Phase 4 Documentation

Ben Gregory, Logan McAllister, Aidan Thorp, Anthony White

Table of Contents

Introduction

We aimed to provide easier visualization and comparison of Pennsylvania educational data. The application was made to be quick and easy to use while providing detailed information from a centralized database. The target market for this application is Pennsylvania policymakers, school administration, private stakeholders, and anyone else with an interest in the efficacy of the Pennsylvania educational system.

Research Question

Our Research Question was:

How can we best display pertinent data to policymakers so that they can analyze the impact of various factors on educational success?

We considered the following factors:

- School fiscal information
- District fiscal information
- PSSA Scores
- Keystone Scores
- Graduation Destinations

Our goal was to give policymakers and educational administration as much freedom as possible to compare schools and districts, and to visualize how different (primarily fiscal) factors correlate to positive or negative performance in PSSA exams, Keystone exams, and where graduates ended up.

Considerations and Parameters

We designed our tool to be simple and easy to use for all constituents, with a focus on making the data easy to read and understand. Potential users have the option to easily interface with the axis on the internal database, creating a seamless selection process for multiple output types such as visual representations, summary information, or comparisons between axes. To achieve this goal, we aimed to create a system that is highly dynamic in its ability to present and handle different forms of data, despite seemingly simple user requests.

Development Roles and Project History

Name	Role	Focus
Aidan	Presentation Manager	Frontend
Anthony	Project Manager	Frontend
Ben	Infrastructure Coordinator	Full Stack
Logan	Database Administrator	Backend

Week	What we did
10/8 - 10/14	Presentation, db schema first design, data cleaning
10/15 - 10/21	data cleaning, db design
10/22 - 10/28	data cleaning, db design, develop initial application
10/29 - 11/4	Finalize schema and create tables, data cleaning, frontend design, develop application request and response format
11/5 - 11/11	Phase 3 document, presentation, data cleaning, frontend design, develop database connection
11/12 - 11/18	Data import, frontend, develop charting utilities
11/19 - 11/25	Data import, dynamic queries for application, frontend, develop charting utilities
11/16 - 12/2	Final document, dynamic queries for application, finalize frontend, develop charting utilities

Limitations of Source Data

The Pennsylvania educational data used in this project provided our group with immediate limitations, the most problematic of which being:

- Missing/incorrect data
- Qualitative attributes and other unhelpful data sets
- Limited years of collection

Missing/incorrect data

Several of the data sets provided for this project contained tuples with no actual substance. Either the school name/number that was supposed to provide context for the data was missing, or the school was there but the data itself was left blank. On top of this there were situations where the data was clearly supposed to follow a certain constraint (i.e. 4 percentiles adding to 100%), but did not, for any number of reasons. These issues were dealt with during data cleaning, but as a result the cleaned data was missing certain tuples that ideally should have been there had the data been better.

Qualitative attributes and other unhelpful data sets

Certain sections in the source data were entirely unhelpful, and this caused problems with our goal of creating as comprehensive of a tool as possible. We originally intended to include as much of the provided data as we could, but certain data files such as “Future Ready Performance Data,” which should have provided us with helpful specific academic information about each school, instead gave their values as a set of colors, with up or down on the end. For example, the 21st Century Cyber CS Annual Progress Growth Science/Biology (All Student), which one

might think should be a percentage, was “bluedown.” This made the data almost entirely unusable, and limited our ability to account for certain academic factors.

Limited years of collection

Another limiting factor we ran into with the data was the amount of years it was collected from. This was particularly problematic with the school and district fiscal information, as we only had 1-4 years of each. While just one year let us at least display the information, it did not give us as full of a picture as we would’ve liked, and caused the section of data to be of limited use compared to others.

Codebase Directory Structure

Directory Map

Using the command: “tree /A /F | Where-Object {$_ -notlike "*node_modules"} > tree.txt”

The directory structure looks as such from…

C:. ./CS320_DatabaseManagement/CS320_V03

| .env

| app.js

| chart.js

| dataUtils.js

| dbConfig.js

| dbUtil.js

| package-lock.json

| package.json

| server.js

| tree.txt

|

+---node_modules… (Contains javascript dependencies)

|

+---public

| | 07CAT-STRIPES-superJumbo.png

| | exampleImg.png

| | siteChart.png

```
|   |   siteChart.svg
|   |
|   \---css
|           styles.css
|
\---views
    |   404.ejs
    |   index.ejs
    |   summary.ejs
    |   visualizer.ejs
    |
    \---partials
            chart.ejs
            chart_data_select.ejs
            head.ejs
            selection_form.ejs
            style.ejs
            summary_data_display.ejs
            summary_data_select.ejs
            title.ejs
```

Notable Directories

“./CS320_V03/”: This is the parent directory of the project. This is where the core javascript (NodeJS) codebase is contained in addition to resource directories. The package.json file for handling npm dependencies is also stored here.

“./CS320_V03/public”: This resource directory contains visual components of the program, such as images and dynamic files generated at runtime.

“./CS320_V03/public/css”: This directory contains CSS styling for frontend visualization and content.

“./CS320_V03/views”: This directory contains the dynamic EJS files that we use for rendering frontend HTML based on different program states.

