A Dynamic Weighted Algorithm for an E-Commerce Recommendation System

Milestone 3 (Final Report)

Andrew Gillard
05021219
22 April 2009

Supervisors:
Nathan Thomas
Duncan McPhee

E-commerce product recommendation systems are becoming increasingly popular and important for online store owners. However they all suffer from a major flaw: purchasing a gift for someone else from a store that uses such a recommendation system will cause the recommendations to be polluted by products that do not appeal to the customer. This project addresses this problem by extending such a recommendation system to allow customers to mark purchases as being gifts, thus solving the problem of having polluted recommendation lists.

Statement of Originality

STATEMENT OF ORIGINALITY

SCHOOL OF COMPUTING

DEGREE SCHEME IN COMPUTING
LEVEL THREE PROJECT

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerised sources have been fully referenced, and I fully realise the consequences of plagiarising any of these sources.

Student Name (Printed)	Andrew Gillard
Student Signature	……..
Registered Scheme of Study	BSc Computer Studies
Date of Signing	22 April 2009

Table of Contents

Introduction and Objectives

Hypothesis and Aim

There are two main problems that this project will aim to address:

1. E-commerce websites that examine a customer's previous purchases in order to recommend related products to the customer suffer from a fundamental flaw: when someone buys an item as a gift, that item is included for future recommendations. Unless they buy a lot of gifts for that same person, having products that are related to that gift recommended to them is unhelpful and distracting. In fact, that customer is then less likely to make further purchases based upon those recommendations because the useful results – i.e. products likely to appeal to the customer – get ignored by the recommendation algorithm in favour of products likely to appeal to the gift's recipient.

2. Choosing a gift for someone that they will actually appreciate can be difficult. It can be especially complicated if the purchaser is not aware of which items the recipient already possesses, or if the full extent of the recipient's interests is unknown.

The hypothesis, therefore, is that the effectiveness of e-commerce product recommendation systems can be vastly improved, solving the above two problems in the process.

Objectives

- Create a basic, mock e-commerce website with a backend database containing a selection of sample products and customers for development and demonstration purposes. This website will allow "customers" to mark their "purchases" as being either for themselves or as a gift and, if marked as a gift, let them indicate the recipient.
- Research available Web development technologies (Web servers, server-side languages, database management systems, etc.) to determine the most appropriate for this project.
- Research existing recommendation algorithms and systems, including Amazon.com's.
- Develop an algorithm that recommends products from a database to a customer based upon the customer's past purchases.
- Extend the above algorithm to exclude purchases marked as being gifts.
- Extend the algorithm a second time to include gift purchases from others for the customer.
- Add weightings to the algorithm in order to prioritise purchases by customers for themselves over gift purchases from others for them.
- Allow customers to rate products they purchase and gifts that they receive in order to further enhance the aforementioned algorithm weighting.
- Add a method for a customer to find suitable gifts for another customer using that second customer's recommendation data.

Test the finished system by populating the database with a lot of randomly generated data in order to accurately evaluate the performance and effectiveness of the new algorithm.

Project Plan

The stages and scale of each stage of this project are expected to be as follows:

- Research – 5 weeks
- Research evaluation – 1 week
- Milestone 1 preparation – 1 week
- Design – 3½ weeks
- Development – 10 (-1) weeks
- Milestone 2 preparation – 1 week (part way through the "Development" stage)
- Evaluation methodology selection – 2 weeks
- Evaluation report – 2 weeks
- Conclusions – 2 weeks
- Milestone 3 (final report) preparation – 2 weeks

*See also "*Appendix III – Gantt Chart*" for a Gantt chart.*

Research

E-Commerce Introduction

E-commerce – "e-" being a common prefix for Internet or computer-related terms to differentiate them from their traditional versions – is a term applied to online stores: shops selling products to customers through a website instead of (or in addition to) a traditional "bricks-and-mortar" store. These stores have been growing in popularity over the past decade or so, since around the time of the Dot-com Bubble, and provide customers with a convenient way of browsing, comparing and purchasing products from the comfort of their own home. Purchasing products from e-commerce stores tends to also be cheaper than buying them from high street stores; while customers often have to pay delivery charges, the lower product prices due to the significantly decreased overheads involved in not having to maintain several hundred high street stores around the country more than compensate.

One of the major benefits that e-commerce offers, at least as far as retailers are concerned, is that each customer's purchasing and browsing habits can be recorded and used to generate a list of products likely to appeal to that customer. These recommended products often generate additional sales and has even been shown to increase customer loyalty (Senecal & Nantel, 2004).

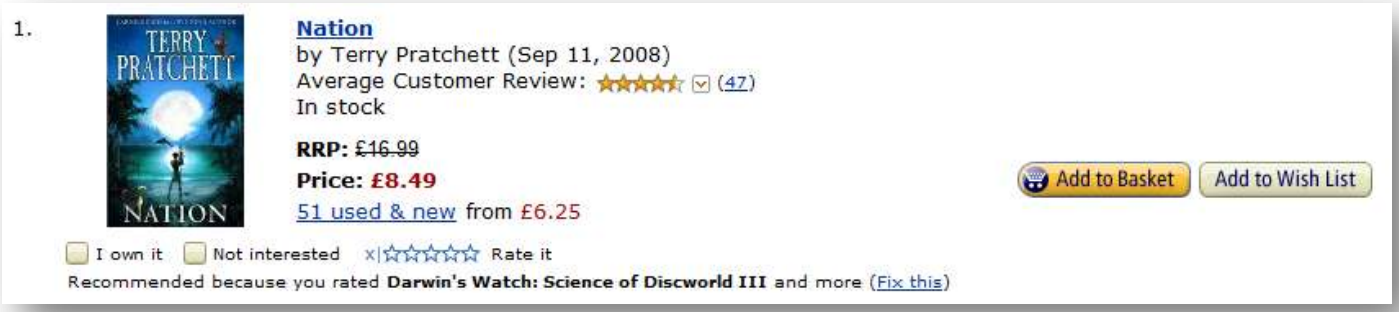


Figure 1 – Example of Amazon.com's Product Recommendation

Existing Traditional Product Recommendation Systems

There exist three methods generally used in recommendation systems – specifically for product recommendation, but they can be used in related systems as well. Collaborative filtering and cluster models aim to find customers similar to the current user so that products purchased or rated by those other customers can be recommended to the current user, while content-based methods attempt to find similar products instead of similar customers (Linden, Smith, & York, 2003).

Collaborative Filtering

Collaborative filtering works by generating, for each customer, a vector with a dimension for each product in the catalogue. The value of each vector component is a number representing whether that product is considered "positive" by that customer; for example a purchase or a positive rating would result in a positive value, while a negative rating would be a negative value. The resulting vectors are then compared with the vector for the current user, and products rated highly by

customers considered to be most similar to the current user are recommended. Optionally, the values of a product in the vector can be multiplied by the inverse of the number of purchasers of that product in order to compensate for very popular items (Linden, Smith, & York, 2003).

The main benefit to using collaborative filtering instead of a content-based method is that it can work with no human interaction – i.e. no product-specific comparison values such as keywords need to be defined – and can even operate in systems where specific comparison data is impossible to define (Ziegler, Lausen, & Schmidt-Thieme, 2004).

The algorithm used to compare two vectors would be something like this (Linden, Smith, & York, 2003):

$$similarity(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

This defines the similarity between two vectors as being equal to the cosine of the angle between them – i.e. the dot product of the vectors, divided by the product of the vectors' lengths. The length of a vector being given by Pythagoras' theorem:

$$\|\vec{A}\| = \sqrt{{A_1}^2 + {A_2}^2 + \cdots + {A_n}^2}$$

For example, if we decide that product ratings can be between one and five stars, represented as vector values -2 to +2, purchases are +1 (they may not necessarily be entirely happy with a purchase, so a five-star review should be considered a better indicator of the customer's interests than a purchase alone) and non-purchased products remain at 0, a very small shop with only five products – *P1* to *P5* – might have customers' vectors – *C1* to *C3* – defined thus:

$$C1 = (0,0,2,0,1)$$
$$C2 = (0,1,-1,0,0)$$
$$C3 = (-2,1,0,0,2)$$

If we add the current user's vector:

$$U = (-1,1,0,0,0)$$

We can calculate the similarity between the other customers and the current user (who would also be amongst the list of customer vectors, but comparing the user to himself is unproductive, so ignored here):

$$similarity(\vec{U}, \overrightarrow{C1}) = \frac{\vec{U} \cdot \overrightarrow{C1}}{\|\vec{U}\| * \|\overrightarrow{C1}\|} = \frac{0}{\sqrt{-1^2 + 1^2 + 0^2 + 0^2 + 0^2} * \sqrt{0^2 + 0^2 + 2^2 + 0^2 + 1^2}}$$
$$= \frac{0}{3.162} = 0$$

$$similarity(\vec{U}, \overrightarrow{C2}) = \frac{\vec{U} \cdot \overrightarrow{C2}}{\|\vec{U}\| * \|\overrightarrow{C2}\|} = \frac{1}{\sqrt{-1^2 + 1^2 + 0^2 + 0^2 + 0^2} * \sqrt{0^2 + 1^2 + -1^2 + 0^2 + 0^2}}$$
$$= \frac{1}{2} = 0.5$$

$$similarity(\vec{U}, \overrightarrow{C3}) = \frac{\vec{U} \cdot \overrightarrow{C3}}{\|\vec{U}\| * \|\overrightarrow{C3}\|} = \frac{3}{\sqrt{-1^2 + 1^2 + 0^2 + 0^2 + 0^2} * \sqrt{-2^2 + 1^2 + 0^2 + 0^2 + 0^2}}$$
$$= \frac{3}{3.162} = 0.949$$

This shows that the current user is most similar to our third customer, C3, because the similarity value is highest for those two customers. This is this case because they have both negatively rated P1, and positively rated (or bought) P2. C3 has also positively rated P5, so that is a good product to recommend to our current user.

This method generates high quality recommendations, however it is extremely expensive in terms of processor time (Linden, Smith, & York, 2003) to the point of being infeasible for any store with more than a couple of thousand products or customers. Additionally, none of the calculation can be done offline and merely referenced by the main website process because such calculations would have to store the result of every customer's vector compared to every other customer's vector, and the data would become stale as soon as any customer purchased or rated a product.

Cluster Models

Cluster models attempt to resolve the computation time issues of collaborative filtering by running a process offline that groups customers into "clusters" – which can be generated either manually or automatically – so that the online process merely has to match the current user to a cluster, then recommend products that are highly rated by other customers in that cluster (Linden, Smith, & York, 2003). The calculation used to group customers together can be largely the same as the calculation used for collaborative filtering.

The problem with this method is that once the online algorithm has determined which cluster the current user fits into the best, it is unaware which of the customers within that cluster most closely match the current user, and so the recommendations generated will be of lower quality (Linden, Smith, & York, 2003). This can be improved by increasing the number of clusters so that the number of customers within each cluster is reduced, but the more clusters that exist, the more work the online algorithm has to do. As such, high quality results with cluster models aren't necessarily any faster to generate than high quality results with collaborative filtering methods.

Content Based Methods

Content-based methods aim to find links between similar products by comparing pre-defined data such as keywords or book authors. They can then recommend similar products by examining the current user's purchases and ratings and then recommending products with similar search data (Linden, Smith, & York, 2003).

A major problem with content-based methods is that comparison data has to be manually defined by a human operator for every product in the catalogue (Lawrence, Almasi, Kotlyar, & Duri, 2001). This can be an extremely time-consuming process, as well as being quite complex. If the comparison data is too precise, such as the author of a book, recommendation groups will be very small (i.e. all books by that author), and only highly related items will be recommended, which doesn't allow customers to find new products they might like outside of a very specific area (Linden, Smith, & York, 2003). However, if the comparison data is too vague, huge numbers of products will be considered to be similar to each other, and the recommendations generated will be irrelevant and may as well have been randomly chosen.

Amazon.com's Product Recommendation System

Amazon.com uses a product recommendation method that they call "Item-to-Item Collaborative Filtering" that is much the same as normal collaborative filtering, however it matches the current user's purchased and rated items to similar items, instead of matching the current user to similar customers, the end result being that their algorithm "scales to massive data sets and produces high-quality recommendations in real time" (Linden, Smith, & York, 2003).

Methodology

Like with cluster methods, Amazon's algorithm has an offline component that performs most of the computationally heavy work, and an online component that just displays the results. This ensures that website page loads are very quick without compromising on recommendation quality.

The offline component of Amazon's system builds a database table of "similar items" by looking at which items customers tend to purchase together. The algorithm used to calculate the similarity between one product and all others can be represented in pseudo-code as (Linden, Smith, & York, 2003):

```
For each item in product catalog, I1
      For each customer C who purchased I1
            For each item I2 purchased by customer C
                  Record that a customer purchased I1 and I2
      For each item I2
            Compute the similarity between I1 and I2
```

Computing the similarity between two items, as in the last line of that pseudo-code, is achieved with the same equation used for traditional collaborative filtering and discussed earlier, with a key difference being that the each vector represents an item instead of a customer, and the vector's dimensions represent the customers who have purchased or rated that item (Linden, Smith, & York, 2003).

For example, where, with standard collaborative filtering we had vectors representing customers and their ratings and purchases of items like this:

$$C1 = (I1, I2, I3, I4, I5)$$

We now have vectors representing items and the rating or purchase values of each customer:

$$I1 = (C1, C2, C3, C4, C5)$$

The "similar items" table can then store the similarity value – as calculated by the cosine method used for collaborative filtering – for each pair of products in the catalogue. While this calculation a very time-consuming task, it is run in the background where the end-users won't notice it, and it isn't very time-sensitive, so it doesn't matter too much if it is only run every few days.

The online component – i.e. the method used to display a list of recommended products to a user on request – is very quick to execute. It merely has to, for each of the current user's purchased or positively-rated products, search the "similar items" table for other products with high similarity values and return a list of recommended products based upon the result, excluding products that the user has already purchased.

Advantages

The main advantage of this method – indeed, the very reason for its creation – is that it scales to very large data sets, with hundreds of thousands of customers and products. The reason for this is that the speed of the online component of the algorithm depends only on the number of items that the current user has rated or purchased (it has to perform a database lookup for each of those ratings and purchases), so the catalogue size and total number of customers doesn't affect it at all (Linden, Smith, & York, 2003).

Additionally, because it is based upon the high-quality collaborative filtering algorithm, recommendation quality remains excellent. And lastly, since Item-to-Item Collaborative Filtering compares items directly and only performs calculations specific to the current user in the online component of the algorithm, any changes in the customer's purchase or rating history will be reflected immediately in their recommendations.

Limitations and Potential Improvements

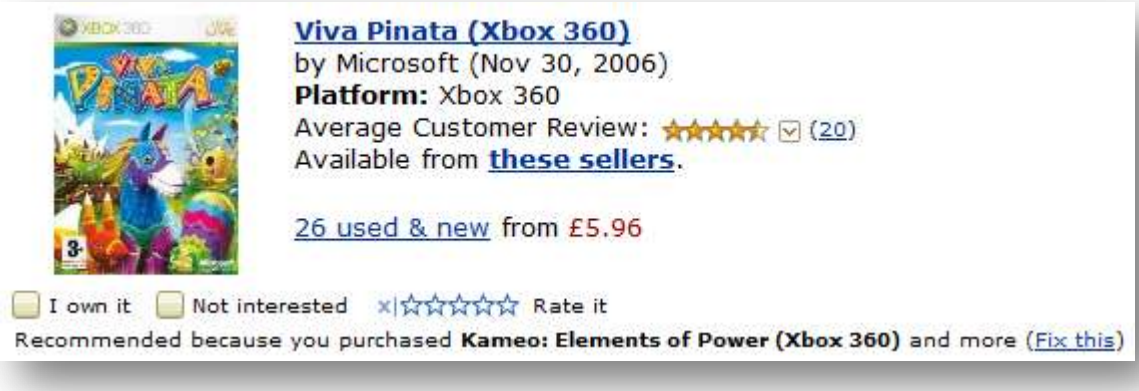


Figure 2 – Another Example of Amazon.com's Product Recommendation

The biggest limitation for the purposes of this project is that purchasing items from Amazon.com as a gift for someone else results in that purchase being used to populate your recommendations list. Until recently, the only solution to this problem was to tick the "Not interested" box (see Figure 2)

for each of the recommended products related to the gift purchase. Obviously this could be a fairly extensive task.

More recently, Amazon have added a feature to their recommendations listing that allows customers to prevent certain items – purchases or ratings – from being used to generate recommendations. The user can click the "Fix this" link next to recommendations, which causes a window to be displayed (see Figure 3) that lists each of the purchases and ratings that cause that product to be recommended. From there, the "Use to make recommendations" checkbox can be unchecked for any of those purchases and ratings.

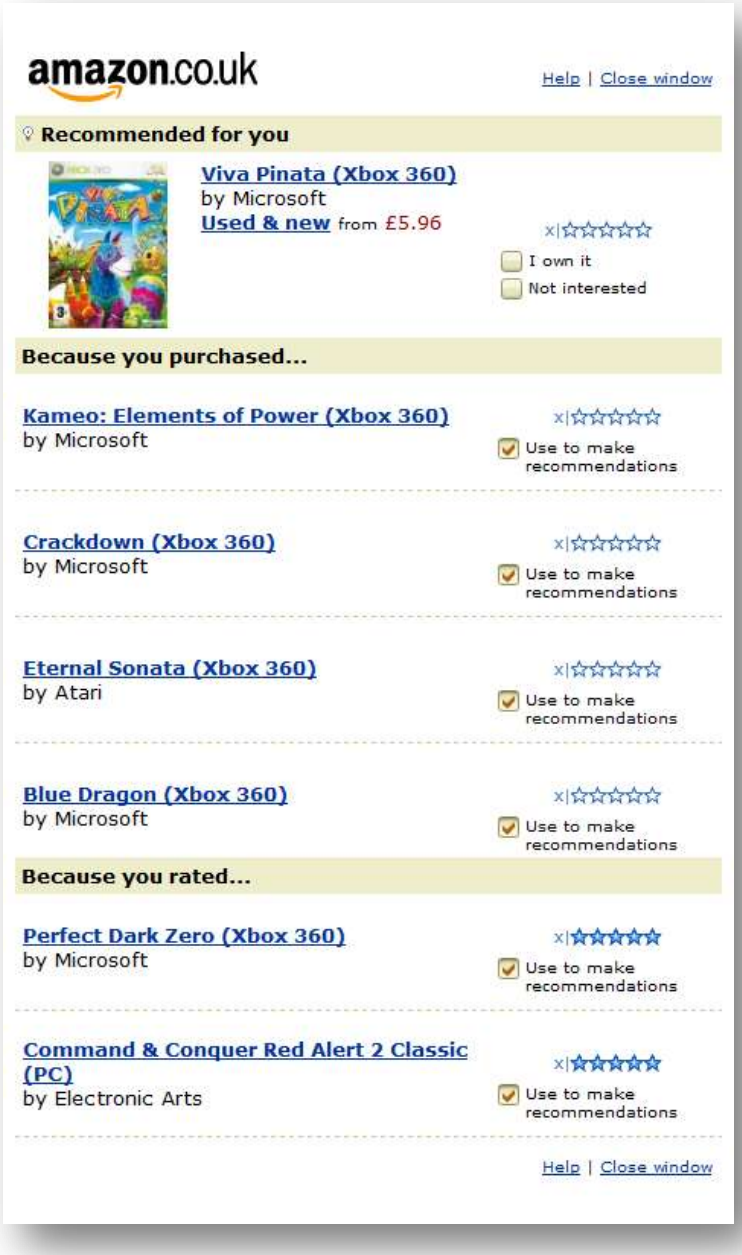


Figure 3 – Amazon.com's "Fix This" Dialog Window

This solves the immediate problem, as gift purchases can simply be removed from generating recommendations; however this project aims to improve upon this by allowing purchases to be marked as a gift for a specific person, and thus those gifts can be used to generate further recommendations for the gift recipient. Additionally, a customer looking to purchase a gift for another customer could make use of the recommendation data for the intended recipient in order to increase the chances of the gift being appropriate.

Obviously there are some major privacy issues that need to be addressed, such as revealing customers' purchase history to other customers, but given the popularity of social networking sites (Alexa, 2008 (1)) (Alexa, 2008 (2)) through which a lot of people routinely share large aspects of their lives, this may not be as big a problem as it initially appears to be. As long as what data will be made available and to whom is displayed clearly, and customers can choose whether or not to allow that data to be exposed – with it preferably being an opt-in service rather than opt-out to ensure that customers explicitly give their permission and know exactly what they're agreeing to – the system should be accepted by users. It should, of course, be an opt-in service, rather than opt-out, to ensure that

customers explicitly give their permission to having their purchases and ratings being made available and a situation similar to the controversy surrounding Facebook's Beacon system (BBC, 2007) is avoided.

Appropriate Web Development Technologies

While it does not matter so much for a prototype, as will be developed for this project, ideally any new software product will be developed using technologies that are the most prevalent in the field, while remaining appropriate. Obviously, using an inappropriate technology (such as a programming language designed for a totally different purpose) simply because it is the most popular is inadvisable, but choosing tools that already enjoy widespread deployment will help ensure that any new software developed has the best possible chance of being accepted and used in industry. It may be possible to sway the platform choices of new businesses; however established businesses are likely to be reluctant to switch existing systems to a new platform, or to buy new servers to run the new software. Additionally, businesses tend to prefer tried-and-tested solutions over new systems that have yet to prove themselves in industry; a new programming language or database management system that programmers like to experiment with in their free time won't necessarily get the same enthusiasm from company managers.

Web Server

The web server is the server software that client browsers (also known as user agents), such as Microsoft Internet Explorer or Mozilla Firefox, communicate with directly. By themselves they can only usually provide the user with static content such as images or very basic, unchanging web pages. However, by linking them to other programs or scripts they can perform much more complex tasks and return highly dynamic web sites.

Sometimes this can take the form of the web server directly executing programs on the server machine, such as .exe files ("applications") in Windows environments. On Unix and Linux systems, shell scripts – text files containing lists of instructions that are marked as being "executable" and thus allowing users to run them like any other program – are commonly used to create dynamic web pages, with Perl being one of the most popular languages to write these scripts in. For extremely high-traffic websites, languages such as C or C++ are used, with the resulting compiled program providing much higher performance than interpreted languages like Perl, at the expense of the greatly increased time needed to make changes to the application.

Luckily, for the purposes of this project, the main web servers are largely compatible with each other: if a website runs on one, it is likely to run on the others. Where one has a popular feature missing in another – such as Apache's mod_rewrite extension – alternatives providing the same functionality to other servers are usually available, such as ISAPI_Rewrite (Helicon Tech, 2008).

As such, the choice of web server to use is almost irrelevant, and is simply a matter of compatibility with programming languages and existing market share.

Netcraft, a UK company specialising in Internet security, research and analysis, reports that, as of October 2008, the open source Apache web server has a market share of 50.43%, while Microsoft's flagship web server Internet Information Services (IIS) has a 34.41% market share (Netcraft, 2008).

Meanwhile, Google shows Apache as having a 66% market share compared to IIS' share of 23% (Modadugu, 2007), though that data was from the first half of 2007, so is considerably older than Netcraft's report. Of note is that both the Netcraft and Google data show that the combined total of Apache's and IIS' market share is 89%. As such, it is probably safe to develop for both IIS and Apache, as that covers the vast majority of the Web, and other web servers should run the developed software as well.

Programming Language

As detailed above, the programming language is what allows the web server to produce and return dynamic web pages. Choice of programming language is probably the most important choice to make regarding the technology to use, and while it should be suited to the task at hand, choosing a language with which the development team is already familiar has obvious benefits.

The popular programming language choices for Web development in businesses are PHP, ASP (and ASP.NET), Java Server Pages and ColdFusion. It is difficult to quantify this, as many sites mask their backend system's details for security reasons, but, generally speaking, sites with URLs ending ".jsp" are using Java Server Pages, those ending ".asp" are using "classic" ASP, those ending ".aspx" are using ASP.NET, with ".php" sites running PHP and ".cfm" sites running ColdFusion. It is theoretically possible to detect the server and programming languages used by a site by examining the "Server" and "X-Powered-By" HTTP headers returned by the web server, but these are often spoofed or suppressed entirely, again for security reasons. For example, Ebuyer's "Server" headers were, for some time, claiming that their site was running on decades-old machines such as Commodore 64s and ZX Spectrums (Modine, 2007).

ASP and ASP.NET only properly run on Windows servers, vastly reducing their potential usage. There are efforts to port them to other operating systems, but they always lag behind the official Windows implementation.

ColdFusion requires an expensive licence for non-academic use (Adobe, 2008), unlike the other options which are either entirely free (PHP and Java Server Pages) or effectively free (ASP and ASP.NET, which are free for machines that are already running Windows).

PHP is free, open source and runs on a wide variety of operating systems (PHP Group, 2008), and now that full object orientation features have been added to it as of PHP 5.2 (PHP Group, 2006) it is much easier to develop large, scalable and maintainable applications in PHP.

Database Management System (DBMS)

MySQL claims to have a high market share (MySQL AB, 2008 (2)), at least on a par with the two main alternatives – Microsoft SQL Server and Oracle DB – which, given that MySQL is a free and open source product, makes it an excellent choice, assuming that it has a roughly equal feature set and performance to the competition. Of course, that claim is from MySQL themselves and on their own website, so, despite being a study by a third part, the possibility of it being biased is fairly high.

MySQL is available for a wide range of operating systems, including Microsoft Windows, Linux, Apple Mac OS X, Sun Solaris and FreeBSD (MySQL AB, 2008 (3)), and the code quality of MySQL is claimed to be considerably better than the "industry average" (MySQL AB, 2008 (1)).

The incredibly popular community-written encyclopaedia Wikipedia uses MySQL as its backend database management system, claiming a query rate of over 25,000 queries per second (MySQL AB, 2008 (4)), so it would appear to be able to scale to massive data sets and huge numbers of simultaneous users. Additionally, Google, who Netcraft indicate power more than 5% of the entire Web (Netcraft, 2008), make extensive use of MySQL, customising it for their specific needs, and they believe that it is better than the commercial alternatives (Xooglers, 2005).

Finally, the commercial rivals to MySQL, Microsoft SQL Server (Microsoft, 2008) and Oracle DB (Oracle, 2008), have expensive and often confusing licensing schemes, and require users to purchase licences before use, which would hinder the use of those products – and thus our prototype – by small businesses who may be unable or unwilling to afford these commercial packages.

Operating System

Most of the popular web servers, database management systems and programming languages run on a wide range of operating systems. Indeed, this could be the very reason for their popularity. Alternatively, the fact that the popular technologies are open source, thus allowing the community to port them to any operating system they please, may cause their popularity. Since developers usually want their products to run on as many systems and be used by as many people as possible, they are more likely to choose technologies that run on numerous platforms in order to give their product a good chance at success. Linux is a particularly popular choice of operating system because it is totally open source (i.e. the full source code used to create it is freely available for viewing and modification) and free to use, with no need for expensive licences unless a dedicated support package is required by the server owners.

