TÉCNICO LISBOA

Projeto BD - Parte 3

Nome	Número	Percentagem de esforço (%)	Horas totais de esforço (H)
Bernardo Sousa	103191	33.(3)	25
Miguel Sol	102710	33.(3)	25
Mariana Carvalho	102956	33.(3)	25

Grupo: 31
Turno: BD2L02
Professor: João Caldeira

Explicação Arquitetura

A aplicação começa com uma página Home onde podemos escolher dar login como Manager ou Customer, nas quais encontramos uma sidebar com as principais categorias de ação.
No manager podemos escolher editar os Products, Suppliers ou Customers. Em cada uma destas categorias temos dois botões onde podemos adicionar ou consultar a mesma. Na página de adicionar temos os respectivos campos necessários para a adição, complementado por um backbutton. Na página de consultar, dependendo se a categoria permite, temos possibilidade de editar e remover um elemento da mesma. Esta levará a uma página de confirmação, no caso da remoção, ou a uma página de edição, outra vez com dois botões para as duas possibilidades de edição seguidas por as respectivas caixas de input e botões para submissão.
Ao clicarmos no botão da sidebar a dizer Home, podemos voltar a Home page e dar login como Customer, onde encontraremos uma dropdown bar junto com uma searchbar para o cliente colocar o seu email e entrar na sua area de cliente. Aqui encontramos a sidebar com a opção Home e Orders, a última que contém dois botões, um para adicionar uma encomenda e outro para consultar as encomendas do Customer. Na página para adicionar encomendas encontramos uma tabela com os produtos, uma searchbar, a quantidade a inserir na encomenda e uma checkbox para adicionar o produto. Ao colocarmos as quantidades e selecionar as respectivas checkboxes podemos clicar no botão submit para criar a encomenda. Na parte da consulta temos uma lista das orders do cliente junto com um link onde, caso seja possível, o cliente pode pagar a order em questão.

Link web

https://web2.tecnico.ulisboa.pt/ist1102710/Home.html

Indices

Na primeira query acrescentamos um indice composto na tabela product com os atributos (sku, price). Este índice facilita a satisfazer a procura dos skus cujo price é maior que 50 fazendo apenas um index scan para dar join de product com contains e order, visto estarem organizado em B+TREE, pelo que, os sku's dos produtos estão ordenados junto com o price. Contudo este índice só funciona se houver poucos produtos cujo price seja maior que 50.

Sem índice

```
 Nested Loop  (cost=252.32..480.94 rows=14 width=4)
   ->  Hash Join  (cost=252.05..468.40 rows=42 width=72)
         Hash Cond: (contains.order_no = "order".order_no)
         ->  Seq Scan on contains  (cost=0.00..194.02 rows=8502 width=72)
         ->  Hash  (cost=251.35..251.35 rows=56 width=4)
               ->  Seq Scan on "order"  (cost=0.00..251.35 rows=56 width=4)
                     Filter: (date_part('year'::text, (date)::timestamp without time
zone) = '2023'::double precision)
   ->  Index Scan using product_pkey on product  (cost=0.27..0.30 rows=1
width=68)
         Index Cond: ((sku)::text = (contains.sku)::text)
         Filter: (price > '50'::numeric)
(10 rows)
```

Com índice composto (sku, price)

```
 Nested Loop  (cost=252.33..481.14 rows=14 width=4)
   ->  Hash Join  (cost=252.05..468.40 rows=42 width=72)
         Hash Cond: (contains.order_no = "order".order_no)
         ->  Seq Scan on contains  (cost=0.00..194.02 rows=8502 width=72)
         ->  Hash  (cost=251.35..251.35 rows=56 width=4)
               ->  Seq Scan on "order"  (cost=0.00..251.35 rows=56 width=4)
                     Filter: (date_part('year'::text, (date)::timestamp without time
zone) = '2023'::double precision)
   ->  Index Only Scan using price_and_sku_index_product on product
(cost=0.28..0.30 rows=1 width=68)
         Index Cond: ((sku = (contains.sku)::text) AND (price > '50'::numeric))
(9 rows)
```

Adicionámos também um índice na tabela “order” com o atributo HASH(EXTRACT(YEAR FROM date)), pois este já deixa os valores que a query precisa pré-calculados e se existirem poucas orders com o ano 2023 ainda melhor visto que o join de order com contains será ainda mais rápido pois poderá realizar um index only scan (que neste caso é um bitmap index scan).

Com índice HASH(EXTRACT(YEAR FROM date))

```
 Nested Loop  (cost=62.40..291.52 rows=14 width=4)
  ->  Hash Join  (cost=62.13..278.48 rows=43 width=72)
      Hash Cond: (contains.order_no = "order".order_no)
      ->  Seq Scan on contains  (cost=0.00..194.02 rows=8502 width=72)
      ->  Hash  (cost=61.50..61.50 rows=50 width=4)
          ->  Bitmap Heap Scan on "order"  (cost=4.39..61.50 rows=50
width=4)
              Recheck Cond: (date_part('year'::text, (date)::timestamp without
time zone) = '2023'::double precision)
              ->  Bitmap Index Scan on date_hash_index_order
(cost=0.00..4.38 rows=50 width=0)
                  Index Cond: (date_part('year'::text, (date)::timestamp
without time zone) = '2023'::double precision)
  ->  Index Scan using product_pkey on product  (cost=0.28..0.30 rows=1
width=68)
      Index Cond: ((sku)::text = (contains.sku)::text)
      Filter: (price > '50'::numeric)
(12 rows)
```

