Principios de Big Data

Clase 2 — ¿Cómo se guarda y procesa? Arquitectura sin código

Cristian Lazo Quispe

Diplomado en Ciencia de Datos en Salud · Universidad Científica del Sur

20 de junio de 2026

Curso **Principios de Big Data** · Diplomado en Ciencia de Datos en Salud (DICDASA)
Universidad Científica del Sur · Docente responsable del diplomado: Mg. Jorge L. Maguiña.

clazoq@uni.pe · cristian-lazo-quispe · CristianLazoQuispe

Itinerario de hoy (8:00 am – 1:00 pm)

8:00 – 8:20	Repaso + el caso de hoy (la UCI saturada)
8:20 – 9:30	**Bloque 1:** Por qué una computadora no basta + Map-Reduce
9:30 – 9:45	*Pausa activa*
9:45 – 11:00	**Bloque 2:** Tipos de dato + Data Lake (con ejemplos)
11:00 – 11:15	*Pausa activa*
11:15 – 12:00	**Bloque 3:** Hadoop/Spark + on-premise vs nube (debate)
12:00 – 12:40	**Práctica 1:** consultar la base con SQL (en vivo)
12:40 – 12:55	**Práctica 2:** datos no estructurados con MongoDB
12:55 – 1:00	Cierre + Trabajo 2

Para llevarte

Hoy decides **dónde se guardan** y **cómo se procesan** tus datos. Sigues sin programar: **entiendes, juzgas y decides**.

¿Qué veremos hoy?

1. Repaso + el caso de hoy
2. Por qué una sola computadora no basta
3. Map-Reduce: dividir y vencer
4. Los tres tipos de dato (con ejemplos)
5. ¿Dónde guardo todo eso? El Data Lake
6. Hadoop y Spark, sin asustarse
7. ¿En mi hospital o en la nube?
8. Práctica: consultar la base con SQL (datos estructurados)
9. Práctica: datos no estructurados con MongoDB
10. Cierre

Recordemos: ¿qué fue la Clase 1?

Ya sabemos distinguir las palabras

- **Big Data**: tanto dato que las herramientas de siempre no alcanzan.
- Las **5 V**: Volumen, Velocidad, Variedad, Veracidad, Valor.
- **IA** $\supset$ **ML** $\supset$ **Deep Learning**; un dashboard *muestra*, no decide.

Un solo monitor de UCI: miles de datos por hora.

Para llevarte

La Clase 1 respondió **“¿qué es?”**. Hoy respondemos **“¿dónde lo guardo y cómo lo proceso?”**.

El caso que nos acompaña hoy

Caso: la UCI saturada

La Dra. Ramírez tiene un problema real

"Mi UCI se **satura** y no sé por qué. Tengo **años** de datos de monitores, ingresos y altas... pero **dispersos** y son **millones** de registros. Mi Excel **se cuelga**."

Lo que ella necesita

Guardar todo eso en un lugar, procesarlo **sin esperar horas**, y poder **preguntarle** cosas a sus datos.

El problema de la Dra. Ramírez es de *arquitectura*.

Para llevarte

Cada tema de hoy resuelve un pedazo del problema de la Dra. Ramírez.

Calentamiento: ¿esto es Big Data? (a mano alzada)

A mano alzada: Vota en cada caso: ¿Big Data, sí o no?

1. Señales de **todos los monitores de UCI**, en tiempo real.
2. Una **planilla de 200 pacientes** en Excel.
3. El **archivo de imágenes** (rayos X, TC) del hospital, de varios años.

Para llevarte

Lo que importa no es el "sí/no", sino **por qué**: volumen, velocidad y variedad. Justo lo que hoy aprenderemos a **guardar y procesar**.

Analogía: un solo médico de guardia para todo el hospital

Caso: la UCI saturada

¿Te imaginas...?

Un **solo** médico de guardia atendiendo **emergencias, UCI, hospitalización y consulta** al mismo tiempo. **No alcanza**.

La solución no es un médico "más rápido". Es un **equipo** que se **reparte** el trabajo.

Con las computadoras pasa igual

Una sola máquina no puede con **millones** de registros. Se necesita un **equipo de máquinas** que colaboren.

Pongámoslo en números (los datos de la Dra. Ramírez)

Caso: la UCI saturada

Un año de monitoreo de UCI

- 1 monitor: $\sim$**86.400** lecturas por día.
- 20 camas $\times$ 365 días $\Rightarrow$ **cientos de millones** de lecturas al año.

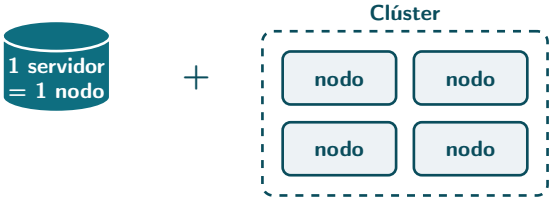
Abrir eso en Excel

Excel se queda en $\sim$1 millón de filas. La laptop se **cuelga** o tarda una eternidad. **No es que esté mal: es que no es la herramienta.**

Para llevarte

El problema de la Dra. Ramírez no se arregla con "una compu mejor": se arregla con **varias computadoras trabajando juntas**.

