- Most applications are built by layering one data model on top of another
- Objects or data structures and APIs –> General-purpose data model (e.g., JSON or XML documents) –> Bytes in memory, on disk, or on a network –> Electrical currents, pulses of light, magnetic fields

## Data Models

- SQL, based on Edgar Codd's relational model from 1970 is a widely recognized and dominant data model. The roots of relational databases lie in business data processing, which can be categorized into two main use cases:
    - Transaction Processing:
        - Entering sales or banking transactions
        - Airline reservations
        - Stock-keeping in warehouses
    - Batch Processing:
        - Customer invoicing
        - Payroll management
        - Reporting
- Over the years, there have been many competing approaches to data storage and querying:
    - Network model (1970s-1980s)
    - Hierarchical model (1970s-1980s)
    - Object databases (brief resurgence in late 1980s and early 1990s)
    - XML databases (limited adoption since early 2000s)
- Relational databases have generalized well beyond business data processing and remain a key driving force even behind the web today.

The birth of NoSQL

- NoSQL emerged as a challenge to the relational model's dominance. The term was coined in 2009 as a catchy hashtag for a meetup on distributed, non relational databases. It stands for Not Only SQL, representing a range of technologies beyond traditional relational databases.
- Several driving forces behind the rise of NoSQL include:
    - Need for greater scalability
    - Widespread preference for free and open-source software
    - Specialized query operations
    - Frustration with the restrictiveness of relational schemas
- NoSQL is diverging into two main directions:
    - Document databases: designed for use cases where data is stored in self-contained documents, with limited relationships between documents.
    - Graph databases: tailored for use cases where any element can potentially relate to everything else, emphasizing complex relationships.

- All three models - document, relational, and graph - are widely utilized today, each excelling in its respective domain.

The Object-Relational Mismatch

- The SQL data model faces criticism due to the **impedance mismatch** between relational tables and object-oriented programming languages. Object-oriented models represent data as objects with properties and methods, while relational models use tables with columns and rows. Flattening object hierarchies or joining multiple tables may be necessary. This requires a translation layer, like object-relational mapping (ORM) tools, adding complexity in handling data.
- Ex: A LinkedIn profile:
    - How to represent this One-to-many relationship:
        - Separate Tables: Positions, education, and contact information are stored in tables with foreign key reference to the user table. This is the normalized way.
        - Structured Datatypes: SQL versions support structured datatypes, XML, and JSON storing multi-valued data within a single row.
        - JSON/XML as a text: Encode jobs, education, and contact info as JSON or XML documents stored in a text column in the database.
- For a self-contained document-like data structure like a résumé, a JSON representation is appropriate. It also provides a better data locality. Document-oriented databases such as MongoDB, RethinkDB, CouchDB, and Espresso support this model.

Ex:
```
{
 "positions": [
   {
    "job_title": "Co-chair",
    "organization": "Bill & Melinda Gates Foundation"
   },
   {
    "job_title": "Co-founder, Chairman",
    "organization": "Microsoft"
   }
 ]
}
```

- The JSON representation has better locality than the multi-table schema. **Data Locality** - storing all relevant information in one place, which facilitates efficient retrieval using a single query.

## Many-to-One and Many-to-Many Relationships

- **Normalized schema** - As a rule of thumb, duplicating values that could be stored in just one place indicates that the schema is not normalized.
- If we choose to normalize the data by replacing actual values with IDs and utilizing separate tables, there are several advantages:
    - Standardization: Values are standardized, ensuring consistent style and avoiding ambiguity (e.g., cities with the same name).
    - Ease of Updating: Data updates become easier as the values are stored in a single place.
    - Localization Support: Standardized lists can be easily localized when the site is translated into different languages.
- However, there are some drawbacks to normalization in the document model:
    - Many-to-One Relationships: Normalizing data with many-to-one relationships is challenging in the document model.
    - Join Limitations: Limited join support in document databases leads to complex joins emulated through multiple queries, impacting performance.
    - Increasing Interconnections: Growing application features may require many-to-many relationships due to increasing data interconnections.
- Relational and document databases both use unique identifiers (foreign keys in relational, document references in the document) to represent many-to-one and many-to-many relationships.

## Data Models

- Many-to-many relationship:
    - Document model: Suitable for applications with mostly one-to-many relationships or tree-structured data.
    - Relational model: Handles simple cases of many-to-many relationships.
    - Graph model: More natural for modelling complex relationships and interconnected data.
- In the **Network Model**, in the tree structure of the hierarchical model, every record has exactly one parent; in the network model, a record could have multiple parents.
    - The links between records in the network model were not foreign keys, but more like pointers in a programming language
    - The only way of accessing a record was to follow a path from a root record along these chains of links. This was called an access path.
- What the **relational model** did, by contrast, was to lay out all the data in the open: a relation (table) is simply a collection of tuples (rows), and that's it
    - In a relational database, the query optimizer automatically decides which parts of the query to execute in which order, and which indexes to use.

