recursos, pero garantiza que no ejecuta nada y por otra parte nos libramos de todo el árbol de procesos.

Paralelismo

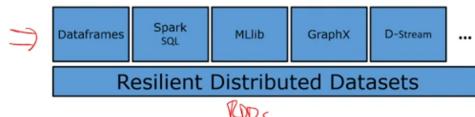
Está basado en las máquinas/cores disponibles. Hay que tener unos cuantos para que trabajen en paralelo. Pero además, hace falta que el dataframe esté particionado para que se ajusten a los recursos que se tengan disponibles. Por defecto se crean particiones, pero mientras las transformaciones van aumentando con joins podemos aumentar el número de datos o al filtrar quitar datos. Y es mejor ir reparticionando para ajustar el número de datos a los cores que queremos utilizar, que se ajuste a los recursos que tenemos disponibles. Si tenemos muchas particiones pero pocos cores, seremos muy inefficientes porque gastamos más en gestionar las particiones, pero si no particionamos el dataframe, por mucho que tengamos 10000 máquinas disponibles, no se paraleizará nada.

Es interesante ajustar el numero de particiones, sobre todo cuando cambia el volumen de datos. Para ello tenemos dos opciones.

- Tenemos las operaciones de **repartition**, que crea el número de particiones que queramos y con una función de *hash* redistribuye todos los datos a través de la red (puede llegar a ser muy costoso) aunque eres libre a determinar el numero de particiones.
- La alternativa es **coalesce** que solo permite reducir (no aumentar) el número de particiones pero se quedan los datos en la misma máquina donde estaban.

10.3 Abstractions

Spark es un sistema extensible y, por lo tanto, permite las abstracciones por encima. El concepto básico son los RDD's. Encima de estos RDD's se definen diferentes tipos de abstracciones; las más básicas serían los dataframes, pero no son los únicos. También existen SQL, grafos, librerías de machine learning...



10.3.1 Spark SQL

Además de tener una interfaz declarativa (API) de un lenguaje **orientado a objetos** como Python, podemos tener una declaración no procedural. Y lo que se trataría es que

esto no solo funciona sobre bases de datos relacionales, sino que se extiende a cualquier fuente de datos. Cargamos nuestros datos en Spark, ya sea JSON, CSV, etc., y podemos hacer consultas SQL sobre ellos.

Esto, obviamente, se tiene que traducir a un sistema procedural no declarativo, y por eso se coge la aproximación de SGBD que es tener un optimizador de consultas. Primero hay una **fase basada en reglas** que hace los predicados lo antes posible (filtrado, selección) y la reducción de columnas también tan temprano como sea posible. Una vez hecho esto, trabaja en un **sistema basado en costes** donde intenta estimar el coste de cada ejecución y se queda con el coste más pequeño. La tercera característica es que este sistema basado en costes es **extensible**, es decir, como se trata de código abierto (Spark), cualquiera puede implementar nuevas operaciones o nuevos mecanismos de implementación de las consultas.

10.3.2 Spark SQL interface

Cada sesión de Spark tiene un catálogo en el cual podemos registrar dataframes como si fueran tablas. Cogemos un dataframe y le damos un nombre de tabla; entonces, este nombre se guarda automáticamente en el catálogo.

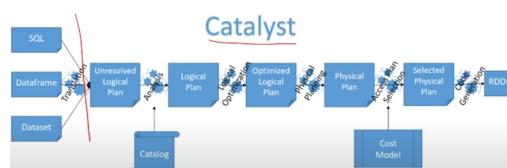
A partir de aquí, utilizando nuestra sesión de Spark, la cual está ligada a nuestro catálogo, podemos pasar cualquier consulta SQL que mediante el catálogo se traducirá la consulta.

La entrada es una cadena de caracteres de la consulta (el string de la consulta), y el output es un dataframe.

- There is a catalog with all tables available
`SparkSession.catalog`
- Dataframes are registered as views in the catalog
`DataFrame.createOrReplaceTempView(<tablename>)`
- Queries:
`SparkSession.sql(<query>)`
 - Input is simply a string
 - Output is a dataframe

10.3.3 Shared Optimization and Execution

Todo esto lo hace un optimizador de consultas de software abierto que se llama **Catalyst**. La entrada puede variar entre distintos tipos de datos, ya sea SQL, Dataframe, Dataset, etc.



El Catalyst genera un plan de ejecución lógico (comprueba la sintaxis de la consulta) a partir del catálogo e tablas que hayamos definido, lo que sería equivalente al árbol lógico que teníamos en sistemas relationales. Hace la optimización del árbol sintáctico.

También generará el plan de acceso físico, las operaciones concretas que se tienen que ejecutar. Aquí entra la parte del modelo de costes, generará diferentes alternativas y estimará el coste de cada una de ellas, escogiendo el mejor de ellas.

A partir de este plan, lo que acaba generando son RDD's, que es lo que hay por debajo de Spark.

11 Big Data Architectures and Model Management

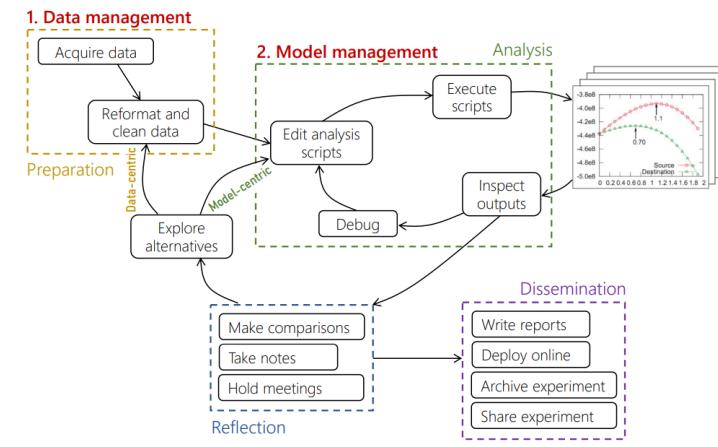
Data science es un proceso colectivo de teorías/procesos que nos permiten analizar/extrair con lo que tiene que ver con entender mejor los datos.

Descriptive analysis: guardamos datos que son realmente importantes. Son típicamente objetos de negocio (clientes, ventas, compras).

- Actividades comunes son las ETL de múltiples fuentes de datos. Lo importante es el esquema y se define un modelo fundamental y se nos permitía hacer las operaciones y optimizarlas.

Predictive analysis: Aprovechamos los datos importantes pero también una parte serían lo que se llaman *not obviously* importante (no esta del todo claro si son relevantes para el análisis). P.e. los clicks del usuario, su comportamiento de búsqueda, etc.

- Las actividades son comúnmente usar técnicas de machine learning para entrenar modelos predictivos.
- El proceso es diferente del descriptive analysis.

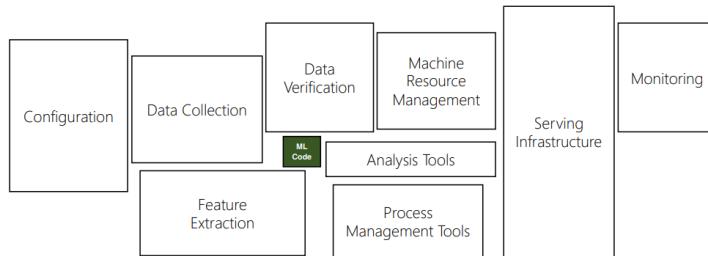


1. Data Management: Recopilación de datos y su limpieza (formateo definido a lo largo del proyecto).

2. Model Management: Una vez que tenemos los datos limpios y preparados empezamos con el análisis. Podemos entrenar un modelo, seguir limpiando los datos, imputar los valores... Utilizamos estadística i gráficos para entender mejor los datos i la relación que tienen con nuestro análisis.

Este es un proceso interactivo, pero una vez un algoritmo esta decente, lo que se tiene que hacer es reflexionar un poco para definir si seguimos explorando alternativas o ya es suficiente. Tenemos dos opciones, ir a mejorar los algoritmos o hacer un data-centric (volver a los datos y redefinir que necesitamos para mejorar el análisis).

Por ultimo hacemos **dissmination**, desplegar esto en producción ya que ahora tenemos un modelo. Lo importante es tener los mecanismos para reproducir todo esto y desplegarlo en producción.



Lo que es importante es que tenemos un reto grande; **Data Management** y por el otro lado **Model Management**. Si se hace sin sistematizar el proceso pueden surgir desafíos.

Challenges

- Como asignamos el nombre de los ficheros.
- Como guardmos los ficheros.
- Como seguimos la evolución de los ficheros.
- Como sabemos si los datos están actualizados.
- Como se guardan los datos si son muy grandes. Y como lo hacemos de manera eficiente?
- Como arreglamos errores, datos faltantes, errores inconsistentes,...
- Como integramos los datos

11.1 Data Management Backbone

La idea de como se pasa de los DatWarehouses a los DataLakes.

11.1.1 Data Managment

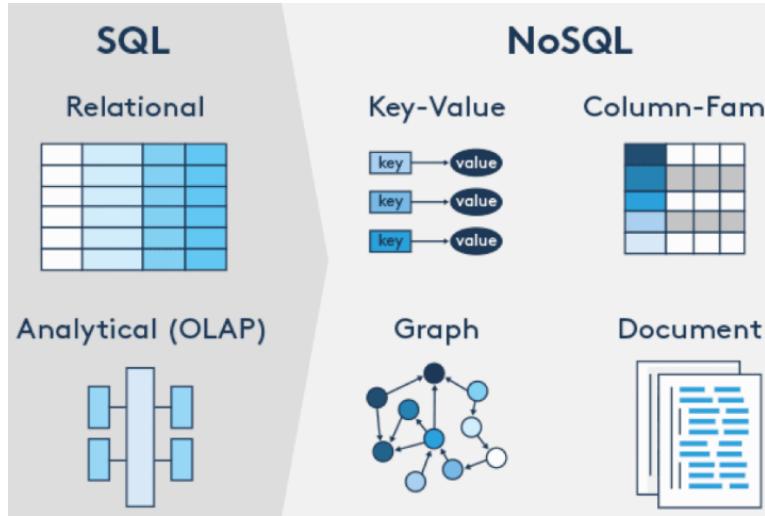
Se necesita tener una serie de sistemas que deben proporcionar las DBMS:

- **Ingestion:** proveer los recursos necesarios para poder hacer inserts/updates de los datos.
- **Storage:** estructuras que preserven los datos.
- **Modelling:** tiene que ver con que frecuencia vamos a acceder a los datos.
- **Processing:** como procesamos de manera eficiente, si es necesario en paralelo.
- **Querying:** consultas.

En un entorno **Big Data** son los mismos conceptos pero suponiendo no SQL, no podemos suponer que todo se puede resolver con sistemas de bases de datos relacionales. Normalmente debería ser un sistema distribuido con posibles diferentes modelos de datos, nos debería permitir modelar los datos en diferentes modos.

Implementamos arquitecturas ad-hoc. Dependiendo del usuario una arquitectura podría ser mejor que otra. En el modelo relacional siempre se debe definir un esquema, es rígido y no flexible. Si tenemos varios casos de recursos que requiere flexibilidad necesitamos algún modelo con *schemaless*. Podemos directamente insertar los datos y se permiten las inconsistencias en los datos.

En el caso de *key-value* cada valor se identifica con una *key* que se guardan físicamente. Estos tipos de modelos permiten distribuir los datos de manera más fácil ya que toda la información que tiene que ver con un usuario está guardado en un mismo lugar. Además no hay dependencia entre las tablas, como en el caso relacional. Es una unidad



muy sencilla que se puede distribuir, podemos hacer **partición horizontal**. Se pueden procesar de manera independiente y paralela.

11.1.2 1st Generation: Data Warehouses

Queremos hacer *descriptive analysis* y tenemos que hacer un ETL y definir el schema. La idea es que no es fácil guardar información que es obviamente importante como imágenes, vídeos, ...

Model first: Creamos un esquema fijo que nos deja hacer cierto análisis sobre los datos y después convertimos los datos hacia este esquema (**modelamos en tiempo de carga**). El problema es que estas transformaciones son casi permanentes. No se puede cambiar la semántica de la transformación porque sino habría inconsistencia en los datos.

También esta el problema que el schema es fijo, si queremos añadir un análisis nuevo habría conflicto con las tablas relacionales que se tienen. Los que hacen análisis sobre estos datos tienen que tener conocimiento del schema.

Es costosos ya que los DW se conservan en servidores, debemos preparar una arquitectura para un máximo uso, no hay elasticidad.

11.1.3 2nd generation: Data Lakes

Poder insertar datos sin tener que definir unos esquemas rígidos. Tenemos diferentes tipos de datos que se pueden insertar en el Data Lake, podemos poner lo que queremos como vídeos, fotos, etc. Hay low-cost storage porque es fácil construir diferentes ordenadores en un cluster para guardar esta información, no hace falta tener un superordenador como en el caso de DW, podemos tener nodos sencillos.

A la hora de hacer análisis es *on-demand*, dependiendo del análisis tenemos que leer los datos por nuestra cuenta. No hay un sistema que devuelva los datos de manera automática. Esto se inició todo con *Apache-Hadoop i Google File System*.

Load first - Model later: A la hora de hacer el análisis, el analista es quien define como analizar los datos, no hay esquema. Se puede hacer tanto machine learning como descriptive analysis (se puede crear un Data Warehouse sobre el Lake).

Esto también lleva a ciertas complicaciones y posibles soluciones:

- Si ponemos todos los datos sin ninguna estructura ni criterio lo que podemos tener es un Data Swamp, si no podemos orden tendremos un sistema como un Desktop de cada uno. No hay una manera de distinguir los datos.
- Aunque las transformaciones para hacer el análisis son mas complejas. Normalmente no se reutiliza por los diferentes analistas. Mucha redundancia.
- A la hora de introducir los datos necesitamos metadatos que nos permitan entender un poco que hay en los datos. En lugar de acceder solamente por el nombre de fichero saber también los atributos/esquema de los datos que contiene.
- Desarrollar mecanismos para gestión y procesamiento de los datos de manera automática.
- **Formatted Zone:** pasar los datos a formato relacional
- **Trusted Zone:** pasar los datos por un criterio de calidad del dominio.
- **Explottation Zone:** Zona final que nos crea las relaciones/dataframe/etc. que se utilizan para el análisis. Nada nos evita de crear un schema fijo.

Tenemos una heterogenidad de diferentes formatos de datos:

11.1.4 Heterogeneity of data formats

- **General-Purpose Formats:** csv, json, cli/api
- **Sparse Matrix Formats:** para cuando hay mucha falta de datos.
- **Large-Scale:** cuando queremos almacenar *Big Data*, muchos datos.
- **Domain-Specific Formats:** específicos para dominios en concreto.

Paquet: no solo se pueden utilizar en la Landing Zone, sino que se pueden trasladar al resto de zonas dependiendo de su implementación. Con **Paquete** se reduce el espacio de memoria de los ficheros al tener métodos de optimización y tener un almacenamiento por columnas.

Lo que hace es un horizontal partition pero luego se definen row groups pero para cada una de las particiones estas se guardan por columnas. Tenemos un header y un footer, Nos permite conocer los tipos de datos que tenemos y luego también tenemos algunas estadísticas como min, max, avg.

11.1.5 3rd gen: Cloud Data Lakes

Utilizar el cloud. Ahora en vez de hacer toda la gestión de la infraestructura que cuesta bastante, lo podemos hacer con menos coste en un cloud. Nos proporciona casi las mismas características que un Lake normal pero son mas disponibles porque tenemos la infraestructura que funciona "mejor".

Según los usuarios que acceden al cloud, se tienen Data Farms en diferentes regiones del mundo y es mas óptimo a la hora de acceder a los datos. Otra ventaja es que pagas por lo que usas. El coste es determinado por el tiempo de uso de la herramienta. No se malgastan recursos como en el caso de los Warehouses.

Drawbacks

- Inconsistencias entre data lake y data warehouse. Mantenerlo constante es difícil y costoso.
- No hay un soporte para frameworks complejos de análisis de datos avanzado.
- Coste menor por la elasticidad, pero dado que tenemos 2 layer arquitectura por el data lake y DW podemos tener redundancia de los datos.

11.1.6 4th gen: Lakehouses

Un Lakehouse, combinar de alguna manera el Data Warehouse y el Data Lake. Se necesitan muchos metadatos. Se usa parquet pero muy mejorada. Se proporciona alguna característica de los DW y todas las de Data Lakes.