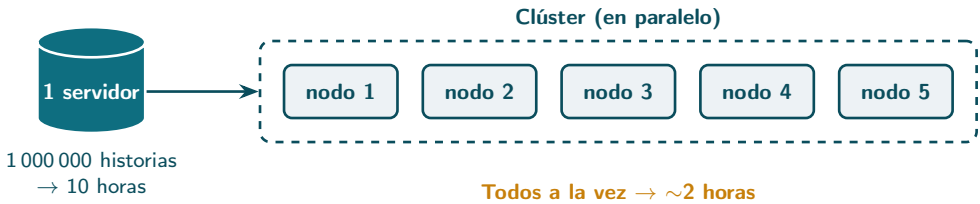
Tres palabras que vas a oír (sin misterio)



Para llevarte

Nodo = una computadora. **Clúster** = varios nodos trabajando juntos. No necesitas más: con eso entiendes el 90 % de las conversaciones técnicas.

La solución: un sistema distribuido



Para llevarte

Ejemplo clínico: procesar **todas las radiografías** del hospital de los últimos 5 años: una máquina tardaría días; **un clúster**, horas.

Pregunta rápida: sistemas distribuidos

Un servidor tarda 10 h en procesar un año de monitoreo de UCI. ¿Qué hace un *sistema distribuido*?

A. Compra un servidor más caro y potente.
B. Reparte el trabajo entre varias máquinas que trabajan a la vez.
C. Borra datos viejos para ir más rápido.
D. Lo abre todo en Excel.

Pregunta rápida: sistemas distribuidos

Un servidor tarda 10 h en procesar un año de monitoreo de UCI. ¿Qué hace un *sistema distribuido*?

A. Compra un servidor más caro y potente.
B. Reparte el trabajo entre varias máquinas que trabajan a la vez.
C. Borra datos viejos para ir más rápido.
D. Lo abre todo en Excel.

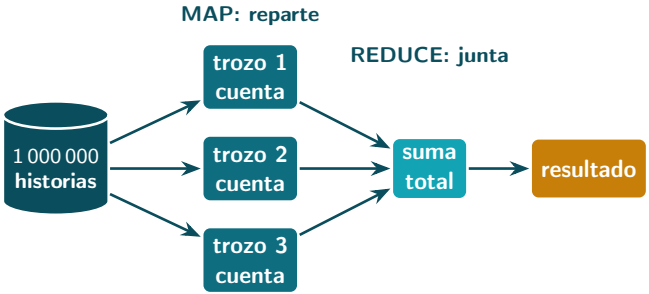
B. Reparte el trabajo entre varias máquinas en paralelo.

Por qué: Más máquinas colaborando > una máquina más cara. Así escala el Big Data.

¿Cómo se reparten el trabajo? Map-Reduce

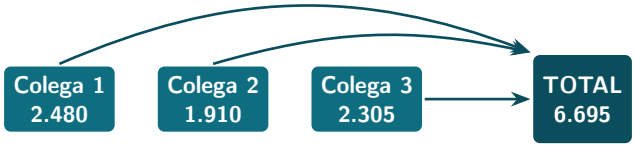
La receta de "dividir y vencer" (dos pasos)

1) **Map (repartir):** cada máquina procesa **su parte**.
2) **Reduce (juntar):** se **combinan** los resultados parciales.



Lo mismo, con números (cuenta “diabetes”)

MAP: cada quien cuenta su parte → REDUCE: se suman



Para llevarte

Nadie leyó el millón de historias solo: **se repartió** y luego **se sumó**. Eso es Map-Reduce.

Dinámica: seamos un Map-Reduce humano

Toda el aula · 12 min | Contemos diagnósticos sin que nadie se sature

Cada quien recibe **10 tarjetas** con diagnósticos (diabetes, hipertensión, asma...).

1. **MAP:** en silencio, cada uno cuenta sus tarjetas y anota "diagnóstico: cuántos".
2. **SHUFFLE:** hay 3 sillas rotuladas (*diabetes*, *hipertensión*, *otros*); lleva tus papelitos a la silla que corresponde.
3. **REDUCE:** quien está en cada silla **suma** y canta el total.

12 min

Para llevarte

Nadie contó el total solo: se **repartió** (Map), se **agrupó** (Shuffle) y se **sumó** (Reduce). **Acaban de hacer lo que Google hace con miles de máquinas.**

Antes de guardar, hay que entender qué dato tenemos

Tres “formas” de dato

No todo el dato es igual. Según su **forma**, se guarda y se procesa distinto. Veamos cada uno con un ejemplo de hospital.

Para llevarte

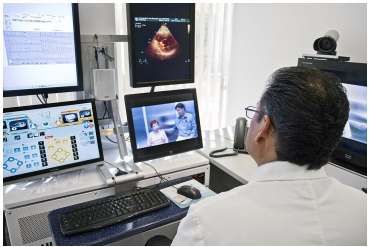
Regla simple: ¿**entra en una tabla de filas y columnas**? Entonces es “estructurado”. Si no... se complica (y es lo más común en salud).

1) Dato estructurado: entra en una tabla

En el hospital

Altas del mes, signos vitales en columnas, resultados de laboratorio, lista de camas. **Filas y columnas**, todo ordenadito.

