Homework 10 of 10: Machine Problem (MP10) / Project Report Scalability, Concurrency, Performance Measurement

Project Team
| Cole Cooper | Blake Koblitz | Dane Miller |

The five database management system requirements we implemented are parallel transactions, basic or materialized views, triggers and integrity constraints, GUI-driven database tools (PHPMyAdmin), and NoSQL (MemcacheDB). The four GUI requirements we implemented are the dynamic data browser, record editor, data import/export tool, and visualization module. In the following paragraphs, we will explain what these nine requirements are, how we implemented them, and how they look. We will also include code snippets and screenshots to show our program in action.

The first of the database requirements we will talk about are parallel transactions. Parallel transactions allow us to have multiple users adding records or importing data files simultaneously, without messing up other people's work. Parallel transactions work in our project because we are leveraging MySQL's InnoDB database engine. When multiple transactions are sent to our database simultaneously, InnoDB will internally queue those transactions with its scheduling engine to ensure that they happen sequentially and do not overwrite each other; this results in fast, yet ACID-compliant writes to the database. This has been tested by having multiple people upload data files with our import script at the same time.

The second database requirement we implemented is a basic or materialized view. We have a couple of basic views that allow us to easily call up a few complex SQL statements that have lots of JOINs and complex WHERE clauses. They are primarily used in the getDataControls.php file. We have created 2 basic views in our database, via the command line. This gets all measurements and measurement IDs:

```
CREATE OR REPLACE VIEW agcom_get_m AS SELECT agcom_commodities.mid,
agcom_commodities.type, measurement FROM agcom_measurement JOIN
agcom_commodities WHERE agcom_commodities.mid=agcom_measurement.mid;
```

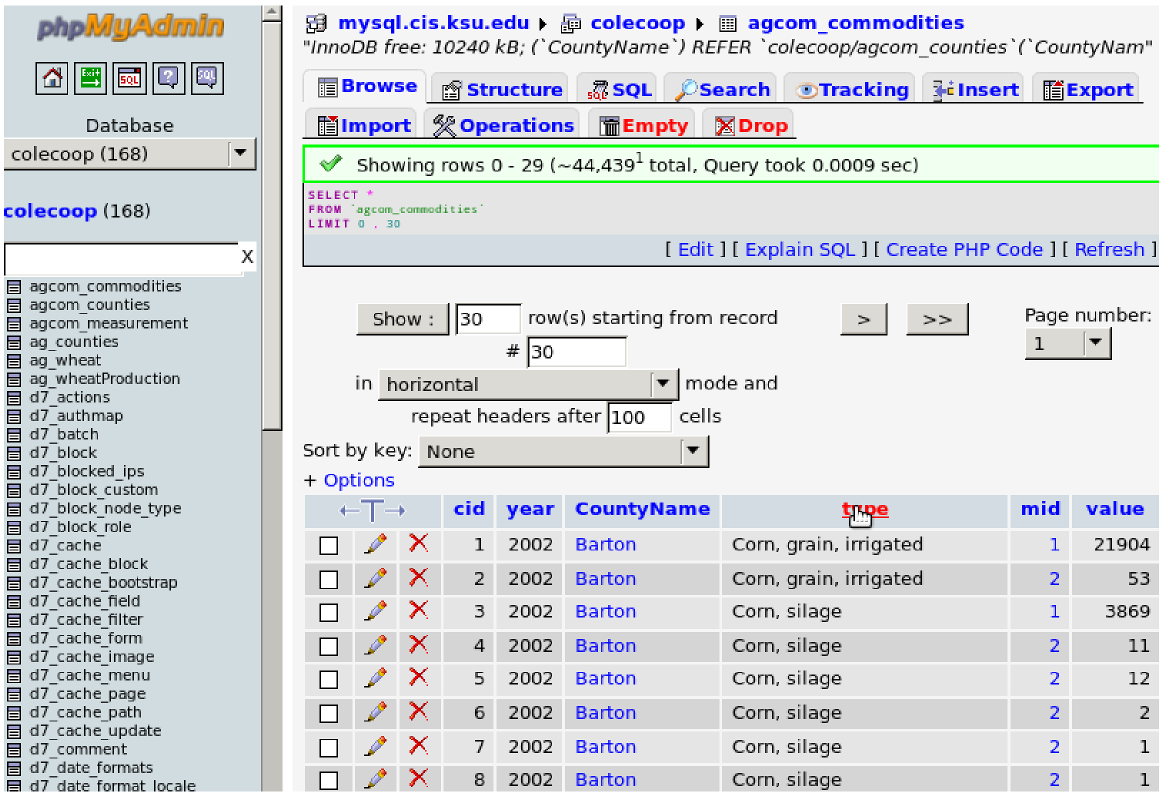
This gets all years for a commodity:

```
CREATE OR REPLACE VIEW agcom_get_y AS SELECT year, type FROM
agcom_commodities;
```

