

Exploración y Curación de Datos

Clase 2
Ariel Wolfmann
Mayo 2025

Ingesta de Datos

Formatos de Datos

- **Tabulares:** como una planilla, con filas y columnas.
 - Formatos de Archivos: CSV, TSV, XLS
 - Estructura de Datos: Dataframe
 - Son fáciles de manipular y analizar utilizando herramientas como Pandas en Python
- **Jerárquicos:** con valores anidados dentro de otros valores.
 - Formatos de Archivos: JSON, XML
 - Estructura de Datos: Lista de Objetos
 - utilizado para el intercambio de datos en aplicaciones web y servicios web.
- **Crudos:** sin estructura específica
 - Formato de Archivos: TXT
 - Estructura de Datos: String
 - Requieren procesamiento adicional antes de poder ser analizados y utilizados

Formatos: Tabulares vs Jerárquicos vs Crudos

	Sepal.Length	Sepal.Width
1	5.1	3.5
2	4.9	3.0
3	4.7	3.2

Tabular Data

```
↳ Name: Robin
  ↳ Species: Hedgehog
  ↳ Owner: Justice Smith
    ↳ Address: 1234 Main St.
    ↳ Phone #: 123-4567
↳ Name: Bunny
  ↳ Species: Rabbit
  ↳ Breed: Holland Lop
  ↳ Color: Brown and white
```

Hierarchical Data

石室诗士施氏，嗜狮，誓食十狮。氏时时适市视狮。十时，适十狮适市。是时，适施氏适市。氏视是十狮，恃矢势，使是十狮逝世。氏拾是十狮尸，适石室。石室湿，氏使侍拭石室。石室拭，氏始试食是十狮尸。食时，始识是十狮，实十石狮尸。试释是事。

Raw Text

CSV - Comma Separated Values

- Archivos de texto delimitado que usa coma para separar valores.
- Cada línea es un registro con uno o más campos.
- No está formalmente especificado!

```
latitud,longitud,Nombre
-54.832543,-68.3712885,SAN SEBASTIAN  ( USHUAIA )
-54.8249379,-68.3258626,AERO PUBLICO DE USHUAIA
-54.8096728,-68.3114748,PUERTO USHUAIA (PREFECTURA)
-54.8019121,-68.3029511,PUERTO USHUAIA
-51.6896359,-72.2993574,PASO LAURITA CASAS VIEJAS
-51.5866042,-72.3649779,PASO DOROTEA
-51.2544488,-72.2652242,PASO RIO DON GUILLERMO
-53.3229179,-68.6063227,PASO SAN SEBASTIAN
-53.78438,-67.7173342,TERMINAL RIO GRANDE
```

Lectura de CSV

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

```
In [1]: data = 'col1,col2,col3\na,b,1\na,b,2\nc,d,3'
```

```
In [2]: pd.read_csv(StringIO(data))
```

Out[2]:

	col1	col2	col3
0	a	b	1
1	a	b	2
2	c	d	3

Lectura de JSON

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_json.html

```
In: data = [{ 'state': 'Florida',
              'shortname': 'FL',
              'info': {
                  'governor': 'Rick Scott'
              },
              'counties': [{ 'name': 'Dade', 'population': 12345},
                           { 'name': 'Broward', 'population': 40000},
                           { 'name': 'Palm Beach', 'population': 60000} ]],
              { 'state': 'Ohio',
                'shortname': 'OH',
                'info': {
                    'governor': 'John Kasich'
                },
                'counties': [{ 'name': 'Summit', 'population': 1234},
                             { 'name': 'Cuyahoga', 'population': 1337} ]}]
```

```
In: from pandas.io.json import json_normalize
```

```
In: json_normalize(data, 'counties', ['state', 'shortname',
                                       ['info', 'governor']])
```

Out:

	name	population	state	shortname	info.governor
0	Dade	12345	Florida	FL	Rick Scott
1	Broward	40000	Florida	FL	Rick Scott
2	Palm Beach	60000	Florida	FL	Rick Scott
3	Summit	1234	Ohio	OH	John Kasich
4	Cuyahoga	1337	Ohio	OH	John Kasich

Bases de Datos

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Clasificadas según

- Variabilidad de los datos: estáticas o dinámicas
- Contenidos: bibliográficas, texto completo, directorios, etc
- Modelo de administración: [no] transaccionales, [no] relacionales, distribuidas, etc

Bases de Datos Relacionales

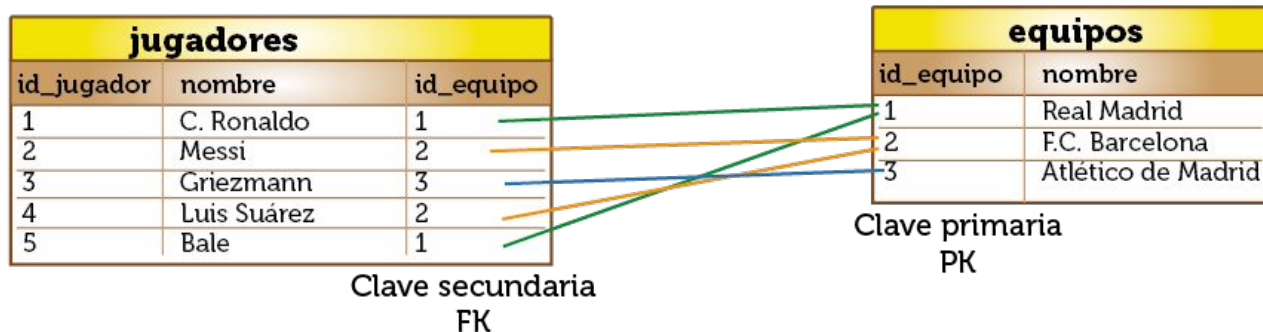
- Basada en tuplas (listas ordenadas de elementos).
- Se compone de varias tablas (relaciones)
- Cada tabla es un conjunto de campos (columnas) y registros (filas)
- Se establecen relaciones entre tablas usando claves primarias
- Permiten combinar columnas de una o más tablas (JOIN)
- Utilizan el lenguaje SQL (Structured Query Language) para consultar y manipular los datos

