

APUNTES DE INTRODUCCIÓN

Bases de datos

lo esencial

Modelo relacional, diseño, SQL, índices, transacciones, NoSQL, analítica, nube (AWS, Azure, Google Cloud, Databricks, Snowflake, BigQuery), primeros pasos, seguridad y glosario

Autor: David M.
davmacsolutions.com

Octubre de 2026

ÍNDICE

1. Qué es una base de datos	2
2. Tipos de bases de datos	2
3. El modelo relacional	3
4. Diseño: del problema a las tablas	4
5. SQL: el lenguaje de las bases de datos	5
6. Índices y rendimiento	7
7. Transacciones y concurrencia	8
8. Bases de datos NoSQL	9
9. Bases de datos para analítica	9
10. Bases de datos en la nube	10
11. Primeros pasos con Databricks, Snowflake y BigQuery	12
12. Seguridad y copias de seguridad	15
13. Herramientas	16
14. Errores frecuentes	16
15. Lista de comprobación	16
16. Glosario	17

1. QUÉ ES UNA BASE DE DATOS

Una base de datos es un conjunto de datos organizados para poder guardarlos, consultarlos y modificarlos de forma fiable. El programa que la gestiona se llama **sistema gestor de bases de datos** (SGBD): se encarga de almacenar la información, responder a las consultas, impedir que dos usuarios se pisen y recuperar los datos si algo falla.

1.1 Por qué no basta con una hoja de cálculo

- **Volumen:** una base de datos maneja millones de registros sin degradarse.
- **Integridad:** puede impedir datos imposibles (un pedido sin cliente, una fecha inválida).
- **Concurrencia:** muchas personas y programas pueden leer y escribir a la vez.
- **Seguridad:** permisos por usuario, tabla o columna, y registro de accesos.
- **Recuperación:** copias de seguridad y restauración a un punto concreto del tiempo.

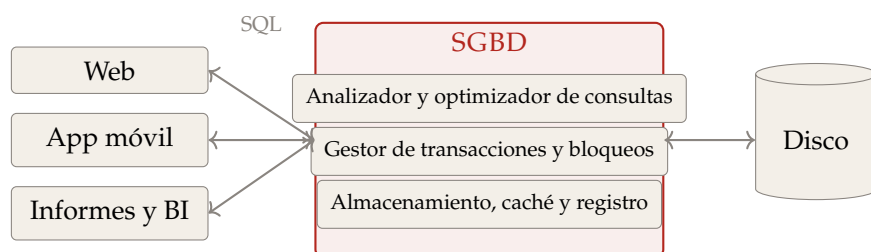


Figura 1. Arquitectura básica: las aplicaciones nunca tocan los ficheros; siempre pasan por el SGBD.

2. TIPOS DE BASES DE DATOS

Tipo	Cómo organiza los datos	Ejemplos
Relacional	Tablas con filas y columnas relacionadas entre sí; se consultan con SQL.	PostgreSQL, MySQL, SQL Server, Oracle, SQLite
Documental	Documentos tipo JSON, con estructura flexible.	MongoDB, Couchbase
Clave-valor	Pares clave → valor; muy rápida para leer y escribir.	Redis, DynamoDB
Columnar ancha	Filas con columnas variables, distribuidas en muchos servidores.	Cassandra, HBase
De grafos	Nodos y relaciones; ideal para redes y recomendaciones.	Neo4j
Analítica	Almacenamiento por columnas para agregar grandes volúmenes.	BigQuery, Snowflake, ClickHouse
Vectorial	Vectores numéricos para búsquedas por similitud (IA).	pgvector, Pinecone

Observación. La mayoría de proyectos deberían empezar con una base de datos relacional. Es la opción más madura, más documentada y más flexible; las demás resuelven problemas concretos que conviene tener antes de adoptarlas.

3. EL MODELO RELACIONAL

Propuesto por Edgar F. Codd en 1970, organiza los datos en **tablas** (relaciones). Cada tabla representa un tipo de cosa (clientes, pedidos, productos); cada **fila** es un elemento concreto y cada **columna**, un atributo con un tipo de dato definido.

TABLA CLIENTES

id	nombre	email	ciudad
1	Ana Ruiz	ana@correo.es	Madrid
2	Luis Gil	luis@correo.es	Vigo
3	Marta Sanz	marta@correo.es	Sevilla

fila →

↑
clave primaria

↑
columna

Figura 2. Anatomía de una tabla. La clave primaria identifica cada fila de forma única.

3.1 Claves

Clave primaria (PK). Columna o combinación de columnas que identifica cada fila sin repetirse nunca. Suele ser un número autoincremental o un identificador único (UUID).

Clave foránea (FK). Columna que apunta a la clave primaria de otra tabla. Así se relacionan las tablas sin duplicar datos.

Clave candidata o única. Cualquier columna que también podría identificar la fila (por ejemplo, el email) y a la que se impone que no se repita.

CLIENTES			PEDIDOS		
id	nombre		id	cliente_id	importe
1	Ana	←	101	1	49,90
2	Luis	←	102	3	15,00
3	Marta	←	103	1	120,00
		←	104	2	9,95

Figura 3. La clave foránea `cliente_id` enlaza cada pedido con su cliente sin copiar sus datos.

