# Enterprise Distributed Systems

## Peer Reviewed Class Project – LinkedIn.com Simulation (Fall 2018)

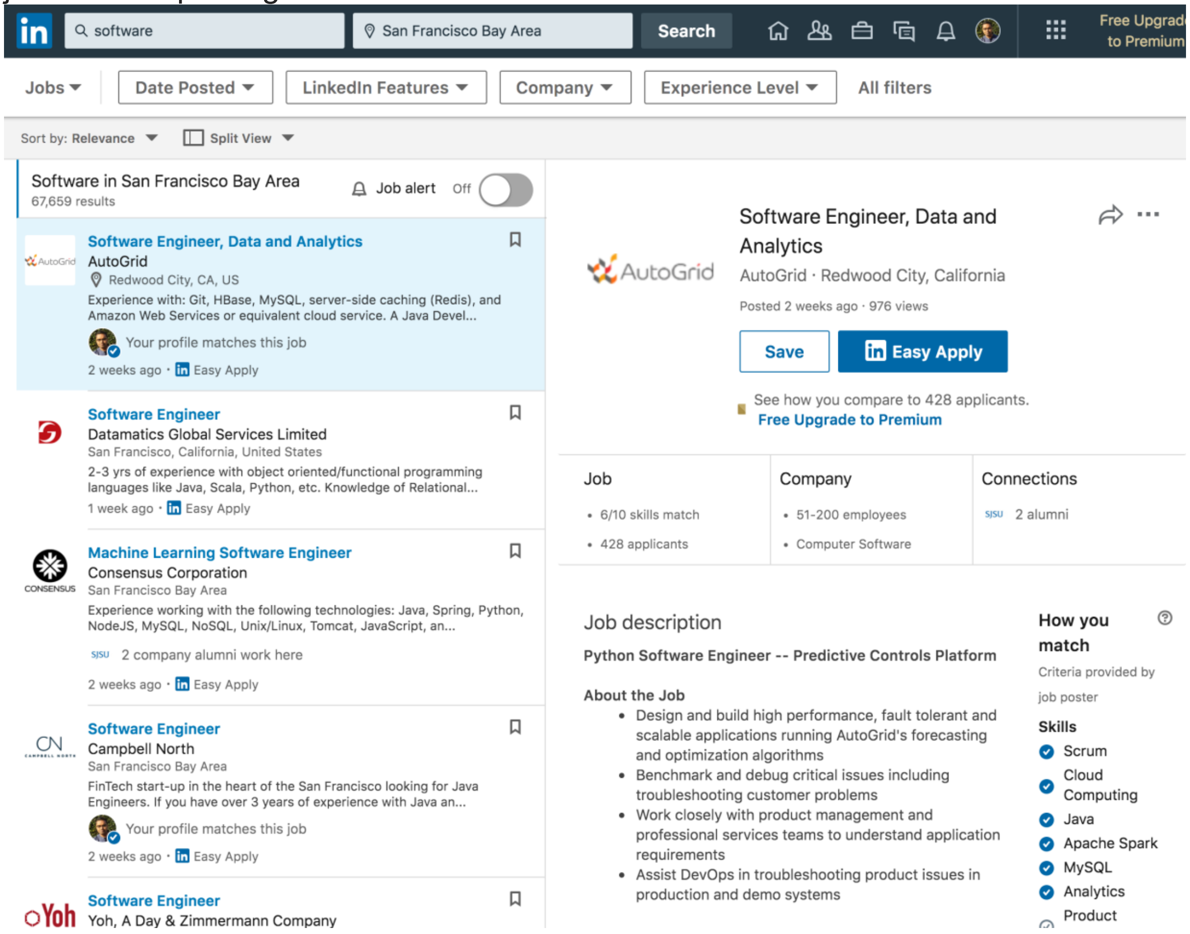
**<u>Project Due Date</u>: 2nd December 2018**
**Demo: 12/4/2018**
**Presentation: 11/27/2018( Upload Presentation to canvas before the class.)**

**This is a group project up to 6 people in the team.**
**Turn in the following on or before November 11, 2018 on canvas. No late submissions will be accepted!**

- An API design document of services provided by the application.