Modelo Relacional: Ejemplo

Modelo relacional



Ejemplo de datos



Bases de Datos No Relacionales

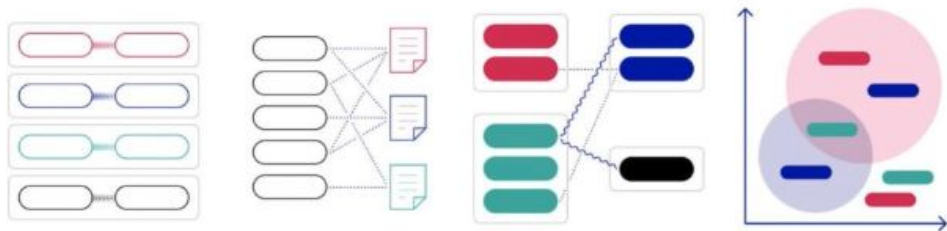
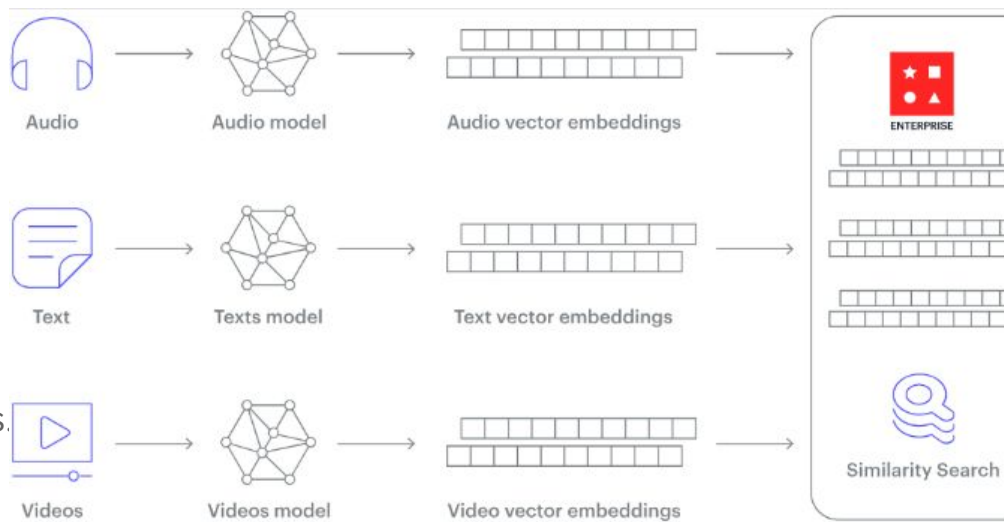
- Conocidas como NoSQL.
- Manejan datos semi estructurados
- No permiten JOIN, cruzar tablas
- Varios tipos:
 - Documentales ej MongoDB
 - Grafos Neo4j
 - Vectorizadas, representan embeddings.

Key-Value

Document

Graph

Vector



Exploración de Datos

Demo notebook

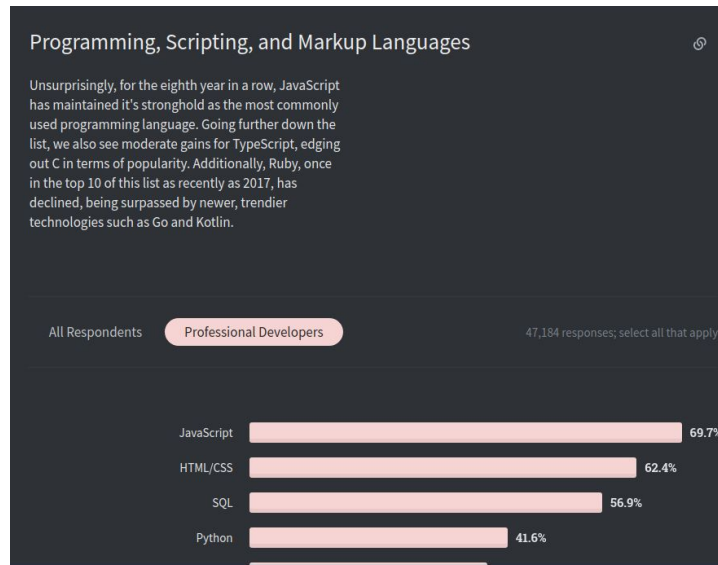
01 Exploracion.ipynb

Consultas SQL

Por qué SQL?

- 🔍 Ideal para explorar grandes volúmenes de datos directamente desde la fuente.
 - El estándar en la industria para consultar bases de datos.
- 🧰 Herramienta esencial para cualquier rol en ciencia de datos.
- 3ro en encuesta de popularidad de Stack Overflow 2020
- 🤝 Permite colaborar con equipos que usan otras tecnologías.

“Si no sabés SQL, probablemente no estés viendo todo el iceberg del problema.”



[Why You Can't Do All of Your Data Engineering with SQL](https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers)
[A Beginner's Guide to Data Engineering — Part I](https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers)

<https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers>

Por qué SQLite?

- Es muy pequeña y su instalación es sencilla
- Es ideal para desarrollo: no tiene dependencias externas
- Es ideal para aprender SQL
- Es la DB más desplegada (celulares, Skype, iTunes, TVs, autos, etc)
- Tutorial: <https://www.sqlitetutorial.net/tryit/>



Consultas - Sintaxis General

SELECT DISTINCT *column_list* -- selecciona una lista de columnas opcionalmente, filas únicas

FROM *table_list* -- de una lista de tablas (*table_list*)

JOIN *table* **ON** *join_condition* -- combinada con la tabla (*table*) si se cumple la condición (*join_condition*)

WHERE *row_filter* -- filtrando solo las que cumplen un criterio (*row_filter*)

ORDER BY *column* -- ordenadas por los valores de cierta columna (*column*)

LIMIT *count* **OFFSET** *offset* -- limitada a un número de registros (*count*) a partir del registro (*offset*)

GROUP BY *column* -- agrupadas por el valor de la columna (*column*)

HAVING *group_filter*; -- solo aquellos grupos que cumplen la condición (*group_filter*)

Consultas - SELECT / FROM

Para realizar una consulta, usamos SELECT y debemos indicar

- la tabla de la cual obtener los registros usando la cláusula FROM
- la lista de columnas separada por comas

Ejemplo: Canciones y su compositor

```
SELECT name, composer  
FROM tracks
```

tracks
* TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Consultas - ORDER BY ordenamiento

Para ordenar los resultados de una consulta, agregamos ORDER BY

- ASC para ordenamiento ascendente
- DESC para ordenamiento descendente

Ejemplo: Canciones y su compositor

```
SELECT name, composer
FROM tracks
ORDER BY
    composer ASC
    name DESC;
```

tracks
* TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Consultas - DISTINCT remoción de duplicados

Para filtrar los resultados de una consulta, agregamos la palabra clave DISTINCT

Ejemplo: Ciudades y países de los clientes

```
SELECT DISTINCT city, country
FROM customers
ORDER BY
    country ASC,
    city ASC;
```

customers
* CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Consultas - WHERE filtrado de resultados

Para filtrar los resultados que cumplen un determinado criterio se usa la cláusula WHERE con diferentes operadores

- =, <>, <, >, <=, >=
- IN (...), BETWEEN x AND y
- LIKE
-

Ejemplo: Canciones con compositor de nombre SMITH

```
SELECT name, composer
FROM tracks
WHERE composer LIKE '%SMITH%'
```

tracks
* TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Consultas - LIMIT/OFFSET limitar cantidad

Para restringir la cantidad de resultados en una consulta, se usa la cláusula LIMIT con un número. Adicionalmente, se puede incluir la cláusula OFFSET para indicar a partir de qué registro se obtienen los resultados.

Ejemplo: Canciones 31 a 40 ordenadas por título.

```
SELECT name
FROM tracks
ORDER BY name
LIMIT 10 OFFSET 30
```

tracks
* TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Consultas - GROUP BY agrupar resultados

Para agrupar los resultados, usamos la cláusula GROUP BY. Retorna un resumen de las filas agrupadas por el valor de una o más columnas. En cada grupo podemos aplicar una función de agregación: MAX, MIN, SUM, COUNT, AVG.