11.2 Data Analysis Backbone

11.3 ML Lifecycle Management

11.3.1 AutoML Overview

Porque se necesita un almacenaje de los modelos? Para optimizar se necesitan probar varios parámetros y después seleccionar seleccionar el modelo con los mejores resultados.

1. Model Selection: seleccionar el mejor modelo
2. Hyper Parameter Tuning
 - Validation: Generalization Error: cross validation, train val split.
 - AutoML Systems/Services:

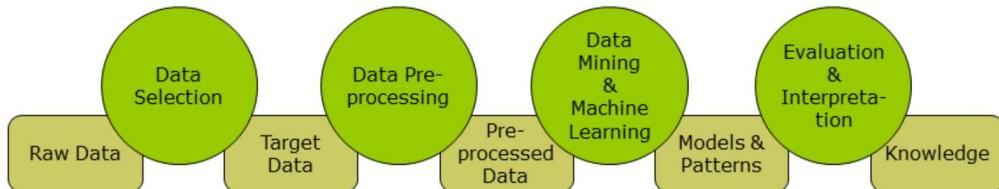
Se necesitan guardar los modelos para que puedan ser posteriormente evaluados y seleccionados. Hay varios sistemas que proporcionan unos métodos de búsqueda y selección optimizada de los diferentes modelos. **AutoML**.

Basic Grid Search: Dado una serie de hiper parámetros y sus dominios de valores. Evaluar todo el espacio de hyperparametros de manera exhaustiva. Solo puede ser aplicado para dominios pequeños. Podemos discretizar los valores continuos de manera uniforme. En algunos casos es mejor usar un crecimiento exponencial para definir el espacio de búsqueda.

Monte-Carlo: Para mitigar el problema exhaustivo, podemos aplicar una búsqueda random a través de la grid.

Appendix A Theoretical questions

1. Give examples of metadata that can be used at every step of the Knowledge Discovery in Databases process.



- **Selección de Datos:**
 - **Datos de Validez:** Información sobre la validez de los datos.
 - **Descripción de la Fuente de Datos:** Información sobre la fuente de los datos, incluyendo su origen, formato y método de acceso.
 - **Marca de Tiempo de la Recopilación de Datos:** Cuándo se recopilaron o extrajeron los datos.
 - **Método de Muestreo de Datos:** Detalles sobre cómo se muestrearon los datos, si es aplicable.
 - **Evaluación de la Calidad de Datos:** Metadatos que indican la calidad de los datos, incluyendo evaluaciones de completitud, precisión y confiabilidad.
 - **Preprocesamiento de Datos:**
 - **Reglas de Negocio**
 - **Registros de Limpieza de Datos:** Registros de las operaciones de limpieza de datos realizadas, como el manejo de valores faltantes y valores atípicos.
 - **Descripciones de Transformación de Datos:** Información sobre cómo se transformaron, normalizaron o codificaron los datos.
 - **Notas sobre Ingeniería de Características:** Documentación de decisiones de creación o selección de características.
 - **Metadatos sobre Reducción de Datos:** Si se aplicaron técnicas de reducción de dimensionalidad, metadatos sobre el método de reducción y los parámetros utilizados.
 - **Minería de Datos y Aprendizaje Automático:**
 - **Parámetros de Entrenamiento del Modelo:** Detalles sobre los parámetros utilizados para entrenar modelos de aprendizaje automático.
 - **Métricas de Rendimiento del Modelo:** Métricas utilizadas para evaluar el rendimiento del modelo (por ejemplo, precisión, puntuación F1).
 - **Versionamiento del Modelo:** Seguimiento de diferentes versiones de modelos junto con sus hiperparámetros.
 - **Puntuaciones de Importancia de Características:** Información sobre qué características tuvieron un mayor impacto en las predicciones del modelo.
 - **Evaluación e Interpretación:**
 - **Criterios de Evaluación:** Explicación de los criterios utilizados para evaluar los resultados de la minería de datos o el aprendizaje automático.
 - **Notas de Interpretación:** Documentación de percepciones e interpretaciones obtenidas del análisis de datos.
 - **Metadatos de Implementación del Modelo:** Información sobre cómo y dónde se implementa el modelo para uso práctico.
 - **Información sobre el Ciclo de Retroalimentación:** Si corresponde, metadatos sobre cómo los resultados influyen en la toma de decisiones y la futura recopilación de datos.

2. Enumerate the advantages of having separate files (e.g., CSV) to be analyzed and compare it against the advantages of having a DBMS.

Advantages of using a DBMS	Advantages of using separate files
Estructurar los datos (índices) Integrated ² (centralizado) information SQL-able ³ Escalabilidad Propiedades ACID Uso de ML en el propio DBMS Gestión de permisos (GRANT, REVOKE)	Poca cantidad de datos Evita sobrecostes en casos concretos (ficheros temporales)

3. Explain in which sense each of the four characteristics of a DW in W. Inmon's definition increases the amount of data in the company.

- 3.1 **Subject-oriented:** puede haber duplicación de datos al usar las mismas tablas en distintos temas.
- 3.2 **Integrated:** Porque integramos datos de diferentes fuentes, no solo de los datos operacionales que tenemos sino también de *webs*, etc. y por lo tanto el conjunto de datos es mucho más grande.
- 3.3 **Time-variant:** Mantener datos a través del tiempo multiplica la cantidad de datos, uso de datos históricos que permitan ver la evolución de estos.
- 3.4 **Non-volatile:** Porque no borramos nada, lo único que hacemos es añadir. Si queremos modificar alguna cosa lo que hacemos es añadir una nueva versión del dato.

4. Explain the difference between "Time-variant" and "Nonvolatile" characteristics in the DW definition of W. Inmon. Illustrate it with an example different from that in the materials of the course.

Variante en el timepo hace referencia a que contiene datos históricos (por ejemplo datos de una empresa de hace 5 años) para poder hacer un buen análisis. **Non-volatile** significa que la información no se puede borrar ni modificar, lo único que se puede hacer es inserir nuevos datos.

Appendix B OLAP and the Multidimensional Model

B.1 Theoretical questions

1. Briefly explain the difference between the different kinds of multidimensional schemas:

a) Star vs. Snowflake

El esquema estrella tiene la tabla de hechos y las tablas de dimensiones alrededor con relaciones de muchos a uno. A diferencia del esquema snowflake, en el estrella no aparecen las jerarquías de agregación de las dimensiones. Por lo que dan más información al usuario.

b) Star vs. Galaxy

El esquema de galaxia son un conjunto de esquemas de estrellas, por lo que hay más de una tabla de hechos, que comparten algunas dimensiones. Están interconectadas.

2. Would you implement a snowflake schema in a Relational DBMS (RDBMS)?

No es buena idea porque haríamos muchos *joins*, que es la operación más costosa de las bases de datos relacionales. En este caso la alternativa que tendríamos es hacer una única tabla que englobe los atributos de todas las jerarquías de agregación de esa dimensión. No hacemos el snowflake (conceptualmente sí), no hace falta que cada concepto esté en una tabla diferente.

También tiene problemas, ya que repetiríamos muchos valores. Que preferimos, repetir los datos o separar las tablas y hacer muchos joins? La decisión depende de condicionantes que tengamos. Si tenemos que cambiar o modificar muchos los valores (sistemas operacionales) esto no es muy buena idea. Si no sucede esto es mejor la primera opción, aunque el volumen de datos sea más grande haremos menos *joins* y las consultas serán más rápidas.

Llevando esto al extremo, una mala idea sería poner todo en la tabla de hechos. Ya que si tenemos 4 dimensiones de 10 filas cada una tendríamos potencialmente 10000 filas en la tabla de hechos.

3. Consider a multidimensional schema with a dimension that does not have any attribute or aggregation hierarchy (e.g., order number). Briefly justify whether you would implement it like:

(a) A dimension table with only the primary key, pointed from the fact table.

(b) Only an attribute in the fact table, but not pointing to any dimensional table.

Es la mejor opción ya así no creamos una tabla de dimensión innecesaria con un solo atributo y hacemos menos *joins*. Esto es así porque los usuarios no modifican nada. Si estamos en un entorno OLT donde el usuario puede modificar la tabla, tener una dimensión extra nos permitiría gestionar mejor.

(c) No corresponding attribute or table in the star schema.

(d) Other option

4. One of the three necessary summarizability conditions is disjointness.

How can you implement a dimension with a level whose elements are overlapping avoiding summarizability problems? Give a concrete example using a UML class diagram with only two levels (you do not need to draw the whole snowflake schema).

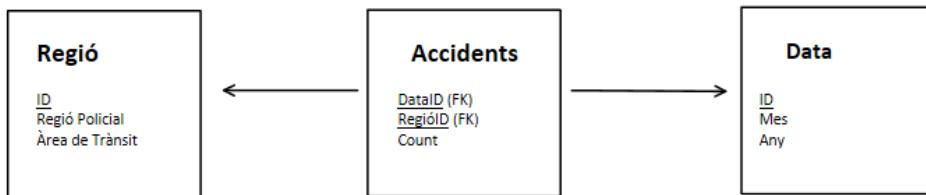
Esto solo sucede con relaciones muchos a muchos. Tenemos que crear una tabla intermedia de pesos con la cual tendremos que hacer *join* también.

B.2 Problems

- Identify factual and dimensional information in the following CSV sample file, and create a conceptual multidimensional schema.

ATTESTATION OF TRAFFIC ACCIDENTS

Month	Month name	Year	Police Region (RP)	Transit Regional Area (ART)	Number
1	Gener	2011	RP PIRINEU OCCIDENTAL	ART PIRINEU OCCIDENTAL	3
1	Gener	2011	RP GIRONA	ART GIRONA	10
1	Gener	2011	RP PONENT	ART PONENT	2
1	Gener	2011	RP CENTRAL	ART CENTRAL	7
1	Gener	2011	RP METROPOLITANA NORD	ART METROPOLITANA NORD	20
1	Gener	2011	RP METROPOLITANA SUD	ART METROPOLITANA SUD	10
1	Gener	2011	RP TERRES DE L'EBRE	ART TERRES DE L'EBRE	4
1	Gener	2011	RP CAMP DE TARRAGONA	ART TARRAGONA	5
2	Febrer	2011	RP PIRINEU OCCIDENTAL	ART PIRINEU OCCIDENTAL	3
2	Febrer	2011	RP GIRONA	ART GIRONA	6
2	Febrer	2011	RP PONENT	ART PONENT	4
2	Febrer	2011	RP CENTRAL	ART CENTRAL	1



- Identify factual and dimensional information in the following CSV sample file, and create a conceptual multidimensional schema.

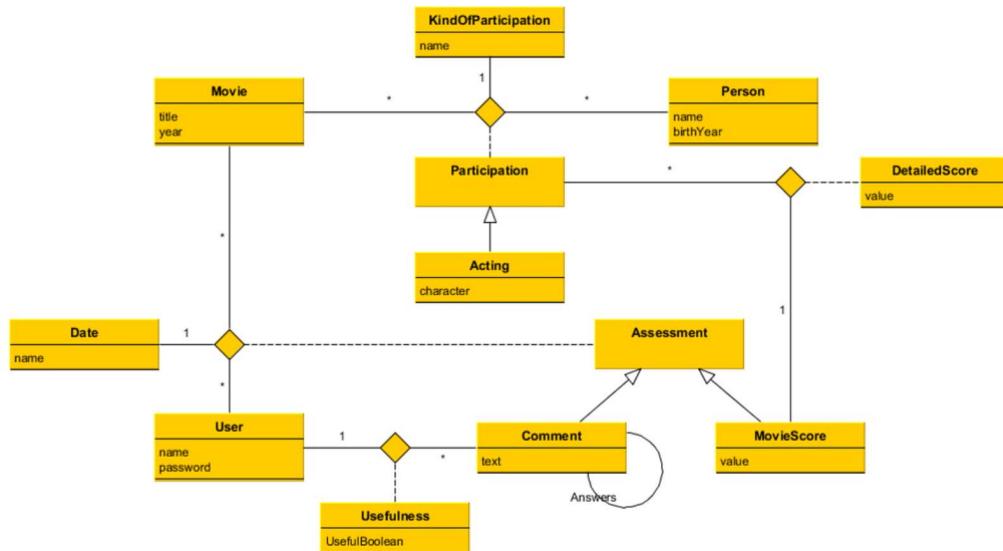
CITIZEN SECURITY

DISTRICTS	RELATED TO PEOPLE	RELATED TO PROPERTY	WEAPON POSESSION	DRUG POSSESSION	DRUG CONSUMPTION
CENTRO	48	127	10	190	82
ARGANZUELA	61	45	3	10	6
RETIRO	0	19	4	12	0
SALAMANCA	18	58	0	7	0
CHAMARTÍN	12	29	0	16	9

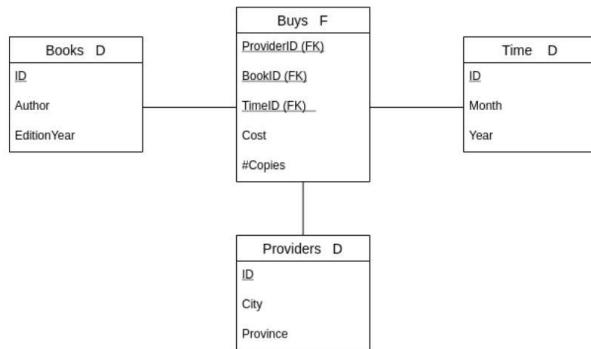
Una solución posible puede ser la tabla de hechos **Detención** con el número de identificador del delito y las tablas de **Distrito** y **Tipo** como dimensiones. Eso sí, si se añade un nuevo tipo de delito, se tiene que añadir a la tabla de hechos.

Sino podría ser la tabla de hechos **Detención** con todos los tipos de delitos como variables booleanas y únicamente la tabla **Distrito** como dimensión.

3. Identify some multidimensional schema (i.e., fact subject of analysis and its corresponding analysis dimensions) in the next UML class diagram.



5. Given the multidimensional schema, the sequence of multidimensional operations below, and the equivalent SQL query, justify if they correspond to each other. If they don't, briefly explain why and how the SQL query should be fixed.



