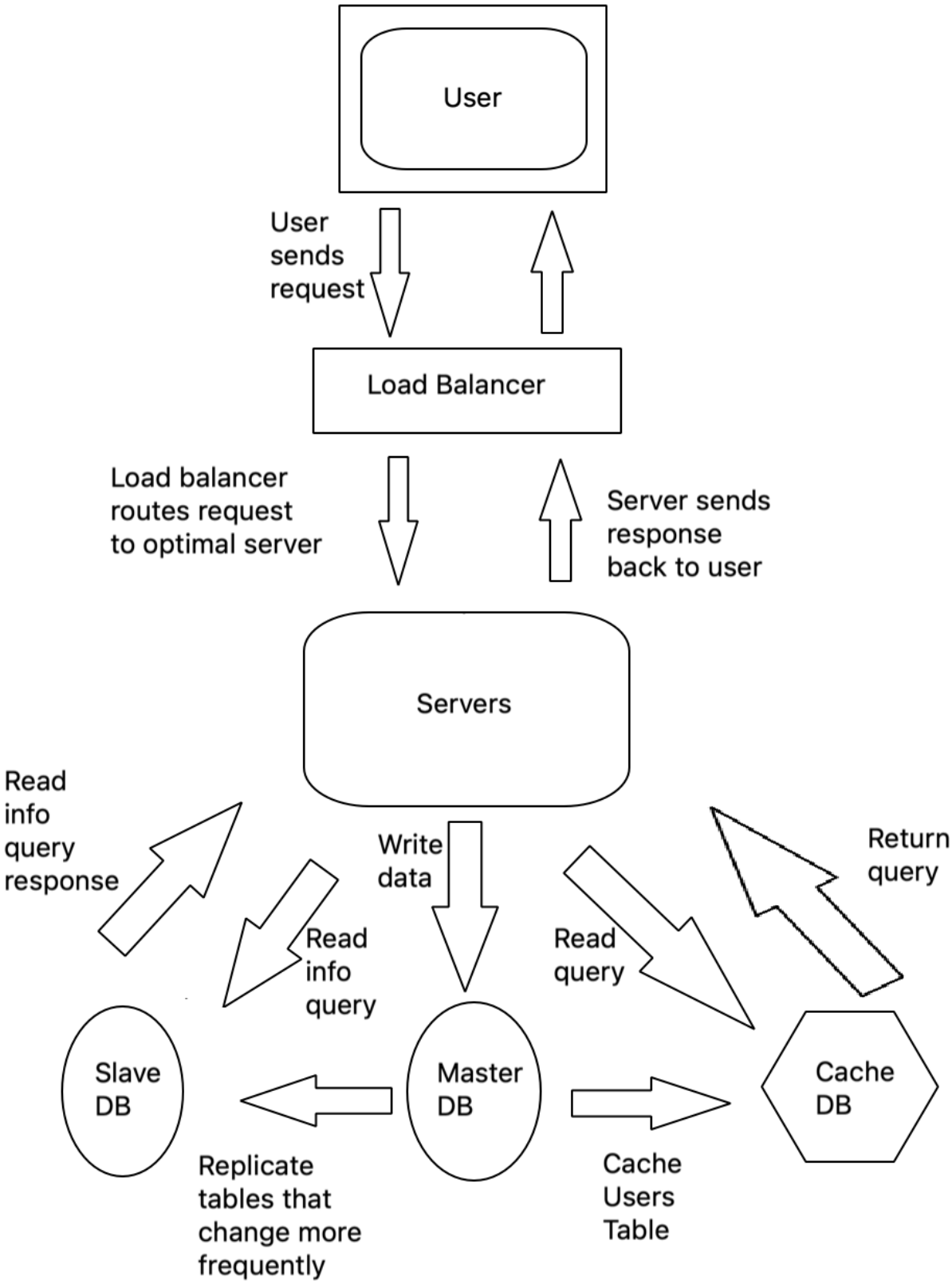
User
User sends request
Load Balancer
Load balancer routes request to optimal server
Server sends response back to user
Servers
Read info query response
Write data
Return query
Read info query
Read query
Slave DB
Master DB
Cache DB
Replicate tables that change more frequently
Cache Users Table

Above is a high level overview of how the system would work.

1. A user submits a request
2. The load balancer handles the request and sends it to a free server to work on
3.
    a. If it is a read query from the visit_table, the server will read the query from the slave db
    b. If it is a read query from the user table, the server will read from the cached data
    c. If it is a write operation, the server will write to the master database, which will then update the slave db
4. The server will get the result back from the query and send back the result to the user

For these tools, I would use a relational database such as postgreSQL, since the data is relational and the read/writes are not that heavy. Also, postgres scales really well, so it should be more than enough for this type of problem. I would also use redis as a cache to allow for quicker lookups for user info since that data shouldn't change much. I replicate the master database to create a read-only database as this will free up the master database to handle the write operations. Lastly, I would use a load balancer to route the requests to the freest servers to increase traffic productivity.