Conclusion

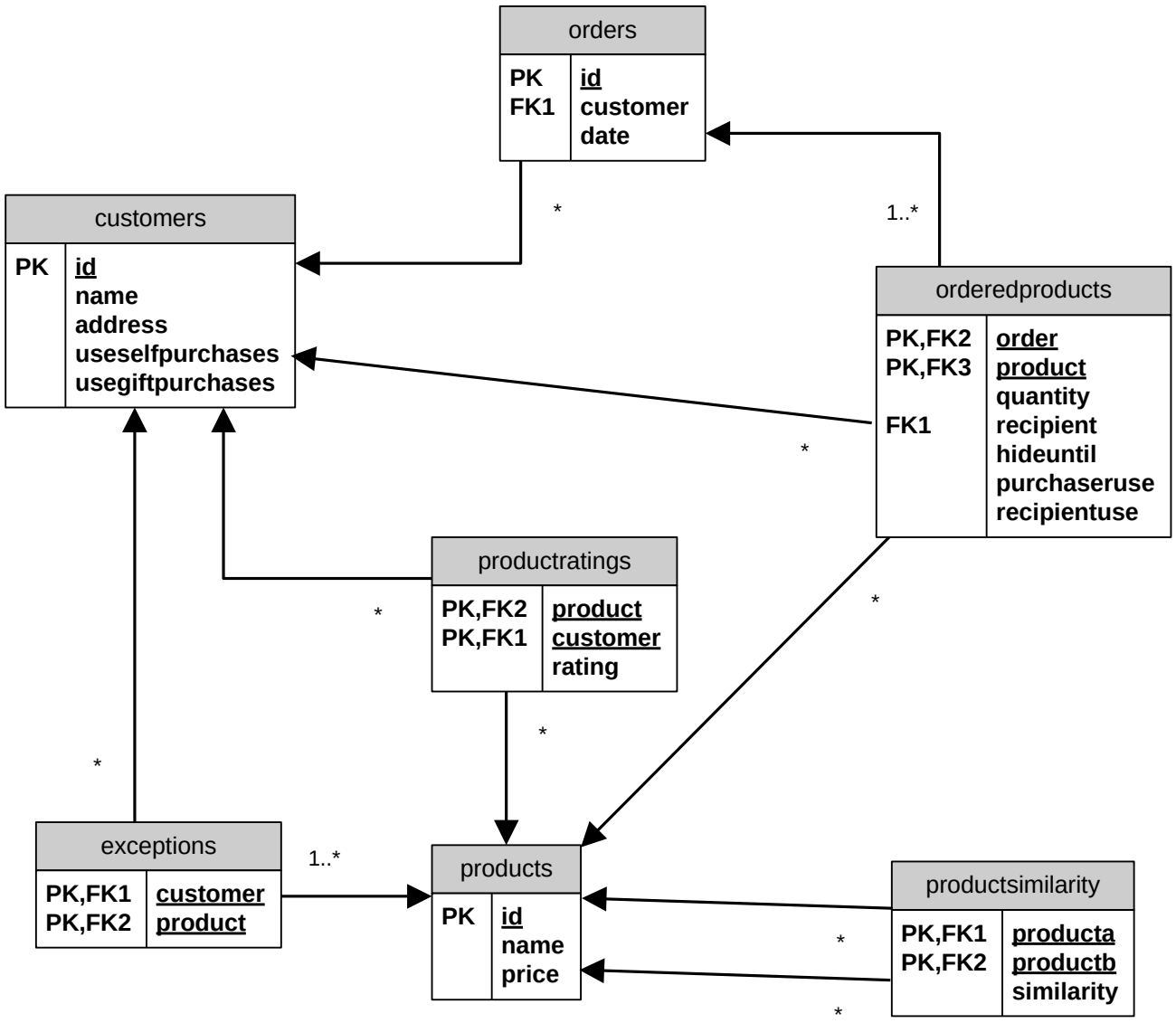
This project will make use of what is known as the LAMP stack of software – i.e. **L**inux as the operating system, **A**pache as the web server, **M**ySQL as the database management system and **P**HP as the programming language. This combination of software enjoys widespread deployment in industry, and all are totally free and open source software products. Alternative software stacks exist, making use of some of the other products discussed above to replace one or more of the LAMP components, but these combinations are less popular and are frequently non-free and/or closed-source. Some of the components of the LAMP stack can be easily swapped out, after the creation of this project's prototype, with very few changes, however for ease of development and testing, this project will focus on LAMP.

Design

Database Design

A good starting point for designing a data-driven system such as this is by designing the database structure. To that end, an entity relationship diagram was produced by examining the system's requirements that had previously been determined in the research.

Entity Relationship Diagram



Further details of each of the fields mentioned above are listed below.

Table and Column Details

orders

Field Name	Type	Unsigned?	Options	Indices
id	Integer	Yes	Auto-Increment	PK
customer	Integer	Yes		FK
date	Datetime			

The `orders` table stores a list of orders placed by customers. Few details need to be stored in this prototype system – simply the time (`date`) at which the order was placed and the `customer` (the customer `id` from the `customers` table) who placed the order. The `id` field is used only to link to records in this table from other tables.

customers

Field Name	Type	Unsigned?	Options	Indices
id	Integer	Yes	Auto-Increment	PK
name	Varchar(128)			
address	Text			

The `customers` table is a simple list of customers registered with the system, including their name and postal address. In this prototype, this list will be static and pre-defined, and there will be no passwords and authentication as it is unnecessary and would make testing and evaluation more time-consuming and difficult, as well as lengthening the development process. The address field will be used to differentiate between customers with similar names when selecting the recipient of gift purchases.

orderedproducts

Field Name	Type	Unsigned?	Options	Indices
order	Integer	Yes		PK, FK
product	Integer	Yes		PK, FK
quantity	Smallint	Yes		
recipient	Integer	Yes	Null	FK
hideuntil	Datetime			
purchaseruse	Boolean			
recipientuse	Boolean			

This table lists each product ordered as part of an order – linking the `products` and `orders` tables together. The number of each item purchased (`quantity`) is stored, as is the `recipient` (a `customer` ID) if the purchase is a gift purchase. If the product has a recipient, a `hideuntil` date can be specified, indicating the date when the purchase can be revealed to the recipient in the list of products used to recommend products to them as the research requires. Boolean fields exist to indicate whether the purchaser (`purchaseruse`) and recipient (`recipientuse`) – if one exists – wish this purchase to be used to recommend further products to them.

products

Field Name	Type	Unsigned?	Options	Indices
id	Integer	Yes	Auto-Increment	PK
name	Varchar(128)			
price	Decimal(10,2)	Yes		

The `products` table stores a list of products, with their `name`s and their `price`s. The `price` column's DECIMAL(10,2) size indicates that it's a decimal number with ten digits, with two of those digits being after the decimal point. As it's an unsigned field, this gives a price range of £0.00 to £99,999,999.99, which will be more than sufficient for this demonstration.

productratings

Field Name	Type	Unsigned?	Options	Indices
product	Integer	Yes		PK, FK
customer	Integer	Yes		PK, FK
rating	Float(3,2)	Yes		

This table stores a list of ratings that customers have made of products. The overall rating of each product will be calculated on the fly for each product when it's needed. In a production environment this value would need to be cached to improve performance.

productsimilarity

Field Name	Type	Unsigned?	Options	Indices
producta	Integer	Yes		PK, FK
productb	Integer	Yes		PK, FK
similarity	Float(10,9)	Yes		

This is the most important table in the system's database: it stores the similarity value for each pair of products. The similarity value is a floating point number from 0.0 to 1.0, with 9 decimal places of accuracy. This range was chosen because it matches the output of the similarity calculating functions detailed in the research.

exceptions

Field Name	Type	Unsigned?	Options	Indices
customer	Integer	Yes		PK, FK
product	Integer	Yes		PK, FK

Customers will be able to indicate that they are not interested in recommended products and have those products removed from recommendations. Such indications will be stored here, in the `exceptions` table.

Relationships

Parent Field	Cardinality	Foreign Key	Cardinality	Referential Action	
				On Update	On Delete
customers.id	1	orders.customer	0..*	Restrict	Restrict
customers.id	1	orderedproducts.recipient	0..*	Restrict	Restrict
customers.id	1	productratings.customer	0..*	Cascade	Cascade
customers.id	1	exceptions.customer	0..*	Cascade	Cascade
orders.id	1	orderedproducts.order	1..*	Restrict	Restrict
products.id	1	exceptions.product	1..*	Cascade	Cascade
products.id	1	productratings.product	0..*	Cascade	Cascade
products.id	1	orderedproducts.product	0..*	Restrict	Restrict
products.id	1	productsimilarity.producta	0..*	Cascade	Cascade
products.id	1	productsimilarity.productb	0..*	Cascade	Cascade

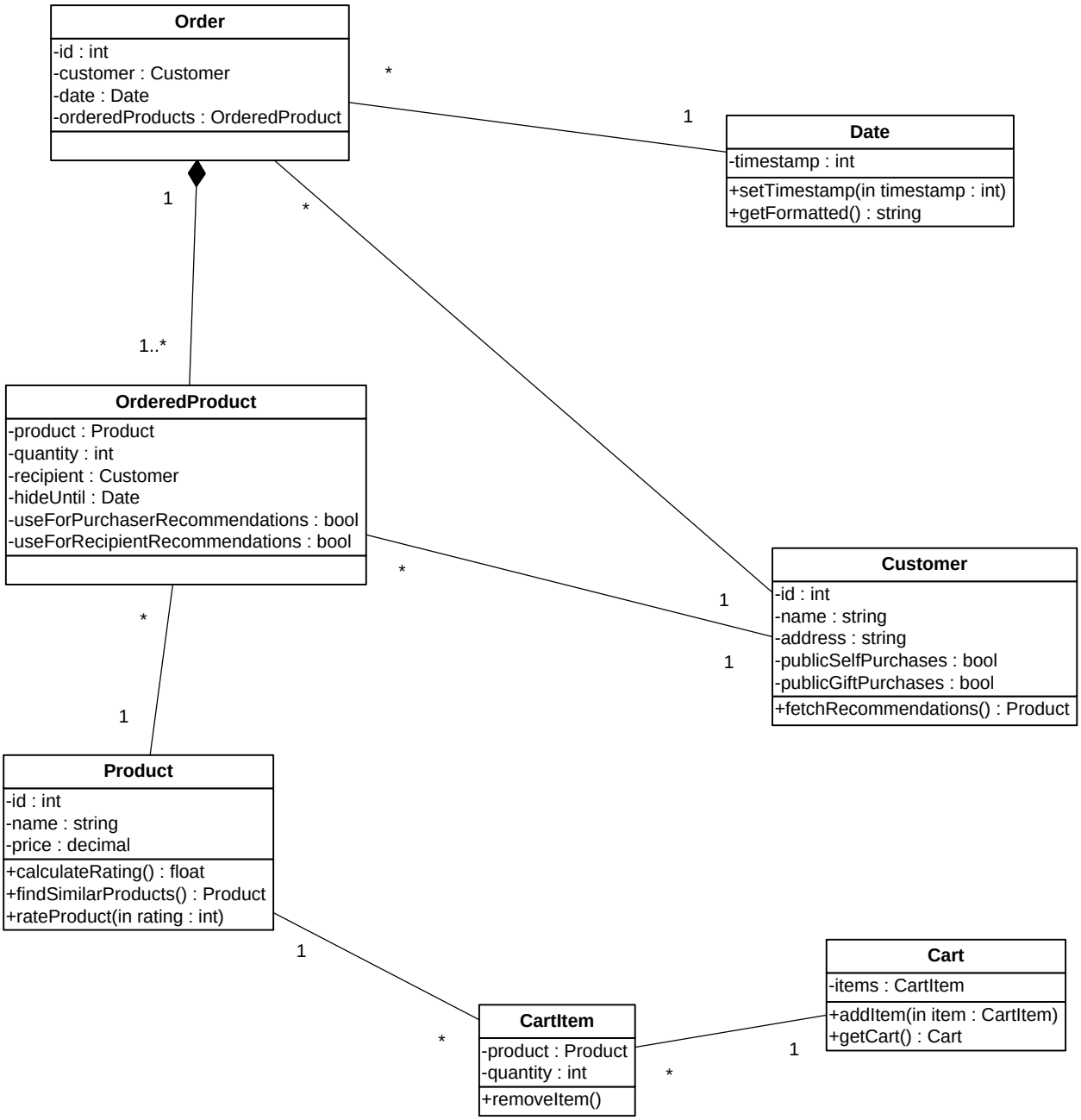
Because this is a MySQL database using the InnoDB engine, proper constrained relationships (foreign keys) can be used, complete with RESTRICT/CASCADE referential actions. This allows fields in tables to be linked and allows us to specify what should happen if rows in one table are deleted while other tables reference those rows (or even if such an action should be allowed). In this system, when a customer is deleted, all of their product ratings and exceptions will be automatically deleted, for example, while customer records cannot be deleted if there are orders placed by that customer in the `orders` table.

Sample Data

To ease the process of creating sample products and customers, as well as the development, testing and evaluation processes, sample data will be copied from third party sources. Products will be taken from the Play.com website (Play.com, 2009), while customer details will be sourced from Terry Pratchett's Discworld series of novels (Pratchett & Briggs, 2004).

Class Diagram

Once an entity-relationship diagram has been created, a class diagram is easy to create by basing it on the entity-relationship diagram. Here is such a diagram showing how the classes in the system will interact with each other.



Algorithm Design

Some parts of this system are complex or important enough that their algorithms should be designed and explained here.

Calculating Product Similarity

The most complex part of the system will be the method that calculates the similarity of each pair of products. This is not surprising, given that it is the main focus of this project.

Building Similar Items Table

The algorithm used to generate similarity values for each pair of products was designed in pseudo-code and was based heavily on the algorithm that Amazon.com use, as was detailed in the research. Converting this algorithm from pseudo-code to real programming code should be a simple process.

```
For each product in catalogue, P1:
      Create empty product array, Pa
      For each customer, C, who purchased P1:
            For each other product, P2, purchased by C:
                  Add P2 to product array Pa
      For each product, P2, in array Pa:
            Compute and store similarity between P1 and P2
```

Calculating Similarity

The "computer similarity" part of the above algorithm needs to be elaborated upon; however it is the same method used by Amazon.com and so this is really just copied from the previous research.

Get vector for P_1: $\overrightarrow{V_1}$
Get vector for P_2: $\overrightarrow{V_2}$
Calculate similarity using:

$$Similarity = \frac{\overrightarrow{V_1} \cdot \overrightarrow{V_2}}{\|\overrightarrow{V_1}\| \times \|\overrightarrow{V_2}\|}$$

Using Stored Procedures

Because the current stable release of MySQL – the database management system chosen for this project – supports stored procedures, this product similarity calculation algorithm will be written and run as a stored procedure. Stored procedures are vastly preferable over writing similar code in a standard application programming language (PHP in this case) for numerous reasons. The most important of these reasons is that they are often much faster than using a separate application, primarily due to a massive reduction in network traffic (Gulutzan, 2005): because MySQL runs all of the code itself, no data has to travel between MySQL and the application – and with this algorithm there will be a very large amount of data to transfer.

There is a disadvantage to using stored procedures, however: they function like a series of SQL statements and are often quite different from traditional programming languages. There are usually alternatives to missing traditional features, but they can require a lot of research to learn. The *MySQL 5.0 Stored Procedures* document by Peter Gulutzan (Gulutzan, 2005) is very useful for this purpose.

One of the biggest problems with using stored procedures for this algorithm is that they do not support arrays in their traditional sense. Instead, one has to create temporary database tables and fill those tables with the array contents. In some ways, this is easier than using arrays, as an "`INSERT INTO ... SELECT ...`" SQL query can be used to populate the temporary tables, as explained below.

The creation of the temporary tables needed to store the vector of customers for each product in a pair is simple:

```
CREATE TEMPORARY TABLE vector1 (
  customer INT UNSIGNED NOT NULL,
  rating FLOAT(10,9) UNSIGNED NOT NULL,
  PRIMARY KEY (customer)
);
CREATE TEMPORARY TABLE vector2 (
  customer INT UNSIGNED NOT NULL,
  rating FLOAT(10,9) UNSIGNED NOT NULL,
  PRIMARY KEY (customer)
);
```

Populating Vector Tables

As mentioned above, an "`INSERT INTO ... SELECT ...`" SQL query can be used to populate a vector table with customers. Below is the design of a query that could accomplish this task. It fetches a list of customers from the database that have purchased or rated the product with a specified ID. Ratings will range from -2 to +2, and a purchased but unrated product is given a rating of +1 – i.e. four stars, as indicated in the research.

```
INSERT INTO <vector table> (customer, rating)
  SELECT c.id, IF (rating IS NOT NULL, rating, IF (quantity, 1, 0) )
    FROM customers AS c
    LEFT JOIN productratings AS r ON c.id=r.customer
    LEFT JOIN orders AS o ON c.id=o.customer
    LEFT JOIN orderedproducts AS op ON o.id=op.`order`
    WHERE r.product=<product id> AND op.product=<product id>;
```

Calculating Vector Length

To calculate the length of a vector (one of the temporary tables mentioned above), Pythagorus' theorem is used:

```
SELECT SQRT(SUM(POW(rating, 2))) FROM <vector table>;
```

Calculating Dot Product

And to calculate the dot product of the two vectors, this query is used:

```
SELECT SUM(a.rating * b.rating) FROM <vector table 1> AS a
  LEFT JOIN <vector table 2> AS b ON a.customer=b.customer;
```

Website Design

The design of the website used for the prototype of this system will not be particularly important. The prototype is only going to be used to demonstrate the recommendation algorithm. The algorithm and the rest of the system could be easily integrated into an existing application, thus rendering the design of the prototype's website irrelevant. A reasonable design will still be needed to enhance the demonstration, and to give viewers a better impression of how the system could look in a production environment, however.

Layout

Because this will only be a demonstration prototype, the layout should focus on being clean, simple and easy to use and understand. Spending a lot of time creating images that make the site look "pretty" but without aiding usability would be wasted effort, so should be avoided.

In keeping with the theme of having a simple layout, the prototype's website will have a very plain header across the top of the page just containing a sample store name, which is what research has suggested users expect to see at the top of the page (Lynch & Horton, Page Structure and Site Design, 2009). Down the side of the page will be a sidebar containing navigation links, the login form, and the list of items currently in the shopping cart. Again, users generally expect to see these features down the side of the page (Lynch & Horton, Page Structure and Site Design, 2009). Research suggests that navigation links can be placed at either the left or the right hand side of the page without impeding navigation and that most users expect to see the shopping cart at the far top-right of the page (Lynch & Horton, Page Structure and Site Design, 2009). This prototype, however, will need to list items in the cart, which could be a long list, and should therefore be placed lower in the page in the sidebar. The overall height of the sidebar will not be great, and so the whole of the sidebar should be visible on the user's screen at any time – unless they've scrolled down the page, of course. Positioning the sidebar on the right hand side of the page will also allow the main page content to be more visibly important on the page.

Accessibility

Accessibility in web design is the concept of ensuring that websites can be used fully by as many people as possible, regardless of any disabilities that they may have. Keeping the design simple – as detailed above – will help greatly with the accessibility of the prototype website by keeping the site usable with styles disabled or font sizes increased (Lynch & Horton, Sidebar: Universal Design Principles, 2009).

The other main consideration for accessibility in the prototype website will be in providing "alt-texts" for images – text that is displayed instead of images if they cannot be loaded or have been disabled by the user (Lynch & Horton, Sidebar: Universal Design Principles, 2009). This allows websites to still make sense if the images cannot be seen for whatever reason – and even allows the screen reading software that blind people use to "describe" the image by reading the alt-text. Due to the aforementioned limited use of images on the prototype site, this will not be a large concern, but any images that do get used must be given alt-texts.

Colour Scheme

The colour scheme for the prototype site will be developed using the *Color Scheme Designer 3* application (Stanicek, 2009).

Figure 4 - Color Scheme Designer - Main View

This Web-based application allows colours to be easily selected from a colour wheel and it can automatically choose colours that work well with the selected colour (see Figure 4), showing example website designs that use the selected colour(s) (see Figure 5).

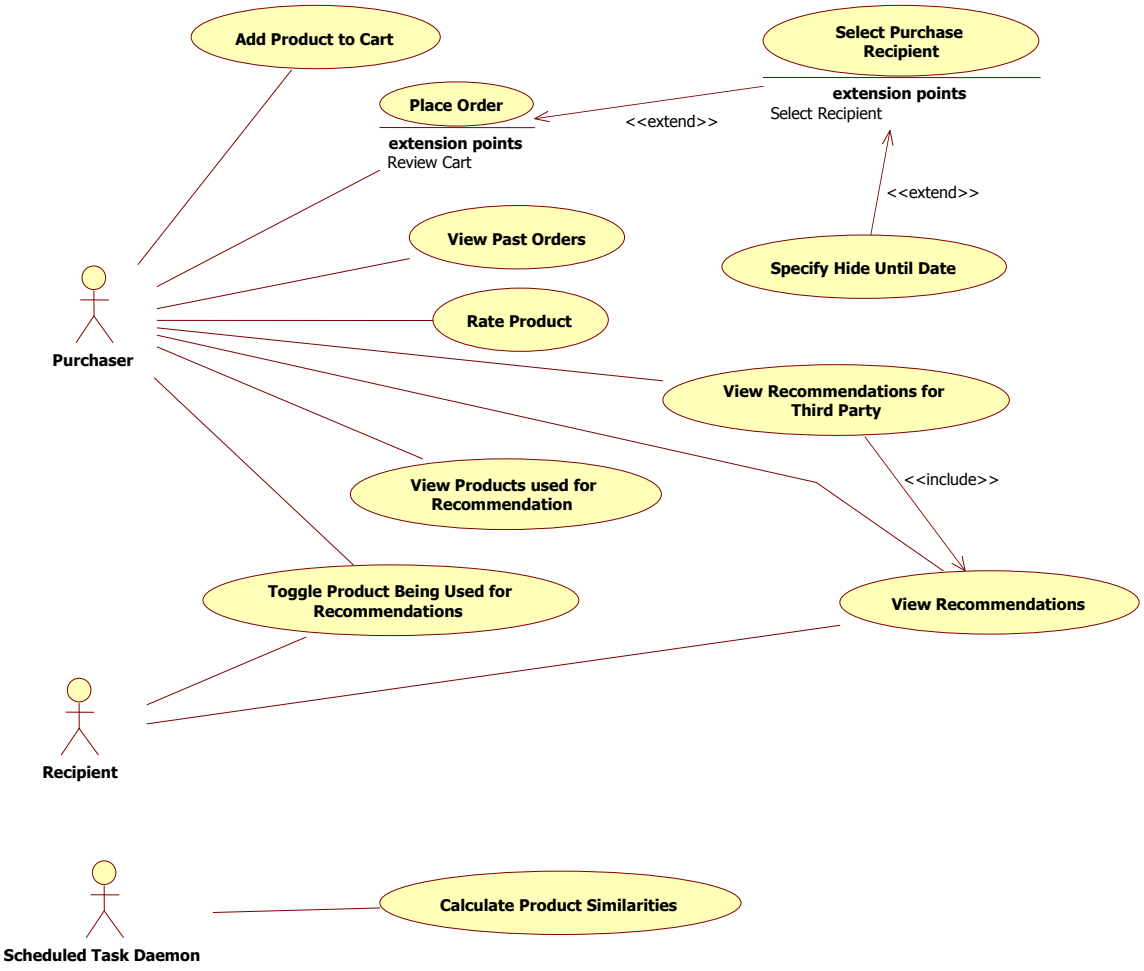
Figure 5 - Color Scheme Designer - Scheme Example

Figure 6 - Color Scheme Designer - Full Colour-Blindness

Additionally it can demonstrate how the colour scheme would appear to people with various visual impairments – different variations of colour-blindness – as well as indicating approximately what percentage of the population suffer from each deficiency (see Figure 6 for an example, showing a simulation of full colour-blindness). This can help ensure that the resulting website is accessible to all users.

Use Case Diagram

To help design how the system will be used, a use case diagram was produced.



Prototype Development

Database

The first stage of development was to create the database by referring to the database design section that was created as part of the design process. Due to the extensive detail that the database was designed with, transferring the design to an implementation was very simple and no problems were encountered. One minor change was made, however: the default value of the `purchaseruse` and `recipientuse` columns in the `orderedproducts` table was set to the integer 1 so that products are used by the recipient & purchaser by default.

Following database creation, the database had to be populated with sample data. As specified in the design documentation above, characters from Terry Pratchett's Discworld novels were used as sample customers, while products on the Play.com website were used for sample products. During the process of populating the `products` table with sample products it was decided to use product images and a way of linking products to their image files needed to be created, so an `image` column was added to the table, containing the filename of each product's image.

Similarity Calculation

Once the database was created and populated, the stored procedure used to calculate the similarity values between products could be produced. A large amount of the process of writing the stored procedure involved researching stored procedure code, primarily in Peter Gulutzan's *MySQL 5.0 Stored Procedures* article.

There were three main problems encountered. The first was in getting a "`SELECT`" query to run inside the reading loop of another – a loop within a loop. This was solved by using a named "`BEGIN`" block for the inner loop.

The second problem was getting the procedure to fail gracefully, in a way that could be detectable by the calling script or client, when an error occurred. This was only ever partially solved: the entire process takes place inside a transaction and when an error occurs in an SQL query a "`ROLLBACK`" query is executed to undo all of the changes. This ensures that the database is always in a valid, stable state, with a fully-populated similarities table, even if an error occurs halfway through the stored procedure – although such an event would leave the table out of date. The calling script or client has no idea that the rollback took place, however.

The final major problem was in debugging logic errors. Due to the nature of stored procedures being fairly "closed", it is difficult to see the values of variables at various stages throughout the procedure. Nor is it possible to echo strings back to the calling client. This problem was overcome by creating a `calcerrs` (calculation errors) table containing a `DATETIME` column and a `TEXT` column, and then using "`INSERT`" queries at important locations throughout the stored procedure to send debugging information to that table. By having that table use the MyISAM storage engine (which doesn't support transactions) the records inserted to it remain even after a "`ROLLBACK`" is executed.

While writing the stored procedure it became apparent that the result of the similarity calculation could be a negative value as well as positive, but the column type used to store the result was an unsigned type, preventing negative values being stored in it. Because of this, the `similarity` column

in the `productsimilarity` table was changed to be a signed type. The temporary "vector" tables used by the stored procedure had to be changed in the same way.

To avoid using too many "cursors" (variables used to access "`SELECT`" query results) – which are cumbersome to write and require a lot of code – in the stored procedure, a new temporary table was added to the stored procedure that contains various calculation results. Having a temporary table like this allows the results of "`SELECT`" queries to be inserted directly into the temporary table.

Figure 7 - Prototype Website Viewed in Opera Mobile 9 Beta on a Windows Mobile 6.1 Device

Despite planning, in the design stage, some of the SQL queries that would be required by the stored procedure, those queries – primarily the queries used to populate the vector tables – had to be extended upon quite significantly in order to achieve the desired result. The vector population queries had to be changed so that they retrieved the correct rating value, because ratings are stored as values from 1 to 5, but the similarity calculation uses values from -2 to +2. Additionally the same queries had an extra "`WHERE`" clause added to make them respect the `exceptions` table, and their "`LEFT JOIN`"s of the `orders` and `orderedproducts` tables were changed to a single join to the results of a subquery joining those same two tables but with a "`WHERE`" clause to get the query to respect the `purchaseruse` and `recipientuse` columns of the `orderedproducts` tables.

Also changed from the design stage were the queries used for calculating the vectors' lengths and dot products; though these queries were only changed so that they used the `tempcalcresults` (temporary calculation results) temporary table rather than local variables.

The only remaining non-trivial part of the similarity calculation stored procedure is in the SQL query that retrieves a list of other products that were also purchased by each customer who purchased the first product in each pair – i.e. the "`For each other product, P2, purchased by C`" line in the pseudo-code detailed in the design. This query had to fetch a list of rated products and ordered products that had been rated or bought by a specific customer (it is run for each customer that purchased or rated each product in the database) in order to find out who rated or bought each product besides the initial customer in the iteration. It also had to respect the contents of the `exceptions` table. The easiest and clearest way of doing this was to use a "`UNION`" query, with one half of the union selecting rated products and the other half selecting purchased products. The result is a fairly long query – and one that is arguably slightly redundant in its "`WHERE`" clauses – but a query that is much easier to understand and modify than the equivalent query written without a "`UNION`".

Website

Front-End Layout and Style

The client-side code was written in XHTML 1.0 Strict, although in "HTML compatibility mode" in order to remain compatible with certain older and less-compliant browsers (notably Internet Explorer 6, 7 and 8 to varying degrees (Wilson, 2005)). XHTML enforces neat code that is easy for computer programs to parse and understand, and also allows faster page rendering – especially on low-power devices such as mobile phones (see Figure 7).