- A:=Roll-up(Buys, Providers.city, Sum)
- B:=Roll-up(A, Books.AllBooks, Sum)
- C := Roll-up(B, Time.AllDays, Sum)
- D := ChangeBase (C, { Providers })
- E:=Selection(D, city="Barcelona")
- F := Projection(E, #copies)
- R := Drill-down(F, Providers.ID)

```

SELECT bo.ID, t.ID, p.City, SUM(bu.cost)
FROM Books bo, Buys bu, Providers p, Time t
WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
GROUP BY bo.ID, t.ID, p.City
ORDER BY bo.ID, t.ID, p.City;
  
```

```

SELECT 'Allbooks', t.ID, p.City, SUM(bu.cost), SUM(bu.copies)
FROM Books bo, Buys bu, Providers p, Time t
WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
GROUP BY t.ID, p.City
ORDER BY t.ID, p.City;
  
```

```

SELECT 'Allbooks', 'Alldays', p.City, SUM(bu.cost), SUM(bu.copies)
FROM Books bo, Buys bu, Providers p, Time t
WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
GROUP BY p.City
ORDER BY p.City;
  
```

```

SELECT p.City, SUM(bu.cost), SUM(bu.copies)
FROM Books bo, Buys bu, Providers p, Time t
WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
GROUP BY p.City
ORDER BY p.City;
  
```

```

SELECT p.City, SUM(bu.cost), SUM(bu.copies)
FROM Books bo, Buys bu, Providers p, Time t
  
```

```

WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
AND p.City = 'BARCELONA'
GROUP BY p.City
ORDER BY p.City;

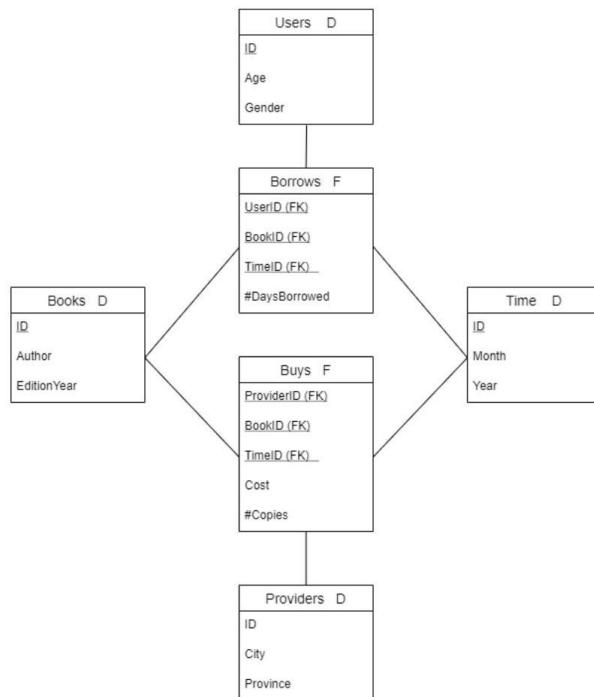
SELECT p.City, SUM(bu.copies)
FROM Books bo, Buys bu, Providers p, Time t
WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
AND p.City = 'BARCELONA'
GROUP BY p.City
ORDER BY p.City;

SELECT p.ID, SUM(bu.copies)
FROM Books bo, Buys bu, Providers p, Time t
WHERE bu.BooksID = bo.ID AND bu.TimeID = t.ID AND bu.ProviderID = p.ID
AND p.City = 'BARCELONA'
GROUP BY p.ID
ORDER BY p.ID;

SELECT p.ID, SUM(bu.copies)
FROM Buys bu, Providers p
WHERE bu.ProviderID = p.ID AND p.City = 'BARCELONA'
GROUP BY p.ID
ORDER BY p.ID;

```

6. Write multidimensional queries using the following constellation schema and the steps required in each exercise:



1. • A:=Roll-up(borrows, Users.Gender, Sum)
 • B:=Roll-up(A, Time.Month, Sum)
 • R:=Selection(B, Time.Month='January')

```
SELECT u.Gender, b.BookID, t.Month, SUM(b.#DaysBorrowed)
FROM Borrows b, Time t, Users u
WHERE b.UserID = u.ID AND b.TimeID = t.ID AND t.Month = 'January'
GROUP BY u.Gender, b.BookID, t.Month
ORDER BY u.Gender, b.BookID, t.Month
```

2. • A:=Roll-up(buys, Providers.AllProviders, Sum)
 • B:=Roll-up(A, Books.Allbooks, Sum)
 • C := Roll-up(B, Time.AllDays, Sum)
 • D := ChangeBase (C, { Providers })
 • R := Projection(D, #Copies)

```
SELECT 'AllProviders', SUM(bu.#copies)
FROM Buys bu
```

3. • A:=Roll-up(borrows, Users.AllUsers, Sum)
 • B:=Roll-up(A, Books.EditionYear, Sum)
 • C := ChangeBase(B, { Books, Time })
 • D := Drill-across (C, buys)
 • R := Projection(D, Cost)

```
SELECT bo.EditionYear, b.TimeID, ('AllProviders',) SUM(bu.Cost)
FROM Borrows b, Books bo, Buys bu
WHERE b.BookID = bo.ID AND b.BookID = bu.BookID AND b.TimeID = bu.TimeID
GROUP BY bo.EditionYear, b.TimeID
ORDER BY bo.EditionYear, b.TimeID
```

4. • A:=Roll-up(buys, Providers.AllProviders, Sum)
 • B := ChangeBase (A, { Books, Time })
 • C := Drill-across (B, borrows, Sum)
 • D:=Projection(C, #Daysborrowed)
 • R:=ChangeBase(D, { Books, Time, Users })

```
SELECT b.BookID, b.TimeID, 'AllUsers', SUM(b.#DaysBorrowed)
FROM Buys bu, Borrows b
WHERE b.BookID = bu.BookID AND b.TimeID = bu.TimeID
GROUP BY b.BookID, b.TimeID
ORDER BY b.BookID, b.TimeID
```

Hay que mirar el nivel de agregación (los atributos de agregación en el GROUP BY). Como userID no aparece, el nivel de agregación es AllUsers. Se pueden eliminar las dimensiones porque la relación es de FK y PK y no hay diferencias de tuplas.

5. • A:=Roll-up(borrows, Time.Month, Sum)
 • B:=Selection(A, Time.Month='January')
 • R:=Drill-down(B, Time.Id)

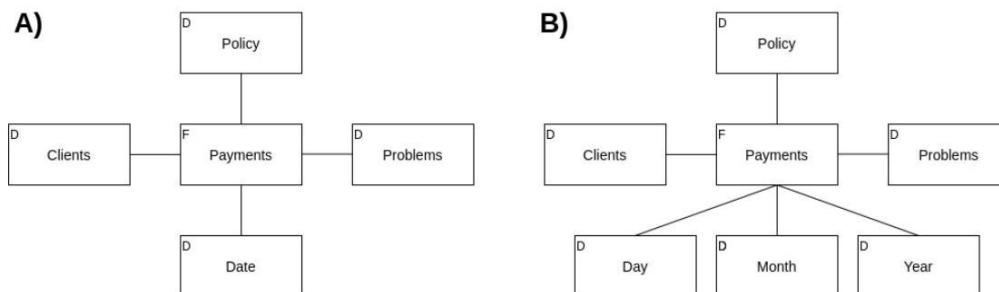
```
SELECT b.BookID, t.ID, b.UserID, SUM(b.#DaysBorrowed)
FROM Borrows b, Time t
WHERE b.TimeID = t.ID AND t.Month = 'January'
GROUP BY b.BookID, t.ID, b.UserID
ORDER BY b.BookID, t.ID, b.UserID
```

7. Identify the problem in this sequence of algebraic multidimensional operations, and briefly explain how you would solve it.

- A:=Roll-up(Sales, Providers.Province, Sum)
- B := ChangeBase(A, { Books, Time })
- C := Drill-across(B, Loans, Sum)
- D := Projection(C, days)
- R:=ChangeBase(D, { Books, Time, Users })

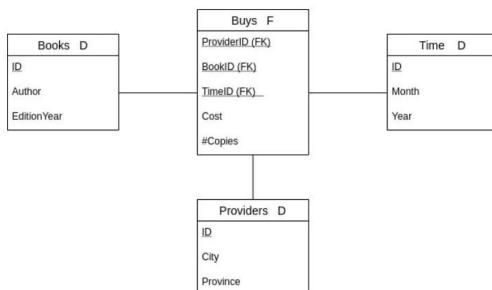
En el primer cambio de base para obtener B, la dimensión Providers tendría que ser una constante (el nivel mas alto de la jerarquía de agregación 'AllProviders', granularidad más baja) para poder realizar su eliminación.

8. Which schema of options A and B would you choose and why:



Escogería el esquema de estrella A ya que nos evitamos hacer más joins. Las tablas de Dia, Mes y Año están juntas en una sola tabla de Fecha. Aunque se repitan muchos datos es la mejor opción. Tiene algunos problemas ya el número de filas de la tabla de Fecha puede aumentar bastante, pero no superaría la cantidad de datos de la tabla de hechos.

9. Given the multidimensional schema and the sequence of multidimensional operations below, give the equivalent SQL query (simplify it as much as possible).



- A:=Roll-up(Buys, Providers.AllProviders,Sum)
- B:=Roll-up(A, Books.AllBooks,Sum)
- C:=Roll-up(B, Time.AllDays,Sum)
- D := ChangeBase (C, { Providers })
- E := Projection(D, #copies)
- R:=Drill-down(E, Providers.City)

```

SELECT p.City, SUM(bu.copies)
FROM Providers p, Buys bu
WHERE p.ID = bu.ProviderID
GROUP BY p.City
ORDER BY p.City;

```

10. Give the relational representation (i.e., tables and their corresponding integrity constraints) of the geographical dimension for a data mart in United Nations, so that it allows to keep track of changes in territories. Assume you only need to keep track of countries and their first administrative level (e.g., Autonomous communities in Spain). We would like to consider the split of countries (like in the case Catalonia would become independent), as well as annexations (like the case of Crimea peninsula by Russian Federation, assuming that peninsula was already a pre-existing component of Ukraine at the first administrative level). Briefly justify your answer and explicit any assumption you make.

Territories (Country, FAL)

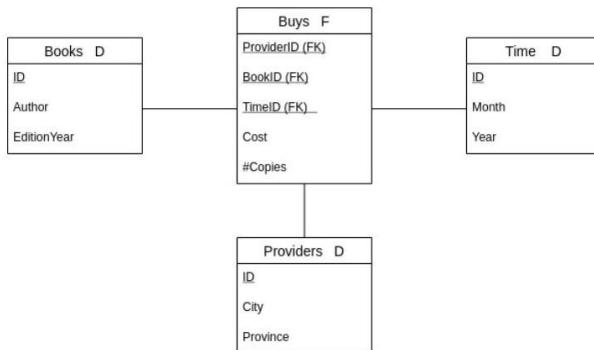
Tenemos diferentes opciones, una de ellas es añadir una columna para cuando haya un cambio en los territorios. En el caso que haya cambiado esa columna tendría el valor anterior y en el caso que no tendría valor null.

Territories (Country, FAL, OldFAL)

Otra opción sería añadir el campo de valid time para saber cuando se ha producido el cambio y poder ver la evolución, este caso es más difícil pero generaliza mejor, podemos poner más cambios.

Territories (Country, FAL, ValidTime)

11. Given the multidimensional schema and the sequence of multidimensional operations below, give the equivalent SQL query (simplify it as much as possible).



- A:=Projection(Buys, Cost)
- B:=Roll-up(A, Providers.Province, Sum)
- C := Selection(B, Province='Barcelona')
- D:=Drill-down(C, Providers.City, Sum)
- E:=Selection(D, TimeID='20210115')
- R:=ChangeBase(E, { Providers, Books })

```

SELECT bo.ID, t.ID, p.City, SUM(bu.cost)
FROM Books bo, Time t, Providers p, Buys bu
WHERE bo.ID = bu.BookID AND t.ID = bu.TimeID AND p.ID = bu.ProviderID
AND p.Province = 'BCN' AND t.TimeID = '20210115'
GROUP BY bo.ID, t.ID, p.City
ORDER BY bo.ID, t.ID, p.City;

```

No se puede hacer el cambio de base ya que la dimensión de Time no está en la jerarquía de agregación más alta ('AllTime').

12. Identify the main problem in the multidimensional schema below, and propose a solution to solve it. You should assume athletes do not change the club in the middle of the year.

D	Athlete	Club	F	Athlete	Year	Income	D
Club	Leo Messi	FCB	Athlete	2020	30M€ <th>Year</th> <td>...</td>	Year	...
...	Rafa Nadal	NULL	...	Rafa Nadal	2020	34M€	...

El principal problema, desde un punto de vista relacional, es que hay una violación de la clave foránea ya que esta es NULL, no puede ser ya que la clave primaria a la que referencia no puede ser NULL. Hay un error de completez, deberíamos añadir valores ficticios para que cuando agreguemos de el valor correcto. Por ejemplo si agregamos el income por clubes no nos dará el valor total de income ya que hay deportistas que no tienen equipo. Deberíamos añadir a los deportistas individuales un valor fantasma como 'Individual' en la tabla que apunte a la dimensión.

13. The multidimensional schema below corresponds to events in football matches with and without ball (e.g., pass, tackling) involving players with any role in the team (e.g., goalkeeper, defender). Write a cube-query (SQL) over it. If you cannot do it, briefly explain why.

No tiene atributos en la tabla de hechos, no da información . P.e no podemos hacer agregaciones. Podemos contar número de faltas, goles, etc. (*factless fact*). Pero ninguna consulta más.

Appendix C Extraction, Transformation and Load

C.1 Theoretical questions

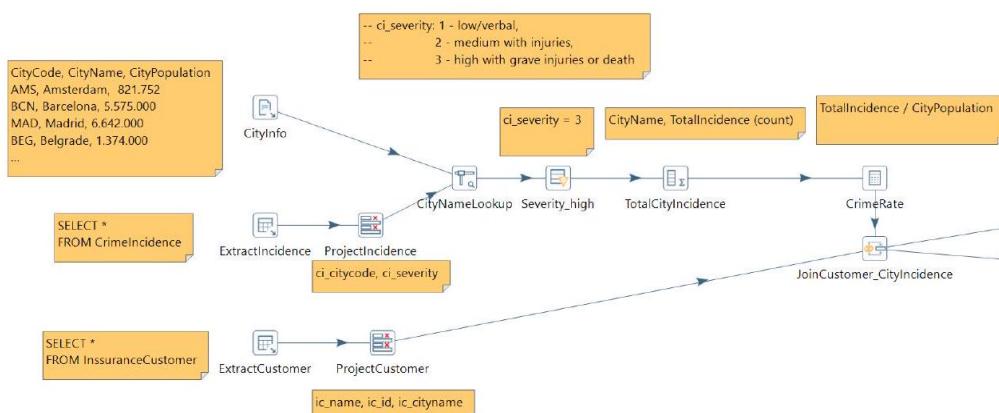
1. Name three extraction mechanisms using exclusively the DBMS and briefly explain the requirements each of these pose on it.
 - **Trigger-based**
 - **Log-based:** dietario donde se guardan todas las operaciones de la base de datos por si se va la luz de manera que se puede reejecutar todo.
 - **Timestamp:** Si estamos delante una BD temporal y tiene una gestión de datos temporal podemos ver qué se ha modificado desde la última carga que hicimos en el DW.

2. Name three advantages of creating ETL processes using ETL tools (against hand-coded scripts).
 - **Repositorio de metadatos integrado**
 - **Análisis automático de la línea de datos y dependencias**
 - **Manejo de dependencias complejas y manejo de errores**

3. Which ETL process quality characteristic is primarily improved by introducing Logical and Conceptual designs? Briefly explain how.
 Sobre todo al mantenimiento de la BD. Un poco también la auditabilidad ligada a que se entiendan mejor las cosas. Al rendimiento y a la robustez no ayuda.

C.2 Problems

1. A worldwide life insurance company, performs periodically a check for all its customers of the crime rate in their city of residence (by using an external DB with CrimeIncidence registry) in order to adapt their policies accordingly.
 - The first problem is that the CrimeIncidence registry is using city codes, while the company's Customer DB table stores city names instead.
 - The company is specifically interested in high severity incidences (those that can yield in customer death).
 - The CrimeRate is calculated as the total number of incidences in the city divided by the city's population.



a. Propose possible performance optimizations of the ETL flow.

- i. Considering that the Incidence DB does not provide any query mechanism (it only provides reading the entire incidence registries).

Como el `join` es la operación más cara, filtrar por `ci_severity` antes de realizar el `join`. Hacer el conjunto de datos los más pequeño posible y lo antes posible.

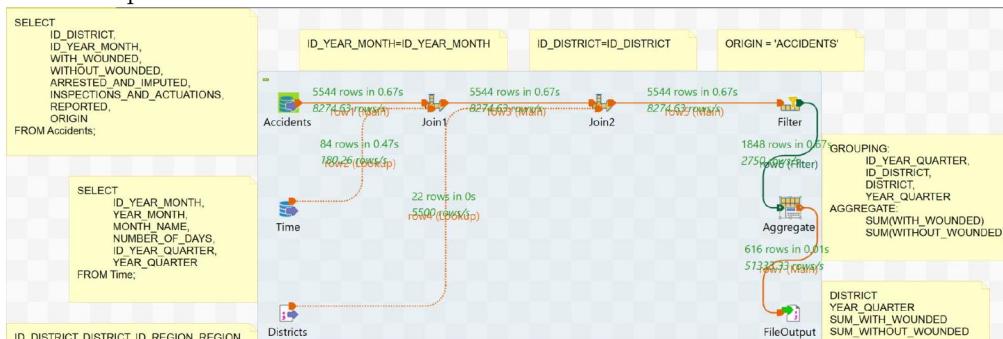
Podemos cambiar también el `groupby`. Mirando que el nombre de la ciudad viene definido por el `ci_citycode`, se puede modificar el `groupby` en función del `ci_citycode` para moverlo antes del `join` y después del filtrado.

ii. Exploiting the functionalities of the source DBMS engine.

La selección (filtrado) en vez de ponerla en la rama ponerla directamente en la BD. Le pedimos al BD que nos envíe los datos que toquen. Pasa lo mismo con las proyecciones, no tiene sentido cargar atributos para después eliminarlos. Nos quedamos solo con los atributos que sean útiles. Si se puede reducir el número de filas y de columnas antes de cargar los datos mucho mejor. Esto no se puede hacer con ficheros. El GROUP BY también lo podemos poner dentro del SGBD. Esto es más discutible. Hacer una selección/proyección es "gratis", en cambio hacer el GROUP BY requiere memoria que tenemos que dedicar a realizar esta consulta. Por lo tanto hacerlo en el servidor tiene un coste, este servidor es nuestro o no lo es? el servidor se esta ahogando? puede ser que estemos afectando a otros usuarios. Lo mismo pasa con las joins.

