

Startup Investments Project



CS 3265: Database Management Systems

Sarah Zhang, Ethan Piper

Fall 2021

Introduction

Our project consisted of using the [Crunchbase 2013 Snapshot © 2013 Dataset](#) in the backend to create a website that displays data from the dataset.

About The Dataset

According to their website, Crunchbase is “the leading destination for company insights from early-stage startups to the Fortune 1000.” The dataset we used contains information about the startup ecosystem up to December 2013: organizations, individuals, company news, funding rounds, acquisitions, and IPOs.

The raw dataset consists of 11 tables:

1. Objects: *Main file containing base information.*
2. Funds: *Contains data on the venture capital funds that make investments.*
3. Funding_Rounds: *Contains information about startup funding rounds.*
4. Investments: *Contains data on the various different investments made by venture capitalists.*
5. IPOs: *Contains data on initial public offerings.*
6. Acquisitions: *Contains information about startups that have been bought.*
7. Milestones: *Contains significant events within the startup ecosystem.*
8. Offices: *Contains information about startup company offices.*
9. People: *Contains information about individuals in the startup world.*
10. Degrees: *Contains the education backgrounds of individuals involved in the startup world.*
11. Relationships: *Contains relationship data that links companies to individuals and their positions.*

We were interested in exploring this dataset to learn more about the process behind creating and acquiring startups, including what kind of people are involved in founding startups and how startups become successful.

About The Application

We developed a website to display and alter the data from the dataset. We were interested in developing a website because neither of us had much experience working in the front-end before this project. We were both interested in learning the new languages associated with web development, including Javascript, PHP, CSS, and HTML, as well as gaining experience in full stack development.

The main goal for the application was to display the data in interesting ways in order to learn more about the startup world. The different displays we wanted to implement were a map with points for all the companies represented in the database, a pie chart displaying different categories of data for a single attribute, and a line graph displaying change over time for a single attribute.

The map was out of the scope of this project, but we implemented the pie chart, the line graph, and an additional view that displays rows of a table. We were able to create these displays using relationships in the database that may not be overly obvious. For example, we used the relationships between the academic degrees earned by the people in the database and the companies they work for to display the distribution of academic degrees that a specific company has. These seemingly simple relationships allowed for displays of interesting and useful data on the various companies.

We also wanted to be able to insert our own information into the dataset. As college students, we hope to get involved in the startup and technological world when we graduate. We implemented a functionality that can add ourselves or other people to the database and link ourselves with a specific company.

In order to make sure the database can stay up to date, we also wanted to implement a way to show that a company was bought or acquired by another. This happens often in the startup space, and therefore implementing an easy way to update all of the necessary fields seemed like the perfect job for our web application.

Final Implementation

The final implementation of the project is a website that can display, insert, and delete data from the database. The three functionalities of the website are the inserting and deleting of people in the startup space, the updating of companies by simulating an acquisition, and displaying data on specific companies or types of companies.

Final Database Design

The final database design consists of 20 tables: 10 of the original tables in the raw dataset, plus 10 normalized tables derived from the objects table. The objects table is the main table, and its “id” attribute is also in the other tables under different names. The “id” attribute in the objects table provides a connection to all the other tables in the database.

- **object_id** is also in funding_rounds, ipos, milestones, offices, people, and degrees
- The investments table includes two different “id”s per row: **funded_object_id** and **investor_object_id**
- The acquisitions table includes two different “id”s per row: **acquiring_object_id** and **acquired_object_id**
- The relationships table includes two different “id”s per row: **person_object_id** and **relationship_object_id**

Normalization Process and Business Rules

Although the raw dataset was already divided into 11 tables, the largest table (objects) was not normalized. All the other tables were already normalized, so the only necessary changes were adding constraints, and parsing the DATE or DATETIME attributes that were imported into the mega-tables as CHAR data types.

All the raw tables contained DATETIME columns “created_at” for when the row was created, and “updated_at” for when the row was updated. I added a constraint to updated_at to check if updated_at comes after created_at. All other constraints are either attribute-specific (such as UNIQUE, NOT NULL, etc), or they are keys.