## <u>LinkedIn</u>

LinkedIn is a business and employment-oriented service that operates via websites and mobile apps. Founded on December 28, 2002, and launched on May 5, 2003, it is mainly used for professional networking, including employers posting jobs and job seekers posting their CVs



Software in San Francisco Bay Area
67,659 results
Job alert Off

**Software Engineer, Data and Analytics**
AutoGrid
Redwood City, CA, US
Experience with: Git, HBase, MySQL, server-side caching (Redis), and Amazon Web Services or equivalent cloud service. A Java Devel...
Your profile matches this job
2 weeks ago · Easy Apply

**Software Engineer**
Datamatics Global Services Limited
San Francisco, California, United States
2-3 yrs of experience with object oriented/functional programming languages like Java, Scala, Python, etc. Knowledge of Relational...
1 week ago · Easy Apply

**Machine Learning Software Engineer**
Consensus Corporation
San Francisco Bay Area
Experience working with the following technologies: Java, Spring, Python, NodeJS, MySQL, NoSQL, Unix/Linux, Tomcat, JavaScript, an...
2 company alumni work here
2 weeks ago · Easy Apply

**Software Engineer**
Campbell North
San Francisco Bay Area
FinTech start-up in the heart of the San Francisco looking for Java Engineers. If you have over 3 years of experience with Java an...
Your profile matches this job
2 weeks ago · Easy Apply

**Software Engineer**
Yoh, A Day & Zimmermann Company

Software Engineer, Data and Analytics
AutoGrid · Redwood City, California
Posted 2 weeks ago · 976 views
Save | Easy Apply
See how you compare to 428 applicants.
Free Upgrade to Premium

| Job | Company | Connections |
|---|---|---|
| • 6/10 skills match | • 51-200 employees | 2 alumni |
| • 428 applicants | • Computer Software | |

Job description

**Python Software Engineer -- Predictive Controls Platform**

**About the Job**
- Design and build high performance, fault tolerant and scalable applications running AutoGrid's forecasting and optimization algorithms
- Benchmark and debug critical issues including troubleshooting customer problems
- Work closely with product management and professional services teams to understand application requirements
- Assist DevOps in troubleshooting product issues in production and demo systems

How you match
Criteria provided by job poster
Skills
- ✓ Scrum
- ✓ Cloud Computing
- ✓ Java
- ✓ Apache Spark
- ✓ MySQL
- ✓ Analytics
- Product

## Project Overview

In this project, you will design a 3-tier application that will implement the functions of **LinkedIn** for different services. You will build on the techniques used in your lab assignments to create the system.

In LinkedIn prototype, you will manage and implement different types of objects:

- Applicant
- Job Search board
- Application
- Recruiter

For each type of object, you will also need to implement an associated **database schema** that will be responsible for representing how a given object should be stored in a database.

Your system should be built upon some type of distributed architecture. You have to use message queues as the middleware technology. You will be given a great deal of "artistic liberty" in how you choose to implement the system. **These are the basic fields that needed. You can modify this schema according to you need and assign primary keys and foreign keys. Example: Create relation between Applicant and Jobs, applications etc. Above mentioned are just examples.**

Your system must support the following types of entities:

• **Applicant** – It represents information about an individual user registered on LinkedIn. You must manage the following information for this entity:

⇒ First Name
⇒ Last Name
⇒ Address
⇒ City
⇒ State
⇒ Zip Code
⇒ Experience
⇒ Education
⇒ Skills
⇒ Profile Summary
⇒ Resume

• **Job**

⇒ Job ID
⇒ Title
⇒ Job Description
⇒ Industry

⇒ Employment type
⇒ Location
⇒ Job function
⇒ Company Logo
⇒ Number of applicants
⇒ Number of Views
⇒ Posted Date/Time

- **Job Search Board**
  ⇒ List of Job
  ⇒ Applicant can view individual job details by clicking on a particular job.
  ⇒ Applicant can apply on individual job.
  ⇒ On clicking Apply custom LinkedIn provided application which you will develop as a template should open in a new tab.
  ⇒ Some applications can have Easy Apply button which will open a dialog which will contain pre-filled data of the user. (In this case only first name, last name, email and resume should be pre-filled).

