Curso
Herramientas de Computación en la Nube

Clase 2

Febrero 7 de 2026

Modelos de datos y lenguajes de consulta

Profesor

Alvaro Mauricio Montenegro Díaz
Universidad de la Sabana

Febrero 2026

Fuente: Generado por ChatGPT

1. Objetivos de aprendizaje

Al finalizar esta lección, el estudiante será capaz de:

1. Diferenciar con precisión conceptual un **modelo de datos**, un **motor de base de datos** y un **formato de almacenamiento**.
2. Seleccionar el modelo de datos más adecuado en función del **patrón de acceso**, la naturaleza de las relaciones y los requisitos de escalabilidad.
3. Explicar los compromisos entre **normalización relacional**, **anidamiento documental** y **traversal en grafos**.
4. Formular la misma consulta conceptual utilizando distintos lenguajes (SQL, agregación documental, Cypher o SPARQL), identificando ventajas y limitaciones.
5. Argumentar cuándo es preferible un enfoque declarativo frente a uno imperativo.
6. Comprender que la persistencia políglota no es una moda tecnológica, sino una consecuencia de requisitos heterogéneos.

2. Introducción

Los **modelos de datos** constituyen la abstracción fundamental mediante la cual un sistema representa, organiza y accede a la información. La elección de un modelo de datos determina no solo la forma de almacenar la información, sino también **qué operaciones resultan naturales, eficientes o complejas**, y qué tipo de lenguajes de consulta se pueden utilizar. En consecuencia, el modelo de datos condiciona la arquitectura completa del sistema y el diseño del software que se construye sobre él.

3. Esquema en escritura vs esquema en lectura

Una de las decisiones arquitectónicas más importantes en sistemas de datos consiste en determinar **cuándo** y **dónde** se valida la estructura de la información.

3.1. Esquema en escritura (schema-on-write)

La estructura se valida en el momento de insertar los datos. Es característico del modelo relacional tradicional.

Ventaja: consistencia estructural fuerte desde el inicio.

Costo: menor flexibilidad ante cambios frecuentes del modelo.

3.2. Esquema en lectura (schema-on-read)

La estructura se aplica en el momento de consumir los datos. Es frecuente en modelos documentales y arquitecturas de data lake.

Ventaja: flexibilidad y evolución ágil.

Costo: mayor responsabilidad en la etapa de análisis y posible complejidad en validación tardía.

La elección entre ambos enfoques no es técnica sino estratégica. Determina cómo se distribuye la complejidad a lo largo del ciclo de vida del dato.

4. Modelos relacionales

El modelo relacional organiza la información en **relaciones (tablas)** compuestas por filas y columnas, y se apoya en un lenguaje **declarativo** para expresar consultas. Su principal fortaleza radica en:

- La separación entre **modelo lógico** y **representación física**.
- El uso de **joins** para modelar relaciones complejas (*many-to-one*, *many-to-many*).
- La normalización como mecanismo para reducir redundancia y mantener consistencia.

Este modelo resulta especialmente adecuado cuando el dominio presenta relaciones complejas y cuando se requiere flexibilidad para combinar datos de múltiples entidades en consultas arbitrarias.

5. Origen y significado de NoSQL

El término NoSQL no designa una tecnología única, sino una **familia heterogénea de sistemas** que emergen como respuesta a:

- Necesidades de **escalabilidad horizontal**.
- Preferencia por software **abierto y distribuido**.
- Patrones de acceso no cubiertos eficientemente por modelos relacionales tradicionales.
- Deseo de mayor flexibilidad en la gestión de esquemas.

NoSQL no implica el abandono del modelo relacional, sino la coexistencia de múltiples tecnologías, dando lugar a arquitecturas de **persistencia políglota**, donde cada componente utiliza el modelo más adecuado a su función.

6. Taxonomía operativa de NoSQL

6.1. Taxonomía operativa de sistemas NoSQL

El término “NoSQL” designa uuna familia heterogénea de enfoques. Entre los principales se encuentran:

Clave-valor (Key-Value)

Optimizado para acceso rápido por clave primaria.

Uso típico: cachés, sesiones, configuraciones.

Documental

Almacena agregados completos como documentos JSON/BSON.

Uso típico: aplicaciones web, microservicios con estructuras jerárquicas.

Columnar distribuido (Wide-Column)

Organiza datos por familias de columnas y favorece escrituras masivas y consultas por partición. *Uso típico:* sistemas de alta escalabilidad horizontal.

