SeeQL

Increasing SQL Explorability with In Situ Dataset Visualization

INFO-247: Information Visualization Final Project Report

Soravis Prakkamakul
<soravis@berkeley.edu>

OVERVIEW

SQL Querying remains one of the most popular ways to do exploratory data analysis. However, comparing to other methods, it does not help users to quickly understand the underlying dataset characteristics such as distribution, null values, effects of filtering conditions. Usually, some effort has to be spent going back and forth reformulating the query to get the right results. I propose an approach of real-time query augmentation with embedded mini visualizations to help users understand more about the dataset. I also propose a design along with a prototype query interface demonstrating two key features: column value auto suggestion and table join visualization. The usability testing conducted on users with some SQL experience suggests that such tool would be useful for exploring and validating the dataset. The result also raised some usability issues to be addressed in future research.

PROJECT GOAL

To design a realtime SQL query augmentation method that achieve the following

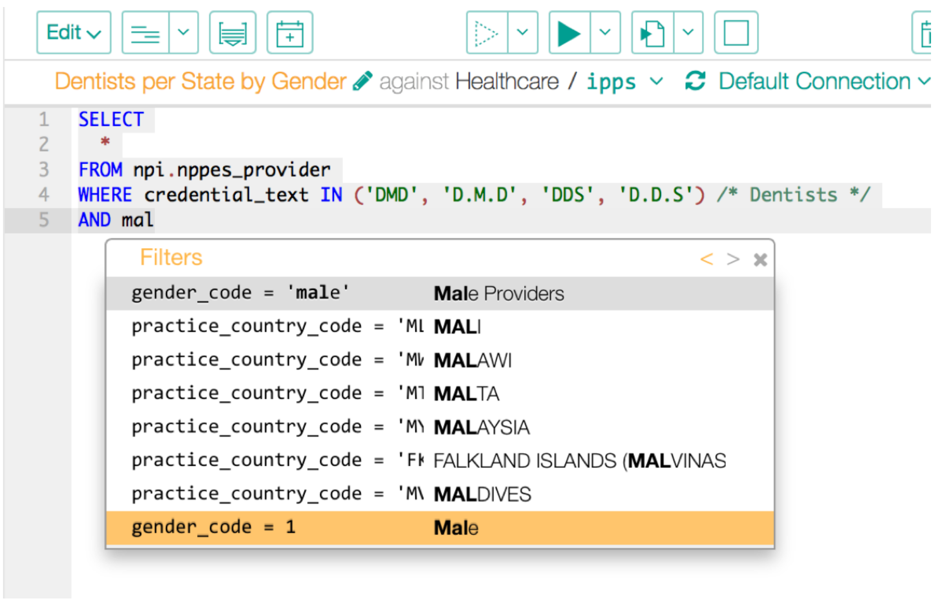
1. Help users understand more about the characteristics of datasets.
2. Help users foresee the effects of conditional statements in queries.
3. Help users reduce the effort spent in trial and error in the process of constructing complex SQL queries.

LITERATURE REVIEW

Usability issues of database systems has been explored in some past works. Jagadish et at. summarized a number of database "pains" [1] which include the difficulty of constructing join queries from normalized database, the pain of having too many options that does not lead to satisfying results (e.g. too many ways to perform the same query, the use of query optimizations, and query plans), and the problem of not being able to see the results before submitting the query which causes the user to go back and forth between revising and resubmitting it. The work also brought up an assumption that, "a query is being reformulated because the user is 'exploring' the data" and proposed that existing querying methods require users to predict the right parameters.

There were several approaches for addressing the usability issues of database. A number of works took the direct manipulation approach where users interact with the data displaying interface directly instead of through a querying interface. Etable [2] explores query building by allowing users to interact with a spreadsheet interface in order to construct a query. VISAGE [3], allows a visual alternative interface for the Cypher graph query language by showing graph visualizations of the nodes in the database and that allow users to interact with the nodes.

While direct manipulation approaches can be more intuitive for many users, they often require specific technical infrastructure implementations on top of the original datastore. Meanwhile, SQL remains one of the most common languages [5] and its interoperability allows usage of the query language even on non-relational database systems, for example, through Apache Drill which provides SQL interface on top of NoSQL databases. Therefore, industry efforts have been put to improve the usability of SQL. For example, Alation [6] provides query condition auto completion driven by collaborative data dictionary.

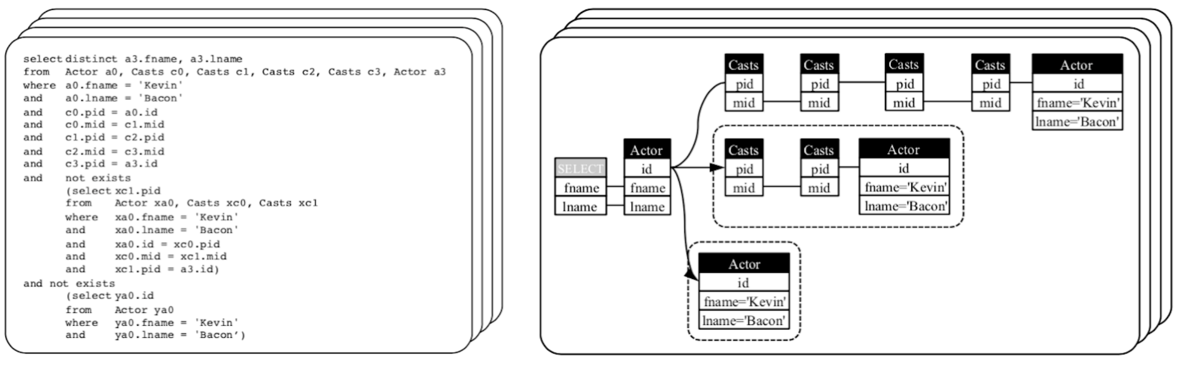


```
SELECT
  *
FROM npi.nppes_provider
WHERE credential_text IN ('DMD', 'D.M.D', 'DDS', 'D.D.S') /* Dentists */
AND mal
```

```
gender_code = 'male'            Male Providers
practice_country_code = 'ML     MALI
practice_country_code = 'MW     MALAWI
practice_country_code = 'MT     MALTA
practice_country_code = 'MY     MALAYSIA
practice_country_code = 'FK     FALKLAND ISLANDS (MALVINAS
practice_country_code = 'MV     MALDIVES
gender_code = 1                 Male
```

