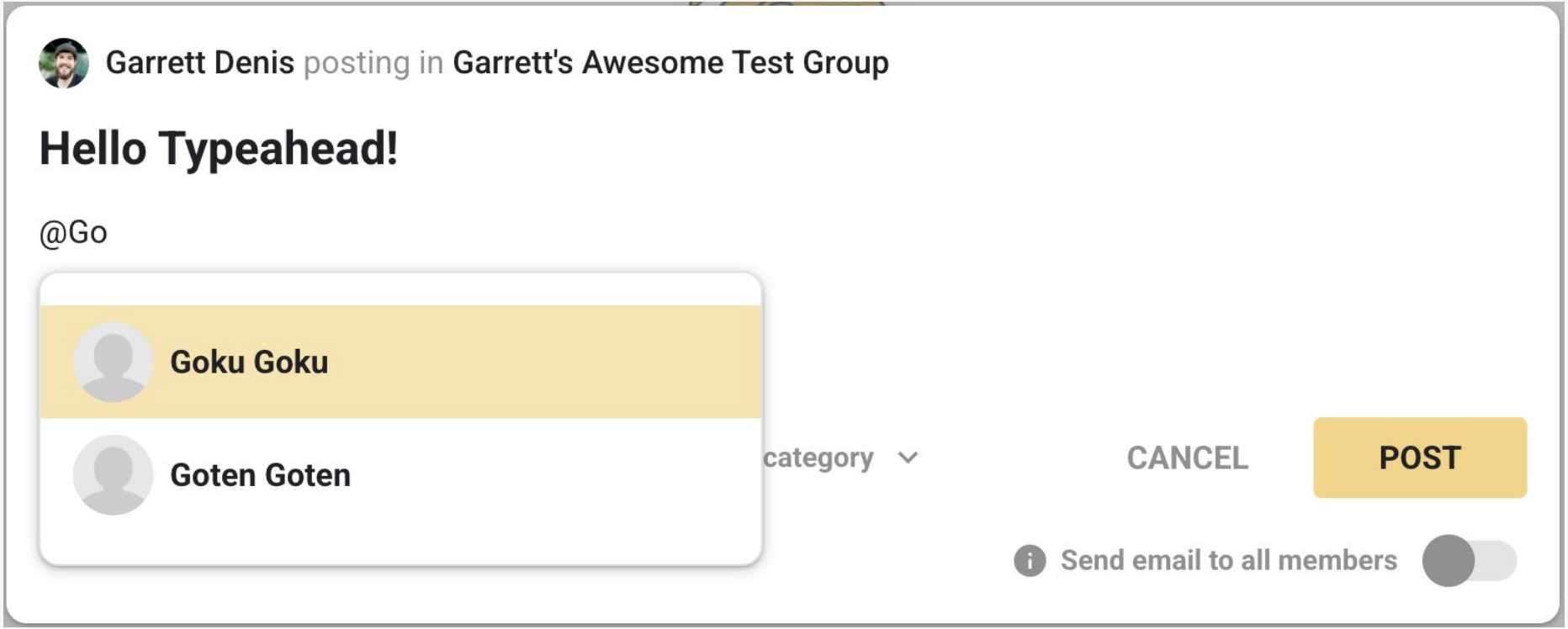
To Trie or Not to Trie:

Typeahead Completion Using Redis

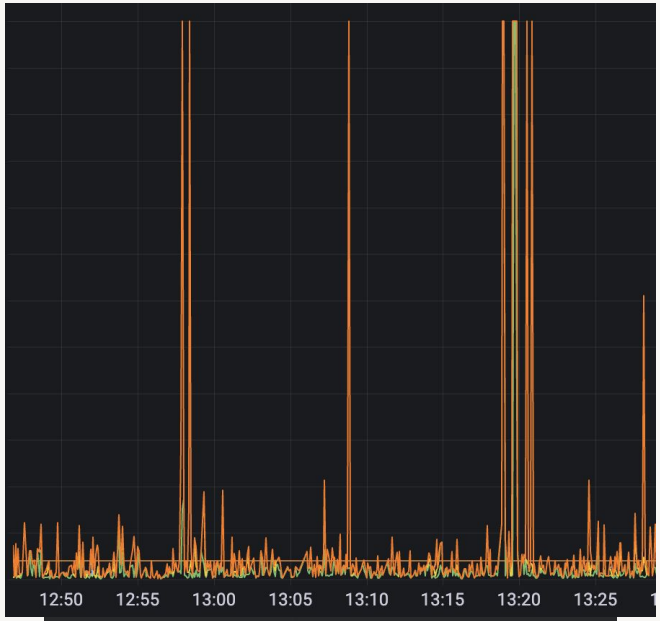
Garrett Denis
Principal Engineer @ Skool