- **Job Application** – It represent job application to be submitted by the applicant. Prepare a standard template using below fields.
  ⇒ Application Id
  ⇒ Upload Resume
  ⇒ Optional Upload Cover Letter
  ⇒ First Name
  ⇒ Last Name
  ⇒ Address
  ⇒ How Did you hear about us.
  ⇒ Diversity and sponsorship questions.
  ⇒ Disability Question.

- **Recruiter** – It represents information about the recruiter of the Company. You must manage the following state information for this entity:
  ⇒ Admin ID
  ⇒ First Name
  ⇒ Last Name
  ⇒ Address
  ⇒ City
  ⇒ State
  ⇒ Zip Code
  ⇒ Phone number
  ⇒ Email
  ⇒ Company Name

## <u>Project Requirements</u>

Your project will consist of three tiers:
- The client tier, where the user will interact with your system
- The middle tier/middleware/messaging system, where the majority of the processing takes place
- The third tier, comprising a database to store the state of your entity objects

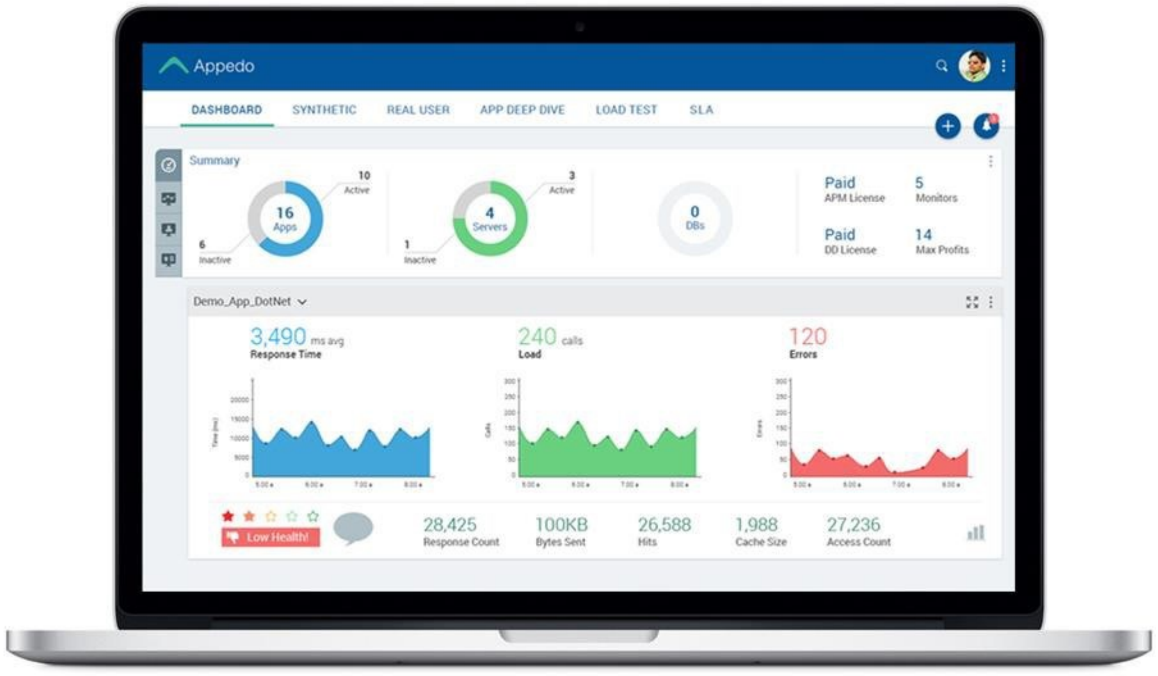
## Tier 1 — Client Requirements

The client will be an application that allows a user to do the following:

