Course: DBI202 - Database Management Systems

<LAB 1: STUDY OF DATA MODELS>

Group: 1	
Group Members	<Trần Minh Huy > - <SE203499 > <Lê Hoàng Bách(Leader)> - <SE201089> <Nguyễn Thư Kỳ> - <SE203407 > <Trần Thành Đạt> - <SE203479 >
Lecturer	VanTTN
Class	AI2009

Ho Chi Minh, January 9 - 2026

Table of Contents

1. Objective

The objective of this laboratory is to study and understand the concepts of modern data models used in database management systems.
The lab focuses on how data is organized, stored, constrained, and manipulated under different modeling approaches.

Through this study, students gain insight into the strengths, limitations, and suitable application scenarios of various data models in real-world systems.
The knowledge gained from this lab provides a theoretical foundation for designing databases and selecting an appropriate data model for practical projects, including the group project on a movie ticket booking system.

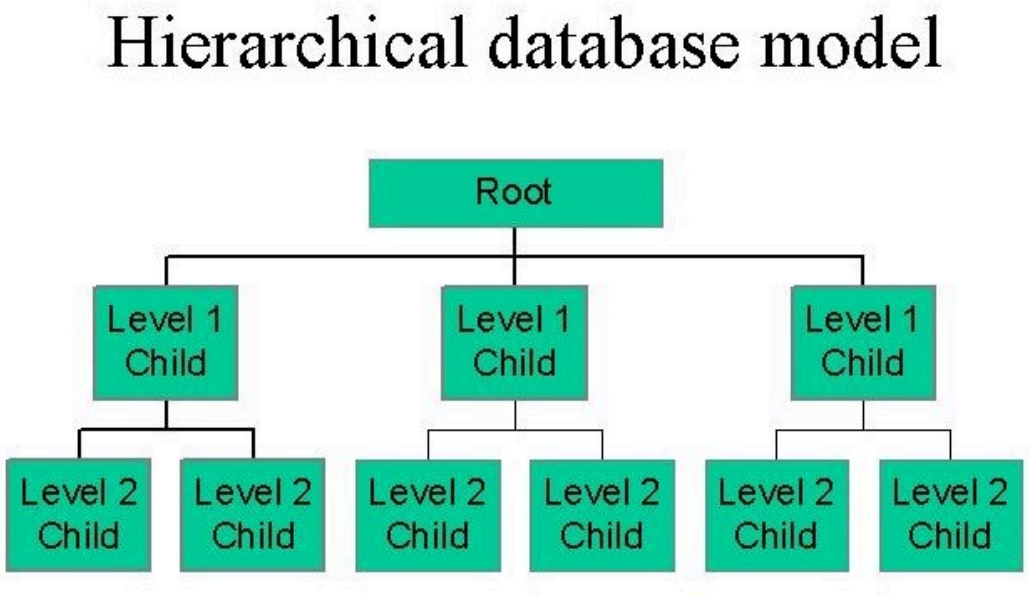
2. Core Data Models

2.1. Classical Data Models (Early DBMS Era)

Before the relational data model became dominant in the late 1970s and early 1980s, early database management systems relied on classical data models to organize, store, and retrieve data efficiently.
These models were designed in an era when computing resources were limited, storage devices were slow, and system efficiency was the primary concern.

Among these classical models, the Hierarchical Data Model and the Network Data Model were the most influential.
They were widely applied in large-scale enterprise and government systems and played a crucial role in shaping fundamental database concepts such as data organization, data integrity, and navigational access methods.

2.1.1. Hierarchical Data Model



1. General Concept

The Hierarchical Data Model organizes data in a tree-like structure, where records are arranged into multiple levels according to predefined relationships. Each record type represents a node in the hierarchy, and relationships between records strictly follow a parent–child pattern.

In this model:

- Each child record is associated with exactly one parent record
- Each parent record may have multiple child records
  This creates a one-to-many relationship that is enforced at the structural level of the database.

Data access in the hierarchical model is navigational, meaning that users and application programs must begin at the root record and traverse the hierarchy step by step through predefined paths.
As a result, this model is particularly suitable for data that naturally follows a hierarchical structure, such as organizational hierarchies, directory structures in file systems, and product category classifications.

2. Historical Development

The hierarchical data model was developed in the 1960s, during the earliest phase of database system development.
Its most significant and widely used implementation is IBM Information Management System (IMS).

IBM IMS was originally developed to support data management for the Apollo space program, where reliability, performance, and predictability were critical.
Later, IMS was adopted extensively in banking, insurance, and government systems, where large volumes of structured data needed to be processed efficiently.

At that time, hardware limitations such as limited memory, slow processors, and sequential storage devices strongly influenced database design.
The hierarchical model was favored because it enabled fast, deterministic data access, especially when data was stored on magnetic tapes or early disk systems.

3. Storage Structure

In a hierarchical database, data is stored as a collection of record types organized in a strict tree structure:

- Root record: The top-level record that has no parent and serves as the entry point for all data access.
- Parent records: Records that can have one or more child records.
- Child records: Records that depend on exactly one parent record.
- Pointers: Logical or physical links that connect parent records to their child records.

To improve performance, child records are often stored physically close to their parent records on storage media.
Each record has only one access path, which simplifies navigation and reduces search overhead but significantly limits flexibility when accessing data in alternative ways.

4. Constraints

The hierarchical data model enforces several strict structural constraints to maintain data integrity:

- Each child record must have only one parent
- Many-to-many relationships are not supported
- All access paths must be defined in advance during database design

- Deleting a parent record often triggers cascading deletions of all dependent child records
- Modifying the hierarchy may require reorganizing the entire database

These constraints ensure strong consistency and efficient access but make the model rigid and difficult to adapt to evolving business requirements.

5. Data Manipulation

Data manipulation in hierarchical databases is based on procedural and navigational operations, rather than declarative query languages.

Typical operations include:

- Navigating from the root record to a specific target record
- Sequentially traversing parent–child relationships
- Inserting new child records under a designated parent
- Updating existing records by following predefined paths
- Deleting parent records along with all associated child records

Because all operations depend on predefined access paths, ad-hoc queries that do not follow the hierarchy are difficult or inefficient to perform.

6. Advantages and Limitations

Advantages:

- Simple and intuitive data organization
- High performance for predictable hierarchical access patterns
- Efficient handling of one-to-many relationships
- Strong data consistency enforced by structural rules

Limitations:

- Inflexible schema design
- Inability to represent many-to-many relationships
- Tight coupling between database structure and application logic
- High maintenance cost when application requirements change

7. Suitable Application

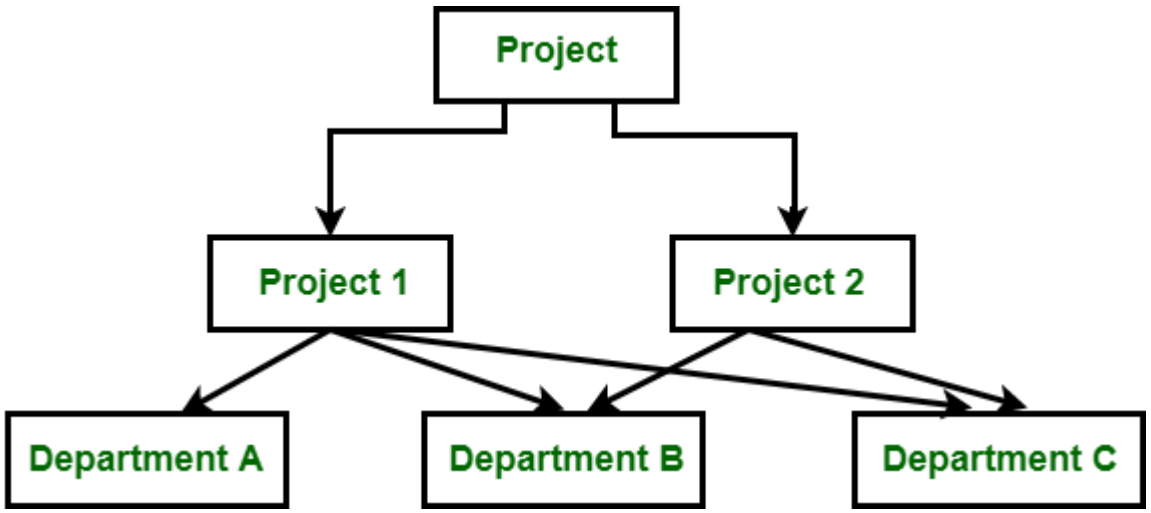
The hierarchical data model is most suitable for applications in which data naturally follows a strict parent–child structure and access patterns are predictable.

Typical applications include:

- Banking systems, where data is organized in fixed hierarchies such as Customer → Account → Transaction.
- Enterprise and government systems that manage structured records with stable relationships.
- File systems and directory management, where folders and files form a clear tree structure.
- Organizational management systems, such as company departments and employee hierarchies.
- Product categorization systems, where products are grouped into predefined categories and subcategories.

These applications benefit from the model's high performance and strong consistency when navigating hierarchical relationships.

2.1.2. Network Data Model



1. General Concept

The Network Data Model extends the hierarchical model by allowing records to have multiple parent records.
Instead of a strict tree, data is represented as a graph structure, where nodes represent records and edges represent relationships.

This model supports many-to-many relationships, enabling more accurate modeling of complex real-world scenarios.
As a result, it is suitable for applications such as airline reservation systems, manufacturing control systems, and telecommunications networks.

2. Historical Development

The network data model was developed in the late 1960s and early 1970s by the CODASYL Database Task Group (DBTG).
Its primary goal was to overcome the limitations of the hierarchical model, particularly the inability to represent complex relationships.

CODASYL-based database systems became widely used in enterprise environments and remained popular until relational databases gained dominance in the 1980s.

3. Storage Structure

The network data model organizes data using the following components:

- Record types: Represent entities stored in the database.
- Set types: Define relationships between record types.
- Owner records: Records that control a relationship.
- Member records: Records that participate in one or more relationships.
- Pointers: Used to connect records within sets.

A single record can belong to multiple sets, enabling flexible representation of many-to-many relationships.

4. Constraints

The network data model enforces several constraints:

- All relationships must be explicitly defined using set types
- Navigation paths must be known before execution
- Pointer integrity must be strictly maintained
- Deleting a record requires updating all related pointers
- Schema changes often require modifying application programs

These constraints increase expressive power but significantly raise system complexity.

5. Data Manipulation

Data manipulation in network databases relies on explicit navigational commands, including:

- Locating owner records

- Traversing member records through defined sets
- Inserting records into multiple sets
- Deleting records while maintaining pointer consistency
- Navigating complex relationship paths to retrieve related data

Because traversal logic is embedded in application programs, development and maintenance are more difficult compared to relational systems.

6. Advantages and Limitations

Advantages:

- Supports many-to-many relationships
- More flexible than hierarchical models
- Efficient handling of highly interconnected data
- Reduced data redundancy

Limitations:

- Complex database design and maintenance
- Strong dependency between database structure and application code
- Difficult for end users to understand and query
- Limited support for ad-hoc querying

7. Suitable Applications

The network data model is suitable for applications that require the representation of complex relationships, especially many-to-many associations between data entities.

