Project 1: The transactional part of the shopping application

SUBMIT before: Monday, May 1, 9PM (tentative)

The “Signup” page

In this page a user declares

- the unique name he would like to have, in a textbox
- his role (owner or customer), in a dropdown menu
- his age, in a textbox
- his state, in a dropdown menu. The states may be displayed as “CA”, “NV’, etc or as full name “California”, “Nevada”, etc. Be consistent across all pages on how you display.

All of the above data are necessary for a successful signup. The database should have corresponding constraints requiring unique name, role, age and state.

Upon a successful signup, the user should reach a page that simply declares “You have successfully signed up”. If any of the data requirements is violated the next displayed page should declare “Your signup failed” (we refer to this as “signup failure”). Providing a reason for the failure (e.g., “name was not provided”) is optional[1] but welcome.

Notes:

1. In the event that a user asks for a name that is already taken by another user, there will be a signup failure. If a user happens to pick the same name with a prior user, then the application’s response will be signup failure. Be alert of the possibility that two users may be attempting to register at the exact same time and then concurrency-related bugs may happen if you are not careful with your database access code.
2. For Project 1 simplicity, your web application code does not need to return to the signup page in the case that the data provided in the forms violate one or more of the constraints that we have assumed. It is sufficient to reach the “signup failure” page. For example, if the user does not provide age, a “signup failure” will happen. (In practice it makes more sense to stay on the signup page, indicate to the user that he has not provided age and ask him to provide the age and resubmit. But we neglect it here.)
3. For Project 1 simplicity we require no passwords and no verification process. In the rest of this project some of the pages are supposed to be accessible by owners only. We do not require any check that the currently logged-in user has the right to see the requested page.
4. We will assume that when you place an “R” marker on an E/R attribute it means that the attribute is not null and it is not an empty string (assuming it is a string). However, your

[1] “Optional” means that you do not lose points for not providing it.

database constraints may also need a database CHECK constraint to prevent placing an empty string in string attributes that should not have empty strings.

The “Login” page

A user’s session with the application always starts from the “Login page”. In the “Login” page the user provides his name. The name is stored in the session. Consequently, all pages that this user visits in the session will have a “Hello <user name>” at the top. If the provided user name is not the name of a signed-up user the “Login” page is displayed again also showing a message “The provided name <provided name> is not known” and asking the user to complete the “name” textbox. Upon provision of the name of a signed up user, the next page to be shown is the “Home” page.

Notes:

- You are not required and you are not expected to write Javascript/Ajax code in order to handle the case where the user name is not provided and/or is not the name of a signed-up user. (Javascript/Ajax assignments come later.) However, if you are already comfortable with Javascript/Ajax and you want to handle this case with Javascript/Ajax, you are free to do so.
- No password required. No “log out” is required.

Any of the following pages become available only after the user has logged in. If the user has not logged in and attempts to access any of the pages, she gets a message “No user logged in”.

The “Home” page

The “Home” page provides links to the other pages, listed below. The set of provided links depends on whether the user is an owner or a customer. The owners have access to all the pages that customers have access but not vice versa.

The “Categories” page

At the “Categories” page the owners view and create product categories. If the accessing user is not an owner, the page should only display that “this page is available to owners only”. The page looks like the “students” page of the class example, in the sense that it displays the name and description of each category in text box and text area respectively, and offers “Insert”, “Delete” and “Update” buttons. Each category has a required unique name (collected by a text box) and a required description (collected by a small text area). The owners can insert, delete and update categories.

The Categories page also offers the links that the Home page offers.

Notes:

5. Any of the data modifications (insert/delete/update) enabled by this page may lead to a violation of a constraint on category data. For example, one may attempt to update the name of a category by placing a null or an empty string in it. Any data modification leading to a constraint violation should be prevented, the “Categories” page should be redisplayed stating at the top that the requested data modification failure (referred to as “data modification failure”). It is

optional but welcome to explain the reason of the failure. (Again, more user-friendly form interaction and verification techniques are not necessary at Project 1.)

6. A product category cannot be deleted if there are already products belonging to it. In particular, there should be no “Delete” button next to a product category if this category already has at least one product.
7. You may make either one of the following two assumptions regarding who has the right to delete and update product information. You should design your database according to the assumption you choose
    - Assumption A: An owner can delete and/or update every category, regardless of whether she was the one who created the category.
    - Assumption B: An owner can delete/update only categories that she created.
8. Be careful of concurrency related bugs where a condition/button that was applicable when the Categories page was displayed is no more applicable at the time the button is clicked. For example, it is possible that the owner *A* sees a “Delete” button next to product category *c*, because no product refers to *c*. While the owner *A* still stares at the page, the owner *B* creates a product *p* that refers to *c*. Then *A* clicks to delete *c*, since he still sees the “Delete” button. However, the category *c* is not deletable any more, since products refer to it.