Firgure 1, The query autocompletion in Alation

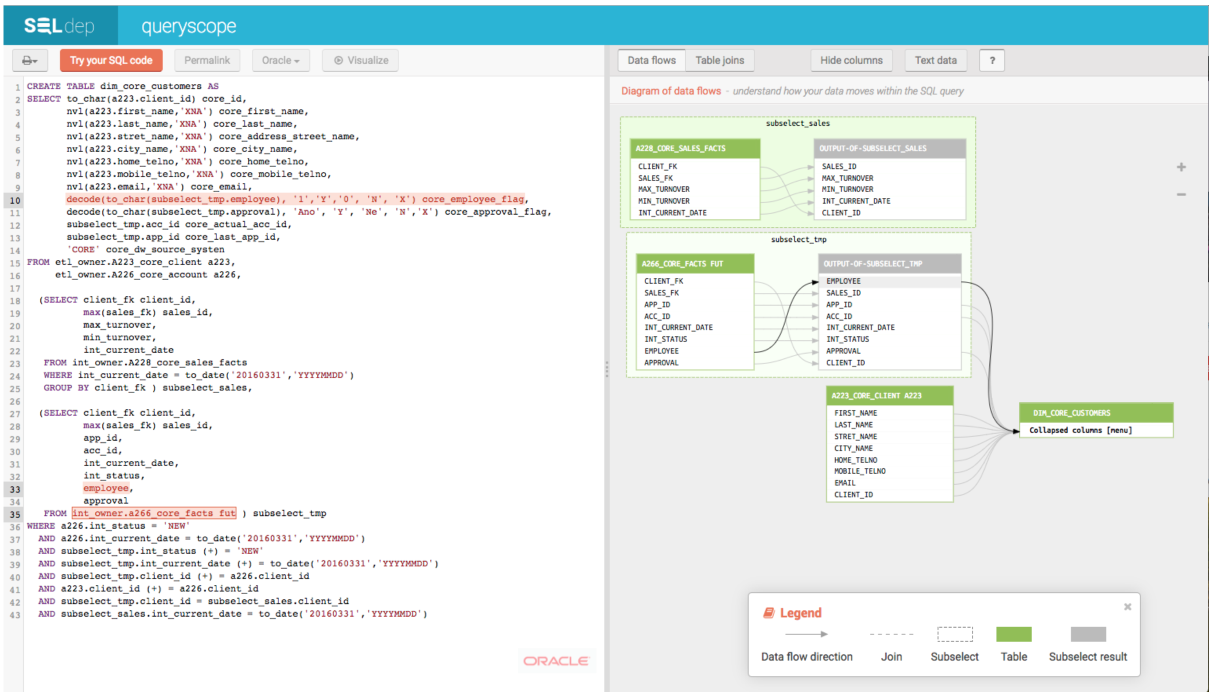
Another group of works augment SQL querying by visualizing it in a more understandable way. QueryViz [4] presented a tool for SQL query visualization that "reduces the time needed to read and understand existing queries". This is illustrated in Figure 2 in which, given a query, the system generates a tree diagram of tables that shows how the result of the query is formed. A similar effort from the industry includes SQLDep [6] which took the concept further with interactivity. Figure 3 illustrates the highlighting of the relevant parts of the query when an element in the visualization is focused.



```
select distinct a3.fname, a3.lname
from   Actor a0, Casts c0, Casts c1, Casts c2, Casts c3, Actor a3
where  a0.fname = 'Kevin'
and    a0.lname = 'Bacon'
and    c0.pid = a0.id
and    c0.mid = c1.mid
and    c1.pid = c2.pid
and    c2.mid = c3.mid
and    c3.pid = a3.id
and    not exists
       (select xc1.pid
       from    Actor xa0, Casts xc0, Casts xc1
       where   xa0.fname = 'Kevin'
       and     xa0.lname = 'Bacon'
       and     xa0.id = xc0.pid
       and     xc0.mid = xc1.mid
       and     xc1.pid = a3.id)
and not exists
       (select ya0.id
       from    Actor ya0
       where   ya0.fname = 'Kevin'
       and     ya0.lname = 'Bacon')
```

(a) SQL query (b) QueryViz representation

Figure 2, QueryViz



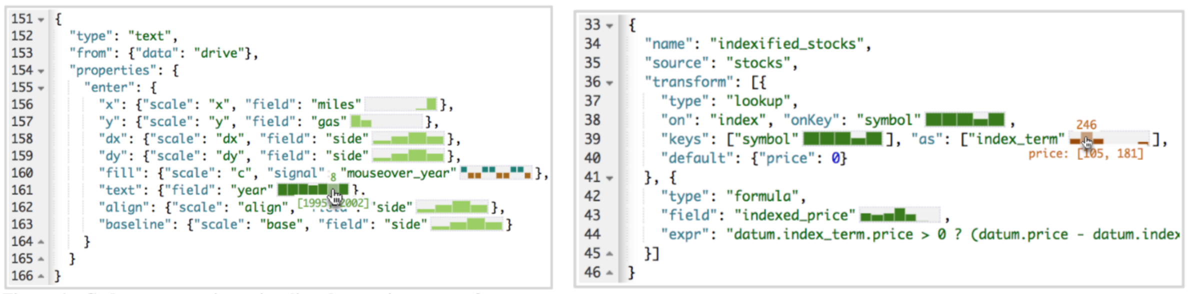
```
CREATE TABLE dim_core_customers AS
SELECT to_char(a223.client_id) core_id,
       nvl(a223.first_name,'XNA') core_first_name,
       nvl(a223.last_name,'XNA') core_last_name,
       nvl(a223.stret_name,'XNA') core_address_street_name,
       nvl(a223.city_name,'XNA') core_city_name,
       nvl(a223.home_telno,'XNA') core_home_telno,
       nvl(a223.mobile_telno,'XNA') core_mobile_telno,
       nvl(a223.email,'XNA') core_email,
       decode(to_char(subselect_tmp.employee), '1','Y','0', 'N', 'X') core_employee_flag,
       decode(to_char(subselect_tmp.approval), 'Ano', 'Y', 'Ne', 'N','X') core_approval_flag,
       subselect_tmp.acc_id core_actual_acc_id,
       subselect_tmp.app_id core_last_app_id,
       'CORE' core_dw_source_systen
FROM etl_owner.A223_core_client a223,
     etl_owner.A226_core_account a226,

  (SELECT client_fk client_id,
          max(sales_fk) sales_id,
          max_turnover,
          min_turnover,
          int_current_date
   FROM int_owner.A228_core_sales_facts
   WHERE int_current_date = to_date('20160331','YYYYMMDD')
   GROUP BY client_fk ) subselect_sales,

  (SELECT client_fk client_id,
          max(sales_fk) sales_id,
          app_id,
          acc_id,
          int_current_date,
          int_status,
          employee,
          approval
   FROM int_owner.a266_core_facts fut ) subselect_tmp
WHERE a226.int_status = 'NEW'
  AND a226.int_current_date = to_date('20160331','YYYYMMDD')
  AND subselect_tmp.int_status (+) = 'NEW'
  AND subselect_tmp.int_current_date (+) = to_date('20160331','YYYYMMDD')
  AND subselect_tmp.client_id (+) = a226.client_id
  AND a223.client_id (+) = a226.client_id
  AND subselect_tmp.client_id = subselect_sales.client_id
  AND subselect_sales.int_current_date = to_date('20160331','YYYYMMDD')
```

Figure 3, the interactivity in SQLDep

While the mentioned query visualizations are useful for understanding the structure of the query, it does not give the user the glimpse into the characteristics of the dataset, namely, how data is distributed or the number of null values as well as the predicted runtime behavior of that query that could prevent user error. That bring us to the method of dataset-driven code augmentation. A work by Jane Hoffswell et. al. [8] introduced a method of augmenting code with small visualizations of the dataset to aid program understanding (Illustrated in Figure 4). The method was found to help users to better understand the relationship between the code and the runtime behavior of the program.