“./CS320_V03/views/partials”: This directory contains partials of dynamic EJS files so that we are able to divide frontend design during the development process.

System Design

Because of the multi-page layout of our program and the differing page designs between user requests, our program was built to handle multiple overlapping states. The program states refer to the page, content, user requests, request order, page selection options, and a multitude of other dynamic values. By handling a dynamic program state, we were able to create a minimal frontend layout which is dependent on a few details. Furthermore, user selection options and generated graphs can be handled conditionally by registering the program state. Our program uses an SSH tunnel configuration for our database connection. This connection is opened once at the beginning of the program and is closed at termination, allowing the program to handle a multitude of sequential database queries. Database queries are predefined in structure, but they are further defined using formatted strings so that table, attribute, or subject values can be determined at runtime. In addition, our graphing utilities use a similar layout to access the Plotly JS API for creating charts and graphs of datasets. Since we are using NodeJS, which runs via the backend, we have no capability to render dynamic images via the Plotly library or our own without introducing a large amount of complexity. As a solution, the images generated by the Plotly API are streamed to an internal *.svg (Scalable Vector Graphics) file within our directory structure and rendered on the front end by referencing a URL. The overall program is organized into a number of complete modules. These modules include, but are not limited to:

Server.js – This module is used to open an express app and define initialize program dependencies.

App.js – This module holds functionality for handling user requests and program responses for the web application.

Chart.js – This module consists of the charting/graphing utilities and connection to the Plotly JS API.

DataUtils.js – This module is used for handling the complex program states and organizing data requested by other program modules.

DBConfig.js – This module holds the functionality for our SSH tunnel to the MYSQL database connection.

DBUtils.js – This module is used to query the database connection defined in the 'dbConfig.js' module.

The program uses a number of toolsets, frameworks, and dependencies. The most important toolsets involved in development and our codebase are as follows…

I. NodeJS – The core functionality of our program is written via the NodeJS javascript framework.
II. Express – Communication between program functionality and user requests is handled via an Express app.
III. EJS & CSS – Dynamic frontend designs are constructed using EJS and CSS.
IV. NPM – Package management for program and development dependencies is handled via this tool.
V. MYSQL – The program data is stored within a MYSQL database.

General Database Queries

Get the number of schools that have ranking information for PSSA or Keystone scores

```
Unset
select count(*) as cnt

       from

       (select school, sum(num_scored), avg(p_advanced) as avg_adv,

avg(p_proficient) as avg_prof, avg(p_basic) as avg_bas, avg(p_below_basic) as

avg_bbas

       from ${table}

       where school in (

              select school_num

              from School

              where district = ${district[0].district}
```

Get the rank of each school within its district with regard to weighted Keystone or PSSA scores

```
Unset
select n

       from

       (select @n := @n + 1 n, x.school as school, (x.avg_adv * 3 + x.avg_prof * 2 +

x.avg_bas * 1) as weighted_avg

       from (select school, sum(num_scored), avg(p_advanced) as avg_adv,

avg(p_proficient) as avg_prof, avg(p_basic) as avg_bas, avg(p_below_basic) as

avg_bbas

       from ${table}
```

```
        where school in (
        select school_num
        from School
        where district = ${district[0].district})
        group by school) as x, (SELECT @n := 0) m
        order by weighted_avg desc) as y
        where school = ${school_num};
```

Get district and school name for specific school.

```
Unset
SELECT district, name FROM School WHERE school_num = ${school_num};
```

Get schools from a district excluding those without certain values

```
Unset
SELECT DISTINCT * FROM School WHERE school_num != "-1" AND name != "-1" AND district IN
(SELECT aun FROM District);
```

Get all attributes of all schools that have values in both variable tables

```
Unset
SELECT DISTINCT * FROM School WHERE school_num IN (SELECT DISTINCT t1.school FROM
${attributes[horizontal].table} t1, ${attributes[vertical].table} t2 WHERE
t1.school = t2.school);
```

Get number of schools with data in both given tables per district for each district in descending order

```
Unset
SELECT DISTINCT aun, name, cnt FROM District as t1, (SELECT Count(*) as cnt, district
from School Group By district) as t2 WHERE aun IN (SELECT district FROM School WHERE
school_num IN (SELECT t1.school FROM ${attributes[horizontal].table} t1,
${attributes[vertical].table} t2 WHERE t1.school = t2.school)) AND district != 0
Order By cnt DESC;
```

Get two variable attributes from all schools in a given district

```
Unset
SELECT t1.school AS school, (t1.${value1}) AS ${value1}, (t2.${value2}) AS ${value2}
FROM ${table1} t1, ${table2} t2 WHERE t1.school = t2.school AND t1.school IN (SELECT
school_num FROM School WHERE district = ${aun}) GROUP BY t1.school;
```

Get average of a variable attribute and another variable attribute

```
Unset
SELECT DISTINCT AVG(t1.${value1}) AS ${value1}, t1.${value2} FROM ${table} AS t1
WHERE school = ${school_num} GROUP BY t1.school, t1.${value2};
```

