The project’s goal is a shopping application. The application enables the customers to browse and search a product catalog and eventually buy. The owners of the application analyze the purchasing and browsing patterns of the users in order to obtain insights.

The assignments escalate as follows:

- The first assignment is primarily about the transactional/operational part of the application and involves only basic database design/programming and web application programming.
- The second assignment presents performance challenges in the analytics part. It requires the implementation of a bunch of performance-enhancement techniques involving indexing and materialized views.
- The third assignment takes a different take on performance challenges. Its hands-on part asks you to emulate column database and bitmap indexing optimizations on an open source database (Postgres).
- Finally, the fourth assignment brings Ajax and Javascript components in live visualizations.

Generally, an assignment’s deadline is one week after the last lecture that covered material pertaining to the assignment.

Assignment 1: The transactional part of the shopping application

SUBMIT on: Friday of 5th week, 9PM.

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. Just be consistent across all pages on how you display.

All of the above are necessary for a successful signup.

Upon a successful signup, the user should reach a page that simply declares “You have successfully signed up”. If anything goes wrong the page should simply declare “Your signup failed”. The typical reason for things going wrong is that a constraint on the data is violated, as the following points explain.

Notes:

1. You should prevent the possibility that two users pick the same name. In the unfortunate event that two users end up asking for the same name, there will be a signup failure. Be alert of the possibility that the users may be attempting to register at the exact same time. If they happen to pick the same name, the application’s response will be signup failure.

2. For 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. That is, you may assume that the user indeed has provided name, age, role and state. If the user fails to provide name, age, role or state then it is sufficient to just have the database throw an exception because the corresponding constraint has been violated and the application will report "signup failure". We will come back to user-friendly functionality for form verification in Project 4.
3. For 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 (during Project 1) that the currently logged-in user has the right to see the requested page.

The "Login" page

In the "Login" page the user declares 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 displays "The provided name <provided name> is not known" and keeps showing the textbox where the user can declare his name.

No password required. No "log out" is required.

The "Categories" page

At the "Categories" page the owners create product categories, where each category has a unique name (collected by a text box) and a description (collected by a small text area). The owners can insert, delete and update categories.

The page looks like the "students" page of the class example, in the sense that it displays the name and description of each category n text box and text area respectively, and offers "Insert", "Delete" and "Update" buttons.

Notes:

4. Any of the data modifications (insert/delete/update) enabled by this page may lead to a violation of some constraints placed on category data. Any data modification leading to a constraint violation should throw an exception and the "Categories" page should display that the requested data modification failed. Again, we do not engage yet in the more user-friendly form verification techniques.
5. A product category cannot be deleted if there are already products referring to it. (You will see in the "Products" page that product categories are referred by products.) In particular, there should be no "Delete" button next to a product category if this category is already referred by a product.
6. 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 do *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:

- [Preferred method] Just before issuing the database modification statement *x* the application code checks whether *x* is still applicable. (Notice the needed use of transactions.) If the action is not applicable any more, the page will contain a message about what happened.
- The application code issues the database modification statement *x*. When *x* violates a constraint, the statement will throw an exception. The application code catches the exception (as usual) and inspects it to figure out why the statement did not execute. The application page reports accordingly.

The "Products" page

At the "Products" page the owners of the application 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 submitting a new product, the owner sees on the page a confirmation of what he 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.

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.

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:

7. 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.
8. 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 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.

However, in the list of qualifying tables 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:

9. The jsp:include directive allows you to use the same jsp fragment in more than one jsp’s.
10. The page product searching functionality is offered only when a user is logged in. If no user is logged in then the page simply says “Please log in first” and provides a link to the login page.

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:

11. Make sure that an accidental refresh or back button will not re-issue the order action. Appropriately choose between GET and POST. Recall the notes about when we choose GET Vs when we choose POST.
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.

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 application transfers to the "Products Browing" page.

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 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. (See above regarding assumptions.)
- 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 E/R 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.