- **Applicant Module/Service:**
  - Create a LinkedIn account (Reference LinkedIn websites for fields information)
  - Delete an existing Account.
  - Change a applicant's information (name, address, profile image etc) – *This function must support the ability to change ALL attributes*
  - Display information about an applicant.
  - Search listing for different jobs
  - Search Jobs/Companies
    - Filter based on location, Job type, industry and so on (Same as LinkedIn)
  - Apply for a Job
  - Save a job and apply it later.
  - Send/Receive messages
  - Make/Accept Connections
  - View list of connections

- **Recruiter Module/Service:**
  - Allow only the authorized (admin user) to access Module
  - Add Job Posting to the systems.
  - Search for job postings and edit it.
  - View/Modify his/her account
  - View posted applications (should be able to view Resumes of the applicants) (Use ReactPDF).
  - Send/Receive messages.
  - Make/Accept Connections.
  - View list of connections

- Sample Admin Analysis Report



Recruiter will have his dashboard. Which will have different types of graphs and charts for the statistics that needs to be displayed.
Graphs Criteria

1) Retrieve data from database and show first 10 Job posting with its applications/month (Bar, Pie or any kind of graph)
2) City wise application/month (Bar, Pie or any kind of graph) for a Job Posting.
3) Top 5 job posting with less number of applications.

Add below graphs and charts for the statistics for admin in the Dashboard. **Data will be fetched from logging.**
Graphs Criteria

1) Graph for clicks per Job Posting (Bar, Pie or any kind of graph)
2) Graph for number of saved jobs.
3) Trace diagram for tracking one user or a group of users (Users from San Diego) (After clicking on job details how many users are completing the form, how many are leaving it half filled, how many are just reading the application and not applying to it.)

Applicant's Graph Criteria

1) Graph for his profile views/day (Just a count. Graph should have 30 days limit.)

The client should have a pleasing look and be simple to understand. Error conditions and feedback to the user should be displayed in a status area on the user interface.
Ideally, you will use an IDE tool to create your GUI.

## Tier 2 — Middleware

You will need to provide/design/enhance a middleware that can accomplish the above requirements. You have to implement it using REST based Web Services as Middleware technology. Next, decide on the Interface your service would be exposing. You should ensure that you appropriately handle error conditions/exceptions/failure states and return meaningful information to your caller. Make sure you handle the specific failure cases described below.
ON OR BEFORE the date listed below, you must turn in a document describing the interface your server will use which precisely means that you have to give API request-response descriptions.

Your project should also include a model class that uses standard modules to access your database. Your entity objects will use the data object to select/insert/update the data in the relational database.

User Kafka as a messaging platform for communication between front-end channels with backend systems.

