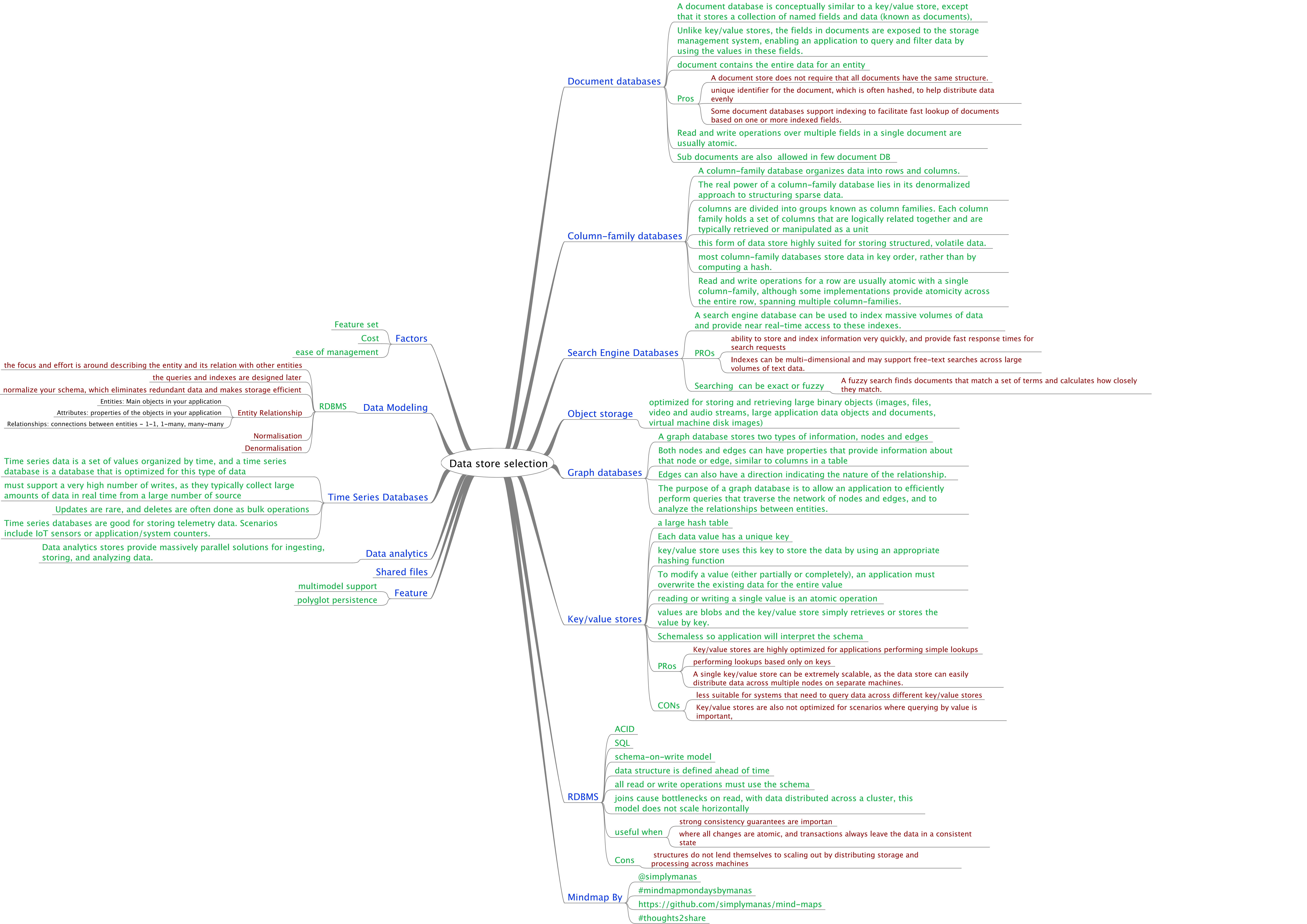

- Data store selection
  - Document databases
    - A document database is conceptually similar to a key/value store, except that it stores a collection of named fields and data (known as documents),
    - Unlike key/value stores, the fields in documents are exposed to the storage management system, enabling an application to query and filter data by using the values in these fields.