   You will see that this kind of issue can occur in many more cases. Generally, the pattern is as follows: The page allows the user to perform some database modification statement (insert/delete/update) *x* but by the time the user clicks to perform *x*, the situation has changed (due to an action of another user) and performing *x* should not be allowed any more.

   Issues of this kind are solved in one of the following two methods. You may choose either of the two but the second will cost you a point:
    - [Preferred method] Just before issuing the database modification statement *x* the application code checks whether *x* is still applicable. If the action is not applicable any more, the page will report data modification failure.
        - Notice the needed use of transactions, which was discussed in lecture: Without transactions it is possible that the application code first finds that x is still applicable but in the milliseconds between checking and performing the modification, another owner has inserted a product and the category is not deletable any more.
    - [Lesser Method] The application code issues the database modification statement *x*. When *x* violates the relevant database constraint, the statement will throw an exception. (You should have came up with the database constraint when you designed the schema.)
    - The preferred method is better because it escalates easier to code that reports the failure reasons.

The “Products” page

If the accessing user is not an owner, the page should only display that “this page is available to owners only”. At the “Products” page the owners of the application can insert new products, where each product has a name, unique SKU, category and price. The name, SKU and list price are collected by text boxes. The category is collected by a dropdown menu that presents all categories.

Upon successfully submitting a new product, the owner sees on the Products page a confirmation about the product he just submitted. If the submitted data violate a constraint (eg, non-unique SKU, a list price that is not a positive number, a non-provided category, an empty string name, etc) then the page should simply report “Failure to insert new product”.

In the “Products” page the owner sees (on the left hand side) one link for each category. If the user clicks one of the category links then the “Products” page will show a table with all products of this category. The previous category selection is dismissed. For example, if the user first clicks on the “Category A” link and then on the “Category B” link then the page will only display “Category B” products (the previous, “Category A” selection, is dismissed). One of the category links is “All Products” and upon being clicked all products are shown (unless there is a text search condition, as discussed below.)

Furthermore, the owner may also issue a search for products that contain a string provided by the owner in a search textbox. In response, the “Products” page shows all qualifying products. Note that

- there may be two conditions at the same time: (a) the category selection and (b) the search box selection. The displayed products should be those that belong to the selected category *and* match the string of the search box (it is not an “*or”*).
- the search textbox string should be matched against product names only. The qualified products must contain the string of the search textbox as a substring. No wildcards, regular expressions etc are needed. Just plain substring matching.
- It may be the case that the category selection is placed first and the search box selection second. It could also be vice versa.

The category selection and/or the search box selection should be maintained while the user inserts, deletes or updates products. For example, assume that the user selects “Category A”. Then she successfully inserts a new product. The newly displayed Products page, in addition to the confirmation will also still be displaying “Category A” products. In another example, assume that the user selects “Category A” and then searches for products containing “foo”. The next displayed page shows “Category A” products containing “foo”. Then she updates a product leading to an “update failure”. The newly displayed page shows the “update failure” message and also still displays “Category A” products containing “foo”.

The table where the qualifying products are displayed has text boxes that display (and allow the modification of) the product name, product SKU and product price of each displayed product. Also, for each product there is a drop down menu that is initialized to the category of the product. There is an update button next to each product and a “delete” button. (Notice, this pattern of displaying data and updating them is reminiscent of updating in the students example we saw in class.)

Notes:

9. Deletions and updates may also lead to constraint violations that should be prevented and in such cases the requested modification should fail and the user should be informed it failed

("update failure", "delete failure") by appropriate message at the top of the next displayed "Products" page.

10. Notice that the owner may submit any of the following three requests in the Products page
    - either insert a new product in the database
    - or search for products by use of the category links and/or search box
    - or update/delete one of the displayed products

    This raises the following question: Say, the user has typed a new product in the relevant textboxes but then (instead of submitting the new product) he types a string in the search box and clicks the "Search" button. What should happen to the new product data that he typed but did not submit? For now, nothing should happen: the product data will be lost in this case. The same tactic applies to the other similar cases. E.g., the user typed a new product but then submitted a product update.

The products page also offers the links offered by the Home page.

The "Products Browsing" page

The "Products Browsing" page offers to the customers the same product searching functionality that the "Products" page offers to the owners. Namely, the users may search for products using the category links and the search box. The Products Browsing page does not offer links to the other customer-accessible pages.

However, in the list of qualifying products now there is a product link also for each displayed product. If this link clicked, it will lead to the "Product Order" page, so that the customer can order this product.

Notes:

11. The jsp:include directive allows you to use the same jsp fragment in more than one jsp's.

The "Product Order" page

The "Product Order" page displays the current contents of the shopping cart. It also shows the product that was just chosen (by clicking on it in the "Products Browsing") and asks the quantity of it that should be placed in the shopping cart. Upon a quantity being submitted, the shopping cart obtains one or more item. The application transfers the user to the "Products Browsing" page.