Common applications include:

- Airline reservation systems, where passengers, flights, tickets, and reservations are highly interconnected.
- Manufacturing systems, which manage relationships between products, components, suppliers, and production processes.
- Telecommunication systems, where devices, users, and network connections form complex relationship networks.
- Logistics and supply chain systems, involving suppliers, warehouses, transportation routes, and delivery schedules.
- Enterprise information systems that require flexible relationship modeling beyond strict hierarchies.

These applications benefit from the network model's ability to efficiently manage highly connected data structures.

2. 2. Structured Data Model (Relational Era)

Representative: Relational Database Management System (RDBMS)

2.2.1. Relational Data Model

1. Storage Structure

In the structured data model, information is organized into tables, traditionally referred to as relations. Each relation consists of rows and columns, where rows represent individual records and columns define the attributes of those records. Before data can be inserted, the structure of each table must be defined in advance, including attribute names, types, and constraints. This design approach is commonly described as *schema-on-write*.

On the physical level, most relational systems store data on disk in the form of pages. To support efficient data retrieval, RDBMSs typically employ B+ Tree indexes. In a B+ Tree, actual data entries reside in the leaf nodes, which are linked to one another. This structure provides strong support for range queries and ordered scans.

2. Constraints

The relational model relies on several integrity rules to guarantee the accuracy and consistency of stored data:

- **Entity Integrity:** Every table includes a primary key that uniquely distinguishes each row. Primary key values cannot be null.
- **Referential Integrity:** Foreign keys ensure that relationships between tables remain valid. For example, an enrollment record cannot reference a student who does not exist in the Students table.
- **Domain Integrity:** Each attribute must follow the defined data type and any additional conditions, such as numerical ranges.

3. Data Manipulation Operators

Data manipulation within the relational model is conducted using relational algebra and SQL. Common operations include:

- Selection: Retrieves rows that meet a specified condition.

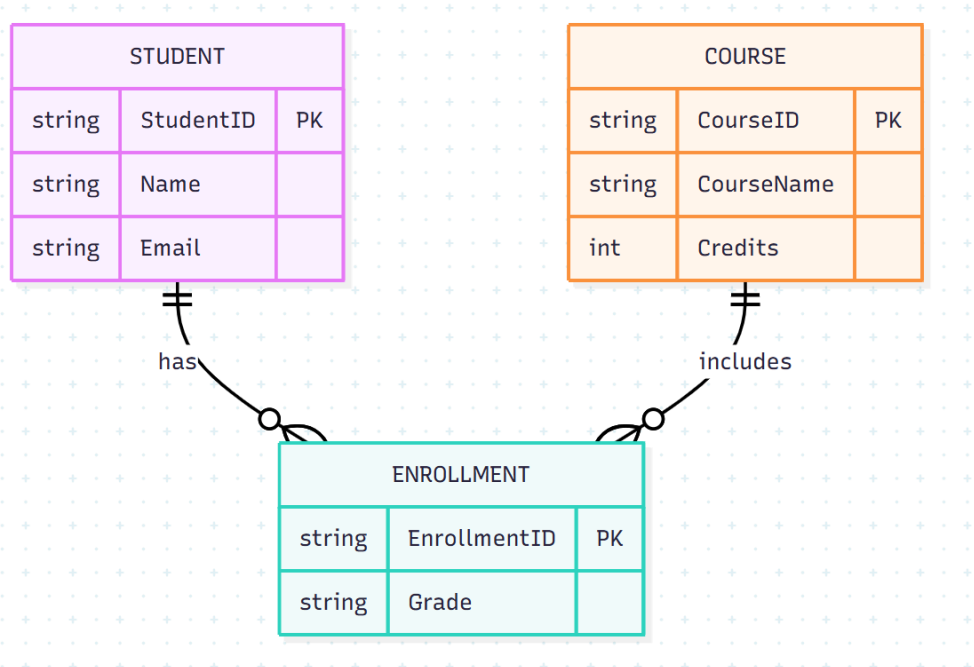
- Projection: Extracts particular columns from a table.
- Join: Merges data from multiple tables based on common attributes, enabling complex cross-table queries.

4. Advantages and Limitations

- **Advantages:**
  - ACID Properties: Relational systems maintain strong transactional consistency and reliability.
  - Normalization: Helps reduce redundancy and ensures a well-structured database design.
- **Limitations:**
  - Rigid Schema: Once the schema is established, structural changes can be challenging and may require downtime.
  - Limited Horizontal Scalability: Relational databases are not naturally optimized for large-scale distributed environments.

5. Suitable Applications

- Banking and financial applications that require strict consistency.
- Enterprise systems such as ERP or inventory management.
- Academic and administrative systems used by universities.



2.3. Semi-structured Data Model (Web/Data Exchange Era)

Representative: MongoDB (NoSQL)

2.3.1. Semi-Structured Data Models / Document-Oriented

1. Storage Structure

In semi-structured databases, data is managed as documents instead of using rigid tables like in relational systems.

These documents are commonly stored in JSON or BSON formats, which makes them suitable for representing complex data.

The structure is hierarchical, so a single document can include arrays, nested elements, or even multiple embedded sub-documents.

Collections allow polymorphism, meaning documents do not need to follow the exact same structure. This provides schema-on-read flexibility and reduces the need to define a strict schema beforehand.

2. Constraints

Even though the model is flexible, most modern document-oriented systems still support mechanisms to ensure data quality:

Schema validation can be configured to require certain fields, restrict data types, or enforce basic rules depending on the application requirements.

Write operations are atomic at the document level. This ensures that modifications to a deeply nested structure are applied consistently without leaving the document in a partial state.

3. Data Manipulation Operators

Standard CRUD operations—such as insertOne, find, updateOne, and deleteOne—form the basis of data manipulation.

The Aggregation Pipeline provides a more advanced mechanism for processing data through multiple stages. Operators like $match, $group, or $project support analytical workloads, while $lookup allows limited join-like behavior between collections, although performance is generally lower compared to traditional SQL joins.

4. Advantages and Limitations

- **Advantages:**
    - The flexible schema allows frequent updates or structural changes without service interruption, which aligns well with Agile development practices.

    - Because documents resemble common data structures used in programming languages, developers often work more efficiently without relying on complex ORM layers.
- **Limitations:**
    - Storing embedded or repeated data can lead to redundancy, increasing storage usage and making updates more expensive if the same data appears in many documents.
    - While the system supports multi-document transactions, they typically require more overhead and do not perform as efficiently as transactions in relational databases.

5. Suitable Applications

- Content management platforms where data formats vary across entries.
- E-commerce product catalogs that include diverse and frequently changing product descriptions.
- Logging systems and real-time analytics applications that require rapid ingestion and flexible structures.

Example of semi-structured student data

```
JSON

{
  "student_id": "101",
  "name": "Siu",
  "contact": {
        "email": "siu@uni.edu",
        "phone": ["090123", "090456"]
  },
  "major": "Computer Science"
 }
```

2.4. NoSQL Data Models (Scalability Era)

Representative: Redis, Amazon DynamoDB

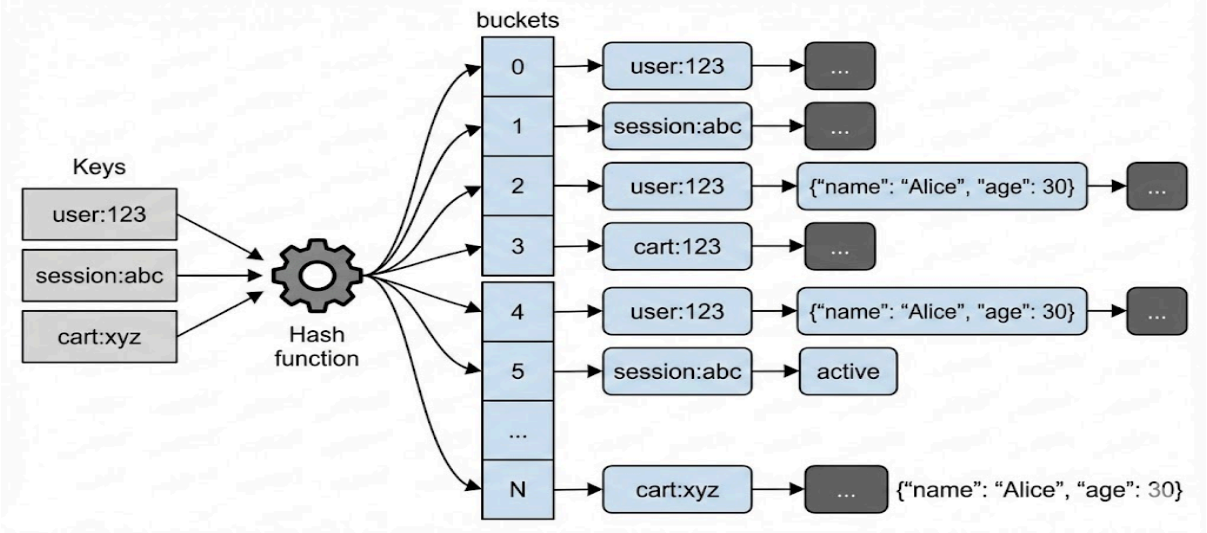
2.4.1. Key-Value Data Model

1. Storage Structure

The key-value data model is considered one of the most basic approaches within the NoSQL category.

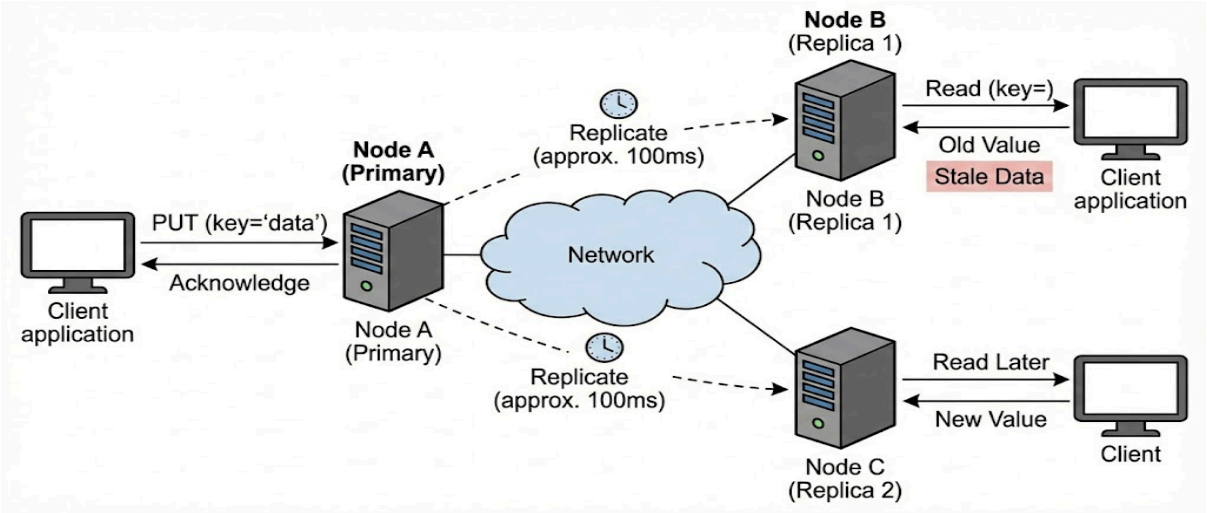
- Data is kept as pairs where each key is linked to a single value.