Platforms and Technologies:

- Use the Postgres SQL database.
  - For this project only you are allowed to use an ORM if you wish, but we discourage it.
- The web application programming should be done in Java/JSP.
  - You may use an MVC framework if you wish, but is not required.
  - There is no browser-side functionality. You do not need to code in Javascript. If you think that Javascript is required in order to build a functionality you have probably understood incorrectly the requested functionality.
- Use the Tomcat app server.

Assignment 2: The basic analytics part

Deadline: Wednesday May 28, 2PM.

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.

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 (a) individual customers or (b) customer states.

- If "Customers" is chosen, then each displayed row corresponds to a customer and the rows are sorted according to the customer's name, ascending (A to Z).
- If "States" is chosen then each row corresponds to a state. The states are ordered alphabetically according to the state's full name (A to Z).

- 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, sorted according to the products’ names (A to Z).

The dashboard gives the following *sales filtering options*:

- A state dropdown menu allows the user to choose a particular state (eg, Arizona). If the rows menu is set to “Customers” then (after the “Run Query” is clicked) only customers of the chosen state will be displayed. If the rows menu is set to “States” only the particular state will be displayed.
- A product category dropdown menu limits the reported products to the chosen category.
- An age dropdown menu offers age group options 12-18, 18-45, 45-65, 65-. If the rows menu is set to “Customers” then (after the “Run Query” is clicked) only customers of the chosen age group will be displayed. If the rows menu is set to “States” then the aggregations of the report should only include sales to customers of the chosen age group.
- All drop down menus offer an “All” option (“All states”, “All Categories”, “All Ages”). If no option is chosen at a menu the behavior is the same with choosing the respective “All” option.
- When more than one filtering options are chosen, their effect is conjunctive. That is, the aggregations of the report are limited to the sales that meet all the chosen filtering options. Similarly any displayed customers, states or products must be compatible with all the filtering options described above.

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 options. For example, assume that the product category dropdown menu is set to “beverages” and the rows menu is set to customers. Then the displayed total sales associated with each customer x is the total beverages’ sales $ to customer *x*. It is *not* any more the total amount of sales to *x* across all product categories.
- 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 options.
- 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 options, the first 20 alphabetically will be shown.
- If the rows drop down menu was set to “States” then the 20 row headers show the first 20 states in alphabetic order (assuming the state menu is not set to a particular state.

- A product (respectively, customer or state) should appear at a column (respectively row) even if it is associated with $0 sales.
- Next consider an interior cell in the i-th row and j-th column of the 2-dimensional report and assume that the i-th row corresponds to customer *x* and the j-th column corresponds to product *y*. Then the cell displays a number that is the total sales $ of product *y* to customer *x*.
- Next consider an interior cell in the i-th row and j-th column and assume that the i-th row corresponds to state *x* and the j-th column corresponds to product *y*. Then the "Age" sales filtering option also affects the number shown in the cell. If it is set to "All" then the cell displays a number that is the total sales $ of product *y* to customers of the state *x*. If the Age is set to an age group *z*, then the displayed number is the aggregate of product *y* sales to customers of the state *x* that belong to the chosen age group *z*.
- The rows dropdown menu and the sales filtering options should be defaulted to the options they had when the user clicked "Run Query".
- Recall that it is possible that a customer bought many times the same product.

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. 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 and the sales filtering options, i.e., the filtering options and row menu options cannot change (without going back to the original page).

Performance tuning

Your goal will be to deliver the requested functionality and also deliver top response time performance. Expect a significant amount of experimentation until you tune the performance. This is actually what happens most often in practice. Here is the performance requirements and techniques.