UI Design

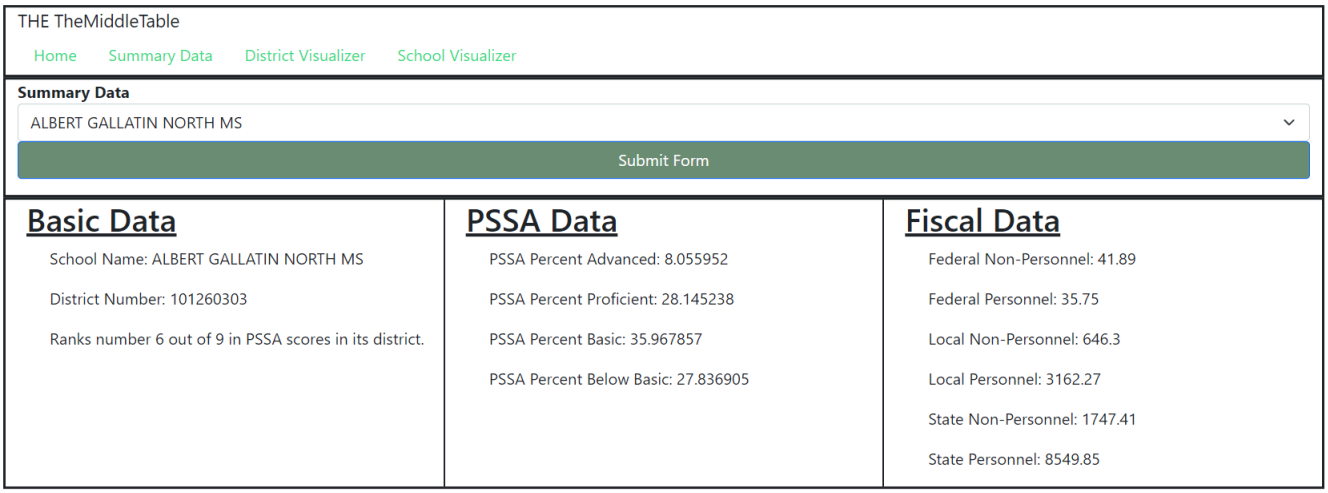
When developing the UI of our website we wanted to focus on ease of use and clarity, minimizing the potential friction the user could experience from unclear design. Our home page has our three pages front and center, clearly guiding the user to the different pages that they can access.

Our summary page is extremely simplistic in design, allowing the users to pick a school and see available general, test, and financial data on it. The data is organized in a way where it can all be clearly seen on the screen at once.

The Data visualizer for district and school data creates a simple graph from options that the user can select from. The first option is from any available dataset for the vertical axis, the second option is from available datasets for the horizontal axis that corresponds to the option selected for the vertical axis, and the final option selects the district/school that the data is sourced from. Users can also add more districts/schools, which will update the graph to add the new data.