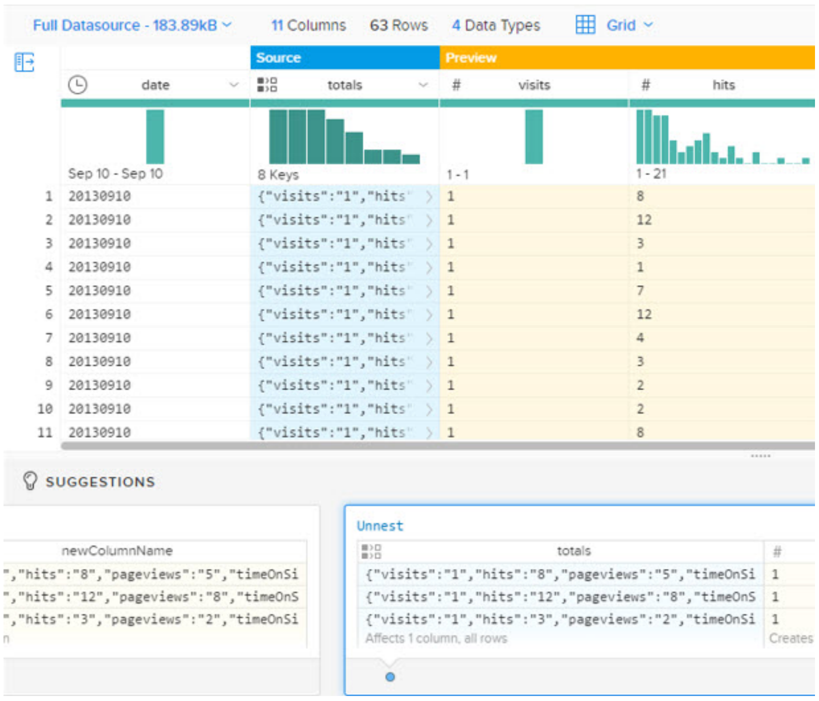


```
{
  "type": "text",
  "from": {"data": "drive"},
  "properties": {
    "enter": {
      "x": {"scale": "x", "field": "miles" },
      "y": {"scale": "y", "field": "gas" },
      "dx": {"scale": "dx", "field": "side" },
      "dy": {"scale": "dy", "field": "side" },
      "fill": {"scale": "c", "signal" "mouseover_year" },
      "text": {"field": "year" }.
      "align": {"scale": "align", "field": 'side" },
      "baseline": {"scale": "base", "field": "side" }
    }
  }
}
```

```
{
  "name": "indexified_stocks",
  "source": "stocks",
  "transform": [{
    "type": "lookup",
    "on": "index", "onKey": "symbol" ,
    "keys": ["symbol" ], "as": ["index_term" ],
    "default": {"price": 0}
  }, {
    "type": "formula",
    "field": "indexed_price" ,
    "expr": "datum.index_term.price > 0 ? (datum.price - datum.index
  }]
}
```

Figure 4, Code augmented in a Vega specification by Hoffswell et al.

Seeing data while you work has been proven to be useful in not only in querying but also data manipulation. A data transformation tool Trifacta Wrangler shows the use of embedded visualizations of the column's data at the head of each column (illustrated in Figure 5). With this approach, the user could quickly get a sense of the data distribution, the number of distinct values, and duplicate values within each column.



	date	totals	visits	hits
	Sep 10 - Sep 10	8 Keys	1 - 1	1 - 21
1	20130910	{"visits":"1","hits"	1	8
2	20130910	{"visits":"1","hits"	1	12
3	20130910	{"visits":"1","hits"	1	3
4	20130910	{"visits":"1","hits"	1	1
5	20130910	{"visits":"1","hits"	1	7
6	20130910	{"visits":"1","hits"	1	12
7	20130910	{"visits":"1","hits"	1	4
8	20130910	{"visits":"1","hits"	1	3
9	20130910	{"visits":"1","hits"	1	2
10	20130910	{"visits":"1","hits"	1	2
11	20130910	{"visits":"1","hits"	1	8

newColumnName
","hits":"8","pageviews":"5","timeOnSi
","hits":"12","pageviews":"8","timeOnS
","hits":"3","pageviews":"2","timeOnSi
n

totals	#
{"visits":"1","hits":"8","pageviews":"5","timeOnSi	1
{"visits":"1","hits":"12","pageviews":"8","timeOnS	1
{"visits":"1","hits":"3","pageviews":"2","timeOnSi	1
Affects 1 column, all rows	Creates

Figure 5, The use of column visualization Trifacta Wrangler

DESIGN

The design is done by constructing and iterating in a design space which consist of the information to display to the user, the method of visualization to use, and the format of the visualization (illustrated in Figure 6).

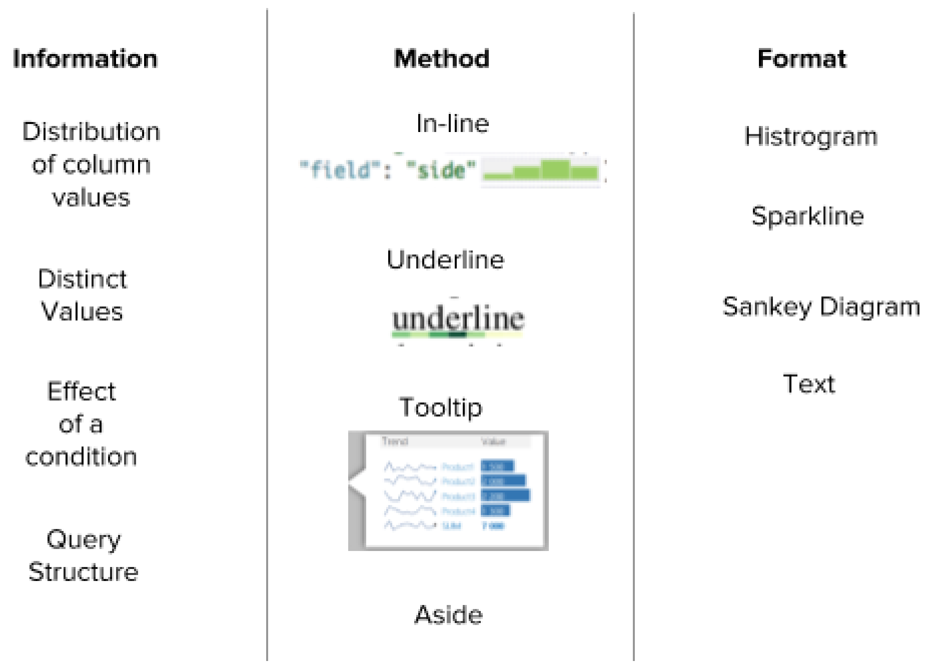


Figure 6, the design space

Components of the Design Space

The Information Space

I hypothesized that the following information would be the most useful to the users for learning about their dataset.

- **Distribution of column values**: For numeric columns, the number of rows in each value range. For categorical columns, the number of rows in each category
- **Distinct Values**: The outstanding values in the column, or, in case that there are few values, all the distinct values in that column.
- **Effect of a condition**: How much data would the condition filter out
- **Query Structure**: How does this join look like in terms of tables, keys, and how much of involved tables could be joined.

The Method Space

My methods are defined by the affordances available in code editing interfaces varying in the level of attention demand and saliency/

- **In-line:** Showing the visualization in-line with the next
- **Underline**: Show values encoded as bars under (or above) the text
- **Tooltip**: Show the visualization modally in a pop-up tooltip
- **Aside**: Put the visualization outside the code editing interface

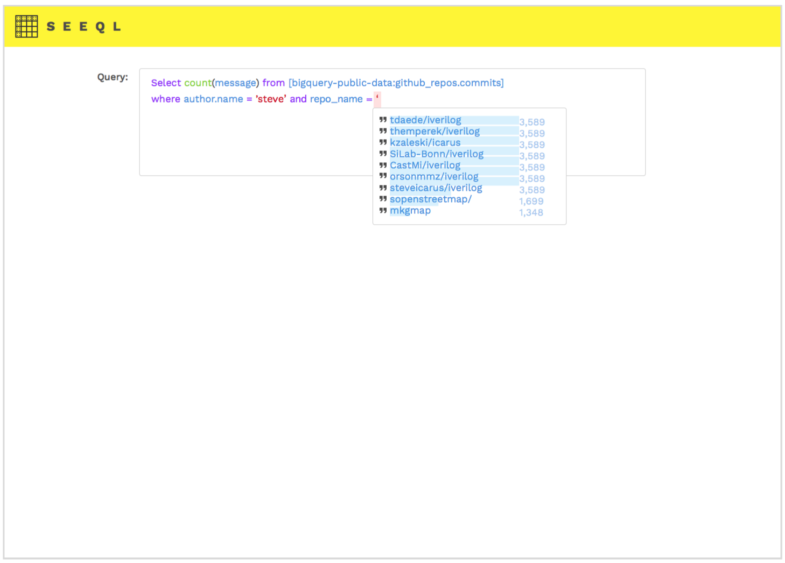
The Format Space

The information to be shown to the user can be displayed in different shapes and forms.