Notes:

12. We do not require any delete/update functionality for the items of the shopping cart.
13. The shopping cart may be either session-scoped (i.e., dismissed when the session expires) or persistent (i.e., it is remembered across multiple visits). You may choose either session-scoped or persistent and state your assumption.
14. You may assume that the same user does not engage in concurrent actions from two browsers.
15. You may assume that there is an infinite supply of all products. So, when a user orders *n* units of product *x*, assume that there are *n* units available.

The “Buy Shopping Cart” page and the “Confirmation” page

Whenever the session belongs to a customer, a link “Buy Shopping Cart” should appear at the top of the page. The link takes the customer to the “Buy Shopping Cart” page. There, the customer sees the products, amounts and prices of what he has chosen. He sees the amount price for each product and also sees the total price of the shopping cart.

He provides his credit card in a text box and clicks “Purchase”. (Ignore the shipping address and shipping method.) A successful purchase leads to a “Confirmation” page that shows what was bought. After the purchase the shopping cart is emptied. The data of the purchase (what products/quantities were in the bought cart, who bought, when and at what price) are stored in the database. The confirmation page provides a link to the “Products Browsing” page so that the customer can start again.

Notes:

- Notice that a product’s price may be updated between the time the product was placed on the cart, the time it was displayed on the “Buy Shopping Cart” page and the time the user actually purchases the cart. Therefore, the displayed “amount price for each product” is open to two interpretations: In the first interpretation, the price is fixed at the point that the user added the product to the cart. In the second interpretation, the displayed price is the current price of the product. Choose one of the two interpretations and state your assumption. In either case, assume that the final purchase price (charged price) is the one that was displayed on the “Buy Shopping Cart”. I.e., dismiss the possibility that the price may change after the “Buy Shopping Cart” was displayed and before it is purchased.
- The insertion of a purchase should be atomic, as the lectures discuss in transactional properties.
- In later stages we will need the complete information about which carts were bought, by whom, when, what products they included, what quantities of each product, and at what price each of the purchased products was bought.
- It is possible that after the time a product purchase is stored in the database, the product name changes. Your database should be organized in such a way that the amount of updating that needs to happen in such a case, is minimal.

Project Clarifications and Hints

- If a page is reached without the appropriate type of request (eg, the “Product Order” page reached without coming from a link from the “Products Browsing” page) a message should inform the user that the request is invalid.
- Notice how database constraints can simplify your coding.

Requirements and assumptions:

- Read carefully the project requirements.
- A few options are open-ended. For example, the requirements are open-ended on whether the search should be case-sensitive or case-insensitive. You may make your own common sense assumptions about these open-ended options. However

    - do not make assumptions that contradict the project requirements. For example, it is unacceptable to introduce an assumption that "the category name need not be unique", since the project requirements imply the opposite.
    - do not make assumptions that go against obvious common sense.
    - your application must be compatible with your stated assumptions.

Deliverables: You will need to submit before the deadline a zip file with the following content:

- A text file describing any assumptions you have made.
- The E/R diagram for the database of Project 1. Draw the E/R diagram on Powerpoint or Visio.
- A wire diagram of the web application pages, along the format discussed in class under "Model 1" programming. Draw the wire diagram on Powerpoint or Visio.
- The complete code, including the .sql file that you used to create the tables of the database.

The TAs will provide instructions on where to upload the zip file.

Soon after the deadline you will meet one of the TAs to demo and answer questions about your project. The TAs will provide instructions on how to book a time with them. The questions will include "how did you code …" questions and many of them will lead to code inspection and database design inspection. The entire team should be present at the demo and every member of the team should be able to answer any question, regardless of whether she/he is the member that wrote the code pertaining to the question. A team member that does not appear at the demo will receive 0 points, even if the other team members receive full points. That is, before the demo, each team should do database design review and code review, so that every member is aware of the code base.

Note that we utilize software for detecting highly similar submissions and detect potential cases of cheating.

Platforms and Technologies:

- May I use a relational database other than Postgres?
  The short answer is "practically no".
  The long answer is: You can use another database (eg, mySQL) during Project 1 without any substantial difference from using Postgres. However, on Projects 2 and 3 there is substantial analytics performance tuning. While the performance tuning and performance debugging techniques and fundamentals we teach you are very general, their details are not standard across all databases. That's where you practically get a large advantage by working with the same database (Postgres) that our lecture and discussion examples are based on. Theoretically, you can reproduce the same advantages on the database of your choice.
- May I use a NoSQL database? No. (If you are into NoSQL, you are very welcome to read my paper on a SQL++ query language for NoSQL and NewSQL and their frequent missing functionalities!)
- May I use an ORM? (eg Hibernate)? In Project 1 it is OK but discouraged. The reason we discourage it is because Project 2 and Project 3 require writing analytic queries. That's where

you will have to write queries yourself, either directly in SQL or using the ORM's advanced query formulation methods. In the latter case, you have less control on the specifics of the issued queries and therefore on their performance tuning. Furthermore, the common patterns of how to use ORMs may drive you down inefficient computations. (We will see such dangerous examples of ORM use very late in the class.) Nevertheless, if you like the lines-of-code savings that automated object-relational mapping provides AND you believe you master enough the particular ORM that you will use to avoid performance pitfalls (without asking the TA help on the particular ORM you use), go for it. In such case, it will be your responsibility to demonstrate the SQL queries that are issued by your code.