- The value itself is treated as a raw object, meaning the system does not analyze or index the content inside it. Its job is simply to store and return that value when needed.
- Most key-value stores make use of hash tables underneath, which helps them locate data quickly by converting keys into direct memory references.



2. Constraints

- Keys must be unique inside the same namespace, and this is essentially the main rule of the model.
- Many distributed systems in this category follow an eventual consistency approach. This means updates may take a short moment before appearing consistently across all nodes, as these systems tend to prioritize availability.



3. Data Manipulation Operators

The set of operations in this model is very simple:

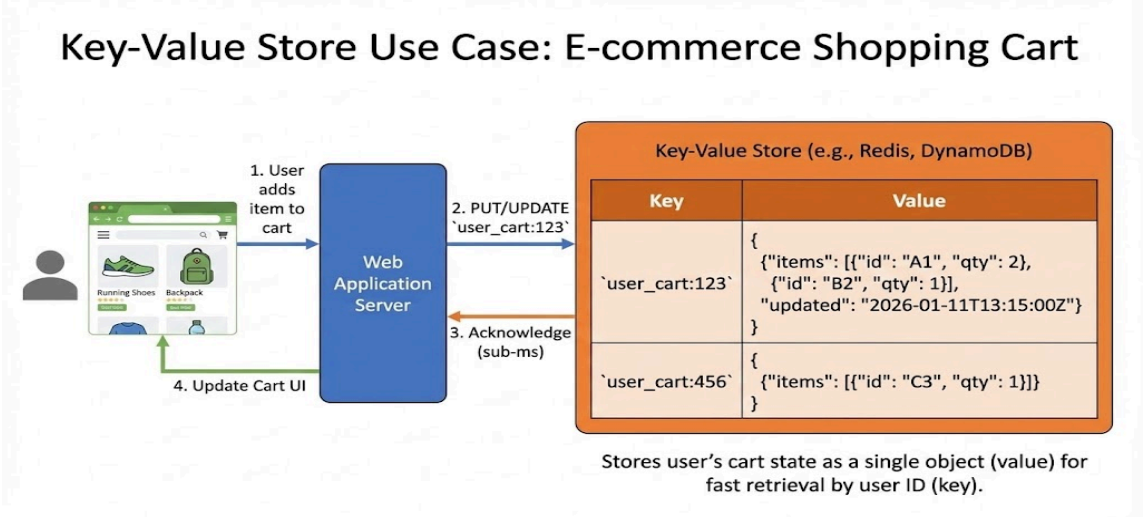
- SET/PUT is used to save a value under a specific key.
- GET allows the system to retrieve the value associated with a given key.
- DELETE removes the key-value entry.
- Because there is no query language like SQL, the system cannot filter or search by value unless it checks every key-value pair.

4. Advantages and Limitations

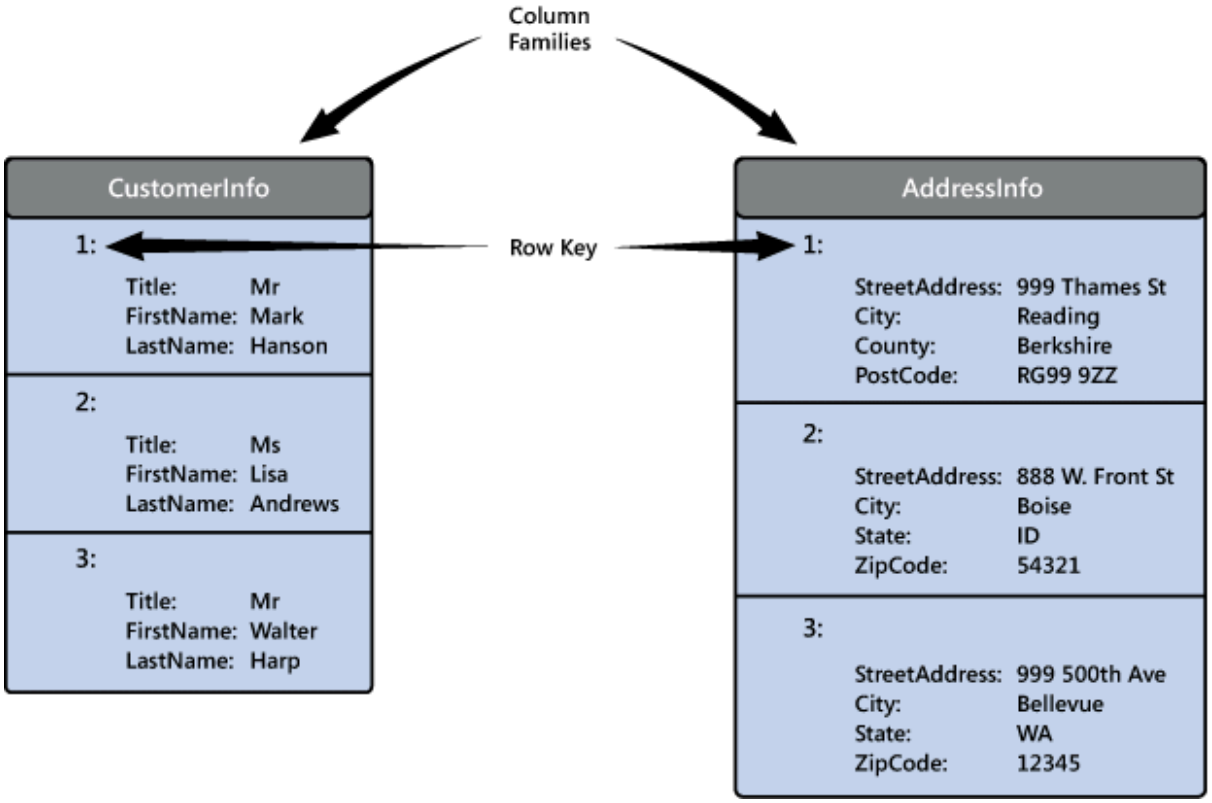
- **Advantages:**
  - Provides extremely fast performance for reading and writing data.
  - Easy to scale horizontally since data can be distributed across multiple servers based on hashing.
- **Limitations:**
  - Does not support advanced queries or joins, limiting its use in analytical or relational scenarios.
  - Handling relationships between data elements must be done in the application layer, not by the database.

5. Suitable Applications

- User Session Management To efficiently store login states and user preferences, ensuring sub-millisecond retrieval on every page load.
- High-Speed Caching To temporarily store the results of expensive database queries in memory, significantly reducing latency and system load.
- E-commerce Shopping Carts To manage dynamic lists of selected items with high write speeds, ensuring data persists reliably across user sessions.



2.4.2. Column-Family (Wide-Column) Data Model



1. General Concept

The Column-Family (Wide-Column) Data Model is a NoSQL data model designed to store large volumes of data in a flexible, scalable, and distributed manner.
Data is organized into **column families**, where each row can contain a large number of columns, and each row does not need to share the same schema. This model is optimized for fast read and write operations over massive datasets.

2. Historical Development

The column-family data model originated from Google's Bigtable to handle large-scale web data.
It was later adopted and extended by distributed NoSQL databases such as Apache Cassandra and HBase to support high availability, fault tolerance, and horizontal scalability across multiple servers.

3. Storage Structure

Column-family databases organize data as follows:

- Keyspace: The top-level namespace, similar to a database.
- Column Families: Logical groupings of related columns.
- Rows: Identified by a unique row key.
- Columns: Stored as key–value pairs and can vary between rows.

This structure allows efficient access to specific columns without scanning entire rows.

4. Constraints

Column-family databases enforce constraints mainly at the application or configuration level, including:

- Unique row keys
- Data type consistency within columns
- Optional schema definitions per column family

These constraints provide flexibility while maintaining basic data integrity.

5. Data Manipulation

Data manipulation is performed using database-specific query languages or APIs.
Operations typically include inserting rows, updating columns, deleting data, and querying by row key or column range.
The model is optimized for high-throughput writes and reads rather than complex joins.

6. Advantages and Limitations

Advantages:

- High scalability and fault tolerance
- Flexible schema suitable for evolving data
- Efficient for large-scale distributed systems

Limitations:

- Limited support for complex queries and joins
- Data modeling can be more complex than relational databases

- Strong consistency may require trade-offs in distributed environments

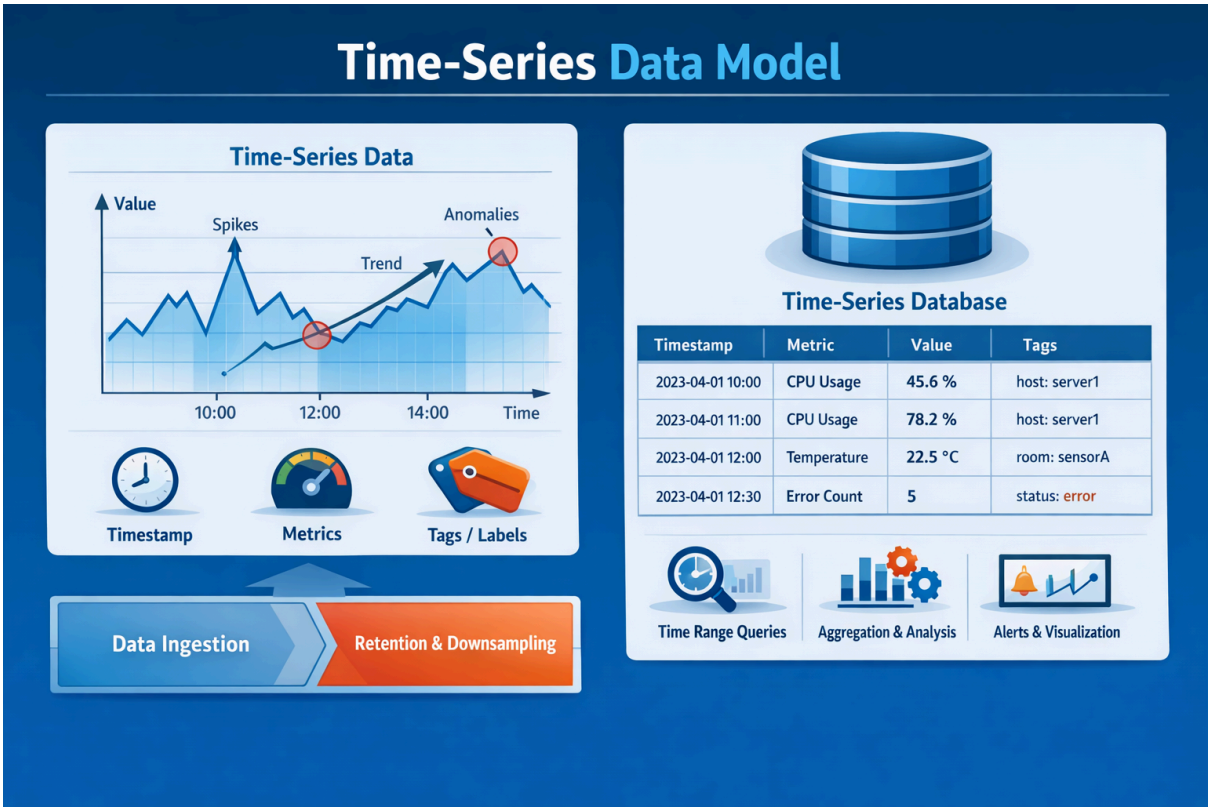
7. Suitable Applications

It is commonly used in:

- **Streaming platforms**
  Storing user activity, watch history, and content metadata.
- **Social media systems**
  Managing posts, likes, followers, and activity logs.
- **Online gaming platforms**
  Tracking player profiles, game events, and scores.
- **IoT and sensor networks**
  Storing time-series data from millions of devices.
- **Recommendation systems**
  Storing user behavior and content popularity at large scale.