This is implemented in our project file getDataControls.php, and can be viewed there.

Our third database requirement includes triggers and integrity constraints. When a file is uploaded or a record is added to the database, the program checks the unit of measurement for the commodity being entered, and compares it to the existing list of unit types. When a commodity is selected by the user on the index page, the drop-down menus for "year" and unit measurement are updated to only include options applicable to that commodity. This is implemented in our project files importFileSQL.php and addRecordSQL.php, and can be viewed there.

The fourth database requirement we implemented includes the GUI-driven database tool, PHPMyAdmin. We use the CIS Department's instance of PHPMyAdmin to help manage our database, along with using the normal MySQL command line. This aids us as it helps easily create and configure tables and change settings without having to know many of the commands. PHPMyAdmin is a web application written in PHP that is used to manage MySQL databases. The CIS Department installed and configured the application already, and it is available at: http://phpmyadmin.cis.ksu.edu/. Below is a sample screenshot of our PHPMyAdmin client with the agcom_commodities table included:



"InnoDB free: 10240 kB; (`CountyName`) REFER `colecoop/agcom_counties`(`CountyNam"

Showing rows 0 - 29 (~44,439[1] total, Query took 0.0009 sec)

```
SELECT *
FROM `agcom_commodities`
LIMIT 0 , 30
```

Show : 30 row(s) starting from record # 30 > >>

in horizontal mode and

repeat headers after 100 cells

Page number: 1

Sort by key: None

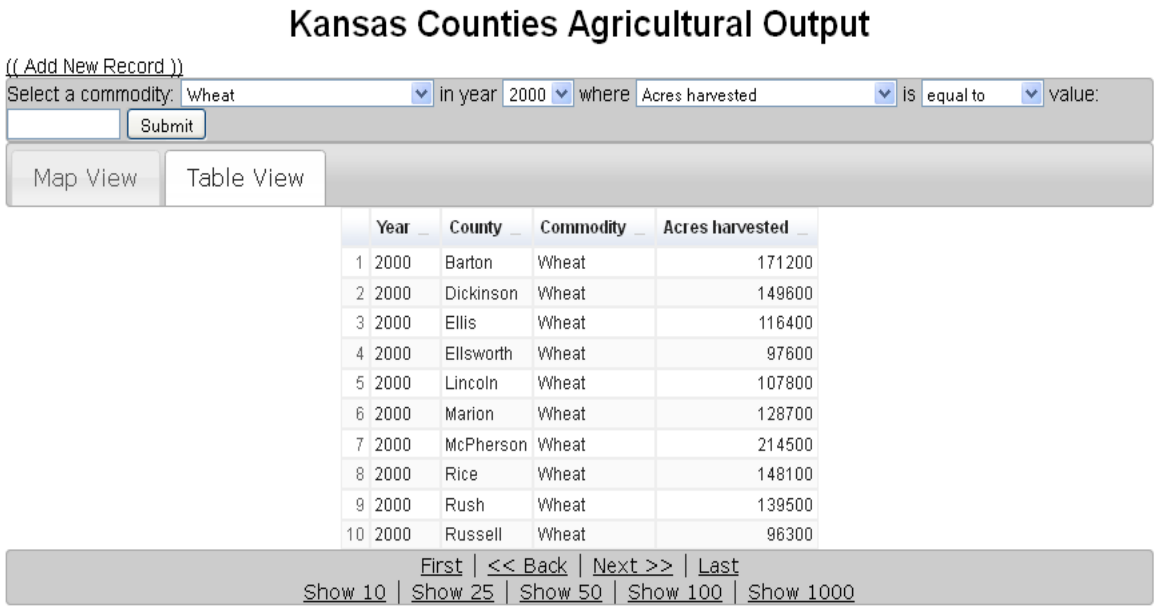
			cid	year	CountyName	type	mid	value
☐	✎	✗	1	2002	Barton	Corn, grain, irrigated	1	21904
☐	✎	✗	2	2002	Barton	Corn, grain, irrigated	2	53
☐	✎	✗	3	2002	Barton	Corn, silage	1	3869
☐	✎	✗	4	2002	Barton	Corn, silage	2	11
☐	✎	✗	5	2002	Barton	Corn, silage	2	12
☐	✎	✗	6	2002	Barton	Corn, silage	2	2
☐	✎	✗	7	2002	Barton	Corn, silage	2	1
☐	✎	✗	8	2002	Barton	Corn, silage	2	1