Grafos

Modela explícitamente relaciones como aristas navegables.

Uso típico: redes sociales, recomendación, análisis de dependencias.

La persistencia políglota surge cuando distintos dominios del mismo sistema requieren modelos distintos para funcionar eficientemente.

7. Modelos documentales

Los modelos documentales representan la información como **documentos autocontenidos**, generalmente estructurados en formatos jerárquicos (JSON, BSON, XML). Son apropiados cuando:

- El principal patrón de acceso consiste en leer o escribir **agregados completos**.

- Las relaciones internas del agregado son más importantes que las relaciones externas.
- Se prioriza la **localidad de los datos** y la simplicidad del acceso.

En este enfoque, se favorece el **anidamiento** sobre la normalización. Sin embargo, cuando el dominio presenta relaciones complejas entre documentos, la gestión de referencias puede trasladar complejidad a la aplicación.

8. Desajuste objeto–relacional

En sistemas desarrollados con paradigmas orientados a objetos surge un **desajuste conceptual** entre estructuras de memoria (objetos, colecciones, jerarquías) y estructuras relacionales (tablas y filas). Este fenómeno obliga a introducir capas de traducción, como ORMs, que mitigan pero no eliminan la diferencia conceptual. Los modelos documentales reducen este desajuste al permitir estructuras más cercanas a los objetos de la aplicación.

9. Relaciones y complejidad estructural

Los modelos relacionales manejan de forma nativa relaciones complejas mediante claves foráneas y joins. Los modelos documentales tienden a ser más eficientes cuando las relaciones son simples o se concentran dentro de un agregado. Cuando la densidad relacional crece, el modelo documental puede perder claridad o eficiencia si no se diseña cuidadosamente.

10. Convergencia entre modelos

En la práctica moderna se observa una **convergencia**:

- Sistemas relacionales incorporan soporte nativo para documentos (JSON/XML) con indexación y consulta.
- Sistemas documentales añaden capacidades de agregación, referencias y, en algunos casos, joins limitados.

Esto permite enfoques híbridos que combinan la flexibilidad documental con la expresividad relacional.

11. Modelos ≠ Motores ≠ Formatos

Modelos, motores y formatos: una distinción necesaria

En el estudio de bases de datos es fundamental distinguir tres niveles conceptuales que con frecuencia se confunden:

Modelo de datos

Es la abstracción lógica utilizada para representar y organizar la información. Ejemplos: modelo relacional, documental, grafo, clave-valor.

Motor o sistema gestor

Es la implementación concreta que materializa un modelo y lo hace operativo. Ejemplos: PostgreSQL, MongoDB, Neo4j.

Formato de almacenamiento

Es la representación física o semiestructurada en la que los datos son almacenados o intercambiados. Ejemplos: JSON, Parquet, ORC.

Confundir estos niveles conduce a errores conceptuales frecuentes, como considerar que un formato es un modelo o que un motor define necesariamente una única forma de estructurar datos.
La arquitectura correcta comienza por elegir el **modelo**, luego el **motor**, y finalmente optimizar el **formato** según los patrones de acceso.

12. Lenguajes de consulta: declarativos vs imperativos

Los lenguajes **imperativos** describen paso a paso cómo obtener un resultado, mientras que los lenguajes **declarativos** especifican únicamente qué resultado se desea. Las ventajas de los enfoques declarativos incluyen:

- Mayor independencia del plan de ejecución.
- Posibilidad de optimización automática.
- Mejor paralelización y razonamiento formal.

Esta filosofía se extiende más allá de las bases de datos y aparece en otros dominios como hojas de estilo y lenguajes de transformación.

13. Procesamiento distribuido y MapReduce

MapReduce introduce un modelo de procesamiento basado en funciones de transformación y reducción, adecuado para grandes volúmenes de datos distribuidos. Aunque poderoso, su naturaleza de bajo nivel ha dado lugar a lenguajes y frameworks de mayor nivel que combinan expresividad declarativa con control sobre el procesamiento.

14. Modelos de grafos

Cuando el dominio está dominado por **relaciones densas y variables**, los modelos de grafos resultan más naturales. Estos representan entidades como nodos y relaciones como aristas, permitiendo consultas basadas en patrones de conexión. Existen dos enfoques principales:

- **Grafos con propiedades**, orientados a consultas declarativas de patrones.
- **Almacenes de triples**, basados en relaciones sujeto–predicado–objeto y consultas declarativas formales.