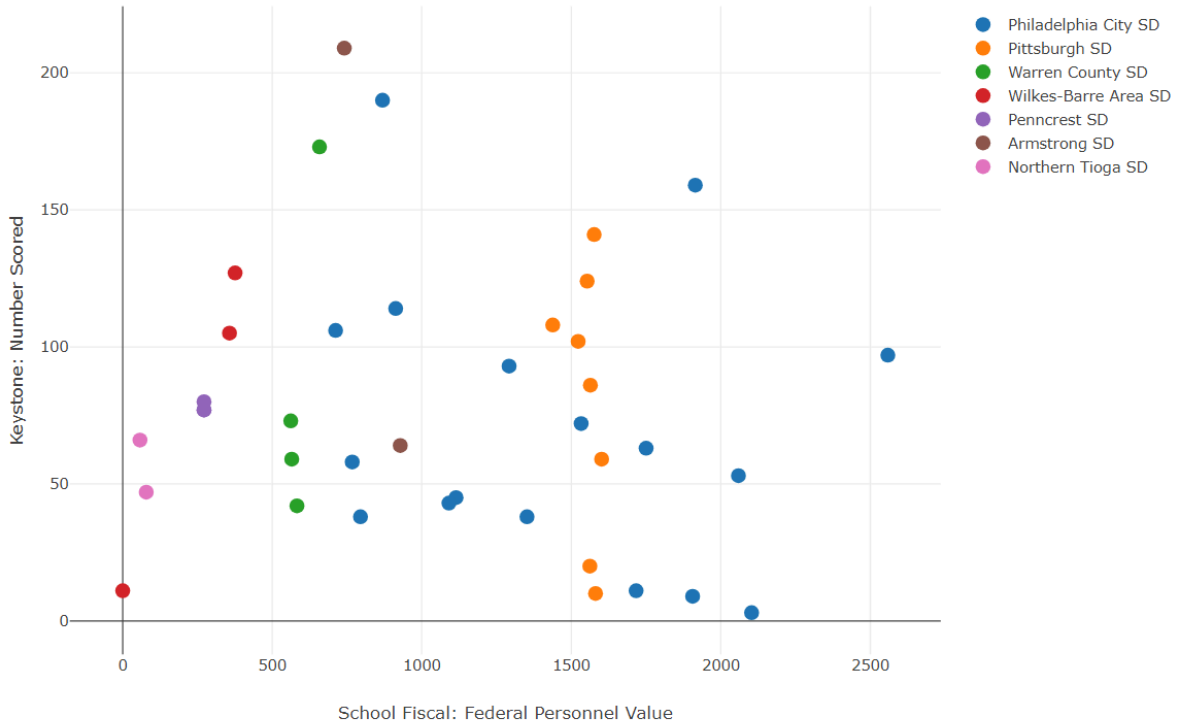
Example Usage

Summary Data Page



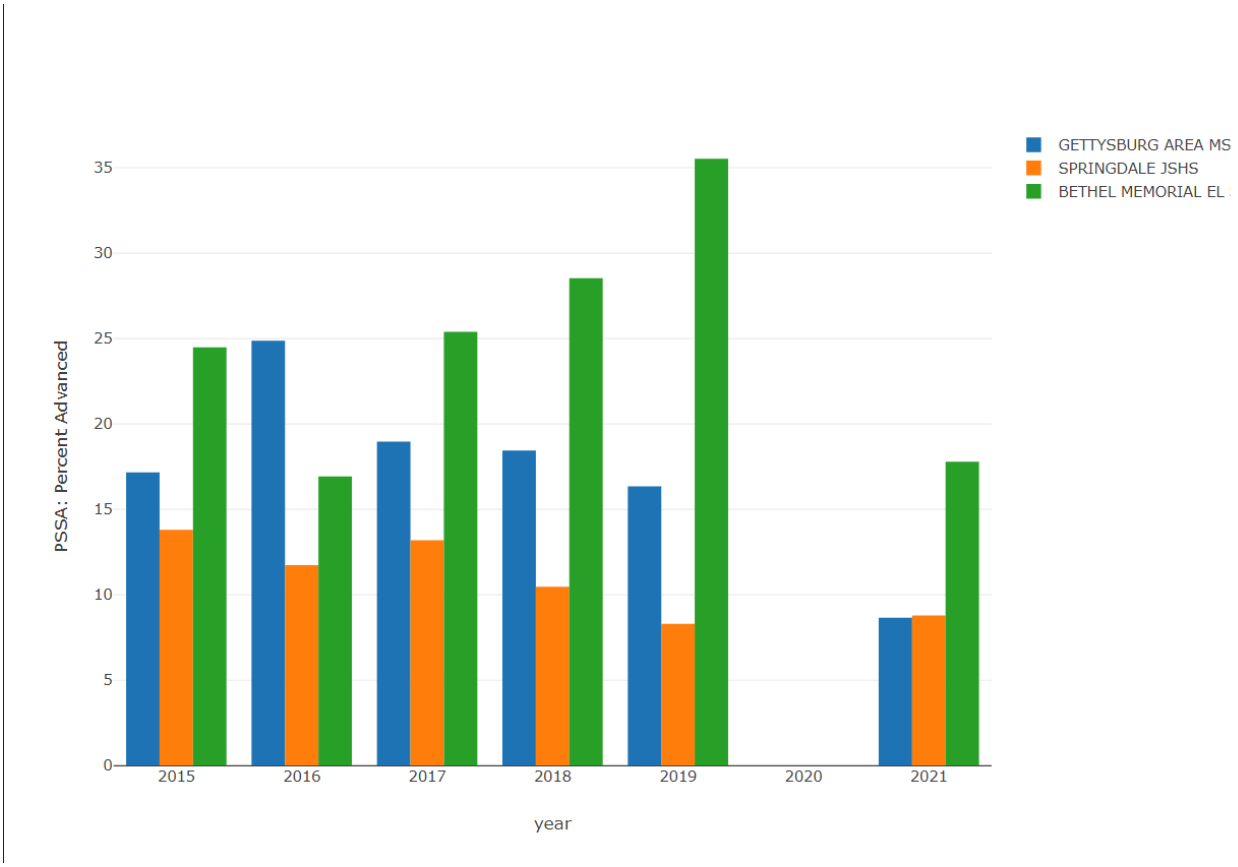
This is an example of our summary page, with the summary data for Albert Gallatin North Middle School. Each page is broken into 2-3 sections depending on if fiscal data is reported for the school. The basic data section displays the school name, AUN number (identifying number between districts), and its ranking in either PSSA or Keystone scores depending on if it's a middle school or high school. The PSSA Data or Keystone Data shows data about students taking these tests at the selected schools. While unaware of the specific metric they use to differentiate the different groupings, it goes from generally scoring well (Percent Advanced) to scoring poorly (Percent Below-Basic) on the respective exam. The fiscal data represents the school's funding for different areas from different levels of government.

District Visualizer Example



This graph depicts the schools in several districts, with the horizontal axis representing the Federal Personnel Value for the school, and the vertical axis representing the number of students who took the Keystone exam at that school. Each district is denoted by a color, which can be seen in the key in the top right.

School Visualizer Example



This graph shows a comparison of 3 different schools on the PSSA percent advanced attribute over a period of 7 years. This demonstrates one of the applications central features, the ability to look at schools' side by side values for a number of different factors, focused primarily on test scores and funding in the current state of the website.

