BD Project - Part 3

Work done by:
Pedro Gonçalves nº 87559
Alexandre Bento nº 96723
Tiago Delgado nº 96770

Relative percentage of each student's contribution:
Pedro Gonçalves - 33.3%
Alexandre Bento - 33.3%
Tiago Delgado - 33.3%

Total effort (in hours):
Pedro Gonçalves - 30h
Alexandre Bento - 30h
Tiago Delgado - 30h

Group: 5
Shift: Friday - 10h00
Teacher: Paulo Carreira

Application Architecture

URL:http://web.tecnico.ulisboa.pt/alexandre.e.bento/

The "front-end" components, manage categories and manage products and suppliers, are generated with the following flow:

Connect to database → Run main function →
detect if a form was submitted and which one →
redirect control flow to process submission --> print the result
→ render the main page (always happens regardless if there was an submission or not) --> get all form fields and print them
These last two steps compound nicely to give the illusion that the page is not being constantly refreshed.

If any exception occurs when processing a form it will flow to a try catch in the main function. This try catch will rollback any changes and make the error friendly to the user by translating the raw messages from psycopg and filtering internal errors in order to not expose too much information.

There is a python and scripts file that contains the html/JS used making the integration of python with the web stack more seamless.

Additional features:

- Each time an action is successfully performed an “random fact” is shown just to change the UI and confirm that something happened.
- When adding or removing suppliers the page will automatically scroll to the supplier section.

In question C, we initially show all the products available at the supermarket and when we click on the show replenishment button we are taken to the replenishments of that product.
In question D, we initially show all the products available at the supermarket and when we click on the change description button we are taken to a web page that allows us to insert the new product description.

In question E two files were used to list all the sub-categories of a super-category, at all levels of depth. pergunta_e.cgi displays a table with all the super categories present in the database. This file also links each super categories to its respective sub categories using the list_sub_aux.cgi. Since the list_sub_aux.cgi always calls itself when checking the sub categories of a category it allows the user to list all the sub-categories of a super-category, at all levels of depth.

INDEXES

5.1-
CREATE INDEX IDX_products_ean ON product USING BTREE(ean) INCLUDE (category);

Assuming the primary suppliers table is smaller than the product table there is no need for an index, but in the end we decided to create one. Here's our thought process:

The primary suppliers table, since it's smaller, will be loaded to memory and read sequentially while at the same time matching each row's ean to all rows in the product table that also have the same ean. Since the column being matched is the primary key of products, the qualified rows will always be at most one.
Hence, from these obtained rows there is no need to have an index to find the ones that match the second condition " category = 'vegetables' " . In other words, there is no need for a composite index with ean and category.
Another benefit of searching for the primary key(ean) is that there is already a BTREE index, making at first sight unnecessary to create any index.

Despite having the database already set up to find which rows apply to the query very efficiently there is another problem: retrieving them.
Since the tables are incredibly large the most likely thing to happen is that the wanted rows are going to be scattered through the filesystem, requiring many IO operations.
To overcome this problem we propose the creation of an index that already includes all the remaining information necessary to perform the query (the categories column).

5.2 -
CREATE INDEX IDX_secondary_supplier_ean ON supplies_sec USING HASH(ean)

Given that a product can have multiple secondary suppliers is reasonable to assume that the table supplies_sec is larger than the product table.

Taking this into consideration, the product table will be scanned sequentially and it will be appropriate to have an index for secondary_supplier table to assist the resolution of the WHERE clause. Given that the condition in the WHERE is a point equality, the best index will be a HASH on the attribute ean of secondary_supplier table.

To improve the group by performance an BTREE index can be used. Since the grouping is being done by ean, that is the primary key of product, there is already a BTREE index and therefore no need for another index.