- May I use Ruby on Rails or <complete with other programming language and framework you like>? Yes you can use another framework but be aware of the above point if your framwork comes with an ORM. For one thing, I like many aspects of Ruby on Rails (including aspects of its ORM) and I briefly go over them at the last lectures. I also like Angular and will go over it.
  Still, the preferred programming language is Java/JSP and database connectivity via JDBC. While Java/JSPs are not the "hottest" web technology, all UCSD students and our TAs know Java. Furthermore
  1. html templating languages are not that different across frameworks
  2. MVC programming can also be done in Java/JSP in case you like MVC frameworks
  3. a lot of the web action is on the browser-side Javascript code, where the choice of server side framework does not matter that much.
- May I use a Java MVC framework? You may use a Java MVC framework (Struts, Spring) if you wish, but is not required. We will briefly discuss MVC in class.
- May I use Javascript in Project 1? Yes, but notice that there is no required browser-side functionality in Project 1 that necessitates Javascript. If you think that Javascript is required in order to build a certain functionality you have probably assume more than what we ask you and may end up doin more work than needed. While at that, the redirect command is also not needed and you have probably misunderstood a requirement if you feel it is needed.
- May I use another application server, i.e., not Tomcat? Yes, but do not expect application server troubleshooting from the TAs.

Other notes:

- Our class examples have been using the TEXT type for variable length strings. The TEXT is not part of the SQL standard, though it is supported by most databases (including, of course, Postgres) and is their suggested type for variable length strings. The VARCHAR(*n*) type is the SQL standard type that is closest to TEXT. It also allows variable length strings but of size up to *n* only.
- Dismiss the issues pertaining to refreshing or pressing a back button.

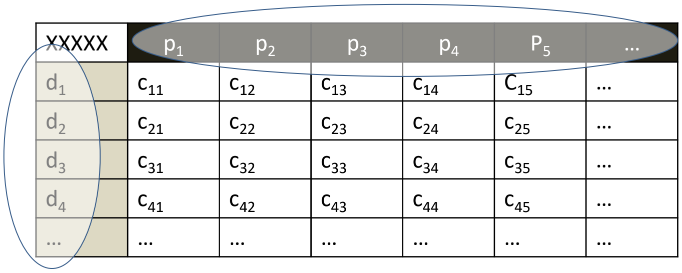
Assignment 2: The basic analytics part

Deadline: Wednesday, May 17, 9PM (tentative)

The owners want to understand the purchasing habits of the customers. So, they set an analytics dashboard…

The "Sales Analytics" page

An owner can see a 2-dimensional report about sales aggregates on this dashboard page, similar to the figure below. The row dimension points (d_1, d_2, d_3, …), the column dimension points (p_1, p_2, p_3, …) and the cells (c_{11}, c_{12}, …, c_{21}, c_{22}, …) in the 2-dimensional report are determined by the following requirements.



XXXXX	p_1	p_2	p_3	p_4	P_5	…
d_1	c_{11}	c_{12}	c_{13}	c_{14}	C_{15}	…
d_2	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	…
d_3	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	…
d_4	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}	…
…	…	…	…	…	…	…

Rows dropdown menu: A rows dropdown menu offers the two options "Customers" and "States". This dropdown menu allows the owner to choose whether the rows of the 2-dimensional report correspond to (option a) individual customers or (option b) customer states.

- If "Customers" is chosen, then each displayed row corresponds to a customer. That is, d_1, d_2, d_3, … are the names of individual customers.
- If "States" is chosen then each row corresponds to a state. That is, d_1, d_2, d_3, … are the names of states.
- If no option is chosen, the behavior is equivalent to having chosen "Customers".

The columns of the 2-dimensional report always correspond to individual products, i.e., p_1, p_2, p_3, … are the names of products.

Order dropdown menu: The order dropdown menu offers the two options "Alphabetical" and "Top-K". This dropdown menu allows the owner to choose whether the display of customers, states and products is in "Alphabetical" order or according to the total sales dollars in the case of Top-K.

If it is alphabetic then the first displayed customer name (d_1) is the first in alphabetic order, d_2 is the second in alphabetic order etc. Similarly for states and products.