3.2 Tipos de datos habituales

Tipo	Uso
INT, BIGINT	Números enteros: identificadores, cantidades.
DECIMAL(p, s)	Importes exactos. Nunca uses coma flotante para dinero.
VARCHAR(n), TEXT	Texto corto o largo.
DATE, TIMESTAMP	Fechas y fechas con hora (mejor con zona horaria).
BOOLEAN	Verdadero o falso.
JSON	Datos semiestructurados dentro de una base relacional.

4. DISEÑO: DEL PROBLEMA A LAS TABLAS

Diseñar bien al principio ahorra meses de parches. El proceso habitual tiene tres pasos:

1. **Modelo conceptual:** qué cosas existen en el negocio (entidades), qué datos tienen (atributos) y cómo se relacionan.
2. **Modelo lógico:** traducir ese esquema a tablas, columnas y claves.
3. **Modelo físico:** tipos de datos concretos, índices y ajustes del SGBD elegido.

4.1 El modelo entidad-relación

Las relaciones entre entidades tienen una **cardinalidad**: uno a uno (1:1), uno a muchos (1:N) o muchos a muchos (N:M).

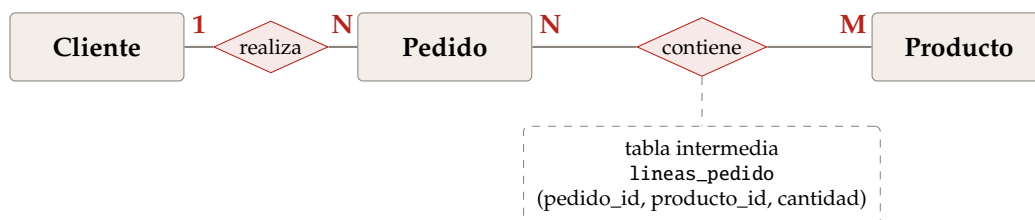


Figura 4. Modelo entidad-relación de una tienda. Una relación N:M se resuelve con una tabla intermedia.

4.2 Normalización

Normalizar es repartir los datos en tablas de forma que cada dato se guarde *una sola vez*. Así se evitan incoherencias: si la dirección de un cliente está copiada en cien pedidos, tarde o temprano alguna copia quedará desactualizada.

Forma normal	Regla, en palabras sencillas
1FN	Cada celda contiene un solo valor; nada de listas como «camiseta, gorra».
2FN	Cada columna depende de <i>toda</i> la clave primaria, no solo de una parte.
3FN	Ninguna columna depende de otra columna que no sea la clave (el código postal no debe arrastrar la ciudad en cada fila).

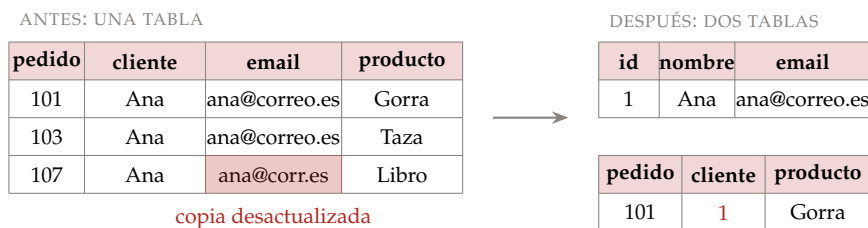


Figura 5. Antes y después de normalizar: los datos del cliente pasan a guardarse una sola vez.

Observación. En analítica a veces se *desnormaliza* a propósito (se duplican datos) para que las consultas sean más rápidas. Es una decisión consciente, no un descuido.

5. SQL: EL LENGUAJE DE LAS BASES DE DATOS

SQL (*Structured Query Language*) es el lenguaje estándar para trabajar con bases de datos relacionales. Es declarativo: se describe *qué* se quiere y el SGBD decide *cómo* obtenerlo.

Familia	Órdenes	Para qué
DDL (definición)	CREATE, ALTER, DROP	Crear y modificar tablas y estructuras.
DML (manipulación)	INSERT, UPDATE, DELETE	Añadir, cambiar y borrar datos.
DQL (consulta)	SELECT	Leer datos.
DCL (control)	GRANT, REVOKE	Dar y quitar permisos.
TCL (transacciones)	BEGIN, COMMIT, ROLLBACK	Agrupar operaciones.

5.1 Crear tablas

```
CREATE TABLE clientes (
  id      SERIAL PRIMARY KEY,
  nombre  VARCHAR(100) NOT NULL,
  email   VARCHAR(150) UNIQUE NOT NULL,
  alta    DATE DEFAULT CURRENT_DATE
);

CREATE TABLE pedidos (
  id          SERIAL PRIMARY KEY,
  cliente_id  INT NOT NULL REFERENCES clientes(id),
  importe     DECIMAL(10,2) CHECK (importe >= 0),
  fecha       DATE NOT NULL
);
```

5.2 Insertar, modificar y borrar

```
INSERT INTO clientes (nombre, email) VALUES ('Ana Ruiz', 'ana@correo.es');
UPDATE clientes SET email = 'ana.ruiz@correo.es' WHERE id = 1;
DELETE FROM pedidos WHERE fecha < '2020-01-01';
```

ATENCIÓN. Un UPDATE o DELETE sin WHERE afecta a *todas* las filas de la tabla. Antes de ejecutarlo, prueba el mismo filtro con un SELECT y, si puedes, hazlo dentro de una transacción.