Ejemplo: Los 10 compositores más prolíficos y cantidad de canciones

```
SELECT composer, count(trackid)
FROM tracks
WHERE composer <> '' -- filtrar las que no tienen compositor
GROUP BY composer
ORDER BY count(trackid) desc
LIMIT 10
```

tracks
* TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Consultas - HAVING agrupar con condición

También podemos hacer que GROUP BY restrinja los grupos a aquellos que cumplen una condición utilizando la cláusula HAVING.

Ejemplo: Los compositores que escribieron más de 30 canciones

```
SELECT composer, count(trackid) AS cant -- un alias
FROM tracks
WHERE composer <> '' -- filtrar las que no tienen compositor
GROUP BY composer
HAVING cant > 30
ORDER BY cant desc
LIMIT 10
```

tracks
* TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

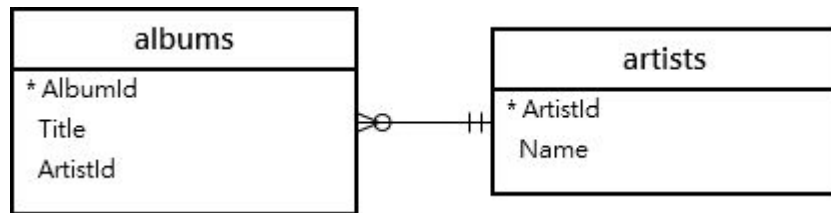
Consultas - JOIN

Cuando tenemos una relación entre tablas, podemos combinar sus campos con la cláusula JOIN.

JOIN busca las coincidencias entre los campos de las tablas tal como se indica con la palabra ON.

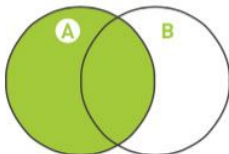
Ejemplo: Álbums y sus artistas

```
SELECT title, name
FROM albums
INNER JOIN artists
  ON artists.artistId = albums.artistId
```

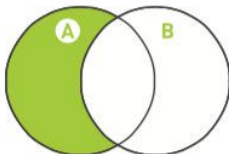


SQL JOINS

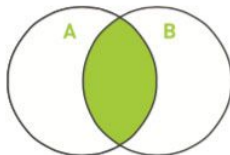
A CHEATSHEET BY WEBDEZIGN.CO.UK



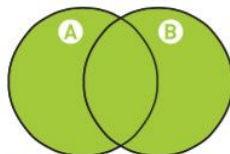
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



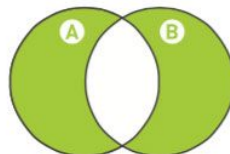
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



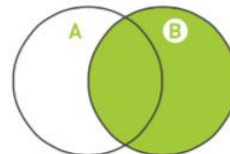
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



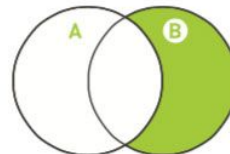
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



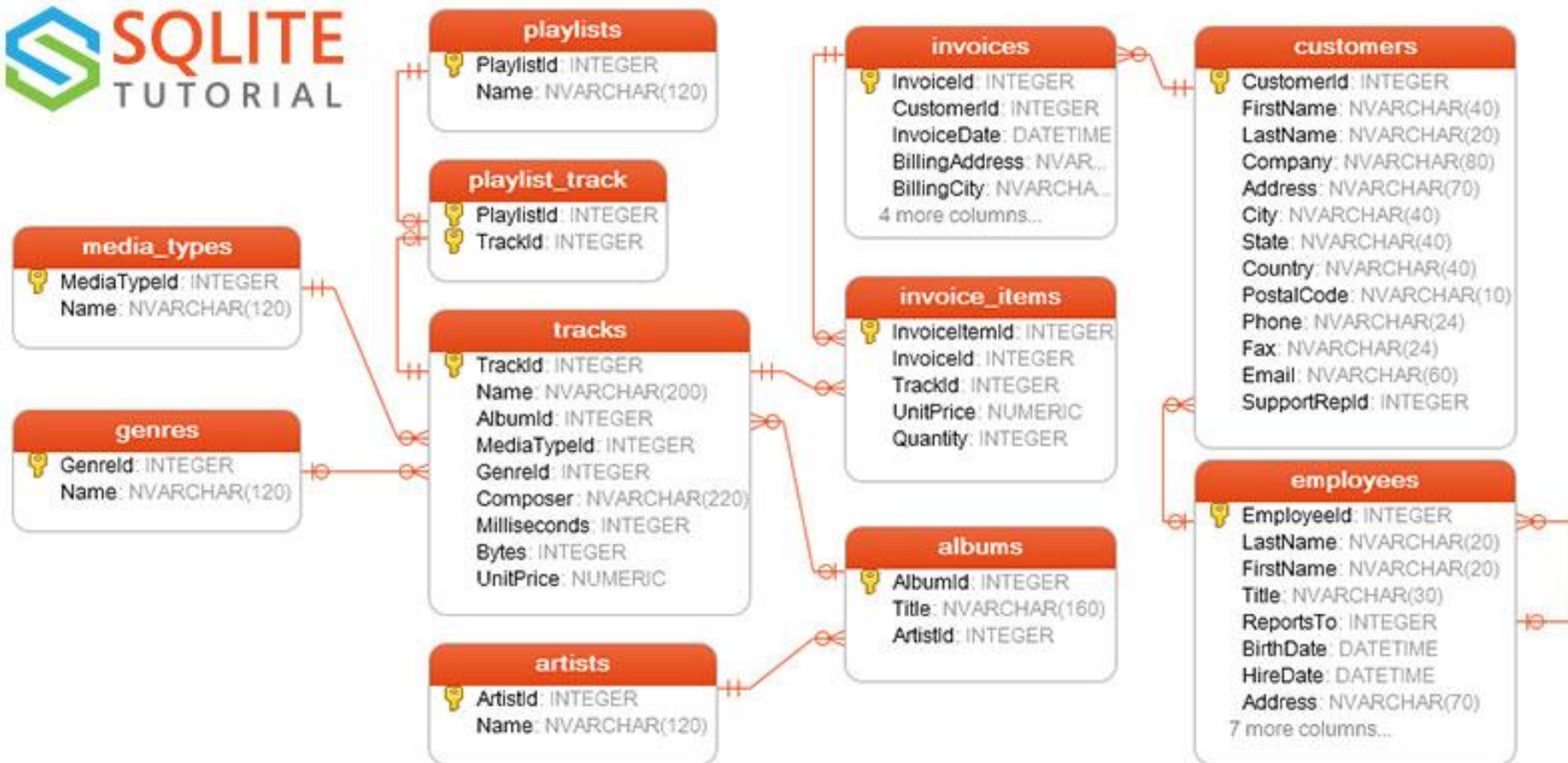
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



Notebook SQL

02 SQL

Pandas - Data Integration

Agrupamiento y agregación

1. groupby:

- Tomar una serie de columnas A, B, C
- Por cada combinación de valores de las columnas (a1, b1, c1), agrupar las filas que tengan esos valores.

2. agg:

- Toma una función f, por ej: sum(), mean(), count().
- Por cada grupo de filas, aplicar la función f a cada columna.

