Unicorn

A System for Searching the Social Graph
The Social Graph

- Inter-relationships between people and things
  - Nodes: People, things (sites, images, gender, etc)
  - Edges: Relationships (friend, likes, likers, etc)
- **Goal:** efficiency and scalability with complex set operations
- Data is sparse
Data Model and API

- Data is sparse → Adjacency lists
- term: \(<\text{edge-type}>:<\text{id}>\)
- Query language:
  - AND: Union
  - OR: Intersection
  - DIFFERENCE: subtraction

- E.g:

\[(\text{difference} \ (\text{and} \ \text{friend}:5 \ \text{gender}:1) \ \text{friend}:6)\]
Architecture

- Partitioned through sharding by result-id (query output)
- Top aggregator:
  - Communicates with client and rack aggregators
- Rack aggregator:
  - Communicate with index servers
- Index server:
  - Adjacency lists for a particular shard
  - Indexes built using Hadoop Framework
  - Timestamp for latest update
Architecture - Verticals
Typeahead

- Get results while typing: person, events, pages
  - “gr”, “gre”, “greg”
- Social relevance
- **WeakAnd**
  - Allow hits outside
- **StrongOr**
  - Ensure fraction of hits inside

(weak-and (term friend:3 :optional-hits 2) (term melanie) (term mars*))
Graph Search

- More advanced queries more than one edge away from the source node - requires
- **APPLY**: Apply outer operation on result from inner query
  - e.g. *Friends-of-Friends*: (apply friend: friend:5)
- For some data it makes sense to partially denormalize it to save space
- **EXTRACT**: Looks up data from a forward index
Lineage

- Privacy is handled through lineage
- Lineage shows edges traversed to yield a result
- Frontend queries authorization service using lineage
  - check if sufficient number of edges are visible
  - include IDs if so

Q: (apply friend: friend:5) → Chuck (10), ...

Jon(5), Amy (8), Bob(9), Chuck(10),

![Diagrams showing friendships and IDs]
Conclusion

- Unicorn applies core IR concepts in the social graph domain
- Index edges and entities
- Serve complex graph queries
  - APPLY & EXTRACT
- Round-trip algorithm for queries without using denormalization
- Some nested queries require truncation of inner results to be efficient