5.3 Consultar

```
SELECT nombre, email
FROM clientes
WHERE alta >= '2026-01-01'
ORDER BY nombre
LIMIT 10;
```

El orden en que se *escribe* una consulta no es el orden en que se *ejecuta*. Entenderlo explica muchos errores, como por qué no se puede usar en el WHERE un alias definido en el SELECT.



Figura 6. Orden lógico de ejecución de una consulta SELECT.

5.4 Agrupar y resumir

```
-- Ventas totales por cliente, solo los que superan 100 euros
SELECT cliente_id, COUNT(*) AS n_pedidos, SUM(importe) AS total
FROM pedidos
GROUP BY cliente_id
HAVING SUM(importe) > 100
ORDER BY total DESC;
```

Las funciones de agregación más usadas son COUNT, SUM, AVG, MIN y MAX. WHERE filtra filas antes de agrupar; HAVING filtra grupos después.

5.5 Combinar tablas: JOIN

```
SELECT c.nombre, p.fecha, p.importe
FROM clientes c
JOIN pedidos p ON p.cliente_id = c.id;
```

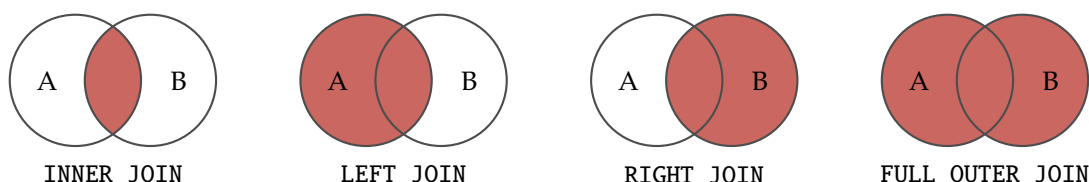


Figura 7. Los cuatro JOIN más usados. En rojo, las filas que devuelve cada uno (A = clientes, B = pedidos).

JOIN	Devuelve
INNER	Solo las filas que tienen pareja en las dos tablas.
LEFT	Todas las filas de A y, si existen, sus parejas de B (si no, columnas vacías: NULL).
RIGHT	Lo mismo, pero conservando todas las de B.
FULL OUTER	Todas las filas de ambas tablas, emparejadas cuando es posible.

5.6 Subconsultas, CTE y funciones de ventana

```
-- CTE: una consulta con nombre, mas legible que anidar
WITH gasto AS (
  SELECT cliente_id, SUM(importe) AS total FROM pedidos GROUP BY cliente_id
)
SELECT c.nombre, g.total,
       RANK() OVER (ORDER BY g.total DESC) AS puesto -- funcion de ventana
FROM gasto g JOIN clientes c ON c.id = g.cliente_id;
```

Observación. NULL significa «desconocido», no cero ni texto vacío. Por eso `x = NULL` nunca es cierto: hay que escribir `x IS NULL`.

6. ÍNDICES Y RENDIMIENTO

Un índice es una estructura auxiliar que permite encontrar filas sin recorrer toda la tabla, como el índice alfabético de un libro. El tipo más común es el **árbol B**: mantiene los valores ordenados y llega a cualquiera en pocos saltos.

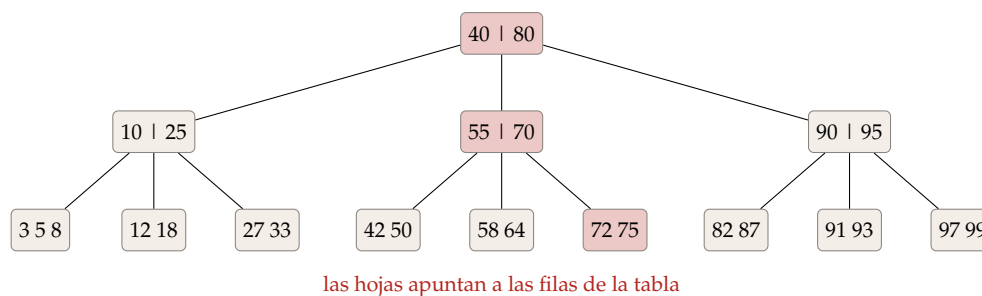


Figura 8. Índice en árbol B: para encontrar el 72 basta con bajar tres niveles en lugar de leer toda la tabla.

```
CREATE INDEX idx_pedidos_cliente ON pedidos (cliente_id);
EXPLAIN SELECT * FROM pedidos WHERE cliente_id = 42; -- ver como se ejecuta
```

6.1 Cuándo crear un índice

- En columnas que se usan mucho en WHERE, JOIN y ORDER BY.
- En las claves foráneas (muchos SGBD no lo hacen solos).
- Con cabeza: cada índice ocupa espacio y ralentiza las escrituras, porque hay que actualizarlo en cada INSERT, UPDATE o DELETE.

6.2 Buenas prácticas de rendimiento

- Pedir solo las columnas necesarias, no `SELECT *`.
- Revisar el plan de ejecución (`EXPLAIN`) de las consultas lentas.
- Evitar aplicar funciones a la columna filtrada (`WHERE YEAR(fecha) = 2026`), porque impide usar el índice; mejor un rango de fechas.
- Pagar los resultados y evitar el problema de las «N+1 consultas» desde la aplicación.