Objects Table

Functional Dependencies

1. $id \rightarrow object_id, entity_type, entity_id, parent_id, name, normalized_name, permalink, category_code, status, domain, twitter_username, logo_url, overview, short_description, description, tag_list, first_investment_at, first_funding_at, first_milestone_at, relationships, created_by, created_at, updated_at$
2. $id, first_investment_at \rightarrow last_investment_at, investment_rounds, invested_companies$
3. $id, first_funding_at \rightarrow last_funding_at, funding_rounds, funding_total_usd$
4. $id, first_milestone_at \rightarrow last_milestone_at, milestones$
5. $id, status \rightarrow founded_at, closed_at$
6. $domain \rightarrow homepage_url$
7. $logo_url \rightarrow logo_width, logo_height$
8. $overview \rightarrow short_description, description, tag_list$
9. $id \rightarrow country_code, city, state_code, region$

Normalized Tables

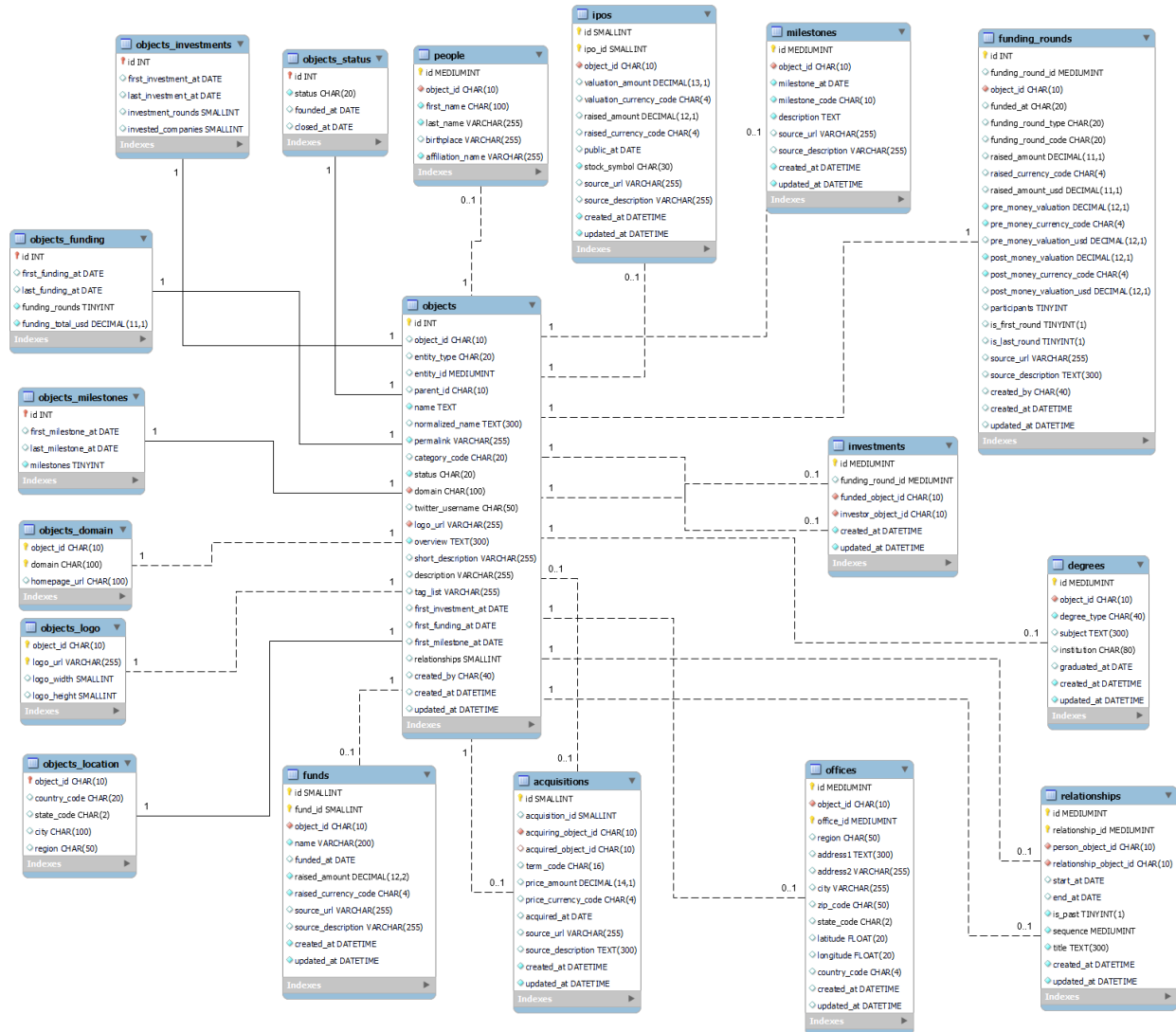
- **objects_info**(id, object_id, name, entity_type, entity_id, parent_id, category_code, status, domain, twitter_username, logo_url, overview, relationships, created_by, created_at, updated_at, first_investment_at, first_funding_at, first_milestone_at)
- **objects_investments**(id, first_investment_at, last_investment_at, investment_rounds, invested_companies)
- **objects_funding**(id, first_funding_at, last_funding_at, funding_rounds, funding_total_usd)
- **objects_milestones**(id, first_milestone_at, last_milestone_at, milestones)
- **objects_status**(id, status, founded_at, closed_at)
- **objects_domain**(domain, homepage_url)
- **objects_logo**(logo_url, logo_width, logo_height)
- **objects_location**(id, country_code, city, state_code, region)