- Use the schema of the Project 1 solution 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. If you are not sure whether your current schema is sufficiently close (for the purpose of this Project 2) to our Project 1 solution, contact your TA.
- To facilitate your experimentation, use Chunbin's data generator, which allows you to easily make hundreds of thousands of customers, thousands of products and billions of sales.
- The amount of RAM in the database buffer and the OS cache of your computer should be at least 3 times smaller than the size of the database. You may utilize our data generator to create a large database that meets this requirement. You can identify the size of the Postgres database buffer by Tools->Server Configuration->shared_buffers. It is usually set to a very small value. But the OS cache of your system will probably be much larger. If you cannot identify its size, just

make the database size at least 3x larger than the total RAM in your system. (The larger the better.)

  - You can find the precise size of the database by following the instructions at https://wiki.postgresql.org/wiki/Disk_Usage. Or, you may use a back-of-the-envelope calculation that goes as follows: Most of the database size is due to the Sales table, so it is enough to calculate the Sales table size in these two steps: Calculate the size of each Sales tuple as sum of the sizes of its fields + 16 bytes. (For various reasons, each tuple has 16 bytes of overhead.) Then multiply the size of each Sales tuple with the number of Sales tuples and this is almost the size of the table.
  - If you are not sure whether your database is sufficiently large for the performance tuning purposes, contact the TAs.
  - In the initial phases of your development your program may issue some very non-optimal and slow queries. Therefore I suggest you do not immediately load a multi-GB database on your system. It will slow down the early stages of your debugging/performance tuning. Grow the database as you proceed.

- Write the most efficient jsp, following techniques we covered/will cover in the lectures, that produces the 2-dimensional report above. Obviously the data processing heavy-lifting of your program will be done by its database processing part.
  - Your jsp may involve queries that are parameterized by the results of other queries. If so it should make appropriate use of prepared statements.
  - Your jsp may involve creating temporary results in the database.
- Create any indices that will indeed be used by your queries. Do not create useless indices.
  - During the development/tuning phase you may create any indices that you believe that will boost performance and see whether your queries indeed achieve better performance once you create the index. The simple way to find out if an index is beneficial is to first run your queries without having created the index and then create the index and run the queries again. If you see no performance difference, the index was not worthy. But be careful of caching effects before you draw a hasty result. Keep a log of your experimentation. You can use it later to justify your choices.
  - Your experimentation may be impacted by caching, as discussed in the lectures. The safest approach is to clear the caches.
- Do not compromise at all the response time for the 2-dimensional report that comes after the “Run Query” in order to accelerate the response time of the “Next”.
- Do *not* precompute anything. Your jsp’s and their queries should operate directly on the tables of the Project 1 solution (or equivalent, see 1st bullet).
  - Actually the project can benefit from precomputation but this will be topic of Project 3.
- *Hint:* Avoid queries that fetch unnecessarily a lot of data from the database to the application’s data structures and then rely on Java 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 10x20 matrix.

- You will not be graded on the absolute response time (in seconds) but you will be graded on whether you wrote appropriate programs/queries for maximizing the performance of your solution.
    - Since everyone has a different system, large response time differences can occur for the same solution. They will depend on the type and speed of disk(s) used by the system, on the size of the database that you generated, and likely on the amount of available RAM. On systems with SSD the CPU can also make a difference. Long story short, we will not grade on the absolute response time.
    - Furthermore, we expect that the best solution will not be the same across all systems. In particular, SSD-based Vs hard-disk based storage can lead to different solutions.
    - Utilize the TAs' advice!
    - To expedite your experimentation, you do not need to build fully working jsp's before you run experiments. It may be beneficial to test the performance of queries from pgadmin and/or little JDBC test programs.
- You will need to argue You can argue for the appropriateness of your solution in many ways:
    - Experimental arguments: I experimented with Solution A and Solution B and you can see that Solution B's response time is superior (show chart with response time results), so I implemented Solution B.
    - Analytical arguments: I implemented Solution A because it only makes few page accesses since…. I did not implement Solutions B and C because they would do at least that many page accesses since …, which is much worse than Solution A.
    - Be warned that purely analytical thinking may be dangerous for many reasons. First, as we see in the lecture, the involved algorithms are complex, which can complicate your analyses. Second, in some occasions you may guess what the database's best option is, analyze accordingly, but the database may do something else. Long story short, experimentation always helps.
- We will *not tell you* line by line what solutions to code. You will need to synthesize your own solutions from your knowledge of SQL and JDBC and the techniques we covered.