Development Pattern

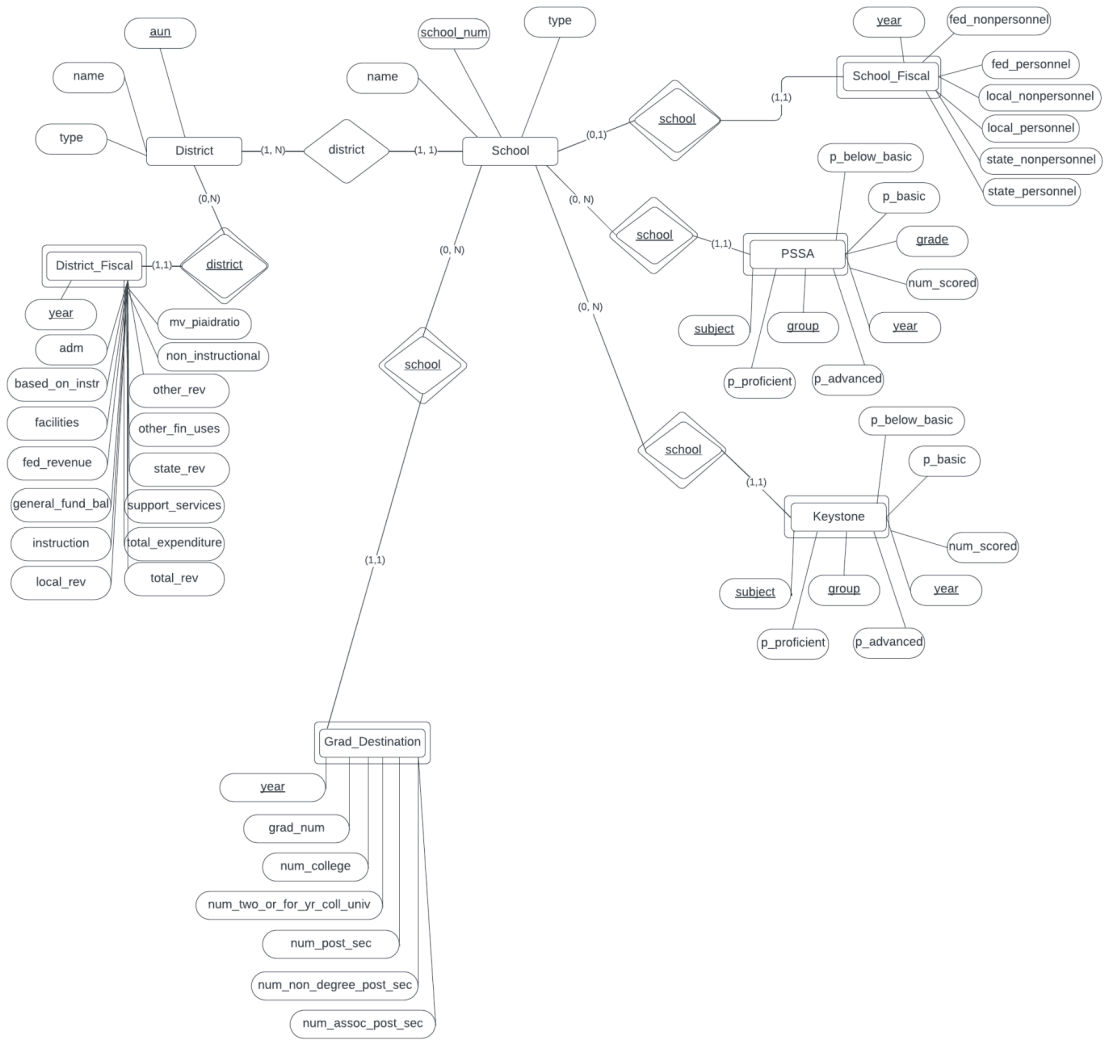
As a focus of the development process, the group met more than once per week on average. These meetings were used to prioritize tasks and set realistic expectations for future development goals. In addition, we used this time to determine development requirements, faults, roadblocks, misunderstandings, or any difficulties that might have arisen during individual development. A major focus of our administrative leader, Anthony, was the prioritization and compartmentalization of development tasks. This meant that tasks could be assigned separately and deadlines could be maintained, avoiding conflicts between development outcomes. During the development process, several overarching goals were determined, such as…

I. Initial comprehension of Pennsylvania public school data
II. Designing the program objective/purpose
III. Designing the database schema based on the program's objective/purpose
IV. Designing the program organization
V. Implementation of the program design
VI. Implementation of frontend design

Between key development outcomes, our weekly meetings were used to determine how well we have met our goals and to readjust the current trajectory.

Database Design

ER Diagram


aun
name
type
District
(1, N)
district
(1, 1)
School
school_num
name
type
(0,N)
district
District_Fiscal
(1,1)
year
adm
based_on_instr
facilities
fed_revenue
general_fund_bal
instruction
local_rev
mv_piaidratio
non_instructional
other_rev
other_fin_uses
state_rev
support_services
total_expenditure
total_rev
(0,1)
school
(1,1)
year
School_Fiscal
fed_nonpersonnel
fed_personnel
local_nonpersonnel
local_personnel
state_nonpersonnel
state_personnel
(0, N)
school
(1,1)
PSSA
p_below_basic
p_basic
grade
num_scored
subject
group
year
p_proficient
p_advanced
(0, N)
school
(1,1)
Keystone
p_below_basic
p_basic
num_scored
subject
group
year
p_proficient
p_advanced
(0, N)
school
(1,1)
Grad_Destination
year
grad_num
num_college
num_two_or_for_yr_coll_univ
num_post_sec
num_non_degree_post_sec
num_assoc_post_sec

ER Diagram Explanation

This ER diagram is centered around the school and district entities, each holding all of the relevant basic information about a school or district, with school number and AUN (Administrative Unit Number) as the primary keys. School also has a relationship to the district, representing which district the school is in. Every other entity in this ER diagram is a weak entity, with part of its primary key being the relationship between it and the school or district entities. All of these weak entities are tied to the District and School entities by the school number or AUN, and most only have this as well as the year that the data is from as their primary key. The exceptions to this are the PSSA and Keystone entities, where more identifiers are needed to determine all attributes in the entity, so subject, group, and (in PSSA only) grade are added to the primary keys.

