Project Phase 2

dre4mte4m

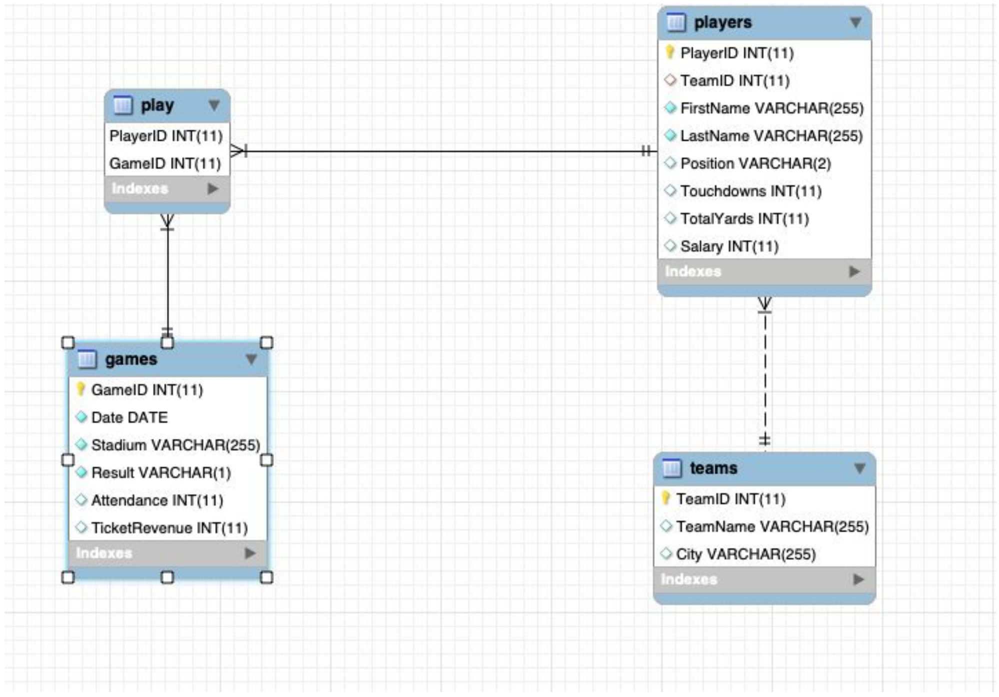
Cyrus Baker
Amin Chamsaz
Kay Sweebe
Jerrick Waugh

Introduction

Creating user interfaces (UIs) that are user-friendly can be a challenge when the data being handled is very large causing less than par performance. Data can be inserted, read, updated, and deleted from a database. SQL provides insertion techniques for adding a single element to a database and also provides insertion techniques for adding a set of elements to a database. Data insertion itself can have a profound impact on performance. The single insertion technique inserts one element into the database for each call. The multiple row insertion technique and the load data insertion technique both are used for adding sets of data at a time. Knowing the user requirements for a UI as well as the performance of the backend implementation is vital for creating a worthy UI.

Method

For this project, an NFL database of football teams, players, games, and plays was instantiated. The entity-relation (ER) diagram for our database is shown below:



To implement the UI, we chose to use Jupyter Notebook and ipywidgets. Jupyter Notebook is a platform that allows users to share documents that contain live code. Many types of users exist for Jupyter. Jupyter's power users come from fields like data science, education, healthcare, and economics. We chose to connect to a local instance of SQL through a Jupyter Notebook. From within the notebook, we created a widget that can:

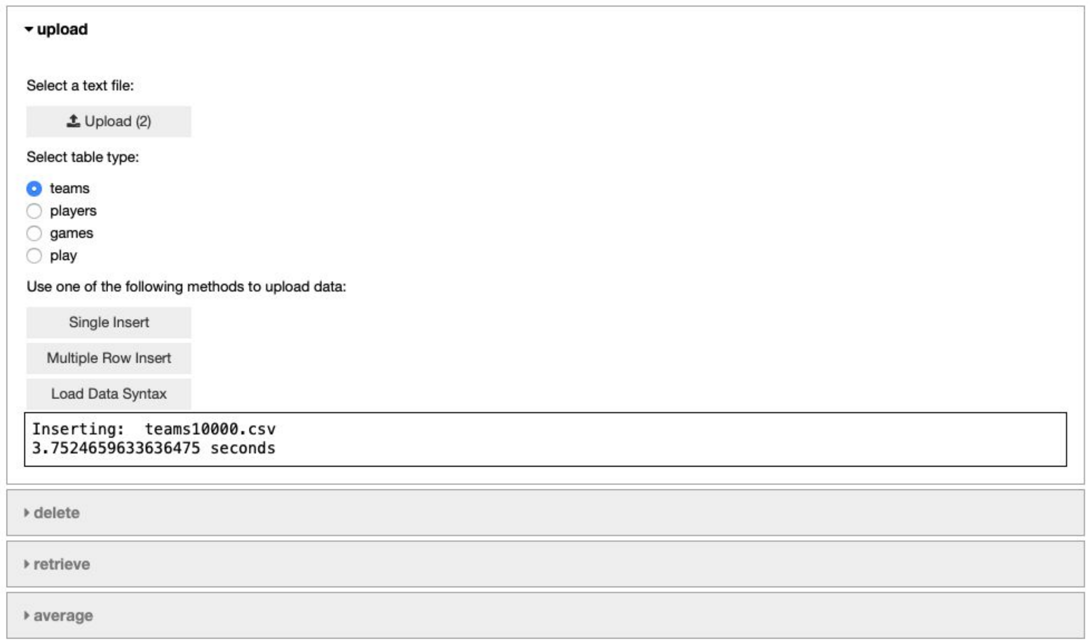
- Let a user select a csv file containing table data
- Upload the data into the database using either single insertion, multiple row insertion, or bulk load data insertion
- Delete table data
- Retrieve table data
- Find the average value for an integer column within a table

The widget uses an accordion-style approach to section off the four basic functionalities of uploading, deleting, retrieving, and averaging information. Each of these sections is called a tab. Any time a user does an action, an output widget is used to communicate back to the user a message.

We used time functions wrapped around the insertion calls. We inserted table data containing 10,000, 100,000, and 1,000,000 rows. We stored the values in a list and then used matplotlib to create the graphs for the results. We plotted each graph by table and depicted the three different insertion techniques against time. The amount of columns for each table produces a slightly different speed for each table. The teams table has 3 columns, the players table has 8, the games table has 6, and the play table has 2.