    - document contains the entire data for an entity
    - Pros
      - A document store does not require that all documents have the same structure.
      - unique identifier for the document, which is often hashed, to help distribute data evenly
      - Some document databases support indexing to facilitate fast lookup of documents based on one or more indexed fields.
    - Read and write operations over multiple fields in a single document are usually atomic.
    - Sub documents are also allowed in few document DB
  - Column-family databases
    - A column-family database organizes data into rows and columns.
    - The real power of a column-family database lies in its denormalized approach to structuring sparse data.
    - columns are divided into groups known as column families. Each column family holds a set of columns that are logically related together and are typically retrieved or manipulated as a unit
    - this form of data store highly suited for storing structured, volatile data.
    - most column-family databases store data in key order, rather than by computing a hash.
    - Read and write operations for a row are usually atomic with a single column-family, although some implementations provide atomicity across the entire row, spanning multiple column-families.
  - Search Engine Databases
    - A search engine database can be used to index massive volumes of data and provide near real-time access to these indexes.
    - PROs
      - ability to store and index information very quickly, and provide fast response times for search requests
      - Indexes can be multi-dimensional and may support free-text searches across large volumes of text data.
    - Searching can be exact or fuzzy
      - A fuzzy search finds documents that match a set of terms and calculates how closely they match.
  - Object storage
    - optimized for storing and retrieving large binary objects (images, files, video and audio streams, large application data objects and documents, virtual machine disk images)
  - Graph databases
    - A graph database stores two types of information, nodes and edges
    - Both nodes and edges can have properties that provide information about that node or edge, similar to columns in a table
    - Edges can also have a direction indicating the nature of the relationship.
    - The purpose of a graph database is to allow an application to efficiently perform queries that traverse the network of nodes and edges, and to analyze the relationships between entities.
  - Key/value stores
    - a large hash table
    - Each data value has a unique key
    - key/value store uses this key to store the data by using an appropriate hashing function
    - To modify a value (either partially or completely), an application must overwrite the existing data for the entire value
    - reading or writing a single value is an atomic operation
    - values are blobs and the key/value store simply retrieves or stores the value by key.
    - Schemaless so application will interpret the schema
    - PRos
      - Key/value stores are highly optimized for applications performing simple lookups
      - performing lookups based only on keys
      - A single key/value store can be extremely scalable, as the data store can easily distribute data across multiple nodes on separate machines.
    - CONs
      - less suitable for systems that need to query data across different key/value stores
      - Key/value stores are also not optimized for scenarios where querying by value is important,
  - RDBMS
    - ACID
    - SQL
    - schema-on-write model
    - data structure is defined ahead of time
    - all read or write operations must use the schema
    - joins cause bottlenecks on read, with data distributed across a cluster, this model does not scale horizontally
    - useful when
      - strong consistency guarantees are importan
      - where all changes are atomic, and transactions always leave the data in a consistent state
    - Cons
      - structures do not lend themselves to scaling out by distributing storage and processing across machines
  - Mindmap By
    - @simplymanas
    - #mindmapmondaysbymanas
    - https://github.com/simplymanas/mind-maps
    - #thoughts2share
  - Factors
    - Feature set
    - Cost
    - ease of management
  - Data Modeling
    - RDBMS
      - the focus and effort is around describing the entity and its relation with other entities
      - the queries and indexes are designed later
      - normalize your schema, which eliminates redundant data and makes storage efficient
      - Entity Relationship
        - Entities: Main objects in your application
        - Attributes: properties of the objects in your application
        - Relationships: connections between entities - 1-1, 1-many, many-many
      - Normalisation
      - Denormalisation
  - Time Series Databases
    - Time series data is a set of values organized by time, and a time series database is a database that is optimized for this type of data
    - must support a very high number of writes, as they typically collect large amounts of data in real time from a large number of source
    - Updates are rare, and deletes are often done as bulk operations
    - Time series databases are good for storing telemetry data. Scenarios include IoT sensors or application/system counters.
  - Data analytics
    - Data analytics stores provide massively parallel solutions for ingesting, storing, and analyzing data.
  - Shared files
  - Feature
    - multimodel support
    - polyglot persistence