7. TRANSACCIONES Y CONCURRENCIA

Una transacción agrupa varias operaciones que deben ocurrir *todas o ninguna*. El ejemplo clásico es una transferencia: restar de una cuenta y sumar en otra.

```
BEGIN;
UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;
UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;
COMMIT;  -- o ROLLBACK si algo ha fallado
```

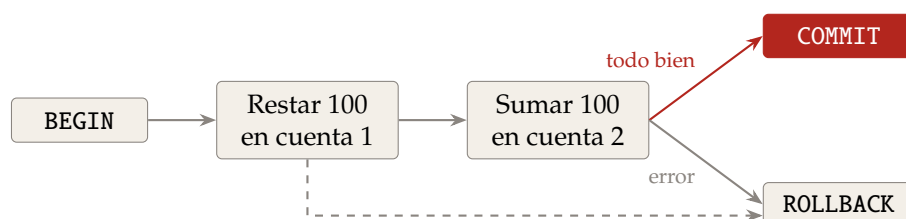


Figura 9. Una transacción termina siempre en uno de dos estados: confirmada o deshecha por completo.

7.1 Propiedades ACID

Propiedad	Garantía
Atomicidad	Se ejecuta todo o nada.
Consistencia	Tras la transacción, se cumplen todas las reglas (claves, restricciones).
Aislamiento	Las transacciones simultáneas no ven los cambios a medias de las demás.
Durabilidad	Lo confirmado sobrevive a un corte de luz o un fallo del servidor.

7.2 Problemas de concurrencia

Cuando muchas transacciones trabajan a la vez pueden aparecer **lecturas sucias** (leer datos aún no confirmados), **lecturas no repetibles** (un dato cambia entre dos lecturas) y **lecturas fantasma** (aparecen filas nuevas). Los *niveles de aislamiento* (*read committed*, *repeatable read*, *serializable*) permiten elegir cuánta protección se quiere a cambio de rendimiento. Cuando dos transacciones se esperan mutuamente se produce un **interbloqueo**: el SGBD cancela una de ellas y la aplicación debe reintentarla.

8. BASES DE DATOS NOSQL

NoSQL agrupa a las bases de datos que no siguen el modelo de tablas. Suelen sacrificar parte de la rigidez relacional a cambio de escalar en muchos servidores o de adaptarse mejor a un tipo concreto de dato.

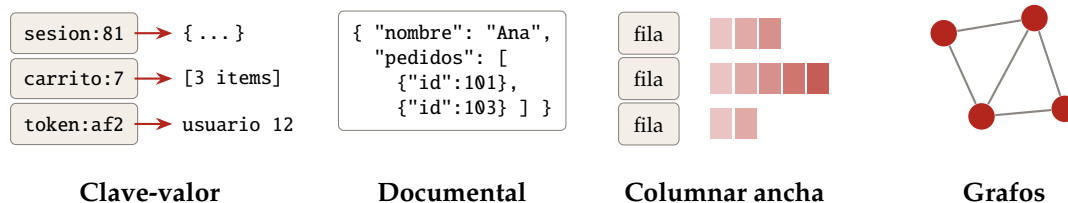


Figura 10. Las cuatro familias NoSQL clásicas.

8.1 Teorema CAP

En un sistema distribuido, si la red se parte (P), hay que elegir entre **consistencia** (C: todos ven el mismo dato) y **disponibilidad** (A: el sistema siempre responde). Las bases relacionales tradicionales priorizan la consistencia; muchas NoSQL ofrecen *consistencia eventual*: los datos acaban siendo iguales en todos los nodos, pero no al instante.

Elige relacional si...	Considera NoSQL si...
Los datos tienen estructura clara y relaciones.	La estructura cambia mucho o es muy irregular.
Necesitas transacciones fiables (dinero, stock).	Necesitas escalar escrituras masivas en muchos servidores.
Vas a hacer consultas variadas que aún no conoces.	El patrón de acceso es simple y conocido (por clave).

9. BASES DE DATOS PARA ANALÍTICA

Las bases de datos del día a día (**OLTP**) están pensadas para muchas operaciones pequeñas: registrar un pedido, actualizar un saldo. Para analizar años de historia se usan sistemas **OLAP**: almacenes de datos (*data warehouse*) que guardan por columnas y agregan millones de filas en segundos.

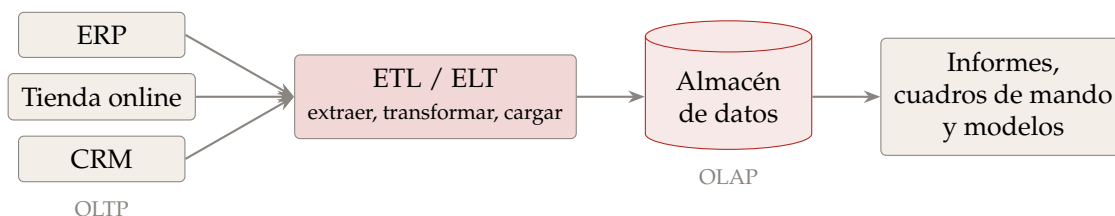


Figura 11. Flujo típico de datos para analítica: de los sistemas operativos al cuadro de mando.

Modelo en estrella. Una tabla central de *hechos* (ventas, con importes y cantidades) rodeada de tablas de *dimensiones* (fecha, cliente, producto, tienda). Es el diseño estándar para analítica.

Lago de datos. Almacén de ficheros en bruto (CSV, JSON, Parquet) a bajo coste, para procesarlos después.