Es el **más fácil** de analizar… pero el **menos común**.



Historia clínica electrónica: campos estructurados.

2) Dato semiestructurado: tiene etiquetas, no tabla fija

En el hospital

El mensaje de un **wearable** o un equipo: viene con **etiquetas** (“ritmo: 72”, “oxígeno: 96”) pero no en una tabla rígida.

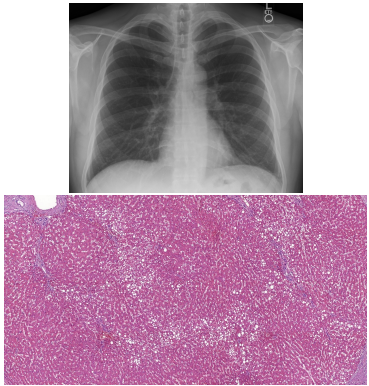
Formatos típicos: **HL7/FHIR**, **JSON**. Lo veremos en la Clase 3.

Un wearable manda datos con etiquetas (semiestructurado).

3) Dato no estructurado: texto, imagen, señal, audio

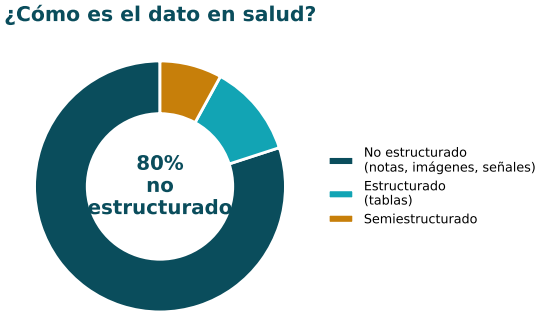
En el hospital (¡lo más valioso!)

La **nota de evolución** dictada, la **placa de rayos X**, la **lámina de patología**, el **audio** de una consulta.



Una imagen no "entra" en una tabla: es no estructurado.

El dato en salud es, sobre todo, NO estructurado



Para llevarte

El $\sim$**80 %** del dato en salud es **no estructurado**: el más valioso y el más difícil de aprovechar. Por eso necesitamos arquitecturas especiales.

Pregunta rápida: tipología de datos

¿Cuál de estos es un dato *NO estructurado*?

A. La tabla de altas del mes en Excel.
B. La nota de evolución dictada / la placa de rayos X.
C. El registro de signos vitales en columnas.
D. La lista de camas ocupadas.

Pregunta rápida: tipología de datos

¿Cuál de estos es un dato *NO estructurado*?

A. La tabla de altas del mes en Excel.
B. La nota de evolución dictada / la placa de rayos X.
C. El registro de signos vitales en columnas.
D. La lista de camas ocupadas.

B. La nota dictada y la imagen: no entran en una tabla. ✓

Por qué: A, C y D ya vienen en filas/columnas: son estructurados.

Dinámica: clasifiquen estos datos de su hospital

En parejas · 5 min | ¿Estructurado, semi o no estructurado?

Coloquen cada ejemplo en su columna (y discutan los casos dudosos):

Estructurado	Semiestructurado	No estructurado
Excel de turnos	Mensaje HL7/FHIR	Nota de evolución
Tabla de laboratorio	JSON de un wearable	Radiografía
Lista de camas	Correo con etiquetas	Audio de consulta

Para llevarte

Casi todo lo **valioso** (la nota, la imagen, el audio) cae en la columna **difícil**. Por eso el Data Lake importa tanto.

El problema de guardar: cada servicio tiene su “cajita”

Caso: la UCI saturada

Lo que le pasa a la Dra. Ramírez

Los monitores guardan en un sistema, los ingresos en otro, las altas en una planilla, las imágenes en otro lado… **todo disperso**. Nadie puede cruzar la información.

Para llevarte

Necesita **un solo lugar** donde quepa **todo**, de cualquier forma. Ese lugar se llama **Data Lake** (lago de datos).

La imagen que lo explica: ¿agua embotellada o un lago?

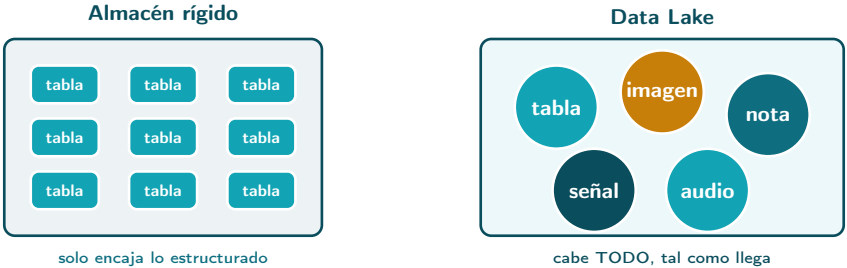
Almacén / Data Warehouse: agua *embotellada*.

Data Lake: el lago en su *estado natural*.

La analogía de James Dixon (quien acuñó "data lake", 2010)

El almacén es **agua embotellada**: filtrada y empaquetada para un uso fácil. El lago es **agua en estado natural**: llega cruda de la fuente y cada quien puede examinarla, sumergirse o tomar una muestra.