These applications include large-scale data analytics, log aggregation, and content management systems.They benefit from high write throughput, horizontal scalability, and efficient storage of sparse, wide datasets.

2.4.3. Time-Series Data Model



1. General Concept

The Time-Series Data Model is designed to store and manage data points indexed by time.
Each record represents a value measured or observed at a specific timestamp, making this model ideal for tracking changes over time, such as sensor data, logs, financial metrics, and monitoring data.

2. Historical Development

Time-series data models evolved from monitoring and logging systems used in engineering and finance.
With the growth of IoT, cloud computing, and real-time analytics, specialized time-series databases such as InfluxDB, Prometheus, and TimescaleDB were developed to efficiently handle high-frequency temporal data.

3. Storage Structure

Time-series databases store data in the following structure:

- Timestamp: The primary index for each data point
- Measurements / Metrics: Values recorded over time
- Tags / Labels: Metadata used for filtering and grouping data
- Fields: Actual numeric or textual values

Data is often stored in time-partitioned chunks to optimize performance.

4. Constraints

Time-series data models enforce constraints such as:

- Mandatory timestamp for each record
- Consistent data types for metrics
- Time-based retention policies

These constraints ensure efficient storage and accurate temporal analysis.

5. Data Manipulation

Data manipulation focuses on time-based operations rather than traditional CRUD.
Common operations include inserting new data points, querying time ranges,

aggregating data over intervals, and downsampling historical data for long-term storage.

6. Advantages and Limitations

Advantages:

- Optimized for high write throughput
- Efficient time-based queries and aggregations
- Ideal for real-time monitoring and analytics

Limitations:

- Not suitable for complex relational queries
- Limited support for transactional operations
- Long-term storage may require data compression or downsampling

7. Suitable Applications

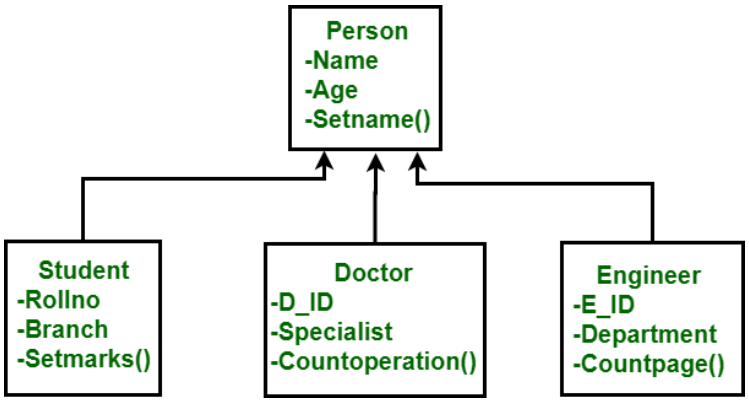
It is widely used in:

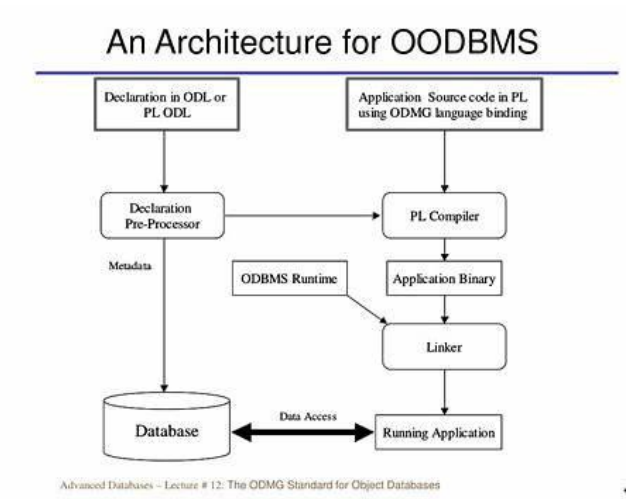
- **System and application monitoring**
  Tracking CPU usage, memory, network traffic, and service health.
- **IoT and sensor networks**
  Storing temperature, humidity, energy usage, and machine readings.
- **Financial markets**
  Recording stock prices, trading volume, and market indicators.
- **Industrial and manufacturing systems**
  Monitoring equipment performance and detecting failures.
- **Website and user analytics**
  Tracking page views, clicks, and traffic over time.

These applications include IoT monitoring, financial market analysis, and system performance tracking.They benefit from optimized storage and fast querying of time-ordered data with high ingestion rates.

2.5. Advanced / Specialized Data Models (Modern Applications)

2.5.1. Object-Oriented Database (OODBMS)





1. General Concept

The Object-Oriented Database Model is a data model that stores data as objects, similar to objects in object-oriented programming languages. Each object encapsulates attributes (data), methods (behavior), and has a unique object identity (OID). The model supports core OOP concepts such as encapsulation, inheritance, and polymorphism.

2. Historical Development

The Object-Oriented Database Model emerged in the late 1980s and early 1990s, driven by the widespread adoption of object-oriented programming languages such as C++ and Java.Its primary goal was to address the object–relational impedance mismatch, where relational tables did not naturally map to application objects.

3. Storage Structure

- Data stored as persistent objects
- Objects belong to classes
- Objects may reference other objects directly
- Each object has a unique OID
- No table–row structure as in relational databases

4. Constraints

- Object Identity Constraint: each object has a unique OID
- Type Constraint: objects must conform to their class definitions

- Encapsulation Constraint: access to object data is controlled via methods
- Inheritance Constraint: subclasses must satisfy superclass rules

Constraints are enforced through class definitions and the database engine.

5. Data Manipulation Operators

- Object Query Language (OQL)
- Method invocation on objects
- CRUD operations via object APIs
- Navigation through object references

Unlike SQL, manipulation is often embedded directly in application code.

6. Advantages and Limitations

Advantages

- Natural integration with object-oriented programming
- Eliminates object–relational mapping overhead
- Supports complex data structures
- Efficient traversal via object references

Limitations

- Lack of a standardized query language
- Smaller ecosystem and limited tooling
- Vendor lock-in risk
- Limited interoperability with other systems

7. Suitable Applications

Object-Oriented Databases are ideal for applications that rely on **complex, nested data structures** and **high-performance object processing**.

They are commonly used in:

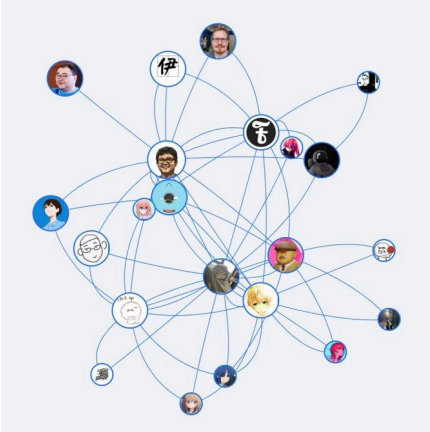
- **Computer-Aided Design (CAD/CAM)** (engineering drawings, 3D modeling, and component hierarchies)
- **Multimedia Databases** (managing video, audio, GIS data, and image processing methods)
- **Scientific and Medical Research** (genome data, molecular simulations, and complex biological records)

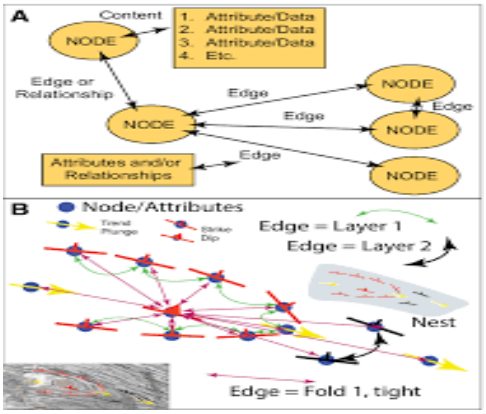
- **Real-time Systems** (telecommunications, high-frequency trading, and industrial control systems)
- **Software Engineering (CASE Tools)** (managing source code, version control, and software dependencies)

These applications benefit from **seamless integration with programming code** and efficient handling of **complex data hierarchies**.

These applications include **scientific and simulation systems** (e.g., physics simulations, bioinformatics models).They benefit from encapsulating data and behavior together, enabling reusable and extensible object models.

2.5.2. Graph Data Model





1. General Concept

A Graph Database is a data model designed to represent and manage data based on relationships. Data is stored as nodes (entities), edges (relationships), and properties (attributes). Unlike relational databases, graph databases treat relationships as first-class citizens, enabling efficient traversal and complex relationship queries.

2. Historical Development

Graph databases evolved from graph theory and network analysis to address limitations of relational databases in handling highly connected data. Modern graph database systems, such as Neo4j, popularized this model by providing optimized graph storage and query languages for real-time relationship analysis.

3. Storage Structure:

Graph databases store data using the following components:

- Nodes: Represent entities such as users, products, or locations.
- Edges: Represent relationships between nodes, often directional and typed.
- Properties: Key–value pairs stored on nodes and edges.
  This structure allows fast traversal across multiple relationships without costly joins.

4. Constraints

Graph databases support constraints such as unique node identifiers and property existence constraints. These rules ensure data validity while maintaining a flexible schema design suitable for evolving relationship structures.

5. Data Manipulation

Data is queried using graph query languages that focus on pattern matching and traversal. Queries efficiently explore paths, connections, and network patterns, enabling deep relationship analysis that is difficult to express using traditional SQL joins.

6. Advantages and Limitations

+Advantages: Graph databases excel at handling highly connected data and complex relationship queries with predictable performance. They provide flexible schemas and are well-suited for recommendation systems and network analysis.

+Limitations: They are less efficient for simple tabular data and large-scale aggregation tasks. Graph databases may also lack the mature transactional ecosystems found in traditional relational systems.