10. BASES DE DATOS EN LA NUBE

Hoy la mayoría de bases de datos nuevas no se instalan en un servidor propio: se contratan como servicio en la nube. La gran diferencia entre las opciones es *quién se ocupa de qué*.

Servidor propio	Máquina virtual (IaaS)	Gestionada (DBaaS)	Serverless
Datos y consultas	Datos y consultas	Datos y consultas	Datos y consultas
Esquema e índices	Esquema e índices	Esquema e índices	Esquema e índices
Escalado y capacidad	Escalado y capacidad	Escalado y capacidad	Escalado y capacidad
Copias y parches	Copias y parches	Copias y parches	Copias y parches
Sistema operativo	Sistema operativo	Sistema operativo	Sistema operativo
Hardware y red	Hardware y red	Hardware y red	Hardware y red

lo gestionas tú
 lo gestiona el proveedor

Figura 12. Modelo de responsabilidad compartida: cuanto más a la derecha, más tareas asume el proveedor.

Máquina virtual (IaaS). Alquilas un servidor y tú instalas y mantienes la base de datos. Máximo control, máximo trabajo.

Gestionada (DBaaS). El proveedor instala, actualiza, hace copias y replica; tú diseñas y consultas. Es la opción habitual.

Serverless. No hay servidor que dimensionar: escala solo y se paga por uso. Ideal para cargas variables; puede salir cara con uso intensivo y constante.

10.1 Los servicios de los tres grandes proveedores

Necesidad	AWS	Microsoft Azure	Google Cloud
Relacional gestionada	RDS (PostgreSQL, MySQL, SQL Server, Oracle), Aurora	Azure SQL Database, Azure Database for PostgreSQL/MySQL	Cloud SQL, AlloyDB
Relacional global	Aurora Global Database	Cosmos DB (API PostgreSQL)	Spanner
Documental y clave-valor	DynamoDB, DocumentDB	Cosmos DB	Firestore, Bigtable
Caché en memoria	ElastiCache, MemoryDB	Azure Cache for Redis	Memorystore
Almacén analítico	Redshift	Microsoft Fabric, Synapse Analytics	BigQuery
Almacenamiento de objetos	S3	Blob Storage / ADLS	Cloud Storage
Consultas sobre ficheros	Athena	Fabric / Synapse serverless	BigQuery (tablas externas)
Integración (ETL)	Glue	Data Factory	Dataflow, Dataproc

10.2 Plataformas de datos: Databricks y Snowflake

Además de los servicios de cada nube, hay plataformas independientes que funcionan sobre cualquiera de ellas:

Databricks. Creada por los autores de Apache Spark. Popularizó el **lakehouse**: guardar todos los datos en ficheros baratos del almacenamiento de objetos (S3, ADLS, Cloud Storage) pero con las garantías de una base de datos gracias a un formato de tabla abierto, **Delta Lake**. Permite trabajar con SQL, Python y Spark sobre los mismos datos, entrenar modelos de aprendizaje automático y gobernar permisos y linaje con Unity Catalog.

Snowflake. Almacén de datos en la nube que separa almacenamiento y cómputo: varios equipos consultan los mismos datos con «almacenes virtuales» independientes que se encienden, escalan y apagan solos. Se trabaja casi todo con SQL y facilita compartir datos entre empresas.

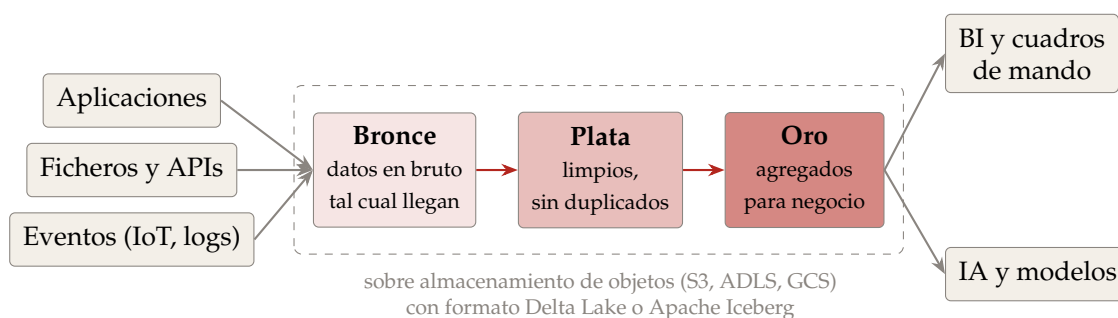


Figura 13. Arquitectura lakehouse con capas medallón: los datos ganan calidad en cada paso, siempre sobre el mismo almacenamiento.

10.3 La pila de datos moderna

Pieza	Herramientas habituales
Ingesta	Fivetran, Airbyte, AWS Glue, Azure Data Factory
Almacenamiento y cómputo	Databricks, Snowflake, BigQuery, Redshift, Microsoft Fabric
Transformación	dbt (modelos en SQL versionados y probados), Spark
Orquestación	Apache Airflow, Dagster, Databricks Workflows
Visualización	Power BI, Tableau, Looker, Metabase

10.4 Claves para elegir en la nube

- **Coste:** se paga por almacenamiento, cómputo y transferencia de datos. Activa alertas de presupuesto y apaga lo que no se use; una consulta mal escrita sobre terabytes se nota en la factura.
- **Ubicación:** con datos personales, elige regiones de la Unión Europea y revisa las condiciones de transferencia internacional (RGPD).
- **Dependencia del proveedor:** los formatos abiertos (PostgreSQL, Parquet, Delta, Iceberg) facilitan cambiar de nube más adelante.
- **Tamaño real:** para la mayoría de pymes, un PostgreSQL gestionado cubre años de crecimiento. Databricks o Snowflake tienen sentido cuando hay grandes volúmenes o muchos equipos analizando datos.