- **Histogram**: A candidate for showing the distribution of data
- **Sparkline**: A candidate for showing numeric trends over time
- **Sankey Diagram**: A candidate for showing join operations
- **Text**: A candidate for showing any information

Initial Design

Initial designs are done in Sketch for some combinations in the design space to get initial feedback and to be used a guide for prototype implementation. The design is basically based on a simple query typing interface (illustrated in Figure 7)



```
Select count(message) from [bigquery-public-data:github_repos.commits]
where author.name = 'steve' and repo_name = '
```

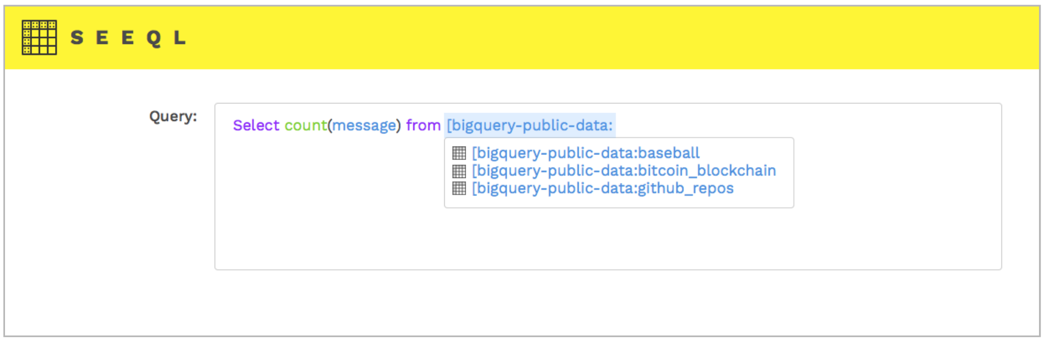
```
” tdaede/iverilog        3,589
” themperek/iverilog     3,589
” kzaleski/icarus        3,589
” SiLab-Bonn/iverilog    3,589
” CastMi/iverilog        3,589
” orsonmmz/iverilog      3,589
” steveicarus/iverilog   3,589
” sopenstreetmap/        1,699
” mkgmap                 1,348
```

Figure 7, Initial mockup

The initial mockup shows the following features

Identifier Auto Suggestions

The system should be able to suggest the names of the tables in the dataset along with their columns similar to other SQL auto suggestion systems.

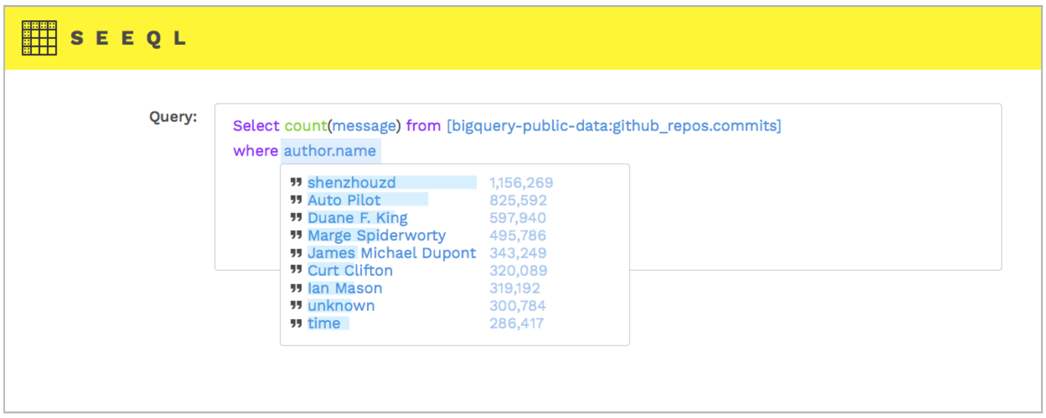


```
Select count(message) from [bigquery-public-data:
    [bigquery-public-data:baseball
    [bigquery-public-data:bitcoin_blockchain
    [bigquery-public-data:github_repos
```

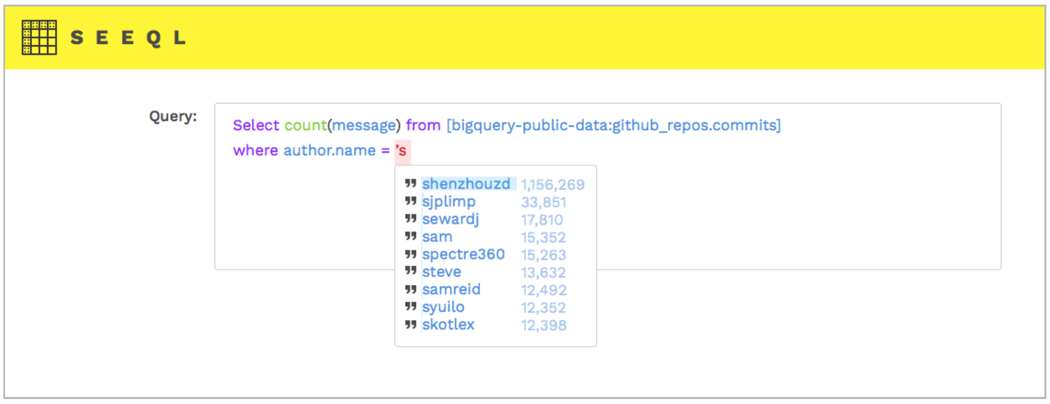
Figure 8, A mockup screen for identifier auto suggestions

Column Value Auto Suggestions

The system should be able to suggest possible values for that column as well as showing the rough distribution of the data set, e.g. how many rows have each value. The suggestion should also be able to update according to the partial input the user has given. For instance, if a letter ‘s’ is already typed, the suggested options should only include the values that begin with an ‘s’ (illustrated in Figure 9 (bottom))



```
Select count(message) from [bigquery-public-data:github_repos.commits]
where author.name
    shenzhouzd              1,156,269
    Auto Pilot                825,592
    Duane F. King             597,940
    Marge Spiderworty         495,786
    James Michael Dupont      343,249
    Curt Clifton              320,089
    Ian Mason                 319,192
    unknown                   300,784
    time                      286,417
```



SEEQL

Query:

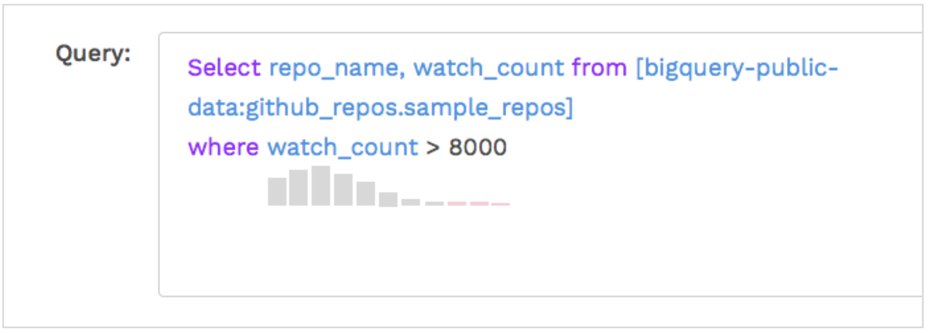
```
Select count(message) from [bigquery-public-data:github_repos.commits]
where author.name = 's
```

shenzhouzd	1,156,269
sjplimp	33,851
sewardj	17,810
sam	15,352
spectre360	15,263
steve	13,632
samreid	12,492
syuilo	12,352
skotlex	12,398