Other Tables

- **funds_info**(id, fund_id, object_id, name, funded_at, raised_amount, raised_currency_code, source_url, source_description, created_at, updated_at)
- **funding_rounds_info**(id, object_id, funded_at, funding_round_type, funding_round_code, raised_amount, raised_currency_code, raised_amount_usd, pre_money_valuation, pre_money_currency_code, pre_money_valuation_usd, post_money_valuation, post_money_currency_code, post_money_valuation_usd, participants, is_first_round, is_last_round, source_url, source_description, created_by, created_at, updated_at)
- **investments_info**(id, funding_round_id, funded_object_id, investor_object_id, created_at, updated_at)
- **ipos_info**(id, ipo_id, object_id, valuation_amount, valuation_currency_code, raised_amount, raised_currency_code, public_at, stock_symbol, source_url, source_description, created_at, updated_at)
- **acquisitions_info**(id, acquisition_id, acquiring_object_id, acquired_object_id, term_code, price_amount, price_currency_code, acquired_at, source_url, source_description, created_at, updated_at)
- **milestones_info**(id, object_id, milestone_at, milestone_code, description, source_url, source_description, created_at, updated_at)
- **offices_info**(id, object_id, office_id, description, region, address1, address2, city, zip_code, state_code, country_code, latitude, longitude, created_at, updated_at)

- **people_info**(id, object_id, first_name, last_name, birthplace, affiliation_name)
- **degrees_info**(id, object_id, degree_type, subject, institution, graduated_at, created_at, updated_at)
- **relationships_info**(id, relationship_id, person_object_id, relationship_object_id, start_at, end_at, is_past, sequence, title, created_at, updated_at)

UML Diagram



Testing

Most of the data that we acquired for testing was not hard to find. For the most part, we could make up data or use ourselves for inserting, searching, or deleting people in the database. If we needed to test the functionality with actual people in the database, many well known and wealthy people are already in the tables, such as Mark Zuckerberg.

All that was needed to test creating an acquisition and displaying the degree distribution of a company were the names of companies in the tech world. This is fairly simple to find, because most of them we already know, such as google, amazon, or facebook.

The last pieces of data we needed to test our application were categories of companies. This was a little harder to find because some of the categories that the database specified are not intuitive. To find this data, we simply queried the database for unique values of the “category_code” in the objects_info table.

All of the functionalities were tested multiple times with generally faulty data to make sure the correct error messages were shown, as well as with various inputs that succeeded. But, because our inputs are on the simpler side, testing was not arduous.

Implemented Use Cases

The three general groups of functionality we implemented in the database were the inserting and deleting of people in the startup space, the updating of companies by simulating an acquisition, and displaying data on specific companies or types of companies. This would not only allow us to use our application to gain insight on the startup space, but also to easily update it to keep its information accurate.

The inserting and deleting of people in the startup space is fairly self explanatory. The application allows people to be inserted into the database and links them to a specific company. This basically means that that person was newly hired by the company. In addition, you are optionally able to specify a job title for the person.