If it is "Top-K", then the order of customers, states and products depends on the sales and the presence of filtering conditions. For a short while, let us assume there are no filtering conditions. The requirements on customers (similar logic applies to states and products) is as follows: If the rows drop down menu was set to "Customers" then the 20 row headers show the Top-20 customers ordered by their total sales.

The following example illustrates the ordering requirement: Let us assume for a minute that the database has three customers only, *x*, *y* and *z.* Customer *x* has made only one purchase: $20 of apples, which belong to the category "Produce". Customer *y* has made two purchases: $15 of Coke, which belongs to the category "Beverages" and $10 of Tide detergent, which belongs to the category "Cleaning Products". Customer *z* has bought nothing. Then, if the rows dropdown menu is "Customers" and the order dropdown menu is "Top-K", the name of customer *x* appears as d_2 and the name of customer *y* appears as d_1. Customer *z* appears last, as d_3. Later, we will discuss how these requirements evolve in the presence of filtering conditions.

The dashboard gives the following *sales filtering option*:

- A product category filter dropdown menu limits the reported products to the chosen product category.
- The drop down menu offers an "All" option ("All Categories"). If no option is chosen at a menu the behavior is the same with choosing the respective "All" option.
- The filtering option may affect the order of customers, states and products in the top-K ordering option. Continuing from the previous example, if the product category filter is set to "Produce", then customer *x* will appear as d_1, i.e. above customer *y* ($20 total Vs $0 total). Notice that customer *y* is still qualified to be displayed even though he has zero sales that pass the filtering conditions. The order between *y* and *z* is nondeterministic.

The dashboard provides a "Run Query" submit button. When this button is clicked, the 2-dimensional report is produced. The details of the 2-dimensional report produced by the "Run Query" are as follows:

- The 2-dimensional report's leftmost column shows row headers (customers or states), in bold. In particular, each row header of the 2-dimensional report shows the name of a customer or state and within parentheses the total sales $ associated with the displayed customer or state, after having taken into account the sales filtering option. Continuing from the previous example with customers *x*, *y* and *z,* assume that the product category filter dropdown menu is set to "Beverages" and the rows menu is set to "Customers". Then the displayed total sales associated with customer x is the total beverages' sales $ to customer *x* (i.e. $15). It is *not* any more the total amount of sales to *x* across all product categories (i.e., it is not $25).
- The 2-dimensional report's top row shows column headers (product names), also in bold. In particular, each column header shows the name of a product, truncated to the first 10

characters and below (also within parentheses) the total $ associated with the displayed product, after having taken into account the sales filtering option.

- The dashboard shows at any point 10 columns and 20 rows plus the header column and the header row. For example, if 1000 customers qualify according to the filtering option, then the top-ordered 20 customers will be shown.
- A qualified product (respectively, customer or state) should appear at a column (respectively row) even if it is associated with $0 sales. For example, customer *z* should still appear.
- The cell c_{ij} of the dashboard aggregates the total sales that correspond to the entity of row *i*, column *j* and any applicable filter.
- All menus should be defaulted to the options they had when the user clicked “Run Query”.
- Be careful of various duplicates: A customer may have bought many times the same product.
- You may print the states either with their full names (e.g. “California”) or their 2-letter abbreviations (e.g., “CA”).

If the rows are individual customers (respectively states) then a “Next 20 customers” (respectively “Next 20 states”) submit button is also displayed. If it is clicked we get a 2-dimensional report that shows the next 20 customers (respectively states), sorted under the same criteria with above. If the user keeps clicking the last batch of customers or states may be fewer than 20. It is OK that you may provide a button “Next 20 *<something>*”, while the next page may display fewer than 20. However, the “Next 20” should not be provided if there are no more customers or states to display.

A “Next 10” products is shown at all times and has similar behavior. The dashboard does not provide “Previous”. Also, after any of the two “Next” buttons is clicked, the dashboard page stops showing the rows menu, order menu and the sales filtering options, i.e., the filtering options, the order options and row menu options stay in effect “as is” and cannot change (without visiting again the original “Sales Analytics” page).

Performance tuning

Your goal will be to deliver the requested functionality and also deliver top response time performance. Here is the performance requirements and techniques:

- Use the schema of the data generator issued by us. You may continue with your own schema of Project 1 only if your schema is identical, modulo renaming, to the solution we provided. In such case you may need to tweak the generator. If you are not sure whether your current schema is sufficiently close (for the purpose of this Project 2) to the data generator, contact your TA and he/she will tell you if you can continue with your Project 1 solution.
    - The reason for recommending that you may have to give up your Project 1 schema is that some schemas, though sufficient for correctly solving Project 1, may lead to inferior performance in Project 2.
- Write the most efficient jsp and queries (following techniques we covered/will cover in the lectures) that produces the 2-dimensional report of the “Sales Analytics” page. Obviously the data processing heavy-lifting of your program will be done by its database processing part.