Agrupamiento y agregación

```
df.groupby('species').agg('sum')
```

	species	sepal_length	sepal_width	petal_length	petal_width
0	setosa	5.1	3.5	1.4	0.2
1	setosa	4.9	3.0	1.4	0.2
2	setosa	4.7	3.2	1.3	0.2
3	setosa	4.6	3.1	1.5	0.2
4	setosa	5.0	3.6	1.4	0.2
50	versicolor	7.0	3.2	4.7	1.4
51	versicolor	6.4	3.2	4.5	1.5
52	versicolor	6.9	3.1	4.9	1.5
53	versicolor	5.5	2.3	4.0	1.3
54	versicolor	6.5	2.8	4.6	1.5
100	virginica	6.3	3.3	6.0	2.5
101	virginica	5.8	2.7	5.1	1.9
102	virginica	7.1	3.0	5.9	2.1
103	virginica	6.3	2.9	5.6	1.8
104	virginica	6.5	3.0	5.8	2.2

SUM

SUM

SUM

	sepal_length	sepal_width	petal_length	petal_width
species				
setosa	24.3	16.4	7.0	1.0
versicolor	32.3	14.6	22.7	7.2
virginica	32.0	14.9	28.4	10.5

Data Integration - Join y merge

Combinar diferentes fuentes de información en un solo dataset coherente y utilizable.

```
df1.join(df2, how='outer')
```

- Une, basado en su índice (horizontalmente) los DataFrames y hace coincidir las filas donde el valor del índice es el mismo

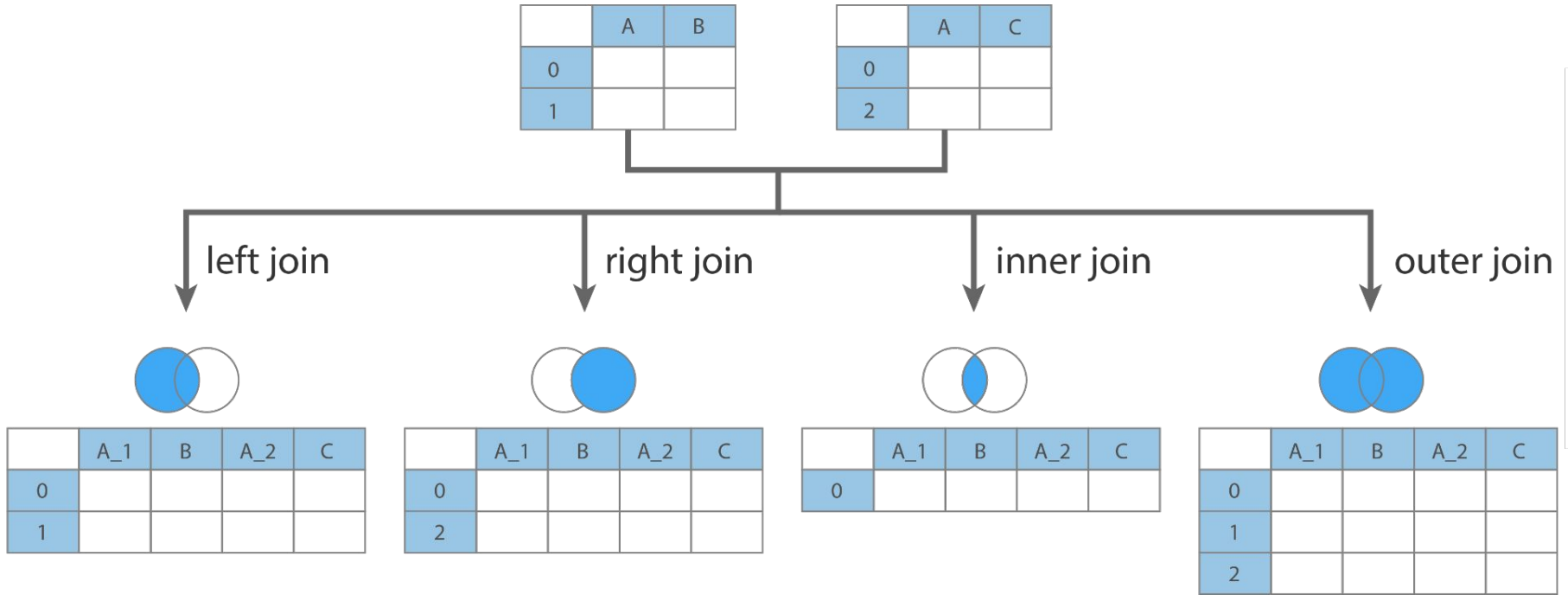
left			right			Result				
	A	B		C	D		A	B	C	D
K0	A0	B0	K0	C0	D0	K0	A0	B0	C0	D0
K1	A1	B1	K2	C2	D2	K1	A1	B1	NaN	NaN
K2	A2	B2	K3	C3	D3	K2	A2	B2	C2	D2
						K3	NaN	NaN	C3	D3

Join y merge

1. `df1.merge(df2, on='key')`
 - Igual al join, pero en lugar de comparar los índices, compara un conjunto de columnas.

left				right				Result					
	key	A	B		key	C	D		key	A	B	C	D
0	K0	A0	B0	0	K0	C0	D0	0	K0	A0	B0	C0	D0
1	K1	A1	B1	1	K1	C1	D1	1	K1	A1	B1	C1	D1
2	K2	A2	B2	2	K2	C2	D2	2	K2	A2	B2	C2	D2
3	K3	A3	B3	3	K3	C3	D3	3	K3	A3	B3	C3	D3

Join y merge





Errores comunes en integración de datos

- Claves duplicadas o inconsistentes.
- Tipos de datos incompatibles (ej. int vs str).
- Columnas con nombres iguales pero distintos significados.
- No revisar si hay valores faltantes después del merge.
- Merge sin especificar on= (pandas usa intersección de columnas con el mismo nombre, lo que puede ser confuso).

Notebook Combinación de datasets

[Link a la notebook](#)

Pandas Vs SQL

Operación	Descripción	Equivalente en Pandas
SELECT	Elegir columnas	<code>df[['col1', 'col2']]</code>
WHERE	Filtrar filas	<code>df[df['col'] > 10]</code>
GROUP BY + agregados	Agrupar y resumir	<code>df.groupby('col').mean()</code>
JOIN	Combinar tablas	<code>pd.merge(df1, df2, on='col')</code>
ORDER BY	Ordenar resultados	<code>df.sort_values('col')</code>
LIMIT	Limitar resultados	<code>df.head(n)</code>