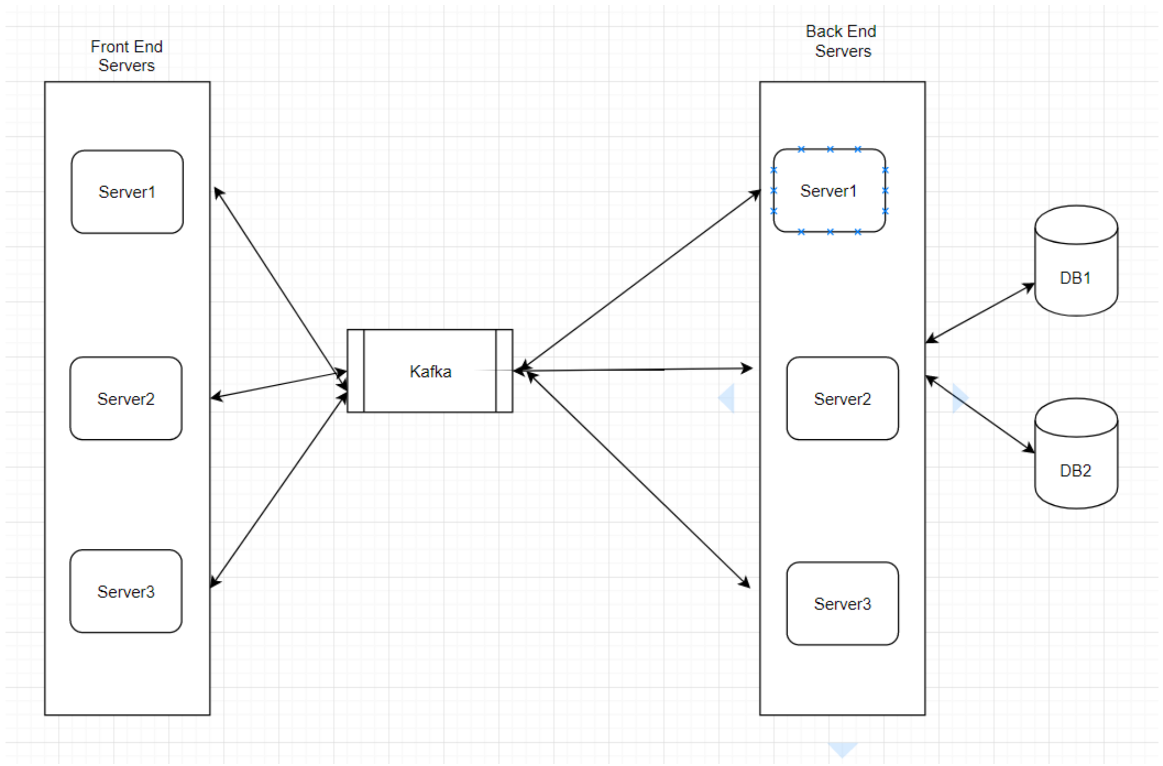
## Tier 3 — Database Schema and Database Creation

You will need to design a database table that will store your relational data. Choose column names and types that are appropriate for the type of data you are using. You will also need a simple program that creates your database and its table. The MySQL Workbench is a very efficient and easy tool for it.

You will need to decide which data are stored in MongoDB and MySQL. (Hint: think about pros and cons of MongoDB vs MySQL)

## Distributed System Deployment

Follow the distributed services architecture shown below.



## Scalability, Performance and Reliability

The hallmark of any good object container is scalability. A highly scalable system will not experience degraded performance or excessive resource consumption when managing large numbers of objects, nor when processing large numbers of simultaneous requests. You need to make sure that your system can handle many listings, users and incoming requests.

Pay careful attention to how you manage "expensive" resources like database connections.

Your system should easily be able to manage 10,000 movies, 10,000 Users and 100,000 Reservation records. Consider these numbers as **minimum** requirements.

Further, for all operations defined above, you need to ensure that if a particular operation fails, your system is left in a consistent state. Consider this requirement carefully as you design your project as some operations may involve multiple database operations. You may need to make use of transactions to model these operations and roll back the database in case of failure.

Performance Charts that shows difference in performance by adding different distributed features at a time
Chart 1: Single Chart of No Performance Optimization, Connection Pooling, Kafka, SQL Caching, Other optimization technique discussed in the class.
Chart 2: Performance Comparison of services with below deployment configurations

1. 1 UI Web Server 1 DB instance 1 Services server
2. 1 UI Web Server 1 DB instance 3 Services Servers
3. 2 UI Web Server 2 DB instance 4 Services Servers

**PowerPoint Presentation**

1. Names and group number
2. Performance graph to show different features combination such as B (Base), S (SQL Caching), D (DB connection pooling), K (Kafka)
   a) B b)B+D c) B+D+S d) B+D+S+K e) B+D+S+K+ other techniques
   Populate DB with at least 1000 users and 10,000 connections before measure performance.
3. Performance graph for distributed systems
4. **Extra Credit:** Graph database use case; explain what the schema is and how to use GD to represent connections, etc

## Testing

To test the robustness of your system, you need to design a test harness that will exercise all the functions that a regular client would use. This test harness is typically a command-line program. You can use your test harness to evaluate scalability (described above) by having the test harness create thousands of listings and users. Use your test harness to debug problems in the server implementation before writing your GUI.