7. Suitable Applications

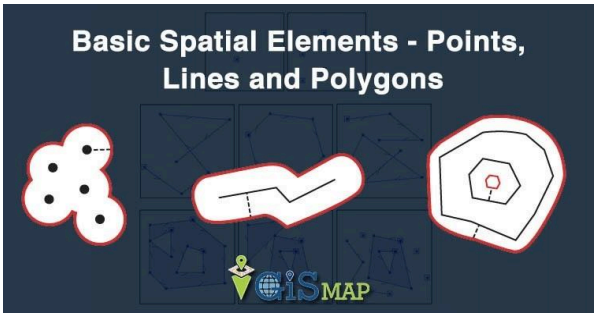
Graph databases are ideal for applications that rely on **relationships and network structures**.

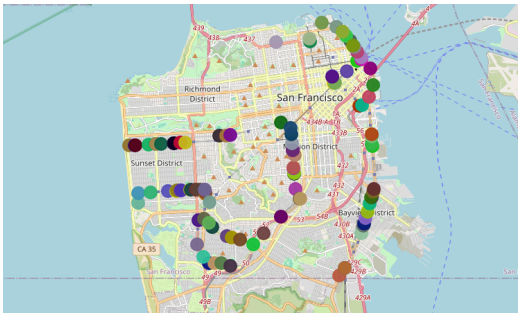
They are commonly used in:

- **Social networks** (friends, followers, interactions)
- **Recommendation systems** (products, movies, or friends)
- **Fraud detection and security analysis**
- **Knowledge graphs and semantic networks**
- **Network and IT infrastructure management**

These applications benefit from **fast traversal and relationship-centric queries**.

2.5.3. Spatial / Geospatial Data Model





1. General Concept:

A Spatial (Geospatial) Database is a database system designed to store, manage, and query spatial data that represents real-world geographic objects. These objects are defined using geometric data types such as points, lines, and polygons, allowing the system to model locations, distances, and spatial relationships.

2. Historical Development:

Spatial databases emerged from Geographic Information Systems (GIS) to address the limitations of traditional databases in handling geographic data. With the integration of spatial extensions into relational databases, systems such as PostGIS enabled efficient storage and querying of spatial data using standard SQL with spatial functions.

3. Storage Structure

Spatial databases store data in tables that include spatial attributes alongside traditional data types.

- Spatial Objects: Points (locations), Lines (roads, routes), and Polygons (regions, areas).
- Spatial Reference Systems: Coordinates are stored using defined coordinate systems to represent real-world locations accurately.
- Spatial Indexing: Index structures such as R-trees are used to accelerate spatial queries.

4. Constraints:

To ensure spatial correctness, spatial databases enforce constraints including valid geometry types, coordinate system consistency, and topological rules. These constraints prevent invalid shapes and ensure accurate spatial relationships between objects.

5. Data Manipulation

Spatial data is manipulated using spatial query languages and extensions of SQL. Operations include spatial selection, distance calculation, intersection detection, containment checks, and proximity searches, enabling advanced geographic analysis.

6. Advantages and Limitations

+Advantages: Spatial databases efficiently handle complex geographic data and spatial queries that are difficult or inefficient in traditional relational databases. They support advanced spatial analysis and real-time location-based services.

+Limitations: Spatial data processing can be computationally expensive, and managing large-scale spatial datasets requires careful indexing and system optimization.

7. Suitable Applications

Spatial databases are best suited for systems that manage and analyze **location-based data**.

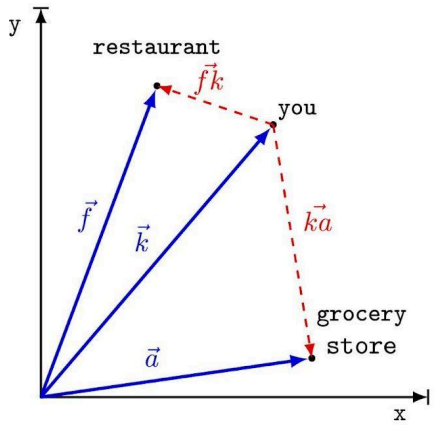
They are widely used in:

- **Navigation and mapping systems** (road maps, routing, traffic analysis)
- **Urban planning and land management**
- **Geographic Information Systems (GIS)**
- **Logistics and delivery tracking**
- **Environmental and climate monitoring**

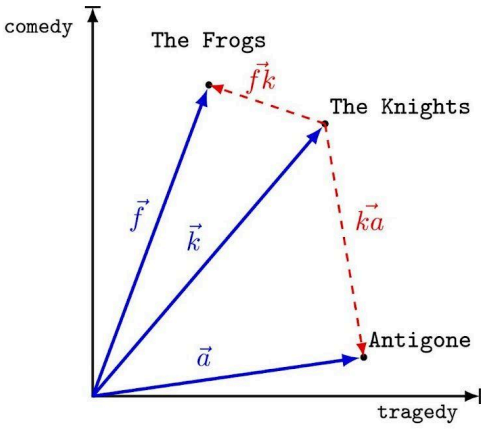
These applications include GIS systems, location-based services, and urban planning platforms.They benefit from specialized spatial indexing and efficient querying of geographic relationships and distances.

2.5.4. Vector Data Model

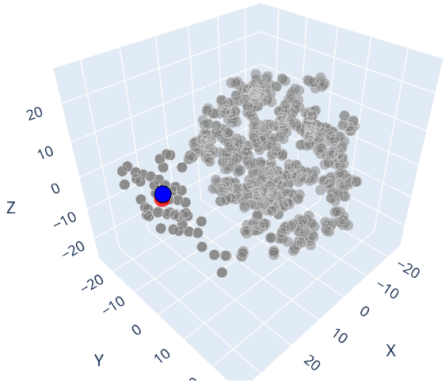
1. General Concept

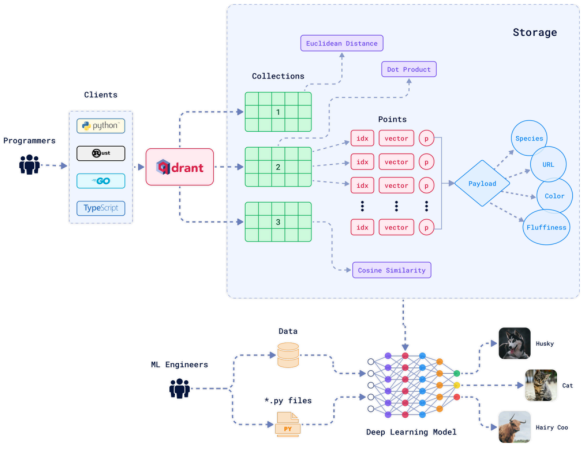


(a) Closeness in geospatial locations



(b) Closeness in genres





The Vector Data Model represents data as high-dimensional numerical vectors (embeddings). Each vector captures semantic meaning, allowing systems to compare data based on similarity rather than exact matches. This model is widely used in machine learning and AI-driven applications such as semantic search and recommendation systems.

2. Historical Development

Vector data models gained importance with advances in machine learning and natural language processing. As embedding techniques became common, specialized vector databases such as **Pinecone** and **FAISS** emerged to support efficient similarity search over large-scale vector data.

3. Storage Structure

Vector databases store data as collections of vectors along with optional metadata.

- Vectors: High-dimensional numerical arrays generated by embedding models.
- Metadata: Descriptive attributes associated with vectors.
- Indexes: Approximate nearest neighbor (ANN) indexes used to accelerate similarity search.
  This structure enables fast comparison of vectors based on distance metrics.

4. Constraints

Vector data models apply constraints related to vector dimensionality and data type consistency. All vectors within a collection must share the same dimension to ensure valid similarity calculations.

5. Data Manipulation

Data manipulation focuses on vector similarity operations rather than traditional CRUD queries. Common operations include inserting vectors, updating metadata, and performing similarity searches using distance measures such as cosine similarity or Euclidean distance.

6. Advantages and Limitations

+Advantages: Vector data models enable semantic understanding and similarity-based retrieval, making them ideal for AI applications. They scale efficiently and support real-time search over massive unstructured datasets.

+Limitations: They are not suitable for transactional data or strict consistency requirements. Vector search results are approximate and may trade accuracy for performance.

7. Suitable Applications

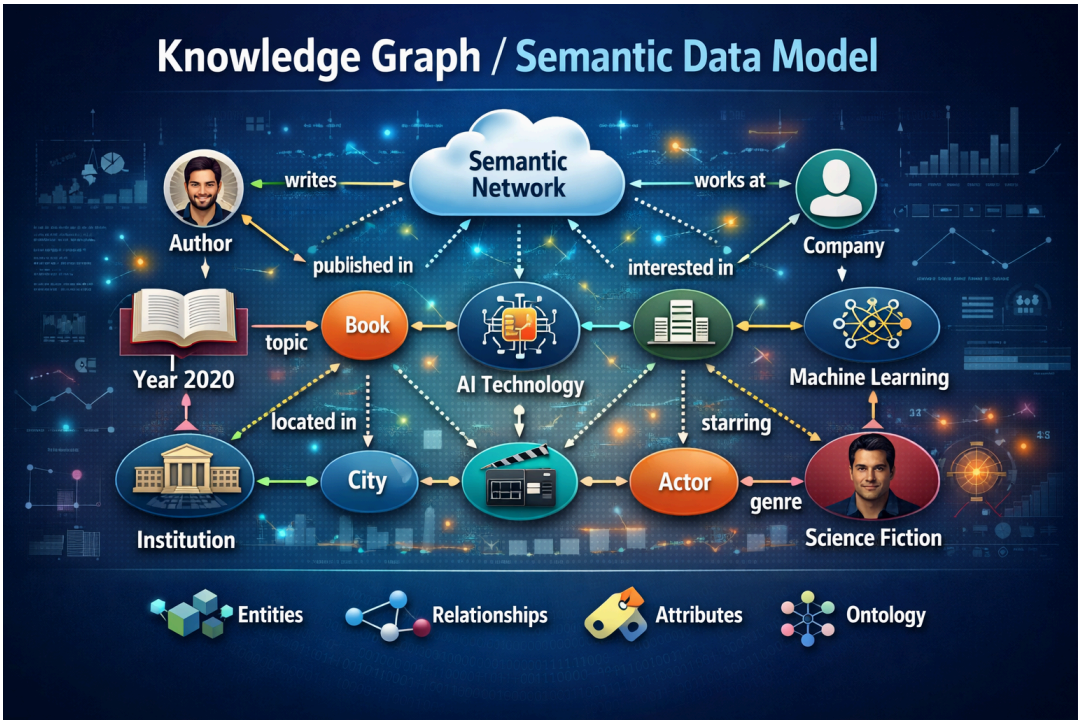
Vector data models are best suited for **AI-driven and semantic applications**.

They are widely used in:

- **Semantic search engines**
- **Chatbots and AI assistants**
- **Document and image similarity search**
- **Recommendation systems**
- **Fraud and anomaly detection**

These systems require **similarity-based retrieval rather than exact matching**, which vector databases provide.

2.5.5. Knowledge Graph/ Semantic Data Model



1. General Concept

A Knowledge Graph, also called a Semantic Data Model, represents information as a network of interconnected entities and relationships. Instead of storing data in tables or documents, it focuses on meaning, context, and connections between data elements.