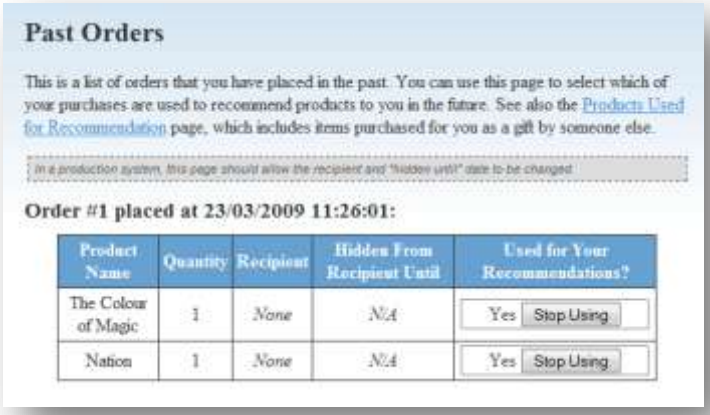


Figure 8 - Past Orders Page

As specified in the design stage, the layout is simple, with a plain header bar across the top of the screen and a sidebar down the right hand side, containing navigation links, the drop-down list of customers to allow the user to change who they are logged in as, and the list of items in the cart. There are only three images – all very simple: a gradient for the background image of the page, a zig-zag pattern image used for horizontal lines, and a circle used as the bullet image in lists. The styling in the Cascading Style Sheet (CSS) is similarly basic and serves only to make the demonstration system easy to navigate and use, and to provide a basic design to avoid everything being rendered in the "ugly" monochromatic HTML default styles given by Web browsers.

No client-side scripting (e.g. JavaScript) was required, due to the simple nature of the prototype website.

Figure 9 - Recommendation Products Page

Website Pages

Home / Product List

This page simply lists all of the products in the database, showing their details and providing forms for rating each product or adding it to the user's cart. See also Figure 7.

Past Orders

The Past Orders page lists each order placed by the currently logged in user, and the products purchased as part of each of those orders. There is also a button to toggle whether each purchase is used to recommend products. See Figure 8.

Recommendation Products

This page lists all of the products that could be used for recommending products. This includes products purchased by the current user, those rated by the current user, and those purchased for the current

user by another customer. There is then a button to toggle whether each product is actually used for recommendations or not. See Figure 9.

Recommendations

This page serves two purposes. The first is to show the products that have been recommended to the current user based on their purchases, received gifts and ratings. However it also allows the user to select another customer and see products that are recommended to them in order to allow the user to select an appropriate product for a gift. See Figure 10.

Calculate Product Similarities

This is a page that wouldn't normally exist in a standard website as it calls the MySQL stored procedure that calculates the similarities between pairs of products; however for this demonstration it was useful to be able to re-calculate the product similarities directly on the website without resorting to a MySQL command line. See Figure 11.

Checkout

The checkout page displays the products that are in the user's cart, allows the user to select a recipient for each product if they want, and allows a date to be specified that the product will be hidden from the recipient until. When the checkout form is submitted, a new order is created containing all of the cart's products. See Figure 12.

Figure 10 - Recommendations Page

Prototype Shortcut Notices

Throughout the website are grey boxes containing notices indicating where "shortcuts" were taken in the development of the prototype for the purposes of achieving a working demonstration as quickly and easily as possible, and in order to aid testing and evaluation (see Figure 13). For ease, these are repeated below, with a more detailed explanation.

In the section of the sidebar used for logging in

> *"Note that in a production system, this would be a standard login dialog, with username and password boxes, etc."*

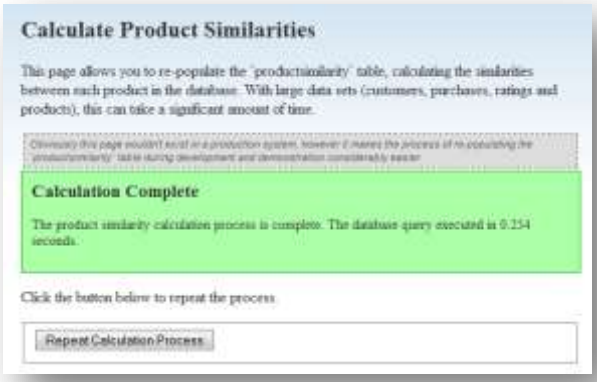


Figure 11 - Calculate Product Similarities Page

Developing a full authentication system, with usernames, passwords and other security features was unnecessary for this demonstration site. It would also have frustrated people using the demonstration for evaluation purposes, and would have slowed down development and testing. As such, a drop-down box listing all of the customers in the database was used, and the user is allowed to log in as any user on demand with the minimum of hassle.

Above the list of products on the home page

> *"In a production system there would be product categories and only a subsection of the full product list would be displayed on the home page. Each product would also have its own page, displaying a full description, reviews, etc. Such features are unnecessary for a demonstration prototype like this."*

This message was included because the list of products is only available on the home page and simply consists of a table displaying the image, name, price and rating of each product, along with a text box in which a quantity can be entered and an "add to cart" button. Obviously this isn't the way that e-commerce sites display products; normally there would be categories, different ways of searching for products, and each product would get its own page with all of its details on it, such as a full description and reviews of the product by other customers. These features are not necessary for this demonstration and so were omitted.

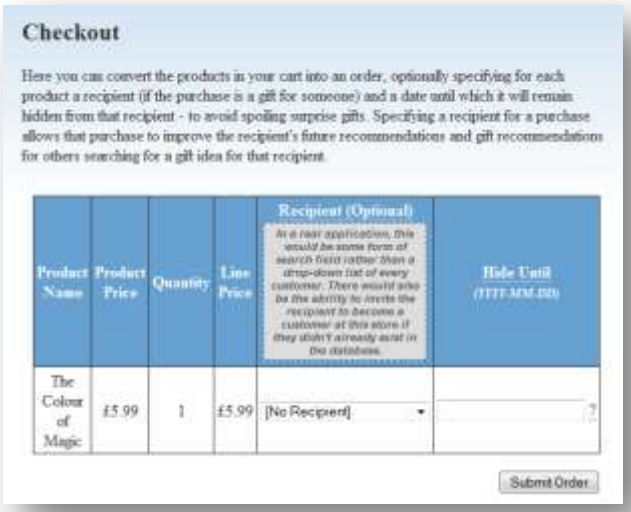


Figure 12 - Checkout Page

On the page listing previously-placed orders

> *"In a production system, this page should allow the recipient and "hidden until" date to be changed."*

While the list of previously placed orders displays the "hide until" date that was specified by the customer when the order was placed, it does not allow the date to be changed. In a real system, it would be useful for the customer to be able to change this value, but for an example system it would merely be a distraction and unnecessary work.

Below the form allowing users to find recommendations for a third party

> *"Again, this would be a search form, rather than a drop-down list of customers, in a real system."*

The form that allows users to find recommended products for other customers contains a drop-down list of customers to choose from, much like the login form. On a real site with potentially thousands of customers this would not be an effective way of handling this feature, not only due to the obvious speed and network transfer issues of sending the full list of customers to the user, but also because providing a list of every customer would violate the privacy of those customers.

Use of a drop-down list of customers in this situation also makes the `address` field of the `customers` database table redundant.

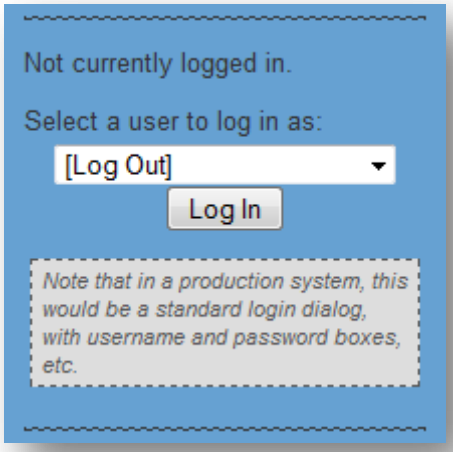


Figure 13 - A "Prototype Shortcut" Notice Box

On the page used to call the "calculatesimilarities" stored procedure

> *"Obviously this page wouldn't exist in a production system, however it makes the process of re-populating the `productsimilarity` table during development and demonstration considerably easier."*

The demonstration site contains a page that runs the "calculatesimilarities" stored procedure to re-calculate the similarity of each pair of products. In a production site this page would not exist, and that stored procedure would be run by a scheduled task (on Windows systems) or a cron job (on Unix/Linux systems), however waiting for such an automated system to update the similarities would be unrealistic and frustrating when testing and evaluating the system. As such, a manual method of calling the stored procedure was created instead.

With the "Recipient" box on the checkout page

> *"In a real application, this would be some form of search field rather than a drop-down list of every customer. There would also be the ability to invite the recipient to become a customer at this store if they didn't already exist in the database."*

This box is used to select which customer the purchase is a gift for, if it is a gift. For the sake of simplicity in the demonstration – and to make evaluation, development and testing easier – the box is a drop-down list of all customers, but as with the two previous examples of customer drop-downs, this would normally be a text box which could be used to search for a specific customer by name. This box would also normally allow the user to invite the recipient to become a registered customer of the store if they weren't already registered.

Below the Cart box in the sidebar

> *"A production system would have more controls (adjusting quantity, etc.), and possibly a full page for viewing the cart."*

The cart box in this demonstration is very basic, merely allowing the user to remove individual items from the cart and proceed to the checkout. There is no page dedicated to displaying the contents of the cart. Normal stores would allow users to adjust the quantity of each product in the cart and potentially other options, too.

Server-Side Code Structure

In order to keep the code tidy and maintainable, no global variables were used and only a very limited number of custom functions were written outside of classes. Almost all of the code exists within classes, limiting the possibility of conflicts occurring if it was combined with a third party library. Most errors are handled using exceptions to further promote manageable code. All page loads are processed by `index.php`, which includes various files from the `/inc/blocks/` directory for each section of the sidebar, as well as a file from the `/inc/pages/` directory for the main body of the page which varies depending on which page was requested (via a "`page`" GET argument in the URL).

Database access was provided by a set of custom classes that wrap the functionality of PHP's `MySQLi` (MySQL Improved) extension to make it easier to use. Of note is that the result of "`SELECT`" queries executed through the database classes is a class that implements PHP's "`Iterator`" interface, allowing the returned object to be passed to the `foreach()` construct, again leading to neater code. These classes also provide parameterised queries – i.e. variables that need to get passed to the database are passed to the database classes as separate variables rather than being embedded inside SQL query strings. This negates the risk of SQL injection vulnerabilities. Various class objects can be passed directly to the database classes in their native object forms, with the database class calling the object's `__toMySQL()` method automatically.

All dates throughout the website are handled using the custom `Date` class which, much like the database classes, wraps PHP's `DateTime` class, providing additional functionality and making the existing functionality easier to use. All dates are stored in the database as UTC (aka GMT) and are converted to the local time zone (set to "Europe/London" – i.e. GMT or BST depending on the time of year – by default) by the `Date` class prior to being displayed.

There is a `Formatting` class which provides formatting functions to the system – although there is only one method: `formatPrice()`, which, as expected, formats a price by prefixing a pound sign (£), rounding the number to two decimal places, ensuring that there are two decimal places (even if the price doesn't require two decimal places for display), and adding "," characters to separate groups of thousands, if necessary.

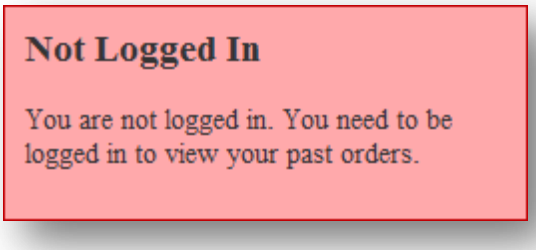


Figure 14 - A Standard Error Message Box

There is a class dedicated to validating user input: `Validation`. This class contains various methods that can be used to validate user input, such as ensuring that a value was supplied, that it is numeric or that it matches a given (Perl-compatible) regular expression. Input validation is an aspect of programming that many developers dislike, so making it as quick and easy as possible is advantageous by increasing the chance that proper validation will take place.

The `Utility` class provides methods for retrieving `GET`, `POST`, `SERVER` and `COOKIE` values while specifying a default value in case the requested variable doesn't exist. This reduces the need to write complex structures such as:

```
$foo = (isset($_GET['bar']) ? $_GET['bar'] : 'baz');
```

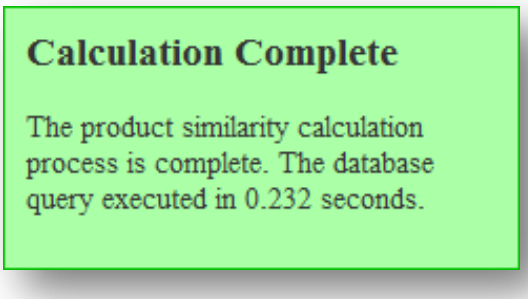


Figure 15 - A Standard Success Message Box

There are also two functions that are used to display "Success" and "Error" messages – simple messages inside coloured boxes that provide a standard way of displaying errors and success messages to the user (see Figure 14 and Figure 15).

Customers (aka users) can be represented as a `User` object, and the `Users` class provides methods for retrieving lists of customers, determining whether the current user is logged in, logging the user in as a specific customer and fetching a specific customer's `User` object.

The contents of the user's cart is handled by the `Cart` class, with each item represented using a `CartItem` object. The `Cart` class stores the contents as a PHP session variable that persists for six hours – a time that is reset every time the user loads a page, so the cart remains for six hours of inactivity. The `Cart` class also allows the current cart to be converted to an order – a process that uses a database transaction to prevent inconsistent data being stored in the database if, for example, the script stopped executing after inserting only some of the cart's contents into the `orderedproducts` table.

The `Orders` class provides a method for fetching a list of all orders placed by a specified customer. This will be returned as an array of `Order` objects. The `Order` class encapsulates the details of a single order, including all of the products ordered in that order – an array of `OrderedProduct` objects. `OrderedProduct` objects contain all the details of the product ordered, the recipient of the purchase if one was specified, and allows the "recipientuse" and "purchaseruse" (whether the purchase should be used to generate recommendations for the recipient and/or purchaser) flags to be toggled. `OrderedProduct` objects can also return an instance of the `Product` class to allow general product details to be obtained.

The final group of classes is those related to products. The `Products` class retrieves lists of products with various properties (such as being recommended to a specific customer, or simply a list of all products in the database), and also has methods for rating products. The `Product` class has attributes to contain all of the various properties of products, such as their name, price and rating, and has a method to determine if the current user has rated that product. `SimilarProduct` is a class that extends `Product`, adding a "similarity" property so that the list of recommended products can display the recommendation's similarity rating (how similar it is to a purchase or rating that the customer has made). Similarly, `RatedProduct` consists of a `Product` object and a rating for that product, and has methods for toggling whether the product is used for generating recommendations.

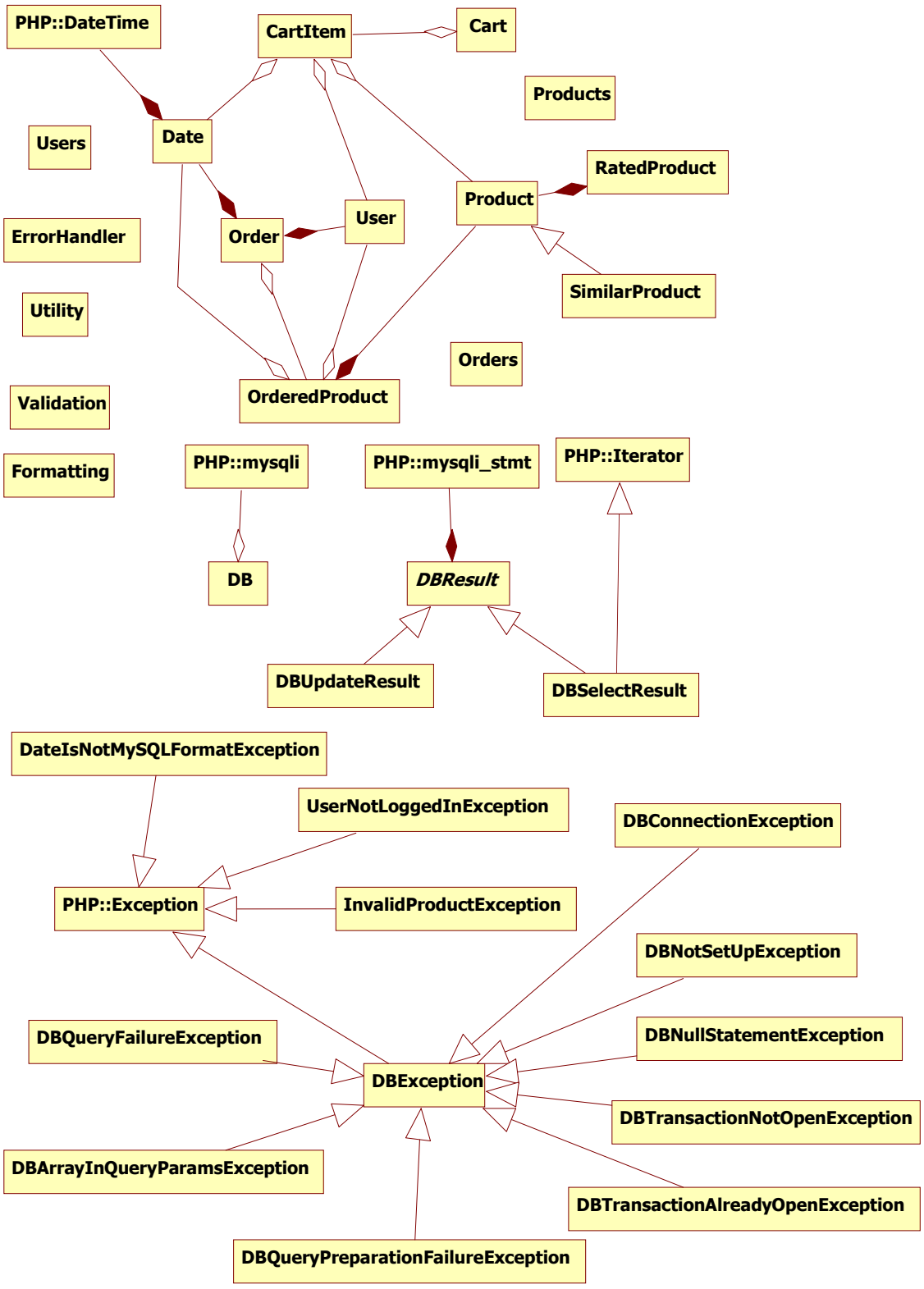
The method in the `Products` class that returns a list of recommended products makes use of a "`VIEW`" in the database that wasn't planning in the planning stage and which combines all of the columns of the `products` table with a calculated rating and count of the number of rating for each product in the `products` table. This allows the SQL query that fetches recommended products to be

somewhat simpler – although it still consists of a "`UNION`" of three separate queries and a total of ten SQL queries when "`UNION`"s and "`JOIN`"s are counted. The recommendation-generating query weights the similarity values given to recommendations so that products purchased by the current customer are given a higher priority than those merely rated and those purchased for the current user.

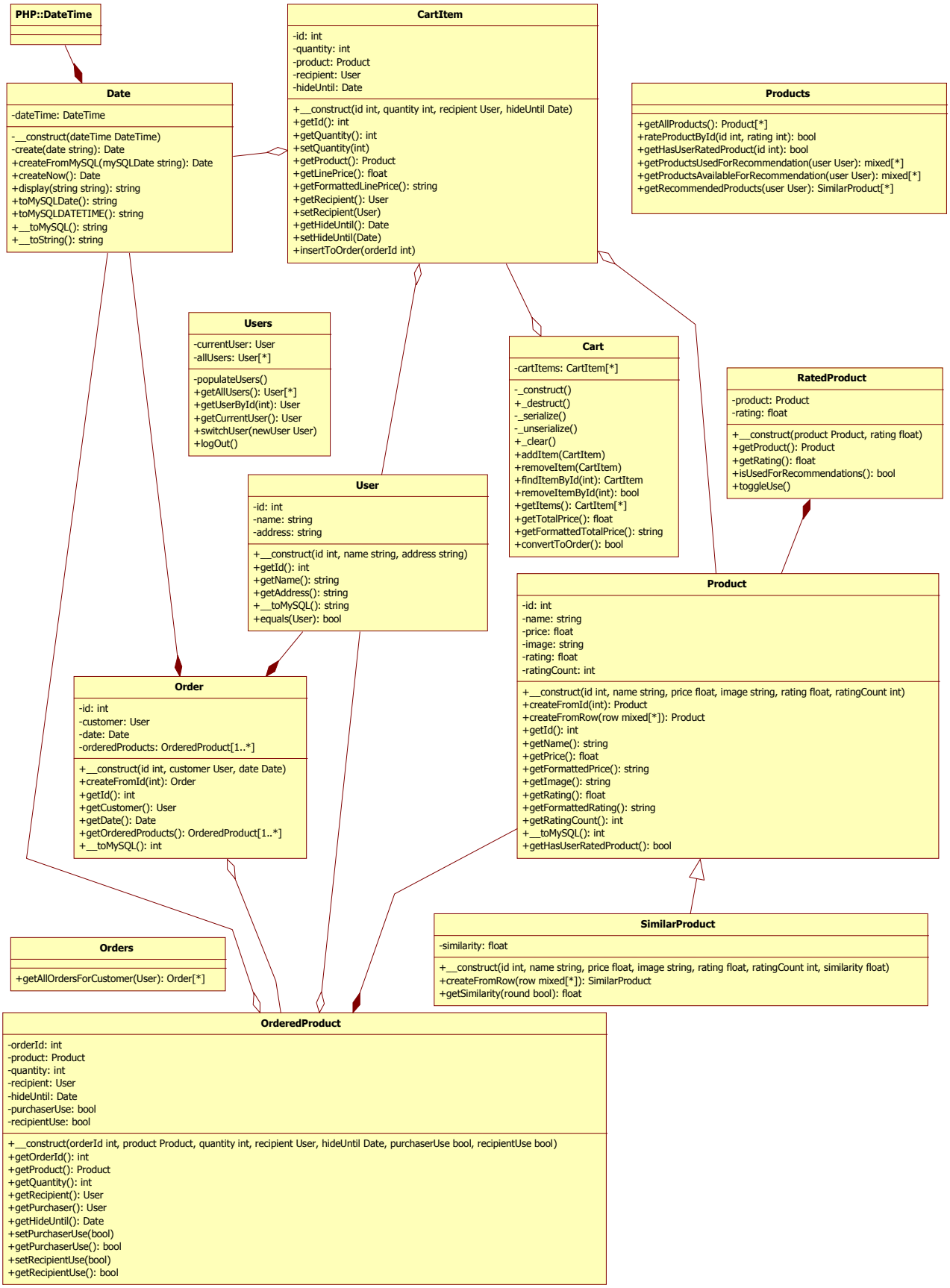
New Class Diagram

During the development of the prototype, the system of classes changed considerably. Below are updated class diagrams – the first is an overview of all of the classes, with the later three being detailed versions.

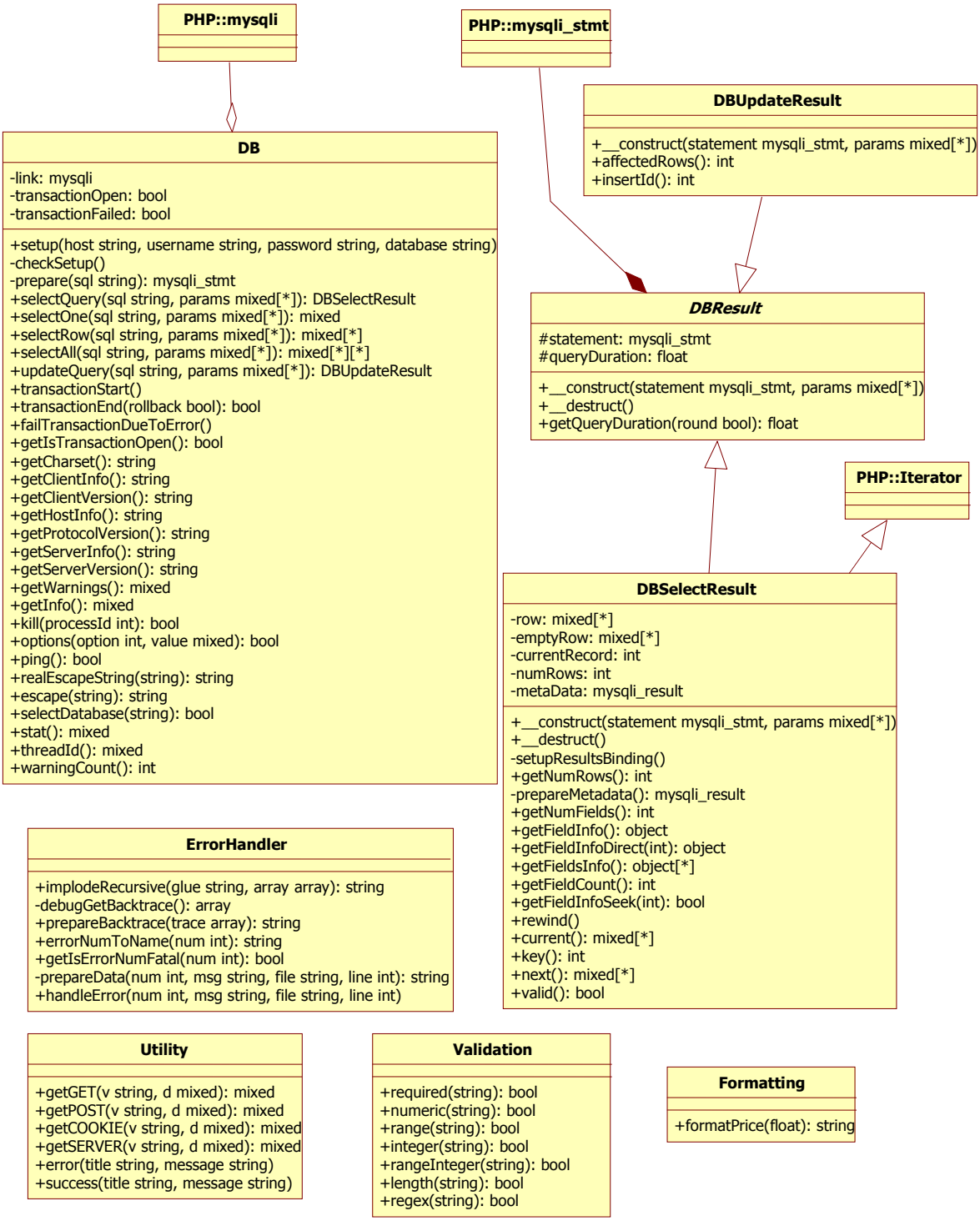
Overview



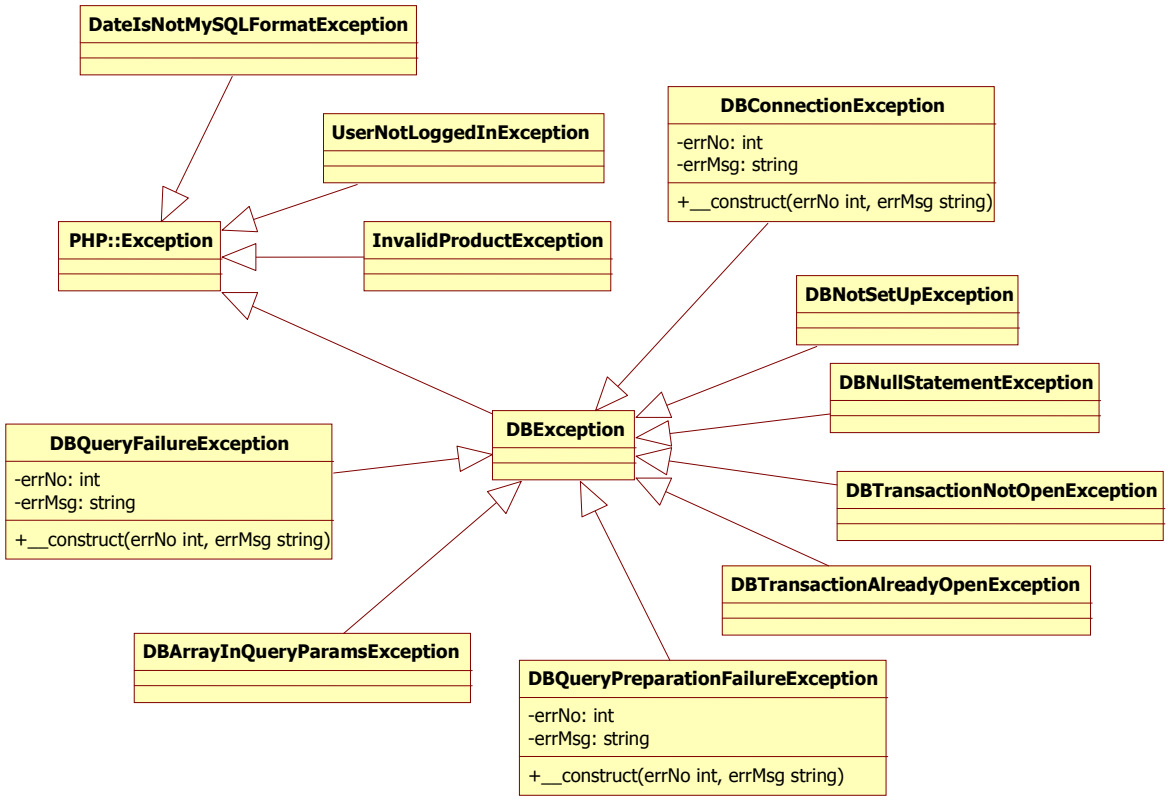
Detailed Part 1



Detailed Part 2



Detailed Part 3



Evaluation

Due to the issues mentioned in Appendix V – Diary, there was insufficient time for a full and proper evaluation to take place. Because this is a very user-orientated project – i.e. it focuses on solving a problem in a very user-obvious part of e-commerce systems – it would have been very valuable for the evaluation to be performed by a group of typical users of such a system. This could have been in the form of a questionnaire distributed to users after allowing them to use the system for a short period of time along with an explanation of the system. The questionnaire would have asked the users whether they considered it to be an improvement over existing systems, in what specific ways it improved upon existing systems and whether they found any problems with it.