    - Your program may involve queries that are parameterized by the results of other queries. If so it should make appropriate use of prepared statements.
    - Your program may involve the creation of temporary results.
    - For maximum performance, your program should run without enforcing transactions.
- Do not compromise the response time of the 20x10 2-dimensional report that comes after the “Run Query” in order to accelerate the response time of the “Next”. (I.e., your program should do the minimum amount of computation needed to produce the first 20x10 report.)
- Do *not* precompute anything. Your jsp’s (or other web app framework code) and their queries should operate directly on the tables of the provided data generator (or equivalent tables, see 1st bullet).
    - Actually the project can benefit greatly from precomputation but this will be a topic of Project 3.
- Create any indices that will be beneficially used by your queries. Do not create useless indices. In particular, perform the following performance tuning steps and report on them:
    - As a first step, assess which indices *may* benefit queries. Prepare an *indices’ report*, where for each query emitted by the “Sales Analytics” page you provide a list of indices that you suspect that may benefit the particular query. Notice, “may benefit” does not imply “will benefit”. Therefore some of these indices may simply be reasonable candidates that experimentation is needed in order to figure out whether they are actually useful. Eventually you will mark which ones of these indices are useful and which ones are not – read on about this.
    - The worthiness of an index may depend on whether the required database data are already cached in the main memory of the database server at the time of being queried (i.e., at the time of running the “Sales Analytics” page). Tune your program and your index choices for the following two extreme cases. For each of the two cases report (a) running time of the overall jsp, (b) running time of the individual queries and (c) your index choices for these two cases.
        - Case 1, the “Small and Hot” database. Use the generator to create a small database. Ensure that the measured runs have ran with all the relevant data being fully cached in main memory. A simple way to achieve this is to ensure that the entire database has been cached. (In practice, you need not ensure that the entire database data and indices are cached in main memory. It is sufficient to ensure that the data needed for the page’s evaluation are already cached in memory.)
        - Case 2, the “Large and Cold” database. Use the generator to create a large database. Ensure that for the measured runs (almost) no database data are cached at the time of the program’s run (i.e., the machine is “cold”). The safest approach to ensure a cold database is to clear the DB and OS caches between experiments. (Other tricks discussed in class: Turning on and off the computer can also do the trick. Running a scan query on a guge irrelevant table can also do the trick.)

- Generally, it is possible that you may find an index X to be useful in the “hot database” but not in the “cold database”. And vice versa. Report accordingly.
- In the two extreme cases, adjust the Postgres sequential-Vs-random access ratio accordingly.
- Be aware that some (but not all) analyses on the “Large” database will be unavoidably very slow, i.e., even your best program and best index choices will still stay far away from what a reasonably patient user would call “online performance”. The performance of these queries will be fixed in Project 3.
- You may also generate “medium and warm” databases. We do not ask you to report on sizes between the small and large. But medium may be a potentially good staging step, as you progress from the small to the large.

- For each index Y that you are sure that it is beneficial, you need not contact any experiment to verify its usefulness. Just mark that you are sure and provide your reasoning. Nevertheless, debug: Use EXPLAIN to check whether some query (or queries) of your program indeed used the index Y. (i.e., avoid the embarrassment where you declare that an index is surely useful and it turns out that your queries never use it!)
- Next, for each candidate index in the indices’ report where you are not sure whether it actually benefits performance, do an experiment. The simple way to experiment if an index *X* is beneficial is to first run the “Sales Analytics” page without having created the index *X* and then create the index *X* and run the page again. If you see no performance difference, the index was not worthy. Be careful of caching effects when you execute such experiments. It is very possible that an index may useful in the “small and hot” but not in the “large and cold” and vice versa. Notice, there is also a (remote but not impossible) possibility that the database system may use an index though it is not actually beneficial. In such case, you may even see worse performance upon using the index. Of course, in such a case the index should not be created.

- Use the provided data generator, with the suggested settings for small and large. Tuning the settings will require profiling the memory capacity of your machine. (Ask for TA advice as well.) The generator allows you to easily make up to hundreds of thousands of customers, thousands of products and billions of sales.
- *Hint:* Avoid developing programs that fetch unnecessarily a lot of data from the database to the application’s data structures and then rely on Java (or the programming language that you use) to do the remaining processing. For example, a suboptimal solution would be to write a query and JDBC code that fetches to memory every non-zero combination of <customer x, product y, total sales s> and do the rest with Java. This approach can be problematic in many ways: First, is there enough RAM available to store it? For large number of customers and products there won’t be. Second, even if there is enough RAM, why bring every combination to memory when the page eventually shows just a 20x10 matrix.
- *Hint:* Prefer solutions that issue a few queries. As discussed in class examples, very often this approach pushes the optimization task to the database as opposed to you being responsible.