Figure 9, Column value auto suggestions with prefix (bottom) and without prefix (top)

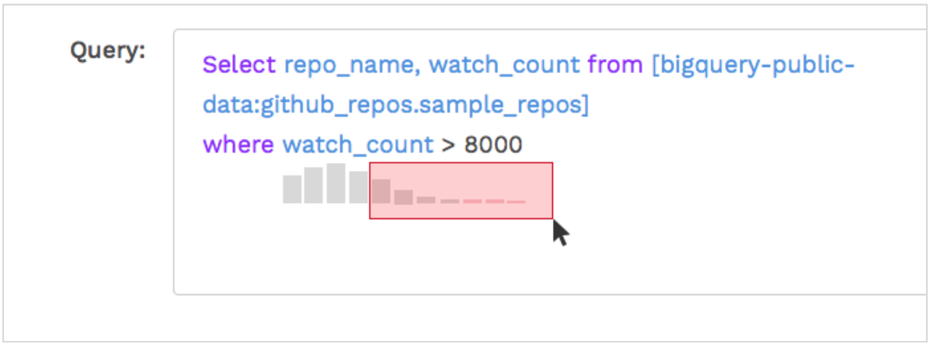
Numeric Condition Effect Visualization

A common concern when applying a filter on a numeric variable is that users are unsure whether the operand applied are low or high (e.g. “Is $40,000 in sales per month considered high?”). Without a priori knowledge about the dataset, this is impossible to know. This feature intends to show the user where their operand are in the distribution and allows a quick way to revise the query by brushing.



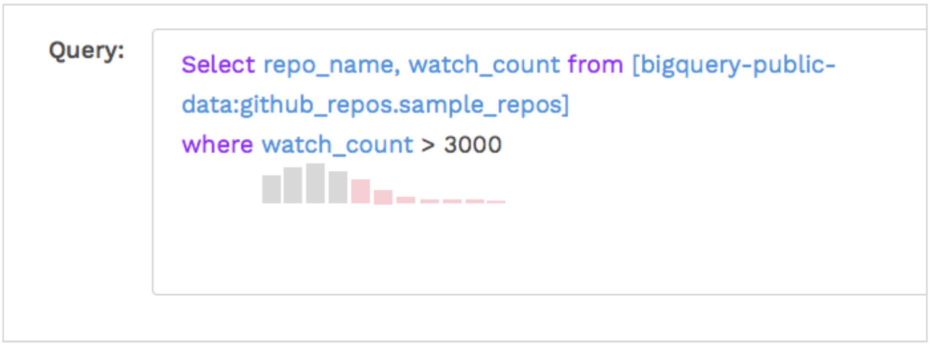
Query:

```
Select repo_name, watch_count from [bigquery-public-
data:github_repos.sample_repos]
where watch_count > 8000
```



Query:

```
Select repo_name, watch_count from [bigquery-public-
data:github_repos.sample_repos]
where watch_count > 8000
```


Query:

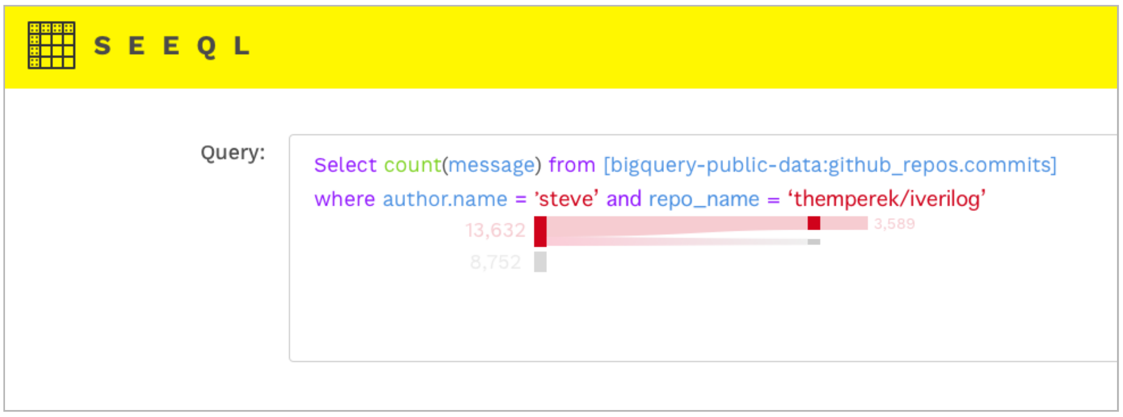
```
Select repo_name, watch_count from [bigquery-public-
data:github_repos.sample_repos]
where watch_count > 3000
```

Figure 10, numeric condition effect visualization (top)

and the brushing feature (second, bottom)

Compound Conditions Effect

As conditions can become complex, it might be useful to help the user validate their conditions by showing what effect the sub conditions would make collectively. For example, if there are two conditions connected by AND and one of them erratically yields 0 rows, the user might need need to make two separate queries to check which one of them has gone wrong. This feature intends to show how each condition would affect each other in runtime.



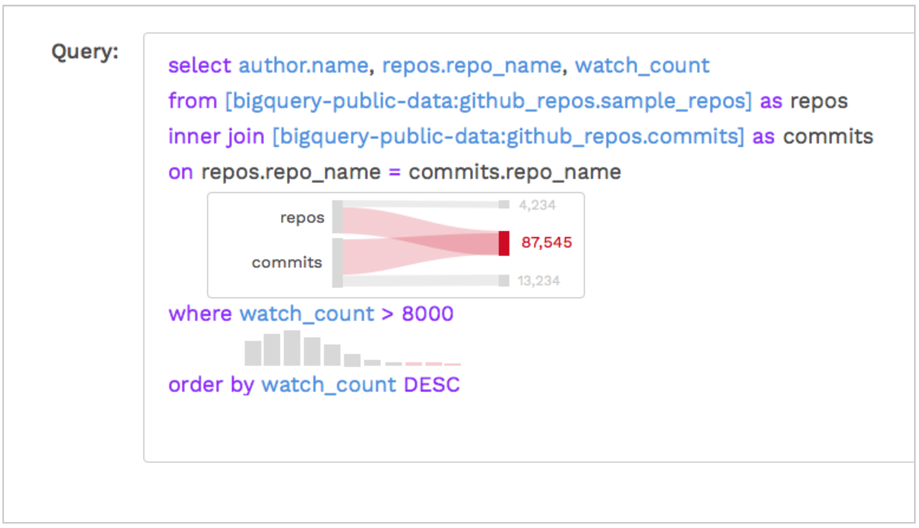
Query:

```
Select count(message) from [bigquery-public-data:github_repos.commits]
where author.name = 'steve' and repo_name = 'themperek/iverilog'
```

Figure 11, Compound condition effect visualization

Table Joins Visualization

Join operations are not only hard to visualize for beginners but also prone to error in a way that users might end up choosing a join condition that causes many null values on either of the tables. For this, I chose Sankey Diagram to visualize the join. The tables involved are visualized by each source in the diagram where destinations are separated into the different outcomes of the join (null on the left table, non-null, null on the right table) and the number of rows.



```
select author.name, repos.repo_name, watch_count
from [bigquery-public-data:github_repos.sample_repos] as repos
inner join [bigquery-public-data:github_repos.commits] as commits
on repos.repo_name = commits.repo_name
where watch_count > 8000
order by watch_count DESC
```

Figure 10, table joins visualization

FEEDBACK ON INITIAL DESIGN

Generally, the class found the tool to be useful for some certain parts of the data exploration process. However, some issues have been raised on the initial design.