Each entity (for example, a person, place, or object) is linked to other entities through well-defined relationships. This allows the system to understand not only facts, but also how those facts relate to each other, making the data more meaningful and easier to interpret.

Knowledge graphs are especially useful for answering complex questions, discovering hidden relationships, and supporting intelligent applications.

2. Historical Development

The Knowledge Graph model originated from Semantic Web research, which aimed to make web data understandable by machines.

Early efforts such as RDF (Resource Description Framework) and OWL (Web Ontology Language) provided standards for representing and reasoning about data.

Over time, this approach was adopted by large-scale systems, including:

- Google Knowledge Graph
- Wikidata
- DBpedia
- Microsoft Satori

These systems use knowledge graphs to power search engines, digital assistants, and recommendation systems.

3. Storage Structure

A knowledge graph stores data using graph-based and semantic structures:

Nodes (Entities)

Represent real-world objects such as people, organizations, places, or concepts.

Edges (Relationships)

Define how entities are connected, such as *“is part of”*, *“works for”*, or *“located in”*.

Properties

Store attributes of entities and relationships, such as names, dates, or numerical values.

Triples

Data is usually stored as:
Subject – Predicate – Object
Example:
(Albert Einstein – bornIn – Germany)

These triples form a graph that can be traversed and analyzed.

4. Constraints

To maintain semantic correctness and consistency, knowledge graphs enforce several constraints:

- Unique entity identifiers to ensure that each entity is clearly defined
- Defined relationship types to avoid ambiguous or incorrect links
- Ontology rules that describe valid classes, properties, and relationships

Ontologies allow the system to validate data and ensure that relationships make logical sense, such as a person having a birthplace but not a city having a birthday.

5. Data Manipulation

Knowledge graphs support a wide range of operations:

- Adding or removing entities
- Creating, deleting, or modifying relationships
- Updating properties of entities
- Querying data using SPARQL or graph query languages
- Applying reasoning to infer new facts

For example, if the system knows that *“Paris is in France”* and *“France is in Europe”*, it can infer that *“Paris is in Europe”* even if that fact was not explicitly stored.

6. Advantages and Limitations

Advantages

- Represents complex and meaningful relationships
- Supports semantic search and intelligent queries
- Enables reasoning and inference
- Excellent for large, interconnected knowledge systems
- Improves data integration across different sources

Limitations

- Data modeling is complex and requires expertise
- Higher storage and processing costs
- Slower performance for simple lookup queries
- Requires well-designed ontologies

7. Suitable Applications

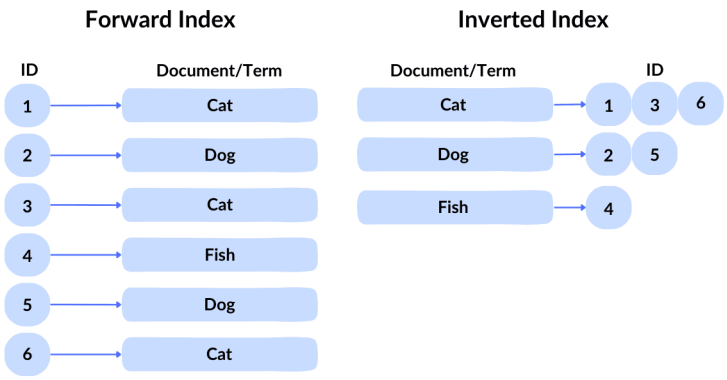
Knowledge graphs are best suited for systems that require a deep understanding of relationships and meaning.

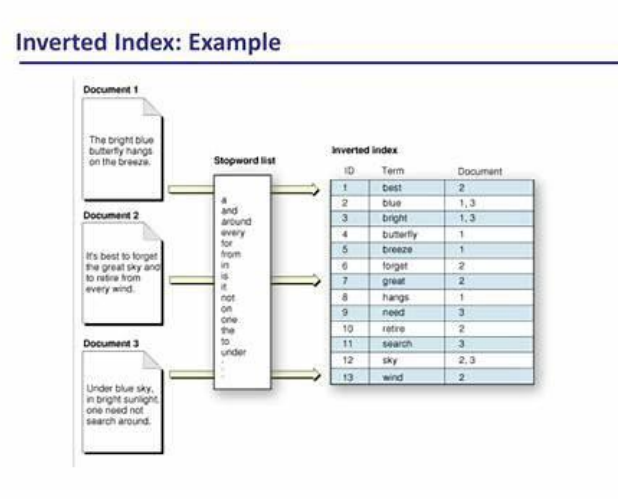
They are commonly used in:

- Search engines
  To understand entities and their relationships, improve search accuracy.
- Recommendation systems
  To suggest products, movies, or articles based on user interests and connections.
- Digital assistants
  To answer complex questions and provide context-aware responses.
- Scientific and medical databases
  To link diseases, treatments, genes, and research papers

Knowledge graphs are ideal for applications that need rich semantic relationships, reasoning, and intelligent data interpretation.

2.5.6. Inverted Index Data Model





1. General Concept

The Inverted Index Data Model is a data model designed to support efficient full-text search.Instead of mapping documents to their contents, the model maps terms (keywords) to the list of documents in which they appear.
This structure enables fast keyword-based retrieval and relevance ranking, making it fundamental to modern search systems.

2. Historical Development

The inverted index originated in the 1950s–1960s within the field of information retrieval.
It became widely adopted with the growth of digital libraries and later formed the core data structure of web search engines.

With the rise of the internet and large-scale text data, inverted indexes became essential for scalable search systems.

3. Storage Structure

An inverted index consists of:

- Dictionary (Vocabulary): a list of unique terms
- Posting Lists: for each term, a list of document identifiers
- Optional Metadata: term frequency, document frequency, and word positions

This structure is optimized for read-heavy search workloads.

4. Constraints

The inverted index model enforces several constraints:

- Each term appears only once in the dictionary
- Posting lists must reference valid documents
- Index consistency must be maintained when documents are updated or deleted
- Posting lists are often stored in sorted order to improve search performance

These constraints ensure correct and efficient retrieval.

5. Data Manipulation

Data manipulation focuses on search-oriented operations:

- Index construction from document collections
- Inserting documents and updating posting lists
- Removing documents from the index
- Term-based lookup and ranking
- Query processing using relevance scoring algorithms

Unlike relational models, operations are optimized for search, not transactions.

6. Advantages and Limitations

Advantages

- Extremely fast full-text search
- Scales well to very large document collections
- Supports relevance ranking and keyword-based retrieval

Limitations

- Not suitable for transactional or relational data
- Index maintenance overhead
- Limited support for complex structured queries

7. Suitable Applications

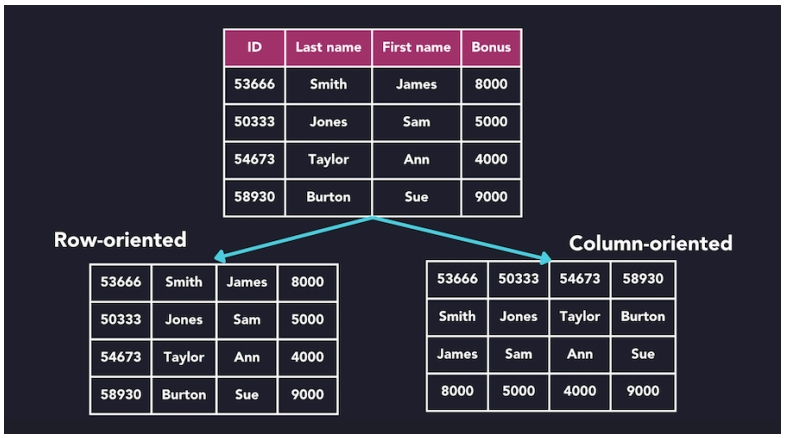
The inverted index model is ideal for large-scale text search and information retrieval.

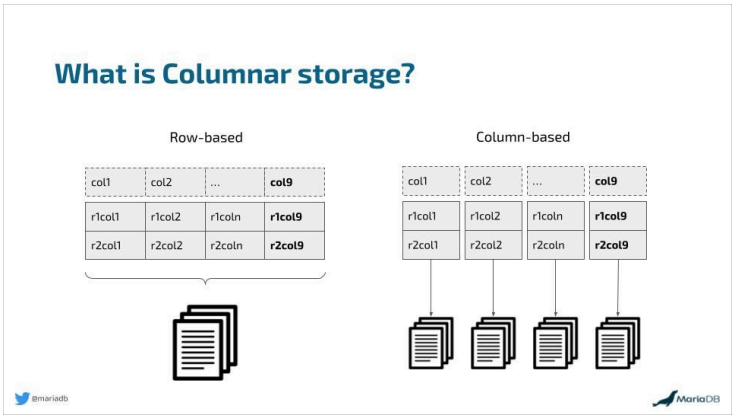
It is used in:

- Web search engines
- Enterprise document search systems
- E-commerce product search
- Digital libraries and archives
- Log analysis and log search platforms

These systems depend on fast keyword lookup and relevance ranking.

2.5.7. Columnar Analytical Data Model





1. General Concept

The Columnar Analytical Data Model is a data model optimized for analytical workloads (OLAP).
Instead of storing data row by row, this model stores data column by column, allowing systems to efficiently scan, aggregate, and analyze large datasets.

Row database	Column database
Easier to add and delete data	Data modification could be more complicated
Ideal for OLTP	Ideal for OLAP
Slower data aggregation	Fast data aggregation
Poor compression	High-speed compression
Requires more space for data storage	Requires less space for data storage

It is especially suitable for queries involving a subset of columns over a large number of records.

2. Historical Development:

Column-oriented storage concepts emerged from research in data warehousing and analytical databases in the late 1990s and early 2000s.
As data volumes grew rapidly, traditional row-based relational databases struggled with analytical queries.
This led to the adoption of columnar systems in modern analytics platforms and cloud data warehouses.

3. Storage Structure

Columnar analytical databases organize data as:

- Columns stored separately rather than full rows
- Each column contains values of the same attribute
- Data is often compressed using column-specific compression techniques
- Columns are stored in contiguous blocks to optimize sequential reads

This structure significantly reduces I/O for analytical queries.

4. Constraints

The columnar analytical data model enforces constraints such as:

- Consistent data types within each column
- Fixed schema defined before data loading

- Read-optimized storage with limited support for frequent updates
- Batch-oriented data ingestion rather than row-level transactions

These constraints prioritize analytical performance over transactional flexibility.

5. Data Manipulation

Data manipulation is designed for analytics:

- Bulk data loading
- Column-based scanning
- Aggregation operations (SUM, AVG, COUNT)
- Filtering and grouping across large datasets

Row-level updates and deletes are typically limited or expensive compared to OLTP systems.