Instead of performing a standard, full evaluation, the system's code and user interface was reviewed thoroughly, with any problems found being reported in detail below. This allowed the system to be evaluated to some degree, and have its flaws discovered and discussed.

Issues Discovered During Documentation of Development Process

Due to the extremely rushed development process which took place in around a fifth of the originally intended amount of time (see Appendix V – Diary) there are a number of problems with the developed prototype solution. Few of them impact upon the ability of the prototype system to demonstrate a working algorithm, and none of them affect it enough to break the demonstration entirely. These problems were all discovered while re-reading the application code and testing the resulting website in order to document the development process.

The first of these issues were discovered in the MySQL stored procedure that is used to generate the similarity calculations between products. There is currently no method for determining if the stored procedure failed in some way. The entire procedure takes place inside a transaction with a `ROLLBACK` command being executed if an error occurs, so there is no obvious way of detecting an error: the database table remains unchanged and the call to the procedure itself seems to succeed, as far as the script calling it is concerned. This could potentially be fixed by creating a new database table that tracks calls to the procedure. If the new table used the MyISAM table engine rather than InnoDB, it would be unaffected by transactions, so any entries in it would remain even after a `ROLLBACK` command. This could be used to keep a history of calls to the procedure and whether or not they succeeded.

The other problem with the stored procedure is in a "`DELETE`" query towards the end. It deletes all entries in the `productsimilarity` table where the similarity is calculated as 0. The original intention of this was to remove entries where the two products were calculated to be not similar at all (under the assumption that a zero-similarity means no similarity at all). This was implemented before it was realised that calculation results could be negative, however. Given that the range of similarity values is now -1 to +1, it seems unlikely that a value of 0 indicates no similarity. Therefore this "`DELETE`" query should probably be removed or altered somewhat.

After discovering problems with the stored procedure, the database structure was reviewed, ensuring that every database table and column made sense for its purpose, was using an appropriate data type, and was actually used.

The first database structure problem is with the primary key on both the `order` and `product` columns in the `orderedproducts` table. For most situations this will work fine, however it prevents someone ordering the same product multiple times for different people in the same order. This could be easily fixed by giving the `orderedproducts` table its own auto-incrementing `id` column primary key instead, but the PHP code for handling the cart would need to be significantly changed as well. This problem doesn't really affect the ability to demonstrate the algorithm using this application, so fixing it is low priority.

The `rating` column in the `productratings` table uses the "`FLOAT`" data type, however it is only ever used to store integers (values 1-5) that could otherwise fit in a `TINYINT` column. Storing tiny integers as floats not only uses more storage space (four bytes for a float instead of one byte for a tiny int) but probably also uses more processing time to handle, and is certainly not the "proper" choice of data type for this data.

The `purchaseruse` column of the `orderedproducts` table doesn't serve a useful purpose. It was originally intended to indicate whether the person who purchased a product wanted to use the product to recommend other products, even when the product was purchased as a gift for a third party; however the chance that anyone would ever want this to actually be the case is very slim. After all, the purchase is being used as a gift for someone else, so how often would someone find that product useful for themselves as well? If that situation did ever occur, it would require very little effort for the purchaser to also rate the product, thereby adding it to their product list. The option could be better served with a flag to indicate if the recipient (the selected recipient if the purchase is a gift or the buyer if the purchase isn't a gift) wants the item to be used to recommend further products – i.e. removing the `purchaseruse` column and leaving only `recipientuse`.

The multiple methods for determining whether purchased and rated products should be used to generate recommendations leads to a confusing situation where different methods are used in different parts of the system. There are the `recipientuse` and `purchaseruse` columns in the `orderedproducts` table and also the `exceptions` table. The stored procedure and the recommendation-fetching PHP code don't agree on how these two indicators should be used, primarily due to a lack of planning in that regard. The PHP code nearly always uses both methods, however the stored procedure often doesn't. The main problem is that there are two reasons why people might not want certain products to be used: firstly because they are no longer interested in a specific product, and secondly because recommended products aren't as appropriate as they ought to be. There's also the issue that it should be possible to remove recommended products in addition to purchased and rated products. Currently this is not possible and probably should have been the use of the `exceptions` table. Again, a lack of planning and database documentation (i.e. fully documenting what each specific field and table was supposed to do) caused this oversight. This issue will definitely affect the quality of recommendations, however it doesn't detract from the algorithm's main concept: the algorithm still works, but its implementation needs tweaking quite considerably.

As mentioned previously in the Development section of this report, the `address` column of the `customers` database table is not being used because searching for customers (e.g. to specify a recipient of a purchase) is being achieved through the use of a drop-down box listing all customers. The original plan was to provide a customer search form that allowed users to search for a specific

customer by providing their name and address, thus solving the problem of multiple customers having the same name. This idea was dismissed in order to keep the development process simple, however the database column remained with no purpose.

Due to the aforementioned time constraints, there is very little user input validation in the PHP code. Thanks to the prepared statements used for database queries, there shouldn't be any SQL injection vulnerabilities in the system, and the foreign key constraints in the database will prevent a lot of nonsensical data being added to the database (such as a purchase's recipient being set to a non-existent customer). However there could be places in the system where invalid data can be added. Also, even if the database constraints kept invalid data out of the system, the result would be an unhandled database exception which would cause an "ugly" PHP error message to display on screen. This is arguably not much of a problem, though, because people will normally have to edit the returned HTML or manually craft HTTP requests in order to insert bad data, and displaying an unintelligible error message in these situations could be considered acceptable. An example of bad data being used is that it is possible to enter an alphabetic quantity (such as "a") when adding an item to the cart, which results in "0" amounts of it in the cart, which makes no sense. It is also possible to edit the HTML to give a product a rating of "100" (which was truncated to 9.99 by MySQL due to the column type), and to attempt to log in as a non-existent user, resulting in a NULL object being passed to a method expecting a User object and therefore a PHP error was thrown.

Unlike Amazon.com's list of recommended products, the same list in this system doesn't show the user why each product was recommended – i.e. which products they bought or rated in order to trigger each recommendation (compare Figure 1 with Figure 10). This would be a simple addition with a database query that looked, for each recommendation, for similar products that had been purchased or rated.

Additionally there is no way for users to indicate on the list of recommended products that they already own a recommendation. Doing so would allow the recommendation algorithm to select products to recommend based upon products that the user's indicated that they already own, but weren't purchased or rated at the store in question. One way of solving this would be to add a button entitled "I own this" to the recommendation list that, when clicked, rates the product with a four-star review – the same value given internally by the stored procedure to products that were purchased but not rated – unless, of course, it had already been rated separately. This solution would leave each such product with a mysterious 4-star rating that the customer might not expect, though, so an alternative solution where a list of owned-but-not-purchased products is stored separately could be desired instead.

The final point, which isn't actually a problem, but instead a potential improvement, is that the similarity weightings used to weight rated products higher than purchases could theoretically be changed to be dynamic on a per-user basis. These weightings are implemented inside the huge database query in the Products class, rather than the stored procedure, so each customer could easily select their own weightings which get applied to the similarity values here and thus affect the returned recommendations.

Conclusions

There were a number of objectives set at the beginning of this project. In turn, these were:

> *Create a basic, mock e-commerce website with a backend database containing a selection of sample products and customers for development and demonstration purposes. This website will allow "customers" to mark their "purchases" as being either for themselves or as a gift and, if marked as a gift, let them indicate the recipient.*

This was fully achieved as demonstrated by Figure 7 to Figure 12.

> *Research available Web development technologies (Web servers, server-side languages, database management systems, etc.) to determine the most appropriate for this project.*

This was covered in the research section. Numerous technologies were researched, and a combination of PHP and MySQL, with any suitable operating system and web server, being chosen (Linux and Apache were used for the demonstration site).

> *Research existing recommendation algorithms and systems, including Amazon.com's.*

Despite initial scepticism that Amazon.com's algorithm would be documented or published anywhere public, details of it were indeed found and researched, as were a number of alternative, older and less-effective algorithms.

> *Develop an algorithm that recommends products from a database to a customer based upon the customer's past purchases.*
>
> *Extend the above algorithm to exclude purchases marked as being gifts.*
>
> *Extend the algorithm a second time to include gift purchases from others for the customer.*
>
> *Add weightings to the algorithm in order to prioritise purchases by customers for themselves over gift purchases from others for them.*
>
> *Allow customers to rate products they purchase and gifts that they receive in order to further enhance the aforementioned algorithm weighting.*
>
> *Add a method for a customer to find suitable gifts for another customer using that second customer's recommendation data.*

The above six objectives all link together and can be commented on as one; they describe creating an algorithm to initially match Amazon.com's, and then extending it to include gift purchases and ratings. The algorithm used in this project's end product was largely a minor extension to Amazon.com's documented algorithm. It was then extended and weighted as required. See the development sections entitled Similarity Calculation and Server-Side Code Structure for further details of the algorithm developed.

The research outcomes required the developed site to allow customers to entirely opt-out of other customers seeing their purchases and received gifts. While it is not possible for any customer to see any other customer's purchases or received gifts directly, there is no opt-out ability that would

prevent others seeing someone's recommendations, which could reveal information about their purchase history. This could be implemented fairly easily, but it would be an important addition to a production version of this system.

Also in the research was speculation as to why Amazon.com might not use this style of algorithm already. One possibility is that the SQL query required to fetch all of the required data is simply too large and complex to be executed on a large set of data such as Amazon's customer and product database. The "`SELECT`" query used to fetch a user's recommended products uses a "`UNION`" that joins three other queries' results, and in total (including subqueries) the query uses nine "`SELECT`" queries. When an "`EXPLAIN`" is run on the query in question, this is the result:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	7	Using where; Using temporary; Using filesort
9	DEPENDENT SUBQUERY	orderedproducts	index_subquery	recipient, product	product	4	func	1	Using where
10	DEPENDENT SUBQUERY	orders	unique_subquery	PRIMARY, customer	PRIMARY	4	func	1	Using where
2	DERIVED	<derived11>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	31	
2	DERIVED	ps	ref	PRIMARY	PRIMARY	4	p.id	1	Using where
11	DERIVED	p	index	*NULL*	PRIMARY	4	*NULL*	31	
11	DERIVED	r	ref	PRIMARY	PRIMARY	4	lvl3.p.id	1	
3	DEPENDENT SUBQUERY	op	ref	PRIMARY, recipient, product	product	4	func	1	Using where
3	DEPENDENT SUBQUERY	o	eq_ref	PRIMARY, customer	PRIMARY	4	lvl3.op.order	1	Using where
4	UNION	<derived12>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	31	
4	UNION	ps	ref	PRIMARY	PRIMARY	4	p.id	1	Using where
12	DERIVED	p	index	*NULL*	PRIMARY	4	*NULL*	31	
12	DERIVED	r	ref	PRIMARY	PRIMARY	4	lvl3.p.id	1	
5	DEPENDENT SUBQUERY	op	ref	recipient, product	recipient	5		1	Using where
5	DEPENDENT SUBQUERY	o	eq_ref	PRIMARY	PRIMARY	4	lvl3.op.order	1	Using where; Using index
6	UNION	<derived13>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	31	
6	UNION	ps	ref	PRIMARY	PRIMARY	4	p.id	1	Using where
13	DERIVED	p	index	*NULL*	PRIMARY	4	*NULL*	31	
13	DERIVED	r	ref	PRIMARY	PRIMARY	4	lvl3.p.id	1	
7	DEPENDENT SUBQUERY	productratings	unique_subquery	PRIMARY, customer	PRIMARY	8	func	1	Using index; Using where
8	DEPENDENT SUBQUERY	exceptions	eq_ref	PRIMARY, product	PRIMARY	8	func	1	Using where; Using index
NULL	UNION RESULT	<union2,4,6>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	

There is a lot of data in this result set, and much of it is not worth worrying about, although the sheer amount of rows returned shows that MySQL has to do a lot of work to execute the query. There is some information that shows that this is a poor and slow-to-execute query, though. In the "type" column, any values of "ALL" or "index" are a bad sign: this column indicates the type of table join being used, and "ALL" and "index" are comparatively slow types of joins (Gilfillan, 2001). There are

five references to “ALL” in the “type” column, and three “index” values. The other particularly bad values to see in the above table are “Using temporary” and “Using filesort” in the “Extra” column (Gilfillan, 2001), both of which are displayed in the first row.

This query could potentially be improved and optimised in order to make it perform better, however it would always be a relatively slow query, and most stores would want to execute it very often – as much as on every single page load, for example, although short-term query caching could become an option if it was being called several times a minute for each customer.

In general, the objectives of this project have been achieved and it shows that the described improvements of Amazon’s recommendation system can be implemented successfully.

Appendix I – Objective Settings Proforma

FINAL YEAR DEGREE PROJECT

Objective Settings Proforma

(to be completed and submitted to your supervisor by Friday 17th October, 2008)

Student's Name: Andrew Gillard

First Assessor: Nathan Thomas

Second Assessor: Duncan McPhee

Project Title:

A Dynamic Weighted Algorithm for an E-Commerce Recommendation System

Project Objectives & Deliverables

- Create a basic, mock e-commerce website with a backend database containing a selection of sample products and customers for development and demonstration purposes. This website will allow "customers" to mark their "purchases" as being either for themselves or as a gift and, if marked as a gift, let them indicate the recipient.
- Research available Web development technologies (Web servers, server-side languages, database management systems, etc.) to determine the most appropriate for this project.
- Research papers on recommendation systems and similar.
- Research as much as is possible of Amazon.com's product recommendation system and alternative systems employed elsewhere.
- Research possibilities of why Amazon.com utilises its current method.
- Develop an algorithm that recommends products from a database to a customer based upon the customer's past purchases.
- Extend the above algorithm to exclude purchases marked as being gifts.
- Extend the algorithm a second time to include gift purchases from others for the customer.
- Add weightings to the algorithm in order to prioritise purchases by customers for themselves over gift purchases from others for them.
- Allow customers to rate products they purchase and gifts that they receive in order to further enhance the aforementioned algorithm weighting.
- Add a method for a customer to find suitable gifts for another customer using that second customer's recommendation data.

Please tick this box to indicate your awareness of the university's policy on ethical issues ✓

The deliverables and objectives can often change due to unforeseen circumstances, or through the student's research causing the project to follow a different path. If this is the case, and the project objectives change significantly, then the first assessor should make a note of the date and fill in a new objectives proforma, which should also be included as an appendix to the project report. The project organiser is to be consulted at this stage.

Appendix II – Agreed Marking Scheme Weightings

Degree Scheme in Computing
FINAL Project Assessment and Comment Form 2008-9

Student Andrew Gillard

Project Title: A Dynamic Weighted Algorithm for an E-Commerce Recommendation System

Supervisor: (1 or 2) Nathan Thomas

Mark Category	Weighting Ranges	Agreed Weighting	Mark Allocated
Project Management *(Only set by Supervisor 1)*	50 – 80		
Originality & Self-Direction	40 – 80		
Technical Complexity	20 – 80		
Solutions, Evaluation & Conclusions	80 – 120		
Final & Sub-Report Quality	50		
Prototype / System Demo Or Project Deliverable	50 – 100		
Sponsor Mark	00 – 60		
Sub-Total Marks	-------		
Sub-Total Percentage	-------	**50%**	%
Milestone 1 **Initial Research**	10%	**10%**	%
Milestone 2 **Continued Research** **Research Applied to Design** **Prototype Development**	20%	**5%** **10%** **5%**	% % %
Final Presentation	-------	**20%**	%
TOTAL PERCENTAGE	-------	**100%**	%

The weightings for the various aspects of the project are to be set at the initial objectives setting stage. The ranges shown are guidelines and the actual weighting set may be outside of these ranges if deemed appropriate.

Appendix III – Gantt Chart

ID	Task Name	Start	Finish	Duration
1	Research	29/09/2008	02/11/2008	5w
2	Initial Presentation	13/10/2008	13/10/2008	0w
3	Research Evaluation	03/11/2008	12/11/2008	1w 3d
4	Milestone 1 Preparation	13/11/2008	19/11/2008	1w
5	Milestone 1	19/11/2008	19/11/2008	0w
6	Design	20/11/2008	14/12/2008	3w 4d
7	Development	15/12/2008	22/02/2009	10w
8	Milestone 2 Preparation	21/01/2009	27/01/2009	1w
9	Milestone 2	28/01/2009	28/01/2009	0w
10	Evaluation Methodology Selection	23/02/2009	06/03/2009	2w
11	Evaluation Report	09/03/2009	22/03/2009	2w
12	Conclusions	23/03/2009	05/04/2009	2w
13	Milestone 3 Preparation	06/04/2009	21/04/2009	2w 2d
14	Milestone 3	22/04/2009	22/04/2009	0w
15	Final Presentation	27/04/2009	27/04/2009	0w

Appendix IV – References

Adobe. (2008). *ColdFusion 8 Purchase Options*. Retrieved November 9, 2008, from Adobe ColdFusion 8: http://www.adobe.com/products/coldfusion/buy/

Alexa. (2008 (1), November 17). *facebook.com*. Retrieved November 17, 2008, from Traffic Details from Alexa: http://www.alexa.com/data/details/traffic_details/facebook.com

Alexa. (2008 (2), November 17). *myspace.com*. Retrieved November 17, 2008, from Traffic Details from Alexa: http://www.alexa.com/data/details/traffic_details/myspace.com

BBC. (2007, December 6). *Facebook founder apology over ads*. Retrieved Novermb 17, 2008, from BBC News | Technology: http://news.bbc.co.uk/1/hi/technology/7130349.stm

Gilfillan, I. (2001, November 26). *Optimising MySQL: Queries and Indexes*. Retrieved April 19, 2009, from Database Journal: http://www.databasejournal.com/features/mysql/article.php/1382791/Optimizing-MySQL-Queries-and-Indexes.htm

Gulutzan, P. (2005, March). *MySQL 5.0 Stored Procedures.* Retrieved April 11, 2009, from MySQL 5.0 New Features Series: http://dev.mysql.com/tech-resources/articles/mysql-storedprocedures.pdf

Helicon Tech. (2008). *ISAPI_Rewrite 3*. Retrieved November 9, 2008, from Helicon Tech - Webserver Enhancements: http://www.helicontech.com/isapi_rewrite/

Lawrence, R. D., Almasi, G. S., Kotlyar, V., & Duri, S. S. (2001). Personalization of Supermarket Product Recommendations. *Data Mining and Knowledge Discovery* , 11-32.

Linden, G., Smith, B., & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* , 76-80.

Lynch, P. J., & Horton, S. (2009). *Page Structure and Site Design*. Retrieved April 11, 2009, from Web Style Guide 3: http://www.webstyleguide.com/wsg3/6-page-structure/3-site-design.html

Lynch, P. J., & Horton, S. (2009). *Sidebar: Universal Design Principles*. Retrieved April 11, 2009, from Web Style Guide 3: http://www.webstyleguide.com/wsg3/2-universal-usability/3-universal-design.html

Microsoft. (2008). *SQL Server 2008 Pricing*. Retrieved November 9, 2008, from SQL Server 2008: http://www.microsoft.com/sqlserver/2008/en/us/pricing.aspx

Modadugu, N. (2007, June 5). *Web Server Software and Malware*. Retrieved November 1, 2008, from Google Online Security Blog: http://googleonlinesecurity.blogspot.com/2007/06/web-server-software-and-malware.html

Modine, A. (2007, June 7). *Ebuyer.com runs on a Commodore 64*. Retrieved November 9, 2008, from The Register: http://www.theregister.co.uk/2007/07/07/ebuyer_runs_site_on_commodore64/

MySQL AB. (2008 (1)). *Code Quality*. Retrieved November 9, 2008, from MySQL: http://www.mysql.com/why-mysql/quality/

MySQL AB. (2008 (2)). *Market Share*. Retrieved November 9, 2008, from MySQL: http://www.mysql.com/why-mysql/marketshare/

MySQL AB. (2008 (3)). *MySQL 5.0 Downloads*. Retrieved November 9, 2008, from MySQL: http://dev.mysql.com/downloads/mysql/5.0.html#downloads

MySQL AB. (2008 (4)). *Scale-Out*. Retrieved November 9, 2008, from MySQL: http://www.mysql.com/why-mysql/scaleout/wikipedia.html

Netcraft. (2008, October 29). *October 2008 Web Server Survey*. Retrieved November 1, 2008, from Netcraft: http://news.netcraft.com/archives/2008/10/29/october_2008_web_server_survey.html

Oracle. (2008, November 3). *Oracle Technology Global Price List.* Retrieved November 9, 2008, from Software Investment Guide: http://www.oracle.com/corporate/pricing/technology-price-list.pdf

PHP Group. (2008, November 7). *Installation and Configuration*. Retrieved November 9, 2008, from PHP Manual: http://www.php.net/manual/en/install.php

PHP Group. (2006, November 2). *PHP 5 ChangeLog*. Retrieved November 9, 2008, from PHP: http://www.php.net/ChangeLog-5.php

Play.com. (2009). *Home*. Retrieved March 18, 2009, from Play.com (UK): DVDs, Music CDs, MP3s, Video Games, Books, Electronics & Gadgets - Free Delivery: http://www.play.com/

Pratchett, T., & Briggs, S. (2004). *The New Discworld Companion.* London: Victor Gollancz Ltd.

Senecal, S., & Nantel, J. (2004). The influence of online product recommendations on consumers' online choices. *Journal of Retailing* , 159-169.

Stanicek, P. (2009). *Color Scheme Designer*. Retrieved April 11, 2009, from Color Scheme Designer 3: http://colorschemedesigner.com/

Wilson, C. (2005, September 15). *The <?xml> prolog, strict mode, and XHTML in IE*. Retrieved April 13, 2009, from IEBlog: http://blogs.msdn.com/ie/archive/2005/09/15/467901.aspx

Xooglers. (2005, December 9). *Xooglers*. Retrieved November 9, 2008, from Let's get a real database: http://xooglers.blogspot.com/2005/12/lets-get-real-database.html

Ziegler, C.-N., Lausen, G., & Schmidt-Thieme, L. (2004). Taxonomy-driven Computation of Product Recommendations. *Proceedings of the thirteenth ACM international conference on Information and knowledge management* , 406-415.

Appendix V – Diary

6th October 2008	Discussed basics of project, initial presentation, etc.
13th October 2008	Initial presentation.
27th October 2008	Discussed the research material I've found so far.
3rd November 2008	Further discussion of the research material and initial conclusions regarding the research.
10th November 2008	Briefly looked over Milestone 1 draft; left copy with supervisor (Nathan Thomas) to have a more thorough read-through.
15th November 2008	Received feedback about Milestone 1 draft.
17th November 2008	Discussed a few last-minute changes for Milestone 1.
1st December 2008	Discussed Milestone 1 feedback.
12th January 2009	Discussed Milestone 2 & design section. Also discussed my mitigating circumstances claim for issues that arose over late December. This mitigating circumstances stuff will affect the rest of the project due to me being limited in how much time I can spend on computer work.
19th January 2009	Occurred immediately following a meeting with Student Services and simply discussed the mitigating circumstances issues and how I could proceed with Milestone 2. Decided to put in a mitigating circumstances claim for non-submission of Milestone 2, with a plan to submit it along with the rest of the project at the end of April. A small amount of design work had been done.
	No meetings for a couple of months due to mitigating circumstances issues – my time had to be focused on other assignments for which I still struggled to find time.
23rd March 2009	Discussed how to continue with the rest of Milestone 2, and how to finish the project.

My project schedule was adhered to very closely until Milestone 1's submission. After that point, however, it was effectively ignored for all purposes except noticing how far behind schedule I was and knowing that there was little I could do to resolve that. The three weeks following Milestone 1's submission were largely consumed by assignments for other modules. I knew that the number of other assignments needing to be done would greatly reduce after the Christmas break, so planned to catch up throughout January and early February. What I didn't count on, however, were the mitigating circumstances that arose at the end of December and which have persisted through into the summer term. This issue severely restricted the amount of time that I could spend working on the project. As such, I put in a mitigating circumstances claim form indicating non-submission of Milestone 2. The original plan was to submit Milestone 2 at a later date, but with no set deadline for the late submission, so the only option on the form was to indicate non-submission but submit it normally at an unspecified later date. Upon speaking to the project organiser and level three degree tutor, Phil Davies, this plan was altered to submitting Milestone 2 along with Milestone 3 in April. Due to these setbacks, almost all of the post-Milestone 1 work was completed in the latter half of March and in April. A small amount of design work was completed in January, and was followed by a little more design work in the two weeks of March prior to the Easter break. The rest of the design, all of the development, and the evaluation and conclusion was all done in the last ten days of the Easter break – again because of pressure from other modules' assignments and the physically limited amount of time I could spend doing computer work.