Estos modelos reducen la fricción al modelar dominios altamente interconectados, como redes sociales, rutas, dependencias o conocimiento semántico.

15. Property Graph vs RDF

Dos enfoques principales en bases de datos de grafos

No todas las bases de datos de grafos implementan el mismo modelo conceptual.

15.1. Property Graph

Representa nodos y aristas con propiedades asociadas. Las consultas se formulan mediante patrones estructurales (por ejemplo, MATCH en Cypher). Es especialmente eficiente para recorridos complejos y análisis transaccional.

15.2. RDF (Resource Description Framework)

Modela la información como triples sujeto-predicado-objeto. El lenguaje SPARQL permite consultas declarativas sobre patrones de triples. Es especialmente útil en integración semántica, ontologías y web de datos. Ambos modelos representan grafos, pero responden a necesidades distintas: uno optimiza traversal práctico; el otro prioriza interoperabilidad semántica.

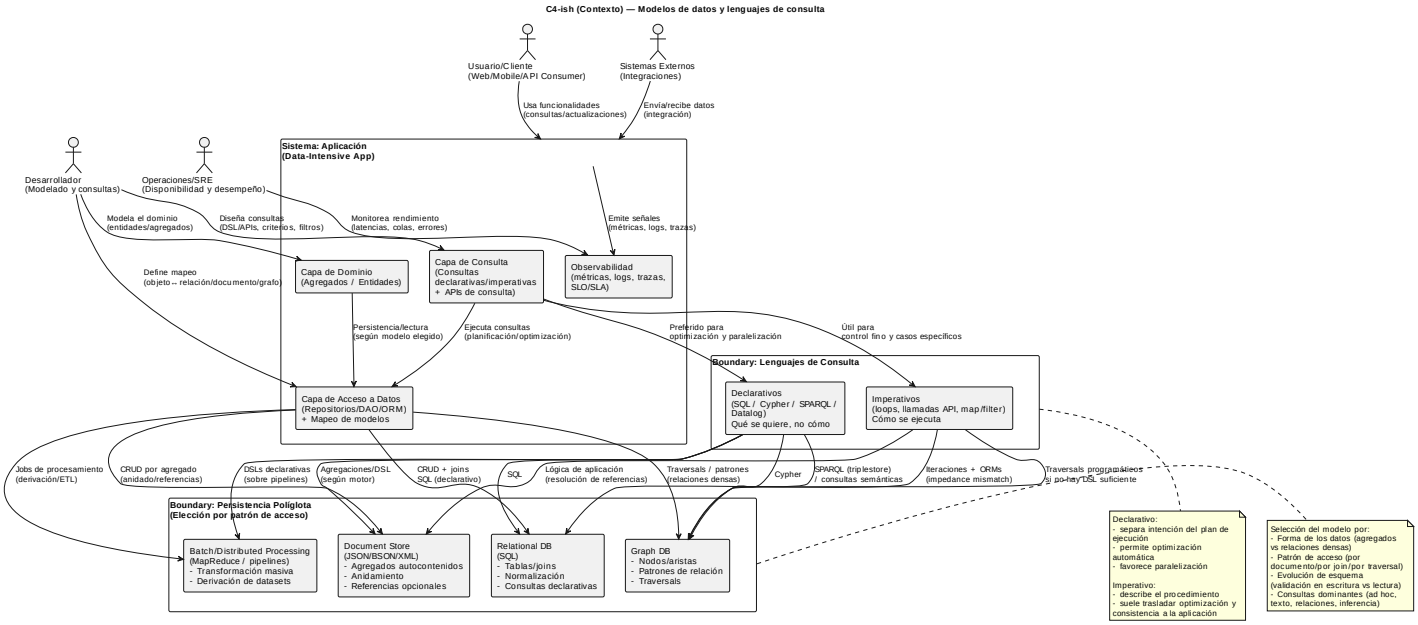
16. Datalog como fundamento teórico

Datalog es un lenguaje declarativo basado en lógica de primer orden restringida. A diferencia de SQL o Cypher, no se centra únicamente en recuperar datos, sino en **derivar hechos nuevos a partir de reglas**. unque menos práctico para consultas simples, resulta potente para dominios donde la inferencia y la relación entre datos son centrales.