Pandas Vs SQL

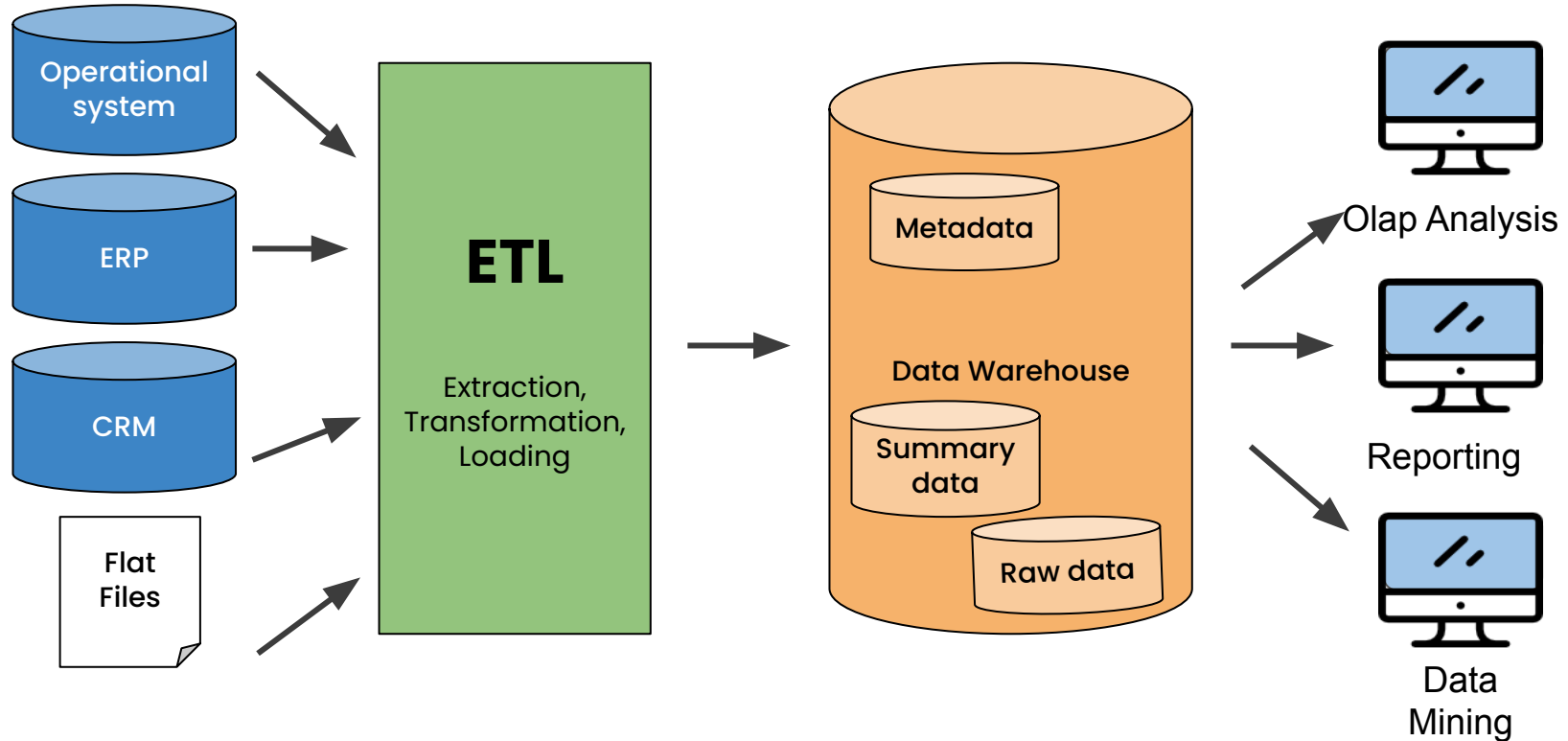
- **Exploración dirigida:** usar SQL para contestar preguntas específicas sin cargar todo el dataset.
- **Pre-filtrado:** usar SQL para reducir el volumen antes de importar a Python.
- **Limpieza inicial:** detección de nulos, duplicados, valores extremos.

https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_sql.html

<https://towardsdatascience.com/pandas-vs-sql-compared-with-examples-3f14db65c06f>

Mecanismos de Ingestión de Datos

Data Warehouse / Data Lake

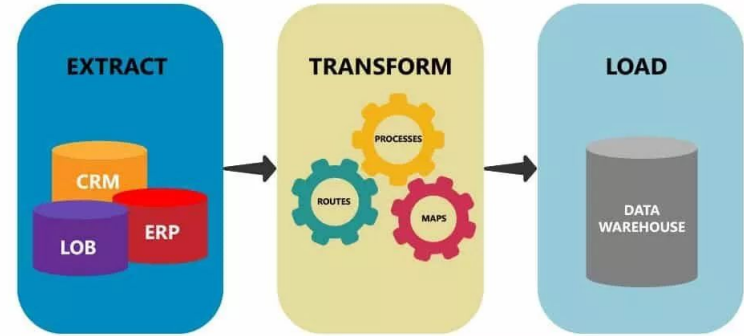


Data Warehouse / Data Lake

Característica	Data Warehouse	Data Lake
Tipo de datos	Estructurados	Estructurados, semi y no estructurados
Ejemplos de datos	Ventas, métricas financieras	Logs, texto libre, imágenes, JSON
Tecnología común	BigQuery, Redshift, Snowflake	S3, Azure Blob, Hadoop HDFS
Validación	Datos limpios, esquema definido	Ingesta sin validar, esquema flexible
Casos de uso	BI, reportes, tableros	Ciencia de datos, IA

ETL

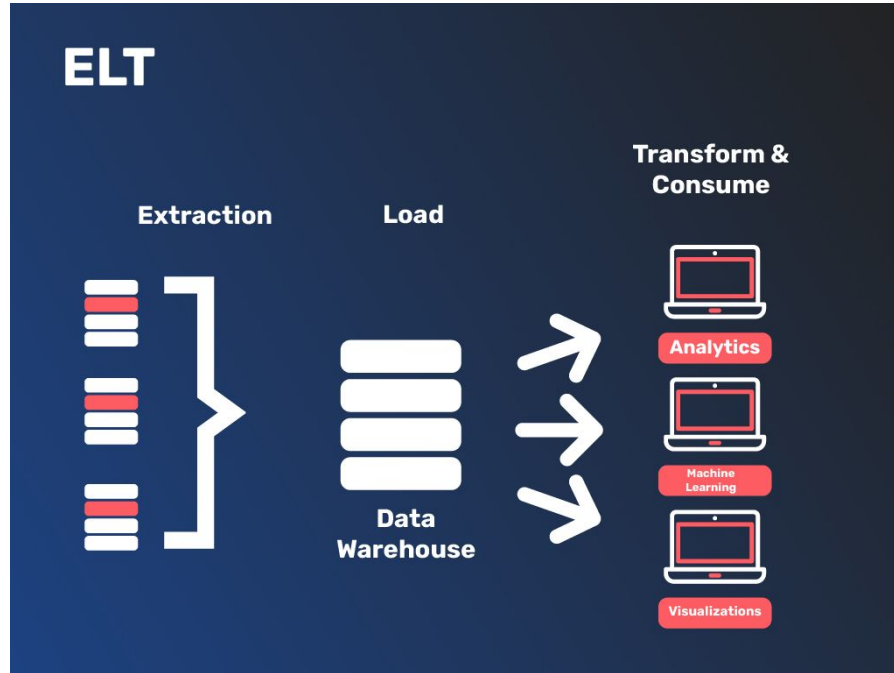
- **Extracción:**
 - Extraer datos heterogéneos de las distintas fuentes provistas.
- **Transformación:**
 - Normalizar
 - Elegir solo las partes relevantes
 - Interrelacionar los datos de las distintas fuentes
 - Adaptar al esquema de destino
- **Load (carga):**
 - Agregar toda estos datos al Data Warehouse
- Actualmente se está usando en otro orden:
ELT



ETL - Extract, Transform, Load

ELT

Carga datos en el warehouse **antes de que sean transformados**. Luego, cada business user los transforma de la forma más eficiente para c/u.