RESUMEN: intentamos reducir el numero de datos lo antes posible, las selecciones y proyecciones las ponemos lo antes posible y otras operaciones más costosas las tendríamos que consultar con el gestor de la BD para ver si son viables.

2. Given the ETL below, where the stickers indicate the attributes and conditions of the nearest node, briefly explain how you would optimize its performance (a.k.a. execution time) or justify why you think it is already optimal. Explicit any assumption you need to make and annotate any change in the same diagram to facilitate the explanation.



Primero de todo podríamos mover el filtro por origen antes del primer join para filtrar directamente desde accidentes. De esta manera recudimos el número de filas antes de entrar al join. También podemos mover el group by antes del segundo join, puede parecer que no ya que no tenemos el atributo de District porque se obtiene de el csv Districts, pero podemos suponer que el ID_district distingue únicamente el distrito y por lo tanto podemos hacerlo. Por último también podríamos seleccionar solo los campos que necesitamos en vez de cargar tantos atributos de la BD.

Appendix D Data Quality

D.1 Problems

1. Compute the completeness ($Q_{Cm}(R)$) of the following tables:
 (a) 50%

ID	A	B	C	D	E
1	a1	b1	c1	d1	e1
2	a2	b2	c2	d2	e2
3	a3	b3	c3	d3	e3
4	a4	b4	c4	d4	e4
5	a5	b5	c5	d5	e5
6	null	b6	c6	d6	e6
7	a7	null	c7	d7	e7
8	a8	b8	null	d8	e8
9	a9	b9	c9	null	e9
10	a10	b10	c10	d10	null

(b) 90%

ID	A	B	C	D	E
1	a1	b1	c1	d1	e1
2	a2	b2	c2	d2	e2
3	a3	b3	c3	d3	e3
4	a4	b4	c4	d4	e4
5	a5	b5	c5	d5	e5
6	a6	b6	c6	d6	e6
7	a7	b7	c7	d7	e7
8	a8	b8	c8	d8	e8
9	a9	b9	c9	d9	e9
10	null	null	null	null	null

2. Compute the accuracy ($Q_A(R)$) of the following table, ignoring all attributes but A. Consider that the true value of attribute A is the inverse of the ID (i.e., ID^{-1}) and $\epsilon = 0.3$:

ID	A	B	C	D	E
1	1	b1	c1	d1	e1
2	0.9	b2	c2	d2	e2
3	0.8	b3	c3	d3	e3
4	0.7	b4	c4	d4	e4
5	0.6	b5	c5	d5	e5
6	0.5	b6	c6	d6	e6
7	0.4	b7	c7	d7	e7
8	0.3	b8	c8	d8	e8
9	0.2	b9	c9	d9	e9
10	0.1	b10	c10	d10	e10

$$\begin{aligned}
& |1/1 - 1| \leq \epsilon \\
& |1/2 - 0.9| > \epsilon \rightarrow +1 \\
& |1/3 - 0.8| > \epsilon \rightarrow +1 \\
& |1/4 - 0.7| > \epsilon \rightarrow +1 \\
& \dots \\
& Q_A(R) = 5/10 = 0.5
\end{aligned}$$

The ID's 1, 7, 8, 9 and 10 do comply with the accuracy threshold $\epsilon = 0.3$

3. Compute the timeliness ($Q_T(X)$) of each of the attributes A, B , and C of the following table, considering that A changes once per second (i.e., $f_u(v) = 1$), B changes once every 10 seconds (i.e., $f_u(v) = \frac{1}{10}$), C changes once every 100 seconds (i.e., $f_u(v) = \frac{1}{100}$); and the different values changed depending on the id of the row and the attribute $age_A(v) = id \cdot 100$ seconds, $age_B(v) = id \cdot 10$ seconds, and $age_C(v) = id$ seconds:

ID	A	B	C	D	E
1	a1	b1	c1	d1	e1
2	a2	b2	c2	d2	e2
3	a3	b3	c3	d3	e3
4	a4	b4	c4	d4	e4
5	a5	b5	c5	d5	e5
6	a6	b6	c6	d6	e6
7	a7	b7	c7	d7	e7
8	a8	b8	c8	d8	e8
9	a9	b9	c9	d9	e9
10	a10	b10	c10	d10	e10

$$Q_T(A) = \frac{1}{10} \cdot \sum_{id=1}^{10} \left(\frac{1}{1 + id \cdot 100} \right) = 0.002913$$

$$Q_T(B) = \frac{1}{10} \cdot \sum_{id=1}^{10} \left(\frac{1}{1 + id} \right) = 0.20198$$

$$Q_T(C) = \frac{1}{10} \cdot \sum_{id=1}^{10} \left(\frac{1}{1 + \frac{1}{100} \cdot id} \right) = 0.94857$$

$$Q_T(R) = \frac{1}{5} \cdot (Q_T(A) + Q_T(B) + Q_T(C) + Q_T(D) + Q_T(E))$$

4. Compute the consistency ($Q_{Cn}(R, BR)$) of the table considering the set of business rules BR contains these:
- All rows must have its id number in all the values.
 - All values in a column must start with the letter of the corresponding attribute.

ID	A	B	C	D	E
1	a1	b1	c1	d1	e10
2	a2	b2	c2	e2	e2
3	a3	b3	k5	d3	e3
4	ab	ba	c4	d4	e4
5	a5	b5	c5	d5	e5
6	a6	b6	c6	d6	e6
7	a7	b7	c7	d7	e7
8	a8	b8	c8	d8	e8
9	a9	b9	c9	d9	e9
10	a10	b10	c10	d10	e10

- **Fila 1:** no, E tiene 10 y no 1 (si se entiende el atributo como string, entonces podríamos interpretar que el valor si que contiene el 1 dentro del valor y entonces si cumpliría.)
- **Fila 2:** no, D empieza por e
- **Fila 3:** no, c empieza por k y no tiene 3
- **Fila 4:** no tiene números en A y B
- **Fila 5:** si
- **Fila 6:** si
- **Fila 7:** si
- **Fila 8:** si
- **Fila 9:** si
- **Fila 10:** si

Un 60% de las filas cumplen todas las business rules.

5. Which logic property (or properties) of data quality rules is (are) violated by the given relational schema? Briefly explain why.

```

CREATE TABLE Flights (
    flightID CHAR(26) NOT NULL,
    departureAirport CHAR(3) NOT NULL REFERENCES Airports (ID),
    arrivalAirport CHAR(3) NOT NULL REFERENCES Airports (ID)
    actualDeparture TIMESTAMP WITHOUT TIME ZONE,
    actualArrival TIMESTAMP WITHOUT TIME ZONE,
    duration INTERVAL,
    CONSTRAINT ckDepartureAirport CHECK (departureAirport <> 'BCN'),
    CONSTRAINT ckArrivalAirport CHECK (arrivalAirport = 'BCN'),
    CONSTRAINT ckMinDuration CHECK (duration > interval '02:30:00'),
    CONSTRAINT ckMaxDuration CHECK (duration < interval '01:25:00')
);

CREATE TABLE Airports (
    id CHAR(3) PRIMARY KEY,
    name VARCHAR(20),
    CONSTRAINT ckAirport CHECK (id <> 'BCN')
);

```

El esquema es satisfactible ya que podemos inserir una fila en la tabla Airports.

1. Falta *primary key* en la tabla `flights`, pero esto no afecta en nada en las propiedades lógicas (no viola ninguna de las reglas), esto no significa que esté bien.
2. Los dos últimos checks de la tabla `flights` son contradictorios, pero nos salva que `duration` puede ser `NULL` y por lo tanto podríamos insertar filas.
3. `arrivalAirport` tiene que ser BCN y hace referencia a `Airports` donde la `id` tiene que ser diferente que BCN. Nos podría salvar que se permitiesen valores nulos en `arrivalAirport` pero pone `NOT NULL`.
4. El check en `Flights` sobre que `departureAirport` sea diferente que BCN es redundante, ya que es clave foránea a `Airports` y allí ya se comprueba, por lo tanto deberíamos eliminar la constraint de la tabla `Flights`.

6. Logic properties of data quality rules:
 - (a) Which logic property (or properties) of data quality rules is (are) violated by the given relational schema? Briefly explain why.

```

CREATE TABLE Airports (
    id CHAR(3) PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    capacity INT NOT NULL,
    CONSTRAINT ckCapacity CHECK (capacity > 1200)
);

CREATE TABLE Pilots (
    id SMALLINT PRIMARY KEY,
    firstName VARCHAR(20) NOT NULL,
    lastName VARCHAR(20) NOT NULL
    hubAirport CHAR(3) NOT NULL REFERENCES Airports(id),
    CONSTRAINT ckAirportPilot CHECK (hubAirport <> 'BCN')
);

CREATE TABLE TechnicalLogBookOrders (
    workOrderID INT PRIMARY KEY,
    aircraftRegistration CHAR(6) NOT NULL,
    executionDate DATE,
    executionPlace CHAR(3) NOT NULL REFERENCES Airports(id),
    reporterID SMALLINT NOT NULL REFERENCES Pilots(id),
    reportingDate DATE,
    due DATE,
    deferred BOOLEAN,
    CONSTRAINT ckAirportOrder CHECK (executionPlace = 'BCN')
);

CREATE VIEW anonymousReports AS (
SELECT *
FROM TechnicalLogBookOrders t WHERE NOT EXISTS(
    SELECT *
    FROM Pilots b
    WHERE b.id = t.reporterID));
  
```

Las tres tablas están vivas pues se puede agregar por lo menos una fila en cada una de ellas. Las restricciones de integridad no presentan problemas. Por ende el esquema es satisfactible.

La view no está viva pues según las restricciones de las tablas, no tendrá la vista ninguna fila. Esto sucede porque en la vista se deben mostrar las filas de `t` en las cuales no haya pilotos de `t` (que hayan reportado) en la tabla `p`. Esto claramente nunca se satisface, pues el piloto de `t` referencia a la tabla `p`.

- (b) Given the relational schema above, is the state represented with the instances of the Pilots and Airports tables below, reachable? Briefly explain why.

Pilots (id, firstName, lastName, hubAirport);	Airports (id, name, Capacity)
1 Jerry Garcia BCN	BCN Barcelona-E1 Prat 1250
2 Helen Smith MAD	LHR London-Heathrow 2500
3 Susan Blair LHR	MAD Madrid-Barajas 1300

No porque la primera tupla de la tabla Pilots no cumple con la restriccion de integridad intra-tupla, pues el valor de hubAirport no puede ser 'BCN'.

7. Determine the liveliness of the next three tables. Briefly justify your answer.

```

create table Teams (
    name varchar(25),
    budget int,
    city varchar(15) not NULL,
    primary key (name),
    check (name='Futbol Club Barcelona')
);

create table Players (
    name varchar(30) not NULL,
    age int check (age>0 and age<50),
    team varchar(25) not NULL,
    number int check (number>=0 and number<100),
    primary key (name),
    foreign key (team) references teams(name)
);

create table Matches (
    local varchar(25),
    visiting varchar(25),
    primary key (local, visiting),
    foreign key (local) references teams(name),
    foreign key (visiting) references teams(name)
    check (local <> visiting)
)

```

- Teams está viva si inserimos el Barça.
- Players está viva pero tenemos que poner jugadores del Barça.
- Matches muerta ya que tanto local como visiting referencian Teams que tienen que ser el Barça, pero a la vez el check impide que jueguen Barça contra Barça.

8. For the database schema defined below, list all the Intra-Attribute, Intra-Tuple, Intra-Relation, and InterRelation integrity constraints defined in the schema.

```

CREATE TYPE FrequencyUnitsKind AS ENUM ('Flights', 'Days', 'Miles');

CREATE TABLE ForecastedOrders (
    workOrderID INT PRIMARY KEY,
    aircraftRegistration CHAR(6) NOT NULL REFERENCES Aircrafts(registration),
    deadline DATE NOT NULL,
    planned DATE NOT NULL,
    frequencyUnits FrequencyUnitsKind,
    CONSTRAINT ckAirportOrder CHECK (executionPlace <> 'BCN'),
    CONSTRAINT ckDeadline CHECK (deadline < planned)
);

CREATE TABLE Aircrafts (
    registration CHAR(6) PRIMARY KEY,
    manufacturer_serial_number CHAR(8) NOT NULL UNIQUE,
    aircraft_model VARCHAR,
    manufacturer
);

```

- **Intra-Attribute:**

- Comprobación de valores NOT NULL
- Que sean de un dominio determinado. e.g. INT, DATE.

- **Intra-Tuple:**

- El CHECK sobre executionPlace no se puede hacer ya que executionPlace no es un atributo de la tabla, sino que se refiere a un valor de ckAirportOrder. Por lo tanto solo el segundo check.

- **Intra-Relation:**

- Ambas PRIMARY KEY de las dos tablas y el UNIQUE de la relación Aircrafts.

- **Inter-Relation:**

- La FOREIGN KEY de aircraftRegistration hacia la tabla Aircrafts

9. Consider the following data sources,

- Source 1: Data from a surveillance system about the reported cases of diagnostics of a disease in a country (e.g., Rabies in South Africa).
- Source 2: Data from a medicine distribution system about the number of requested Rabies medicines from South Africa.

Which of the four metrics can we use to measure the quality of these two sources?

Comprobar que tanto los ministerios como las farmacéuticas nos dan datos presentes. Los números tendrían que coincidir, si un ministerio dice que ha enviado 5 medicamentos de la rabia a España, la farmacéutica debería poner que ha pedido 5 medicamentos. Una manera de medir esto sería medir como de cerca se encuentran estos números.

La accuracy, timeliness, etc. son métricas útiles pero no son las únicas. Hay que adaptar las métricas al problema que tenemos.

Appendix E Schema and Data Integration

E.1 Theoretical questions

1. Name the three major steps to solve semantic heterogeneity.
Schema alignment, schema mapping, entity resolution, record merging

2. Which is the main difference between a federated architecture and an architecture based on wrappers and mediators?

Wrappers esconden la complejidad. A partir de aquí todos tienen el mismo formato. El segundo paso es otra pieza de software, el **mediador**, que dados diferentes fuentes de datos y una consulta hace esta consulta global y la divide entre los diferentes data sources.

La diferencia es que en la federada es más institucional (más rígida), único esquema global al cual intentamos hacer una consulta centralizada. En la de wrappers y mediators hay más autonomía.

3. Give an example of schematic discrepancy (different from that in the slides), where data in one source is represented as metadata in the other. Draw and briefly explain it.

En *one hot encoding* convertimos datos (variables categóricas) en metadatos (columnas de categorías con valores binarios). Entonces en una tabla Jugadores se puede tener nombre, años, club, etc. En donde club contiene los diferentes **datos** de los nombres de los clubs (Barça, Madrid, etc). En cambio una representación del tipo nombre, años, Barça, Madrid presentaría a los datos de los clubs como metadatos y habría una codificación one-hot para indicar en cuál de estos club está el jugador.

4. Which is the worst-case cost of the R-Swoosh algorithm and when would it happen?

El peor caso es cuando todos tienen similitud pero no nos damos cuenta hasta el final del bucle, es decir, siempre encontramos como similar el último elemento del output.

La función de similitud se ejecuta más veces, pues la complejidad del algoritmo es cuadrática. Mientras que en el mejor caso, la función de *merge* se ejecuta tantas veces como el orden de cardinalidad de la entrada (todos coinciden).

E.2 Problems

1. Suppose that there are two dealers A and B, who respectively use the schemas:

We define a global schema like the first one, except that we record the option of having an automatic transmission, and we include an attribute that tells which dealer has the car.