## Other Project Details

Turn in the following on or before the due date. No late submissions will be accepted!
· A title page listing the members of your group
· A page listing how each member of the group contributed to the project (keep this short, one paragraph per group member is sufficient)
· A short (5 pages max) write-up describing:
  a) Your object management policy
  b) How you handle "heavyweight" resources
  c) The policy you used to decide when to write data into the database
· A screen capture of your client application, showing some actual data
· A code listing of your client application
· A code listing of your server implementations for the entity objects
· A code listing of your server implementation for the session object
· A code listing of your main server code
· A code listing of your database access class
· A code listing of your test class
· Any other code listing (utility classes, etc)
· Output from your test class (if applicable)
· A code listing of your database creation class (or script)
· A screen capture showing your database schema
· Observations and lessons learned (1 page max)

The possible project points are broken down as follows:
*(One of grading criteria is subjective and relative strength and weakness of your project compared to other projects)*
**Total Score – 105 points**

• **40pts for basic operation** – Your server implementation will be tested for proper operation of some aspect of your project. Each passed test is worth some point(s) toward the total score, depending on the test. Each failed test receives 0 points.

• **15 pts for scalability and robustness** – Your project will be judged on its scalability and robustness. I will test your program with thousands of objects; it should not exhibit sluggish performance, and it definitely should not crash. In addition to performance improvement techniques cover in the class, you are required to implement **SQL caching** using Radis and show performance analysis.
• **15 pts for distributed services** – Divide client requirements into distributed services and deploy them to AWS/Heroku etc. and Deploy the database to AWS/MLab etc. Each service will be running on backend connected by Kafka and divide your data into MongoDB and MySQL and provide performance data with justification on results.
• **15 pts for Analysis report, web/user/item tracking** – Devise your own web pages/user/item tracking using logs and explain why it is effective to analyze a web site and user behavior.
• **10 pts for the client** – You will get more points if your GUI is similar to LinkedIn site.
• **10 pts** for your test class and project write-up

## Hints:
· Maintain a pool of DB connections – Remember that opening a database connection is a resource-intensive task. It would be advisable to pre-connect several database connections and make use of them throughout your database access class so you don't continually connect/disconnect when accessing your database.

· Cache entity lookups – To prevent a costly trip to the database, you can cache the state of entity objects in a local in-memory cache. If a request is made to retrieve the state of an object whose state is in the cache, you can reconstitute the object without checking the database. Be careful that you invalidate the contents of the cache when an object's state is changed. In particular, you are required to implement **SQL caching using Radis**.

· Don't write data to the database that didn't change since it was last read.

· Focus FIRST on implementing a complete project – remember that a complete implementation that passes all of the tests is worth as much as the performance and scalability part of the project.

· Do not over-optimize. Project groups in the past that have tried to implement complex Optimizations usually failed to complete their implementations.

## Exceptions/Failure Modes

Your project MUST properly handle the following failure conditions
- Creating Duplicate User
- Addresses (for Person) that are not formed properly (see below)

For more failure conditions see Other Notes below

## Other Notes

*State abbreviation parameters*

State abbreviations should be limited to valid US state abbreviations or full state names. A Google search will yield these valid values. Methods accepting state abbreviations as parameters are required to raise the 'malformed_state' exception (or equivalent) if the parameter does not match one of the accepted parameters. You do not need to handle US territories such as the Virgin Islands, Puerto Rico, Guam, etc.

*Zip code parameters*

Zip codes should adhere to the following pattern:
[0-9][0-9][0-9][0-9][0-9]
or
[0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]
Examples of valid Zip codes:
95123
95192
10293
90086-1929
Examples of invalid Zip codes:
1247
1829A
37849-392
2374-2384

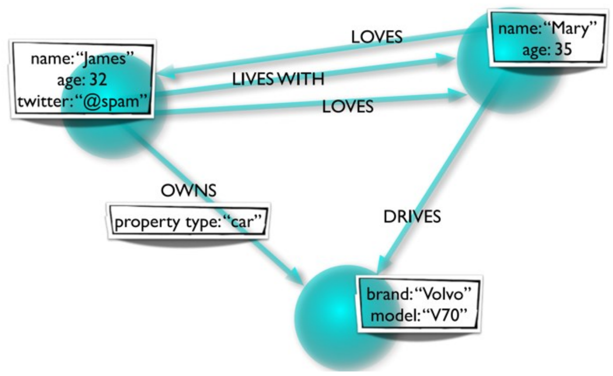
## Peer Review

Each team member should write review about other team members except himself/herself. It is confidential which means that you should not share your comments with other members. Peer Review is submitted separately on Canvas.