The fifth and final database requirement we will talk about is an implementation of NoSQL called MemcacheDB. MemcacheDB is a NoSQL database originally written for the site LiveJournal. It is an in-memory cache that stores key-value pairs. This allows for quick access, and can be used to alleviate load on our database server. Memcache works through a network protocol, in which our PHP script will connect to the memcache service, and query to see whether or not a particular key has been set to a value. If it has, we will pull that value out and return it, rather than looking it up in our MySQL database. If the result does not yet exist in the memcache service, we will query our database, get the result, and then store it into memcache for future reference. Our code that implements this is in three files: getDataControls.php, getDataGeochartJSON.php, and getDataTableJSON.php.

The first GUI requirement we will talk about is the dynamic data browser. In our project, this was done in the form of a table view. It displays an ordered list of the counties' production values for a given year and implements features that allow it to show a limited selection of data, and reorder the displayed values alphabetically according to county name, or by the total output values for that year. This works by using Javascript to call the getDataTableJSON.php, which in turn pulls the information from the database.

This feature currently has a CSS bug that causes the table width to display incorrectly the first time it is viewed after a new data set is displayed. We are planning to find a fix for this in later versions, but it is a minor issue for now.

Included below is a screenshot of our Dynamic Data Browser tab (the Table):


Kansas Counties Agricultural Output

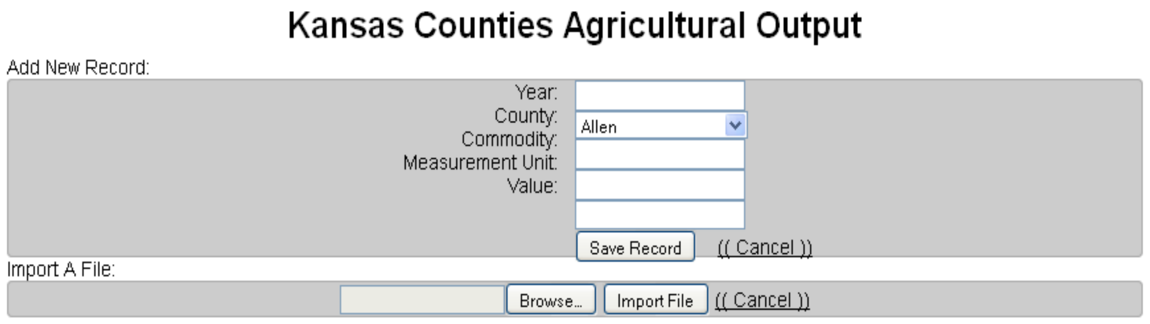
((Add New Record))

Select a commodity: Wheat in year 2000 where Acres harvested is equal to value:

	Year	County	Commodity	Acres harvested
1	2000	Barton	Wheat	171200
2	2000	Dickinson	Wheat	149600
3	2000	Ellis	Wheat	116400
4	2000	Ellsworth	Wheat	97600
5	2000	Lincoln	Wheat	107800
6	2000	Marion	Wheat	128700
7	2000	McPherson	Wheat	214500
8	2000	Rice	Wheat	148100
9	2000	Rush	Wheat	139500
10	2000	Russell	Wheat	96300

First | << Back | Next >> | Last
Show 10 | Show 25 | Show 50 | Show 100 | Show 1000

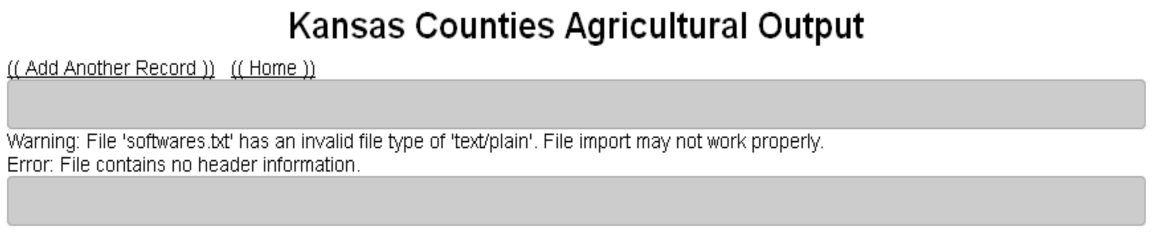
The second GUI requirement is the record editor. This was accomplished in the form of an add new record portion of our project. This record editor is a page that includes options to add individual values for counties. This page call the addSQLrecord.php file to add the new record to the database. This file includes code to add the values the user submitted to our PHPMyAdmin database. When someone successfully adds a new record to a database, that person will be taken to a new page and told their record was added successfully to the database. This page includes an add new record button and text-boxes for year, county, commodity, measurement, and value. It also includes a cancel button to return to the map and table view. Included below is a screenshot of our record editor and import tool:


Kansas Counties Agricultural Output

Add New Record:

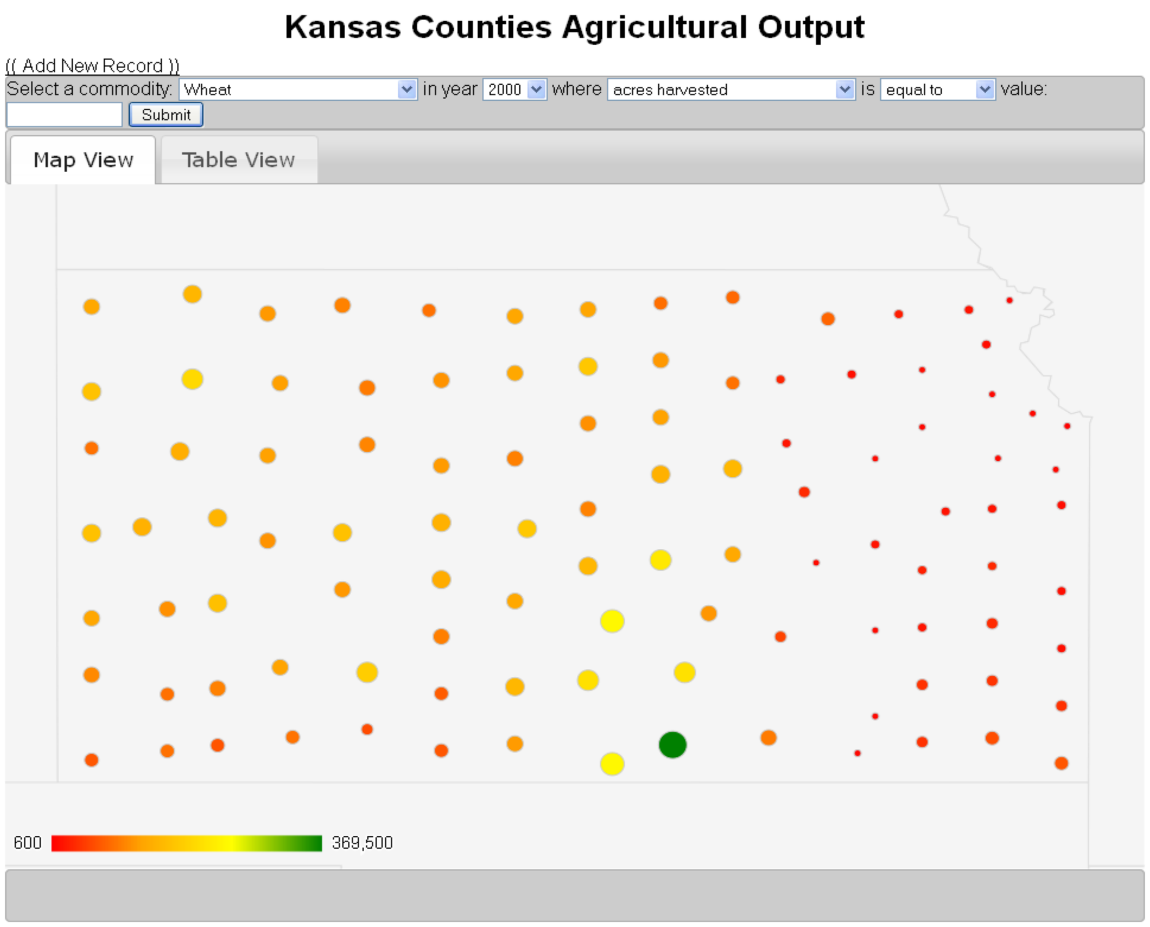
Year:	
County:	Allen
Commodity:	
Measurement Unit:	
Value:	