- **How does all the visualizations stack against each other**
  For example, with a numeric condition involved, what would take priority between the compound condition visualization and the numeric condition effect visualization.
- **How to support more complex queries**
  The examples shown are relatively simple, yet they already reveal how problematic the visualizations could be. Is there are way to scalably support more complex cases such as sub queries.
- **How to simplify the prototyping process without going through the hassle of implementing a full-blown SQL client**

IMPLEMENTATION

Due to time constraints, only two features are implemented into the interactive prototype to validate some hypotheses that might be useful for future research. The prototype is implemented with ReactJS, CodeMirror, Google Cloud Functions, and Google BigQuery.

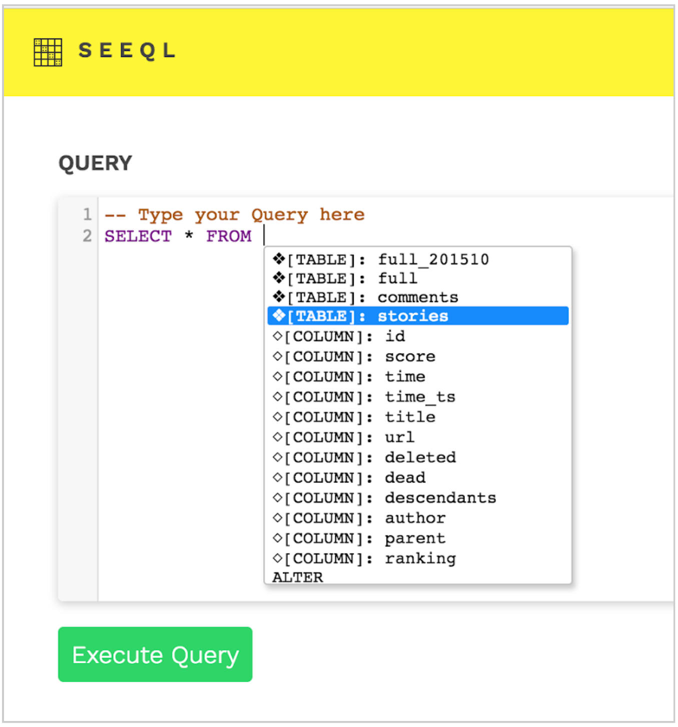
The source code for the prototype is available at https://github.com/5un/seeql

The live demo can be accessed at http://seeql.ml

Features

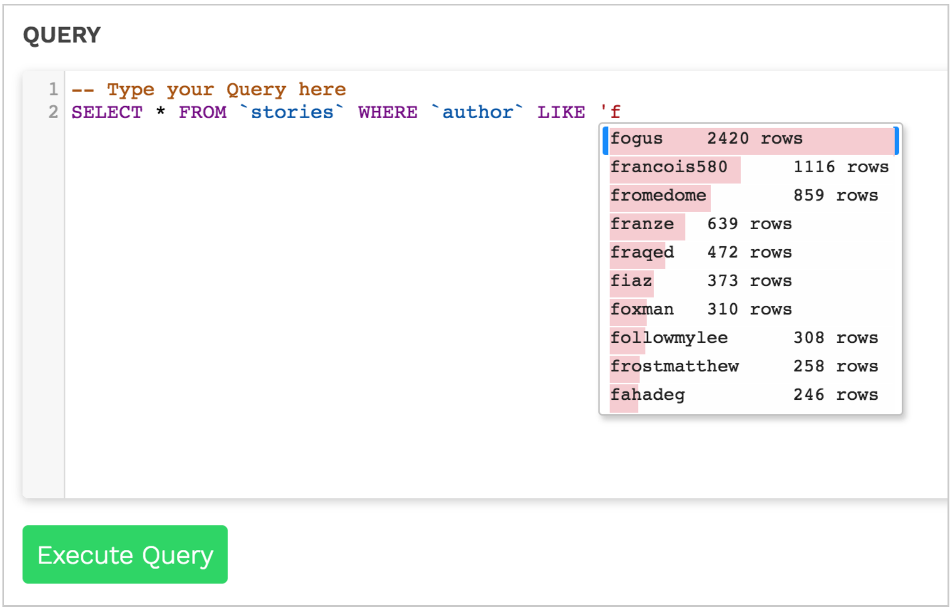
Identifier and column value auto suggestion

The prototype could suggest the names of tables and columns in the dataset as well as the distinct values in a column along with the number of rows.



```
-- Type your Query here
SELECT * FROM
❖[TABLE]: full_201510
❖[TABLE]: full
❖[TABLE]: comments
❖[TABLE]: stories
◇[COLUMN]: id
◇[COLUMN]: score
◇[COLUMN]: time
◇[COLUMN]: time_ts
◇[COLUMN]: title
◇[COLUMN]: url
◇[COLUMN]: deleted
◇[COLUMN]: dead
◇[COLUMN]: descendants
◇[COLUMN]: author
◇[COLUMN]: parent
◇[COLUMN]: ranking
ALTER
```

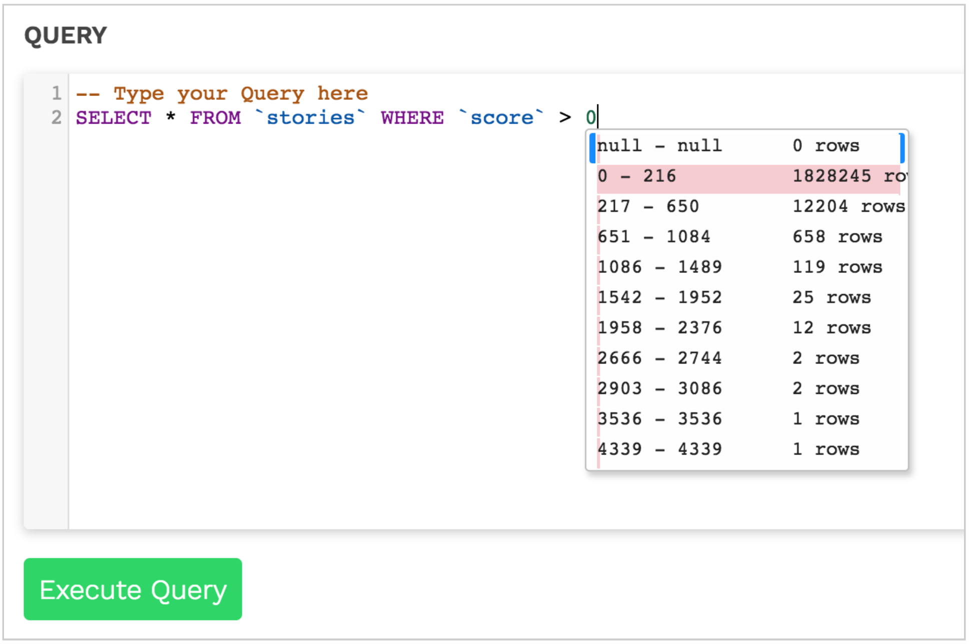
Figure 11, Identifier auto suggestion



```
-- Type your Query here
SELECT * FROM `stories` WHERE `author` LIKE 'f
fogus        2420 rows
francois580       1116 rows
fromedome         859 rows
franze       639 rows
fraqed       472 rows
fiaz         373 rows
foxman       310 rows
followmylee       308 rows
frostmatthew      258 rows
fahadeg           246 rows
```

Figure 12, column value auto suggestion

For numeric columns, the whole range will be separated into 10 equally-sized bins and displayed with a vertical histogram. Figure 13 shows the feature in action in the case that the data is highly skewed towards zero. This is because the `score` column represents the score of each post and most of the posts have the score of zero.