6. Advantages and Limitations

Advantages

- Extremely fast analytical queries
- Efficient compression and reduced storage cost
- Ideal for large-scale data analysis and reporting

Limitations

- Poor performance for frequent row-level updates
- Not suitable for transactional systems
- Higher latency for single-record access

7. Suitable Applications

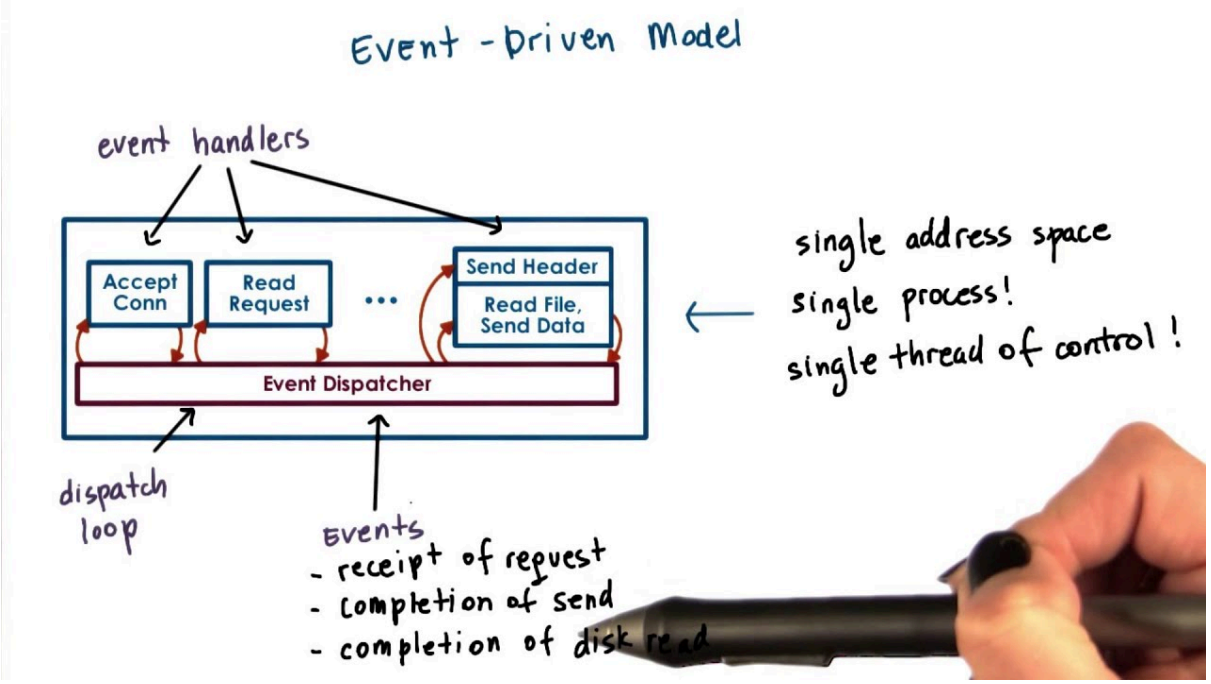
Columnar analytical databases are best suited for data analysis and reporting.

They are commonly used in:

- Business intelligence (BI) systems
- Data warehouses and data lakes
- Financial and sales reporting
- Customer behavior analysis
- Large-scale analytics and dashboards

These applications benefit from fast aggregations over massive datasets.

2.5.8. Event-Driven Data Model



1. General Concept

The Event-Driven Data Model is a data architecture in which all changes in a system are represented as events. An event is a record that something has happened, such as a user clicking a button, a payment being completed, or a sensor sending a reading.

Instead of storing only the current state of data, this model stores a chronological sequence of events. Each event contains information about:

- What happened
- When it happened
- Who or what caused it
- Any relevant data

By replaying events, the system can reconstruct past states and understand how the system evolved over time.

2. Historical Development

The Event-Driven Data Model became widely used with the rise of:

- Distributed systems
- Microservice architectures

- Real-time web and mobile applications

Traditional databases were designed for direct data updates, but modern systems need to react instantly to user actions and system changes. Event-driven architectures solved this by allowing systems to communicate through events rather than direct calls.

Technologies that popularized this model include:

- Apache Kafka
- RabbitMQ
- AWS EventBridge
- Apache Pulsar

These platforms provide reliable, scalable event streaming for large systems.

3. Storage Structure

In an event-driven system, data is stored as ordered event logs.

The main components are:

Event

Each event contains:

- A unique ID
- An event type (for example, OrderCreated)
- A timestamp
- A key (such as user ID or order ID)
- A value (the event data)
- Optional metadata

Event Stream

A stream is a sequence of related events, usually grouped by a key.

Topics or Channels

Events are published to topics. Applications subscribe to topics to receive and process events.

Events are stored in append-only logs, meaning new events are added but old events are never changed or removed.

4. Constraints

To guarantee reliability and traceability, the Event-Driven Data Model enforces several constraints:

- Immutability
  Once an event is written, it cannot be changed or deleted.
- Uniqueness
  Each event has a unique identifier to avoid duplication.
- Ordering
  Events in a stream are stored in the order they occurred.

These constraints make it possible to audit data, debug systems, and recover from failures by replaying events.

5. Data Manipulation

Instead of updating records directly, the system works by handling events.

Key operations include:

- Publish – Producers send new events into the system
- Subscribe – Consumers listen to events from specific topics
- Process – Applications analyze or transform events
- Replay – Past events can be reprocessed to rebuild system state

This makes the model ideal for real-time analytics, monitoring, and automation.

6. Advantages and Limitations

Advantages

- Supports real-time data processing
- Highly scalable and fault-tolerant
- Ideal for distributed and cloud-based systems
- Enables full data history and auditing
- Decouples services, improving flexibility

Limitations

- More complex system design
- Debugging can be difficult

- Requires careful event schema management
- Higher storage usage due to event logs

7. Suitable Applications

The Event-Driven Data Model is most suitable for real-time and highly distributed systems.

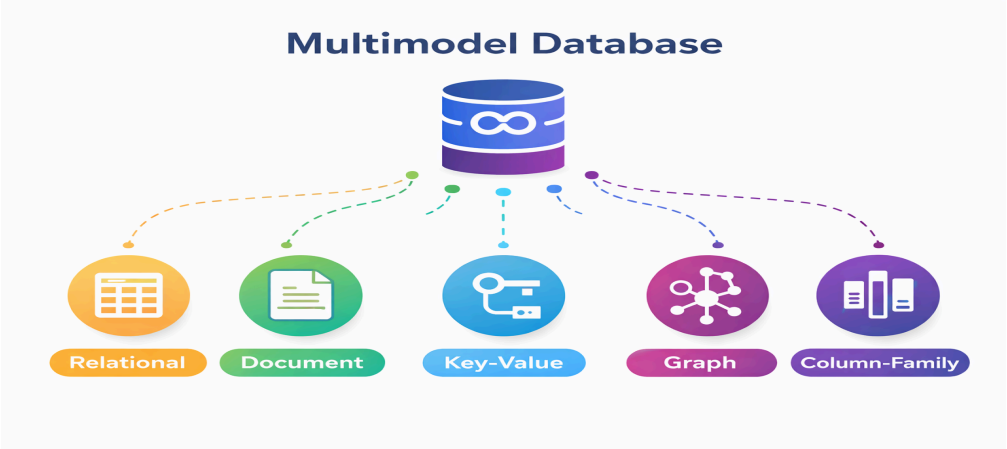
It is widely used in:

- Ride-sharing and delivery platforms
  Events track ride requests, driver status, trip progress, and payments.
- Financial systems
  Transactions, balance updates, and fraud detection rely on streams of events.
- Online shopping platforms
  Events represent orders, shipments, and customer interactions.
- Monitoring and logging systems
  Every system action is recorded as an event for analysis and debugging.

This model is best for applications that need high speed, scalability, and real-time data processing.

This allows Uber to handle millions of real-time operations efficiently.

2.5.9. Multimodel Database



1. General Concept

A Multimodel Database is a database management system designed to support multiple data models within a single, integrated platform. Instead of using different databases for different data types, a multimodel database allows applications to store, query, and manage diverse data structures in one system.

The data models typically supported include:

- Relational model (tables, rows, columns)
- Document model (JSON, XML, BSON documents)
- Key–Value model (pairs of keys and values)
- Graph model (nodes and edges)
- Column-family model (grouped columns for scalable storage)

This unified approach simplifies system architecture and improves data consistency across applications.

2. Historical Development

Multimodel databases emerged as a response to the rapid growth of Big Data, cloud computing, and modern web applications.

Traditional relational databases were not always flexible enough to handle:

- Unstructured and semi-structured data
- Highly connected data
- Large-scale distributed workloads

To solve this, developers started using multiple specialized databases (NoSQL, graph databases, document stores). However, managing many systems increased cost and complexity.

Multimodel databases were created to combine these models into one platform. Well-known examples include:

- ArangoDB
- Azure Cosmos DB
- OrientDB
- MarkLogic

3. Storage Structure

A multimodel database stores data using different physical structures, depending on the data model:

Data Model	Storage Format
Relational	Tables with rows and columns
Document	JSON or XML documents
Key–Value	Simple key-value pairs
Graph	Nodes (entities) and edges (relationships)
Column-family	Columns grouped into families

Although these formats differ, they are all managed by the same database engine, allowing cross-model queries and transactions.

4. Constraints

Constraints ensure data accuracy, consistency, and integrity. Each data model uses different types of constraints:

- Relational model
    - Primary keys
    - Foreign keys
    - NOT NULL, UNIQUE, CHECK
- Document model
    - Schema validation rules
    - Field type constraints
- Key–Value model

    - Unique keys
    - Optional value validation
- Graph model
    - Unique node IDs
    - Relationship type rules

These constraints allow multimodel databases to be both flexible and reliable.

5. Data Manipulation

Multimodel databases allow data to be manipulated using either:

- Unified query languages (e.g., SQL-like or JSON-based)
- Model-specific queries (e.g., graph traversal queries, document filters)

Typical operations include:

- Insert – Add new records, documents, or nodes
- Update – Modify existing data
- Delete – Remove unwanted data
- Query – Retrieve and analyze data

Many multimodel systems also support transactions across different data models, ensuring consistency when multiple models are involved.

6. Advantages and Limitations

Advantages

- Supports multiple data models in one database
- Reduces system complexity
- Improves data integration
- Suitable for modern, data-driven applications
- Easier maintenance and deployment

Limitations

- More complex internal architecture
- Performance may not match specialized single-model databases
- Developers must learn multiple data models
- More difficult to optimize than single-model systems

7. Suitable Applications

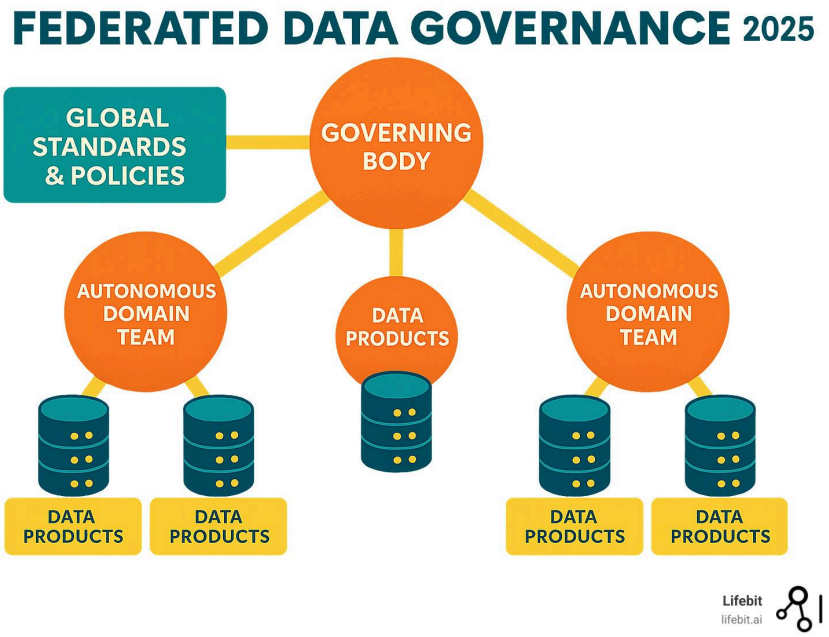
Multimodel databases are best suited for systems that must handle different types of data in one platform.