Su estructura básica se compone de:

- **Hechos (facts)**
- **Reglas (rules)**
- **Consultas (queries)**

17. Diagrama conceptual: Modelos de datos y lenguajes de consulta



Fuente: Código fuente en formato puml generado por ChatGPT

18. Ejemplo: Mini-sistema: “Mini-CV / Perfil profesional”

Dominio mínimo (pero realista):

- **Persona**: id, nombre, ciudad, rol actual

- **Habilidades**: nombre, nivel
- **Experiencia**: empresa, cargo, inicio, fin
- **Educación**: institución, título, año
- **Proyectos**: nombre, tecnologías usadas

Relaciones clave

- Persona **tiene** muchas habilidades
- Persona **tuvo** muchas experiencias
- Persona **realizó** muchos proyectos
- Proyecto **usa** muchas tecnologías (habilidades)

Datos de ejemplo (pequeños)

- Persona: “Ana Pérez”, Bogotá, “Data Engineer”
- Habilidades: SQL (avanzado), Python (intermedio), Kafka (básico)
- Experiencia: (Imanku, Data Engineer, 2023–Actual), (ACME, Analyst, 2021–2023)
- Proyecto: “ETL Fiscalía”, usa: Python, SQL, Kafka

18.1. Dataset exacto (único, consistente)

Persona

- **P1**: Ana Pérez — Ciudad: Bogotá — Rol actual: Data Engineer

Habilidades/Tecnologías (mismo catálogo para los 3 modelos)

- **T1**: SQL
- **T2**: Python
- **T3**: Kafka

Experiencia (para contexto; no es necesaria para la consulta principal)

- **E1**: Imanku — Data Engineer — 2023-01 a presente
- **E2**: ACME — Analyst — 2021-01 a 2022-12

Proyecto

- **PR1**: “ETL Fiscalía” — persona: Ana
    - Tecnologías usadas: SQL, Python, Kafka

Consulta transversal (objetivo):

> “Listar los proyectos de Ana y las tecnologías usadas en cada proyecto.”

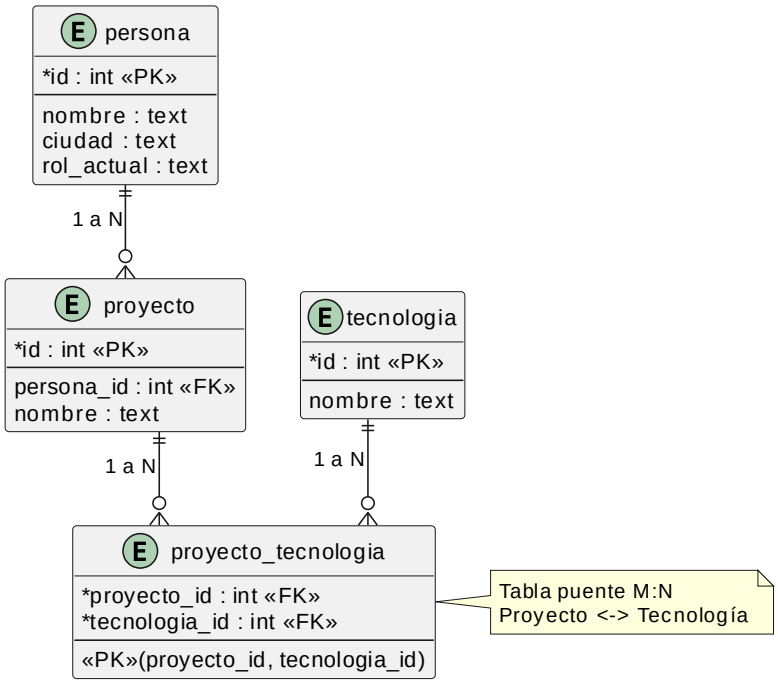
18.2. Modelo Relacional (SQL)

Tablas típicas:

- `persona(id, nombre, ciudad, rol_actual)`
- `proyecto(id, persona_id, nombre)`
- `habilidad(id, nombre)`
- `proyecto_habilidad(proyecto_id, habilidad_id)`

Consulta: `JOIN persona -> proyecto -> proyecto_habilidad -> habilidad`

Mini-CV — Modelo Relacional (ER-ish)



Fuente: Generado por ChatGPT

Datos (ejemplo)

- persona: (1, 'Ana Pérez', 'Bogotá', 'Data Engineer')
- proyecto: (10, 1, 'ETL Fiscalía')
- tecnologia: (100,'SQL'), (101,'Python'), (102,'Kafka')
- proyecto_tecnologia: (10,100), (10,101), (10,102)