AutosGlobal(serialNo, model, color, autoTrans, dealer)

- a) Which queries should be issued to each wrapper to answer:

```
SELECT serialNo, model  
FROM AutosGlobal  
WHERE color=' red',
```

Dealer A:

```
SELECT c.serialNo, c.model  
FROM Cars c  
WHERE c.color= 'red'
```

Dealer B:

```
SELECT a.serial, a.model  
FROM Autos a  
WHERE a.color= 'red'
```

b) If dealer A wants to insert a new row through the global schema, what SQL insert statements should it use?

```
INSERT INTO AutosGlobal VALUES (serialNox, modelx, colorx,  
autoTransx, 'Dealer A')
```

c) Suppose now that dealer A records 0 or 1 in attribute autoTrans, while the global schema needs to show respectively 'no' or 'yes' for the same attribute. Show how the translation from Cars to AutosGlobal could be done by an SQL query.

```
SELECT c.serialNo, c.model, c.color,  
CASE WHEN c.autoTrans = 1 THEN 'yes' ELSE 'no' END,  
'Dealer A'  
FROM Cars c
```

d) Assuming that a query just passes through the wrapper unchanged until it reaches the relational data source, what would a mediator do in the presence of the following queries (although a mediator is not necessarily implemented in SQL, you can express this answer in terms of SQL):

i. Find the serial numbers of cars with automatic transmission.

```
SELECT c.serialNo AS serialNo  
FROM Cars c  
WHERE c.autoTrans = 1  
UNION  
SELECT o.serial AS serialNo  
FROM Options o  
WHERE o.option = 1
```

ii. Find the serial numbers of cars without automatic transmission

```
SELECT c.serialNo AS serialNo  
FROM Cars c  
WHERE c.autoTrans = 0  
UNION  
SELECT o.serial AS serialNo  
FROM Options o  
WHERE o.option = 0
```

iii. Find the serial number of the blue cars from dealer A.

```
SELECT c.serialNo AS serialNo  
FROM Cars c
```

```
WHERE c.color = 'Blue'
```

- e) Suppose now that we have a semantic heterogeneity in the extension of the attribute color, and it is not codified in the same way in the local schema of Dealer A and the global schema. Assuming that we have a relation GtoL(globalColor, localColor), ³ that translates between the two sets of colors, modify the following wrapper template appropriately:

Template	Query
<pre>SELECT c.serialNo, c.model, g.globalcolor, c.autoTrans, 'DealerA' FROM Cars c, GtoL g WHERE c.color = g.localColor AND g.globalColor='\\$c';</pre>	<pre>SELECT serialNo, model, color, autoTrans, 'dealerA' FROM AutosGlobal WHERE color='\\$c';</pre>

2. Computer company A keeps data about the PC models it sells in the schema:

```
Computers(number, proc, speed, memory, hd)
Monitors(number, screen, maxResX, maxResY)
```

For instance, the tuple [123, Athlon64, 3.1,512,120] in Computers means that model 123 has an Athlon 64 processor running at 3.1 gigahertz, with 512Mb of memory and 120 Gb hard disk. The tuple [456,19,1600,1050] in Monitors means that model 456 has a 19 -inch screen with a maximum resolution of 1600×1050 .

Computer company B only sells complete systems, consisting of a computer and monitor. Its schema is:

```
Systems(id, processor, mem, disk, screenSize)
```

The attribute processor is the speed in gigahertz; the type of processor (e.g., Athlon 64) is not recorded. Neither is the minimum and maximum resolution of the monitor recorded. Attributes id, mem, and disk are analogous to number, memory, and hd from company A, but the disk size is measured in Mb instead of Gb.

- a) Suggest two other schemas (substantially different but not adding more information) that computer companies C and D might use to hold data like that of companies A and B.

C: ComputerPack(number, proc, speed, memory, hd, screen, maxResX, maxResY)

D: Systems(id, processorID, hd, screenSize)
Processor(ID, speed, memory)

- b) Suppose now that we have three wrappers, one for Computers and one for Monitors, in company A, and one for Systems in company B. Which queries should be issued by each wrapper to answer:

- i. Find the system with 512Mb of memory, an 80 Gb hard disk, and a 22 inch monitor.

Computers

```
SELECT c.number
FROM Computers c
WHERE c.memory = 512 AND c.hd = 80
```

Monitors

```
SELECT m.number
FROM Monitors m
WHERE m.screen = 19
```

Systems

```
SELECT s.id
FROM Systems s
WHERE s.mem = 512 AND s.disk = 81,920 AND s.screenSize = 19
```

ii. Find the system with a Pentium-IV processor, running at 3.0 gigahertz with a 22 -inch monitor and a maximum resolution of 1600-by-1050.

Computers

```
SELECT c.number
FROM Computers c
WHERE c.proc = 'Pentium-IV' AND c.speed = 3.0
```

Monitors

```
SELECT m.number
FROM Monitors m
WHERE m.screen = 22 AND m.maxResX ≤ 1600 AND m.maxResY ≤ 1050
```

Systems

```
SELECT s.id
FROM Systems s
WHERE s.screenSize = 22 AND s.processor = 3
```

iii. Find all systems with a G5 processor running at 1.8 gigahertz, with 2 gigabytes of memory, a 300 gigabyte disk, and a 19-inch monitor.

Computers

```
SELECT c.number
FROM Computers c
WHERE c.proc = 'G5' AND c.speed = 1.8 AND c.memory = 2 AND c.hd
= 300
```

Monitors

```
SELECT m.number
FROM Monitors m
WHERE m.screen = 19
```

Systems

```
SELECT s.id
FROM Systems s
WHERE s.processor = 1.8 AND s.mem = 2 AND s.disk = 300 AND
s.screenSize = 19
```

3. We want to gain integrated access to two databases (DB1 and DB2). The former contains two tables (T11 and T12), while the later contains only one table (T2). In order to implement this, we think that the best architecture is having two wrappers and one mediator, just like depicted in the figure.

Fill the following table expliciting for each necessary call (having 5 gaps does not mean there must be exactly 5 calls) to obtain the result of the query above, who is in charge of its execution (either the mediator or one of both wrappers) and the corresponding SQL query. Write the calls following their execution order.

Order	Executor	SQL sentence
1	Wrapper1	<pre>SELECT T11.a, T12.b, T12.link FROM T11, T12 WHERE T11.link = T12.link AND T11.slicer = X</pre>
2	Wrapper2	<pre>SELECT T2.c, T2.link FROM T2</pre>
3	Mediator	<pre>SELECT W1.a, W1.b, W2.c FROM Wrapper1 W1, Wrapper2 W2 WHERE W1.link = W2.link</pre>

³³ Notice that globalColor and localColor are both alternative keys of the table and hence their relationship is one-to-one.

4. Consider an integrated system with two different sources (e.g., Vueling-V and Airbus-A). The tables are the following and the mapping between the local and global schemas is given using GAV with relational algebra expressions as follows ("get_date" is a function with the obvious meaning):

Vueling: Hops(from, to, timestamp, plane, mainIncident)

Airbus: Planes(registry, sits)

- Flights(origin, destination, day, aircraft) := V.Hops[from,to,get_date(timestamp),plane]
- Incidents(aircraft,day, ATACode) := V.Hops[plane,get_date(timestamp), mainIncident]
- Aircrafts(aircraft,capacity) := A.Planes[registry, sits]

Assuming that both databases reside in the same RDBMS (i.e., wrappers are not needed), give the SQL query generated by the mediator on receiving the following global one:

```
SELECT DISTINCT get_date(h.timestamp) AS day, h.plane AS
aircraft
FROM Hops h JOIN Planes p ON h.plane = p.registry
WHERE p.sits > 200 AND h.mainIncident = 21 AND h.from = 'BCN'
```

5. Consider the following table with instances about mobile phones offered for sale extracted from different online retail platforms during the Black Friday promotion.

source	title	model_code	price
AMAZON	SAMSUNG GALAXY A6	SM-A60F2	225
PHONEHOUSE	HUAWEI P20 LITE		299
AMAZON	LG Q6 A		109
AMAZON	HONOR 8X 64G/4G	51092XWQ	249
PHONEHOUSE	HUAWEI P8 LITE		125
PHONEHOUSE	HUAWEI P8	ANE-LX1M235	173
AMAZON	HUAWEI MATE 20 LITE	ANE-LX28K93	320
FNAC	LG G5 SE		239.99
AMAZON	LG G7 THINQ	LGM77675GR	475
FNAC	SAMSUNG GALAXY A7	SM-A750GZ	350
AMAZON	SAMSUNG GALAXY S9+	SM-A7654DP	580
PHONEHOUSE	HONOR 8X 64GB 4GB	51092XWQ	253
BESTBUY	APPLE IPHONE XR		800
AMAZON	APPLE IPHONE XS	A1920	1066.66
AMAZON	APPLE IPHONE X	A1901	650
BESTBUY	IPHONE X	A1901	700.55
PHONEHOUSE	LG Q6 α	LGM700DSK	129
BESTBUY	HUAWEI P8L	ANE-LX1M235L	164.5
PHONEHOUSE	SAMSUNG GALAXY S9 PLUS		599
BESTBUY	SAMSUNG GALAXY S9P		585

The table contains information about the retail platform from where the data is extracted (source), the title of the product in the web page (title), a unique code of the model where available (model_code), and the offered price of the product (price). To integrate these data, the R-Swoosh algorithm is considered.

(a) The DW administrator is thinking of the possible similarity and merge functions for individual attributes of the given table. Answer the following question.

i. Without considering other attributes, is using Edit distance ⁴ on attribute title a good choice for a match function?

- A. If so, justify your answer and propose the possible threshold.
- B. If not, justify your answer.

No sería una buena idea. Por ejemplo nos gustaría unir IPHONE X con APPLE IPHONE X, cuya distancia de Edit es 5. Pero si marcamos un threshold como este hay otros modelos que identificará como similares pero que no son, como por ejemplo el caso de HUAWEI P8 con HUAWEI P8 LITE, que tienen una distancia de edit 4.

ii. Given the characteristics of the attribute `model_code`, which of the following combinations of similarity and merge functions should be used if this is the only attribute considered in the entity resolution process.

Similarity
Edit distance
Hamming distance
Substring matching ⁵
Exact string matching
Merge
Multi-valued ⁶
Source trustworthiness
Generate null ⁶

En este caso cogeríamos la combinación **Substring Matching** con **Source trustworthiness**. La primera es porque las únicas entidades que queremos juntar que son IPHONE X y APPLE IPHONE X tienen exactamente el mismo código de modelo y por lo tanto solo se detectaría como similar las que queremos (ya que las demás no coinciden). La segunda es porque no queremos tener valor multi valued ni tampoco generar un valor nulo por lo tanto cogeremos el valor de la fuente que sea más confiable.

iii. For the attribute `price`, the DW administrator proposed $\frac{\min(r,s)}{\max(r,s)}$ as the similarity function. Thus, we could decide that two records match even if the price does not (but its metric is high enough). Hence, we need a representative for the result of the merge. For each of the following merge functions, explain if it can be used in the R-Swoosh algorithm with the above match function. Justify your answer and if negative, elaborate what consequences it would have.

Merge	Answer (yes/no)	Justification
Mean (average)	no	Si cogemos la media puede ser que el valor que pongamos sea mayor del
Max/Min	sí	Ponemos el máximo por si acaso, el mínimo no por la razón de arriba

También se puede llegar a la conclusión que si la métrica es lo suficientemente alta (precios muy parecidos) tanto mean, max, min se podrían aplicar ya que darían valores similares.

³⁴ https://en.wikipedia.org/wiki/Levenshtein_distance

⁴ Similar if one string is contained inside another or viceversa (e.g., abcccd \approx bcccd)

⁶ For multi-valued attributes, consider they are similar if at least one value from the list matches.

iv. Now, the DW administrator considers that instead of using individual attributes alone in the similarity function, uses a combination of several attributes.

Attribute	Similarity function	Merge function
title	Edit distance, ⁷ with a threshold of 25%	Source trustworthiness with priority in decreasing order: PHONEHOUSE, BESTBUY, FNAC, AMAZON
model_code	Exact string matching	Keep any string
price	$\frac{\min(r,s)}{\max(r,s)}$, with a a a threshold of 90%	Multi-valued

Given that the similarity and merge functions in the table above over individual attributes, and considering data instances given in the mobile phone table, define possible combinations that can be used for entity resolution in order to merge tuples of same mobile phone models and list all offered prices for them. Give examples how these features would work.

Para el ejemplo del iphone tenemos que:

Title

$$dist(\text{IPHONE X}, \text{APPLE IPHONE X}) = \frac{\text{editDistance}(x, y)}{\max(\text{length}(x), \text{length}(y))} = \frac{5}{14} = 0.36 > 0.25$$

En este caso no las identificamos como similares.

Model code

El código A1901 es el mismo por lo tanto las detectamos como similares y nos quedamos con cualquiera.

Price

$$dist(650, 700.55) = \frac{\min(r, s)}{\max(r, s)} = \frac{650}{700.55} = 0.928 > 0.9$$

En este caso si que los consideramos similares ya que supera el threshold y cogemos los dos valores.

³⁷ Edit distance relative to the length of the strings is calculated as follows: $\frac{\text{editDistance}(x, y)}{\max(\text{length}(x), \text{length}(y))}$

6. Consider the following table with instances about travel arrangements offered for sale, extracted from different online travel platforms during the Christmas holidays promotion.

The table contains information about the travel platform from where the data are extracted (source), the title of the offer in the web page, a destination of the trip which is a multivalued attribute (assuming the destination names are same in all the sources), duration where the units may vary (days/weeks), type of the offer if available - multi-valued attributed with fixed and unique type names (enum), and the offered price of the product in different currencies depending on the location of the travel agency (for some offers the price is given only on the explicit demand).

To compare similar offers, and present them jointly to the user, the R-Swoosh algorithm is considered for integrating these sources.

(a) The DW administrator is thinking of the possible similarity functions for individual attributes of the given table. Answer the following questions.

i. Would attribute title be a good choice to define the similarity function exclusively over it, without considering other attributes?

No es buena opción ya que los títulos no son los mismos para viajes iguales y difieren mucho. Por lo tanto no identificaríamos el mismo objeto como similar. Un ejemplo sería para la destinación *New York, Washington* que tenemos títulos: *The best of East Cost, NYC and Washington* y *New York, New Year*.

ii. Given the characteristics of the attribute destination, which of the following similarity functions could be used. Explain the consequences of each solution.

Similarity functions
Substring matching ⁸
Exact string matching
Multi-valued ⁴

Substring matching: funcionaría bastante bien para la mayoría de casos como New York, Washington y Beijing, Xi'an pero fallaría en casos como por ejemplo New York que lo juntaría con viajes combinados que contienen New York como New York, Cancun.

Exact string matching: funcionaría solo para los casos que las distinciones son idénticas pero se dejaría el caso de por ejemplo Beijing, Xi'an con Shanghai, Beijing, Xi'an.

Multi-valued: no funcionaría porque uniría casos como el de Venice, Rome, Naples, Palermo con Venice, Rome, Naples, Suez, Aqaba ya que tiene elementos iguales pero no son el mismo viaje. Funcionaría por eso para otros casos como cuando coinciden exactamente o el caso de Beijing, Xi'an.

iii. If possible, propose an improvement to using multi-valued matching for the destination attribute in the previous exercise.

7. Consider the two relational schemas below (primary key attributes underlined). Would the instances of these schemas represent the same reality? Briefly explain why and exemplify it.

- a) Flight(flightNum: String, origin: Airport, destination: Airport)
- b) FlightEnd(flightNum: String, type: {Origin, Destination}, place: Airport)

Sí que representan el mismo objeto del mundo real, en este caso un vuelo concreto. En el primer caso (Flight) este vuelo está definido por el aeropuerto de llegada y salida, mientras que en el segundo (FlightEnd) está definido por la ciudad de origen (Origin) y el destino (Destination). En cualquier caso, se refieren a lo mismo pero con diferentes atributos.

8. Given two book providers with the database schemas below, write one SQL sentence returning the data in both schemas and loading dimension table BooksDim (below). Specifically: ID, Author, Edition Year, NobelPrize, Genre. Consider that some books can be offered by both providers at the same time. In this case, all the attributes except the Genre should prevail from the Provider1, if available. In case Genre is not available, it should be marked as "Miscellaneous".

```
SELECT FROM P1_Books, P1_Nobels, P2_Books
```

1 Distributed Data

1.1 Theoretical questions

1. Give five types of data or software resource that can usefully be shared. Give examples of their sharing as it occurs in practice in distributed systems.

Videojuegos, internet, etc

2. Consider the implementation strategies for massively multiplayer online. In particular, what advantages do you see in adopting a single server approach for representing the state of the multiplayer game? What problems can you identify and how might they be resolved?¹

mantenimiento mas facil al tener menos complejidad, la comunicacion/sincronizacion no se tienen en cuenta. La rapidez dependera del numero de usuarios y las eficiencias del servidor. Falta de escalabilidad: Un solo servidor no aguantara el aumento de los usuarios y fallara. Lo cual el sistema falla al solo haber uno.