Almacén rígido vs. Data Lake (en un diagrama)



Para llevarte

El almacén tradicional solo acepta **tablas**. El **Data Lake** acepta **todo** (tablas, imágenes, notas, señales, audio), **tal como llega**.

El Data Lake se organiza en zonas

Para llevarte

Ejemplo: la lectura cruda del monitor entra en la **zona cruda**; ya validada pasa a la **zona limpia**; lista para un tablero, a la **zona lista**.

Las 4 cosas que un líder debe EXIGIR de su Data Lake

Seguridad	Quién entra y quién no; datos sensibles protegidos.
Organización	Zonas claras; nadie "pesca" a ciegas en el lago.
Linaje	De dónde vino cada dato y por dónde pasó (trazabilidad).
Gobernanza	Reglas de uso: quién usa qué, para qué y con qué permiso.

Sin gobernanza: el "pantano de datos"

Un Data Lake sin reglas se llena pero **nadie confía** en lo que hay dentro. Esto es justo lo que evalúas en el **Trabajo 2**.

Vas a oír estas palabras: aquí están sin misterio

HDFS	El "disco gigante" repartido entre muchas máquinas (*dónde* se guarda).
Hadoop	El sistema clásico que reparte el trabajo (Map-Reduce sobre HDFS).
Spark	La versión moderna y **rápida**: trabaja en **memoria (RAM)**.

Para llevarte

No tienes que recordar los nombres. Solo saber que **existen** y que sirven para **guardar y procesar** a gran escala.

¿Y si una máquina se daña? HDFS guarda copias

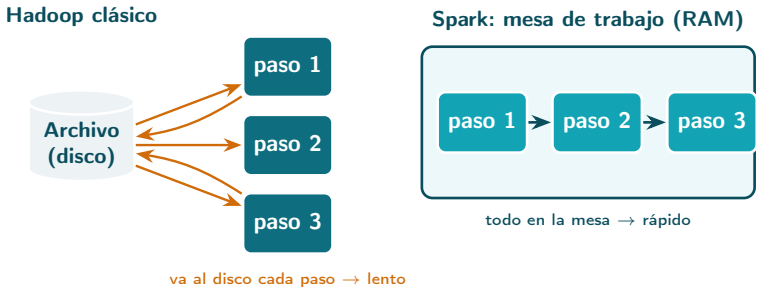
Una historia clínica partida en bloques (B1, B2, B3), cada uno por triplicado

Si el **Nodo 2** se cae, B2 sigue vivo en los Nodos 1 y 4, y B3 en el Nodo 3. **No se pierde nada.**

6 voluntarios · 5 min | Role-play: somos nodos

Cada voluntario es un nodo con sus bloques. **Siéntate** (te "caíste") y el grupo demuestra que **aún puede reconstruir** la historia completa.

¿Por qué Spark es más rápido? (ir al archivo vs. la mesa)



Para llevarte

Hadoop va al **archivo (disco)** en cada paso; Spark mantiene todo en la **mesa de trabajo (RAM)**. Por eso Spark vuela en tareas que repiten pasos.

Pregunta rápida: ¿por qué Spark es más rápido?

¿Por qué Spark suele ser más rápido que el Hadoop Map-Reduce clásico?

A. Porque siempre usa más servidores.
B. Porque trabaja en *memoria (RAM)* y no escribe a disco en cada paso.
C. Porque comprime los datos.
D. Porque está en la nube.

Pregunta rápida: ¿por qué Spark es más rápido?

¿Por qué Spark suele ser más rápido que el Hadoop Map-Reduce clásico?

A. Porque siempre usa más servidores.
B. Porque trabaja en *memoria (RAM)* y no escribe a disco en cada paso.
C. Porque comprime los datos.
D. Porque está en la nube.

B. Trabaja en memoria; evita ir al disco entre etapas.

Por qué: Disco = lento; RAM = rápido. Como tener todo en la mesa en vez de ir al archivo.

¿Para qué se usan en salud? (solo nombrarlos)

Casos típicos

- Calcular **riesgo de reingreso** sobre millones de historias.
- **Segmentar** pacientes parecidos (lo veremos en la Clase 4).
- Buscar **una palabra** en todas las notas clínicas.
- Procesar **millones de señales** de monitores de UCI.

Para llevarte

Hoy no los corremos: los **entendemos**. Como líder, basta con saber **qué hacen** y **cuándo pedirlos**.

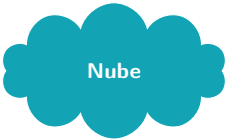
La gran decisión: ¿casa propia o alquiler?

On-premise = casa propia

inversión · control total

VS

Nube = alquiler



pago por uso · escala rápida

Para llevarte

On-premise = servidores en tu hospital (casa propia: control e inversión). **Nube** = los alquilas a un proveedor (pago por uso, escala rápida).

La otra forma de verlo: como la electricidad

La analogía de Nicholas Carr (*The Big Switch*)

Hace un siglo, cada fábrica tenía su **propio generador**. Hoy **nadie** genera su luz: la **compras de la red** y pagas por lo que usas.

El cómputo vive el mismo cambio

En vez de tus propios servidores (generador), consumes cómputo de la **nube** como un **servicio público**: lo enciendes cuando lo necesitas.

