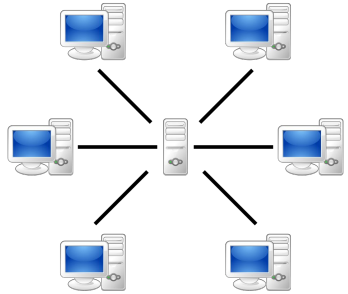
Database Discussion

By:
Travis & Carson

Analysis of System Requirements

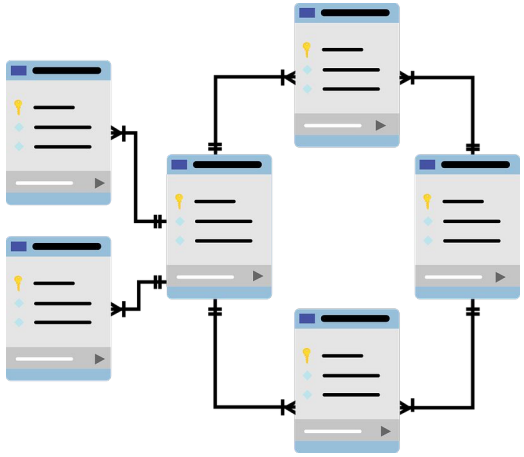


- **Initial background for the System**
  - ***Organization:*** AggieSTEM
  - ***Expected Users:*** Approximately 100 simultaneous users under peak conditions.
  - ***Expected Traffic:*** Lightweight, mostly used as a data repository and communication method.
  - ***Usage:*** Data storage, communication, and group organization.
  - ***Hardware:*** Initially, the database and server will reside on a single AWS node.

- **Future considerations made while engineering the System**
  - ***Ease-of-use:*** Delivered Architecture made use of the common SQL language.
  - ***Flexibility:*** To achieve maximum flexibility, we chose a non-relational NoSQL approach.
  - ***Scalability:*** Although the server just needs to support 100 now, future may demand more.
  - ***Software:*** We used a mature, popular open-source framework with active contributors.

Database Design / Schema

- ***User Table:*** Stores user profile information
  - Username, email, job position, etc...
- ***Security Table:*** Stores user passwords and security question answers
  - This table is always encrypted
- ***Group Table:*** Stores user group information
  - Group name, minimum access level, list of group members, etc...
- ***Library Access Table:*** Stores a list of libraries that a user is able access
  - Each user has their own list of libraries they are able to access
- ***Library Table:*** Stores individual library information
  - Library name, minimum permission level, list of content, etc...
- ***Content Table:*** Stores individual pieces of content
  - Name of content and the content's data



Rationale Behind Software Stack, MongoDB

- ***Scalable Connection Limit***
  - Limited only by the number of available file descriptors on the server
  - Around 1024 by default for most linux operating systems
- ***Non-Relational Database***
  - Updates to the schema can be made without breaking relationships between tables
- ***Ad Hoc Queries***
  - Able to make one-off queries
- ***High Availability and Scalability***
  - Built-in replication and failover
  - Native sharding

Tacit Deletion of Content

- ***Identified Problem:*** Often, website content is deleted on accident. Rather than immediately wiping the data from the server, AggieSTEM administrators would benefit from the ability to roll back deletions, or clean stale data from their servers at whichever interval they so choose.

- ***Developed Solution:*** To achieve this, when an AggieSTEM website user prompts the server for deletion of an item (e.g., a group, library, or file), A flag is raised to prevent the “deleted” content from being served on query. Functionally, this “deletes” the content from the website while keeping it fully accessible to administrators with database credentials. This is essentially a “recycling bin”.

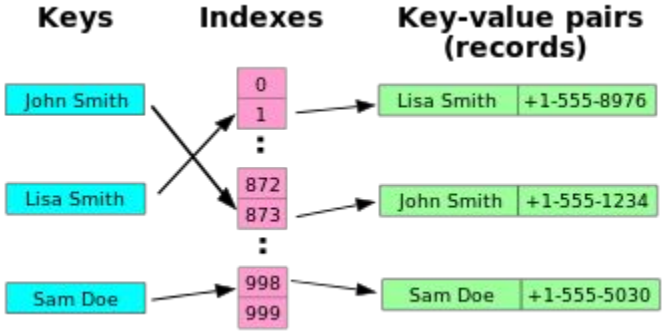
- ***Impacts:*** Flexibility, Ease-Of-Use, Administrator Friendliness.

Query-Driven Index Optimization

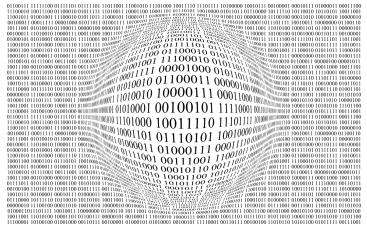
- ***Identified Problem:*** AggieSTEM requires their modestly sized database to be quickly queryable such that the access time to served data is minimal, leading to seamless site navigation.

- ***Developed Solution:*** In order to successfully optimize the database and improve its benchmark performance on queries, for each and every query that serves content, we crafted an associated ***index*** that optimizes primary key search speeds. This trades non-indexed linear scan access times of O(n) for an indexed O(1) access time by computing hash tables for the indexed columns.

- ***Impacts:*** Overall Website Performance, User Experiences.



Generation of Test Data



- ***Identified Problem:*** There needed to be some form of information in the database to test the front and back end. Randomly generated data could have been used. However, this would make testing the various table interactions or database queries difficult.

- ***Developed Solution:*** A script was written to generate structured random data. The content itself was meaningless, but it generated such that it followed the database schema. For example, the generated users had valid user profiles and could be logged into. Libraries had unique lists of content.

- ***Impacts:*** This allowed more elaborate testing of the front and back end environments.

Tiered Access to Database Content

- ***Identified Problem:*** To provide an extra layer of security, the AggieSTEM website needed a simple client-side authentication method to verify user credentials before serving content.

- ***Developed Solution:*** All groups, libraries, and content records were given an associated minimum required level of security to access them automatically. This access credential is checked client-side before data is ever served to the front end of the site.

- ***Impacts:*** This provides an additional layer of protection for the AggieSTEM administrators against user content access violations.