## Extra Credit (Graph DB Implementation, 10 points)

There is no partial credit for Extra credit. You will get either full marks or none based on your mobile application quality

In linkedin similar to other social networks, everyone is connected based on relationships/friendships. First, you need to store these relationships in an efficient way in a graph database. Each node can represent person and relationship are formed using friend requests based on properties such as schools, age, geographic locations, industry, etc.



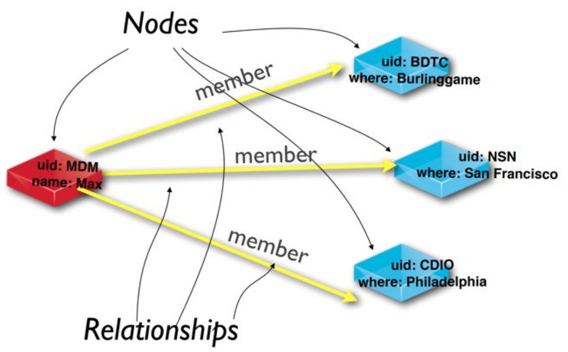
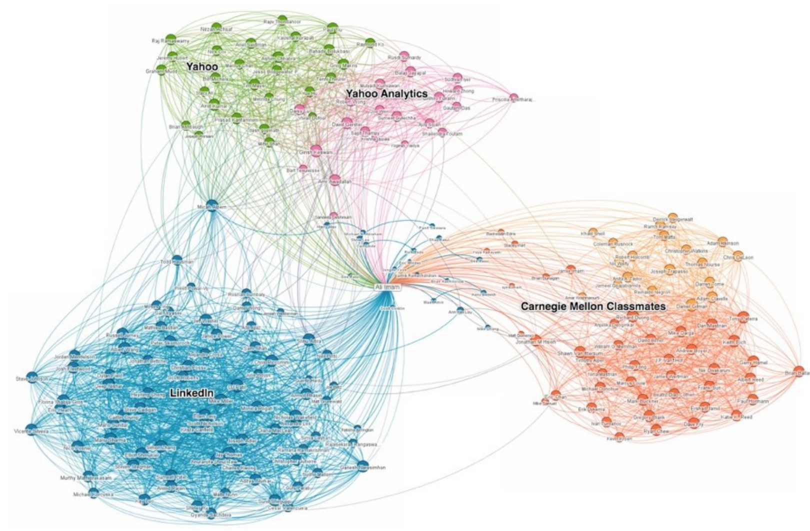


Figure 1: Graph Data Model, source: Neo4j     Figure 2: Property Graph, source: Neo4j



<Figure 3: Visualize Linked network, source: linkedin>

**What does Linkedin use for their graph database?**
LinkedIn has a large team working on a proprietary graph database, which sits at the center of nearly every operation at LinkedIn.

**How about other companies?**
Twitter depends on a graph database, and has released FlockDB, a graph database it created, as open source.
Neo Technology, the creator of Neo4j, the most popular graph database, now has more than 30 Global 2000 companies adopt its technology, including enterprise brands like Wal-Mart, eBay, Lufthansa, and etc. https://neo4j.com/customers-old-2/ Airbnb case study at https://neo4j.com/users/airbnb/

**What else can we use graph database?**
Both Snap and Glassdoor report they have significantly improved the accuracy of their recommendations by using a graph to navigate their connected data. By finding and making better use of networks, many different types companies are breaking new ground with respect to intelligent real-time analytics.

https://neo4j.com/use-cases/
https://www.infoworld.com/article/3251829/nosql/why-you-should-use-a-graph-database.html
https://engineering.linkedin.com/teams/data#3

.