11. PRIMEROS PASOS CON DATABRICKS, SNOWFLAKE Y BIGQUERY

Las tres plataformas se manejan con SQL y comparten muchas ideas, pero cada una usa su propio vocabulario. Este apartado reúne lo mínimo para empezar a trabajar: cómo se organizan, cómo se cargan datos, cómo se consultan y cómo no llevarse un susto con la factura.

11.1 Mismas ideas, distinto nombre

Concepto	Databricks	Snowflake	BigQuery
Dónde se trabaja	<i>Workspace</i> : cuadernos, editor SQL	Snowsight: hojas de trabajo y cuadernos	Consola de Google Cloud
Motor de cálculo	Clúster o <i>SQL warehouse</i>	<i>Virtual warehouse</i>	Automático (<i>slots</i>)
Nombre de una tabla	<code>catalogo.esquema.tabla</code>	<code>base.esquema.tabla</code>	<code>proyecto.dataset.tabla</code>
Se paga por	DBU (unidad de cómputo) + nube	Créditos por segundo encendido	Datos leídos (TB) o capacidad
Lenguajes	SQL, Python, Scala, R	SQL, Python (Snowpark)	SQL, Python
Cargar ficheros	<code>COPY INTO</code> , Auto Loader	<code>Stage + COPY INTO</code> , Snowpipe	<code>LOAD DATA</code> , consola
Volver atrás en el tiempo	<code>VERSION AS OF</code>	<code>AT(OFFSET...)</code> , <code>UNDROP</code>	<code>FOR SYSTEM_TIME AS OF</code>
Permisos y gobierno	Unity Catalog	Roles (RBAC)	IAM de Google Cloud
Para probar gratis	Free Edition	Prueba de 30 días	Sandbox de BigQuery

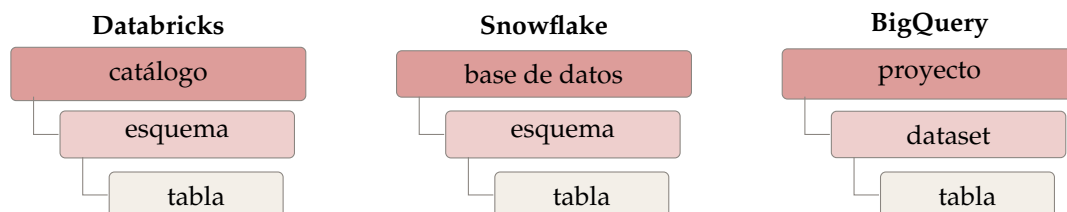


Figura 14. Las tres plataformas organizan las tablas en tres niveles; en las consultas conviene escribir siempre el nombre completo.

11.2 Databricks: lo básico

Workspace. El entorno web. Desde el menú lateral se accede a los cuadernos (*notebooks*), el editor SQL, el explorador de catálogo, los trabajos programados y el cómputo.

Cómputo. Para cuadernos se usa un **clúster** o el cómputo *serverless*; para consultas SQL e informes, un **SQL warehouse**. Activa siempre el apagado automático tras unos minutos sin uso.

Cuadernos. Mezclan celdas de Python, SQL, Scala o R (con `%sql` o `%python` al inicio de la celda) y muestran tablas y gráficos con `display()`.

Unity Catalog. Gobierna permisos, linaje y auditoría de todas las tablas, ficheros (*volumes*) y modelos.

Tablas Delta. El formato por defecto: admiten transacciones, UPDATE, MERGE y consultar versiones anteriores.

Workflows y Lakeflow. Programan cuadernos y consultas como trabajos y construyen tuberías de datos declarativas (antes llamadas Delta Live Tables).

```
-- Cargar un CSV subido a un volume y crear una tabla Delta
CREATE TABLE main.ventas.pedidos AS
SELECT * FROM read_files('/Volumes/main/ventas/bruto/pedidos.csv',
                        format => 'csv', header => true);

-- Actualizar o insertar segun exista (upsert)
MERGE INTO main.ventas.pedidos t
USING main.ventas.pedidos_nuevos s ON t.id = s.id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *;

-- Viaje en el tiempo: historial y version anterior
DESCRIBE HISTORY main.ventas.pedidos;
SELECT * FROM main.ventas.pedidos VERSION AS OF 3;
```

```
# PySpark en un cuaderno: leer, agregar y guardar
from pyspark.sql import functions as F

df = spark.read.table("main.ventas.pedidos")
resumen = (df.groupBy("cliente_id")
            .agg(F.sum("importe").alias("total"),
                 F.count("*").alias("n_pedidos")))
display(resumen)
resumen.write.mode("overwrite").saveAsTable("main.ventas.resumen_clientes")
```

11.3 Snowflake: lo básico

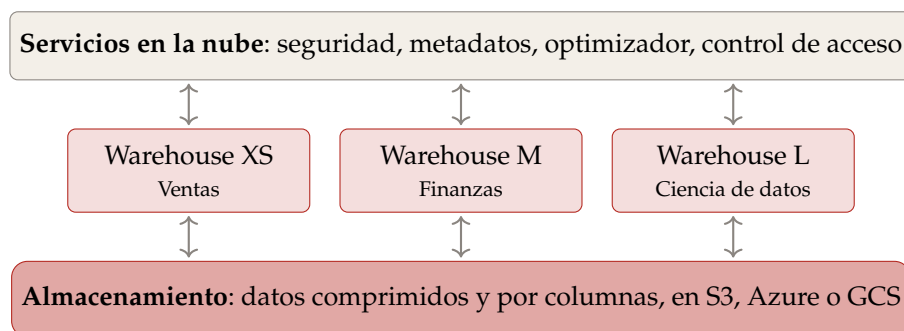


Figura 15. Arquitectura de Snowflake en tres capas: el almacenamiento es único y cada equipo usa su propio motor de cálculo.

