Choosing DB

|  |  |
| --- | --- |
| **Redis DB** | **RDBMS** |
| 1) No self referencing and hence need to create multiple objects | 1) Self referencing is possible and hence single object can be self referenced |
| 2) Redis only offers basic security (in term of access rights) at the instance level. | 2) RDBMS all provide fine grained per-object access control lists (or role management). |
| 3) Organizing and viewing tables in tree structure is very complex and takes lot of efforts, inserting data in tree structure requires many pointers and keys to be stored. | 3) Tree structure os db insertion is possible and comparatively easier. |
| 4) Need some RND to write logic to handle relations between multiple tables and it’s difficult to fetch information from multiple tables. (No joins or query language) | 4) Provides sql ‘JOIN’ operations which can handle this kind of information fetching from multiple tables. |
| 5) To fetch the data by recursively traversing the tree, need to write own logic or API’s which is time consuming. Also need to RND on every function to find the most efficient way. | 5) Easy queries can be used to fetch the information. |
| 6) Data storage is in the form of Strings, lists, sets, hashes, sorted sets | 6) Data storage is in the form of tables of rows, views over tables which is efficient to store tree structural data. |
| 7) The biggest caveat to using Redis, is that it is entirely in memory. If your relational dataset is 2.5GB (not that large), you’ll need a $160/month Linode (4GB RAM) to keep it in Redis. | 7) Memory usage is comparatively better using Relational DB. |
| 8) Redis is a specialized in-memory **data structure server**. Hence, need to manage memory when the data to be stored is going to grow in future. | 8) RDBMS usually stores the data on disks, and cache part of the data in memory. With a RDBMS, you can manage more data than you have memory. With Redis, you cannot. |
| 9) In key-value stores, this responsibility falls squarely on the application logic and many people might leave or miss out this crucial responsibility. | 9) In the relational model, there is an built-in and foolproof method of ensuring and enforcing basic constraint like(unique key, range values etc) and rules at the database layer through primary keys and foreign keys. |
| 10) A unique Redis instance is not scalable. It only runs on one CPU core in single-threaded mode. To get scalability, several Redis instances must be deployed and started. Distribution and sharding are done on client-side (i.e. the developer has to take care of them) | 10) If you compare them to a unique Redis instance, most RDBMS provide more scalability (typically providing parallelism at the connection level). They are multi-processed (Oracle, PostgreSQL, ...) or multi-threaded (MySQL, Microsoft SQL Server, ... ), taking benefits of multi-cores machines. |

Also, Redis is a data structure server, there is no query language (only commands) and no support for a relational algebra. You cannot submit ad-hoc queries (like you can using SQL on a RDBMS). All data accesses should be anticipated by the developer, and proper data access paths must be designed.

Conclusion is that If we’re going to be needing an SQL "JOIN" kind of operations or recursive queries relating multiple tables, then you won't want to use Redis, nor any other non-relational database. If we're only going to be doing key:value pair queries like RSS Feed or tweets, then we would want to use Redis as it only provides key-based search queries, which can be a big problem if we are dealing with inherently relational data.