Extract, Transform and Load

Data Sources

To investigate our research question we required fiscal data measuring the revenue received by public schools annually in each state. We also needed a measure of academic performance. We obtained this data from two sources.

Common Core of Data

The Common Core of Data (CCD) is the Department of Education's database of public school data nationwide. It collected through surveys conducted annually. We used two data sets from the CCD:

- Fiscal provides details of school funding with breakdown into local/state/federal sources and expenses including teacher salaries and benefits
- Non fiscal includes total numbers of teachers and students for calculations such as pupil teacher ratios and funding per student

National Assessment of Educational Progress

The National Assessment of Educational Progress (NAEP) is the largest nationally representative and continuing assessment of student academic progress. Beginning with the 2003 assessments, national assessments are conducted every two years in reading and mathematics. Although the test was performed earlier, year 2003 was the first statewide full participation.
We chose to use the 8th Grade NAEP test scores as the data set was more complete. The NAEP test was first administered to 8th grade and then later added to 12th grade so there is a longer history of data for 8th grade.
We also considered using SAT data, but although we submitted a request for access to SAT data, we received no response. The NAEP data proved adequate for our purposes.

Architecture

Identifying which solution was best suited to our use case began with considering the following characteristics.

Data Characteristics

- Structured. The source data has a well defined structure with columns corresponding to property values and data types defined in the data layout document.
- Small size.
- Low sink latency and high source latency. Batch updates will occur annually, but we would like to be able to query the data as soon as it is imported.

- Medium quality. Most of the data quality issues are resolved in the transform scripts. Some filtering by constraints will be desirable.
- Complete data.

Processing Dimensions

- High to low selectivity. Some of our queries are highly selective and others are not.
- Short query time. We expect sub-second responses to our queries.
- Short processing time for batch jobs.
- Medium aggregation allowing us to aggregate over multiple dimensions.
- Advanced joins across multiple tables
- Exact precision handling numeric data

Other Characteristics

- Low cost. Open source preferred.
- Simple to configure and maintain.

Preferred Solution

These characteristics matched most closely with a relational database. Our desire for a low cost open source solution placed PostgreSQL at the top of our list. It is relatively easy to administer, stable and well maintained, and supports all the standard SQL constructs as well as some of the more advanced features. It also integrates with R, Tableau and other solutions we were considering as part of our implementation.

Data Scaling

Our data sets are small with the potential to grow over time. It is expected that the educational statistical data we use will continue to be generated every year for the foreseeable future. The annual increments impact the number of rows in the tables, but do not greatly increase storage needs. Horizontal scalability would not be a requirement. We were able to store all of our database files locally, reducing the network overhead and cost of external storage solutions. We chose to implement partitioning within the Postgres database to help ensure that query performance remained fast as the number of rows increased over time. We also focused on appropriate indexing to ensure fast performance of our queries.

Cloud

We chose to implement our solution in an Amazon Web Services Linux image for ease of collaboration. We also used a github repository to share the solution implementation.

Schema

We are using third normal form schema on write with row based tables. Constraints are used to validate the data before persisting. The combination of state and year provides a common key to relate the tables. Indexes are defined for faster access. Partitioning is implemented for scalability.

Tables and Columns

Our Entity Relationship Diagram (included in our github repo) identified three classes of data - fiscal, nonfiscal and scores, where both fiscal and nonfiscal data influence the scores. For each table the year/state uniquely identifies the entity and provides a link to corresponding data in other tables.

Fiscal data

survey year	year
state	character
state revenue	numeric
local revenue	numeric
federal revenue	numeric
total revenue	numeric
teacher salaries	numeric
teacher benefits	numeric
current expenditures	numeric

Nonfiscal data

survey year	year
state	character
total teachers	numeric
grade 8 students	numeric
grade 12 students	numeric
total students	numeric

NAEP scores grade 8

test year	year
state	character
mathematics score	numeric
reading score	numeric

In addition, these values would allow us to perform queries to calculate the following:

- Total revenue per pupil as total revenue/total students
- Teacher compensation as (teacher salaries+teacher benefits)/total teachers

- Student teacher ratio as total students/total teachers

PostgreSQL has a remarkably enlightened approach to handling numeric data where it does not require the scale and precision to be defined in advance and will adapt based on the data input.

Constraints

In addition to “not null” modifiers, check constraints have been used to validate important values during import, including the following

- total revenue > 0
- total teachers > 0
- total students > 0
- mathematics and reading scores must be between 0 and 500

Furthermore, the insert trigger checks the state value and will only insert rows which match defined states.
We also defined a domain for the year data type with a constraint which checks that the value for year is valid.

Indexes

On each table, the combination of year and state has been defined as the primary key. This key relates the tables. It is also used in most of our queries and is therefore valuable for performance reasons.