La "red eléctrica" del cómputo: enormes centros de datos.

¿Cuándo conviene cada una?

	On-premise (tu hospital)	**Nube (cloud)**
Control / ley	Máximo control; útil si la ley exige guardarlos en el país	El proveedor opera la infraestructura
Costo	Gran inversión inicial	Pago por uso, sin servidores propios
Escala	Limitada por tu hardware	Casi ilimitada y rápida
Ideal para	Datos muy sensibles, normativa estricta	Crecer rápido, picos, proyectos nuevos

Para llevarte

Variables a definir: **volumetría, tipo de datos, seguridad, ubicación, disponibilidad**.

En la práctica: casi siempre, un modelo híbrido

Lo mejor de ambos mundos

- Lo **muy sensible** (datos de pacientes) → on-premise.
- El **análisis a gran escala** y los picos → nube.

La infraestructura física existe igual; cambia quién la opera.

Debate: ¿on-premise o nube?

Dos bandos · 6 min | Defiende tu lado

Caso: tu hospital quiere analizar a gran escala los datos de un programa de pacientes con **VIH** (dato muy sensible).

- **Lado izquierdo del aula:** defiende **on-premise**.
- **Lado derecho:** defiende **nube**.
- Cada bando da **2 argumentos**; cerramos con la opción **híbrida**.

Para llevarte

No hay respuesta única: depende de **ley, sensibilidad, costo y escala**. Este es el corazón del **Trabajo 2**.

Volvamos a la Dra. Ramírez: ya tiene todo en su Data Lake

Lo que logró hasta ahora

- Sus datos dispersos → en **un solo Data Lake**.
- Millones de lecturas → procesadas con un **clúster (Spark)**.
- Ahora quiere **preguntar**... pero **no sabe programar**.

Para llevarte

Aquí entra la última pieza: **preguntarle a la base en español**.

La idea: consultar el data lake sin saber SQL

¿Qué vamos a ver?

1. Tenemos **millones** de lecturas de sensores (datos 100 % sintéticos).
2. Hacemos una pregunta **en español**.
3. Vemos en qué se traduce (**SQL**) y el resultado en **segundos**.

Tabla de ejemplo (sintética)

`paciente_id` · `frecuencia_cardiaca` · `spo2` · `edad` · `region`
Regiones: Lima, Cusco, Loreto, Arequipa, Piura.

Opcional en vivo: `shell.duckdb.org` (corre en el navegador, sin instalar nada).

Demo en vivo: cómo correrlo (sin instalar nada)

4 pasos, todo en el navegador

1. Abre `shell.duckdb.org` (es solo una página web).
2. Pega el **generador de datos** (crea 2 millones de lecturas; está en la guía del demo).
3. Escribe tu pregunta **en español** a una IA (ChatGPT/Claude/Gemini) y te devuelve el **SQL**.
4. Pega el SQL en el shell y **Enter**: responde en **segundos**.

Para llevarte

Nada se instala en tu computadora; DuckDB corre **dentro de la pestaña**.

`shell.duckdb.org`

El generador de datos (pega esto UNA vez en el shell)

Crea 2 millones de lecturas sintéticas (sin datos reales). También está en `clase2/demo/consultas_alumnos.txt` para compartir.

```
CREATE TABLE lecturas AS
WITH base AS (
  SELECT (random()*5000)::INT+1 AS paciente_id,
       (random()*94)::INT+1  AS edad,
       floor(random()*5)::INT AS rid
  FROM range(2000000)
)
SELECT paciente_id,
      round(78 + rid*0.9 + random()*16-8) AS frecuencia_cardiaca,
      least(100,greatest(70,round(97 - rid*0.4 + random()*8-4))) AS spo2,
      edad,
      (['Lima','Cusco','Loreto','Arequipa','Piura'])[rid+1] AS region
FROM base;
```

Esto es lo que verás (corrido en vivo)

```
DuckDB Web Shell running...
DuckDB v1.5.2 (Variegata)
Enter ".help" for usage hints.
memory D CREATE TABLE lecturas AS WITH base AS (SELECT (random()*5000)::INT+1 AS paciente_id, (random()*94)::INT+1 AS edad, floor(random()*5
)::INT AS rid FROM range(2000000)) SELECT paciente_id, round(78 + rid*0.9 + random()*16-8) AS frecuencia_cardiaca, least(100,greatest(70,rou
nd(97 - rid*0.4 + random()*8-4))) AS spo2, edad, (['Lima','Cusco','Loreto','Arequipa','Piura'])[rid+1] AS region FROM base;
memory D SELECT count(*) AS total_lecturas, count(DISTINCT paciente_id) AS pacientes FROM lecturas;
```

total_lecturas	pacientes
int64	int64
2000000	5001

```
memory D SELECT region, round(avg(frecuencia_cardiaca),1) AS fc_promedio, count(*) AS lecturas FROM lecturas WHERE edad>60 GROUP BY region O
RDER BY fc_promedio DESC;
```

region	fc_promedio	lecturas
varchar	double	int64
Piura	81.6	147039
Arequipa	80.7	147082
Loreto	79.8	146949
Cusco	78.9	146809
Lima	78.0	146588

```
memory D
```

Para llevarte

2.000.000 de lecturas, consultadas **al instante**. Captura real de `shell.duckdb.org`.