Appendix VI – Code Listings

Database Setup

```
CREATE DATABASE `lvl3` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;

CREATE TABLE `lvl3`.`orders` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`customer` INT UNSIGNED NOT NULL ,
`date` DATETIME NOT NULL ,
INDEX ( `customer` )
) ENGINE = InnoDB;

CREATE TABLE `lvl3`.`customers` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`name` VARCHAR( 128 ) NOT NULL ,
`address` TEXT NOT NULL
) ENGINE = InnoDB;

CREATE TABLE `lvl3`.`orderedproducts` (
`order` INT UNSIGNED NOT NULL ,
`product` INT UNSIGNED NOT NULL ,
`quantity` SMALLINT UNSIGNED NOT NULL ,
`recipient` INT UNSIGNED NOT NULL ,
`hideuntil` DATETIME NULL ,
`purchaseruse` BOOL NOT NULL ,
`recipientuse` BOOL NOT NULL ,
PRIMARY KEY ( `order` , `product` ) ,
INDEX ( `recipient` )
) ENGINE = InnoDB;

CREATE TABLE `lvl3`.`products` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`name` VARCHAR( 128 ) NOT NULL ,
`price` DECIMAL( 10, 2 ) UNSIGNED NOT NULL
) ENGINE = InnoDB;

CREATE TABLE `lvl3`.`productratings` (
`product` INT UNSIGNED NOT NULL ,
`customer` INT UNSIGNED NOT NULL ,
`rating` FLOAT( 3, 2 ) UNSIGNED NOT NULL ,
PRIMARY KEY ( `product` , `customer` )
) ENGINE = InnoDB;

CREATE TABLE `lvl3`.`productsimilarity` (
`producta` INT UNSIGNED NOT NULL ,
`productb` INT UNSIGNED NOT NULL ,
`similarity` FLOAT( 10, 9 ) UNSIGNED NOT NULL ,
PRIMARY KEY ( `producta` , `productb` )
) ENGINE = InnoDB;

CREATE TABLE `lvl3`.`exceptions` (
`customer` INT UNSIGNED NOT NULL ,
`product` INT UNSIGNED NOT NULL ,
PRIMARY KEY ( `customer` , `product` )
) ENGINE = InnoDB;

ALTER TABLE `orders` ADD FOREIGN KEY ( `customer` ) REFERENCES `lvl3`.`customers` (
`id`
) ON DELETE RESTRICT ON UPDATE RESTRICT ;

ALTER TABLE `orderedproducts` ADD FOREIGN KEY ( `order` ) REFERENCES `lvl3`.`orders` (
`id`
) ON DELETE RESTRICT ON UPDATE RESTRICT ;

ALTER TABLE `orderedproducts` ADD FOREIGN KEY ( `product` ) REFERENCES `lvl3`.`products` (
`id`
```

```
) ON DELETE RESTRICT ON UPDATE RESTRICT ;

ALTER TABLE `orderedproducts` ADD FOREIGN KEY ( `recipient` ) REFERENCES `lvl3`.`customers` (
`id`
) ON DELETE RESTRICT ON UPDATE RESTRICT ;

ALTER TABLE `productratings` ADD FOREIGN KEY ( `product` ) REFERENCES `lvl3`.`products` (
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

ALTER TABLE `productratings` ADD FOREIGN KEY ( `customer` ) REFERENCES `lvl3`.`customers` (
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

ALTER TABLE `exceptions` ADD FOREIGN KEY ( `customer` ) REFERENCES `lvl3`.`customers` (
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

ALTER TABLE `exceptions` ADD FOREIGN KEY ( `product` ) REFERENCES `lvl3`.`products` (
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

ALTER TABLE `productsimilarity` ADD FOREIGN KEY ( `producta` ) REFERENCES `lvl3`.`products` (
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

ALTER TABLE `productsimilarity` ADD FOREIGN KEY ( `productb` ) REFERENCES `lvl3`.`products` (
`id`
) ON DELETE CASCADE ON UPDATE CASCADE ;

CREATE VIEW ratedproducts AS SELECT p.*,AVG(IF(r.rating IS NULL,2.5,r.rating)) AS
rating,COUNT(r.rating) AS ratingcount FROM products AS p LEFT JOIN productratings AS r ON
p.id=r.product GROUP BY p.id;

INSERT INTO `customers` (`id`, `name`, `address`) VALUES
(1, 'Alberto Malich', 'Death''s Domain,\r\nDiscworld'),
(2, 'Moist von Lipwig', 'Royal Mint,\r\nAnkh-Morpork,\r\nDiscworld'),
(3, 'Rincewind', 'Unseen University,\r\nAnkh-Morpork,\r\nDiscworld'),
(4, 'Ponder Stibbons', 'High Energy Magic Building,\r\nUnseen University,\r\nAnkh-
Morpork,\r\nDiscworld'),
(5, 'Susan Sto Helit', 'Sto Helit,\r\nSto Plains,\r\nDiscworld'),
(6, 'Twoflower', 'Agatean Empire,\r\nDiscworld'),
(7, 'Havelock Vetinari', 'Patrician''s Palace,\r\nAnkh-Morpork,\r\nDiscworld'),
(8, 'Esmerelda Weatherwax', 'Lancre,\r\nDiscworld');

INSERT INTO `products` (`id`, `name`, `image`, `price`) VALUES
(1, 'The Colour of Magic', 'tcom', '5.99'),
(2, 'Nation', 'nation', '8.49'),
(3, 'Dreams from My Father', 'dfmf', '5.00'),
(4, 'Dexter: An Omnibus', 'dexter', '9.99'),
(5, 'Terry Pratchett''s Colour Of Magic', 'tcomdvd', '11.99'),
(6, 'Futurama: Into The Wild Green Yonder', 'itwgy', '9.99'),
(7, 'Yellowstone', 'yellowstone', '12.99'),
(8, 'Firefly: Complete Series 1', 'firefly', '10.99'),
(9, 'Ancora', 'ancora', '3.99'),
(10, 'The Promise', 'promise', '8.95'),
(11, 'Invaders Must Die', 'imd', '8.95'),
(12, 'A State Of Trance Year Mix 2008', 'asot', '11.99'),
(13, 'No Line On The Horizon', 'nloth', '8.95'),
(14, 'Dig Out Your Soul', 'doys', '5.00'),
(15, 'A Hundred Million Suns', 'ahms', '6.99'),
(16, 'Yes', 'yes', '8.95'),
(17, 'Halo Wars: Limited Edition', 'hw', '44.99'),
(18, 'Grand Theft Auto IV', 'gta4', '39.99'),
(19, 'HDMI Cable for Xbox 360 & PS3', 'hdmi', '6.99'),
(20, 'Illuminated Universal Drum Stick - Flame Red', 'iuds', '17.99'),
(21, 'Madcatz Rock Band Guitar Tree Stand', 'rbgts', '16.99'),
```

```
(22, 'skate 2', 'skate', '39.99'),
(23, 'Mirror''s Edge', 'me', '17.99'),
(24, 'Datel Universal Wireless Dual Microphones for PS3 & Wii', 'uwdm', '29.99'),
(25, 'Sonic And The Black Knight', 'sonic', '34.99'),
(26, 'Dead Rising: Chop Till You Drop', 'drctyd', '24.99'),
(27, 'Datel Wii LAN Adapter', 'lan', '14.99'),
(28, 'Exspect Wii Component Cable', 'component', '7.99'),
(29, 'Warhammer 40,000: Dawn Of War II', 'dow2', '22.99'),
(30, 'Burnout: Paradise - The Ultimate Box', 'burnout', '24.99'),
(31, 'Microsoft Xbox 360 Crossfire Wireless Gaming Receiver for Windows', 'wgr', '22.99');
```

"calculatesimilarities" Stored Procedure

```
DELIMITER |

CREATE PROCEDURE calculatesimilarities()
BEGIN
   /*     Set up the temporary tables that we use to store intermediate data */
   DROP TABLE IF EXISTS tempvector1;
   DROP TABLE IF EXISTS tempvector2;
   DROP TABLE IF EXISTS tempcalcresults;
   CREATE TEMPORARY TABLE tempvector1 (
          customer INT UNSIGNED NOT NULL,
          rating FLOAT(10,9) NOT NULL,
          PRIMARY KEY (customer)
   );
   CREATE TEMPORARY TABLE tempvector2 (
          customer INT UNSIGNED NOT NULL,
          rating FLOAT(10,9) NOT NULL,
          PRIMARY KEY (customer)
   );
   CREATE TEMPORARY TABLE tempcalcresults (
          dotproduct FLOAT(10,5),
          length1 FLOAT(10,9),
          length2 FLOAT(10,9),
          result FLOAT(10,9)
   );

   outerblock: BEGIN
          /*     Declare the variables used to handle looping over each product for
                 the first half of the product pair. p1 is the ID of each product
                 found; p1end gets set to 1 when there are no more products; p2 and
                 p2end are used for the same purpose for the other side of the
                 product pairing through the query executed later */
          DECLARE p1, p1end, p2, p2end INT;
          DECLARE p2exception INT DEFAULT 0;
          DECLARE p1cur CURSOR FOR SELECT id FROM products;
          DECLARE CONTINUE HANDLER FOR NOT FOUND SET p1end = 1;
          /*     ROLLBACK if an error occurs */
          DECLARE EXIT HANDLER FOR SQLEXCEPTION ROLLBACK;

          SET p1end = 0;

          /*     Use a transaction so that we either successfully recalculate
                 similarities for all products or we don't change anything. This
                 prevents us being left with a half-populated table that would break
                 everything */
          START TRANSACTION;

          /*     Empty the similarity table ready for new data! */
          TRUNCATE productsimilarity;

          /*     Iterate over each product, fetching its ID into p1 */
          OPEN p1cur;
          REPEAT
                 FETCH p1cur INTO p1;

                 IF p1end != 1 THEN

                        /*     Empty our first vector table then pull a list of customers
                               & their ratings into it. The SELECT query pulls customers
                               who have ever purchased or rated this product, or had it
                               bought for them */
                        TRUNCATE tempvector1;
                        INSERT INTO tempvector1 (customer, rating)
                               SELECT c.id AS customer, IF(rating IS NOT NULL, rating-3, IF(quantity, 1,
0)) AS rating FROM customers AS c LEFT JOIN productratings AS r ON c.id=r.customer LEFT JOIN (SELECT
product,quantity,IF(recipient IS NULL,customer,recipient) AS customer FROM orderedproducts AS op LEFT
```

```
JOIN orders AS o ON op.order=o.id WHERE IF(recipient IS NULL,purchaseruse,recipientuse)=1) AS op ON
c.id=op.customer WHERE (op.product=p1 OR r.product=p1) AND c.id NOT IN (SELECT customer FROM
exceptions WHERE product=p1);

                        p2block: BEGIN
                                /*      Prepare to iterate over all of the products that were
                                        ALSO bought by each customer who was found with the
                                        previous query. This links each product to each other
                                        product via the customers who bought/rated/received
                                        them both */
                                DECLARE p2cur CURSOR FOR
                          SELECT r.product FROM productratings AS r LEFT JOIN exceptions AS e ON
(r.product=e.product AND r.customer=e.customer) WHERE r.customer IN (SELECT customer FROM
productratings WHERE product=p1) AND r.product!=p1 AND (e.customer IS NULL AND e.product IS NULL)
                          UNION
                          SELECT op.product FROM orderedproducts AS op LEFT JOIN orders AS o ON
op.order=o.id LEFT JOIN exceptions AS e ON (op.product=e.product AND o.customer=e.customer) WHERE
IF(op.recipient IS NULL,o.customer,op.recipient) IN (SELECT IF(recipient IS NULL,customer,recipient)
FROM orderedproducts AS op LEFT JOIN orders AS o ON op.order=o.id WHERE op.product=p1) AND
op.product!=p1 AND (e.customer IS NULL AND e.product IS NULL);

                                DECLARE CONTINUE HANDLER FOR NOT FOUND SET p2end = 1;
                                /*      If an error occurs, set p2exception to 1 so that the
                                        later code can ROLLBACK and then exit (hopefully - I'm
                                        not sure that this ever worked) */
                                DECLARE EXIT HANDLER FOR SQLEXCEPTION SET p2exception = 1;

                                SET p2end = 0;

                                /*      Actually perform the iteration over the products
                                        returned by p2cur described above */
                                OPEN p2cur;
                                REPEAT
                                        FETCH p2cur INTO p2;

                                        IF p2end != 1 THEN

                                                /*      Empty our second vector then pull into it a
                                                        list of customers & ratings of customers who've
                                                        rated/purchased/received this product (p2) */
                                                TRUNCATE tempvector2;
                                                INSERT INTO tempvector2 (customer, rating)
                                                        SELECT c.id AS customer, IF(rating IS NOT NULL,
rating-3, IF(quantity, 1, 0)) AS rating FROM customers AS c LEFT JOIN productratings AS r ON
c.id=r.customer LEFT JOIN (SELECT product,quantity,IF(recipient IS NULL,customer,recipient) AS
customer FROM orderedproducts AS op LEFT JOIN orders AS o ON op.order=o.id WHERE IF(recipient IS
NULL,purchaseruse,recipientuse)=1) AS op ON c.id=op.customer WHERE (op.product=p2 OR r.product=p2) AND
c.id NOT IN (SELECT customer FROM exceptions WHERE product=p2);

                                                /*      Clear our temporary calculation results table */
                                                TRUNCATE tempcalcresults;

                                                /*      Calculate the dot product of the two vectors */
                                                INSERT INTO tempcalcresults (dotproduct) VALUES ((SELECT
SUM(a.rating * b.rating) FROM tempvector1 AS a LEFT JOIN tempvector2 AS b ON a.customer=b.customer));

                                                /*      Then calculate the lengths of each vector */
                                                UPDATE tempcalcresults SET length1 = (SELECT
SQRT(SUM(POW(rating, 2))) FROM tempvector1);
                                                UPDATE tempcalcresults SET length2 = (SELECT
SQRT(SUM(POW(rating, 2))) FROM tempvector2);

                                                /*      Ensure that none of the values are NULL */
                                                UPDATE tempcalcresults SET dotproduct=IF(dotproduct IS
NULL, 0, dotproduct), length1=IF(length1 IS NULL, 1, length1), length2=IF(length2 IS NULL, 1,
length2);
```

```
                                        /*     Calculate the final similarity value */
                                        UPDATE tempcalcresults SET result=dotproduct/(length1 *
length2);

                                        /*     Ensure that the result isn't NULL... */
                                        UPDATE tempcalcresults SET result=(IF(result IS NULL, 0,
result));

                                        /*     And finally insert the result into the similarity
table */
                                        INSERT INTO productsimilarity (producta, productb,
similarity) VALUES (p1, p2, (SELECT result FROM tempcalcresults LIMIT 1));
                                    END IF;

                                    UNTIL p2end = 1
                                END REPEAT;
                                CLOSE p2cur;
                            END p2block;

                            IF p2exception = 1 THEN
                                ROLLBACK;
                                LEAVE outerblock;
                            END IF;
                        END IF;

                        UNTIL p1end = 1
                    END REPEAT;
                    CLOSE p1cur;

                    /*     Delete any similarity entries where one product is being described
                           as being similar to itself or the similarity is zero (not similar
                           at all), although I'm beginning to suspect that this last part
                           might be removing genuinely similar products, since we have
                           negative values as well... */
                    DELETE FROM productsimilarity WHERE producta=productb OR similarity=0;

                    /*     Commit the changes to the database: if we've got to this point, no
                           errors have occurred! */
                    COMMIT;

               END outerblock;
            END|

            DELIMITER ;
```

Website Code

classes/cart.php

```php
<?php

/**
* Class to handle cart functions, like the storage of items in each user's cart
*  and adding/removing items to/from the cart
*/
class Cart {
   /**
   * The list of items in this user's cart
   *
   * @var array
   */
   private static $cartItems;

   /**
   * Sets up the cart for this user, using a session variable as storage. It
   *  registers a shutdown function to store the cart contents in the session
   *  variable at the end of the script's execution
   *
   */
   private static function _construct() {
          if (isset($_SESSION['cartitems'])) {
                 self::_unserialize();
          } else {
                 self::$cartItems = array();
          }
          if (!is_array(self::$cartItems)) self::$cartItems = array();
          register_shutdown_function(array('Cart', '_destruct'));
   }

   /**
   * Calls _serialize() to store the contents of the cart in the session var
   *
   */
   public static function _destruct() {
          self::_serialize();
   }

   /**
   * Stores the contents of this user's cart in their session. Is it actually
   *  necessary to serialize the value to a string? Can't PHP sessions store
   *  arrays? I should probably have tested that at some point
   *
   */
   private static function _serialize() {
          $o = array();
          foreach (self::$cartItems as $item) {
                 $o[] = array('i'=>$item->getId(), 'q'=>$item->getQuantity());
          }
          $_SESSION['cartitems'] = serialize($o);
   }

   /**
   * Fetches the cart contents the the session variable and converts them to
   *  CartItem objects
   *
   */
   private static function _unserialize() {
          foreach (unserialize($_SESSION['cartitems']) as $item) {
                 self::$cartItems[] = new CartItem($item['i'], $item['q']);
          }
   }
```

```
/**
* Empties the cart
*
*/
public static function _clear() {
	self::$cartItems = array();
}

/**
* Adds the specified item to this user's cart. If the product is already in
*  the cart, the existing item's quantity will be incremented by the new
*  item's quantity to prevent duplicate items existing in the cart
*
* @param CartItem $item
*/
public static function addItem(CartItem $item) {
	if (self::$cartItems === null) self::_construct();
	if ($existingItem = self::findItemById($item->getId())) {
		$existingItem->setQuantity($existingItem->getQuantity() + $item->getQuantity());
	} else {
		self::$cartItems[] = $item;
	}
}

/**
* Removes the specified item from the cart
*
* @param CartItem $item
*/
public static function removeItem(CartItem $item) {
	if (self::$cartItems === null) self::_construct();
	$k = array_search($item, self::$cartItems);
	unset(self::$cartItems[$k]);
}

/**
* Returns the CartItem object with the specified product ID
*
* @param int $id
* @return CartItem
*/
public static function findItemById($id) {
	if (self::$cartItems === null) self::_construct();
	foreach (self::$cartItems as $ci) {
		if ($ci->getId() == $id) {
			return $ci;
		}
	}
}

/**
* Removes the item with the specified product ID from the cart. Returns a
*  boolean indicating if a product was found (and thus removed)
*
* @param int $id
* @return bool
*/
public static function removeItemById($id) {
	if (self::$cartItems === null) self::_construct();
	if ($ci = self::findItemById($id)) {
		self::removeItem($ci);
		return true;
	}
	return false;
}

/**
```

```
	* Returns an array of all items currently in the cart
	*
	* @return array
	*/
	public static function getItems() {
		if (self::$cartItems === null) self::_construct();
		return self::$cartItems;
	}

	/**
	* Returns the total price of this cart's contents as a float
	*
	* @return float
	*/
	public static function getTotalPrice() {
		if (self::$cartItems === null) self::_construct();
		$tp = 0.0;
		foreach (self::$cartItems as $ci) {
			$tp += $ci->getLinePrice();
		}
		return $tp;
	}

	/**
	* Returns the total price of this cart's contents as a formatted string
	*
	* @return string
	*/
	public static function getFormattedTotalPrice() {
		return Formatting::formatPrice(self::getTotalPrice());
	}

	/**
	* Converts this cart to an order. Returns a boolean indicating success
	*
	* @return boolean
	* @throws UserNotLoggedInException
	*/
	public static function convertToOrder() {
		if ($user = Users::getCurrentUser()) {
			DB::transactionStart();
			$result1 = DB::updateQuery('INSERT INTO orders (customer,date) VALUES
(?,UTC_TIMESTAMP())', array($user));
			$orderId = $result1->insertId();

			foreach (self::$cartItems as $ci) {
				$ci->insertToOrder($orderId);
			}
			self::_clear();
			return DB::transactionEnd();
		} else {
			throw new UserNotLoggedInException();
		}
	}
}

/**
* Represents a single item in the user's cart
*/
class CartItem {
	/**
	* The product ID of the item
	*
	* @var int
	*/
	private $id;
```

```php
/**
* The quantity of this item in the cart
*
* @var int
*/
private $quantity;

/**
* The Product object for this item's product
*
* @var Product
*/
private $product;

/**
* The recipient of this cart item, if it has one. Will be set by the
*  checkout page before the order is converted to a product, hence its
*  appearance here.
*
* @var User
*/
private $recipient;

/**
* The date until which this item will be hidden from its recipient, if
*  relevant. Will be set by the checkout page before the order is converted
*  to a product, hence its appearance here.
*
* @var Date
*/
private $hideUntil;

/**
* Creates a new CartItem object. Should not be called directly except from
*  the Cart class
*
* @param int $id
* @param int $quantity
* @param User $recipient
* @param Date $hideUntil
* @return CartItem
*/
public function __construct($id, $quantity, User $recipient=null, Date $hideUntil=null) {
	$this->id = $id;
	$this->quantity = $quantity;
	$this->recipient = $recipient;
	$this->hideUntil = $hideUntil;
}

/**
* Returns the ID of this cart item's product
*
* @return int
*/
public function getId() {
	return $this->id;
}

/**
* Returns the quantity of this item
*
* @return int
*/
public function getQuantity() {
	return $this->quantity;
}
```

```
/**
* Sets the quantity of this item
*
* @param int $newQuantity
*/
public function setQuantity($newQuantity) {
	$this->quantity = $newQuantity;
}

/**
* Returns the Product object for this cart item
*
* @return Product
*/
public function getProduct() {
	if ($this->product === null) $this->product = Product::createFromId($this->id);
	return $this->product;
}

/**
* Returns the line price (product price multiplied by quantity) of this
*  cart item
*
* @return float
*/
public function getLinePrice() {
	return $this->getProduct()->getPrice() * $this->quantity;
}

/**
* Returns the line price (product price multiplied by quantity) of this
*  cart item as a formatted string
*
*/
public function getFormattedLinePrice() {
	return Formatting::formatPrice($this->getLinePrice());
}

/**
* Returns the recipient of this purchase, if there is one set
*
* @return User
*/
public function getRecipient() {
	return $this->recipient;
}

/**
* Sets the recipient of this product
*
* @param User $recipient
*/
public function setRecipient(User $recipient) {
	$this->recipient = $recipient;
}

/**
* Returns the "hide until" date of this purchase, if it has one
*
* @return Date
*/
public function getHideUntil() {
	return $this->hideUntil;
}

/**
* Sets the "hide until" date of this purchase
```

```
	*
	* @param Date $hideUntil
	*/
	public function setHideUntil(Date $hideUntil) {
		if ($hideUntil->getIsLaterThan(Date::createNow())) {
			$this->hideUntil = $hideUntil;
		}
	}

	/**
	* Inserts this product as a purchase for the specified order. Should ONLY
	*  be called from the Order class
	*
	* @param int $orderId
	*/
	public function insertToOrder($orderId) {
		DB::updateQuery('INSERT INTO orderedproducts (`order`,product,quantity,recipient,hideuntil)
VALUES (?,?,?,?,?)', array($orderId, $this->id, $this->quantity, $this->recipient, $this->hideUntil));
	}
}

?>
```

classes/date.php

```php
<?php

/**
* A class to format, adjust and display dates as needed.
*
* Originally based on a Date class that I use for all of my PHP projects/sites,
*  but with various methods removed that weren't needed for this project
*/
class Date {
   const TIMEZONE_IDENTIFIER = 'Europe/London';
   const DATE_FORMAT = 'd/m/Y H:i:s';

   /**
    * The PHP DateTime object we're using to store our date/time
    *
    * @var DateTime
    */
   private $dateTime;

   /**
    * Creates a new Date object using the passed DateTime object, $dateTime.
    *  Should only be called from static Date::Create*() functions
    *
    * @param DateTime $dateTime
    * @return Date
    */
   private function __construct(DateTime $dateTime) {
         $this->dateTime = $dateTime;
   }

   /**
    * Creates a new Date object from the date string passed. We use this
    *  wrapper in order to specify the timezone as UTC for every creation
    *  method without having to put the "new DateTimeZone()" code in every
    *  Create* static method
    *
    * @param string $date
    * @return Date
    */
   private static function create($date) {
         return new self(new DateTime($date, new DateTimeZone('UTC')));
   }

   /**
    * Creates and returns a new Date object from the supplied Unix timestamp,
    *  assumed to be UTC (as Unix timestamps should be)
    *
    * @param integer $unixtime
    * @return Date
    */
   public static function createFromUnixtime($unixtime) {
         return self::create('@'.intval($unixtime));
   }

   /**
    * Creates and returns a new Date object from the supplied MySQL DATETIME or
    *  DATE string (YYYY-MM-DD HH:MM:SS or YYYY-MM-DD), assumed to be UTC
    *  (which they will be if we stored the value originally)
    *
    * @param string $mySQLDate
    * @return Date
    */
   public static function createFromMySQL($mySQLDate) {
         //Dates from MySQL are assumed to be UTC
         //YYYY-MM-DD HH:MM:SS or YYYY-MM-DD
```

```
        if (preg_match('/^\d{4}-\d{2}-\d{2}(?:| \d{2}:\d{2}:\d{2})$/', $mySQLDate)) {
                return self::create($mySQLDate);
        }
        throw new DateIsNotMySQLFormatException("String passed to Date::createFromMySQL() -
'$mySQLDate' - is not in the MySQL DATETIME or DATE format (YYYY-MM-DD HH:MM:SS or YYYY-MM-DD)");
  }

  /**
   * Creates and returns a new Date object from the supplied date/time
   *  components. Any components that are not supplied (or set to NULL) are
   *  set to the current value (in UTC)
   *
   * @param integer $yr
   * @param integer $mon
   * @param integer $day
   * @param integer $hr
   * @param integer $min
   * @param integer $sec
   * @return Date
   */
  public static function createFromParts($yr=null, $mon=null, $day=null, $hr=null, $min=null,
$sec=null) {
        if ($yr===null)      $yr  = (int) gmdate('Y');
        if ($mon===null) $mon = (int) gmdate('n');
        if ($day===null) $day = (int) gmdate('j');
        if ($hr===null)      $hr  = (int) gmdate('G');
        if ($min===null) $min = (int) gmdate('i');
        if ($sec===null) $sec = (int) gmdate('s');

        return self::CreateFromUnixtime(gmmktime($hr, $min, $sec, $mon, $day, $yr));
  }

  /**
   * Creates and returns a new Date object representing the current time (UTC)
   *
   * @return Date
   */
  public static function createNow() {
        return self::createFromUnixtime(time());
  }

  /**
   * Formats this date using the date() format, $String
   *
   * @param string $string
   * @return string
   */
  public function display($string=false) {
        if ($string === false) {
                $string = self::DATE_FORMAT;
        }

        return $this->dateTime->format($string);
  }

  /**
   * Returns this date as a MySQL DATE format string
   *
   * @return string
   */
  public function toMySQLDATE() {
        return $this->display('Y-m-d');
  }

  /**
   * Returns this date as a MySQL DATETIME format string
   *
```

```
     * @return string
     */
    public function toMySQLDATETIME() {
            return $this->display('Y-m-d H:i:s');
    }

    /**
     * Returns a MySQL DATETIME string
     *
     * @return string
     */
    public function __toMySQL() {
            return $this->toMySQLDATE();
    }

    /**
     * Returns this date as a Unix timestamp
     *
     */
    public function toUnixTimestamp() {
            return $this->display('U');
    }

    /**
    * Returns the date as a string - specifically a full RFC 2822 date
    *
    * @return string
    */
    public function __toString() {
            return $this->display('r');
    }

    public function getIsEarlierThan(Date $date) {
            return ($this->toUnixTimestamp() < $date->toUnixTimestamp());
    }

    public function getIsLaterThan(Date $date) {
            return ($this->toUnixTimestamp() > $date->toUnixTimestamp());
    }
}

?>
```