Partitioning

We have divided the tables into five state range based partitions. Since most of our queries gather data per state, we implemented 5 range partitions with the states divided evenly.
Using state for the partition range also means that new years of data can be added without repartitioning.
The insert trigger function checks the value for state, and inserts the row into the appropriate partition table. If the state does not match a partition, a notice is issued and the row is skipped. This helps us ensure that only data from states common to all the data sets are imported.

Extract, Transform and Load

The first step was to look for existing tools which could be used to import the data into PostgreSQL.

- Unlike some of the more expensive commercial database products, Postgres does not provide full featured ETL tools.
- An open source tool called pgloader initially looked promising. It was able to download files from a website and import the csv contents into database tables. Upon closer examination however, it provided no functionality for extracting specific columns from a csv file.
- The PostgreSQL COPY command is generally considered to perform much better than

other alternatives (including pandas for instance).

We decided to write our own script to extract and transform the data and then use a sql script to create the schema and populate with data using the COPY command.

Download

The script downloads zip files containing the data from the CCD website. There is one fiscal and one nonfiscal zipped data file for each year.

The filenames of the data sets do not use a consistent naming pattern. An earlier version of the script attempted to use pattern matching to retrieve the filenames, but due to website redirects the script was attempting to access files not intended. As a safer approach, the script now uses text files containing the URLs of the data sets to be downloaded. Each year as new data is added to the CCD website, the text files can be updated to add the URLs of the new files.

Each file is downloaded to a temporary file, unzipped and extracted then deleted before the next file is retrieved. With this design we only need to store the extracted content of the files.

Extract and Transform

The most recent fiscal and nonfiscal data files are in tab delimited format with a header row. The extraction process reads each row of the file and parses the rows to output just the required columns into a comma delimited csv file.

We implemented the script to overcome the following challenges:

- the tab delimited data files contained null values
- some field values contained commas
- the format and columns of the nonfiscal data file changed. From 2003 to 2006 the files were in fixed width format. In 2007 the format changed to tab delimited. Starting in 2008 additional columns of demographic data were added to the files.
- some files contained data for extra territories. The CCD include 56 states and territories including District of Columbia, American Samoa, Guam, Northern Mariana Islands, Puerto Rico, Virgin Islands. Some earlier years of nonfiscal data included DoD school information while later years did not. The NAEP data includes 50 states plus District of Columbia and DoDEA (Department of Defense Education Activity military schools). We chose to use the overlapping data from the 50 states and District of Columbia in our analysis.

OpenRefine

The NAEP website provides an interactive form to generate csv file data which precludes the possibility of automating the download. The data file layout is designed for aesthetics not analysis:

- multiple header rows
- blank columns
- year values are provided in the first row of each table but not populated in subsequent rows

Fortunately the data for all years is included in a single file. This made it easy to quickly process the file in OpenRefine.The empty rows and columns were removed. The fill down tool was useful for populating the year value in rows where it was missing. The data was exported as a csv file.

Load

The SQL script which creates the database schema also loads the data from csv files into the tables using the COPY command. The data is checked by constraints on fiscal and state values.

Execution

1. Run ./extract_transform.sh

This process retrieves data files from the National Center for Educational Statistics website based on URLs recorded in the fiscal.txt and nonfiscal*.txt text files in the same directory. The output of the script is two csv files: fiscal.csv and nonfiscal.csv

While it is running, the extract_transform.sh script displays output as below.

```
[w205@ip-172-31-22-103 ETL]$ ./extract_transform.sh
--2017-08-15 05:12:21--  https://nces.ed.gov/ccd/data/zip/stfis14_1a.zip
Resolving nces.ed.gov... 63.145.228.23, 2001:428:7003:11::23
Connecting to nces.ed.gov|63.145.228.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 33374 (33K) [application/x-zip-compressed]
Saving to: `temp.zip'

100%[======================================================>] 33,374      44.6K/s   in 0.7s

2017-08-15 05:12:22 (44.6 KB/s) - `temp.zip' saved [33374/33374]

51 fiscal.csv
--2017-08-15 05:12:22--  https://nces.ed.gov/ccd/data/zip/stfis13_1a.zip
Resolving nces.ed.gov... 63.145.228.23, 2001:428:7003:11::23
Connecting to nces.ed.gov|63.145.228.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34621 (34K) [application/x-zip-compressed]
Saving to: `temp.zip'

100%[======================================================>] 34,621      93.2K/s   in 0.4s

2017-08-15 05:12:23 (93.2 KB/s) - `temp.zip' saved [34621/34621]

102 fiscal.csv
--2017-08-15 05:12:23--  https://nces.ed.gov/ccd/data/zip/Stfis_1a_txt.zip
Resolving nces.ed.gov... 63.145.228.23, 2001:428:7003:11::23
Connecting to nces.ed.gov|63.145.228.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35042 (34K) [application/x-zip-compressed]
Saving to: `temp.zip'
```

2. The load.sql script is executed within PostgreSQL: \i /home/w205/205_Project/ETL/load.sql

After creating the schema, the data is loaded from the csv files created by the previous step.