- You will not be graded on the absolute response time (in seconds) but you will be graded on whether you wrote appropriate programs/queries and selected the right indices for maximizing the performance of your solution.
    - Since everyone has a different system, large response time differences can occur for the same solution. For one, SSD Vs HDD can make a very significant difference in the large case.
    - The best solution will not be the same across all systems and configurations. Again SSD and HDD can make a big difference. Utilize the TAs' advice!
    - If you want to expedite your experimentation, you do not need to build a fully working "Sales Analytics" page before you run experiments. It may be beneficial to test the performance of the queries you have in mind from pgadmin and/or by writing little JDBC test programs.
- You will need to provide the following and argue for the appropriateness of your solution:
    - Code
    - Best index choice.
    - For each query of your program, provide its running time for the small and the large database. To avoid experimental flukes, average over at least 3 runs.
    - The indices' report: candidate indices, which one of them are the indices you are sure that are beneficial (and your reasoning), which ones you experimented with in order to decide whether they are (or they are not) useful, the experimentation results.

Technology:

- We do not recommend the use of ORMs. If you use ORMs make sure that you are aware of the exact queries that they issue.

The "Similar Products Page"

One typical data mining operation is to find products that are often bought by the same customers. There is a 15-year old legendary example, where a retailer discovered that very often the same customers who bought baby diapers also bought beer. The retailer promptly moved the diaper isle close to the beers.

One way to think about this problem is the so-called *cosine similarity*. Consider (conceptually) that for every product there is a vector $<c_1, c_2, c_3,\ldots>$ where c_i stands for how many dollars of this product were bought by Customer i. Let us assume a product p_1 with vector $<c_1^1, c_2^1, c_3^1,\ldots>$ and a product p_2 with vector $<c_1^2, c_2^2, c_3^2,\ldots>$. Then the cosine similarity $p_{1,2}$ of p_1 and p_2 is $SUM_{i=1\ldots n}\, c_i^1\, c_i^2$ where n is the number of customers. There is also the *normalized cosine similarity* that divides $p_{1,2}$ with the totals sales of p_1 and the total sales of p_1. (The idea is that a similarity between milk and eggs may not be so impressive since people buy a lot of milk and a lot of eggs.)

Your task is to produce an efficiently computed page that displays the 100 product pairs that have the highest normalized cosine similarities.

Hint: Really computing vectors is an inefficient way to do the job.

Project 3

Deadline: Monday, June 5, 11pm.

The web application owners were not that happy with the performance of some queries in Project 2. They understand you did the best you could do, given the requirements, but the speed was just not enough. So, they require a Project 3, where the application will use precomputed tables, which will be incrementally maintained as new sales happen. The report will also use Ajax in order to refresh the page without fully recomputing it.

The new "Sales Analytics" page

An owner can see the 2-dimensional report about sales aggregates, which in Project 3 has the following differences, in function, over the 2-dimensional report of Project 2:

- It does not provide a Rows drop down menu. Instead, the rows always correspond to states. The columns, as it were alreadythe case, correspond to columns.
- It does not provide alphabetic Vs Top-K choice. Instead, the Top-K option is always used.
- The category filter remains.
- All 50 states are displayed. Also, 50 products are displayed. Notice that this means that the whole report will not be fully visible at any time. The user will have to pan his browser window in order to see the product columns at the right side. Indeed, depending on how large the browser window is, the user may also have to scroll down to see the bottom state rows.
- The page offers refresh buttons. Upon clicking the refresh button, the page must be updated very fast regardless of what is the displayed analysis. (It needs to be updated since new sales may have happened between the last time the page was ran/refreshed and the current refresh.) The scroll and pan of the browser should not be lost upon the refresh, i.e., the refresh must be an update of the 2d report via Ajax.
    - For usability purposes, since the page will be too large to fit in a window, it is advised that you put four refresh buttons, close to the 4 corners of the 2d report so hat the user can click whichever one is visible.
- Every number that has been updated (after the refresh) must appear in red. "Number" refers to the numbers that have appeared in a row header, a column header or a cell.
- The order of rows and columns will not change by the Refresh, even if this leads to an out-of-order table. For example, assume that product x has $25 and product y has $20. Then product x appears to the left of product y. Now, assume that product y was just sold in a few more purchases, worthing $10. Then the displayed total sales of product y will be updated (after the refresh) to $30. However, x will still be appearing to the left of y. Product x will move to the right of y only after a new "Run".
- There is a Run button, as it were in Project 2. Upon clicking it the whole page is redrawn, i.e., you do not need to use Ajax to refresh the page upon "Run".