classes/db.php

```php
<?php

/**
* A class to handle accessing the database.  It does a lot more than is really
*  needed in this system, but I have plans to use it in future projects as
*  well, so I made it do everything that might ever be needed.
*
* DB::setup() has to be called before anything else can be done with it; after
*  which selectQuery(), selectOne(), selectRow(), selectAll() and updateQuery()
*  can be used to run SQL queries on the database, returning various useful
*  objects or other values
*/
class DB {
   /**
   * The mysqli object used to link to the database internally
   *
   * @var mysqli
   */
   private static $link;

   /**
   * Whether a transaction is currently open
   *
   * @var boolean
   */
   private static $transactionOpen = false;

   /**
   * Whether the current transaction has failed or not
   *
   * @var boolean
   */
   private static $transactionFailed = false;

   /**
   * Opens a connection to the database
   *
   * @param string $host
   * @param string $username
   * @param string $password
   * @param string $database
   * @throws DBConnectionException
   * @link http://php.net/mysqli
   */
   public static function setup($host, $username, $password, $database) {
         self::$link = new mysqli($host, $username, $password, $database);
         if (mysqli_connect_error()) throw new DBConnectionException(mysqli_connect_errno(),
mysqli_connect_error());

         self::$link->set_charset('utf8');
   }

   /**
   * Ensures that the database connection has been established, throwing a
   *  DBNotSetUpException if it hasn't been
   *
   * @throws DBNotSetUpException
   */
   private static function checkSetup() {
         if (self::$link === null) throw new DBNotSetUpException();
   }

   /**
   * Prepares the passed SQL statement for execution, returning a mysqli_stmt
   *  object
```

```
    *
    * @param string $sql
    * @return mysqli_stmt
    * @throws DBQueryPreparationFailureException
    * @link http://php.net/mysqli.prepare
    */
   private static function prepare($sql) {
          if ($p = self::$link->prepare($sql)) {
                return $p;
          }
          throw new DBQueryPreparationFailureException(mysqli_errno(self::$link),
mysqli_error(self::$link));
   }

   /**
    * Runs a SELECT query, returning a DBSelectResult object. The returned
    *  object can be passed directly to foreach() to iterate over the result
    *  set
    *
    * @param string $sql
    * @param array $params
    * @return DBSelectResult
    * @throws DBNotSetUpException
    */
   public static function selectQuery($sql, array $params=null) {
          self::checkSetup();
          return new DBSelectResult(self::prepare($sql), $params);
   }

   /**
    * Runs a SELECT query, returning the value in the first column of the first
    *  row
    *
    * @param string $sql
    * @param array $params
    * @return mixed
    * @throws DBNotSetUpException
    */
   public static function selectOne($sql, array $params=null) {
          self::checkSetup();
          $r = new DBSelectResult(self::prepare($sql), $params);
          $cur = $r->current();
          return reset($cur);
   }

   /**
    * Runs a SELECT query, returning the first row of the result set
    *
    * @param string $sql
    * @param array $params
    * @return array
    * @throws DBNotSetUpException
    */
   public static function selectRow($sql, array $params=null) {
          self::checkSetup();
          $r = new DBSelectResult(self::prepare($sql), $params);
          return $r->current();
   }

   /**
    * Runs a SELECT query, returning all of the result set as a two-dimensional
    *  array.  Only use this method if it's strictly necessary as it is slower
    *  than others; use selectQuery() instead if all you plan on doing is
    *  foreach()'ing over the results, as the DBSelectResult object that
    *  selectQuery() returns can be foreach()'d over
    *
    * @param string $sql
```

```
	* @param array $params
	* @return array
	* @throws DBNotSetUpException
	*/
	public static function selectAll($sql, array $params=null) {
		self::checkSetup();
		$r = new DBSelectResult(self::prepare($sql), $params);
		$out = array();
		foreach ($r as $row) {
			$out[] = $row;
		}
		return $out;
	}

	/**
	* Runs an UPDATE/INSERT/DELETE query, returning a DBUpdateResult object
	*  that can be used to get values such as the number of affected rows
	*
	* @param string $sql
	* @param array $params
	* @return DBUpdateResult
	* @throws DBNotSetUpException
	*/
	public static function updateQuery($sql, array $params=null) {
		self::checkSetup();
		return new DBUpdateResult(self::prepare($sql), $params);
	}

	/**
	* Starts a database transaction. Will throw a
	*  DBTransactionAlreadyOpenException if a transaction is already open
	*
	* @throws DBNotSetUpException
	* @throws DBTransactionAlreadyOpenException
	* @link http://php.net/mysqli.autocommit
	*/
	public static function transactionStart() {
		self::checkSetup();
		if (self::$transactionOpen) throw new DBTransactionAlreadyOpenException();
		self::$link->autocommit(false);
		self::$transactionOpen = true;
		self::$transactionFailed = false;
	}

	/**
	* Ends the transaction.  If $rollback is TRUE or a SQL error has occurred
	*  since opening the transaction, a "ROLLBACK" command will be issued,
	*  rolling back any changes made since opening the transaction.  Otherwise
	*  the transaction will be COMMITted to disk. Returns TRUE if the
	*  transaction was COMMITted; FALSE if it was ROLLBACKed.
	*
	* @param bool $rollback
	* @return bool
	* @throws DBNotSetUpException
	* @throws DBTransactionNotOpenException
	* @link http://php.net/mysqli.commit
	* @link http://php.net/mysqli.rollback
	* @link http://php.net/mysqli.autocommit
	*/
	public static function transactionEnd($rollback=false) {
		self::checkSetup();
		if (!self::$transactionOpen) throw new DBTransactionNotOpenException();

		$return = true;
		if ($rollback || self::$transactionFailed) {
			self::$link->rollback();
			$return = false;
```

```
		} else {
			self::$link->commit();
		}
		self::$link->autocommit(true);
		self::$transactionOpen = false;
		return $return;
	}

	/**
	 * Sets the failure state of the transaction to no errors having occurred.
	 *  This can be used if an error was "expected" to occur in the middle of a
	 *  transaction and you want the transaction to still be COMMITted.  Use
	 *  this with care!!  Bear in mind that an error could potentially have
	 *  occurred before the query that you were expecting to error, and this
	 *  method will clear that query's error state
	 *
	 * @todo Replace the $transactionFailed flag with an integer count of
	 *       errors, and have this method just decrement the value
	 */
	public static function clearTransactionErrors() {
		self::$transactionFailed = false;
	}

	/**
	 * Sets the failure state of the transaction to be TRUE.  This will cause
	 *  the transaction to be rolled back when transactionEnd() is called.
	 *
	 */
	public static function failTransactionDueToError() {
		self::$transactionFailed = true;
	}

	/**
	 * Returns a boolean indicating if there is a transaction currently open
	 *
	 * @return bool
	 */
	public static function getIsTransactionOpen() {
		return self::$transactionOpen;
	}

	/**
	 * Returns the character set currently used by the database. This is almost
	 *  guaranteed to be "utf8", since that is specified upon connection to the
	 *  database.
	 *
	 * @return mixed
	 * @throws DBNotSetUpException
	 * @link http://php.net/mysqli.get-charset
	 */
	public static function getCharset() {
		self::checkSetup();
		return self::$link->get_charset();
	}

	/**
	 * Returns the client information about this database connection
	 *
	 * @return mixed
	 * @throws DBNotSetUpException
	 * @link http://php.net/mysqli.get-client-info
	 */
	public static function getClientInfo() {
		self::checkSetup();
		return self::$link->get_client_info();
	}
```

```php
/**
* Returns the client version of this database connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.get-client-version
*/
public static function getClientVersion() {
	self::checkSetup();
	return self::$link->get_client_version();
}

/**
* Returns the host information of this database connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.get-host-info
*/
public static function getHostInfo() {
	self::checkSetup();
	return self::$link->host_info;
}

/**
* Returns the protocol version about this connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.get-proto-info
*/
public static function getProtocolVersion() {
	self::checkSetup();
	return self::$link->protocol_version;
}

/**
* Returns the server information about this connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.get-server-info
*/
public static function getServerInfo() {
	self::checkSetup();
	return self::$link->server_info;
}

/**
* Returns the server version about this connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.get-server-version
*/
public static function getServerVersion() {
	self::checkSetup();
	return self::$link->server_version;
}

/**
* Returns the warnings that have occurred on this connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.get-warnings
*/
```

```
public static function getWarnings() {
	self::checkSetup();
	return self::$link->get_warnings();
}

/**
* Returns the information about this connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.info
*/
public static function getInfo() {
	self::checkSetup();
	return self::$link->info;
}

/**
* Kills the specified MySQL process ID
*
* @param int $processId
* @return bool
* @throws DBNotSetUpException
* @link http://php.net/mysqli.kill
*/
public static function kill($processId) {
	self::checkSetup();
	return self::$link->kill($processId);
}

/**
* Sets a specific mysqli option
*
* @param int $option
* @param mixed $value
* @return bool
* @throws DBNotSetUpException
* @link http://php.net/mysqli.options
*/
public static function options($option, $value) {
	self::checkSetup();
	return self::$link->options($option, $value);
}

/**
* Pings the database
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.ping
*/
public static function ping() {
	self::checkSetup();
	return self::$link->ping();
}

/**
* Escapes the supplied string for insertion into an SQL query. Note that
*  parameterised queries (prepared statements) should be used instead of
*  embedding values in queries
*
* @param string $s
* @return string
* @throws DBNotSetUpException
* @link http://php.net/mysqli.real-escape-string
*/
public static function realEscapeString($s) {
```

```
        self::checkSetup();
        return self::$link->real_escape_string($s);
}

/**
* Escapes the supplied string for insertion into an SQL query. Note that
*  parameterised queries (prepared statements) should be used instead of
*  embedding values in queries
*
* @param string $s
* @return string
* @throws DBNotSetUpException
* @link http://php.net/mysqli.real-escape-string
*/
public static function escape($s) {
        self::checkSetup();
        return self::$link->real_escape_string($s);
}

/**
* Changes the database to the specified one
*
* @param string $db
* @return bool
* @throws DBNotSetUpException
* @link http://php.net/mysqli.select-db
*/
public static function selectDatabase($db) {
        self::checkSetup();
        return self::$link->select_db($db);
}

/**
* Returns status information about the connection
*
* @return mixed
* @throws DBNotSetUpException
* @link http://php.net/mysqli.stat
*/
public static function stat() {
        self::checkSetup();
        return self::$link->stat();
}

/**
* Returns the MySQL thread ID
*
* @return integer
* @throws DBNotSetUpException
* @link http://php.net/mysqli.thread-id
*/
public static function threadId() {
        self::checkSetup();
        return self::$link->thread_id;
}

/**
* Returns the number of warnings that have occurred for this connection
*
* @return integer
* @throws DBNotSetUpException
* @link http://php.net/mysqli.warning-count
*/
public static function warningCount() {
        self::checkSetup();
        return self::$link->warning_count;
}
```

```
}

/**
* Abstract class that forms the basis of the two types of object that get
*  returned by query methods: DBSelectResult & DBUpdateResult
*/
abstract class DBResult {
   /**
   * The mysqli statement object used internally for this query
   *
   * @var mysqli_stmt
   */
   protected $statement;

   /**
   * The duration, in seconds, that this query took to execute
   *
   * @var float
   */
   protected $queryDuration;

   /**
   * Creates a new DBResult object. Can obviously only be called by child
   *  classes
   *
   * @param mysqli_stmt $statement
   * @param array $params
   * @return DBResult
   * @throws DBArrayInQueryParamsException
   * @throws DBQueryFailureException
   * @link http://php.net/mysqli-stmt.bind-result
   * @link http://php.net/mysqli-stmt.execute
   */
   public function __construct(mysqli_stmt $statement, array $params=null) {
          $this->statement = $statement;

          if ($params) {
                 $types = '';
                 foreach ($params as &$p) {
                        switch (gettype($p)) {
                               case 'boolean':
                               case 'integer':
                                      $types .= 'i'; break;
                               case 'double':
                                      $types .= 'd'; break;
                               case 'string':
                               case 'NULL':
                                      $types .= 's'; break;
                               case 'object':
                                      if (method_exists($p, '__toMySQL')) $p = $p->__toMySQL();
                                      else $p = (string) $p;
                                      $types .= 's';
                                      break;
                               case 'resource':
                                      $p = (string) $p;
                                      $types .= 's';
                                      break;
                               case 'array':
                                      DB::failTransactionDueToError();
                                      throw new DBArrayInQueryParamsException();
                        }
                 }
                 array_unshift($params, $types);

                 call_user_func_array(array($this->statement, 'bind_param'), $params);
          }
```

```
		$start = microtime(true);
		$res = $this->statement->execute();
		$this->queryDuration = microtime(true) - $start;

		if (!$res) {
			DB::failTransactionDueToError();
			throw new DBQueryFailureException($this->statement->errno, $this->statement->error);
		}
	}

	/**
	* Frees the memory used by this query's result set when this object is no
	*  longer required
	*
	* @link http://php.net/mysqli-stmt.free-result
	*/
	public function __destruct() {
		if ($this->statement) $this->statement->free_result();
	}

	/**
	* Returns the duration (in seconds) that this query took to execute. If
	*  $round is passed, the result is rounded to that number of decimal places
	*
	* @param integer $round
	* @return float
	*/
	public function getQueryDuration($round=null) {
		if ($round !== null) {
			return number_format(round($this->queryDuration, (int) $round), (int) $round);
		}
		return $this->queryDuration;
	}
}

/**
* The class that makes up objects returned by the DB::updateQuery() method.
*  Provides methods allowing the retrieval of the number of affected rows and
*  the insert ID of the last inserted row
*/
class DBUpdateResult extends DBResult {
	/**
	* Creates a new DBUpdateResult object
	*
	* @param mysqli_stmt $statement
	* @param array $params
	* @return DBUpdateResult
	* @throws DBArrayInQueryParamsException
	* @throws DBQueryFailureException
	* @see DBResult::__construct
	*/
	public function __construct(mysqli_stmt $statement, array $params=null) {
		parent::__construct($statement, $params);
	}

	/**
	* Returns the number of rows affected by this query
	*
	* @return int
	*/
	public function affectedRows() {
		return $this->statement->affected_rows;
	}

	/**
	* Returns the ID (primary key auto-increment value) of the last row inserted
	*
```

```
	 * @return int
	 */
	public function insertId() {
		return $this->statement->insert_id;
	}
}

/**
* The class of objects returned by the DB::selectQuery() (etc.) methods. Can be
*  iterated over using the foreach() construct thanks to its implementation of
*  Iterator
*/
class DBSelectResult extends DBResult implements Iterator {
	/**
	 * The current row being iterated over by foreach()
	 *
	 * @var array
	 */
	private $row;

	/**
	 * Our version of an "empty row", consisting of all of the named columns of
	 *  the resultset, but with NULL values. Used when foreach() gets to the end
	 *  of the array but ->current() is called again...
	 *
	 * @var array
	 */
	private $emptyRow;

	/**
	 * The number of the current record
	 *
	 * @var int
	 */
	private $currentRecord = -1;

	/**
	 * The number of records returned by this result set
	 *
	 * @var int
	 */
	private $numRows;

	/**
	 * Metadata about this query
	 *
	 * @var mixed
	 */
	private $metaData;

	/**
	 * Creates a new DBSelectResult object
	 *
	 * @param mysqli_stmt $statement
	 * @param array $params
	 * @return DBSelectResult
	 * @throws DBArrayInQueryParamsException
	 * @throws DBQueryFailureException
	 * @see DBResult::__construct
	 * @link http://php.net/mysqli-stmt.store-result
	 * @link http://php.net/mysqli-stmt.num-rows
	 */
	public function __construct(mysqli_stmt $statement, array $params=null) {
		parent::__construct($statement, $params);

		$this->statement->store_result();
```

```
		$this->setupResultsBinding();

		$this->numRows = $this->statement->num_rows();
		$this->rewind();
	}

	/**
	* Frees the metadata results if necessary when this object is disposed of.
	*  Also calls the parent __destruct method to free the main result set
	*
	* @see DBResult::__destruct
	* @link http://php.net/mysqli-stmt.free-result
	*/
	public function __destruct() {
		parent::__destruct();
		if ($this->metaData) $this->metaData->free_result();
	}

	//
	/**
	* Sets up the results binding for this query, binding the fields of the
	*  result set to an array of variables so that the result set records can
	*  be accessed later
	*
	* @link http://php.net/mysqli-stmt.result-metadata
	* @link http://php.net/mysqli-stmt.bind-result
	* @link http://php.net/mysqli-stmt.bind-result#85470
	*/
	private function setupResultsBinding() {
		if ($this->row !== null) return;

		$metadata = $this->statement->result_metadata();

		$this->row = array();
		$params = array();
		foreach ($metadata->fetch_fields() as $field) {
			$params[] = &$this->row[$field->name];
		}
		call_user_func_array(array($this->statement, 'bind_result'), $params);

		$this->emptyRow = $this->row;
	}

	/**
	* Returns the number of rows in this query's record set
	*
	*/
	public function getNumRows() {
		return $this->numRows;
	}

	/**
	* Prepares this query's meta data for accessing by other methods
	*
	* @link http://php.net/mysqli-stmt.result-metadata
	*/
	private function prepareMetadata() {
		if ($this->metaData === null) {
			$this->metaData = $this->statement->result_metadata();
		}
		return $this->metaData;
	}

	/**
	* Returns the number of fields (columns) in this result set
	*
	* @return int
```

```
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getNumFields() {
	return $this->prepareMetadata()->num_fields();
}

/**
 * Returns field information about this record set
 *
 * @return mixed
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getFieldInfo() {
	return $this->prepareMetadata()->fetch_field();
}

/**
 * Returns field information about a specific field in this record set
 *
 * @param int $i
 * @return object
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getFieldInfoDirect($i) {
	return $this->prepareMetadata()->fetch_field_direct($i);
}

/**
 * Returns field information about this record set
 *
 * @return mixed
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getFieldsInfo() {
	return $this->prepareMetadata()->fetch_fields();
}

/**
 * Returns the number of fields in this result set
 *
 * @return int
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getFieldCount() {
	return $this->prepareMetadata()->field_count();
}

/**
 * Returns field information about this result set
 *
 * @param int $i
 * @return bool
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getFieldInfoSeek($i) {
	return $this->prepareMetadata()->field_seek($i);
}

/**
 * Returns field information about this result set
 *
 * @return mixed
 * @link http://php.net/mysqli-stmt.result-metadata
 */
public function getFieldInfoTell() {
	return $this->prepareMetadata()->field_tell();
}
```

```
	/**
	* Rewinds this result set as required by the Iterator interface
	*
	* @link http://php.net/mysqli-stmt.data-seek
	* @link http://php.net/mysqli-stmt.fetch
	*/
	public function rewind() {
		if ($this->currentRecord != 0) {
			$this->currentRecord = 0;
			$this->statement->data_seek(0);
			$this->statement->fetch();
		}
	}

	/**
	* Returns the current row of the record set, as required by the Iterator
	*  interface
	*
	* @return array
	*/
	public function current() {
		return $this->row;
	}

	/**
	* Returns the currow row number (key) of this record set as required by the
	*  Iterator interface
	*
	* @return int
	*/
	public function key() {
		return $this->currentRecord;
	}

	/**
	* Advances the record set pointer to the next record and returns the new
	*  current record, or false if there are no more records, as required by
	*  the Iterator interface
	*
	* @return mixed
	*/
	public function next() {
		if ($this->statement->fetch()) {
			$this->currentRecord++;
			return $this->row;
		} else {
			$this->currentRecord = -1;
			$this->row = $this->emptyRow;
			return false;
		}
	}

	/**
	* Returns a boolean indicating if the current record is a valid record
	*
	* @return bool
	*/
	public function valid() {
		return $this->currentRecord >= 0 && $this->currentRecord < $this->numRows;
	}
}

?>
```

classes/errorhandler.php

```php
<?php

/**
* This class handles PHP errors that occur, printing them with full stack trace
*  details in order to help debugging.
*
* This was based on a much larger class that I use for all PHP projects, though
*  this version has been trimmed down considerably
*/
class ErrorHandler {
	/**
	* Recursively implodes the specified $array with the specified $glue.
	*  Converts objects to "{{OBJECT: <class name>}}" and booleans to TRUE or
	*  FALSE
	*
	* @param string $glue
	* @param array $array
	* @return string
	*/
	public static function implodeRecursive($glue, $array) {
		if (!is_array($array)) return $array;
		$out = '';
		foreach ($array as $k => $v) {
			if (!empty($k) && !is_numeric($k)) $out .= "'$k'=>";
			if (is_array($v))
				$out .= '{'.self::implodeRecursive($glue, $v).'}' . $glue;
			elseif (is_object($v))
				$out .= '{{OBJECT: '.get_class($v).'}}' . $glue;
			elseif (is_bool($v))
				$out .= $v ? 'TRUE' : 'FALSE' . $glue;
			else
				$out .= $v . $glue;
		}
		return substr($out, 0, strlen($glue)*-1);
	}

	/**
	* Returns a properly formatted backtrace to aid in debugging. Certain lines
	*  are removed, such as the call to this function, others within this class
	*  and any call to trigger_error
	*
	* @return array
	*/
	private static function debugGetBacktrace() {
		$bt = debug_backtrace();
		array_shift($bt); //this function call
		foreach ($bt as $k => &$v) {
			if (!empty($v['class']) && $v['class'] == 'ErrorHandler') unset($bt[$k]);
			if ($v['function'] == 'trigger_error') unset($bt[$k]);

			if (!isset($v['class'])) $v['class'] = '';
			if (!isset($v['type'])) $v['type'] = '';
			if (!isset($v['function'])) $v['function'] = '?';
			if (!isset($v['file'])) $v['file'] = '?';
			if (!isset($v['line'])) $v['line'] = '?';
		}
		return $bt;
	}

	/**
	* Converts the backtrace array obtained by debugGetBacktrace() to a string
	*
	* @param array $trace
	* @return string
	*/
```

```
	public static function prepareBacktrace($trace=null) {
		if ($trace===null) $trace = self::debugGetBacktrace();
		$r = '';
		$i = 0;
		foreach ($trace as $v) {
			$arguments = self::implodeRecursive(',', (isset($v['args'])?$v['args']:array('?')));

			$r .= "#$i {$v['class']}{$v['type']}{$v['function']}($arguments) called at
[{$v['file']}:{$v['line']}]\r\n";
			++$i;
		}
		if ($i == 0) $r = '[-None-]';
		return $r;
	}

	/**
	* Converts a PHP error number (usually an E_* constant) to a string of the
	*  error type's name
	*
	* @param int $num
	* @return string
	*/
	public static function errorNumToName($num) {
		$type = 'Unknown';
		switch($num) {
			case E_STRICT:
				$type = 'Strict'; break;
			case E_NOTICE:
			case E_USER_NOTICE:
				$type = 'Notice'; break;
			case E_WARNING:
			case E_USER_WARNING:
				$type = 'Warning'; break;
			case E_ERROR:
			case E_USER_ERROR:
				$type = 'Fatal'; break;
			case E_RECOVERABLE_ERROR:
				$type = 'Catchable fatal'; break;
		}
		return $type;
	}

	/**
	* Returns a boolean indicating if the specified error number (a PHP E_*
	*  constant) is a fatal error
	*
	* @param int $num
	* @return bool
	*/
	public static function getIsErrorNumFatal($num) {
		switch($num) {
			case E_ERROR:
			case E_USER_ERROR:
				return true;
		}
		return false;
	}

	/**
	* Returns the HTML for this error's details box
	*
	* @param int $num
	* @param string $msg
	* @param string $file
	* @param int $line
	* @return string
	*/
```

```
    private static function prepareData($num, $msg, $file, $line) {
            $type = self::errorNumToName($num);
            $backtrace = html(self::prepareBacktrace());
            return '<div class="phperror">'.
                        "<p>A $type error occurred on line $line of $file:</p>".
                        "<pre>$msg</pre>".
                        '<p><strong>Stack Trace</strong></p>'.
                        '<pre>'.
                        $backtrace.
                        '</pre>'.
                        '</div>';
    }

    /**
    * The function that's set as the PHP error handler
    *
    * @param int $num
    * @param string $msg
    * @param string $file
    * @param int $line
    */
    public static function handleError($num, $msg, $file, $line) {
            if (!error_reporting()) return; //Must've been preceeded with @

            echo self::prepareData($num, $msg, $file, $line);

            if (self::getIsErrorNumFatal($num)) die();
    }
}

?>
```

classes/exceptions.php

```php
<?php

/*
* A load of exceptions that are thrown from various classes within the system.
*  There's not really any point in documenting each of them individually
*/

class DateIsNotMySQLFormatException extends Exception {}

class DBException extends Exception {}
class DBNotSetUpException extends DBException {}
class DBNullStatementException extends DBException {}
class DBConnectionException extends DBException {
   private $errNo;
   private $errMsg;
   public function __construct($errNo, $errMsg) {
          parent::__construct("MySQL Connection Error: $errMsg", $errNo);
          $this->errNo = $errNo;
          $this->errMsg = $errMsg;
   }
}
class DBArrayInQueryParamsException extends DBException {}
class DBQueryFailureException extends DBException {
   private $errNo;
   private $errMsg;
   public function __construct($errNo, $errMsg) {
          parent::__construct("MySQL Query Execution Error: $errMsg [$errNo]", $errNo);
          $this->errNo = $errNo;
          $this->errMsg = $errMsg;
   }
}
class DBQueryPreparationFailureException extends DBException {
   private $errNo;
   private $errMsg;
   public function __construct($errNo, $errMsg) {
          parent::__construct("MySQL Query Preparation Error: $errMsg [$errNo]", $errNo);
          $this->errNo = $errNo;
          $this->errMsg = $errMsg;
   }
}
class DBTransactionAlreadyOpenException extends DBException {}
class DBTransactionNotOpenException extends DBException {}

class InvalidProductException extends Exception {}

class UserNotLoggedInException extends Exception {}

?>
```

classes/formatting.php

```php
<?php

/**
* Originally based on a standard formatting class that I use for all of my PHP
*  projects, only the formatPrice() method remains, so I'm not sure it can
*  really be described as being "based on" anything any more
*/
class Formatting {
   /**
    * Returns $price formatted as a price
    *
    * @param float $price
    * @return string
    */
   public static function formatPrice($price) {
         return '£' . number_format($price, 2, '.', ',');
   }
}

?>
```

classes/orders.php

```php
<?php

/**
* This class handles operations relating to multiple orders, such as fetching
*  a list of orders
*/
class Orders {
   /**
   * Returns an array of Order objects representing each order that the
   *  specified customer has placed
   *
   * @param User $customer
   * @return array
   */
   public static function getAllOrdersForCustomer(User $customer) {
          $res = DB::selectQuery('SELECT id,date FROM orders WHERE customer=?', array($customer));
          $out = array();
          foreach ($res as $row) {
                $out[] = new Order($row['id'], $customer, Date::createFromMySQL($row['date']));
          }
          return $out;
   }
}

/**
* Represents a single customer order
*/
class Order {
   /**
   * ID of the order in the database
   *
   * @var int
   */
    private $id;

    /**
    * The customer who placed this order
    *
    * @var User
    */
    private $customer;

    /**
    * The date & time at which this order was placed
    *
    * @var Date
    */
    private $date;

    /**
    * An array of every product purchased as part of this order
    *
    * @var array
    */
    private $orderedProducts = array();

   /**
   * Creates a new instance of the Order class
   *
   * @param int $id
   * @param User $customer
   * @param Date $date
   * @return Order
   */
   public function __construct($id, User $customer, Date $date) {
```

```
        $this->id = $id;
        $this->customer = $customer;
        $this->date = $date;

        $res = DB::selectQuery('SELECT * FROM orderedproducts WHERE `order`=?', array($id));
        foreach ($res as $row) {
              $rec = $row['recipient'] ? Users::getUserById($row['recipient']) : null;
              $hu = $row['hideuntil'] ? Date::createFromMySQL($row['hideuntil']) : null;
              $this->orderedProducts[] = new OrderedProduct($id,
Product::createFromId($row['product']), $row['quantity'], $rec, $hu, (bool) $row['purchaseruse'],
(bool) $row['recipientuse']);
        }
   }

   /**
   * Returns an Order object representing the order with the specified ID
   *
   * @param int $id
   * @return Order
   */
   public static function createFromId($id) {
        $row = DB::selectRow('SELECT * FROM orders WHERE id=?', array($id));
        return new Order($id, Users::getUserById($row['customer']),
Date::createFromMySQL($row['date']));
   }

    /**
    * Returns this order's ID
    *
    * @return int
    */
    public function getId() {
        return $this->id;
    }

    /**
    * Returns the customer who placed this order
    *
    * @return User
    */
    public function getCustomer() {
        return $this->customer;
    }

    /**
    * Returns the date at which this order was placed
    *
    * @return Date
    */
    public function getDate() {
        return $this->date;
    }

    /**
    * Returns an array of products that were ordered as part of this order
    *
    * @return array
    */
    public function getOrderedProducts() {
        return $this->orderedProducts;
    }

    /**
    * Returns this order's ID, for MySQL
    *
    * @return int
    */
```

```
    public function __toMySQL() {
         return $this->id;
    }
}

/**
* Class to represent individual products purchased as part of an order
*/
class OrderedProduct {
   /**
   * The ID of the order that this product is a part of
   *
   * @var int
   */
    private $orderId;

    /**
    * The product that was ordered
    *
    * @var Product
    */
    private $product;

    /**
    * The quantity of the product that was ordered
    *
    * @var int
    */
    private $quantity;

    /**
    * The recipient (if there is one) of this purchase
    *
    * @var User
    */
    private $recipient;

    /**
    * The optional date until which this product will be hidden from its
    *  recipient
    *
    * @var Date
    */
    private $hideUntil;

    /**
    * Whether this purchase will be used to recommend products to the purchaser
    *
    * @var bool
    */
    private $purchaserUse;

    /**
    * Whether this purchase will be used to recommend products to the recipient
    *
    * @var bool
    */
    private $recipientUse;

   /**
   * Creates a new OrderedProduct object
   *
   * @param int $orderId
   * @param Product $product
   * @param int $quantity
   * @param User $recipient
   * @param Date $hideUntil
```

```
    * @param bool $purchaserUse
    * @param bool $recipientUse
    * @return OrderedProduct
    */
   public function __construct($orderId, Product $product, $quantity, User $recipient=null, Date
$hideUntil=null, $purchaserUse, $recipientUse) {
          $this->orderId = $orderId;
          $this->product = $product;
          $this->quantity = $quantity;
          $this->recipient = $recipient;
          $this->hideUntil = $hideUntil;
          $this->purchaserUse = $purchaserUse;
          $this->recipientUse = $recipientUse;
   }

   /**
    * Returns the ID of this order
    *
    * @return int
    */
   public function getOrderId() {
          return $this->orderId;
   }

   /**
    * Returns the product that was purchased
    *
    * @return Product
    */
     public function getProduct() {
          return $this->product;
     }

     /**
     * Returns the quantity of product that was ordered
     *
     * @return int
     */
     public function getQuantity() {
          return $this->quantity;
     }

     /**
     * Returns the optional recipient of this product
     *
     * @return User
     */
     public function getRecipient() {
          return $this->recipient;
     }

     /**
     * Returns a User object for the customer who purchased this item
     *
     * @return User
     */
     public function getPurchaser() {
          return Users::getUserById(DB::selectOne('SELECT customer FROM orders WHERE id=?',
array($this->orderId)));
     }

     /**
     * Returns the optional hide until date for this purchase
     *
     * @return Date
     */
     public function getHideUntil() {
```

```
            return $this->hideUntil;
        }

        /**
        * Sets whether this purchase will be used to recommend products to the
        *  purchaser
        *
        * @param bool $purchaserUse
        */
        public function setPurchaserUse($purchaserUse) {
            $this->purchaserUse = $purchaserUse;

            DB::updateQuery('UPDATE orderedproducts SET purchaseruse=? WHERE `order`=? AND product=?',
array($purchaserUse, $this->orderId, $this->product));
        }

        /**
        * Returns whether this purchase will be used to recommend products to the
        *  purchaser
        *
        * @return bool
        */
        public function getPurchaserUse() {
            return $this->purchaserUse;
        }

        /**
        * Sets whether this purchase will be used to recommend products to the
        *  recipient
        *
        * @param bool $recipientUse
        */
        public function setRecipientUse($recipientUse) {
            $this->recipientUse = $recipientUse;

            DB::updateQuery('UPDATE orderedproducts SET recipientuse=? WHERE `order`=? AND product=?',
array($recipientUse, $this->orderId, $this->product));
        }

        /**
        * Returns whether this purchase will be used to recommend products to the
        *  recipient
        *
        * @return bool
        */
        public function getRecipientUse() {
            return $this->recipientUse;
        }
}

?>
```