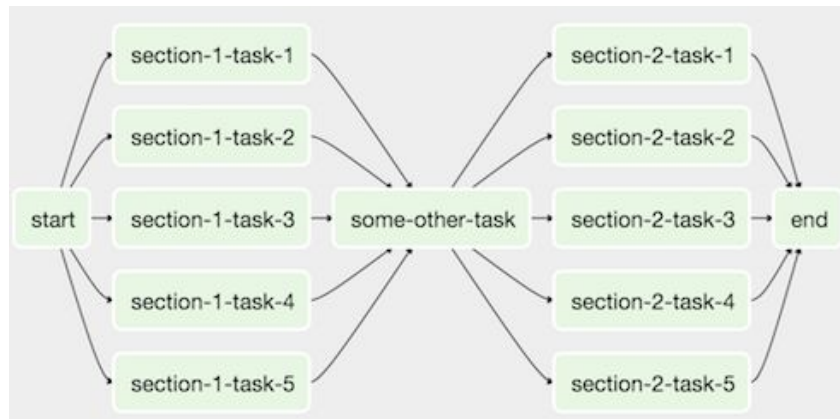


Batch vs Real time

Dimensión	Batch	Real-time / Streaming
Frecuencia	Cada cierto intervalo (horas/días)	Inmediato o con poco retraso
Herramientas	Airflow, Spark, SQL scripts	Kafka, Flink, Spark Streaming
Ejemplo de uso	Reporte diario de ventas	Detección de fraude en tiempo real
Ventajas	Simplicidad, bajo costo	Reacción rápida
Desventajas	Retrasos	Mayor complejidad y costo

Orquestar ETLs: Airflow

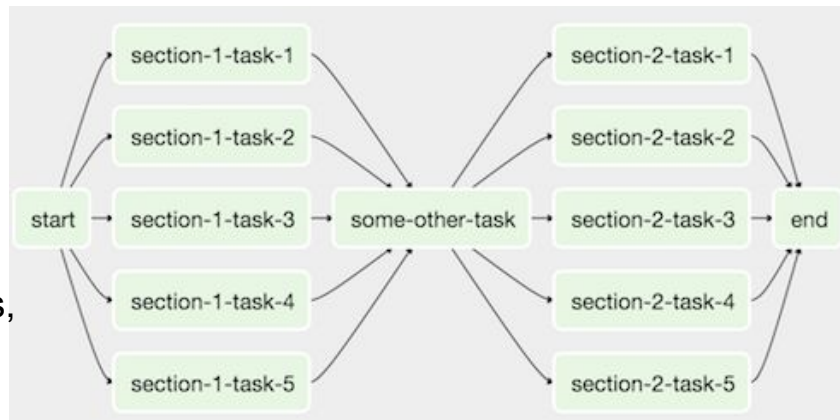
- Es una plataforma para programar, monitorear y orquestar flujos de datos.
- Permite definir tareas como código (DAGs = Directed Acyclic Graphs).
- Controla la dependencia y secuencia de tareas (por ejemplo: primero extraer, luego transformar, luego cargar).
- Administra ejecuciones recurrentes (por día, hora, etc).



Orquestar ETLs: Airflow

- Visibilidad: interfaz web para ver el estado de cada tarea.
- Escalabilidad y extensibilidad con plugins, operadores, sensores, etc.
- Muy útil para cuando tenemos que manejar muchos ETLs en paralelo.
- <https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html>

Airflow no procesa datos, solo coordina las tareas que sí lo hacen.



Apache
Airflow

Orquestar ETLs: Airflow

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
```

```
def extraer():
    print("Extrayendo datos...")
```

```
def transformar():
    print("Transformando datos...")
```

```
with DAG("mi_etl", start_date=datetime(2024, 1, 1),
        schedule_interval="@daily") as dag:
```

```
    t1 = PythonOperator(task_id="extraer",
        python_callable=extraer)
```

```
    t2 = PythonOperator(task_id="transformar",
        python_callable=transformar)
```