Snowsight. La interfaz web: hojas de trabajo SQL (*worksheets*), cuadernos, carga de ficheros y monitor de consumo.

Virtual warehouse. El motor que ejecuta las consultas. Se elige talla (XS, S, M, L...), cada talla dobla potencia y coste, y se cobra por segundo mientras está encendido. Configura el autosuspendido y la reanudación automática.

Roles. Todo permiso se concede a un rol, y los roles a usuarios. ACCOUNTADMIN solo para administrar la cuenta; para el trabajo diario, SYSADMIN o roles propios.

Stages. Zonas donde se dejan los ficheros antes de cargarlos con COPY INTO. Snowpipe los carga automáticamente según lleguen.

Datos semiestructurados. El tipo VARIANT guarda JSON; sus campos se leen con dos puntos.

Clonado sin copia. CLONE crea al instante una copia de una tabla o base de datos completa sin duplicar el almacenamiento: ideal para pruebas.

```
-- Contexto de trabajo
USE ROLE SYSADMIN;
CREATE WAREHOUSE IF NOT EXISTS wh_analisis
  WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 60 AUTO_RESUME = TRUE;
USE WAREHOUSE wh_analisis;
CREATE DATABASE IF NOT EXISTS ventas;
USE SCHEMA ventas.public;

-- Cargar ficheros CSV desde un stage
CREATE STAGE IF NOT EXISTS bruto;
COPY INTO pedidos FROM @bruto/pedidos/
  FILE_FORMAT = (TYPE = 'CSV' SKIP_HEADER = 1);

-- JSON en una columna VARIANT
SELECT datos:cliente.nombre::STRING AS cliente, datos:importe::NUMBER(10,2)
FROM eventos;

-- Time Travel y clonado
SELECT * FROM pedidos AT(OFFSET => -60*30); -- hace 30 minutos
UNDROP TABLE pedidos;
CREATE TABLE pedidos_pruebas CLONE pedidos;
```

11.4 BigQuery: lo básico

- No hay servidores ni warehouses que encender: se escribe la consulta y Google asigna la capacidad.
- En el modo bajo demanda se paga por **datos leídos**. Antes de ejecutar, la consola indica cuántos bytes procesará la consulta.
- Las tablas **particionadas** (normalmente por fecha) y **agrupadas** (*clustered*) reducen lo que se lee y, por tanto, el coste.
- LIMIT no abarata una consulta: BigQuery lee igualmente las columnas completas. Lo que ahorra es seleccionar solo las columnas necesarias y filtrar por la partición.

```
CREATE TABLE miproyecto.ventas.pedidos
PARTITION BY DATE(fecha) CLUSTER BY cliente_id AS
SELECT * FROM miproyecto.bruto.pedidos;

SELECT cliente_id, SUM(importe) AS total
FROM miproyecto.ventas.pedidos
-- filtrar por la particion: solo se leen esas fechas
WHERE DATE(fecha) BETWEEN '2026-01-01' AND '2026-03-31'
GROUP BY cliente_id;
```

11.5 Tu primera hora en cualquiera de ellas

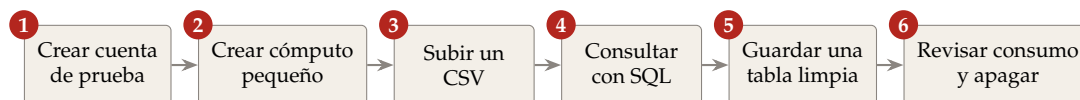


Figura 16. Recorrido de iniciación, válido para las tres plataformas.

11.6 Errores típicos al empezar

- Dejar un clúster o warehouse encendido toda la noche. Configura siempre el apagado automático y alertas de presupuesto.
- Usar el rol o usuario administrador para todo.
- Elegir una talla grande «para que vaya rápido» antes de comprobar si una pequeña basta.
- Hacer `SELECT *` sobre tablas enormes, sobre todo en BigQuery.
- Escribir solo el nombre corto de la tabla y consultar, sin darse cuenta, la de otro esquema.
- Copiar datos entre entornos en lugar de usar clonado o permisos.

Observación. La mejor forma de aprender es con un conjunto de datos propio y pequeño: unas ventas exportadas a CSV bastan para practicar carga, limpieza, agregación y un primer cuadro de mando.