classes/products.php

```php
<?php

/**
* This class handles groups of products (e.g. returning lists), as well as
*  product rating
*/
class Products {
   /**
   * Returns an array of all products in the database
   *
   * @return array
   */
  public static function getAllProducts() {
         $list = DB::selectQuery('SELECT * FROM ratedproducts');
         $out = array();
         foreach ($list as $row) {
               $out[] = Product::createFromRow($row);
         }
         return $out;
   }

   /**
   * Rates the product with the specified ID with the specified rating
   *  (int 1-5). Returns FALSE on failure, or a DBUpdateResult object on
   *  potential success
   *
   * @param int $id
   * @param int $rating
   * @return mixed
   */
  public static function rateProductById($id, $rating) {
         $user = Users::getCurrentUser();
         if ($user) {
               return DB::updateQuery('INSERT INTO productratings (product,customer,rating) VALUES
(?,?,?)', array($id, $user, $rating));
         } else {
               return false;
         }
   }

   /**
   * Returns a boolean indicating if the current user has rated the specified
   *  product
   *
   * @param int $id
   * @return bool
   */
  public static function getHasUserRatedProduct($id) {
         if ($user = Users::getCurrentUser()) {
               return (bool) DB::selectOne('SELECT COUNT(*) FROM productratings WHERE product=? AND
customer=?', array($id, $user));
         } else {
               return false;
         }
   }

   /**
   * Returns an array of products that are used to generate recommendations
   *  for the specified user. The returned array will contain objects of type
   *  both OrderedProduct and RatedProduct, so watch out!
   *
   * @param User $user
   * @return array
   */
  public static function getProductsUsedForRecommendation(User $user) {
```

```
		$res = DB::selectQuery('SELECT * FROM orderedproducts AS op LEFT JOIN orders AS o ON
op.`order`=o.id WHERE (o.customer=? AND op.purchaseruse=1 AND op.recipient IS NULL) OR (op.recipient=?
AND op.recipientuse=1 AND (op.hideuntil IS NULL OR op.hideuntil < UTC_TIMESTAMP()))', array($user,
$user));
		$out = array();
		foreach ($res as $row) {
			$rec = $row['recipient'] ? Users::getUserById($row['recipient']) : null;
			$hu = $row['hideuntil'] ? Date::createFromMySQL($row['hideuntil']) : null;
			$out[] = new OrderedProduct($row['order'], Product::createFromId($row['product']),
$row['quantity'], $rec, $hu, $row['purchaseruse'], $row['recipientuse']);
		}

		$res = DB::selectQuery('SELECT product,rating FROM productratings WHERE customer=? AND
product NOT IN (SELECT product FROM exceptions WHERE customer=?)', array($user, $user));
		foreach ($res as $row) {
			$out[] = new RatedProduct(Product::createFromId($row['product']), $row['rating']);
		}
		return $out;
	}

	/**
	* Returns an array of all products that could possibly be used for
	*  recommending other products, ignoring exceptions and
	*  "purchaseruse"/"recipientuse" flags. Useful for allowing customers to
	*  re-enable products for recommendation generation
	*
	* @param User $user
	* @return array
	*/
	public static function getProductsAvailableForRecommendation(User $user) {
		$res = DB::selectQuery('SELECT * FROM orderedproducts AS op LEFT JOIN orders AS o ON
op.`order`=o.id WHERE (o.customer=? AND op.recipient IS NULL) OR (op.recipient=? AND (op.hideuntil IS
NULL OR op.hideuntil < UTC_TIMESTAMP()))', array($user, $user));
		$out = array();
		foreach ($res as $row) {
			$rec = $row['recipient'] ? Users::getUserById($row['recipient']) : null;
			$hu = $row['hideuntil'] ? Date::createFromMySQL($row['hideuntil']) : null;
			$out[] = new OrderedProduct($row['order'], Product::createFromId($row['product']),
$row['quantity'], $rec, $hu, $row['purchaseruse'], $row['recipientuse']);
		}

		$res = DB::selectQuery('SELECT product,rating FROM productratings WHERE customer=?',
array($user));
		foreach ($res as $row) {
			$out[] = new RatedProduct(Product::createFromId($row['product']), $row['rating']);
		}
		return $out;
	}

	/**
	* Returns an array of products that are recommended for the specified user.
	*  The returned array will consist solely of SimilarProduct objects.  The
	*  SQL query is a little monstrous; the first section uses products
	*  purchased BY the current user, the second uses products purchased FOR
	*  the current user, and the third section uses products RATED by the
	*  current user
	*
	* @param User $user
	* @return array
	*/
	public static function getRecommendedProducts(User $user) {
		$res = DB::selectQuery('
			SELECT DISTINCT * FROM
			(
				SELECT p.*, ps.similarity AS weightedsimilarity
				FROM ratedproducts AS p
				LEFT JOIN productsimilarity AS ps ON p.id=ps.producta
```

```
                        WHERE
                                ps.productb IN (
                                        SELECT op.product
                                        FROM orderedproducts AS op
                                        LEFT JOIN orders AS o ON op.`order`=o.id
                                        WHERE
                                                (o.customer=? AND op.purchaseruse=1 AND op.recipient
IS NULL)
                                        )
                        UNION
                                SELECT p.*, (ps.similarity/2) AS weightedsimilarity
                                FROM ratedproducts AS p
                                LEFT JOIN productsimilarity AS ps ON p.id=ps.producta
                                WHERE
                                        ps.productb IN (
                                                SELECT op.product
                                                FROM orderedproducts AS op
                                                LEFT JOIN orders AS o ON op.`order`=o.id
                                                WHERE
                                                        (op.recipient=? AND op.recipientuse=1 AND
(op.hideuntil IS NULL OR op.hideuntil < UTC_TIMESTAMP()))
                                        )
                        UNION
                                SELECT p.*, (ps.similarity/2) AS weightedsimilarity
                                FROM ratedproducts AS p
                                LEFT JOIN productsimilarity AS ps ON p.id=ps.producta
                                WHERE ps.productb IN (
                                        SELECT product FROM productratings WHERE customer=? AND product
NOT IN (SELECT product FROM exceptions WHERE customer=?)
                                        )
                ) AS r
                WHERE r.id NOT IN (SELECT product FROM orderedproducts WHERE (`order` IN (SELECT id
FROM orders WHERE customer=?) AND recipient IS NULL) OR (recipient=?))
                ORDER BY weightedsimilarity DESC
                ', array($user, $user, $user, $user, $user, $user));
        $out = array();
        foreach ($res as $row) {
                $out[] = SimilarProduct::createFromRow($row);
        }
        return $out;
    }
}

/**
* Represents a single product
*/
class Product {
    /**
    * The product's ID
    *
    * @var int
    */
    private $id;

    /**
    * The products name
    *
    * @var string
    */
    private $name;

    /**
    * The product's price as a float
    *
    * @var float
    */
    private $price;
```

```
/**
* The product's image filename, minus path and file extension
*
* @var string
*/
private $image;

/**
* The product's rating as a float
*
* @var float
*/
private $rating;

/**
* The number of ratings that this product has received
*
* @var int
*/
private $ratingCount;

/**
* Creates a new instance of this class. Should only be called from the
*  Products class or from within this class
*
* @param int $id
* @param string $name
* @param float $price
* @param string $image
* @param float $rating
* @param int $ratingCount
* @return Product
*/
public function __construct($id, $name, $price, $image, $rating, $ratingCount) {
	$this->id = $id;
	$this->name = $name;
	$this->price = $price;
	$this->image = $image;
	$this->rating = $rating;
	$this->ratingCount = $ratingCount;
}

/**
* Creates a new Product object from the specified product ID
*
* @param int $id
* @return Product
* @throws InvalidProductException
*/
public static function createFromId($id) {
	$res = DB::selectRow('SELECT * FROM ratedproducts WHERE id=?', array($id));
	if ($res) {
		return self::createFromRow($res);
	}
	throw new InvalidProductException();
}

/**
* Creates a new Product object from the supplied product details. Should
*  only be called from within the Products or Product classes
*
* @param array $row
* @return Product
*/
public static function createFromRow(array $row) {
	return new self($row['id'], $row['name'], $row['price'], $row['image'], $row['rating'],
```

```
$row['ratingcount']);
    }

    /**
    * Returns the ID of this product
    *
    * @return int
    */
    public function getId() {
            return $this->id;
    }

    /**
    * Returns the name of this product
    *
    * @return string
    */
    public function getName() {
            return $this->name;
    }

    /**
    * Returns the price of this product as a float
    *
    * @return float
    */
    public function getPrice() {
            return $this->price;
    }

    /**
    * Returns the price of this product as a formatted string
    *
    * @return string
    */
    public function getFormattedPrice() {
            return Formatting::formatPrice($this->price);
    }

    /**
    * Returns the image filename (minus path & extension) of this product's
    *  image
    *
    * @return string
    */
    public function getImage() {
            return $this->image;
    }

    /**
    * Returns the rating of this product
    *
    * @return float
    */
    public function getRating() {
            return $this->rating;
    }

    /**
    * Returns the rating of this product rounded to two decimal places
    *
    */
    public function getFormattedRating() {
            return round($this->rating, 2);
    }

    /**
```

```
	* Returns the number of ratings that this product has received
	*
	*/
	public function getRatingCount() {
		return $this->ratingCount;
	}

	/**
	* Returns this product's ID for MySQL
	*
	*/
	public function __toMySQL() {
		return $this->id;
	}

	/**
	* Returns a boolean indicating if the current user has rated this product
	*
	* @return bool
	*/
	public function getHasUserRatedProduct() {
		if ($user = Users::getCurrentUser()) {
			return (bool) DB::selectOne('SELECT COUNT(*) FROM productratings WHERE product=? AND
customer=?', array($this->id, $user));
		} else {
			return false;
		}
	}
}

/**
* Extension of the Product class that includes a similarity attribute to allow
*  recommendation lists to indicate how similar each recommendation is to the
*  product(s) that generated its recommendation
*/
class SimilarProduct extends Product {
	/**
	* Similarity value of this product
	*
	* @var float
	*/
	private $similarity;

	/**
	* Creates a new instance of this class. Should only be called from this class or Products
	*
	* @param int $id
	* @param string $name
	* @param float $price
	* @param string $image
	* @param float $rating
	* @param int $ratingCount
	* @param float $similarity
	* @return SimilarProduct
	*/
	public function __construct($id, $name, $price, $image, $rating, $ratingCount, $similarity) {
		parent::__construct($id, $name, $price, $image, $rating, $ratingCount);
		$this->similarity = $similarity;
	}

	/**
	* Returns a new instance of this class using the supplied details
	*
	* @param array $row
	* @return SimilarProduct
	*/
	public static function createFromRow(array $row) {
```

```
        return new self($row['id'], $row['name'], $row['price'], $row['image'], $row['rating'],
$row['ratingcount'], $row['weightedsimilarity']);
    }

    /**
    * Returns the similarity that this recommended product has to the products
    *  that generated this recommendation. If $round is specified, the returned
    *  value is rounded to that number of decimal places
    *
    * @param int $round
    * @return float
    */
    public function getSimilarity($round=null) {
        if ($round !== null) {
            return number_format(round($this->similarity, (int) $round), (int) $round);
        }
        return $this->similarity;
    }
}

/**
* Similar to the Product class, but is used instead of OrderedProduct to
*  indicate a product that has been rated by the user
*/
class RatedProduct {
    /**
    * The product in question
    *
    * @var Product
    */
     private $product;

     /**
     * The rating that the user who rated this product gave it
     *
     * @var int
     */
     private $rating;

    /**
    * Creates a new instance of this class; should only be called from the Products class
    *
    * @param Product $product
    * @param int $rating
    * @return RatedProduct
    */
    public function __construct(Product $product, $rating) {
        $this->product = $product;
        $this->rating = $rating;
    }

    /**
    * Returns the Product object that this rated product relates to
    *
    * @return Product
    */
     public function getProduct() {
        return $this->product;
     }

     /**
     * Returns the rating that the user who rated this product gave it
     *
     * @return int
     */
     public function getRating() {
        return $this->rating;
```

```
    }

    /**
    * Returns a boolean indicating if this product is used to recommend further products to the
current user
    *
    * @return bool
    * @throws UserNotLoggedInException
    */
    public function isUsedForRecommendations() {
         if ($user = Users::getCurrentUser()) {
               return (bool) (DB::selectOne('SELECT COUNT(*) FROM exceptions WHERE customer=? AND
product=?', array($user, $this->product)) < 1);
         } else {
               throw new UserNotLoggedInException();
         }
    }

    /**
    * Toggles whether this product is used to recommend products to the current user
    *
    * @throws UserNotLoggedInException
    */
    public function toggleUse() {
         if ($user = Users::getCurrentUser()) {
               if (!$this->isUsedForRecommendations()) {
                     DB::updateQuery('DELETE FROM exceptions WHERE customer=? AND product=?',
array($user, $this->product));
               } else {
                     DB::updateQuery('INSERT INTO exceptions (customer, product) VALUES (?,?)',
array($user, $this->product));
               }
         } else {
               throw new UserNotLoggedInException();
         }
    }
}

?>
```

classes/users.php

```php
<?php

/**
* Class to handle groups of users, such as retrieving lists of users. Also
*  handles logging in.
*/
class Users {
   /**
   * The user who is currently logged in - if someone is logged in
   *
   * @var User
   */
   private static $currentUser;

   /**
   * A list of all users in the system
   *
   * @var array
   */
   private static $allUsers;

   /**
   * Fills the $allUsers array with a User object for each customer in the
   *  database. This saves on resources for this system, but would be entirely
   *  inappropriate in a system with an unlimited number of users in the
   *  database
   *
   */
   private static function populateUsers() {
          $res = DB::selectQuery('SELECT * FROM customers');
          foreach ($res as $row) {
                 self::$allUsers[] = new User($row['id'], $row['name'], $row['address']);
          }
   }

   /**
   * Returns an array of all users in the system
   *
   * @return array
   */
   public static function getAllUsers() {
          if (self::$allUsers === null) self::populateUsers();
          return self::$allUsers;
   }

   /**
   * Returns a single User object for the customer with the specified ID
   *
   * @param int $id
   * @return User
   */
   public static function getUserById($id) {
          if (self::$allUsers === null) self::populateUsers();
          foreach (self::$allUsers as $user) {
                 if ($user->getId() == $id) {
                        return $user;
                 }
          }
          return null;
   }

   /**
   * Returns a User object for the currently logged in user, if there is one
   *
   * @return User
```

```
	*/
	public static function getCurrentUser() {
		if (isset($_SESSION['userid'])) {
			return self::getUserById($_SESSION['userid']);
		}
		return null;
	}

	/**
	* Logs the user in as the specified customer
	*
	* @param User $newUser
	*/
	public static function switchUser(User $newUser) {
		if ($newUser !== null) {
			$_SESSION['userid'] = $newUser->getId();
		}
	}

	/**
	* Logs out the user
	*
	*/
	public static function logOut() {
		unset($_SESSION['userid']);
	}
}

/**
* Stores the details of a single user (customer)
*/
class User {
	/**
	* The customer ID
	*
	* @var int
	*/
	private $id;

	/**
	* The customer's name
	*
	* @var string
	*/
	private $name;

	/**
	* The customer's address
	*
	* @var string
	*/
	private $address;

	/**
	* Creates a new User object. Do not call directly, except from the User class.
	*
	* @param int $id
	* @param string $name
	* @param string $address
	* @return User
	*/
	public function __construct($id, $name, $address) {
		$this->id = $id;
		$this->name = $name;
		$this->address = $address;
	}
```

```
    /**
     * Returns the ID of this customer
     *
     * @return int
     */
    public function getId() {
            return $this->id;
    }

    /**
     * Returns the name of this customer
     *
     * @return string
     */
    public function getName() {
            return $this->name;
    }

    /**
     * Returns the address of this customer
     *
     * @return string
     */
    public function getAddress() {
            return $this->address;
    }

    /**
     * Returns the ID of this customer for MySQL
     *
     * @return int
     */
    public function __toMySQL() {
            return $this->id;
    }

    /**
     * Determines if this customer is equal to the specified one
     *
     * @param User $b
     * @return bool
     */
    public function equals(User $b) {
            return $this->id == $b->getId();
    }
}

?>
```

classes/utility.php

```php
<?php

/**
 * Miscellaneous functions that don't fit anywhere else.
 *
 * This is a heavily trimmed and slightly extended version of a class that I
 *  use for most of my PHP projects/sites
 */
class Utility {
   /**
    * Returns the GET variable with the specified name, $v.
    * Returns NULL if the variable doesn't exist
    *
    * @param string $v
    * @param mixed $d
    * @return string|null
    */
   public static function getGET($v, $d=null) {
          if (isset($_GET[$v])) return $_GET[$v];
          return $d;
   }

   /**
    * Returns the POST variable with the specified name, $v.
    * Returns NULL if the variable doesn't exist
    *
    * @param string $v
    * @param mixed $d
    * @return string|null
    */
   public static function getPOST($v, $d=null) {
          if (isset($_POST[$v])) return $_POST[$v];
          return $d;
   }

   /**
    * Returns the COOKIE variable with the specified name, $v.
    * Returns NULL if the variable doesn't exist
    *
    * @param string $v
    * @param mixed $d
    * @return string|null
    */
   public static function getCOOKIE($v, $d=null) {
          $v = self::$SettingCookiePrefix . $v;
          if (isset($_COOKIE[$v])) return $_COOKIE[$v];
          return $d;
   }

   /**
    * Returns the SERVER variable with the specified name, $v.
    * Returns NULL if the variable doesn't exist
    *
    * @param string $v
    * @param mixed $d
    * @return string|null
    */
   public static function getSERVER($v, $d=null) {
          if (isset($_SERVER[$v])) return $_SERVER[$v];
          return $d;
   }

   /**
   * Outputs the HTML for a standardised error message, complete with title
   *  and message (if they're provided)
```

```
	*
	* @param string $title
	* @param string $message
	*/
	public static function error($title=null, $message=null) {
		?>
		<div class="error">
			<? if ($title !== null) { ?>
				<h1><?=html($title)?></h1>
			<? } ?>

			<? if ($message) { ?>
				<p><?=nl2br(html($message))?></p>
			<? } ?>
		</div>
		<?
	}

	/**
	* Outputs the HTML for a standardised success message, complete with title
	*  and message (if they're provided)
	*
	* @param string $title
	* @param string $message
	*/
	public static function success($title=null, $message=null) {
		?>
		<div class="success">
			<? if ($title !== null) { ?>
				<h1><?=html($title)?></h1>
			<? } ?>

			<? if ($message) { ?>
				<p><?=nl2br(html($message))?></p>
			<? } ?>
		</div>
		<?
	}
}

?>
```

classes/validation.php

```php
<?php

/**
 * A class to handle validation of user input on the server side
 *
 * This class is a somewhat modified version of a validation class that I've
 *  used in various projects/sites previously.
 *
 */
class Validation {
   /**
    * Returns a boolean indicating if the supplied $Value was provided
    *  (more than 0 characters long)
    *
    * @param string $value
    * @return boolean
    */
   public static function required($value) {
          return (!empty($value));
   }

   /**
    * Returns a boolean indicating if the supplied $Value is numeric.
    * This could be an integer or a float
    *
    * @param string $value
    * @return boolean
    */
   public static function numeric($value) {
          return (!is_null($value) && is_numeric($value));
   }

   /**
    * Returns a boolean indicating if the supplied $value is numeric,
    *  (a float or integer) and between $min and $max
    *
    * @param string $value
    * @return boolean
    */
   public static function range($value, $min, $max) {
          return (!is_null($value) && is_numeric($value) && $value >= $min && $value <= $max);
   }

   /**
    * Returns a boolean indicating if the supplied $value is an integer
    *
    * @param string $value
    * @return boolean
    */
   public static function integer($value) {
          return (!is_null($value) && $value == (int) $value);
   }

   /**
    * Returns a boolean indicating if the supplied $value is an integer and
    *  between $min and $max
    *
    * @param string $value
    * @return boolean
    */
   public static function rangeInteger($value, $min, $max) {
          return (!is_null($value) && ($value == (int) $value) && ($value >= $min && $value <= $max));
   }

   /**
```

```
    * Returns a boolean indicating if the supplied $value's length is between
    *  $Min and $Max
    *
    * @param string $Value
    * @return boolean
    */
   public static function length($value, $min, $max) {
          $s = ustrlen($value);
          return (!is_null($value) && $s >= $min && $s <= $max);
   }

   /**
    * Returns a boolean indicating if the supplied $value matches the regular
    *  expression $pattern
    *
    * @param string $value
    * @return boolean
    */
   public static function regex($value, $pattern) {
          return (!is_null($value) && preg_match($pattern.'u', $value) > 0);
   }
}

?>
```

inc/blocks/cart.php

```
<h2>Cart</h2>
<?
if (isset($_POST['addproduct'])) {
   Cart::addItem(new CartItem((int)$_POST['addproduct'], (int)$_POST['addproductqty']));
}
if (isset($_POST['delproduct'])) {
   //$_POST['delproduct'] is an array containing a single element that we want
   // to delete. Easier to do it this way than any alternative - in the HTML
   // for the form, at least
   //I suppose that something like
   //   reset($_POST['delproduct']);
   //   $id = key($_POST['delproduct']);
   // would also have worked here
   foreach ($_POST['delproduct'] as $id=>$x) {
         Cart::removeItemById($id);
   }
}
?>
<? if (Cart::getItems()) {?>
   <form action="" method="post">
         <fieldset style="padding-bottom: 0px;">
               <ul style="padding-left: 10px;">
                     <? foreach (Cart::getItems() as /** @var CartItem */ $item) { ?>
                           <li>
                                 <?=$item->getQuantity()?>x
                                 <?= html($item->getProduct()->getName()) ?>
                                 (<?=html($item->getFormattedLinePrice())?>)
                                 <input type="submit" name="delproduct[<?=$item->getId()?>]"
value="x" />
                           </li>
                     <? } ?>
               </ul>
               <p class="carttotalprice">Total Price: <?=html(Cart::getFormattedTotalPrice())?></p>
         </fieldset>
   </form>
   <form action="" method="get">
         <fieldset>
               <input type="hidden" name="page" value="checkout" />
               <input type="submit" value="Checkout" />
         </fieldset>
   </form>
   <p class="proto">A production system would have more controls (adjusting quantity, etc.), and
possibly a full page for viewing the cart.</p>
<? } else { ?>
   <p>Your cart is empty.</p>
<? } ?>
```

inc/blocks/login.php

```
<?
if (isset($_POST['newuser'])) {
   //User wants to switch to a new customer or log out (userid 0 = log out)
   $newUserId = (int) $_POST['newuser'];
   if ($newUserId > 0) {
          Users::switchUser(Users::getUserById($newUserId));
   } else {
          Users::logOut();
   }
}
?>

<? if ($currentUser = Users::getCurrentUser()) { ?>
   <p>Currently logged in as: <strong><?=html($currentUser->getName())?></strong>.</p>

   <p style="margin-bottom: 0px;">Select a user to switch to:</p>
   <? $loginButtonText = 'Switch User'; ?>
<? } else { ?>
   <p>Not currently logged in.</p>

   <p style="margin-bottom: 0px;">Select a user to log in as:</p>
   <? $loginButtonText = 'Log In'; ?>
<? } ?>

<form action="" method="post">
   <fieldset style="text-align: center;">
          <select name="newuser" class="loginbox">
                 <option value="0">[Log Out]</option>
                 <? foreach (Users::getAllUsers() as $user) { ?>
                        <option value="<?=$user->getId()?>"><?=html($user->getName())?></option>
                 <? } ?>
          </select><br />
          <input type="submit" value="<?=$loginButtonText?>" />
   </fieldset>
</form>

<p class="proto">Note that in a production system, this would be a standard login dialog, with
username and password boxes, etc.</p>
```

inc/blocks/navigation.php

```
<h2>Navigation</h2>
<ul class="nav">
   <li><a href="./">Home</a></li>
   <li><a href="?page=orders">View Past Orders</a></li>
   <li><a href="?page=recommendation-products">Products Used for Recommendations</a></li>
   <li><a href="?page=recommendations">View Recommendations</a></li>
   <li><a href="?page=calculate-similarities">Calculate Similarities</a></li>
</ul>
```

inc/pages/404.php

```
<h2>404 Not Found</h2>
<? Utility::error('404 Not Found', 'The page you requested does not exist.'); ?>
```

inc/pages/calculate-similarities.php

```
<h2>Calculate Product Similarities</h2>
<p>This page allows you to re-populate the `productsimilarity` table, calculating the similarities
between each product in the database. With large data sets (customers, purchases, ratings and
products), this can take a significant amount of time.</p>

<p class="proto">Obviously this page wouldn't exist in a production system, however it makes the
process of re-populating the `productsimilarity` table during development and demonstration
considerably easier.</p>

<?
$startOrRepeat = 'start';
if (isset($_POST['recalculate'])) {
   //Form submitted - call the procedure!
   $res = DB::updateQuery('CALL calculatesimilarities()');
   Utility::success('Calculation Complete', 'The product similarity calculation process is complete.
The database query executed in ' . $res->getQueryDuration(3) . ' seconds.');
   $startOrRepeat = 'repeat';
}
?>

<p>Click the button below to <?=$startOrRepeat?> the process.</p>
<form action="" method="post">
   <fieldset>
         <input type="submit" name="recalculate" id="recalculatebutton"
value="<?=ucwords($startOrRepeat)?> Calculation Process" />
   </fieldset>
</form>
```