- A **graph data model** consists of vertices (nodes/entities) and edges (relationships/arcs). Graphs are good for evolvability: as you add features to your application, a graph can easily be extended. Various types of data can be represented as a graph, including:
  - Social graphs: Vertices represent people, and edges indicate relationships between them.
  - Web graphs: Vertices represent web pages, and edges represent links between pages.
  - Road or rail networks: Vertices represent junctions, and edges represent roads or railway lines.
- Facebook, for example, maintains a single graph incorporating different types of vertices and edges. Vertices can represent people, locations, events, check-ins, and comments, while edges represent friendships, check-ins at locations, comments on posts, event attendance, and more.
- A property graph consists of a set of objects or vertices, and a set of arrows or edges connecting the objects. Vertices and edges can have multiple properties, which are represented as key-value pairs. Each vertex has a unique identifier and can have: A set of outgoing edges. A set of incoming edges.
- if we put graph data in a relational structure, we can query it using SQL, but will have to use recursive cte’s to traverse a variable number of edges
- In a triple-store, all information is stored in the form of very simple three-part statements: (subject, predicate, object).

## Relational Versus Document Databases

The main arguments in favor of the document data model are:

- Schema flexibility
- Better performance due to locality
- Closer alignment with application data structures
- On the other hand, the relational model offers:
- Better support for joins
- Handling many-to-one and many-to-many relationships effectively

### Which data model leads to simpler application code?

For highly interconnected data, the document model can be awkward, the relational model is acceptable, and graph models are the most natural and intuitive choice.

- The document model has limitations: for example, you cannot refer directly to a nested item within a document, but instead you need to say something like “the second item in the list of positions for user 251” (much like an access path in the

hierarchical model). However, as long as documents are not too deeply nested, that is not usually a problem
- if your application does use many-to-many relationships, the document model becomes less appealing. It's possible to reduce the need for joins by denormalizing, but then the application code needs to do additional work to keep the denormalized data consistent.

Schema-on-write (relational databases):
- Relational databases enforce an explicit schema during data writing.
- Similar to static (compile-time) type checking.
- Schemas are valuable for documenting and enforcing consistent structure.

Schema-on-read (document/graph databases):
- Data structure is implicit and interpreted during reading.
- Similar to dynamic (runtime) type checking.
- Used for collections with varying structures.

Schema-on-read is similar to dynamic (runtime) type checking in programming languages, whereas schema-on-write is similar to static (compile-time) type checking.
Data locality for queries
- A document is usually stored as a single continuous string, encoded as JSON, XML, or a binary variant thereof. If your application often needs to access the entire document (for example, to render it on a web page), there is a performance advantage to this storage locality.
    - If data is split across multiple tables, multiple index lookups are required to retrieve it all, which may require more disk seeks and take more time.
- Advantages of Locality:
    - Better performance when accessing large parts of the document simultaneously.
- Disadvantages of Locality:
    - Updating the document often requires rewriting the entire document.
    - It is advisable to keep documents small and avoid writes that increase document size to maintain performance.
- The locality is utilized in other databases:
    - Google's Spanner: Allows table rows to be interleaved or nested within a parent table for managing locality.
    - BigTable, Cassandra and HBase: Implements the column-family concept to manage locality.
- Relational and document databases are becoming more similar over time. A hybrid approach combining relational and document models is beneficial for the future of databases.

## Query Languages for Data

Data models have specific query languages or frameworks:

- SQL for relational databases.
- MapReduce for distributed data processing.
- MongoDB's Aggregation Pipeline for advanced aggregation.
- Cypher for graph databases.
- SPARQL for querying RDF data.
- Datalog for logic programming and databases.

## Imperative vs. declarative

| | **Declarative** | **Imperative** |
|---|---|---|
| Used in | SQL | CODASYL, Many commonly used programming languages |
| Definition | You specify the pattern of the data you want but not how to achieve that goal | An imperative language tells the computer to perform certain operations in a certain order |
| Parallel Execution | Good because the database can use a parallel implementation of the query language. CPUs are getting faster by adding more cores, not by running at significantly higher clock speeds | Imperative code is very hard to parallelize across multiple cores and machines because it specifies instructions that must be performed in a particular order. |

**Imperative:**

```
function getOtherAnimals() {
   var otherAnimals = [];
   for (var i = 0; i < animals.length; i++) {
      if (animals[i].family !== "Sharks") {
         otherAnimals.push(animals[i]);
      }
   }
   return otherAnimals;
}
```

**Declarative:**

```
SELECT name
FROM customers JOIN orders on
  customers.id = orders.id
 WHERE customers.country = 'USA'
 GROUP BY customer_name
      HAVING COUNT(id) > 1
```

## MapReduce Querying

MapReduce is a programming model used for processing large data sets across multiple machines. It combines elements of declarative and imperative approaches, using map and reduce functions. NoSQL datastores like MongoDB and CouchDB support a limited form of MapReduce. While higher-level query languages like SQL can be implemented using MapReduce, SQL is not restricted to single-machine execution. NoSQL systems may inadvertently resemble SQL in certain aspects.
See:
https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

### Types of relationships

| Relationship Type | Description | Example |
|---|---|---|
| One-to-One (1:1) | Each record in Table A is associated with exactly one record in Table B, and vice versa. | An employee has one employee ID and an employee ID is assigned to only one employee. |
| One-to-Many (1:N) | Each record in Table A can be associated with multiple records in Table B, but each record in Table B is associated with only one record in Table A. | A customer can have multiple orders, but each order is associated with only one customer. |
| Many-to-One (N:1) | Each record in Table A is associated with only one record in Table B, but each record in Table B can be associated with multiple records in Table A. | Order can have only one customer, but a customer can have multiple orders. |
| Many-to-Many (N:N) | Multiple records in Table A can be associated with multiple records in Table B, and vice versa. This is typically implemented using a bridge/junction table. | A student can be enrolled in multiple courses, and each course can have multiple students. |