Consulta SQL equivalente

```
SELECT
  p.nombre  AS persona,
  pr.nombre AS proyecto,
  t.nombre  AS tecnologia
FROM persona p
JOIN proyecto pr
  ON pr.persona_id = p.id
JOIN proyecto_tecnologia pt
  ON pt.proyecto_id = pr.id
JOIN tecnologia t
  ON t.id = pt.tecnologia_id
WHERE p.nombre = 'Ana Pérez'
ORDER BY pr.nombre, t.nombre;
```

Resultado esperado (conceptual):

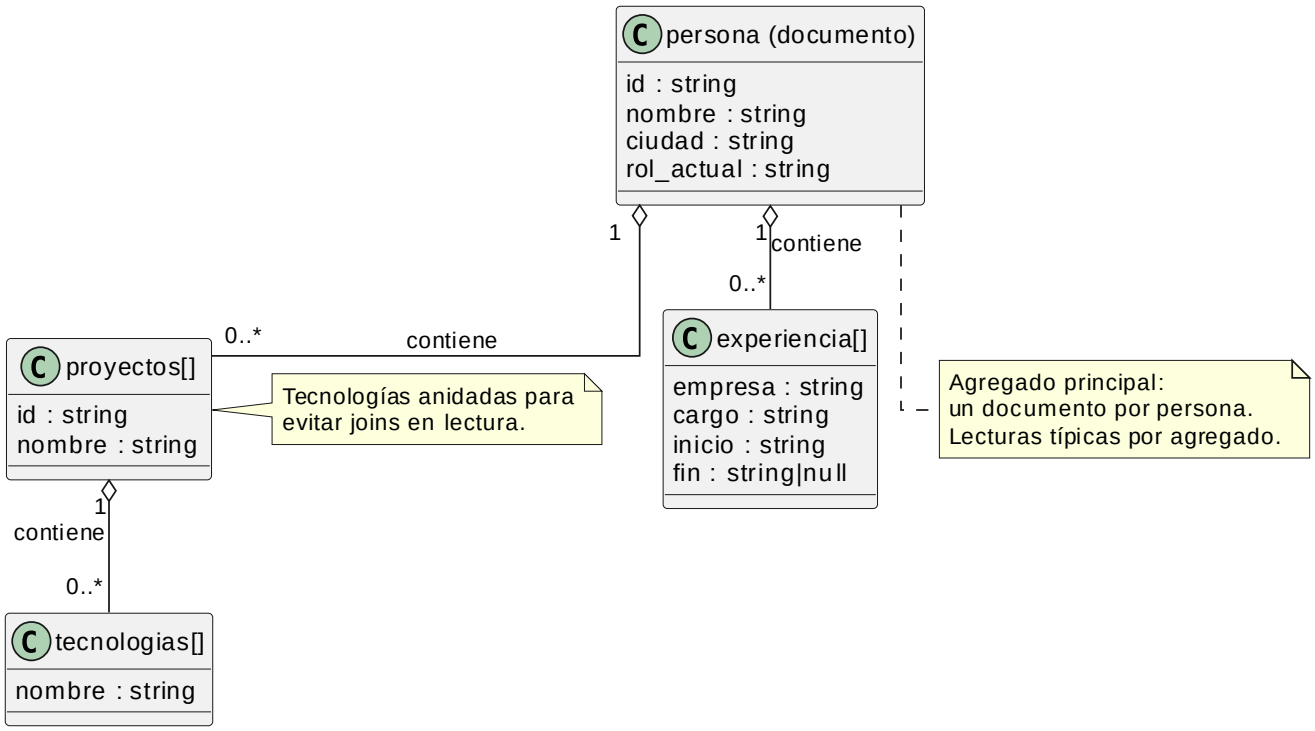
- Ana Pérez | ETL Fiscalía | Kafka
- Ana Pérez | ETL Fiscalía | Python
- Ana Pérez | ETL Fiscalía | SQL

18.3. Modelo documental (JSON)

Documento `persona` (agregado) con:

```
  "id": "P1",
  "nombre": "Ana Pérez",
  "ciudad": "Bogotá",{
  "rol_actual": "Data Engineer",
  "proyectos": [
    {
      "id": "PR1",
      "nombre": "ETL Fiscalía",
      "tecnologias": ["SQL", "Python", "Kafka"]
    }
  ],
  "experiencia": [
    {"empresa": "Imanku", "cargo": "Data Engineer", "inicio": "2023-
01", "fin": null},
    {"empresa": "ACME", "cargo": "Analyst", "inicio": "2021-01",
"fin": "2022-12"}
  ]
}
```