- It may become the case that after additional sales happened, one or more products that were among the top-50 in sales are no longer among the top-50. In such case, the whole column (including the header) for the product that is no longer among the top-50 should become purple. A text on the top of the page should say that the top-50 has changed and should say which are the products that are now in the top-50 but do not appear in the 2d report). Display the name of the products that are in the top-50 but do not appear in the 2d report and also report the total sales of each one of them. Notice the simplification: You do not have to remove the purple columns and put in their place columns for the new Top-50 products. Rather, the user has to Run again in order to obtain the revised 2d report that shows the new 50x50.
    - For example, imagine that at Time 0 the product x had $25 of total sales and it was the 50th in total sales (according to current filtering condition). The product y had $22 of total sales and was not one of the top-50 products. The user clicked Run at Time 0 and, as expected, got a report that showed x but did not show y. Then, at Time 1 a few more sales happened and raised the total sales of y to $26. No new sales of x happened between Time 0 and Time 2. At Time 2 the user clicked Refresh. The product x column (and header) will become purple. The text on the top of the page will say that product y is now in the top-50. At Time 3 the user clicks Refresh. Assume no sales happened between Time 2 and Time 3. The purple x column remains purple and the message about y is still there because y is still one of the top-50 products that do not appear in the report. After the user clicks Run the new report will have y but no x.
- If the user wants to change the product filtering condition then she must choose the option she wants and “Run” again. You do not have to change the report if the user changes the option on the category menu without clicking Run.
- There is no “next” buttons.
- You are being provided a jsp that buys N orders, i.e., makes N insertions in the Orders table. For those of you who do not code in jsp, you can easily separate the code that does the orders from the rest. Employ this code for N=10 orders at a time. The scenarios to prepare for are the following:
    - Your computer contains the large database you used in project 2.
    - This time you can prepare various precomputations (including indices) ahead of running the analytics application.
    - Next run the analytics application by mixing ordering, run and refresh. Eg,
        - A user produces a 2d report, according to the simplifications described above.
        - Another user (not in the same session) runs the buying jsp (or whatever you have instead of the jsp) to make more orders.
        - The first user clicks Refresh. Any changes caused by the purchases of the second user should be propagated to the screen very fast, thanks to smart precomputations that have been done.
        - You may keep repeating the buying and refreshing steps in arbitrary permutations.
        - From time to time, also click the Run again.

Implementation and Performance Requirements

- When computing the refresh of the analytics page, assume that the only change that has happened since the last run or refresh is that new sales have been added. Ignore the possibility that the states, users, products or categories tables have changed.
- Employ precomputed tables to optimize the performance of the analytics page.
- The owners of the application require that the buying jsp is not substantially slowed down by the maintenance of the precomputed tables. Therefore, they ask that you do not add any additional database operation (INSERT, DELETE, UPDATE) for maintenance of the precomputed tables on the purchasing code. The only thing you may do is maintain a “log” table that captures what additional sales have happened since the last time you refreshed the precomputed data. Your application will engage in “lazy”/”deferred” update of the precomputed tables: At the time the “Sales Analytics” page is run or refreshed it should first somehow find the new sales that happened since the last time a “Sales Analytics” page was ran. (A log table is one solution.) Then it should efficiently update the precomputed tables with the new sales. Then it should refresh the analytics page.
    - Notice that there may be multiple owners using the “Sales Analytics” page at the same time. For example, owner A may ran the page at noon. Then owner B may ran the page 10 seconds later. Then owner A may refresh another 15 seconds later. Optimize for it.
    - The “do not add any additional operation” also forbids the use of triggers that are triggered by the purchases and update the precomputed tables at the purchase time. (If you do not even know what is a trigger and you code before we discuss triggers in class, just dismiss this comment.) However, you may use triggers to insert tuples in the log table.
- Utilize precomputation of aggregations aggressively. That is, the queries that fuel the report should not have to perform aggregation. The aggregation will have already happened and be reflected in precomputed results.
- The CREATE TABLE of the precomputed tables can be made via pgadmin.
- Initially load the precomputed tables with the appropriate data, via INSERT commands. In particular, given the Orders table, write a “first precomputation” program or script that will insert data in the precomputed tables efficiently. (The application logic of the program is very simple. Indeed, it need not be a Java program. It can just as well be an SQL script.)
- Querying and updating the precomputed tables will benefit from creating certain indices on them. Build such indices.
- The Ajax code that refreshes the 2-dimensional report should only update the numbers that have been affected by the new sales. “Number” refers to the numbers that have appeared in a row header, a column header or a cell. The Ajax code should *not* replace the entire table with a new table.
    - Though one can replace the entire HTML table with no significant performance penalty, partial updates matter in other cases where, for example, a Google map is used instead of an HTML table.

What to Submit

Submit the following:

- The list of precomputed tables (their CREATE TABLE commands) and any CREATE INDEX done on them.
- The precomputation code.
- The code that takes care of the buying (which includes the code that logs the changes that happen in Orders)
- The code that executes upon the Run and the Refresh. Notice that any update of the precomputed tables happens during the “Run” and/or the “Refresh”

Hints

- Notice that the original orders table may have multiple tuples for the same customer and product pair.
- The key to a simple code that updates only specific numbers is to associate every number with appropriate identification data that can make it easy to figure the sales that correspond to each number.