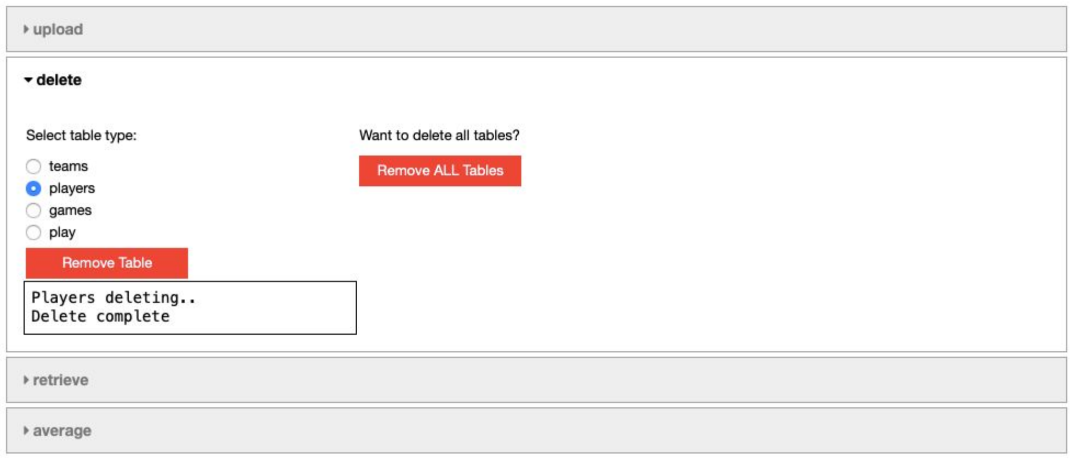
User Interface

Upload Tab



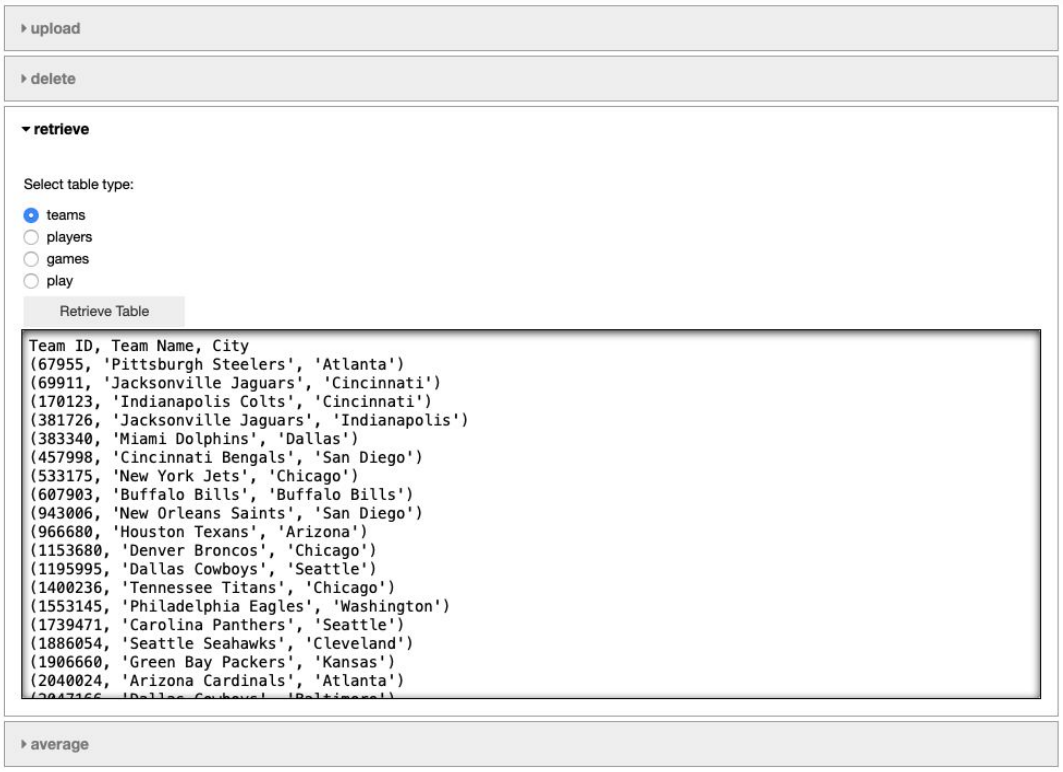
```
Inserting:  teams10000.csv
3.7524659633636475 seconds
```

Delete Tab



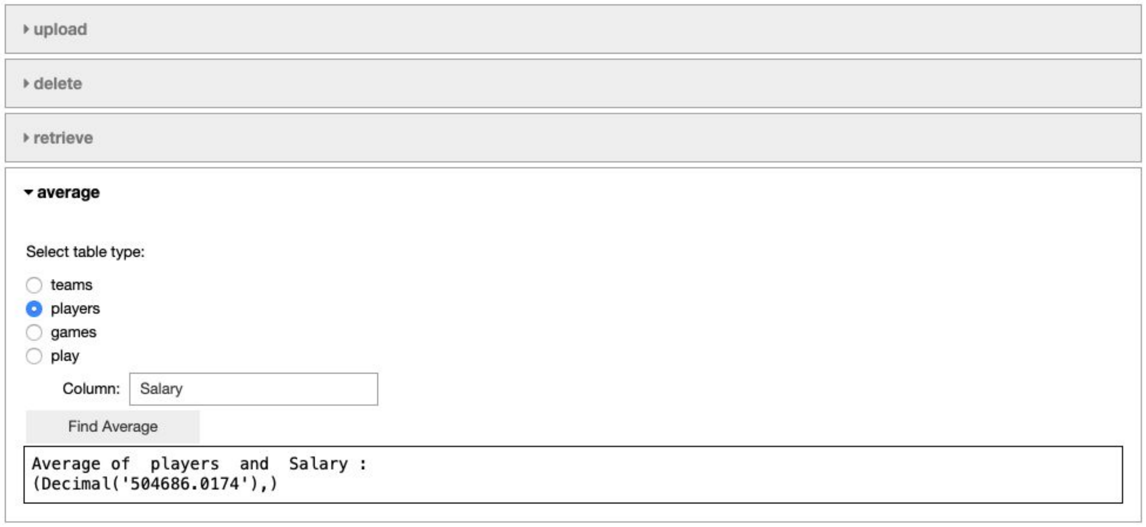
```
Players deleting..
Delete complete
```