Mini-CV — Modelo Documental (Estructura JSON)



Fuente: Generado por ChatGPT

Consulta equivalente (estilo conceptual)

En documental, típicamente es: **leer el documento de Ana** y proyectar:

- `proyectos[].nombre`

- `proyectos[].tecnologias[]`

Ejemplo estilo MongoDB (solo ilustrativo):

```
db.personas.aggregate([
  { $match: { nombre: "Ana Pérez" } },
  { $unwind: "$proyectos" },
  { $unwind: "$proyectos.tecnologias" },
  {
    $project: {
      _id: 0,
      persona: "$nombre",
      proyecto: "$proyectos.nombre",
      tecnologia: "$proyectos.tecnologias"
    }
  },
  { $sort: { proyecto: 1, tecnologia: 1 } }
]);
```

18.4. Modelo Grafo (Cypher / SPARQL)

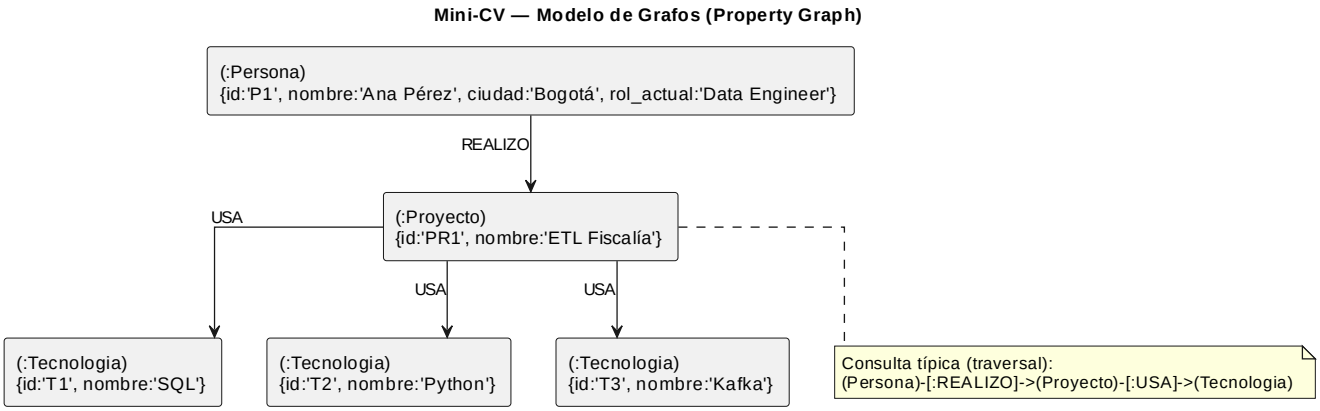
Nodos:

- (:Persona {nombre:"Ana"})
- (:Proyecto {nombre:"ETL Fiscalía"})
- (:Tecnologia {nombre:"Kafka"})

Aristas:

- (Persona)-[:REALIZO]->(Proyecto)
- (Proyecto)-[:USA]->(Tecnologia)

Consulta: patrón `(:Persona)-[:REALIZO]->(:Proyecto)-[:USA]->(:Tecnologia)`



Fuente: Generado por ChatGPT

Consulta equivalente (Cypher)

```
MATCH (p:Persona {nombre: "Ana Pérez"})-[:REALIZO]->(pr:Proyecto)-[:USA]->(t:Tecnologia)
RETURN p.nombre AS persona, pr.nombre AS proyecto, t.nombre AS tecnologia
ORDER BY proyecto, tecnologia;
```

18.5. Modelo Datalog

Un programa Datalog describe relaciones mediante cláusulas del tipo:

cabeza :- cuerpo.