QUERY

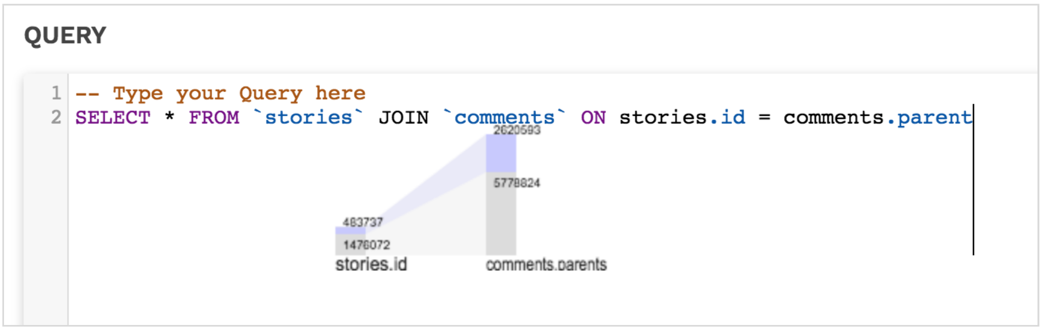
```
-- Type your Query here
SELECT * FROM `stories` WHERE `score` > 0
```

```
null - null        0 rows
0 - 216            1828245 ro
217 - 650          12204 rows
651 - 1084         658 rows
1086 - 1489        119 rows
1542 - 1952        25 rows
1958 - 2376        12 rows
2666 - 2744        2 rows
2903 - 3086        2 rows
3536 - 3536        1 rows
4339 - 4339        1 rows
```

Figure 13, Numeric column auto suggestion

Join Visualization

I realized that Sankey diagrams might not be the best format to visualize table joins as all the sum of the sources and destinations has to equal. Instead, I designed a diagram similar to Sankey which could show the mapping of one to many. In the diagram shown in Figure 14, each bar represents a table in the join and the purple highlighted areas are the parts of each table where the data could be joined.



QUERY

```
-- Type your Query here
SELECT * FROM `stories` JOIN `comments` ON stories.id = comments.parent
```

Figure 14, Join visualization

A demo video can be found at https://drive.google.com/file/d/1qrIBu9i_DMASXblfSdVYBYUYex1o4LLg/view?usp=sharing

Architecture

The overall architecture of the prototype could be illustrated as follows.

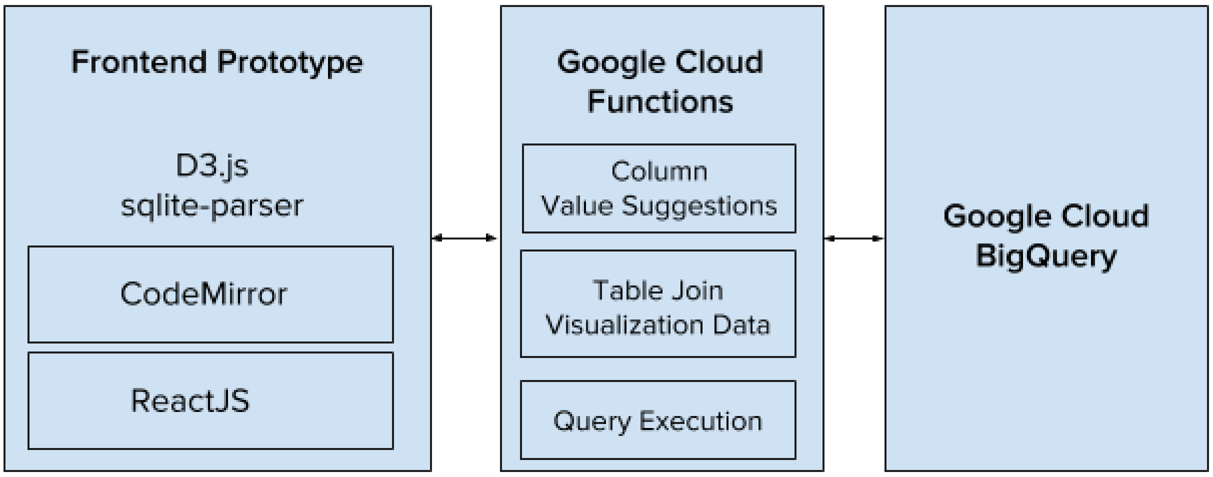


Figure 15, System architecture diagram

Key Technologies Used

- **Google Cloud BigQuery**: A highly scalable enterprise data warehouse from Google which provides a SQL interface. BigQuery was selected because of the speed of executing query over large dataset.
- **Google Cloud Functions**: A serverless infrastructure used for developing HTTP endpoints.
- **ReactJS**: A javascript library to create highly reusable UI components.
- **CodeMirror**: An opensource code editor in javascript which allows extensions to autocompletion system and text marking.
- **CodeSchool's sqlite-parser**: A SQLite parser by CodeSchool.

When an input is entered from the user, the partial SQL statement is then parsed and analyzed. In the case that the statement is not complete, a token analysis is used instead of fully parsing the statement. If the cursor is at the appropriate position for column value suggestion, a suggestion request is send to a cloud function. The function then translate the request into aggregation query according to the type of the column and the prefix value. The query is then execute on BigQuery in real-time and the result is then sent back to the front-end prototype to be displayed. Similarly, when a table join syntax is detected in the input, the prototype then analyzed the join statement into source table names and request for aggregate values from another cloud function.

EVALUATION

A usability testing is conducted with three participants with some experience using SQL. Each user was asked to perform three query tasks without knowing the structure of the dataset. Qualitative feedbacks are collected, analyzed, and synthesized into design recommendations for future work. Some qualitative feedback was also collected fromthe INFO-247 Class Final Project on May 2nd, 2018.

The usability testing is conducted on the hacker_news dataset [9], one of the public datasets available on Google Cloud BigQuery. The dataset represents the posts and comments on HackerNews (https://news.ycombinator.com/)

Usability Tasks

1. Query the stories of the top authors, in terms of number of stories posted, whose name begins with the letter ‘f’.
   Followup Questions:
   a. Was the autocomplete display helpful in completing your query task?
   b. Did the autocomplete display help you learn about the dataset?
   c. What would you expect to see in the popup that would be helpful for your query composition?
2. Query the stories with relatively high scores. The pariticipant should come up with their own definition of high.
   Followup Questions:
   a. Was the autocomplete display helpful in completing your query task?
   b. Did the autocomplete display help you learn about the dataset?
   c. What would you expect to see in the popup that would be helpful for your query composition?
3. Query the join of the stories table and comments table with the condition where the story.id equal to comments.parent.
   Followup Questions:
   a. Was the join annotation helpful in completing your query task?
   b. Did the autocomplete display help you learn about the dataset?
   c. What would you expect to see in the annotation that would be helpful for your query composition?

RESULTS DISCUSSION

Overall, there are cases where the users think the proposed system would be useful for their query tasks. The qualitative feedbacks can be categorized into the following areas

Feature-based feedback

- Categorical Column Value Autocompletion:
    - It's helpful to learn about the top values and the distribution. But, for some users, it is not apparent that the suggestions that come up are the values in that column. Having some headers could help with the problem.
    - Showing the number of rows might not be enough in some cases. The user would like to know how much of that table have the value too e.g. '500 rows out of 1000 roles' instead of just showing '500'
    - The quantity bar in the background of each row might not be very obvious for some users. This deserve some design attention.
    - Users expect to see special treatment for null values. In case that the columns are mostly null, other values will not be displayed in a very helpful way. An option to show or hide null might help.
    - A user think that showing 10 top suggestions is too few and ask that the suggestion pane should also be scrollable.
    - Some users does not realized that the auto suggestion works for empty string literals as the panel took some time to show up.
- Numeric Column Value AutoCompletion:
    - It is nice to be able to see the distribution of the data and to learn that it's skewed early on.
    - However, there's no easy way to drill down the data and numbers could not be prefixed just like text does.
    - Users would like to be able to drill down to see the distribution of a narrower range. Some expected to do it by selecting one of the range options suggested. Another user would like to have a range slider bar within the auto suggestion pane.
- Autocompletion in Compound Condition
    - The user unanimously agree that the auto suggestion for subsequent condition has to take the effects from the former conditions into account. For example, if the user was typing the second condition in an AND condition, the suggests has to be filtered out by the first condition.
- Table joins visualization

- The join visualization is very helpful to see how much of the table are actually joined
- Most users picked up right away that the bars represent columns and the segment connecting them represents the join.
- From the way it's designed right now, it was not clear that the top segment are the parts where the join succeeded. The could probably be solved with better color selections
- The join visualization does not explain the relationship between the involved keys. The user could type "stories.id = comments.parent" while not understand what comments.parent is.
- The join diagram could display some hints on the foreign key relationship. Usually, if there's such relationship, the table with foreign key will be fully mapped. That would be helpful for database administrators who would like to check if such relationship is in place.
- The table join diagram should be interactive to show other useful values.