While it is running the load.sql script displays output as below:

```
postgres=# \i /home/w205/205_Project/ETL/load.sql
DROP DATABASE
CREATE DATABASE
psql (8.4.20)
You are now connected to database "w205project".
CREATE DOMAIN
ALTER DOMAIN
CREATE LANGUAGE
ALTER LANGUAGE
SET
psql:/home/w205/205_Project/ETL/load.sql:42: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit
 index "fiscal_pkey" for table "fiscal"
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
```

After the scripts are completed, the tables can be shown using the \dt command.

```
w205project=# \dt
           List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | fiscal      | table | postgres
 public | fiscal_1    | table | postgres
 public | fiscal_2    | table | postgres
 public | fiscal_3    | table | postgres
 public | fiscal_4    | table | postgres
 public | fiscal_5    | table | postgres
 public | naep8       | table | postgres
 public | naep8_1     | table | postgres
 public | naep8_2     | table | postgres
 public | naep8_3     | table | postgres
 public | naep8_4     | table | postgres
 public | naep8_5     | table | postgres
 public | nonfiscal   | table | postgres
 public | nonfiscal_1 | table | postgres
 public | nonfiscal_2 | table | postgres
 public | nonfiscal_3 | table | postgres
 public | nonfiscal_4 | table | postgres
 public | nonfiscal_5 | table | postgres
(18 rows)
```

The columns, indexes, constraints and triggers for the fiscal table are displayed with \d fiscal.

```
w205project=# \d fiscal
                Table "public.fiscal"
       Column        |         Type         | Modifiers
---------------------+----------------------+-----------
 survey_year         | year                 | not null
 state               | character varying(2) | not null
 state_revenue       | numeric              | not null
 local_revenue       | numeric              | not null
 federal_revenue     | numeric              | not null
 total_revenue       | numeric              | not null
 teacher_salaries    | numeric              | not null
 teacher_benefits    | numeric              | not null
 current_expenditures | numeric              | not null
Indexes:
    "fiscal_pkey" PRIMARY KEY, btree (survey_year, state)
Check constraints:
    "fiscal_total_revenue_check" CHECK (total_revenue > 0::numeric)
Triggers:
    insert_fiscal_trigger BEFORE INSERT ON fiscal FOR EACH ROW EXECUTE PROCEDURE fiscal_insert_trigger()
```

The columns, indexes, constraints and triggers for the fiscal table are displayed with \d nonfiscal.

```
w205project=# \d nonfiscal
             Table "public.nonfiscal"
     Column      |         Type         | Modifiers
-----------------+----------------------+-----------
 survey_year     | year                 | not null
 state           | character varying(2) | not null
 total_teachers  | numeric              | not null
 grade8_students | integer              | not null
 total_students  | integer              | not null
Indexes:
    "nonfiscal_pkey" PRIMARY KEY, btree (survey_year, state)
Check constraints:
    "nonfiscal_grade8_students_check" CHECK (grade8_students > 0)
    "nonfiscal_total_students_check" CHECK (total_students > 0)
    "nonfiscal_total_teachers_check" CHECK (total_teachers > 0::numeric)
Triggers:
    insert_nonfiscal_trigger BEFORE INSERT ON nonfiscal FOR EACH ROW EXECUTE PROCEDURE nonfiscal_insert_trigger()
```

The columns, indexes, constraints and triggers for the fiscal table are displayed with \d naep8.

```
w205project=# \d naep8
               Table "public.naep8"
    Column     |         Type         | Modifiers
---------------+----------------------+-----------
 test_year     | year                 | not null
 state         | character varying(2) | not null
 math_score    | numeric              |
 reading_score | numeric              |
Indexes:
    "naep8_pkey" PRIMARY KEY, btree (test_year, state)
Check constraints:
    "naep8_math_score_check" CHECK (math_score >= 0::numeric AND math_score <= 500::numeric)
    "naep8_reading_score_check" CHECK (reading_score >= 0::numeric AND reading_score <= 500::numeric)
Triggers:
    insert_naep8_trigger BEFORE INSERT ON naep8 FOR EACH ROW EXECUTE PROCEDURE naep8_insert_trigger()
```

After the data is loaded the database is small, justifying the storage choices we made.

```
w205project=# select pg_size_pretty(pg_database_size('w205project')) as size, pg_database_size('w205project') as bytes;
  size   |  bytes
---------+---------
 6294 kB | 6444816
(1 row)
```