The Feature


Garrett Denis posting in Garrett's Awesome Test Group
Hello Typeahead!
@Go
Goku Goku
Goten Goten
category
CANCEL
POST
Send email to all members

The Problem

- @mentions lead to frequent search queries in Postgres
- Slow at scale
- Hits some of our most critical tables



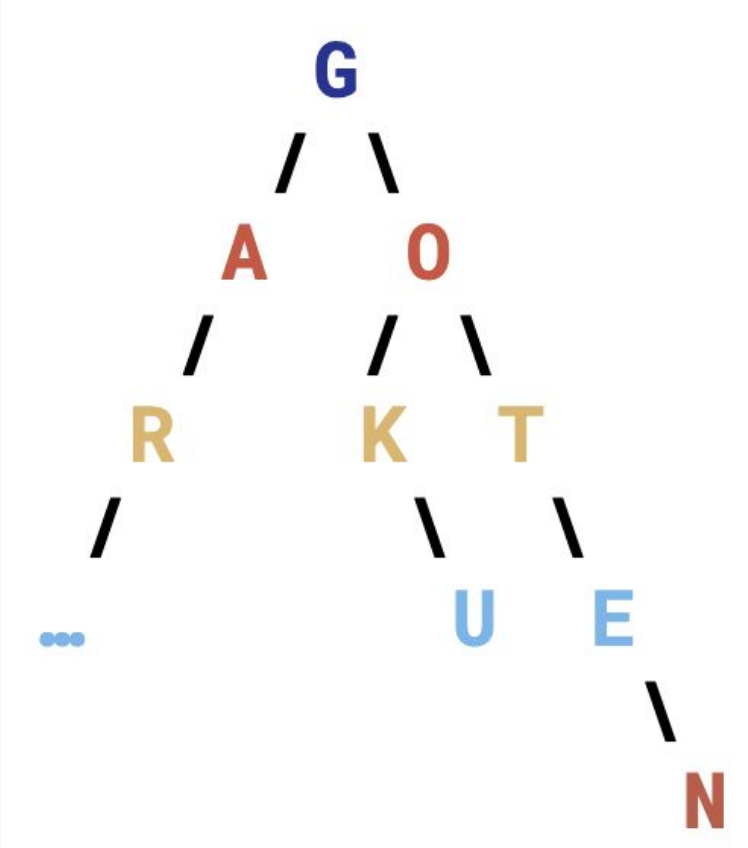
Name	Status	Time
users	200	159 ms
users	408	3.04 s
users	200	237 ms

The Goals

- Make it fast
- Make it scale
- Make it isolated

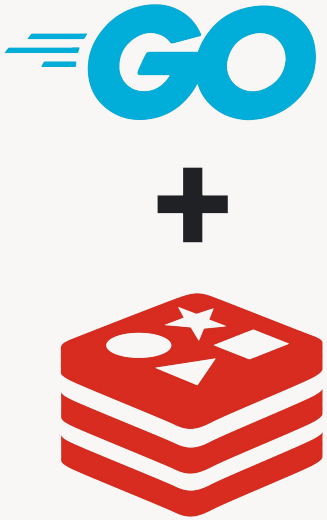
The Solution

- Use a prefix trie
- Lookup and Insert in *O(n)*
- Space complexity of *O(n * k)*, where *k* is the average username length
- Put the data in Redis