Pregunta 1 — del español al "idioma base"

Tu pregunta (en español)

"¿Cuál es la **frecuencia cardíaca promedio** de los mayores de 60, **por región**?"

Pregunta 1 — del español al "idioma base"

Tu pregunta (en español)

"¿Cuál es la **frecuencia cardíaca promedio** de los mayores de 60, **por región**?"

Lo que la base realmente entiende (SQL)

```
SELECT region, ROUND(AVG(frecuencia_cardiaca),1) AS fc
FROM lecturas WHERE edad > 60
GROUP BY region ORDER BY fc DESC;
```

Pregunta 1 — el resultado, en segundos

La consulta (SQL)

```
SELECT region,
  ROUND(AVG(frecuencia_cardiaca),1) AS fc
FROM lecturas WHERE edad > 60
GROUP BY region ORDER BY fc DESC;
```

region	fc
Loreto	81.4
Piura	80.9
Lima	80.2
Arequipa	79.8
Cusco	79.5

Para llevarte

Millones de lecturas, respondidas en **segundos**. La Dra. Ramírez ya le "habla" a su base.

Más preguntas (mismo patrón)

"¿En qué región hay más lecturas con SpO_2 bajo 92?"

```
SELECT region, COUNT(*) AS n FROM lecturas
WHERE spo2 < 92 GROUP BY region ORDER BY n DESC;
```

"¿Cuántos pacientes distintos y cuántas lecturas hay en total?"

```
SELECT COUNT(DISTINCT paciente_id) AS pacientes,
COUNT(*) AS lecturas FROM lecturas;
```

Para llevarte

"Preguntar en español" = alguien (o una **IA**) **traduce** tu frase a SQL. La base no entiende español; entiende **SQL**. Eso lo veremos con la IA en la Clase 3.

Y cuando Excel ya no alcanza...

La misma pregunta, dos herramientas

- Excel / pandas leyendo el archivo completo: **sufre** (se cuelga o tarda mucho).
- DuckDB (herramienta de Big Data) sobre el mismo dato: responde en **fracciones de segundo**.

La frase para recordar

"Pandas y Excel se diseñaron cuando un hospital tenía **miles** de registros al año. Hoy **un solo monitor** genera más que eso en **un día**."

Script de respaldo (sin internet): `scripts/demo1_escala.py` — solo "Ejecutar".

Bonus — banco de consultas (1/3): contar y ver

Copia, pega en `shell.duckdb.org` y corre. Tabla: `lecturas`.

Pregunta (en español)	SQL
Contar	
1. ¿Cuántas lecturas hay?	`SELECT count(*) FROM lecturas;`
2. ¿Cuántos pacientes distintos?	`SELECT count(DISTINCT paciente_id) FROM lecturas;`
3. ¿Cuántas de Loreto?	`SELECT count(*) FROM lecturas WHERE region='Loreto';`
4. ¿Cuántas con SpO_2<90?	`SELECT count(*) FROM lecturas WHERE spo2<90;`
Ver	
5. Muéstrame 5 lecturas	`SELECT * FROM lecturas LIMIT 5;`
6. Solo edad y región	`SELECT edad, region FROM lecturas LIMIT 5;`
7. Mayores de 80 años	`SELECT * FROM lecturas WHERE edad>80 LIMIT 5;`

Bonus — banco de consultas (2/3): promediar y agrupar

Pregunta (en español)	SQL
Promediar / resumir	
8. Edad promedio	`SELECT round(avg(edad),1) FROM lecturas;`
9. FC promedio	`SELECT round(avg(frecuencia_cardiaca),1) FROM lecturas;`
10. SpO_2 mínima y máxima	`SELECT min(spo2), max(spo2) FROM lecturas;`
Agrupar (GROUP BY)	
11. Lecturas por región	`SELECT region, count(*) FROM lecturas GROUP BY region;`
12. Edad promedio por región	`SELECT region, round(avg(edad),1) FROM lecturas GROUP BY region;`
13. FC promedio por región	`SELECT region, round(avg(frecuencia_cardiaca),1) FROM lecturas GROUP BY region;`
14. Pacientes por región	`SELECT region, count(DISTINCT paciente_id) FROM lecturas GROUP BY region;`

Bonus — banco de consultas (3/3): ordenar y combinar

Pregunta (en español)	SQL
Ordenar / top (ORDER BY)	
15. Las 5 de FC más alta	`SELECT * FROM lecturas ORDER BY frecuencia_cardiaca DESC LIMIT 5;`
16. Región con más SpO_2 baja	`SELECT region, count(*) FROM lecturas WHERE spo2<92 GROUP BY region ORDER BY 2 DESC;`
Combinar condiciones (AND / BETWEEN)	
17. >60 años y SpO_2<92	`SELECT count(*) FROM lecturas WHERE edad>60 AND spo2<92;`
18. FC entre 100 y 120	`SELECT count(*) FROM lecturas WHERE frecuencia_cardiaca BETWEEN 100 AND 120;`
19. Niños (edad<12)	`SELECT count(*) FROM lecturas WHERE edad<12;`
20. Adultos mayores de Loreto	`SELECT count(*) FROM lecturas WHERE region='Loreto' AND edad>=65;`

Para llevarte

Tres palabras hacen el 90 %: **SELECT** (qué), **WHERE** (filtra), **GROUP BY** (agrupa).