inc/pages/checkout.php

```
<h2>Checkout</h2>

<p>Here you can convert the products in your cart into an order, optionally specifying for each
product a recipient (if the purchase is a gift for someone) and a date until which it will remain
hidden from that recipient - to avoid spoiling surprise gifts.  Specifying a recipient for a purchase
allows that purchase to improve the recipient's future recommendations and gift recommendations for
others searching for a gift idea for that recipient.</p>

<? if (Users::getCurrentUser()) { ?>
  <? if (Cart::getItems()) {?>
        <?
        $showList = true;
        $recipientErrors = array();
        $hideUntilErrors = array();
        if (isset($_POST['checkout'])) {
              //Want to submit the form
              foreach (Cart::getItems() as /** @var CartItem */ $item) {
                    $id = $item->getId();
                    if (!empty($_POST['recipient'][$id])) {
                          //Set the recipient of this product
                          if ($rec = Users::getUserById($_POST['recipient'][$id])) {
                                $item->setRecipient($rec);
                          } else {
                                $recipientErrors[] = $id;
                          }
                    }
                    if (!empty($_POST['hideuntil'][$id])) {
                          //And set its hide-until date
                          if (preg_match('/^(\d{4})-(\d{2})-(\d{2})$/', $_POST['hideuntil'][$id],
$hum)) {
                                if ($hu = Date::createFromParts($hum[1], $hum[2], $hum[3], 0, 0,
0)) {
                                      $item->setHideUntil($hu);
                                } else {
                                      $hideUntilErrors[] = $id;
                                }
                          } else {
                                $hideUntilErrors[] = $id;
                          }
                    }
              }

              if ($hideUntilErrors || $recipientErrors) {
                    Utility::error('Checkout Errors', 'There were errors in your submission. Please
correct these and try again.');
              } else {
                    //No errors, convert it to an order!
                    if (Cart::convertToOrder()) {
                          $showList = false;
                          Utility::success('Order Placed', 'Your order has been successfully placed
and your cart has been emptied.');
                          //The cart doesn't actually appear to be empty at this point
                          // because it's already been rendered. It will be empty on
                          // the next page load, though
                    } else {
                          Utility::error('Order Failed', 'An unexpected error occurred while
placing your order.');
                    }
              }
        }
        ?>
        <? if ($showList) { ?>
              <form action="" method="post">
                    <fieldset class="checkoutform">
                          <div class="tablesur">
```

```
                                <table border="1" cellspacing="0" cellpadding="3">
                                        <thead>
                                                <tr>
                                                        <th>Product Name</th>
                                                        <th>Product Price</th>
                                                        <th>Quantity</th>
                                                        <th>Line Price</th>
                                                        <th>
                                                                <abbr title="If this product is being
purchased as a gift, you can improve the recipient's future recommendations by selecting their name
from this field">Recipient (Optional)</abbr>
                                                                <p class="proto">In a real application,
this would be some form of search field rather than a drop-down list of every customer. There would
also be the ability to invite the recipient to become a customer at this store if they didn't already
exist in the database.</p>
                                                        </th>
                                                        <th><abbr title="This purchase will be ignored
for recommendations until this date to ensure that surprise gifts are not accidentally revealed">Hide
Until</abbr><br /><span style="font-size:small;font-style:italic;">(YYYY-MM-DD)</span></th>
                                                </tr>
                                        </thead>
                                        <tbody>
                                                <? foreach (Cart::getItems() as /** @var CartItem */
$item) { ?>
                                                        <tr>
                                                                <td><?=html($item->getProduct()-
>getName())?></td>
                                                                <td><?=html($item->getProduct()-
>getFormattedPrice())?></td>
                                                                <td><?=$item->getQuantity()?></td>
                                                                <td><?=html($item-
>getFormattedLinePrice())?></td>
                                                                <td>
                                                                        <select name="recipient[<?=$item-
>getId()?>]">
                                                                                <?
                                                                                $currentRecipientId =
$item->getRecipient();
                                                                                if ($currentRecipientId)
$currentRecipientId = $currentRecipientId->getId();
                                                                                ?>
                                                                                <option value="0">[No
Recipient]</option>
                                                                                <? foreach
(Users::getAllUsers() as $user) { ?>
                                                                                        <option
value="<?=$user->getId()?>"<?=($user->getId()==$currentRecipientId)?'
selected="selected"':''?>><?=html($user->getName())?></option>
                                                                                <? } ?>
                                                                        </select>
                                                                </td>
                                                                <td class="hideuntilfield"><input
type="text" name="hideuntil[<?=$item->getId()?>]" value="<?=($item->getHideUntil()?$item-
>getHideUntil()->display('Y-m-d'):'')?>" /> <abbr title="Enter dates in the format YYYY-MM-DD, e.g.
2009-12-31">?</abbr></td>
                                                        </tr>
                                                <? } ?>
                                        </tbody>
                                </table>
                        </div>

                        <p style="text-align:right;"><input type="submit" name="checkout"
value="Submit Order" /></p>
                </fieldset>
        </form>
<? } ?>
<? } else { ?>
```

```
        <? Utility::error('Empty Cart', 'Your cart is empty. You cannot check out an empty cart.');
?>
   <? } ?>
<? } else { ?>
   <? Utility::error('Not Logged In', 'You are not logged in. You need to be logged in to checkout.');
?>
<? } ?>
```

inc/pages/home.php

```
<h2>Home / Product List</h2>

<p>This page lists all of the products in the database, allowing you to add them to your cart, view
their current ratings and submit a rating yourself (each customer is only allowed to rate each product
once).</p>

<p class="proto">In a production system there would be product categories and only a subsection of the
full product list would be displayed on the home page. Each product would also have its own page,
displaying a full description, reviews, etc. Such features are unnecessary for a demonstration
prototype like this.</p>

<?
if (isset($_POST['rateproduct'])) {
   try {
          //Attempt to rate this product. An exception will be thrown if a DB
          // error occurs (i.e. if it's already been rated or if the
          // customer/product foreign key constraint fails)
          $rateResult = Products::rateProductById($_POST['rateproductid'], $_POST['rateproduct']);
          if ($rateResult === false) {
                 Utility::error('Not Logged In', 'You must be logged in before you can rate a
product.');
          } else {
                 Utility::success('Rating Successful', 'Thank you for rating that product.');
          }
   } catch (DBQueryFailureException $ex) {
          //A DB error occurred
          switch ($ex->getCode()) {
                 case 1062:
                        //Duplicate value on a UNIQUE/PRIMARY key
                        Utility::error('Rating Error', 'Sorry, you have already rated that product.');
                        break;
                 case 1452:
                        //Foreign key constraint failed
                        Utility::error('Rating Error', 'Either you selected an invalid rating value or
your user account is invalid.');
                        break;
                 default:
                        Utility::error('Rating Error', 'An unknown error occurred when trying to rate
that product.');
          }
   }
}
?>

<? $allProds = Products::getAllProducts(); ?>
<? if ($allProds) { ?>
   <div class="tablesur">
          <table border="1" cellspacing="0" cellpadding="3">
                 <thead>
                        <tr>
                               <th></th>
                               <th>Name</th>
                               <th>Price</th>
                               <th></th>
                        </tr>
                 </thead>
                 <tbody>
                        <?
                        foreach ($allProds as $product) { ?>
                               <tr>
                                      <td rowspan="2">
                                             <? if ($product->getImage()) { ?>
                                                    <img src="./images/<?=$product->getImage()?>.jpg"
alt="[Product Image]" />
                                             <? } ?>
```

```
                                        </td>
                                        <td rowspan="2"><?=html($product->getName())?></td>
                                        <td rowspan="2"><?=html($product->getFormattedPrice())?></td>
                                        <td>
                                                <form action="" method="post">
                                                        <fieldset style="border:0px">
                                                                <label for="qtybox_<?=$product-
>getId()?>">Quantity:</label>
                                                                        <input type="text"
id="qtybox_<?=$product->getId()?>" name="addproductqty" value="1" size="2" />
                                                                <input type="hidden" name="addproduct"
value="<?=$product->getId()?>" />
                                                                <input type="submit" value="Add to Cart" />
                                                        </fieldset>
                                                </form>
                                        </td>
                                </tr>
                                <tr>
                                        <td class="rating">
                                                <p class="current"><strong>Current Rating:</strong>
<?=$product->getFormattedRating()?></p>

                                                <? if (!$product->getHasUserRatedProduct()) { ?>
                                                        <form action="" method="post">
                                                                <fieldset style="border:0px">
                                                                        <strong>Rate This Product:</strong><br
/>
                                                                        <input type="hidden"
name="rateproductid" value="<?=$product->getId()?>" />
                                                                        <? for ($i=1;$i<=5;$i++) { ?>
                                                                        <input type="radio" name="rateproduct"
id="rate_<?=$product->getId()?>_<?=$i?>" value="<?=$i?>" /><label for="rate_<?=$product-
>getId()?>_<?=$i?>"><?=$i?></label>
                                                                        <? } ?>
                                                                        <input type="submit" value="Submit
Rating" />
                                                                </fieldset>
                                                        </form>
                                                <? } else { ?>
                                                        <p class="already">You have already rated this
product.</p>
                                                <? } ?>
                                        </td>
                                </tr>
                        <? } ?>
                </tbody>
        </table>
   </div>
<? } else { ?>
   <? Utility::error('No Products Found', 'No products were found in the database.'); ?>
<? } ?>
```

inc/pages/orders.php

```
<h2>Past Orders</h2>
<p>This is a list of orders that you have placed in the past. You can use this page to select which of your purchases are used to recommend products to you in the future. See also the <a href="?page=recommendation-products">Products Used for Recommendation</a> page, which includes items purchased for you as a gift by someone else.</p>

<p class="proto">In a production system, this page should allow the recipient and "hidden until" date to be changed.</p>

<? if ($user = Users::getCurrentUser()) { ?>
  <? $orders = Orders::getAllOrdersForCustomer($user); ?>
  <? if ($orders) { ?>
      <? foreach ($orders as $k => /** @var Order */ $order) { ?>
          <h3 class="ordertitle">Order #<?=$order->getId()?> placed at <?=html($order->getDate()->display())?>:</h3>
          <div class="tablesur">
              <table border="1" cellspacing="0" cellpadding="3">
                  <thead>
                      <tr>
                          <th>Product Name</th>
                          <th>Quantity</th>
                          <th>Recipient</th>
                          <th>Hidden From Recipient Until</th>
                          <th>Used for Your Recommendations?</th>
                      </tr>
                  </thead>
                  <tbody>
                      <? foreach ($order->getOrderedProducts() as /** @var OrderedProduct */ $op) { ?>
                          <?
                          if (isset($_POST['togglepurchaseruse'][$order->getId()][$op->getProduct()->getId()])) {
                              //The user's requested to toggle the purchaseruse flag
                              $op->setPurchaserUse(!$op->getPurchaserUse());
                          }
                          ?>
                          <tr>
                              <td><?=$op->getProduct()->getName()?></td>
                              <td><?=$op->getQuantity()?></td>
                              <td><?=($op->getRecipient() ? $op->getRecipient()->getName() : '<em>None</em>')?></td>
                              <td><?=($op->getRecipient() ? ($op->getHideUntil() ? $op->getHideUntil()->display() : '<em>Not Hidden</em>') : '<em>N/A</em>')?></td>
                              <td>
                                  <form action="" method="post">
                                      <fieldset>
                                          <? if ($op->getPurchaserUse()) { ?>
                                              Yes
                                              <input type="submit" name="togglepurchaseruse[<?=$order->getId()?>][<?=$op->getProduct()->getId()?>]" value="Stop Using" />
                                          <? } else { ?>
                                              No
                                              <input type="submit" name="togglepurchaseruse[<?=$order->getId()?>][<?=$op->getProduct()->getId()?>]" value="Start Using" />
                                          <? } ?>
                                      </fieldset>
                                  </form>
                              </td>
                          </tr>
                      <? } ?>
                  </tbody>
              </table>
```

```
                </div>
                <? if ($k < count($orders)-1) { ?>
                        <hr />
                <? } ?>
        <? } ?>
   <? } else { ?>
        <p>You have not yet placed any orders.</p>
   <? } ?>
<? } else { ?>
   <? Utility::error('Not Logged In', 'You are not logged in. You need to be logged in to view your
past orders.'); ?>
<? } ?>
```

inc/pages/recommendation-proudcts.php

```
<h2>Recommendation Products</h2>
<p>This is a list of products that you have bought and not marked as gifts for someone else, products that others have bought as gifts for you, and products that you have rated.</p>
<p>Here you can choose which products are used to make product recommendations for you in the future. Products that aren't used for recommendations are ignored by the recommendation algorithm.</p>

<? if ($user = Users::getCurrentUser()) { ?>
  <? $prods = Products::getProductsAvailableForRecommendation($user); ?>
  <? if ($prods) { ?>
        <div class="tablesur">
              <table border="1" cellspacing="0" cellpadding="3">
                    <thead>
                          <tr>
                                <th></th>
                                <th>Product Name</th>
                                <th>Purchased By</th>
                                <th>Purchase Date</th>
                                <th>Rating</th>
                                <th>Used for Your Recommendations?</th>
                          </tr>
                    </thead>
                    <tbody>
                          <? foreach ($prods as /** @var OrderedProduct */ $op) { ?>
                                <?
                                //Each element in this array can either be an OrderedProduct or a RatedProduct
                                if ($op instanceof OrderedProduct) {
                                      $order = Order::createFromId($op->getOrderId());
                                      $purchaser = $op->getPurchaser();
                                      $recipient = $op->getRecipient();

                                      if (isset($_POST['toggleuse'][$op->getOrderId()][$op->getProduct()->getId()])) {
                                            if ($purchaser->equals($user)) {
                                                  $op->setPurchaserUse(!$op->getPurchaserUse());
                                            } elseif ($recipient && $recipient->equals($user)) {
                                                  $op->setRecipientUse(!$op->getRecipientUse());
                                            }
                                      }
                                } elseif ($op instanceof RatedProduct) {
                                      if (isset($_POST['toggleurateduse'][$op->getProduct()->getId()])) {
                                            $op->toggleUse();
                                      }
                                }
                                ?>
                                <tr>
                                      <td>
                                            <? if ($op->getProduct()->getImage()) { ?>
                                                  <img src="./images/<?=$op->getProduct()->getImage()?>.jpg" alt="[Product Image]" />
                                            <? } ?>
                                      </td>
                                      <td><?=$op->getProduct()->getName()?></td>
                                      <td>
                                            <? if ($op instanceof OrderedProduct) { ?>
                                                  <? if ($purchaser->equals($user)) { ?>
                                                        You
                                                  <? } else { ?>
                                                        <?=html($purchaser->getName())?>
                                                  <? } ?>
                                            <? } else { ?>
                                                  <em>N/A</em>
                                            <? } ?>
                                      </td>
```

```
                                        <td>
                                                <? if ($op instanceof OrderedProduct) { ?>
                                                        <?=html($order->getDate()->display())?>
                                                <? } else { ?>
                                                        <em>N/A</em>
                                                <? } ?>
                                        </td>
                                        <td>
                                                <? if ($op instanceof RatedProduct) { ?>
                                                        <?=html($op->getRating())?>
                                                <? } else { ?>
                                                        <em>N/A</em>
                                                <? } ?>
                                        </td>
                                        <td>
                                                <form action="" method="post">
                                                        <fieldset>
                                                                <? if ($op instanceof OrderedProduct) {
?>
                                                                        <? if (($op->getPurchaserUse() &&
$purchaser->equals($user)) || ($recipient && $op->getRecipientUse() && $recipient->equals($user))) {
?>
                                                                                Yes
                                                                                <input type="submit"
name="toggleuse[<?=$op->getOrderId()?>][<?=$op->getProduct()->getId()?>]" value="Stop Using" />
                                                                        <? } else { ?>
                                                                                No
                                                                                <input type="submit"
name="toggleuse[<?=$op->getOrderId()?>][<?=$op->getProduct()->getId()?>]" value="Start Using" />
                                                                        <? } ?>
                                                                <? } else { ?>
                                                                        <? if ($op-
>isUsedForRecommendations()) { ?>
                                                                                Yes
                                                                                <input type="submit"
name="toggleprateduse[<?=$op->getProduct()->getId()?>]" value="Stop Using" />
                                                                        <? } else { ?>
                                                                                No
                                                                                <input type="submit"
name="toggleprateduse[<?=$op->getProduct()->getId()?>]" value="Start Using" />
                                                                        <? } ?>
                                                                <? } ?>
                                                        </fieldset>
                                                </form>
                                        </td>
                                </tr>
                        <? } ?>
                </tbody>
        </table>
    </div>
  <? } else { ?>
    <p>You currently don't have any products used for recommendation.</p>
  <? } ?>
<? } else { ?>
  <? Utility::error('Not Logged In', 'You are not logged in. You need to be logged in to view the
list of products used for recommendation.'); ?>
<? } ?>
```

inc/pages/recommendations.php

```
<h2>Recommendations</h2>
<p>This page will recommend products to you based upon the products that you have bought in the past and those that others have bought you. Additionally it will allow you to see a list of recommended products for another customer, for the purposes of gift-purchasing.</p>

<h3 class="recommendationtitle">Your Recommendations</h3>
<? if ($user = Users::getCurrentUser()) { ?>
  <? if ($recommendations = Products::getRecommendedProducts($user)) { ?>
        <div class="tablesur">
              <table border="1" cellspacing="0" cellpadding="3">
                    <thead>
                          <tr>
                                <th></th>
                                <th>Name</th>
                                <th>Price</th>
                                      <th>Similarity</th>
                                <th></th>
                          </tr>
                    </thead>
                    <tbody>
                          <? foreach ($recommendations as /** @var SimilarProduct */ $product) { ?>
                                <tr>
                                      <td rowspan="2">
                                            <? if ($product->getImage()) { ?>
                                                  <img src="./images/<?=$product->getImage()?>.jpg" alt="[Product Image]" />
                                            <? } ?>
                                      </td>
                                      <td rowspan="2"><?=html($product->getName())?></td>
                                      <td rowspan="2"><?=html($product->getFormattedPrice())?></td>
                                      <td rowspan="2"><?=$product->getSimilarity(2)?></td>
                                      <td>
                                            <form action="" method="post">
                                                  <fieldset style="border:0px">
                                                        <input type="text" name="addproductqty" value="1" size="2" />
                                                        <input type="hidden" name="addproduct" value="<?=$product->getId()?>" />
                                                        <input type="submit" value="Add to Cart" />
                                                  </fieldset>
                                            </form>
                                      </td>
                                </tr>
                                <tr>
                                      <td class="rating">
                                            <p class="current"><strong>Current Rating:</strong> <?=$product->getFormattedRating()?></p>
                                      </td>
                                </tr>
                          <? } ?>
                    </tbody>
              </table>
        </div>
  <? } else { ?>
        <? Utility::error('No Recommendations Found', 'No recommendations were found. This can be caused by you not having any products available to determine recommendations from (e.g. not having made any purchases) or by you having already bought - or been bought - every suitable recommendation. It can also happen if the `productsimilarity` table hasn\'t been populated.'); ?>
  <? } ?>
<? } else { ?>
  <? Utility::error('Not Logged In', 'You are not logged in, so you cannot view your recommendations.'); ?>
<? } ?>
```

```
<h3 class="recommendationtitle">Recommend a Gift</h3>
<? if (isset($_GET['recipient'])) { ?>
   <? if ($recipient = Users::getUserById($_GET['recipient'])) { ?>
         <h4 class="recommendationsubtitle">Gift Recommendations for <?=html($recipient-
>getName())?></h4>
         <? if ($recommendations = Products::getRecommendedProducts($recipient)) { ?>
               <div class="tablesur">
                     <table border="1" cellspacing="0" cellpadding="3">
                           <thead>
                                 <tr>
                                       <th></th>
                                       <th>Name</th>
                                       <th>Price</th>
                                       <th>Similarity</th>
                                       <th></th>
                                 </tr>
                           </thead>
                           <tbody>
                                 <? foreach ($recommendations as /** @var SimilarProduct */
$product) { ?>
                                       <tr>
                                             <td rowspan="2">
                                                   <? if ($product->getImage()) { ?>
                                                         <img src="./images/<?=$product-
>getImage()?>.jpg" alt="[Product Image]" />
                                                   <? } ?>
                                             </td>
                                             <td rowspan="2"><?=html($product->getName())?></td>
                                             <td rowspan="2"><?=html($product-
>getFormattedPrice())?></td>
                                             <td rowspan="2"><?=html($product-
>getSimilarity(2))?></td>
                                             <td>
                                                   <form action="" method="post">
                                                         <fieldset style="border:0px">
                                                               <input type="text"
name="addproductqty" value="1" size="2" />
                                                               <input type="hidden"
name="addproduct" value="<?=$product->getId()?>" />
                                                               <input type="submit" value="Add
to Cart" />
                                                         </fieldset>
                                                   </form>
                                             </td>
                                       </tr>
                                       <tr>
                                             <td class="rating">
                                                   <p class="current"><strong>Current
Rating:</strong> <?=$product->getFormattedRating()?></p>
                                             </td>
                                       </tr>
                                 <? } ?>
                           </tbody>
                     </table>
               </div>
         <? } else { ?>
               <? Utility::error('No Recommendations Found', 'No recommendations were found. This can
be caused by the selected recipient not having any products available to determine recommendations
from (e.g. not having made any purchases) or by them having already bought - or been bought - every
suitable recommendation. It can also happen if the `productsimilarity` table hasn\'t been
populated.'); ?>
         <? } ?>
         <h4 class="recommendationsubtitle">Find Recommendations for Someone Else</h4>
   <? } else { ?>
         <? Utility::error('Invalid Customer', 'The customer you selected appears to be invalid.'); ?>
   <? } ?>
```

```
<? } ?>
<form action="" method="get">
   <fieldset class="giftrecommendationfieldset">
          <input type="hidden" name="page" value="recommendations" />
          <p class="giftrecommendationdesc">Looking to buy a gift for someone, but don't know what to
buy? Select the recipient below to see what we would recommend!</p>
          <label for="recipientname">Recipient:</label>
                <select id="recipientname" name="recipient">
                      <? $curUser = Users::getCurrentUser(); ?>
                      <? foreach (Users::getAllUsers() as $user) { ?>
                            <? if (!$curUser || !$user->equals($curUser)) { ?>
                                  <option value="<?=$user->getId()?>"><?=html($user-
>getName())?></option>
                            <? } ?>
                      <? } ?>
                </select>
          <input type="submit" value="See Recommendations" />
   </fieldset>
</form>

<p class="proto">Again, this would be a search form, rather than a drop-down list of customers, in a
real system.</p>
```

style/main.css

```
div.phperror {
   overflow: auto;
   padding: 0px 10px 5px 10px;
   font-size: 0.8em;
   border: 1px solid #f00;
   background-color: #f88;
}

.error, .success {
   padding: 10px;
   margin: 0px 0px 10px 0px;
   font-size: 0.9em;
}
.error {
   border: 1px solid #c00004;
   background-color: #ffa9ab;
}
.success {
   border: 1px solid #06c000;
   background-color: #aaffa7;
}
.error h1, .success h1 {
   font-size: 1.3em;
   font-weight: bold;
   margin: 0px 0px 5px 0px;
}

.proto {
   font-style: italic;
   color: #555;
   font-size: 8pt;
   font-family: sans-serif;
   border: 1px dashed #555;
   background-color: #ddd;
   padding: 3px 5px 3px 5px;
   margin: 3px;
}


body {
   margin: 0px;
   padding: 20px;
   color: #333333;
   background-image: url(images/bg.png);
   background-repeat: repeat-x;
}
hr {
   margin: 20px 0px 10px 0px;
   border-style: solid;
   border-color: #333333;
   border-width: 0px 1px;
   background-image: url(images/hr.png);
}

div#header {
   background-color: #3F8FD2;
   height: 80px;
   padding: 40px 40px 0px 40px;
   font-family: sans-serif;
}
div#header h1 {
   margin: 0px;
}
div#header h1 a:link, div#header h1 a:visited, div#header h1 a:active, div#header h1 a:hover {
   color: #fff;
```

```
   text-decoration: none;
}
div#sidebar {
   width: 200px;
   float: right;
   background-color: #66A1D2;
   padding: 15px 10px;
   font-size: 0.8em;
   font-family: sans-serif;
}
div#sidebar h2 {
   margin: 0px 0px 0px 0px;
}
div#sidebar ul.nav {
    padding-left: 17px;
    /*list-style-image: url(images/bullet.png);*/
    list-style-type: circle;
}
div#sidebar ul.nav a:link, div#sidebar ul.nav a:active, div#sidebar ul.nav a:visited {
   color: #333333;
   text-decoration: none;
}
div#sidebar ul.nav a:hover {
   text-decoration: underline;
}
div#sidebar ul {
   margin: 5px 0px 0px 0px;
}
div#sidebar fieldset {
   border: 0px none;
}
div#sidebar p.carttotalprice {
   font-weight: bold;
   margin: 10px 0px 0px 0px;
}

div#mainbody {
   padding: 0px 240px 20px 20px;
}

div#mainbody table {
   width: 90%;
   border: 1px solid #333333;
   border-collapse: collapse;
   margin: 10px auto 0px auto;
}
div#mainbody table th, div#mainbody table td {
   border: 1px solid #333333;
}
div#mainbody table th {
   font-weight: bold;
   background-color: #66A1D2;
   color: #fff;
}
div#mainbody div.tablesur {
   text-align: center;
}

td fieldset {
   padding: 0px;
}

td.rating {
   padding-bottom: 5px;
}
td.rating p.current {
   margin: 3px 5px 5px 5px;
```

```
}
td.rating p.already {
   margin: 0px;
}

div#mainbody a:link, div#mainbody a:visited, div#mainbody a:active {
   color: #3F8FD2;
   text-decoration: underline;
}
div#mainbody a:hover {
   text-decoration: none;
}

h3.ordertitle {
   margin: 15px 0px 0px 0px;
}

h3.recommendationtitle {
   margin: 15px 0px 0px 0px;
}
fieldset.giftrecommendationfieldset {
   border: 0px none;
   padding: 5px 0px 12px 0px;
}
fieldset.giftrecommendationfieldset p.giftrecommendationdesc {
   margin: 0px 0px 10px 0px;
}

h4.recommendationsubtitle {
   margin-bottom: 0px;
}

fieldset.checkoutform {
   border: 0px;
}

td.hideuntilfield {
   white-space: nowrap;
}
```

index.php

```
<?php

//Use gzipped output buffering to both reduce the size of our generated pages (gzip)
// and to allow us to send HTTP headers even after we've started outputting data (output buffering)
ob_start('ob_gzhandler');

//Two small wrapper functions for some rather commonly-used, but long-named functions
function html($s){return htmlspecialchars($s,ENT_NOQUOTES,'UTF-8');}
function htmlq($s){return htmlspecialchars($s,ENT_QUOTES,'UTF-8');}

//Include our error handling class and set up PHP to use it for handling errors
require 'classes/errorhandler.php';
set_error_handler(array('ErrorHandler', 'handleError'));

//Include the rest of our classes
require 'classes/cart.php';
require 'classes/date.php';
require 'classes/db.php';
require 'classes/exceptions.php';
require 'classes/formatting.php';
require 'classes/orders.php';
require 'classes/products.php';
require 'classes/users.php';
require 'classes/utility.php';
require 'classes/validation.php';

//Set up the database connection for use
DB::setup('localhost', 'level3project', 'lvl3password', 'lvl3');

//Set the session lifetime and start the session
ini_set('session.gc_maxlifetime', 21600);
session_start();

//And specify that we're using UTF-8 as our character set
header('Content-Type: text/html; charset=utf-8');

?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Demo Store</title>
   <link href="style/main.css" rel="stylesheet" rev="stylesheet" type="text/css" />
</head>
<body>

   <div id="header">
         <h1><a href="./">Demo Store</a></h1>
   </div>

   <div id="sidebar">
         <? include 'inc/blocks/navigation.php'; ?>
         <hr />
         <? include 'inc/blocks/login.php'; ?>
         <hr />
         <? include 'inc/blocks/cart.php'; ?>
   </div>

   <div id="mainbody">
         <?
         //Include our main page depending on which one was requested
         switch (Utility::getGET('page', 'home')) {
               case 'checkout':
                     include 'inc/pages/checkout.php';
                     break;
```

```
                case 'orders':
                        include 'inc/pages/orders.php';
                        break;
                case 'recommendation-products':
                        include 'inc/pages/recommendation-products.php';
                        break;
                case 'recommendations':
                        include 'inc/pages/recommendations.php';
                        break;
                case 'home':
                        include 'inc/pages/home.php';
                        break;
                case 'calculate-similarities':
                        include 'inc/pages/calculate-similarities.php';
                        break;
                default:
                        include 'inc/pages/404.php';
        }
        ?>
    </div>

</body>
</html>
```