The third GUI requirement we completed is a data import/export tool. Our particular form of this tool is a file that can import multiple lines of information to the database at once. This implementation works by browsing for a file and then submitting the file into the database. The form submit action button calls importFileSQL.php to take in the input file name and begin processing it. If the the file-path is blank, the operation informs the user of this, and ends without performing any further work. If the filetype is not a CSV, it warns the user that the file may not be imported properly, but does not give them an option to cancel. We felt that this would be a hassle if the user was attempting to import a large number of TXT or similarly formatted files. If the file is readable, the importFileSQL.php will attempt to read the first line. if the headers do not match its standard format, it will cancel the upload and give the error message “Error: File contains no header information.” If the headers are correct, it will begin reading in the new data, adding a new unit for output measurement if it does not already exist in the database. If an error occurs at any point during the import process, it will roll back the invalid entry, but preserve everything that was imported successfully up to that point. When someone successfully imports a file, they will be taken to a new page that shows what values were added to the database. This implementation includes a browse button to find the file and an import file button to start the import of lines into the database. It also includes a cancel button returns to map and table view. The import/export tool is on the same page as our record editor, and is included in the screenshot above.


Kansas Counties Agricultural Output

Warning: File 'softwares.txt' has an invalid file type of 'text/plain'. File import may not work properly.
Error: File contains no header information.

The fourth and final GUI requirement we implemented is the visualization module. This was accomplished in our project by using Geochart. This visualization is a Javascript display with tabs to switch between a thematic map and table view. This implementation works by calling in latitude and longitudes of the Kansas counties from a local file (counties.csv), rather than relying on an external database to pull in the information every time different information display is selected. This visualization displays a thematic map of the different counties. It also includes different colored dots and values for each county. In addition, a pop-up screen is initiated when hovering over a county’s dot and shows the county name, value, and measurement of the commodity. It also has a tabbed view to allow efficient switching between the GeoChart and the dynamic data browser which we have previously talked about. Included below is a screenshot of our Geochart visualization tab:



Kansas Counties Agricultural Output

((Add New Record))

Select a commodity: Wheat in year 2000 where acres harvested is equal to value:

Finally, by following a Model-View-Controller design pattern, our project is implemented in such a way that it can easily be scaled to handle much larger amounts of data, and by keeping our views separate from controllers, we have allowed for extra features and functionality to be added by new administrators with minimum hassle.