Bonus — datos REALES con un clic (menú `Datasets`)

En `shell.duckdb.org`, el menú **Datasets** carga bases reales al instante. Ej. **NYC Taxi**: **14,8 millones** de viajes de taxi (2010), consultados **en segundos** desde el navegador.

Para llevarte

Esto NO es de juguete: son **datos públicos reales** y la misma idea (SELECT / WHERE / GROUP BY) sirve para **cualquier** base, incluida la de tu hospital.

¿Y los datos NO estructurados? → MongoDB

Recuerda: el 80 % del dato en salud NO entra en una tabla

Notas, listas de diagnósticos, signos anidados... Para eso existen las bases de **documentos** (JSON), como **MongoDB**.

Una "fila" es un documento que puede anidar todo

```
{ paciente: "P001", edad: 67, region: "Loreto",
  signos: { fc: 82, spo2: 91 },
  diagnosticos: ["diabetes","hipertension"], nota: "disnea" }
```

Para llevarte

Para jugar sin instalar nada: `mongoplayground.net`. Datasets gigantes y búsqueda de texto: `search-playground.mongodb.com`.

Los datos de MongoDB (pega esto en el panel `Database`)

5 documentos de pacientes. También en `clase2/demo/consultas_alumnos.txt`.

```
[
  { paciente: "P001", edad: 67, region: "Loreto", signos: { fc: 82, spo2: 91 }, diagnosticos:
 ["diabetes","hipertension"], nota: "disnea de esfuerzo" },
  { paciente: "P002", edad: 34, region: "Lima",   signos: { fc: 75, spo2: 98 }, diagnosticos:
 ["asma"], nota: "estable" },
  { paciente: "P003", edad: 71, region: "Cusco",  signos: { fc: 90, spo2: 88 }, diagnosticos:
 ["epoc","hipertension"], nota: "saturacion baja" },
  { paciente: "P004", edad: 52, region: "Loreto", signos: { fc: 78, spo2: 95 }, diagnosticos:
 ["diabetes"], nota: "control de glucosa" },
  { paciente: "P005", edad: 29, region: "Piura",  signos: { fc: 70, spo2: 99 }, diagnosticos:
 [], nota: "sano" }
]
```

MongoDB en vivo: buscar documentos (`find`)

Para llevarte

Misma idea que "los de Loreto", pero sobre **documentos**. Captura real de `mongoplayground.net`.

MongoDB en vivo: agrupar y promediar (`aggregate`)

Para llevarte

FC promedio por región, igual que en SQL pero con `$group` sobre el campo anidado `signos.fc`.

Banco MongoDB (1/2): buscar y filtrar (find)

Colección col de documentos de pacientes. Las 20 están en el .txt para tus alumnos.

```
// 1. Todos los de Loreto
db.col.find({ region: "Loreto" })
// 2. Mayores de 60 años
db.col.find({ edad: { $gt: 60 } })
// 3. Oxigeno bajo (SpO2 < 92)
db.col.find({ "signos.spo2": { $lt: 92 } })
// 4. Con diabetes (esta en la lista)
db.col.find({ diagnosticos: "diabetes" })
// 5. Solo paciente y edad
db.col.find({}, { paciente: 1, edad: 1 })
// 6. Loreto Y mayores de 60
db.col.find({ region: "Loreto", edad: { $gt: 60 } })
// 7. Asma o EPOC
db.col.find({ diagnosticos: { $in: ["asma", "epoc"] } })
// 8. Sin ningun diagnostico
db.col.find({ diagnosticos: { $size: 0 } })
// 9. Nota que menciona "disnea"
db.col.find({ nota: /disnea/ })
// 10. Las 5 de mayor edad
db.col.find().sort({ edad: -1 }).limit(5)
```

Banco MongoDB (2/2): agrupar y resumir (aggregate)

```
// 11. Contar todos los documentos
db.col.countDocuments()
// 12. FC promedio por region
db.col.aggregate([{ $group: { _id: "$region", fc: { $avg: "$signos.fc" } } }])
// 13. Cuantos pacientes por region
db.col.aggregate([{ $group: { _id: "$region", n: { $sum: 1 } } }])
// 14. Edad promedio general
db.col.aggregate([{ $group: { _id: null, edad: { $avg: "$edad" } } }])
// 15. SpO2 minima por region
db.col.aggregate([{ $group: { _id: "$region", min: { $min: "$signos.spo2" } } }])
// 16. Mayores de 60: contar por region
db.col.aggregate([{ $match: { edad: { $gt: 60 } } }, { $group: { _id: "$region", n: { $sum: 1
 } } }])
// 17. Desglosar diagnosticos y contarlos
db.col.aggregate([{ $unwind: "$diagnosticos" }, { $group: { _id: "$diagnosticos", n: { $sum:
1 } } }])
// 18. Marcar alerta si SpO2 < 92
db.col.aggregate([{ $project: { paciente: 1, alerta: { $lt: ["$signos.spo2", 92] } } }])
```

El camino completo del dato (lo que hizo la Dra. Ramírez)

Caso: resuelto

Para llevarte

Cuatro pasos. Ninguno requirió que la Dra. Ramírez **programara**: requirió **decidir** con criterio en cada etapa.