Retrieve Tab



```
Team ID, Team Name, City
(67955, 'Pittsburgh Steelers', 'Atlanta')
(69911, 'Jacksonville Jaguars', 'Cincinnati')
(170123, 'Indianapolis Colts', 'Cincinnati')
(381726, 'Jacksonville Jaguars', 'Indianapolis')
(383340, 'Miami Dolphins', 'Dallas')
(457998, 'Cincinnati Bengals', 'San Diego')
(533175, 'New York Jets', 'Chicago')
(607903, 'Buffalo Bills', 'Buffalo Bills')
(943006, 'New Orleans Saints', 'San Diego')
(966680, 'Houston Texans', 'Arizona')
(1153680, 'Denver Broncos', 'Chicago')
(1195995, 'Dallas Cowboys', 'Seattle')
(1400236, 'Tennessee Titans', 'Chicago')
(1553145, 'Philadelphia Eagles', 'Washington')
(1739471, 'Carolina Panthers', 'Seattle')
(1886054, 'Seattle Seahawks', 'Cleveland')
(1906660, 'Green Bay Packers', 'Kansas')
(2040024, 'Arizona Cardinals', 'Atlanta')
```

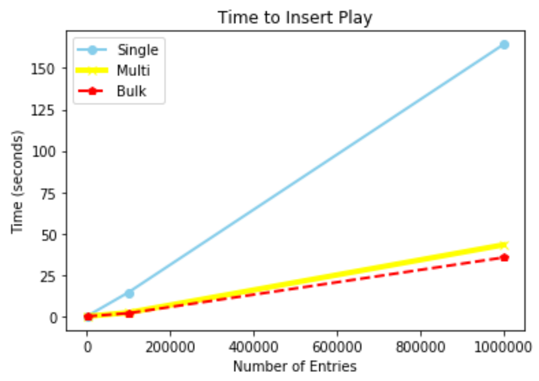
Average Tab

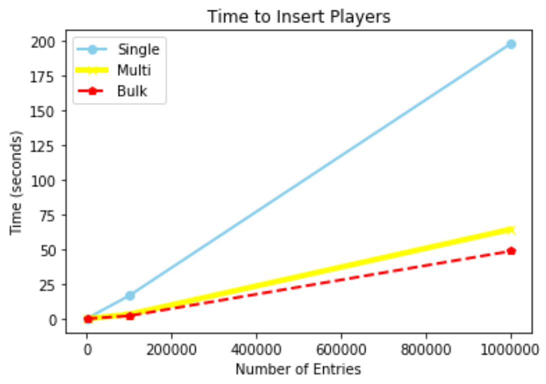


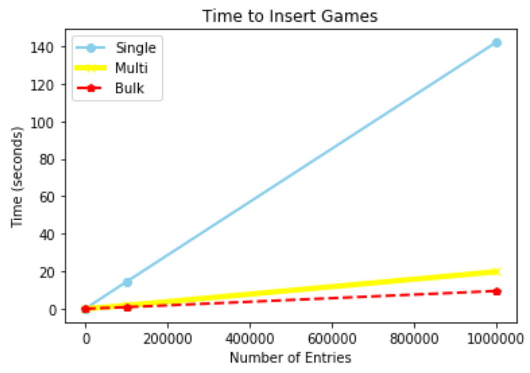
Column: Salary

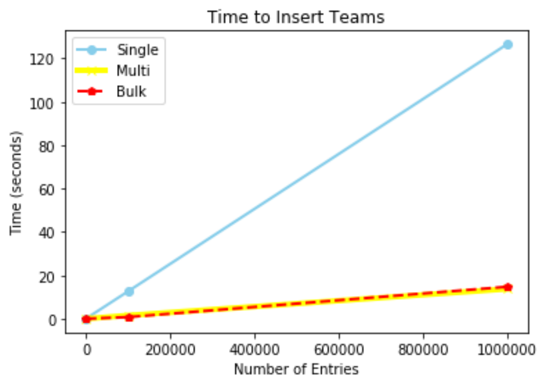
```
Average of  players  and  Salary :
(Decimal('504686.0174'),)
```

Results









Analysis

By looking at the graphs below, we see that for all tables single insert is much slower than the other two inserts. The multiple row is mainly slower than the bulk load with the exception of the teams table. If we look at the trending difference between multiple row and bulk load given column sizes, our results indicate that with more columns, bulk load becomes even faster than multiple row insertion.

Discussion

With the results given, it grants the question as to why we would ever use single insert. In a UI, single insert is beneficial if a user wants to insert a single row of data into a table. For databases that are being updated with a new object at a time, single insert is a simple function that provides the necessary functionality. For databases where large amounts of data might be input at once, using a multiple row or even better, a bulk load, is much faster granting the consideration of its usage within a given UI. With more and more columns within the table data, bulk load is the best option of the three.