The Implementation

- Given a user, compute and store each prefix for first and last names
- Store a hash set with the first and last names for sorting matches
- Store group membership details
- To find a match we pass the prefix and sort the ids in the matching set using the hash set



The Implementation - Adding Users

```go
searchFullName := pf.SearchName(firstName + " " + lastName)
searchLastName := pf.SearchName(lastName)
sortName := firstName + lastName

fullNameRunes := []rune(searchFullName)
lastNameRunes := []rune(searchLastName)
keys := make([]string, len(fullNameRunes)+len(lastNameRunes)+1)
for i := 1; i ≤ len(fullNameRunes); i++ {
    keys[i-1] = s.FullNameIndexKey(string(fullNameRunes[:i]))
}
for i := 1; i ≤ len(lastNameRunes); i++ {
    keys[len(fullNameRunes)+i-1] = s.LastNameIndexKey(
        string(lastNameRunes[:i]),
    )
}
keys[len(keys)-1] = s.UserDataKey(userID.String())
```

The Implementation - Adding Users (cont...)

```go
func SearchName(s string) string {
    var t []rune
    for _, c := range text.RemoveMarks(s) {
        if unicode.IsLetter(c) ||
            unicode.IsDigit(c) {
            t = append(t, unicode.ToLower(c))
        }
    }
    return string(t)
}
```

The Implementation - Finding Matches

```go
searchPrefix := pf.SearchName(prefix)
keys := []string{
    s.FullNameIndexKey(searchPrefix),
    s.LastNameIndexKey(searchPrefix),
    s.ResultsKey(groupID, searchPrefix),
    s.UserDataKey("*→sort"),
}
if groupID ≠ core.NoID {
    keys = append(
        keys,
        s.MembershipsIndexKey(groupID.String()),
    )
}

args := []any{
    limit,
    boolToInt(groupID ≠ core.NoID),
}
```

```go
results, err := s.scriptFindMatches.Run(
    ctx,
    s.conn,
    keys,
    args ... ,
).Result()
if err ≠ nil {
    return nil, fmt.Errorf(
        "redis script find matches: %w",
        err,
    )
}

matches := results.([]any)
ids := make([]core.ID, len(matches))
for i := range matches {
    ids[i] = core.ID(matches[i].(string))
}
return ids, nil
```

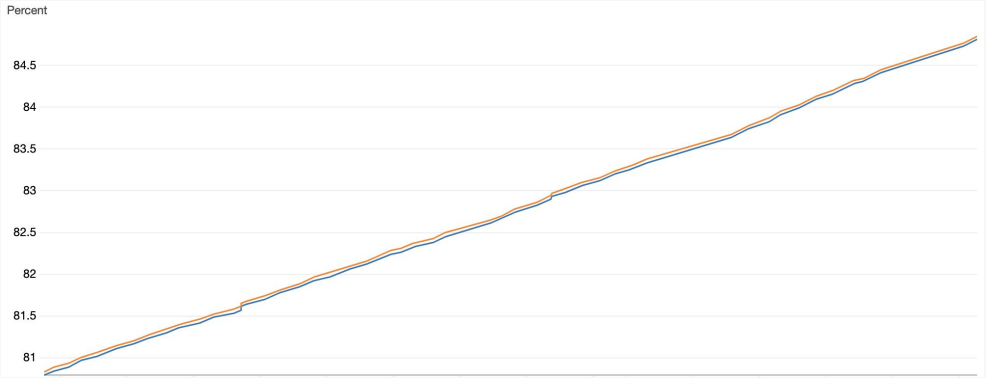
The Results

- It’s fast!
- It scales!
- It’s isolated!

Name	Status	Time ▲
{:} users	200	37 ms
{:} users	200	36 ms
{:} users	200	38 ms
{:} users	200	38 ms
{:} users	200	40 ms

The Gotchas

- Keeping things in sync
- Memory usage over time
- Advanced filtering



The Takeaways

- Tries are a great data structure when you need to quickly look something up by prefix

- Separation of concerns is a good thing

- Make sure you consider both time and space complexity

The End

Slides:

Thank you!
garrett@skool.com
Come build with us:
https://www.skool.com/careers