Assignment 3

Deadline: TBD

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

The "Sales Analytics" page

An owner can see a 2-dimensional report about sales aggregates on this dashboard page.

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 (a) individual customers or (b) customer states.

- If “Customers” is chosen, then each displayed row corresponds to one customer and the rows are sorted according to the customers’ total $ amount of purchases (after having taken into account the sales filtering options listed below).
- If “States” is chosen then each row corresponds to a state. The displayed 50 rows are ordered alphabetically according to the state’s full name.
- 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, sorted according to the products’ total sales (again, after having taken into account the sales filtering options listed below).

The dashboard gives the following *sales filtering options*:

- A state dropdown menu limits all the aggregations of the report to sales by customers of the chosen state only. Furthermore, only customers of the chosen state will be displayed (in case the rows display individual customers). Only the particular state will be displayed if the rows menu is set to “States’.
- A product category dropdown menu limits all the aggregations of the report to sales of products of the chosen category only. Furthermore, only products of the chosen category will be displayed in the report.
- An age dropdown menu offers options 12-18, 18-45, 45-65, 65- and limits all the aggregations of the report to sales by customers of the chosen age group only. Furthermore, only customers of the chosen age group will be displayed in the report.
- All drop down menus offer an “All” option (“All states”, “All Categories”, “All Ages”). When an “All” option is chosen, the aggregations include all sales. If no option is chosen at a menu the behavior is the same with choosing the respective “All” option.
- When more than one filtering options are chosen, the aggregations of the report are limited to the sales that meet all the chosen options. Similarly any displayed customers, states or products must be compatible with all the filtering options described above.

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 options. See example, below on what it means “after having taken into account the filtering options”.
- 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 customer, after having taken into account the sales filtering options.

- The dashboard shows at any point 10 columns and 50 rows plus the header column and the header row.
- If the rows drop down menu was set to “Customers” then the 50 row headers show the Top-50 qualified customers ordered by their total sales that meet the sales filtering options. Therefore it is implied that the order of displayed customers depends on the sales filtering options. Also, recall that a customer is qualified only if his age and state are compatible with the filtering options.
  The following example illustrates the above requirements: Let us assume for a minute that the database has two customers, *x* and *y.* 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”. If all the sales filtering options are set to the “All” options then customer *y* should appear above customer *x* ($25 total purchases Vs $20 total purchases). If the product category is set to “Produce” and the other options set to “All”, then customer *x* will appear 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.
- If the rows drop down menu was set to “States” then the 50 row headers show the 50 states in alphabetic order.
- Next consider an interior cell in the i-th row and j-th column and assume that the i-th row corresponds to customer *x* and the j-th column corresponds to product *y*. Then the cell displays a number that is the total sales $ of product *y* to customer *x*.
- Next consider an interior cell in the i-th row and j-th column and assume that the i-th row corresponds to state *x* and the j-th column corresponds to product *y*. Then the “Age” sales filtering option also affects the number shown in the cell. If it is set to “All” then the cell displays a number that is total sales $ of product *y* to customers of the state *x*. If the Age is set to an age group *z*, then the displayed number is the aggregate of product *y* sales to customers of the state *x* that belong to the chosen age group *z*.
- The rows dropdown menu and the sales filtering options should be defaulted to the options they had when the user clicked “Run Query”.

If the rows are individual customers then a “Next 50 customers” submit button is also displayed. If it is clicked we get a 2-dimensional report that shows the next 50 customers, sorted under the same criteria with above. 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 and the sales filtering options. In effect, the next reports are produced with the same options that the original one was produced.