Supongamos que tenemos las siguientes relaciones:

```
persona("Ana").
habilidad("Ana", "Python").
habilidad("Ana", "SQL").
experiencia("Ana", "ACME").
```

Podemos definir una regla:

```
  perfil_analitico(X) :-
   habilidad(X, "Python"),
   habilidad(X, "SQL").
```

Consulta

```
perfil_analitico(X)?
```

Resultado

X = "Ana"

18.6. Comparación conceptual de lenguajes de consulta

Lenguaje	Enfoque principal
SQL	Álgebra relacional
Cypher	Pattern matching en grafos
SPARQL	Pattern matching sobre triples RDF
Datalog	Reglas lógicas y derivación

19. Tecnologías reales por cada tema abordado

19.1. Modelo Relacional

Motores principales:

- PostgreSQL
- MySQL
- Oracle Database
- Microsoft SQL Server

Lenguaje:

- SQL (ANSI SQL)
- SQL recursivo (WITH RECURSIVE)

Extensiones modernas:

- JSONB en PostgreSQL
- Window functions
- Materialized views

19.2. Modelo Documental

Motores:

- MongoDB
- Couchbase
- Firebase Firestore

- Amazon DocumentDB

Lenguaje:

- MongoDB Query Language
- Aggregation Pipeline

Formatos asociados:

- JSON
- BSON

19.3. Modelo Clave-Valor

Motores:

- Redis
- Riak
- Amazon DynamoDB (modo KV)
- etcd

Uso típico:

- caché
- sesiones
- configuraciones distribuidas

19.4. Modelo Wide-Column (Columnar Distribuido)

Motores:

- Apache Cassandra
- HBase
- ScyllaDB

Lenguaje:

- CQL (Cassandra Query Language)

19.5. Modelo de Grafos (Property Graph)

Motores:

- Neo4j
- Amazon Neptune (modo property graph)
- TigerGraph

- ArangoDB

Lenguaje:

- Cypher
- GQL (estándar emergente)

19.6. Modelo RDF / Triplestore

Motores:

- Apache Jena
- Virtuoso
- Blazegraph
- Stardog

Lenguaje:

- SPARQL

19.7. Procesamiento Distribuido / MapReduce

Primera generación:

- Hadoop MapReduce

Evolución:

- Apache Spark
- Apache Flink
- Apache Beam

19.8. Formatos Columnar Analítico

(No son modelos, pero son clave en arquitectura moderna)

- Parquet
- ORC
- Avro

19.9. Sistemas basados en Datalog

- Soufflé
- Datomic
- DataScript
- Differential Datalog

- Materialize

20. Errores típicos al elegir modelo de datos

1. **Usar un modelo relacional para relaciones altamente dinámicas y densas**, generando consultas excesivamente complejas o costosas.
2. **Anidar documentos indiscriminadamente**, produciendo duplicación masiva y problemas de actualización.
3. **Elegir grafos cuando las consultas son predominantemente agregaciones simples**, donde un modelo tabular sería suficiente.
4. **Ignorar los patrones reales de acceso**, diseñando el esquema en función de la estructura conceptual y no del uso operativo.
5. **Optimizar prematuramente para escalabilidad sin comprender la carga real**.

La selección del modelo no es un acto ideológico; es una decisión basada en patrones de consulta, volumen, frecuencia y evolución esperada.

21. Ejercicios adicionales sobre el Mini-CV

Ejercicios adicionales

1. Formular una consulta que identifique todas las personas que utilizan la tecnología “Kafka”, en los tres modelos estudiados.
2. Determinar cuál es la tecnología más frecuente en el conjunto de datos.
3. Analizar qué modelo resulta más expresivo y cuál más eficiente para consultas de agregación.
4. Identificar qué índices serían necesarios en cada modelo para optimizar la consulta principal.

22. Síntesis conceptual

No existe un modelo de datos universalmente superior. Toda elección es contextual y depende de variables estructurales y operativas tales como:

- la naturaleza y morfología de los datos,
- los patrones reales de acceso,

- la frecuencia y dirección del cambio en el esquema,
- y el tipo de consultas que dominan el sistema.

La cuestión fundamental no es la existencia o ausencia de un esquema, sino **el momento y el lugar en que este se valida**: en la escritura o en la lectura. Esta decisión distribuye la complejidad a lo largo del ciclo de vida del dato y condiciona la arquitectura completa.

En sistemas bien concebidos, los modelos de datos y los lenguajes de consulta no se eligen por preferencia tecnológica, sino como instrumentos complementarios dentro de una arquitectura coherente, consciente de sus compromisos y limitaciones.

Los modelos no compiten entre sí; responden a problemas distintos:

- El modelo **relacional** privilegia consistencia estructural y precisión algebraica en operaciones complejas.
- El modelo **documental** favorece agregados autocontenidos y evolución flexible del esquema.
- El modelo **de grafos** optimiza la representación y exploración explícita de relaciones densas o variables.

La decisión arquitectónica central no consiste en determinar qué modelo es “mejor”, sino en identificar **qué patrón de acceso gobierna el sistema** y dónde resulta más razonable concentrar la complejidad: en la escritura, en la lectura o en la navegación de relaciones.