Así resolvió la Dra. Ramírez su problema

Caso: resuelto

Datos dispersos	→ los junta en un **Data Lake** (con gobernanza).
Millones de lecturas	→ los procesa con un **clúster** (Map-Reduce / Spark).
No sabe programar	→ **le pregunta a la base en español**.
¿Dónde guardarlo?	→ decide **on-premise / nube / híbrido** según ley y escala.

Para llevarte

Ninguna de estas decisiones requirió programar. Requirió **criterio de líder**: exactamente lo que estás construyendo.

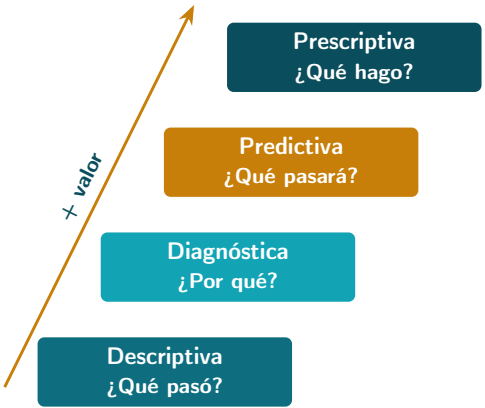
Lo que te llevas de la Clase 2

- Una sola máquina no basta $\Rightarrow$ **sistemas distribuidos** y **Map-Reduce**.
- Tres tipos de dato; el **no estructurado** (80 %) es el más valioso y difícil.
- **Data Lake**: un solo lugar, en zonas + 4 exigencias (**seguridad, organización, linaje, gobernanza**).
- **Hadoop/HDFS/Spark**: qué es cada uno; Spark es rápido por trabajar en memoria.
- **On-premise vs nube**: decides por ley, sensibilidad, costo y escala.

Para llevarte

Trabajo 2: tipología de datos de tu servicio + qué es un Data Lake y su **gobernanza** + una decisión **on-premise/nube** justificada. (Ver `EVALUACION`.)

¿Y para qué sirve todo esto? Los 4 niveles de la analítica



- **Descriptiva**: ¿cuántos ingresos a UCI este mes?
- **Diagnóstica**: ¿por qué se saturó?
- **Predictiva**: ¿se saturará la próxima semana?
- **Prescriptiva**: ¿cuántas camas abro hoy?

Para llevarte

Hoy llegamos a **describir y diagnosticar**. **Predecir** y **prescribir** es donde entra la **IA**: justo lo de la **Clase 3**.

Hacia la Clase 3

Clase 2	Hoy	Cómo se guarda y procesa (arquitectura).
Clase 3	Lun 6 jul	IA aplicada a la clínica + IA responsable (sesgo, privacidad, FHIR).
Clase 4	Mié 8 jul	Liderar el proyecto: tu "AI Charter" + segmentar pacientes.

Para llevarte

Trae **un problema de tu servicio** en mente: en la Clase 4 será tu proyecto final.

Gracias.

¿Dónde guardarías y cómo procesarías los datos de tu servicio?

clazoq@uni.pe · cristian-lazo-quispe · CristianLazoQuispe
Diplomado en Ciencia de Datos en Salud · Universidad Científica del Sur

Fuentes y créditos

Arquitectura y Big Data

- Dean, J. & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters*. Google. (Origen de Map-Reduce.)
- Apache Hadoop / Apache Spark — documentación oficial del proyecto.
- NIST (2019). *NIST Big Data Interoperability Framework*, SP 1500.
- Inmon, B. & Linstedt, D. — sobre Data Lake, zonas y gobernanza del dato.

Imágenes (Wikimedia Commons, uso educativo): monitor de signos vitales (U.S. Navy, dominio público); HCE / telemedicina (Intel Free Press, CC BY-SA 2.0); reloj inteligente (whity, CC BY 2.0); radiografía de tórax (J. Heilman, CC BY-SA 3.0); histología (Oktay98, CC BY-SA 4.0); centro de datos (C. Lender, CC BY 2.0); lago Moraine (Gorgo, dominio público); almacén / estantes (FIONASUN, CC0); biblioteca (J. Steakley, CC BY-SA 3.0). Logos **Apache Hadoop** y **Apache Spark**: marcas de la Apache Software Foundation (Apache License 2.0), uso educativo.
Analogías: "data lake / agua embotellada" — James Dixon (2010); "cómputo como electricidad" — Nicholas Carr, *The Big Switch* (2008); 4 niveles de analítica — Jordan Morrow, *Be Data Literate* (2021). Actividades estilo *CS Unplugged* (Bell et al.) y *Peer Instruction* (Mazur).
Demo: DuckDB (`duckdb.org`; shell en navegador `shell.duckdb.org`, captura real ejecutada el 20-jun-2026); datos 100 % sintéticos. Guion en `clase2/demo/GUIA_DEMO.md`.
Gráficos propios: elaboración propia con Python/matplotlib y TikZ (`scripts/fig_clase2.py`).