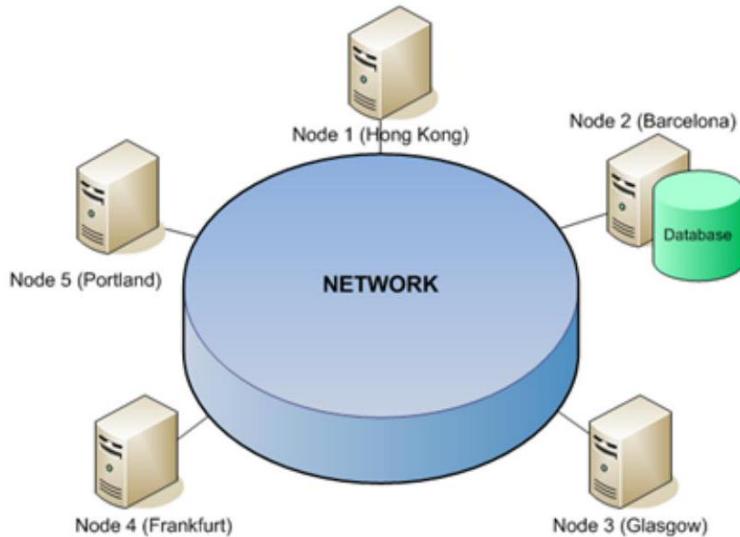
3. The INFO service manages a potentially very large set of resources, each of which can be accessed by users throughout the Internet by means of a key (a string name). Discuss an approach to the design of the names of the resources that achieves the minimum loss of performance as the number of resources in the service increases. Suggest how the INFO service can be implemented so as to avoid performance bottlenecks when the number of users becomes very large.¹

4. A server program written in one language (for example, C++) provides the implementation of a BLOB object that is intended to be accessed by clients that may be written in a different language (for example, Java). The client and server computers may have different hardware, but all of them are attached to the Internet. Describe the problems due to heterogeneity of (a) programming languages, and (b) implementations by different developers that need to be solved to make it possible for a client object to invoke a method on the server object.¹

Give arguments for and against allowing the client requests to be executed concurrently by the server. In the case that they are executed concurrently, give an example of possible "interference" that can occur between the operations of different clients. Suggest how such interference may be prevented.

5. A service is implemented by several servers. Explain why resources might be transferred between them. Would it be satisfactory for clients to multicast all requests to the group of servers as a way of achieving mobility transparency for clients?¹
6. Explain what is (a) a distributed system, and (b) a parallel system. Compare both of them (i.e., what has one and not the other and vice-versa).
un sistema distribuido tiene paralelismo si se puede gestionar, pero el paralelismo no implica que sea distribuido
7. Which kind of database is this according to the distribution of data?

⁴¹ From G. Coulouris et al. Distributed Systems: Concepts and Design, 5th Ed. Addison-Wesley, 2012



8. Which two kinds of schema information contains the Global Conceptual Schema that does not contain the Local Conceptual Schema in the Extended ANSI-SPARC Architecture for DDBMS?
9. In the context of distributed data management, name the four big challenges that need to be carefully considered in the presence of distribution from the tenant/user point of view. I.
II.
III.
IV.
10. Name the three characteristics of fragmentation that make it correct, and for each of them give an example of fragmentation schema where it is not fulfilled.
 - (a)
 - (b)
 - (c)
11. Which is the main problem in having replicas, and which is the innovation introduced by some NOSQL tools to solve it.
 - Problem: EL problema es que las replicas estén sincronizadas, no puede ser que una copia diga A i la otra B. Que es una cosa se modifica se modifique en todas las copias a la vez, tener el mismo resultado. EN sistemas distribuidos es mas difícil, ya que tenemos una red que puede fallar i hacer que las replicas no se puedan sincronizar, en centralizado es mas fácil. Para solucionar esto se hace eventually consistent. Si el sistema deja de recibir modificaciones, este converge a un estado consistente, pero esto casi nunca pasa ya que casi siempre hay modificaciones que hay que propagar.
 - Innovation:
12. How might the clocks in two computers that are linked by a local network be synchronized without reference to an external time source? What factors limit the accuracy of the procedure you have described? How could the clocks in a large number of computers connected by the Internet be synchronized? Discuss the accuracy of that procedure.¹

Hay un axioma que dice que cualquier mensaje llega después de ser enviado, obviamente. Cualquier mensaje que se envía lleva la hora del reloj de la máquina que lo envía. Quien lo recibe, si está por detrás de este reloj, como sabe que esto no puede ser porque la hora que se ha anunciado a tardado un tiempo esta máquina que lo recibe atrasado su reloj. Esto no consigue que todos los relojes vayan a la

vez pero intentan todos sincronizarse. Nunca se resolverá el problema pero como mínimo no estaremos desencaminados. Si el sistema que recibe tiene una hora mas tardía no se puede hacer nada.

13. What is the difference between query cost and query response time ...

a) En centralizados el coste de la query es el numero de lecturas i escrituras en disco, el coste se mide en accesos a disco I/O, tambien es proporcional a los datos de la consulta. El tiempo de respuesta es el numero de accesos a disco por una "constante" de tiempo de acceso a disco i obtenemos el tiempo de ejecución.

b) Tambien tenemos que añadir el impacto del paralelismo, ahora no podemos multiplicar por una constante. Si necesito saber el tiempo de respuesta necesito saber el tiempo en secuencial.

14. Name the two factors that make impossible having linear scalability according to the Universal Scalability Law.

- (a)
- (b)

1.2 Problems

- Briefly explain (a) which fragmentation strategy has been applied for the tables below and whether this fragmentation strategy is (b) complete, (c) disjoint and (d) allows to reconstruct the global relations (if so, (e) indicate the operation).

Global Relations

Kids(kidId, name, address, age)
Toys(toyId, name, price)
Requests(kidId, toyId, willingness)

Note that requests(kidId) is a foreign key to kids(kidId) and similarly, requests(toyId) refers to toys(toyId).

Fragments

K1 = Kids[kidId, name]
K2 = Kids[kidId, address, age]
T1 = Toys(price \geq 150)
T2 = Toys(price < 150)
R1 = Requests \times T1
R2 = Requests \times T2

1. Vertical. Completa ya que todos los atributos estan en los fragmentos, tambien es disjunta ya que aunque tengamos la clave primaria no tenemos opcion, la repetimos para que pueda ser reconstruible la tabla original. En este caso la reconstruimos haciendo una join.

2. Horizontal. Completa si no hay nulos, pero si el precio puede tener valores nulos estos no pueden ser ni > 150 ni < 150 , no seria correcta. Si que es disjunta ya que el igual solo esta en una banda y por lo tanto no se solapan los datos, una fila esta en T1 o esta en T2 pero no en las dos a la vez. Si que es reconstruible si ponemos que no admite nulos el precio.

3. Horizontal derivada, porque deriva de lo que hagamos hecho antes con la tabla de Toys. SI es completa o no depende de T1 y T2, lo que hagamos dicho antes ahora decimos lo mismo, si antes hemos dicho que no tenemos valores nulos y es completa ahora tambien lo es, si antes hemos dicho que no era completa ahora tampoco lo es. Disjunta lo mismo, lo mismo que hagamos dicho de T1 y T2, se cogen las propiedades de antes. Reconstruible igual (con union).

2. You are a customer using an e-commerce based on heavy replication (e.g., Amazon):

- a) Show a database replication strategy (e.g., sketch it) where:

- i. You buy an item, but this item does not appear in your shopping cart.
- ii. You reload the page: the item appears.
⇒ What happened?

Estem en una situació asíncrona, eventually consistent. Cuando tenemos replicas de los datos, la modificación se propaga de forma asíncrona. Lazy primary copy replication. También podría suceder en el caso de lazy distributed replication.

- b) Show a database replication strategy (e.g., sketch it) where:
- i. You delete an item from your command, and add another one: the shopping cart shows both items.

Eventually consistent, concretamente lazy distributed replication.

⇒ What happened? Will the situation change if you reload the page?

3. Consider the following architectures and answer the questions:²

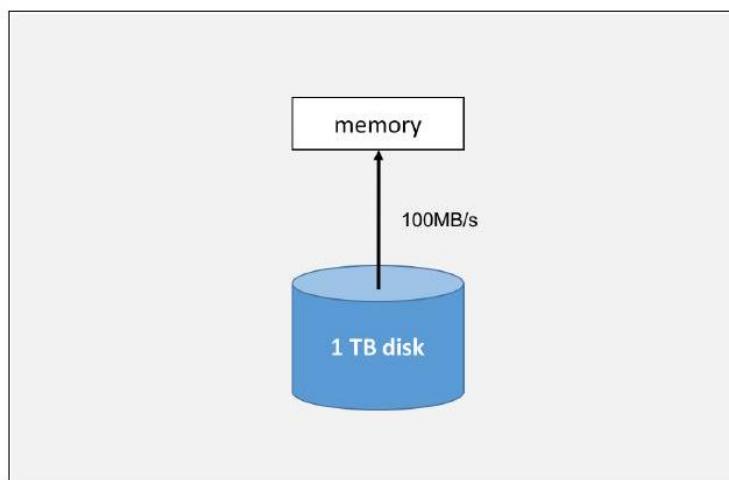


Figure 1: Centralized architecture

Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}$ s (5 millisec.)	At best 100MB/s

- a) How long would it take (i.e., response time) to read 1 TB with sequential access (Figure 1)? (in secs)
- b) How long would a single random access (i.e., reading one tuple, of for example 100 bytes, through an index) take (i.e., response time), assuming we already have the physical address? (in secs)

⁴² From S. Abiteboul et al. Web Data Management. Cambridge Press, 2011.

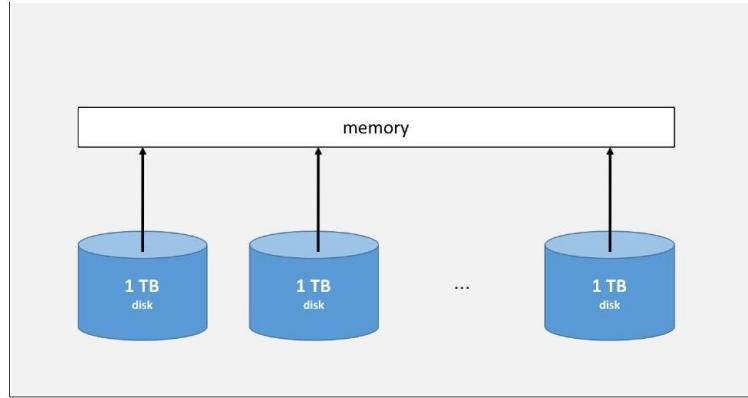


Figure 2: Shared-memory architecture

Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}$ s (5 millisec.).	At best 100MB/s

- c) How long would it take (i.e., response time) to read 1TB with parallel access (Figure 2)? Assume 100 disks (i.e., 100 replicas of the whole data) on the same machine with shared-memory and infinite CPU capacity.
- d) How long would a single random access (i.e., reading one tuple, of 100 bytes, through an index) take (i.e., response time), assuming we already have the physical address? (in secs)

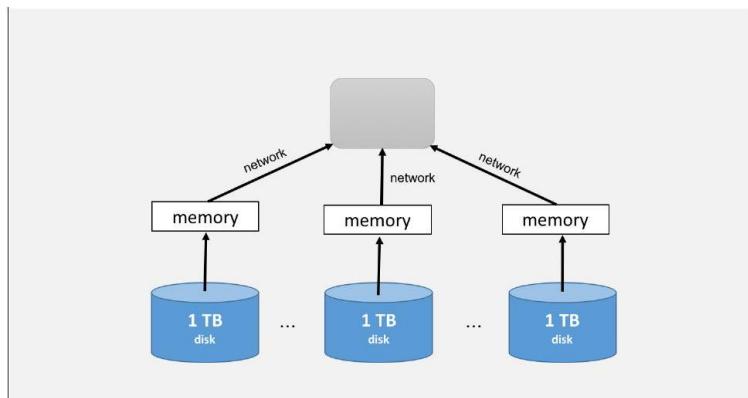


Figure 3: Shared-nothing architecture

Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}$ s (5 millisec.)	At best 100MB/s
LAN	$\approx 1 - 2 \times 10^{-3}$ s (1-2 millisec.)	≈ 1 GB/s(single rack) ≈ 10 MB/s(switched)
Internet	Highly variable (typically 10 – 100 ms)	Highly variable (typically a few MB/s, e.g., 10MB/s)

Note 1: It is approx. one order of magnitude faster to exchange main memory data between 2 machines in a data center, than to read on the disk.

Note 2: Exchanging through the Internet is slow and unreliable with respect to LANs.

- e) How long would it take (i.e., response time) to read 1TB with distributed access (Figure 3)? Assume 100 shared-nothing machines (with all data replicated in each of

them) in a star-shape LAN in a single rack where all data is sent to the center of the star in only one network hop.

f) How long would it take (i.e., response time) to read 1TB with distributed access (Figure 3)? Assume 100 shared-nothing machines (with all data replicated in each of them) in a star-shape cluster of machines connected through the Internet where all data is sent to the center of the star in only one network hop.

g) How long would a single random access (i.e., reading one tuple, of 100 bytes, through an index) take (i.e., response time) in the case of a LAN, assuming we already have the physical address? (in secs)

h) How long would a single random access (i.e., reading one tuple, of 100 bytes, through an index) take (i.e., response time) in the case of an Internet connection, assuming we already have the physical address? (in secs)

a) Response time: $1TB/100MB/s = 10000s$, Latency Time: $5 \cdot 10^{-3}$ (Latency time is insignificant)

b) Response time: $100B/100MB/s = 1/10^6$, Latency Time: $5 \cdot 10^{-3}$ (Response time is insignificant)

c) Response time: $(1TB/100MB/s)/100 = 1000s$, Latency Time: $(5 \cdot 10^{-3})$ No se multiplica por 100 el tiempo de latencia porque como se ejecuta en paralelo no se suma.

d) Response time: $100B/100MB/s = 1/10^6$, Latency Time: $(5 \cdot 10^{-3})$

e) No se añade un tiempo de transferencia a través de la red porque la inicialización de transferencia se hace simultáneamente en la red y en el disco, por lo tanto se tiene en cuenta solo la que tarda más tiempo, que es el caso de la del disco ($100MB/s < 1GB/s$).

	SRQ	RANDOM
a) Centralized	$\frac{1TB}{100MB} + 5 * 10^{-3}$	$\frac{100}{100MB} + 5 * 10^{-3}$
b) Shared-memory	$\frac{1TB/100}{100MB} + 5 * 10^{-3}$	$10^{-6} + 5 * 10^{-3}$
c) Shared-nothing (LAN) with single rack	$\frac{1TB/100}{100MB} + 2 * 10^{-3} + 5 * 10^{-3}$	$10^{-6} + 5 * 10^{-3} + 2 * 10^{-3}$
d) Shared-nothing (INTERNET)	$\frac{1TB/100}{10MB} + 10^{-2} + 5 * 10^{-3}$	$\frac{100}{10MB} + 10^{-2} + 5 * 10^{-3}$

Entre una centralizada y una shared memory, cuantas mas máquinas pongo, vamos mas rápido (en la vida real no es un crecimiento lineal). Poner mas máquinas cuando lo que intento acceder es pequeño no sale a cuenta, no tiene un impacto significativo.

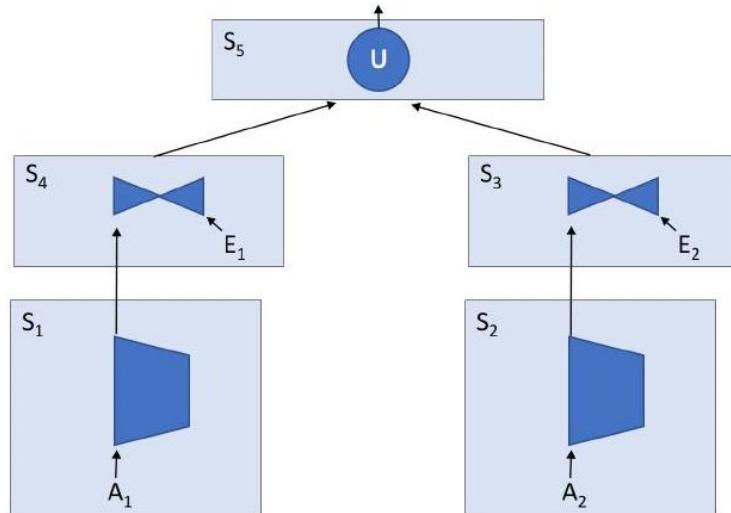
Share memory or share nothing con LAN dan prácticamente los mismos resultados. El beneficio es que es más económico y más escalable (share nothing). En internet, la latencia y ancho de banda son muy variables y se puede llegar a perder los beneficios.