En este sentido, la persistencia políglota no debe interpretarse como una tendencia tecnológica, sino como una consecuencia natural de la heterogeneidad funcional que caracteriza a los sistemas modernos.

23. Diccionario de términos usados

- **Modelo de datos**: abstracción para representar/organizar información y definir operaciones naturales sobre ella
- **Modelo relacional**: representación en tablas (relaciones) con filas/columnas; fuerte en joins y normalización
- **Normalización**: técnica para reducir redundancia y preservar consistencia mediante descomposición en tablas y claves

- **Join**: operación que combina relaciones según claves, esencial para relaciones complejas many-to-many

- **Modelo documental**: almacenamiento como documentos autocontenidos jerárquicos (JSON/BSON/XML), optimizado para lectura/escritura por agregado

- **Agregado (aggregate)**: unidad coherente de lectura/escritura que agrupa datos estrechamente relacionados (en documental, suele ser 1 documento)

- **Anidamiento (embedding)**: inclusión de subestructuras dentro del documento para evitar joins a costa de duplicación potencial

- **NoSQL**: familia heterogénea de sistemas orientados a escalabilidad horizontal y esquemas flexibles; no reemplaza lo relacional, convive con él

- **Persistencia políglota**: arquitectura que usa múltiples modelos/motores según patrón de acceso y tipo de consulta

- **Desajuste objeto–relacional (impedance mismatch)**: fricción entre estructuras OO (objetos) y relacionales (tablas) que motiva capas ORM

- **ORM**: capa de mapeo objeto↔tabla que reduce fricción pero no elimina diferencias conceptuales

- **Lenguaje declarativo**: especifica *qué* resultado se quiere; el motor decide *cómo* ejecutarlo, habilitando optimización

- **Lenguaje imperativo**: especifica *cómo* obtener el resultado paso a paso; traslada control/optimización a la aplicación

- **MapReduce**: modelo distribuido map + reduce para grandes volúmenes; de bajo nivel, dio paso a frameworks más expresivos

- **Modelo de grafos**: entidades como nodos y relaciones como aristas; ideal para relaciones densas y variables

- **Property Graph**: grafo con propiedades en nodos/aristas y consultas por patrones (p.ej., Cypher)

- **Triplestore (RDF)**: grafo como triples sujeto-predicado-objeto; consultas SPARQL; fuerte en semántica e integración

- **SPARQL**: lenguaje estándar W3C para consultar RDF mediante patrones de grafo
- **Cypher**: lenguaje declarativo para consultas por patrones en property graphs; usa MATCH/RETURN/ORDER BY
- **Datalog**: lenguaje basado en reglas para consultas lógicas e inferencia, composable y reutilizable
- **Convergencia de modelos**: incorporación cruzada de capacidades (SQL con JS agregaciones/joins limitados)

24. Bibliografía

Núcleo

- Martin Kleppmann. **Designing Data-Intensive Applications**. O'Reilly, 2017.

Modelos y teoría (relacional / lógica / fundamentos)

- Serge Abiteboul, Richard Hull, Victor Vianu. **Foundations of Databases**. Addison-Wesley, 1995. *(fundamental para Datalog/lógica de bases de datos; clásico de doctorado)*
- Jennifer Widom (Stanford). **Database Systems** (notas/lecturas).

Grafos (práctica y patrones)

- Ian Robinson, Jim Webber, Emil Eifrem. **Graph Databases** (O'Reilly, varias ediciones). *(texto canónico de Neo4j y diseño de grafos)*
- Neo4j. **Cypher Manual (MATCH, RETURN, ORDER BY)**.

Web semántica / RDF

- W3C. **SPARQL 1.1 Query Language (Recommendation)**.

Procesamiento distribuido (origen y evolución)

- Jeffrey Dean & Sanjay Ghemawat. **MapReduce: Simplified Data Processing on Large Clusters** (OSDI 2004).
- Matei Zaharia et al. **Spark: Cluster Computing with Working Sets**.

Documental / JSON / agregación

- MongoDB Manual. **$unwind (Aggregation Stage)**.

Relacional moderno híbrido (SQL + JSONB)

- PostgreSQL Docs (current). **JSON Types / jsonb indexing (GIN)**.

Lakehouse / formatos analíticos

- Armbrust et al. **Lakehouse: A New Generation…** (CIDR 2021).