12. SEGURIDAD Y COPIAS DE SEGURIDAD

12.1 Seguridad

- **Mínimo privilegio:** cada usuario o aplicación, solo con los permisos que necesita.
- **Inyección SQL:** nunca construir consultas pegando texto del usuario; usar siempre consultas parametrizadas.
- **Cifrado** en tránsito (TLS) y en reposo; contraseñas guardadas con funciones de hash específicas, nunca en claro.
- **Datos personales:** el RGPD obliga a protegerlos, limitar su conservación y poder borrarlos o exportarlos si el interesado lo pide.

```
-- MAL: vulnerable a inyeccion
"SELECT * FROM usuarios WHERE email = '" + email + "'"
-- BIEN: consulta parametrizada
SELECT * FROM usuarios WHERE email = $1;
```

12.2 Copias de seguridad: la regla 3-2-1

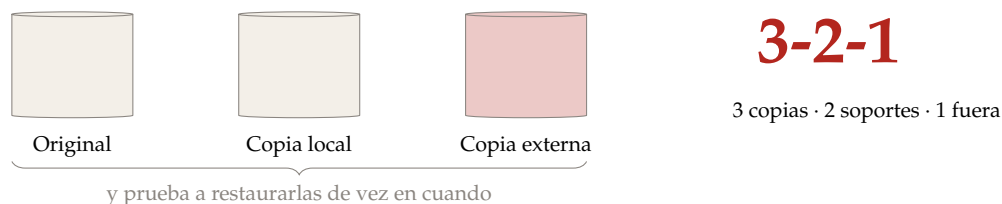


Figura 17. Regla 3-2-1: tres copias, en dos soportes distintos, una fuera de la oficina.

Observación. Una copia de seguridad que nunca se ha restaurado es una suposición, no una copia. Programa pruebas de restauración periódicas.

13. HERRAMIENTAS

Herramienta	Cuándo usarla
SQLite	Base de datos en un único fichero: aplicaciones móviles, prototipos, análisis local.
PostgreSQL	Relacional de código abierto muy completa; la opción por defecto para la mayoría de proyectos.
MySQL / MariaDB	Relacional muy extendida en webs y CMS como WordPress.
SQL Server, Oracle	Entornos corporativos con licencias comerciales.
MongoDB	Documentos JSON con estructura flexible.
Redis	Caché y datos temporales en memoria.
DBeaver, pgAdmin	Clientes gráficos para explorar y consultar bases de datos.

14. ERRORES FRECUENTES

- Guardar listas en una celda o duplicar datos en varias tablas.
- Usar coma flotante para importes de dinero.
- No definir claves foráneas ni restricciones y confiar en que la aplicación lo controle todo.
- Crear índices en todas las columnas «por si acaso», o en ninguna.
- Lanzar UPDATE o DELETE sin WHERE en producción.
- Concatenar texto del usuario en las consultas (inyección SQL).
- No probar nunca la restauración de las copias de seguridad.

15. LISTA DE COMPROBACIÓN

1. ¿Cada tabla tiene clave primaria?
2. ¿Las relaciones están definidas con claves foráneas?

3. ¿Está normalizado el diseño, al menos hasta 3FN?
4. ¿Los tipos de datos son los adecuados (fechas como fechas, dinero como DECIMAL)?
5. ¿Hay índices en las columnas de búsqueda y unión más usadas?
6. ¿Las operaciones relacionadas van dentro de transacciones?
7. ¿Las consultas de la aplicación son parametrizadas?
8. ¿Cada usuario tiene solo los permisos que necesita?
9. ¿Hay copias de seguridad automáticas y se prueba su restauración?

16. GLOSARIO

ACID. Atomicidad, consistencia, aislamiento y durabilidad: garantías de una transacción.

Clave foránea. Columna que apunta a la clave primaria de otra tabla.

Clave primaria. Identificador único de cada fila.

CTE. Subconsulta con nombre definida con WITH.

Databricks. Plataforma de datos e IA basada en Spark y en el modelo lakehouse.

DBU. Unidad de consumo con la que factura Databricks.

DBaaS. Base de datos como servicio: el proveedor la instala, actualiza y respalda.

Data warehouse. Almacén de datos optimizado para analítica.

Delta Lake / Iceberg. Formatos de tabla abiertos que añaden transacciones y versiones a ficheros en la nube.

ETL. Extraer, transformar y cargar datos entre sistemas.

Índice. Estructura que acelera las búsquedas a costa de espacio y escrituras.

Inyección SQL. Ataque que cuela código en una consulta construida con texto del usuario.

JOIN. Combinación de filas de varias tablas según una condición.

Lakehouse. Arquitectura que une lago de datos y almacén analítico sobre el mismo almacenamiento.

Medallón. Organización en capas bronce, plata y oro según la calidad del dato.

Notebook. Cuaderno interactivo que combina código, resultados y texto.

Normalización. Diseño que guarda cada dato una sola vez.

NoSQL. Bases de datos que no usan el modelo de tablas relacional.

NULL. Valor desconocido o ausente.

OLTP / OLAP. Procesamiento transaccional del día a día / procesamiento analítico.

Serverless. Servicio que escala solo y se paga por uso, sin servidores que dimensionar.

Stage. En Snowflake, zona donde se dejan los ficheros antes de cargarlos.

SGBD. Sistema gestor de bases de datos.

SQL. Lenguaje estándar para bases de datos relacionales.

Time Travel. Consulta o recuperación de los datos tal como estaban en un momento anterior.

Transacción. Conjunto de operaciones que se ejecutan todas o ninguna.

Unity Catalog. Capa de gobierno de datos de Databricks.

Virtual warehouse. En Snowflake, motor de cálculo que ejecuta las consultas.