All of these weak entities fully participate in the relationship between them and school or district, as each of the tuples needs to be related to a school or district for the data to make sense. The full participation is denoted by the (1,1) structural constraint rather than the typical double line, as the double line in Lucidchart just made the line look thicker instead. The relationships between school or district and each of these weak entities do not have full participation, as each school or district can have any number of tuples in the weak entities referencing them. This is represented in the ER diagram with the (0, N) structural constraint on these relationships.

Database Normalization

Every table in this schema has been normalized to the third normal form. This was done by removing redundant attributes tied to the school or district that the data is referencing, and keeping only the school number or district AUN outside of the district and school tables themselves. In cases where the original data did not contain the school number, but instead the name of the school as its primary key, this was substituted out for the school number using a natural join matching school names. Each school name is unique, meaning that it can also function as a key, but we decided to keep the key consistently as the school number across all of the weak entities.

Third normal form was chosen over BCNF for this project as there are only a few functional dependencies outside of the super keys, and none of them make sense to drop. One example of this comes from the PSSA and Keystone tables, where there are 4 percentage values that theoretically should add up to 100%. With this constraint, any 3 of these values would technically determine the fourth (though not in practice, as mistakes were made with the data). In BCNF, this table would need to be composed, as none of those functional dependencies are super keys, but in practice we keep all 4, so no calculations are needed and no data is needlessly split off. The other area where this is applicable is the district fiscal data. This table has two total value attributes at the end, which in theory could each be determined by all of the revenue/expense attributes, and are actually entirely redundant given that they can be found with simple addition. In practice these values are already given in the data, so instead of requiring this addition every time the total is needed, the values remain in the table.

Build Tables

```
Unset
'CREATE TABLE `District` (
 `aun` int(11) NOT NULL,
 `name` varchar(255) DEFAULT NULL,
 `type` varchar(255) DEFAULT NULL,
 `county` varchar(255) DEFAULT NULL,
 PRIMARY KEY (`aun`)
)
```

```
Unset
'CREATE TABLE `School` (
 `school_num` int(11) NOT NULL,
 `district` int(11) NOT NULL,
 `name` varchar(255) DEFAULT NULL,
 PRIMARY KEY (`school_num`),
 KEY `district_idx` (`district`),
 CONSTRAINT `district` FOREIGN KEY (`district`) REFERENCES
`District` (`aun`) ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

```
Unset
'CREATE TABLE `School_Fiscal` (
 `school` int(11) NOT NULL,
 `year` int(11) NOT NULL,
 `fed_nonpersonnel` decimal(14,3) DEFAULT NULL,
 `fed_personnel` decimal(14,3) DEFAULT NULL,
 `local_nonpersonnel` decimal(14,3) DEFAULT NULL,
 `local_personnel` decimal(14,3) DEFAULT NULL,
 `state_nonpersonnel` decimal(14,3) DEFAULT NULL,
 `state_personnel` decimal(14,3) DEFAULT NULL,
 PRIMARY KEY (`school`,`year`),
 CONSTRAINT `School_Fiscal_ibfk_1` FOREIGN KEY (`school`)
REFERENCES `School` (`school_num`)
)
```

```
Unset
CREATE TABLE `District_Fiscal` (
 `district` int(11) NOT NULL,
 `year` int(11) NOT NULL,
 `adm` decimal(14,3) NOT NULL,
 `based_on_instruction` decimal(14,3) DEFAULT NULL,
 `facil_acquisition_and_construction` decimal(14,3) DEFAULT
NULL,
 `fed_revenue` decimal(14,3) DEFAULT NULL,
 `general_fund_balance` decimal(14,3) DEFAULT NULL,
 `instruction` decimal(14,3) DEFAULT NULL,
 `local_rev` decimal(14,3) DEFAULT NULL,
 `mv_piaidratio` decimal(14,3) DEFAULT NULL,
 `non_instructional` decimal(14,3) DEFAULT NULL,
 `other_rev` decimal(14,3) DEFAULT NULL,
 `other_fin_uses` decimal(14,3) DEFAULT NULL,
 `state_rev` decimal(14,3) DEFAULT NULL,
 `support_services` decimal(14,3) DEFAULT NULL,
 `total_expenditures` decimal(14,3) DEFAULT NULL,
 `total_revenue` decimal(14,3) DEFAULT NULL,
```

```
 PRIMARY KEY (`district`,`year`),
 CONSTRAINT `District_Fiscal_ibfk_1` FOREIGN KEY (`district`)
REFERENCES `District` (`aun`)
)
```

Unset