```
    t1 >> t2 # Define orden: extraer antes de transformar
```



Bonus

Notebook ETL y DAGs

Modelado de Datos por capas

Modelado Datos por capas

Building reliable, performant data pipelines with  **DELTA LAKE**

IMPROVE DATA QUALITY





Capa Bronce - Datos Crudos

Registro fiel del origen. Ingesta rápida y sin transformación.



Origen:

- APIs, bases, archivos (batch o streaming).
- Estructura igual al sistema fuente + columnas técnicas (*fecha_carga*, *origen*, etc).



Objetivos:

- Captura rápida y confiable.
- Auditoría, lineaje y reprocesamiento.
- Almacenamiento histórico (*cold storage*).



Limitaciones:

- Datos con errores, nulos o duplicados.
- No aptos para análisis directo.



"No lo tocamos, solo lo guardamos."



Capa Plata - Data limpios

Datos depurados, integrados y listos para combinar.



Transformaciones típicas:

- Limpieza de nulos, duplicados y errores.
- Tipificación correcta de columnas.
- Uniones con otras fuentes (por ejemplo, **join** con tablas de dimensiones).
- Estandarización de formatos (fechas, categorías, etc.).



Objetivos:

- Datos confiables y coherentes.
- Preparación para análisis exploratorio o modelado.
- Base reutilizable para múltiples procesos.



Limitaciones:

- Aún puede requerir agregaciones o métricas finales.
- No necesariamente optimizado para consumo por negocio.



"Donde los datos empiezan a tener forma."

Capa Gold

Datos transformados en productos: métricas, KPIs, vistas finales.

Contenido típico:

- Tablas agregadas, reportes listos para BI.
- KPIs, métricas, dashboards y modelos ML.
- Vistas específicas por caso de uso o stakeholder.

Objetivos:

- Facilitar decisiones con datos de alta calidad.
- Acceso simple y directo para analistas, negocio o apps.
- Optimización para lectura y consumo.

Limitaciones:

- No apta para trazabilidad técnica (ya está todo procesado).
- Rígida: cambios requieren volver a Plata o Bronce.

 ***"Datos listos para generar valor."***

Buenas prácticas

Guardar bien los datos no alcanza — hay que pensarlos para consultarlos eficientemente.

Indexación

- Crea estructuras adicionales para acelerar consultas sobre columnas específicas.
- Muy útil para búsquedas (**WHERE**, **JOIN**, **ORDER BY**).
- Usar en:
 - Claves primarias
 - Columnas muy consultadas
 - Columnas de unión entre tablas

Ejemplo: `CREATE INDEX idx_price ON propiedades(Price);`

Particionamiento

- Divide una tabla en subconjuntos (por fecha, zona, etc).
- Mejora tiempos de consulta y manejo de grandes volúmenes.
- Ideal para **data lakes** o **big warehouses** (ej. BigQuery, Redshift).

Ejemplo: particionar ventas por `YEAR(SaleDate)`

Buenas practicas



Clustering (organización física)

- Agrupa filas que comparten valores comunes (por ejemplo, barrio).
- Mejora performance sin necesidad de escanear toda la tabla.
- Se usa mucho en Redshift, BigQuery, Databricks.

Ejemplo: `CLUSTER BY Neighborhood`



Un buen modelo de datos no solo guarda bien: consulta bien, escala bien y cuesta menos.

Arquitectura de Datos

¿Qué es una arquitectura de datos?

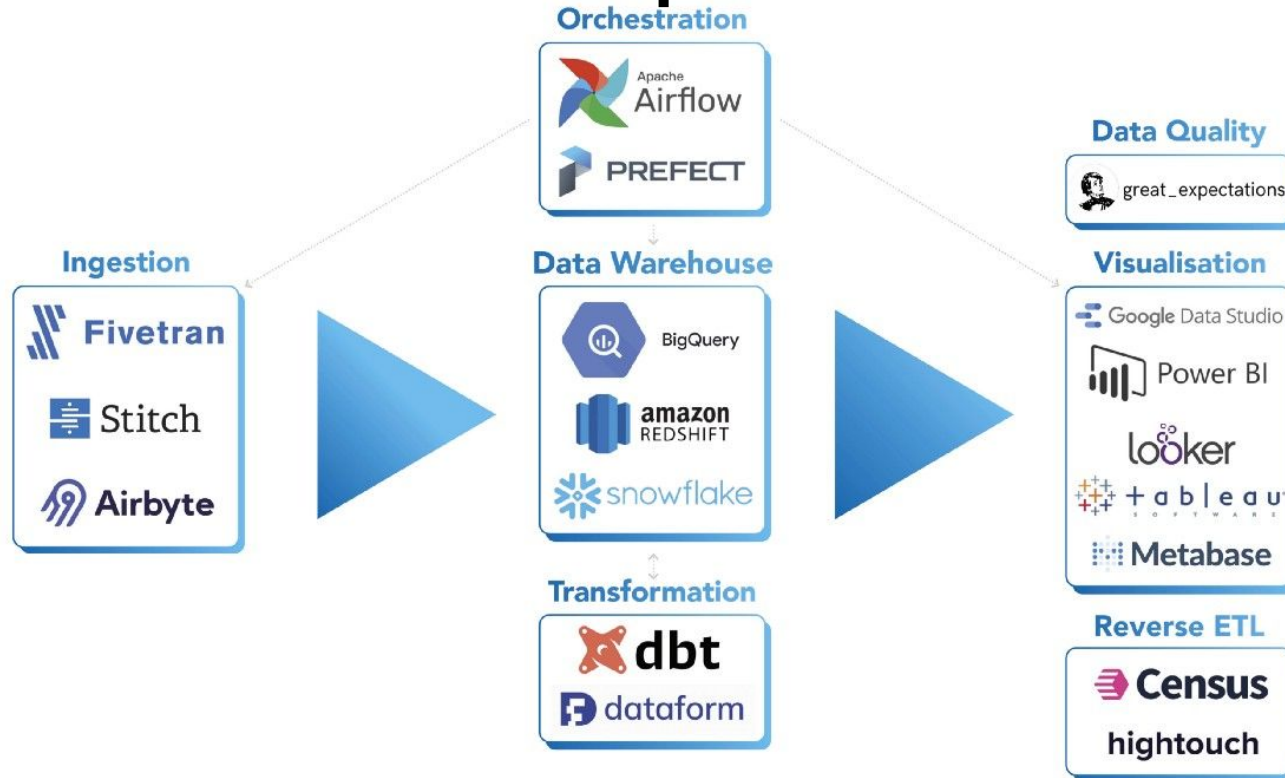
La forma en que organizamos y conectamos herramientas y procesos para capturar, procesar y usar datos.

Componentes clásicos:

- **Ingesta:** captura desde fuentes (APIs, bases, eventos).
- **Almacenamiento:** data lakes, data warehouses.
- **Procesamiento:** ETL/ELT, transformaciones.
- **Modelado:** estructuración en capas (bronce/plata/oro).
- **Consumo:** BI, notebooks, dashboards, ML.




 Todo esto se conecta a través de herramientas, estándares y *pipelines* orquestados.

Modern Data Stack - Arquitectura y Componentes






Data Sources (Collection and Tracking)

Tipos de fuentes:

-  **Bases productivas**
ERP, CRM, microservicios → Transacciones, usuarios, inventario.
-  **Logs y eventos de apps**
Clicks, navegación, abandonos de carrito, errores → Rastreo del comportamiento.
-  **Datos externos**
APIs públicas/privadas, datasets de terceros → Enriquecimiento (geolocalización, clima, divisas, etc.).

Ejemplo práctico:

-  Una venta concreta queda en la base de datos.
-  Un carrito abandonado solo lo ves en los *event logs*.
-  La ubicación estimada la podés obtener de un *servicio externo de IP geolocation*.

 *Combinar fuentes es lo que permite ver el panorama completo.*

➤ Data Ingestion

Captura estandarizada de datos desde múltiples fuentes hacia tu sistema central.

🔗 ¿Por qué es necesaria?

- Los datos vienen de muchos lados (bases, APIs, archivos, eventos).
- Cada fuente tiene su formato, frecuencia y lógica.
- Necesitamos un mecanismo uniforme y confiable para traerlos.

🧰 Herramientas de ingesta modernas:

- [Airbyte](#): open source, flexible, muchos conectores.
- [Fivetran](#): solución comercial plug-and-play.
- [Stitch](#): enfoque simple para data pipelines.

↻ Tipos de ingesta:

- **Batch**: cada X tiempo.
- **Streaming**: en tiempo real.

📦 Destino final:

Data lake o warehouse → donde se sigue transformando y modelando.



➤ Almacenamiento - Data Storage

📦 Espacio centralizado donde se almacenan datos desde múltiples fuentes.

♦ Tipos de almacenamiento:

- 🏛️ **Data Warehouse** → estructurado, consultas rápidas (Snowflake, Redshift, BigQuery)
- 🌊 **Data Lake** → flexible, datos crudos o semiestructurados (S3, GCS)
- 🧬 **Lakehouse** → mezcla de ambos (Databricks, Apache Iceberg, Delta)

♦ Tendencias modernas:

- ☁️ *Separación cómputo / almacenamiento*
Datos en **buckets** (S3, GCS), procesamiento con motores externos.