For simulating an acquisition, the user simply needs to provide the names of the two companies and that amount of money in USD that one paid for the other. This, however, is deceptively simple because the web-app must then link those two companies across the entire database as well as update several values in different tables. This does insert some new data, however most of its functionality is centered around updating different tables.

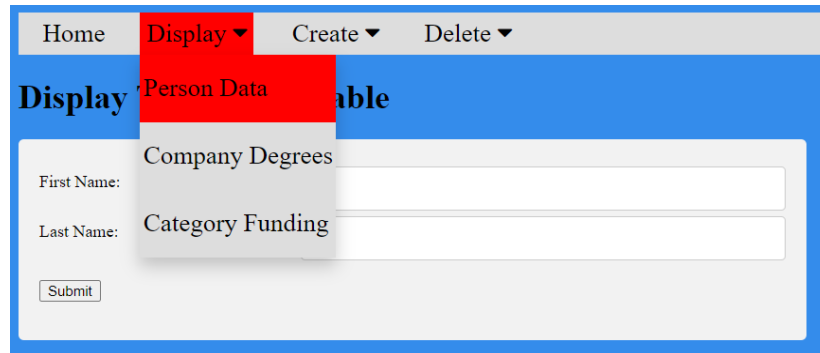
There were two ways that we created to display company data. One way was to simply provide the name of a company and receive a chart outlining distribution of the education level of people employed in that company. This is done by using the academic degree data in the database

and grouping employees that would work for the same company and have the same highest level of education. We also were able to display data on the history of funding of various types of companies. In the database, each company has a category code that gives a simple explanation of the company's purpose. Using this data, the amount of funding received, and year funded, we were able to display a graph showing the aggregate amount of money each category has received over the past twenty years. This also allowed us to find general trends in funding over the past twenty years. For example, the funding of almost every category dropped significantly around the 2008 market crash.

Functionality

Inserting, Deleting, and Displaying People in the Database

This is probably the simplest functionality in the database, as you can basically create your own data. First, to display or search for people in the database, go to the Person Data heading under “Display”. See the screenshot below:

The screenshot shows a web application interface with a blue header bar. In the header, there are four menu items: 'Home', 'Display', 'Create', and 'Delete', each with a dropdown arrow. The 'Display' menu is currently open, showing a red background with the text 'Person Data' and 'able'. Below the header, there is a search form with two input fields: 'First Name:' and 'Last Name:'. The 'First Name' field contains the text 'Company Degrees' and the 'Last Name' field contains the text 'Category Funding'. There is a 'Submit' button at the bottom of the form.

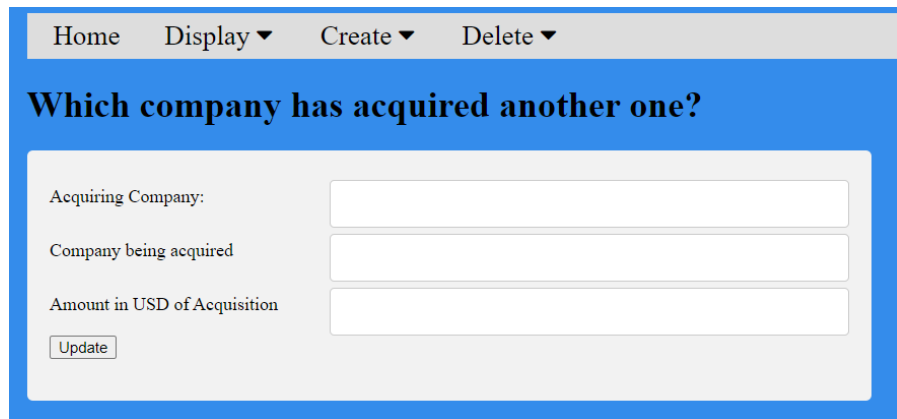
From there, you can enter anyone you know from the startup world, such as Mark Zuckerberg or Steve Jobs. The inputs are not case sensitive. Then, it will display some information on that person you searched for.

To insert a person into the database, mouse over “Create” and select “Person”. From there, you can specify a first name, last name, and a company they work for with options to specify a birthplace and a job title.

Finally, to delete a person you created or someone in the database, mouse over “Delete” and select “Person”. You can then specify a first name and last name with an optionally field for a company. This will only work, however, if there is only one person with that name. If there are multiple people in the database with the same name, you need to specify a company to make sure you delete the right person. The inputs “Mark”, “Zuckerberg”, and “Facebook” work well here.