```
CREATE TABLE `Grad_Destination` (
 `school` int(11) NOT NULL,
 `year` int(11) NOT NULL,
 `grad_num` int(11) DEFAULT NULL,
 `num_college` int(11) DEFAULT NULL,
 `num_two_or_four_yr_college_univ` int(11) DEFAULT NULL,
 `num_post_secondary` int(11) DEFAULT NULL,
 `num_non_deg_post_secondary` int(11) DEFAULT NULL,
 `num_spec_associate_post_sec` int(11) DEFAULT NULL,
 PRIMARY KEY (`school`,`year`),
 CONSTRAINT `Grad_Destination_ibfk_1` FOREIGN KEY (`school`)
REFERENCES `School` (`school_num`)
)
```

```
Unset
CREATE TABLE `Keystone` (
 `year` int(11) NOT NULL,
 `school` int(11) NOT NULL,
 `subject` varchar(255) NOT NULL,
 `group` varchar(255) NOT NULL,
 `num_scored` int(11) DEFAULT NULL,
 `p_advanced` decimal(5,2) DEFAULT NULL,
 `p_proficient` decimal(5,2) DEFAULT NULL,
 `p_basic` decimal(5,2) DEFAULT NULL,
 `p_below_basic` decimal(5,2) DEFAULT NULL,
 PRIMARY KEY (`year`,`school`,`subject`,`group`),
 KEY `school` (`school`),
 CONSTRAINT `Keystone_ibfk_1` FOREIGN KEY (`school`) REFERENCES
`School` (`school_num`)
)
```

Unset