³Hacer *joins* de tablas sin tener que crear nuevos archivos

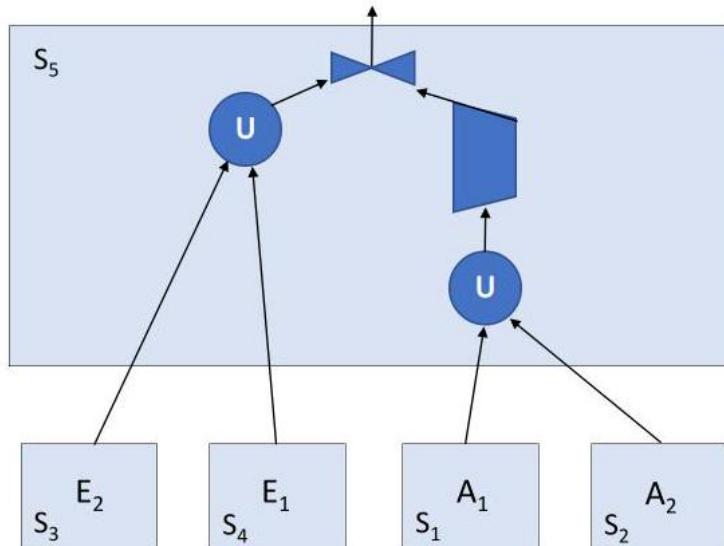
⁴Lenguaje declarativo: dices lo que quieras y encuentra la mejor manera de hacerlo, optimización de consultas

4. What are the main differences between these two distributed access plans? Under which assumptions is one or the other better?

```
SELECT *
FROM employee e, assignedTo a
WHERE e.#emp = a.#emp AND a.responsability = 'manager'
```



Access Plan A



Access Plan B

Figure 4: Distributed Access Plans

Employee(#emp, empName, degree) S4: E1 = Employee(#emp ≤ 'E3') S3: E2 = Employee(#emp > 'E3')
AssignedTo(#emp, #proj, responsability, fullTime) FK: #emp references Employee S1: A1 = AssignedTo(#emp ≤ 'E3') S2: A2 = AssignedTo(#emp > 'E3')

Depende del paralelismo que queramos: A es mas paralelisable.

Depende del tiempo de latencia y transferencia de datos: En el A transfiere menos datos de la tabla de 'Assigned' porque hace la seleccion antes. En el peor caso sera igual al de B en el caso de que 'manager' tenga pocas ocurrencias.

La de la izquierda al tener empleados i proj como clave primaria un unico empleado sera asignado a potencialmente muchas filas y por lo tanto un mismo empleado se pasara muchas veces. Si los empleados estan asignados a muchos proyectos sera mejor la opcion B ya que se transferiran menos datos.

La de la izquierda es mas query shipping, enviar trozos de la consulta a donde estan los datos. La B es data shipping. En el A (bushy tree), los dos trozos de la join se podran hacer de forma paralela.

Afecta muchas cosas, el paralelismo, la transferencia de datos, la carga de las maquinas. En sistemas centralizados la optimizacion de consultas esta mas o menos resuelta. En los distribuidos no, es mas facil que se equivoquen porque añadimos paralelismo, las cargas que tienen i los costes de transferencia de datos que no sabemos exactamente. Cuadro añadimos todo es mucho mas complicado.

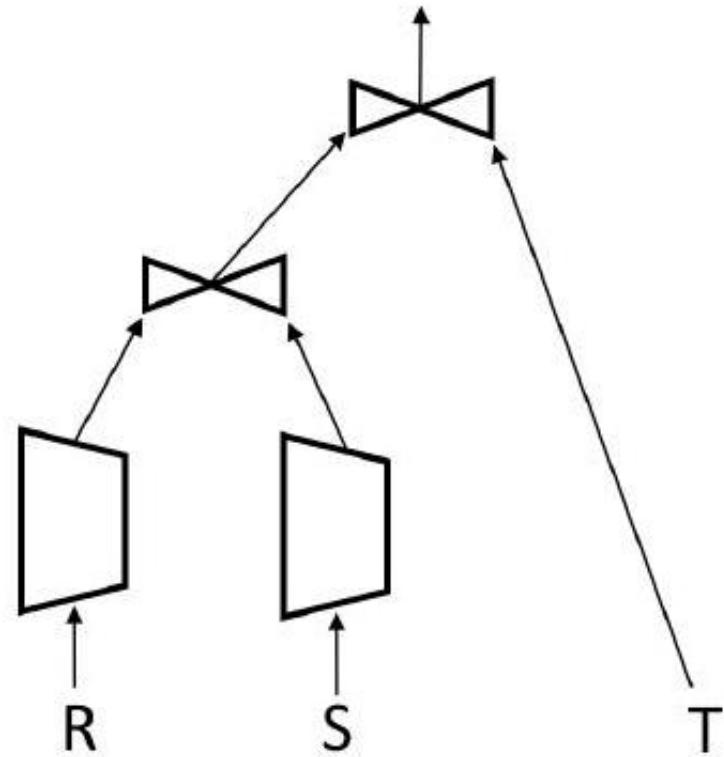
s

5. Compute the fragment query (data location stage) for the database setting and query below and find in how many ways we can assign the operations to the different sites.

⇒ Database setting:

- A distributed database with 5 sites (i.e., database nodes): S_1, S_2, S_3, S_4 and S_5 .
- 3 relations in the database R, S and T
- Each relation is horizontally fragmented in two fragments (we refer to them by the name of the relation and a subindex, for example: R_1, R_2). You can consider them to be correct (i.e., complete, disjoint and reconstructible).
- Each fragment is replicated at all sites.
- We have the following query: $Q_1 = \sigma(R) \bowtie \sigma(S) \bowtie T$

⇒ Process tree of the query:



6. Consider a left-deep process tree corresponding to a query, where each internal node is a join, and every leaf a data source (e.g., relational table). Knowing that the tree contains 9 nodes (including leaves), the system has as much parallelism capacity as needed to run all the joins in pipelining mode (no other kind of parallelism is available), which is the occupancy if the overall cost of the serial query is 4 seconds? Explicit any assumption you need to make.

2 Distributed Processing Frameworks

2.1 MapReduce in use

2.1.1 Theoretical questions

1. Classify a MapReduce system as either query-shipping (i.e., the evaluation of the query is delegated to the site where the data are stored), data-shipping (i.e., the data are moved from the stored site to the site executing the query) or hybrid (i.e., both query- and data-shipping). Clearly justify your answer.

2.1.2 Problems

1. Consider the following implementation of the Cartesian Product with MapReduce (\oplus stands for concatenation) and answer the questions accordingly.

$$T \times S \implies \begin{cases} \text{map(keyk, value v) } \mapsto \\ \quad [(h_T(k) \bmod D, k \oplus v)] & \text{if input } (k \oplus v) = T \\ \quad [(0, k \oplus v, \dots, (D-1, k \oplus v)] & \text{if input } (k \oplus v) = S \\ \text{reduce(key ik, vset ivs) } \mapsto \\ \quad [\text{crossproduct } (T_{ik}, S) | \\ \quad T_{ik} = \{iv \mid iv \in \text{ivs} \wedge \text{input}(iv)T\}, \\ \quad S = \{iv \mid iv \in \text{ivs} \wedge \text{input}(iv)S\}] \end{cases}$$

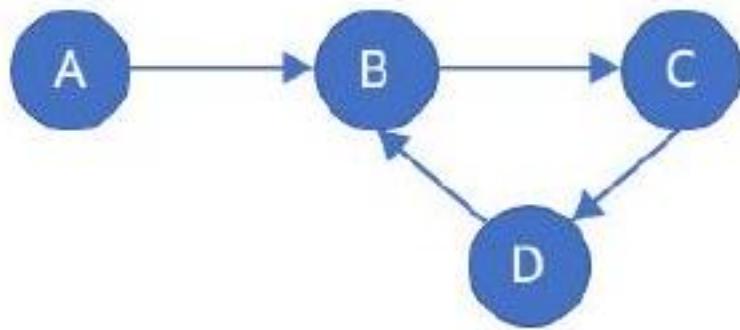
- (a) Which is the relationship of D with parallelism and scalability, if any?
(b) Which is the optimal value for D ?
2. Provide the MapReduce pseudo-code implementation of the following Relational operators. You can use " \oplus " symbol for concatenation, " $\text{proj}_{a_{i_1} \dots a_{i_n}}(t)$ " to get attributes " $a_{i_1} \dots a_{i_n}$ " in t , " $\text{crossproduct}(S_1, S_2)$ " to perform the cross product of two sets of rows, and assume the "key" parameter contains the PK of the table and the "value" one all the others.
- Aggregation ($\gamma_{A,f(B)}(T)$)
 - Selection ($\sigma_P(T)$)
 - Join ($T \bowtie_{T.A=S.B} S$)
 - Union ($T \cup S$)
 - Difference ($T \setminus S$)
 - Intersection ($T \cap S$)
3. In relational algebra, the antijoin operator (\triangleright) is defined as the complement of the semijoin on the primary keys (PKs). Formally, assuming A and B are the PKs of R and S , respectively, then $R \triangleright S = R \setminus R \ltimes_{A=B} S$. Provide the MapReduce pseudo-code implementation of the antijoin operator. Assume the existence of the operator \oplus to concatenate strings, $\text{proj}_{att}(s)$ to project attribute att from the tuple s , and $\text{input}(s)$ to decide the origin (i.e., R or S) from a tuple s .
4. Modify the following MapReduce code in Python so that given a file with key-value pairs representing the edges of a graph ³ (i.e., the key is the label of the first node and the value is the label of the second), it identifies all pairs of nodes connected by a path of length two (if two nodes are connected multiple times, they will be repeated in the output). Do not need to optimize it in any way, just make it work as expected.

```

def map(k, v):
    return [(k, (k, v))]

def reduce(k, lv):
    outgoing = []
    incoming = []
    for v in lv:
        if k == v[0]:
            outgoing.append(v[1])
        elif k == v[1]:
            incoming.append(v[0])
    retValue = []
    for o in outgoing:
        for i in incoming:
            retValue.append(i + "-2->" + o)
    return retValue

```



Expected output

A -2- > C

D -2- > C

B -2- > D

C -2- > B

2.2 Spark in use

2.2.1 Problems

1. Consider a file (wines.txt) containing the following data:

```
wines.txt
type_1,2.064254
type_3,2.925376
type_2,2.683955
type_1,2.991452
type_2,2.861996
type_1,2.727688
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve the minimum value per type. Do not use SQL and minimize the use of other Python libraries or code. Save the results in output.txt.

2. Consider two files containing the following data:

```
Employees.txt
EMP1;CARME;400000;MATARO;DPT1
EMP2;EUGENIA;350000;TOLEDO;DPT2
EMP3;JOSEP;250000;SITGES;DPT3
EMP4;RICARDO;250000;MADRID;DPT4
EMP5;EULALIA;150000;BARCELONA;DPT5
EMP6;MIQUEL;125000;BADALONA;DPT5
EMP7;MARIA;175000;MADRID;DPT6
EMP8;ESTEBAN;150000;MADRID;DPT6
Departments.txt
DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID
DPT5;VENDES;1;MUNTANER;BARCELONA
DPT6;VENDES;1;CASTELLANA;MADRID
```

⁴³ Without reflexive edges like (A,A).

Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve for each employee his/her department information. Do not use SQL and minimize the use of other Python libraries or code. Save the results in output.txt.

3. Consider an error log file (log.txt) like the one bellow:

```
log.txt
20150323;0833;ERROR;Oracle
20150323;0835;WARNING;MySQL
20150323;0839;WARNING;MySQL
20150323;0900;WARNING;Oracle
20150323;0905;ERROR;MySQL
20150323;1013;OK;Oracle
20150323;1014;OK;MySQL
20150323;1055;ERROR;Oracle
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve the lines corresponding to both errors and warnings, but adding Important: at the beginning of those of errors (i.e., only errors). Do not use SQL and minimize the use of other Python libraries or code. Save the results in output.txt.

4. Assume that "spark" variable is a Spark session and the dataset . csv contains the two columns "Color" and "Radius". Clearly identify the problems you find in the following Spark code and propose some fix to obtain the expected result.

```
df0= spark.read.csv("dataset.csv", \
                     header='true', \
                     inferSchema='true', \
                     sep=',')
df1 = df0.select("Color", "Radius")
df2 = df1.withColumn("Pi", 3.141592)
df3 = df2.withColumn("Area", df2.Radius * df2.Radius * df2.Pi)
df4 = df3.groupBy("Color").max("Area")
df5 = df4.sort("Color")
```

5. Given two files containing the following kinds of data:

Employees.txt with fields: EmployeeID; EmployeeName; YearlySalary; CityOfResidence; SiteOfWork
EMP4;RICARDO;250000;MADRID;DPT4

```
EMP5;EULALIA;150000;BARCELONA;DPT5
EMP6;MIQUEL;125000;BADALONA;DPT5
EMP7;MARIA;175000;MADRID;DPT6
EMP8;ESTEBAN;150000;MADRID;DPT6
```

Departments.txt with fields: SiteID; DepartmentName; StreetNumber; StreetName;
City DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA

```
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID
```

Consider the following PySpark code and answer the questions bellow.

```
source1 = spark.read.format('csv').load('employees.txt', header='false', inferSchema='true', sep=";")
source2 = spark.read.format("csv").load('departments.txt', header='false', inferSchema='true', sep=";")
A = source1.toDF('eID','eName','eSalary','eCity','eDpt','eProj")
B = source2.toDF('dID','dArea','dNumber','dStreet','dCity')
C = A.select(A.eCity.alias("city"))
D = B.select( "dArea")
E = D.crossJoin(C)
```

```
F = B.select("dArea",B.dCity.alias("city"))
G = E · subtract(F)
H = G · select( "dArea")
result = D. subtract(H)
```

(a) State in natural language the corresponding query it would answer?

(b) Clearly indicate any mistake or improvement you can fix/make in the code? For each of them give (1) the line number, (2) pseudo-code to implement the fix, and (3) brief rationale.

6. Given two files containing the following kinds of data:

Employees.txt with fields: EmployeeID; EmployeeName; YearlySalary; CityOfResidence; SiteOfWork
EMP1;RICARDO;250000;MADRID;DPT1

```
EMP2;EULALIA;150000;BARCELONA;DPT2
EMP3;MIQUEL;125000;BADALONA;DPT3
EMP4;MARIA;175000;MADRID;DPT4
EMP5;ESTEBAN;150000;MADRID;DPT3
```

Departments.txt with fields: SiteID; DepartmentName; StreetNumber; StreetName; City

```
DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID
```

Give a sequence of Spark operations in pseudo-code (resembling PySpark) to obtain for each city where employees that work in a site of a department in Barcelona live, the sum of the salaries of those employees. The result for the exemplary data would be:

```
MADRID; 400000
BADALONA; 125000
```

7. Consider three files containing the following kinds of data:

Employees.txt
EMP1,CARME,400000,MATARO,DEPT1,PROJ1
EMP2,EULALIA,150000,BARCELONA,DEPT2,PROJ1
EMP3,MIQUEL,125000,BADALONA,DEPT1,PROJ3

Projects.txt
PROJ1,IBDTEL,TV, 1000000
PROJ2,IBDVID,VIDEO,500000
PROJ3,IBDTEF,TELEPHONE, 200000
PROJ4,IBDCOM,COMMUNICATIONS, 2000000

Departments.txt
DEPT1,MANAGEMENT,10,PAU CLARIS,BARCELONA
DEPT2,MANAGEMENT,8,RIOS ROSAS,MADRID
DEPT4,MARKETING,3,RIOS ROSAS,MADRID

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you would need to obtain the departments with all employees assigned to the same project. The result must include department number. Save the results in output.txt. In the previous example, the result should be DEPT2 and DEPT4.