Simulating an Acquisition

To simulate that one company acquired another one, you must mouse over “Create” and select “Acquisition”. You will then see the page that looks like the figure below.

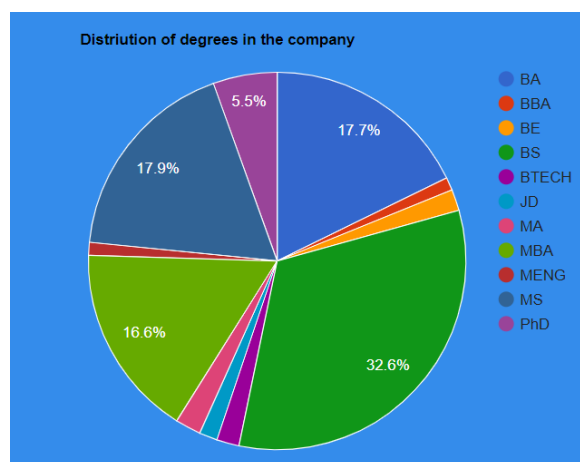


The screenshot shows a web application with a navigation bar at the top containing links: Home, Display ▼, Create ▼, and Delete ▼. Below the navigation bar is a blue header with the text "Which company has acquired another one?". The main content area is a light gray box containing three input fields: "Acquiring Company:", "Company being acquired", and "Amount in USD of Acquisition". Below these fields is an "Update" button.

Now, you must simply specify the name of 2 companies, the one being acquired and the one doing the acquiring, and the amount of money in USD of the acquisition. One example I always use is Google acquiring Amazon for 10 trillion dollars. Most of the changes made will not be visible from the web-app, but many of the tables will be updated.

Displaying data on Companies and categories

There are two ways in which we implemented visualization of data in our database. The first is the distribution of acetic degrees in a company. Mouse over “Display” and select “Company Degrees” to enter the correct webpage. From there you can write the name of any company, although we would suggest larger companies, such as Google. After you click “submit” it will take a while, but you will eventually be shown a pie chart of the distribution of degrees in a company. See for below for Google’s degree distribution:



Finally, to display the funding history of certain categories of companies, mouse over “Display” and select “Category Funding”. Examples of different categories are “news”, “web”, or “games video”. After clicking submit, a line graph will appear showing the funding history for that category of company since around 1990.

Summary

We accomplished most of the goals in this project that we aimed to do. The only functionality we did not have time to implement was the world map displaying points for all the company locations, because maps are relatively complicated to create in a website. However, we were able to finish the views for the pie chart and the line graph; these display the data in a nicer, more user-friendly format compared to a slice of the table rows. Additionally, we were unable to experiment too much with CSS to make the website look fancy, but we did update some minor details in the site such as the displays for entering information into the forms.

On the database side of the project, one challenge was having to clean the data because MySQL wouldn’t read all the lines correctly in the objects.csv file since it got confused by backslashes. We wrote a Python script to preprocess this file to remove the backslashes, and then MySQL was able to load the data in correctly. Another challenge was normalizing and creating a database design for the dataset, since there were so many tables and attributes. Creating the UML diagram helped us to understand the structure of the database better.

For the front end part of the project, most of the challenges were centered around our lack of experience with developing web pages. It was very difficult to learn html, css, php, and javascript for the first time while completing this project. However, this was overcome by keeping our front end simple and using internet resources to help implement some of the more complicated pages.

Regarding the division of responsibilities, Sarah mostly worked on the database side of the project, and Ethan mostly worked on the front-end. More specifically, Sarah created the mega tables, wrote the load data statements, wrote the preprocessing Python script, created the normalized tables and constraints, and wrote the insert statements into the normalized tables. Sarah also wrote the README and the database-related sections of the report document. Ethan created all the website functionality, including the UI and the stored procedures and views in SQL, as well as all the website data displays like the pie chart and the line graph. Ethan also created the UML diagram, recorded the demo video, and wrote the front-end related sections of the report document. The division of one person working on the front-end and one person working on the back-end is relatively fair. Although Ethan had a greater learning curve because neither of us had much experience working in the front-end, Sarah also had a large volume of tables and attributes to work with.