```
CREATE TABLE `PSSA` (
 `year` int(11) NOT NULL,
 `school` int(11) NOT NULL,
 `subject` varchar(255) NOT NULL,
 `group` varchar(255) NOT NULL,
 `grade` int(11) NOT NULL,
 `num_scored` int(11) DEFAULT NULL,
 `p_advanced` decimal(5,2) DEFAULT NULL,
 `p_proficient` decimal(5,2) DEFAULT NULL,
 `p_basic` decimal(5,2) DEFAULT NULL,
 `p_below_basic` decimal(5,2) DEFAULT NULL,
 PRIMARY KEY (`year`,`school`,`subject`,`group`,`grade`),
 KEY `school` (`school`),
 CONSTRAINT `PSSA_ibfk_1` FOREIGN KEY (`school`) REFERENCES
`School` (`school_num`)
)
```

Data Dictionary

Attribute Name	Attribute Description	Domain	Relation	Source
fed_nonpersonnel	federal non personnel costs	double	School_Fiscal	1
fed_personnel	federal personnel costs	double	School_Fiscal	1
local_nonpersonnel	local non personnel costs	double	School_Fiscal	1
local_personnel	local personnel costs	double	School_Fiscal	1
state_nonpersonnel	state non personnel costs	double	School_Fiscal	1
state_personnel	state personnel costs	double	School_Fiscal	1
based_on_instruction	couldn't find description	double	District_Fiscal	1
facil_acquisition_and_construction	cost of facilities acquisition and construction	double	District_Fiscal	1
fed_revenue	federal revenue	double	District_Fiscal	1
general_fund_bal	general fund balance	double	District_Fiscal	1
instruction	cost of instruction	double	District_Fiscal	1
local_rev	local revenue	double	District_Fiscal	1

mv_piaidratio	MV/PIAid Ratio	double	District_Fiscal	1
non_instructional	Non instruction costs	double	District_Fiscal	1
other_rev	other revenue	double	District_Fiscal	1
other_fin_uses	other financing uses	double	District_Fiscal	1
state_rev	state revenue	double	District_Fiscal	1
support_services	cost of support services	double	District_Fiscal	1
total_expenditures	total expenditures	double	District_Fiscal	1
aun	Administrative Unit Number, unique numerical representation of district.	int NOT NULL	District	2
name	Name of district	varchar(255)	District	2
type	Public or private	varchar(255)	District	2
school_num	Unique school number assigned by state	int NOT NULL	School	2
district	AUN of district that school is in	int NOT NULL	School	2
name	Name of school	varchar(255)	School	2
county	County that school is in	varchar(255)	School	2

type	Public or private	varchar(255)	School	2
district	district where data is from	int NOT NULL	District_Fiscal	2
year	year that data is from	int NOT NULL	District_Fiscal	2
year	year of exam	int NOT NULL	Keystone	3
school	school number, used to identify which school the data is from	Int NOT NULL	Keystone	3
subject	subject of exam	varchar(255)	Keystone	3
group	All students or historically underperforming	varchar(255)	Keystone	3
num_scored	total number of students scored	int	Keystone	3
p_advanced	percentage of students that scored in advanced range	int	Keystone	3
p_proficient	percentage of students that scored in proficient range	int	Keystone	3
p_basic	percentage of students that scored in basic range	int	Keystone	3
p_below_basic	percentage of students that scored in below basic range	int	Keystone	3

year	year of exam	int NOT NULL	PSSA	4
school	school where exam data is from	int NOT NULL	PSSA	4
subject	subject of exam	varchar(255)	PSSA	4
group	All students or historically underperforming	varchar(255)	PSSA	4
grade	grade number of those taking exam	varchar(255)	PSSA	4
num_scored	total number of students scored	int NOT NULL	PSSA	4
p_advanced	percentage of students that scored in advanced range	int	PSSA	4
p_proficient	percentage of students that scored in proficient range	int	PSSA	4
p_basic	percentage of students that scored in basic range	int	PSSA	4
p_below_basic	percentage of students that scored in below basic range	int	PSSA	4
adm	Average Daily Membership - all resident pupils that district is responsible for	double	District_Fiscal	5

Data Provenance

Source Identifier	Tables	Link to original
1: General Fund Budget (GFB) Data Pennsylvania Department of Education > Teachers & Administrators > School Finances > Finances > General Fund Budget (GFB) Data	District_Fiscal School_Fiscal	https://www.education.pa.gov/Teachers%20-%20Administrators/School%20Finances/Finances/GFBData/Pages/default.aspx
2: Future Ready PA Index Data Files	District_Fiscal School_Fiscal District School	https://futurereadypa.org/Home/DataFiles
3: Keystone Exams Pennsylvania Department of Education > Data and Reporting > Assessments > Keystone Exams Results	Keystone	https://www.education.pa.gov/DataAndReporting/Assessments/Pages/Keystone-Exams-Results.aspx
4: PSSA Results Pennsylvania Department of Education > Data and Reporting > Assessments > PSSA Results	PSSA	https://www.education.pa.gov/DataAndReporting/Assessments/Pages/PSSA-Results.aspx
5: Average Daily Memberships Pennsylvania Department of Education > Teachers & Administrators > School Finances > Finances > Financial Data Elements	District_Fiscal School_Fiscal	https://www.education.pa.gov/Teachers%20-%20Administrators/School%20Finances/Finances/FinancialDataElements/Pages/default.aspx

Conclusion

Our research question was:

How can we best display pertinent data to policymakers so that they can analyze the impact of various factors on educational success?

Our goal was to give policymakers and educational administration as much freedom as possible to compare schools and districts. To accomplish this, we wanted to incorporate as much of the provided data as possible into our database. Given the state of the data that we had, this goal proved to be difficult to achieve, and we ended up having to make cuts to our final database plan, as we were unable to successfully clean all of the provided data in a timely manner. The tables detailing school enrollment, low income statistics, graduation rates, and dropout rates were all cut from the final version of our database. This data would have been useful to have, but even without it we provided a clear tool to compare several important aspects of educational success.

Our research question was not answered fully, but we provided a very basic response to this problem. We were able to create a broad tool with extremely customizable visualizations, but in doing so we allowed for many of these visualizations to be fundamentally useless. Giving the user freedom to see what they want to see is great, but we needed to focus more on helping the user find the actual relevant information, and could have provided a cleaner tool for doing this. If we were to approach this problem again, we would still focus on allowing heavily customizable graphs, but we would put more emphasis on pushing the user towards creating relevant visualizations. We could do this by providing sample graphs, as well as cutting down on the options that the users can select when they don't make sense in the context of previous choices.

Next Steps

More Data

We mentioned in the Research Question section that certain tables were dropped from the database due to time, if this project were to be continued we would finish cleaning this remaining data, and also look at what other data could be added. The goal of this project was to provide as much data as possible for the user, and while we mostly succeeded with this variety, there are a few important tables still missing.

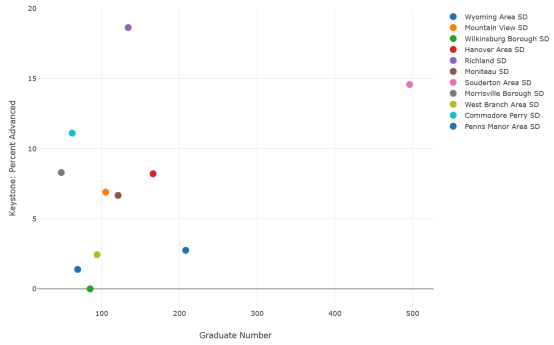
More Graph Options

We provided the user with a lot of customization in the graphs that they create, but this could have been extended further. Currently, a user can add more schools to their graphs, but can't add more things to be compared. PSSA graphs provide a clear example of this. The user can compare each individual percentile (advanced, proficient, basic, below basic), but can't compare each of these percentiles at once between schools. To include this option in the application, we could have certain X axis values that function as sets of values. The user would be able to select PSSA results by year, and see all 4 values for each school they're comparing, instead of just one at a time.

Remove Graph Options That Don't Make Sense

Currently, the user can select essentially whatever elements they want to compare in our graphing functionality. The freedom that this provides is exactly what we were aiming for, but this can go too far. Many graphs that the user can create provide 0 relevant information, or just

don’t make sense to begin with. For example, the user can compare graduate number and Keystone Percent Advanced, which gives the below graph.



This graph is entirely useless. Nothing here tells the user anything, and it doesn’t make sense for the user to even have the option to make it. This could be implemented by checking the users first selection, and narrowing their options based on this selection.

Database Indexes

If this project were to be taken further, certain tables could benefit heavily from indexes. The PSSA table specifically has over 200,000 tuples, and has queries running to it from several different attributes. The query time isn’t too bad, but could be cut down significantly with properly created indexes.