8. Consider two files containing the following kinds of data:

Employees.txt
EMP4;RICARDO;250000;MADRID;DPT4
EMP5;EULALIA;150000;BARCELONA;DPT5
EMP6;MIQUEL;125000;BADALONA;DPT5
EMP7;MARIA;175000;MADRID;DPT6
EMP8;ESTEBAN;150000;MADRID;DPT6

Departments.txt

DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
 DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
 DPT3;MARKETING;1;PAU CLARIS;BARCELONA
 DPT4;MARKETING;3;RIOS ROSAS;MADRID

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you'd need to retrieve the list of department IDs for those departments with workers from all cities where there are employees. Save the results in output.txt.

- Consider three files relating to a bibliographic database: author.csv relates authors with papers (you may assume that author names are unique, that authors have one or more papers, and that papers have one or more authors); title.csv gives the title of a paper (you may assume a paper has one title, but one title may be shared by many papers); and citation.csv indicates which papers cite which other papers (you may assume that each paper cites at least one other paper, that a paper may be cited zero or more times, and that a paper cannot cite itself).

author.csv	
AUTHOR	PAPERID
...	...
C. Gutierrez	GP2014
C. Gutierrez	AGP2013
C. Gutierrez	GZ2011
...	...
J. Perez	GP2014
J. Perez	AGP2013
J. Perez	P2017
...	...
R. Angles	AGP2013
R. Angles	AKK2016
...	...

title.csv	
PAPERID	TITLE
...	...
GP2014	Semantics of SPARQL
AGP2013	Deduction for RDF
GZ2011	Graph databases
...	...

citation.csv	
PAPER	CITES
...	...
GP2014	AGP2013
AGP2013	GZ2011
P2017	AKK2016
...	...

The headers are shown for illustration here. They do not need to be considered.

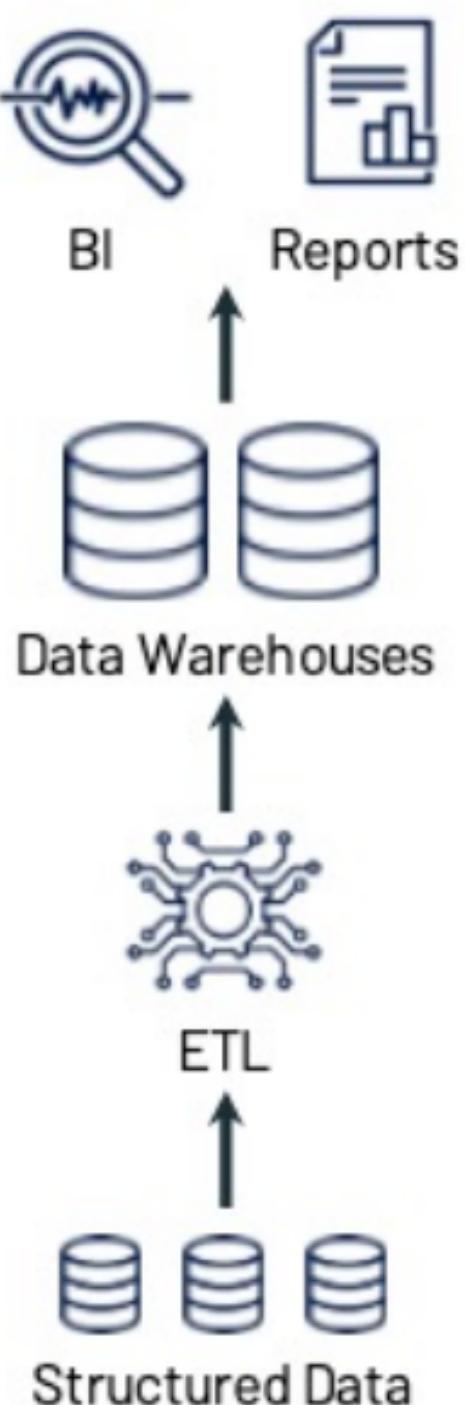
The count of self-citations for an author A , denoted $\text{self}(A)$, is defined as the number of citation pairs (P_1, P_2) where A is an author of both. The count of citations given by an author A , denoted $\text{give}(A)$, is the count of citation pairs (P_1, P_2) such that A is an author of P_1 . The count of citations received by A , denoted $\text{receive}(A)$, is the count of citation pairs (P_1, P_2) where A is an author of P_2 . The ratio of selfcitations to all citations given and received are then defined, respectively, as $\frac{\text{sel}(A)}{\text{give}(A)}$ and $\frac{\text{self}(A)}{\text{receive}(A)}$. In

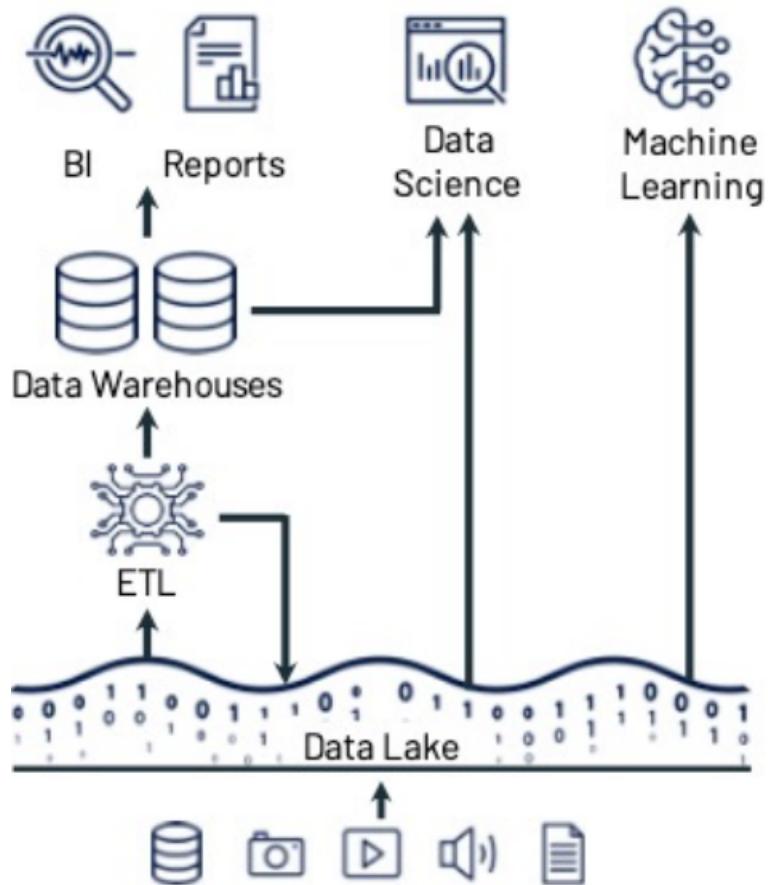
case that receive (A) = 0, you should omit the author A from the results (note that give (A) cannot be 0 as an author must have at least one paper and a paper must have at least one citation). We provide an example output for the input data:

Author	SelfGiveratio	SelfReceiveratio
C. Gutierrez	1.000	1.000
J. Perez	0.333	1.000
R. Angles	0.000	0.000
...	...	

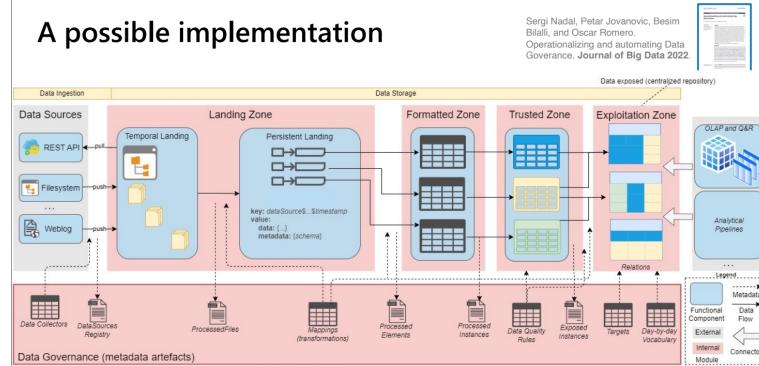
Headers are only shown for illustration. We will use Apache Spark to perform the analysis and compute the output. You should not assume any ordering of the input files. You do not need to order the output file in any particular way.

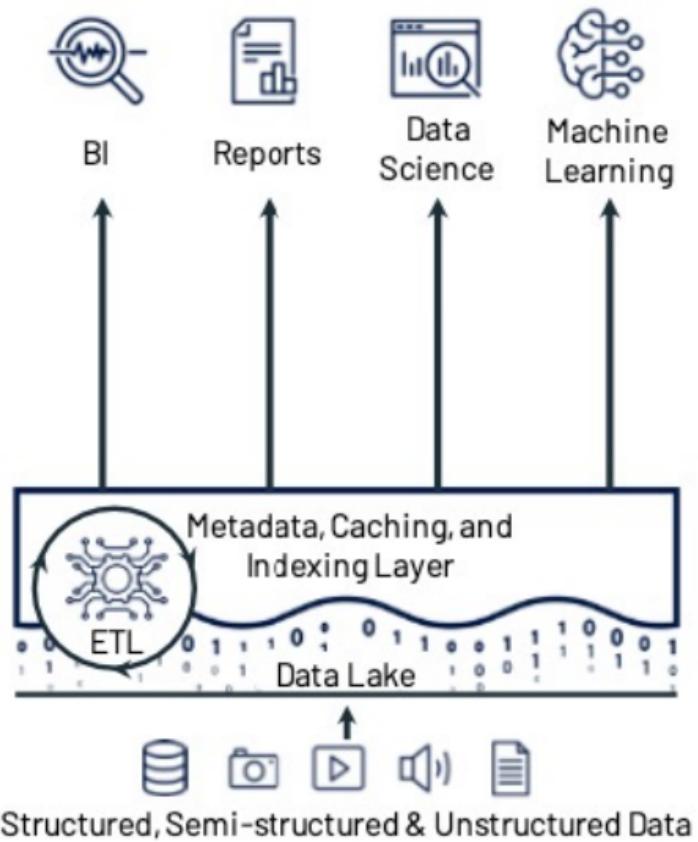
Given this input and desired output, design a Spark process to complete the required processing. In particular, you should draw the high-level DAG of operations that the Spark process will perform, detailing the sequence of transformations and actions. You should briefly describe what each step does, clearly indicating which steps are transformations and which are actions. You should also indicate which RDDs are virtual and which will be materialized. You should use caching if appropriate. You should provide details on any functions passed as arguments to the transformations/actions you use.



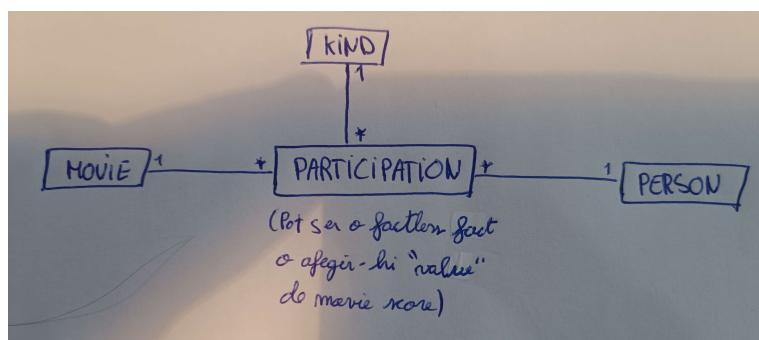


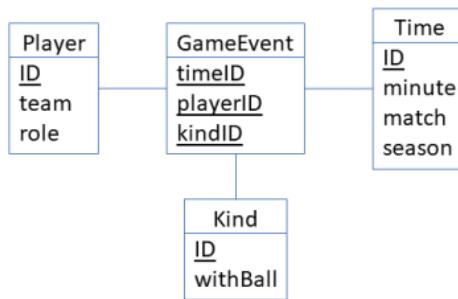
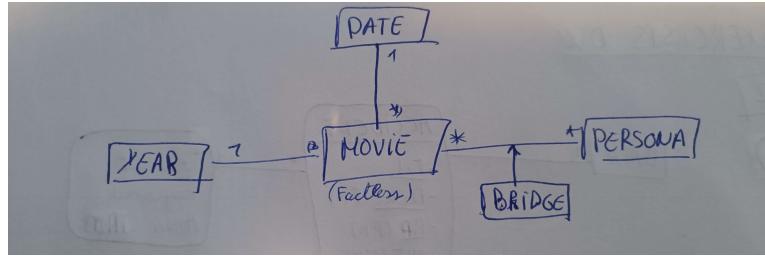
A possible implementation





	Software	AI (Software + Data)
Goal	Functional correctness	Optimization of a metric, e.g., minimize loss
Quality	Depends on code	Depends on data, code, model architecture, hyperparameters, random seeds, ...
Outcome	Works deterministically	Changes due to data drift
People	Software Engineers	Software Engineers, Data Scientists, Data Engineers, Research Scientists, ML engineers
Tooling	Usually standardized within a dev team Established/hardened over decades	Often heterogeneous even within teams Few established standards and in constant change due to open source innovation
AI depends on Code AND Data		AI requires many different roles to get involved





Dealer A

Cars(serialNo, model, color, autoTrans, navi)

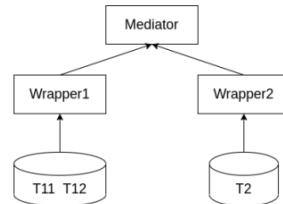
Dealer B

Autos(serial, model, color)

Options(serial, option), serial is FK to Autos

```

SELECT T11.a, T12.b, T2.c
FROM T11, T12, T2
WHERE T11.link=T12.link
AND T12.link=T2.link
AND T11.slicer='X';
    
```



```

SELECT DISTINCT f.day, f.aircraft
FROM Aircrafts a
JOIN Flights f ON a.aircraft = f.aircraft
JOIN Incidents i ON f.aircraft = i.aircraft AND f.day=i.day
WHERE a.capacity >200 AND i.ATAcode=21 AND f.origin='BCN';
    
```

source	title	destination	duration	type	price
Viajes El Corte Ingles	The best of East Cost, NYC and Washington	New York, Washington	1 week	City trips	750€
Viajes El Corte Ingles	Classic China	Beijing, Xi'ian	2 weeks	City trips, Special trips	1855€
Viajes El Corte Ingles	Bohemia of Prague	Prague, Karlovy Vary	1 week	City trips	700€
Viajes El Corte Ingles	Imperial cities	Budapest, Vienna	1 week	City trips	785€
Halcon Viajes	Circuit Italy on the ship, all inclusive	Venice, Rome, Naples, Palermo	15 days	Cruise, Special trips	2000€
Halcon Viajes	Holly Land and Jordan	Jerusalem, Amman	16 days	Special trips, Holiday specials	1800€
Halcon Viajes	New York and Maya's cost	New York, Cancun	15 days	Special trips, Holiday specials	2000€
Halcon Viajes	New York, New Year	New York, Washington	5 days	City trips, Holiday specials	600€
Halcon Viajes	Budapest	Budapest	3 days	City trips	350€
TUI travel	Journey to Jordan 1	Venice, Rome, Naples, Suez, Aqaba	21 days	Cruise, Special trips	772£
TUI travel	New York Holidays	New York	6 days	City trips	593£
TUI travel	Europe's finest	Budapest, Vienna, Prague	7 days	City trips	600£
TUI travel	Spicy Mexico	Maya's cost, Cancun	10 days	Special trips	On demand
TUI travel	Pearls of Orient	Shanghai, Beijing, Xi'ian	19 days	Special trips	On demand
TUI travel	Tanzania	Dar es Salaam, Zanzibar	20 days	Special trips	1500£

Provider1 schema:

```

CREATE TABLE P1_Books (
Id INTEGER,
Author VARCHAR(30),
Year CHAR(4),
Genre VARCHAR(15),
PRIMARY KEY (Id))

CREATE TABLE P1_Nobels (
BookId INTEGER,
Year CHAR(4),
PRIMARY KEY (BookId),
FOREIGN KEY (BookId)
    REFERENCES P1_Books (Id))

```

Books Dimension:

```

CREATE TABLE BookDim (
ID INTEGER,
Author VARCHAR(30),
EditionYear INTEGER,
NobelPrize BOOLEAN,
Genre VARCHAR(15),
PRIMARY KEY (ID))

```

Provider2 schema:

```

CREATE TABLE P2_Books (
Id INTEGER,
Author VARCHAR(30),
Year INTEGER,
Nobel BOOLEAN,
Genre VARCHAR(15),
PRIMARY KEY (Id),
CHECK (Year > 0 AND Year <=9999))

```