Major Themes

- **The benefits of the tool: explore and validate**
  Some users see SeeQL as a tool to learn more about the dataset while a user see it as a tool to validate his knowledge about the query. He brought up a scenario from his previous job where he was anticipate similar dataset every month and usually have the rough idea of what the data should look like. The tool could quickly check if something has gone wrong in his query before starting to work with it. Another user also raise a potential usecase of using the table join diagram to check the integrity of foreign key relationship.
- **The most suitable usage of the tool still unclear:**
  Some users raised an issue that they were not sure whether they suppose to get the answer from the auto suggestion or by executing the query itself. This might have something to do with how the usability testing tasks are formed. The auto suggestion sometimes give the answer that has to be obtained by executing the query. This could be solved by designing more complex tasks to test.
- **The use of SQL:**
  Users would like to be able to use ALIAS syntax to simplify the query.

CONCLUSION AND FUTURE WORK

I designed and implemented SeeQL, a prototype system for real-time SQL query augmentation system to help with query structure and dataset understanding. The usability testing results shows that the users found the proposed features useful for the exploration and validation of the dataset. However, it is still unclear to the user how to best use this tool. Should they expect the answer from executing the query or the code annotations themselves. The usability testing also reveals a number of usability issues to be addressed in the next iterations.

For future research, the prototype has to support more complex queries in order to answer the relevant research questions. With the simple queries introduced in the usability test, the users was unable to see how the suggestions and annotations are helping with the task at hand. Among the high priorities are supporting the ALIAS keyword, finding appropriate design solution for complex conditions, understanding more complex join clauses. Some reach goals includes a support for GROUP BYs and sub queries.

CONTRIBUTOR

Soravis Prakkamakul

Ideation

Conducted Literature Review

Selected an Example Dataset from Google Bigquery Public Datasets

Designed Initial User Interface Mockup in Sketch

Developed the prototype in Javscript/React

Wrote Usability Testing Protocol

Conducted 3 Sessions of Usability Testing

Analyzed and Synthesized Results

Wrote the Project Proposal and Final Report

APPENDIX

Usability Testing Session Notes

Subject #1

Task 1: Categorical value query suggestion
- Generally the suggestion was helpful
- The table and column suggestion come up, which is good
- But I kind of don’t see the suggestion coming up for the empty case. (The suggestion came up too slow)
- The task is confusing as I don’t know if the query should find out the answer or the answer is the query itself
- Tried to use the LIKE statement to complete the first task
- I see the numbers first, not colors
- Is this thing case sensitive? I think it should not be the case
- Definitely help learning about the dataset.
- What else to see? I would like to see null values

Task 2: Numerical value query suggestion
- Is there any way to show the column type?
- To self: The task could be frame as in finding the ‘right’ value for that data set given you don’t know about the dataset.
- Is it helpful: Not really because I cannot drill down on to the range
- There should be a way for me to filter down the range
- I want to be able to identify the nulls in the dataset
- I also should be able to know if this table is valid
- It depends on whether I’m querying the dataset or build upon it

Task 3: Join visualization
- it was kind of helpful
- I cannot think of other information to show
- Or maybe it could show the relationship because I don’t know what comment.parent is
- I picked up that they’re columns right away
- The colors also tell me right away if this is a the matched area

Question 4: Design for multi-conditions
- I don’t see the association with the columns at all. I thought the first bar is for the whole condition

Subject #2

Keyword and Identifier Suggestion

- Ambiguous display: not sure which are keywords, which are table and column names

Column Value Suggestion

- It's not clear whether the the list shown are column values. Even though there are number of rows, it couldve been keywords as well
- Expect it to be scrollable
- The null kind of make other values unreadable
- would be cool if there's a way to show null non null

Numeric Column Value Suggestion

- The range is not continuous
- I should be able to click and see the subdivs of the range

Two conditions

- The second condition should be limited by the first column

Join Visualization

- The colors are actually confusing. I though the bottom part is the joinable part
- It could be improved with diff shades of same color
- It is not clear that this denotes a join operation. Actually it could use a join symbol
- It needs more clarity

Subject #3

Keyword and Identifier Name Suggestion

- picked it up right away, probably because he's familiar with this kind of feature

Column Value Suggestion

- Not helpful in this case but helpful for understanding the dataset in general
- I'd like to see valid vals for the column, for example the column could be 12 values, I should be able to know what can I query
- This acts as a validation for me as in my professional settings I kind of know what I'm expecting from the dataset. For example, monthly daset
- If there's a showing of how much of the table is this value, this could be helpful (the total number of rows should be shown)
- May be this could show mean or other statistics?

Numeric Column Value Suggestion

- this is helpful
- I would expect the press enter to drill down into the range
-

Two condition var
- The first condition should filter the second condition
- again it act as a validation for known dataset

Join visualization
- maybe the condition should come up mid-join
- unclear colors
- does not look like a bar to be. It looks like a line chart at first.
- Bottom part should not be connected
- When I try to construct a join, I always experiments with smaller queries before being able to come up with a full one, this could help me prevent that

Other info:
- The partificapant use terradata

My Thoughts:
- maybe legends could help with the joins

REFERENCES

1. H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. 2007. *Making database systems usable.* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD '07). ACM, New York, NY, USA, 13-24.

2. Minsuk Kahng, Shamkant B. Navathe, John T. Stasko, and Duen Horng Polo Chau. 2016. *Interactive browsing and navigation in relational databases.* Proc. VLDB Endow. 9, 12 (August 2016), 1017-1028.

3. Robert Pienta, Acar Tamersoy, Alex Endert, Shamkant Navathe, Hanghang Tong, and Duen Horng Chau. 2016. *VISAGE: Interactive Visual Graph Querying.* In Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '16), Paolo Buono, Rosa Lanzilotti, and Maristella Matera (Eds.). ACM, New York, NY, USA, 272-279.

4. Jonathan Danaparamita and Wolfgang Gatterbauer. 2011. *QueryViz: helping users understand SQL queries and their patterns.* In Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11), Anastasia Ailamaki, Sihem Amer-Yahia, Jignesh Pate, Tore Risch, Pierre Senellart, and Julia Stoyanovich (Eds.). ACM, New York, NY, USA, 558-561.

5. SQLizer Blog. 2017. *SQL is 43 years old - here's 8 reasons we still use it today.* retrieved from https://blog.sqlizer.io/posts/sql-43/

6. Alation. *Alation Home Page.* retrieved from https://alation.com/

7. SQLDep. *SQLDep Home Page*. retrieved from https://sqldep.com/

8. Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2018. *Augmenting Code with In Situ Visualizations to Aid Program Understanding.* In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). ACM, New York, NY, USA, Paper 532, 12 pages.

9. Google BigQuery. *Hacker News Dataset.* retrieved from https://cloud.google.com/bigquery/public-data/hacker-news