- 📄 Formatos eficientes: **Parquet, Iceberg, Delta**

💡 ***Diseñar bien esta capa impacta en costos, rendimiento y flexibilidad.***



➤ Data Transformation and Modelling



Capa de Transformación

Transformación de datos para convertirlos en modelos más accesibles y útiles.

♦ Capas de Modelo de Datos:

- **Raw:** Datos crudos, sin transformar.
- **Bronze:** Datos limpiados y estructurados parcialmente.
- **Silver:** Datos enriquecidos, listos para análisis complejos.
- **Gold:** Datos optimizados para reportes y dashboards.

♦ Herramientas:

- 🔧 **DBT:** Automatiza la creación de transformaciones de datos y modelos escalables con SQL.

💡 Impacto:

Un buen diseño de la transformación mejora la calidad de los datos y facilita el análisis en todas las capas.



➤ Data Orchestration

⚙️ Gestión de Procesos

Coordinar la ejecución de los procesos de ingesta y transformación de datos de manera eficiente y monitorizada.

♦ Función de la Orquestación:

- 🕒 Ejecutar procesos en momentos específicos (ej. cada día a las 7 am).
- 🔧 Asignar los recursos necesarios y monitorear fallos.
- 🔄 Integrar herramientas para que funcionen de manera fluida.

♦ Herramientas:

- 🔧 **Airflow**: Orquestación de flujos de trabajo en Python.
- 🔧 **Prefect**: Gestión de flujos de trabajo con monitoreo en tiempo real.
- 🔧 **Dagster**: Plataforma para orquestar y monitorizar pipelines de datos.

💡 Impacto:

La correcta orquestación asegura que los procesos de datos se ejecuten a tiempo, sin fallos, y con los recursos adecuados.




➤ Data Visualization

Construcción de Dashboards

Crear tableros fáciles de usar para mostrar insights y monitorear métricas clave del negocio.


♦ Función de la Visualización:

 Monitorear tendencias y métricas de negocio (ej. ventas por país/ciudad).


 Mostrar datos históricos y actuales (último día, semana, mes, 3 meses).

♦ Herramientas Populares:

 **PowerBI:** Intuitiva y fácil de usar, ideal para usuarios operativos.

 **Tableau:** Interfaz visual y sencilla para crear dashboards interactivos.

 **Looker:** Visualización de datos en tiempo real, fácil de explorar para no técnicos.

 **Databricks SQL:** Potente para grandes volúmenes de datos, con opciones de visualización amigables.

 **Microstrategy:** Dashboard con enfoque empresarial y accesible para perfiles no técnicos.

Impacto:

Una visualización amigable y clara facilita la toma de decisiones rápidas y basadas en datos, incluso para equipos operativos.




➤ Data Operationalization / Reverse ETLs

Exponer Datos a Herramientas Externas

Mover los datos procesados y modelados a sistemas operativos externos para su uso en aplicaciones de negocio.


♦ Función de Reverse ETL:

 Transferir segmentos de usuarios o métricas clave a herramientas como CRMs o plataformas de publicidad (ej. usuarios que compraron la semana pasada para campañas en Facebook/Google).

♦ Herramientas Populares:

 **Hightouch:** Sincroniza datos con aplicaciones externas sin necesidad de codificación.

 **Census:** Mueve datos procesados hacia aplicaciones operativas con facilidad.

 **Segment:** Envía datos a herramientas de marketing y productos en tiempo real.


Impacto:

Permite que los datos procesados sean utilizados en tiempo real para acciones de negocio, mejorando la toma de decisiones operativas.



Otros componentes adicionales...

Product Analytics

Analizar los eventos de la web para medir comportamiento de usuarios.  **Amplitude, MixPanel, Google Analytics.**

Events Taxonomy

Estandarizar y definir las propiedades de los eventos para mantener consistencia.  **Avo, Iteratively.**

Data Quality

Validar la calidad de los datos y asegurar su integridad.  **Great Expectations.**

Data Catalog

Definir un catálogo organizado de todos los datos disponibles en la empresa.

Data Governance

Gestionar permisos de acceso, edición y establecer políticas de privacidad y ownership.
Asegurar el manejo adecuado de datos sensibles.

Impacto:

Estos componentes aseguran la calidad, accesibilidad y seguridad de los datos, permitiendo un uso eficiente y responsable a lo largo de la organización.

Data Mesh

🔗 Enfoque Descentralizado

Organizar los datos analíticos por dominio, gestionados por equipos responsables de cada área.

♦ Requisitos:

Requiere madurez en los datos y la organización para ser implementado con éxito.

♦ Beneficios:

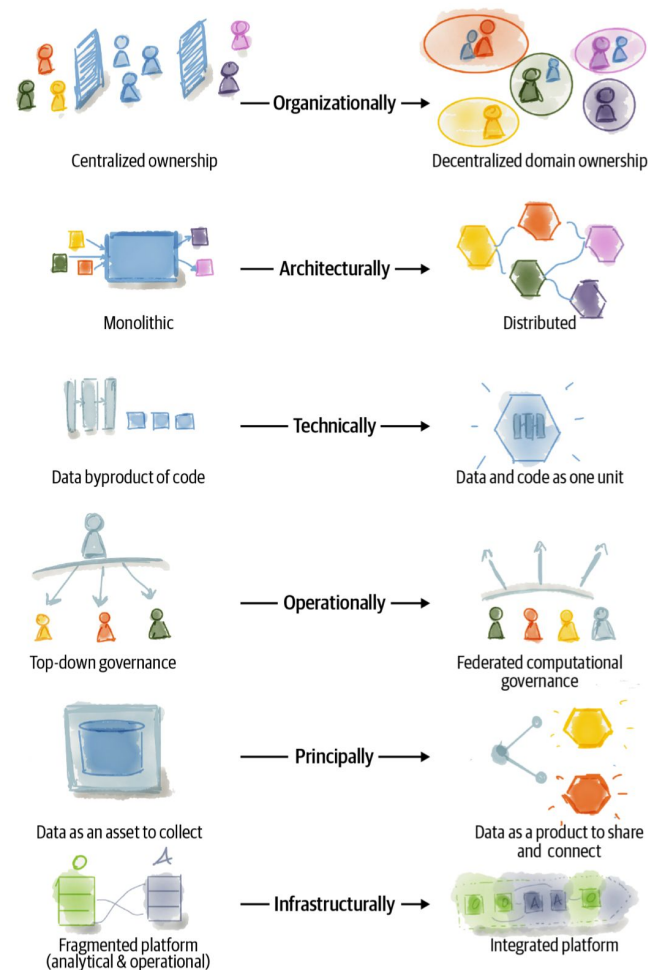
⚡ Agilidad frente a cambios del negocio.

👁️ Mayor visibilidad para los stakeholders sobre el valor de los datos.

💡 Impacto:

Facilita la escalabilidad y la autonomía, permitiendo que cada dominio gestione y explote sus propios datos de manera eficiente.

Libro: Data Mesh: Delivering Data-Driven Value at Scale, Zhamak Dehghani



Entregable parte 1

Material Complementario

- Tutorial de SQLite (incluye consola para aprender online)
 - <https://www.sqlitetutorial.net/tryit/>
- Soporte de SQL en pandas
 - Equivalencia entre métodos de pandas y sintaxis de SQL
https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_sql.html
 - Cargar y guardar datos en SQL
https://pandas.pydata.org/docs/user_guide/io.html?highlight=read_sql#sql-queries
- SQL
 - <https://www.sqlhabit.com/>
- Airflow
 - <https://airflow.apache.org/docs/apache-airflow/stable/tutorial/index.html>
 - <https://www.youtube.com/watch?v=IH1-0hwFZRQ>
 - <https://www.youtube.com/@Marclamberti>
 - <https://medium.com/@dustinstansbury/understanding-apache-airflows-key-concepts-a96efed52b1a>
 - <https://www.udemy.com/course/the-ultimate-hands-on-course-to-master-apache-airflow/>