**Choosing the Best Database**

Factors for choosing a database:

1. Structure of data
2. Query Pattern
3. Scale

Caching:

1. When you want to prevent a lot of calls to the main database, you can use caching.
2. If you are making a remote call to a different service, and it has a high latency, we can cache the response of that service in our system.
3. We have a key-value kind of structure in a cache. Ex: Redis, Memcache

File Storage

1. Data like images, videos, etc. are stored in Blob Storage. They are not databases, they can not be queried upon. Ex: AWS S3.
2. CDN: Distributing the same thin(image, video)g geographically in a lot of locations so that it available easily and faster. Accessing S3 would be slow compared to accessing AWS S3.

Text Search Engine:

1. When you want to search something a lot of data. For example, there are many products with names and description. When a user search something, we have to give them the best result and this is called Text Searching. Similarly, consider Netfix and we use it to search movie by and the search can be anything movie name, genre, etc. and we should give the best result.
2. Ex: Elastic Search/Solr. Both are built on top of Apache Lucine.
3. Fuzzy Search: Think user searched ‘airprot’ instead of ‘airport’, so we want these kinds of mistakes which a user can make to found out by the system. How it does that? Finding the edit distance (eg: Here for this example, it is 2) between the entered word and the words in DB. So, we must provide a level of fuzziness with a search.
4. Elastic Search/Solr are not databases, they are search engines. DB give you a guarantee that your data will never be lost. But, ES/Solr doesn’t give that guarantee. So, they should not be used as primary Database.

Metrics:

1. Suppose the application like Grafana where a lot of systems are pushing the data/metrics continuously like throughput, CPU utilization, then we can use Time Series Database.
2. Time Series Database: Extension of Relational Databases but some functionalities taken and some added. RDBMS give you flexibility to update, query anything but in this case, we don’t do random updates, but only do append only updates in sequential mode. The read queries that we would do in Time Series Database would be a bulk read query for a time range (date, last 60 mins, last week), etc. So, TSD are optimized for this kind of a query pattern. Formally, *“Time Series Databases (TSDB) are designed to store and analyze event data, time series, or time-stamped data, often streamed from IoT devices, and enables graphing, monitoring and analyzing changes over time.”*
3. Example: InfluxDB, OpenTSDB.

Analytics:

1. When we want to do analytics like “Which geography Is giving how many orders?”, or “What is the most popular item?”
2. Data Warehouse: Is a large DB in which we can dump a lot of data and provide various querying capabilities on top of that data to serve a lot of reports. Generally, used for offline reporting.

Diagram

Description automatically generatedDiagram

Description automatically generated

Relational vs Non-Relational Database

1. Structured Information: Information which can easily modelled in form of tables and tables would have rows and columns. Example User (username, email, etc) is structured information.
   1. If we have structured data, then we check if it needs ACID property?
      1. Choose RDBMS
      2. Payment System needs to be Transactional, so ACID is needed.
      3. Ex: MySQL, Oracle SQL Server, Postgres
   2. If we have structured data but we do not need ACID properties, then we can choose any Relational or Non-Relational Database.
2. If we do not have structured data, then for example we have a lot of data types and a lot of different types of queries, then we can choose a Document DB
   1. Consider we have a catalogue kind of service like amazon where we have a lot of products and they have a lot of different properties (Ex: Shirt has different properties color, size, but Fridge has different attributes like power, volume, etc. Other different items will have other different attributes.) When we want to not only see them, but also want to query them, then we would want something like a Document DB. If it was just about seeing the products, then we could have dumped the whole data as a JSON in a relational database and seen it.. but here want to query them as well .. and that will not be efficient if we do it on a json stored in a relational database.
   2. Databases which are optimized for these kinds of searches where there are a lot of data and they have a lot of different types of attributes, and there can be a lot of different kinds of queries on them, then we can choose a Document DB.
   3. Ex: MongoDB, CouchBase.
   4. Elastic Search/Solr are special cases of this document DBs.
3. If we have an ever-increasing data & finite queries
   1. Example: Uber drivers are continuously sending location data every 5 secs. So, we have X location records by all the drivers per day. This X will not be a constant. It is a growing number since the drivers will increase day-by-day. This data will not be increasing linearly but increasing more than that daily. Also, the query is finite, we will always be querying Track Location Pings for Driver X for Trip T.
   2. Columnar Database: These are columnar oriented DB.
   3. Example: Cassandra, Hbase.

**Example:** Amazon

1. Inventory Of Items: RDBMS / Cassandra
   1. The count of items is important. One user should be able to commit the transaction if there is just 1 item and 10 are trying to buy.
   2. Numbers of Orders are increasing so the database is also increasing, so should we use Cassandra?
   3. The answer is the best fit would be to use a combination of both. Use RDBMS for storing the orders which are just places or not yet delivered. But, as soon as the order lifecycle is completed, then remove it from RDBMS and move it to Cassandra.
2. Reporting:
   1. Get me all the users who bought some kind of sugar in the last 5 days.
   2. Sugar is not a single product; different sellers may be selling different companies of sugar and different types of it. So, sugar will be a lot of itemIDs. On top of all these itemIDs, will be a lot of orders which are in Cassandra, or mySQL.
   3. Now, for this there are different types of products with different attributes, and we thought we use DocumentDB for it, right?
   4. So, basically we can store a subset of order information which is required for querying into he MongoDB like userID X, orderID X, .. itemID Y.. Qty 10 Data Z.. Then we can query this mongoDB which will return a list of users and a list of orderIDs. We can then query MySQL and Cassandra for the same and get the details from there.
   5. So, basically, we use all 3 types of DBs. In a real word scenario, we use a combination of Databases.