They are ideal for:

- E-commerce platforms
  Product data can be stored as documents, customer data in relational tables, and recommendations as graph relationships.
- Enterprise systems
  Business records, logs, and relationships between departments can coexist in a single database.
- IoT and smart systems
  Sensor data (key-value), device metadata (documents), and device relationships (graphs) can be stored together.
- Content management systems (CMS)
  Articles, users, comments, and links between content types can be managed efficiently.

Multimodel databases reduce the need to integrate multiple databases, making them suitable for complex, data-rich applications.

2.5.10. Federated Data Model

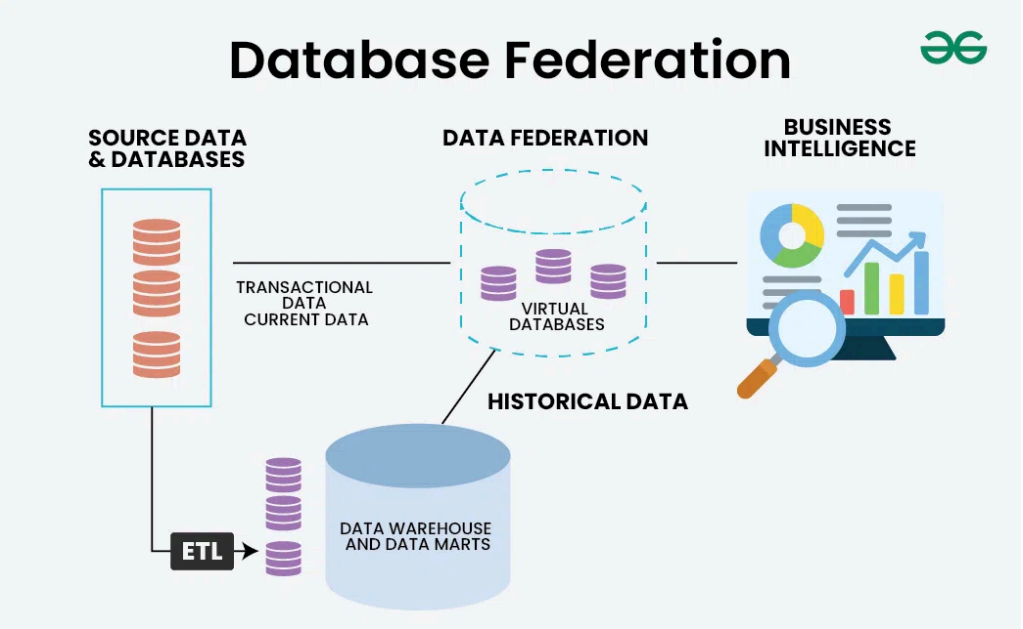


1. General Concept

The Federated Data Model is a data integration model that provides a unified logical view over multiple autonomous and heterogeneous data sources. Instead of physically consolidating data into a single database, the model enables virtual integration, allowing queries to access distributed sources transparently.

2. Historical Development

Federated database concepts emerged in the 1990s as organizations needed to integrate legacy systems without replacing them.

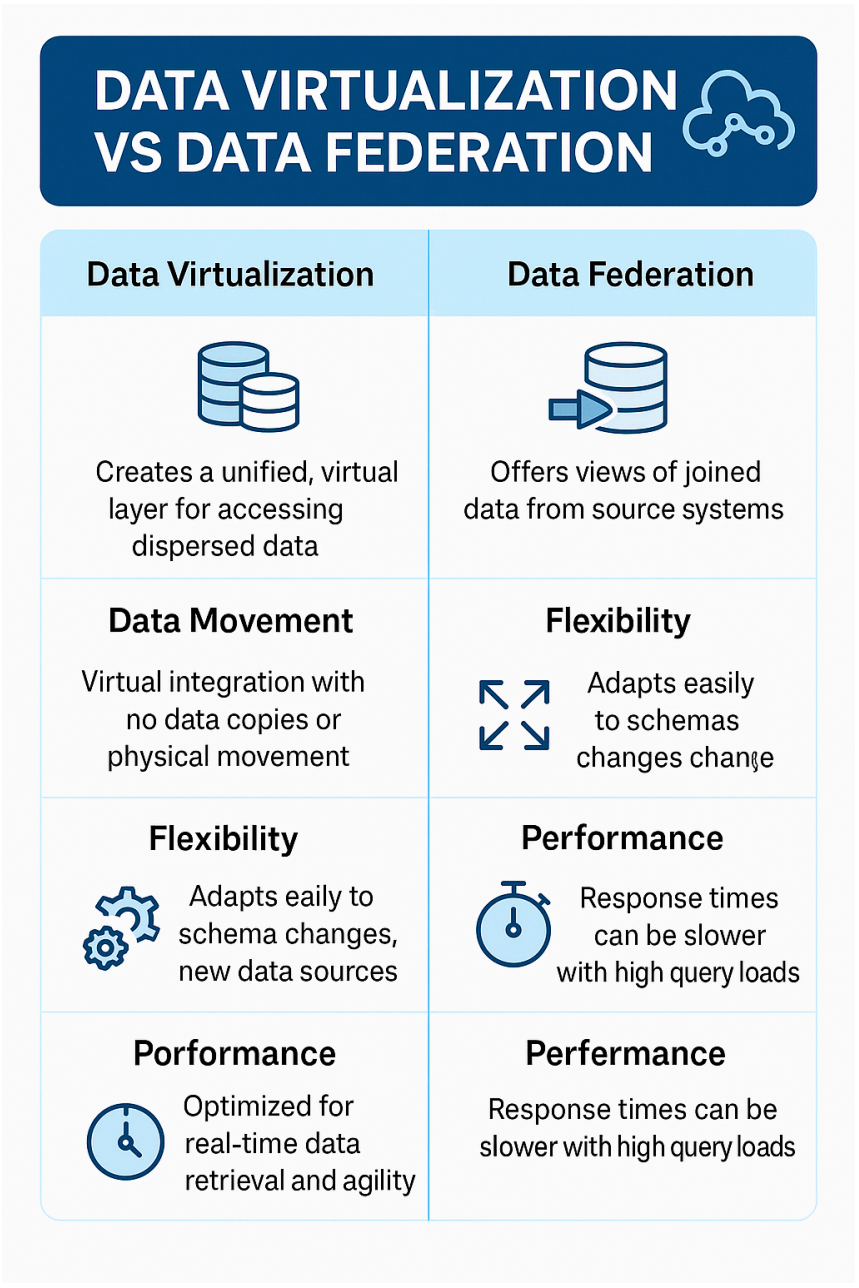


The approach gained renewed importance with enterprise integration, cloud computing, and microservices, where data is naturally distributed across systems.

3. Storage Structure

- Local Databases: independent systems with their own schemas and DBMSs
- Federation Layer (Mediator): provides a global schema and query translation
- Wrappers/Adapters: map global queries to local data sources
- Virtual Views: present integrated data without physical replication

Data remains at the source; only metadata and mappings are centralized.



4. Constraints

The federated data model enforces constraints mainly at the integration layer:

- Schema mapping consistency between global and local schemas
- Limited control over local data constraints and transactions
- Data freshness depends on source system availability
- Global integrity constraints

are often weak or partially enforced

These constraints prioritize autonomy and interoperability over strict consistency.

5. Data Manipulation

Data manipulation is primarily read-oriented:

- Global queries decomposed into subqueries
- Query translation and optimization across sources
- Result aggregation and reconciliation
- Limited or controlled write-back to source systems

Complex transactions across multiple sources are typically avoided.

6. Advantages and Limitations

Advantages

- Integrates heterogeneous systems without data migration
- Preserves autonomy of existing databases
- Reduces data duplication and synchronization cost
- Enables unified access to distributed data

Limitations

- Query performance depends on remote sources
- Limited support for global transactions
- Complex schema mapping and maintenance
- Potential latency and availability issues

7. Suitable Applications

The federated data model is best suited for data integration across multiple systems.

It is widely used in:

- Enterprise data integration
- Business intelligence across multiple databases
- Cloud and hybrid-cloud systems
- Microservice architectures

- Legacy system integration

These environments require unified access to distributed data without physically moving it.

3. Most Appropriate Data Model for Group Project

The group project focuses on developing a movie ticket booking system, which involves managing highly structured and interrelated data such as movies, cinemas, screening rooms, showtimes, seats, customers, bookings, and payments.

After evaluating various data models, the Relational Data Model (RDBMS) is identified as the most appropriate choice for this project.

Firstly, the relational model is well-suited for handling structured data with clearly defined relationships. For example, a booking must reference a valid customer, a specific movie, and an available showtime. These relationships can be effectively enforced using primary keys and foreign keys, ensuring strong data integrity.

Secondly, transactional consistency is a critical requirement in a ticket booking system. Operations such as seat reservation and payment processing must be handled atomically to prevent issues like double booking. Relational databases support ACID properties, which guarantee reliable and consistent transactions.

Additionally, SQL provides powerful query capabilities, allowing the system to easily retrieve information such as available seats, booking history, revenue reports, and showtime schedules. This makes the relational model highly suitable for both operational and reporting purposes.

Although NoSQL or semi-structured models offer greater flexibility and scalability, their weaker support for complex transactions and relational constraints makes them less ideal for this type of application. Therefore, the relational data model provides the best balance between data consistency, reliability, and ease of implementation for the movie ticket booking project.

4. Conclusion and Reflection

Through this laboratory, the group gained a comprehensive understanding of the evolution of data models, from classical hierarchical and network models

to modern NoSQL and specialized models such as graph, spatial, and vector databases.

Each data model has its own strengths and is suitable for specific application scenarios. Classical models emphasize performance and navigational access, while modern NoSQL and advanced models focus on scalability, flexibility, and specialized data processing. However, no single model is universally optimal for all use cases.

For the movie ticket booking project, the relational data model was selected due to its strong support for structured data, integrity constraints, and transactional consistency. This choice aligns well with the system's requirements for accurate booking management and reliable payment processing.

Overall, this lab not only strengthened the group's theoretical knowledge of database systems but also improved practical decision-making skills in selecting an appropriate data model for real-world applications. The insights gained from this lab will be valuable for future database design and system development tasks.