Os dois índices juntos

```
 Nested Loop  (cost=62.40..291.52 rows=14 width=4)
  ->  Hash Join  (cost=62.13..278.48 rows=43 width=72)
      Hash Cond: (contains.order_no = "order".order_no)
      ->  Seq Scan on contains  (cost=0.00..194.02 rows=8502 width=72)
      ->  Hash  (cost=61.50..61.50 rows=50 width=4)
          ->  Bitmap Heap Scan on "order"  (cost=4.39..61.50 rows=50
width=4)
              Recheck Cond: (date_part('year'::text, (date)::timestamp without
time zone) = '2023'::double precision)
              ->  Bitmap Index Scan on date_hash_index_order
(cost=0.00..4.38 rows=50 width=0)
```

```
                Index Cond: (date_part('year'::text, (date)::timestamp
without time zone) = '2023'::double precision)
  ->  Index Only Scan using price_and_sku_index_product on product
(cost=0.28..0.30 rows=1 width=68)
      Index Cond: ((sku = (contains.sku)::text) AND (price > '50'::numeric))
(11 rows)
```

Os dois índices juntos notamos numa redução de 50% a 30% do tempo inicial

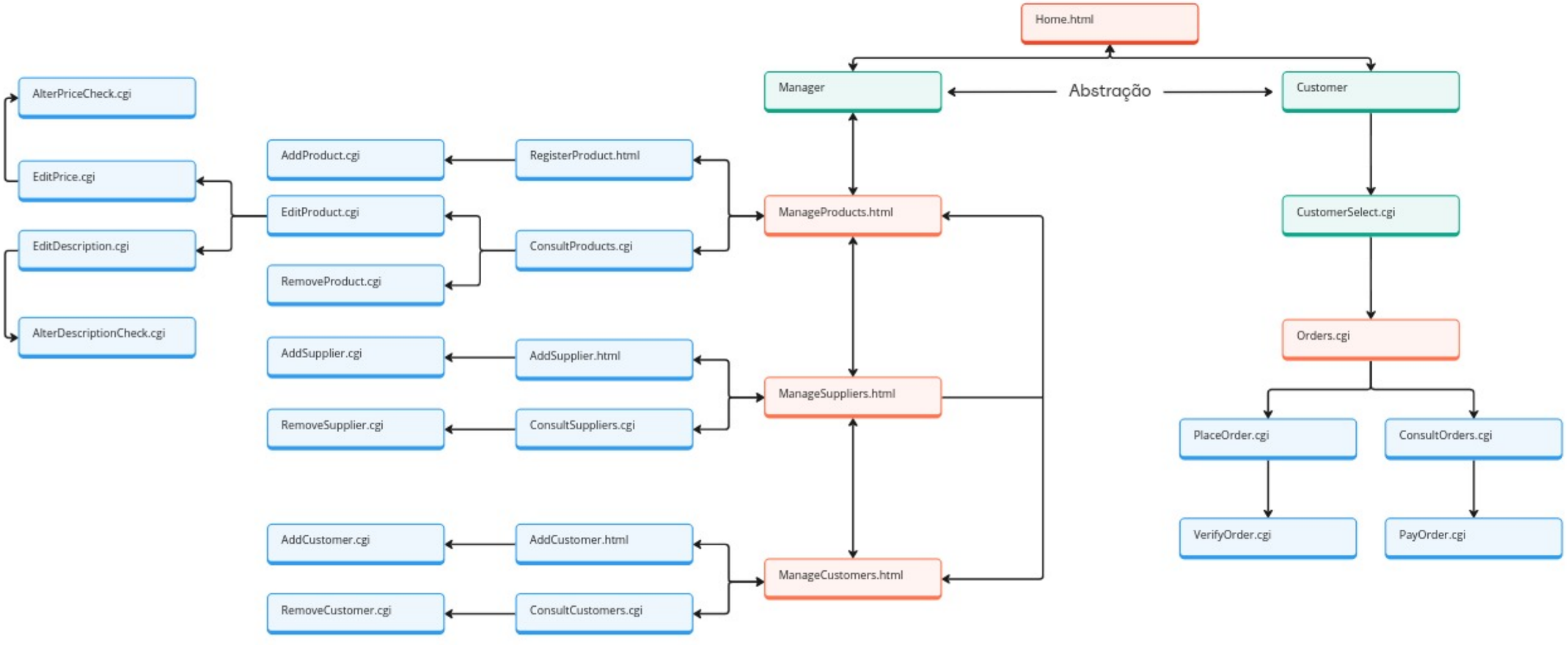
Na segunda query colocámos um índice organizado em B+TREE na tabela contains no atributo sku para que fosse mais fácil fazer o join com product visto este já ter um índice composto com os atributos order_no e sku, contudo a tabela contains nunca está organizada de forma a favorecer o join pelo sku. Desta forma o algoritmo permitiu uma procura mais eficiente, realizando um Bitmap Heap Scan.

Com índice na tabela contains no atributo sku

```
HashAggregate  (cost=353.81..355.06 rows=100 width=36)
  Group Key: contains.order_no
  ->  Nested Loop  (cost=5.06..352.81 rows=100 width=24)
      ->  Seq Scan on product  (cost=0.00..24.50 rows=5 width=84)
          Filter: ((name)::text ~~ 'A%'::text)
      ->  Bitmap Heap Scan on contains  (cost=5.06..64.66 rows=100
width=76)
          Recheck Cond: ((sku)::text = (product.sku)::text)
          ->  Bitmap Index Scan on order_no_index_contains
(cost=0.00..5.04 rows=100 width=0)
              Index Cond: ((sku)::text = (product.sku)::text)
(9 rows)
```

Verificamos uma redução em cerca de 30% do tempo inicial.

Relações entre ficheiros


Home.html
Manager
Abstração
Customer
AlterPriceCheck.cgi
EditPrice.cgi
EditDescription.cgi
AlterDescriptionCheck.cgi
AddProduct.cgi
EditProduct.cgi
RemoveProduct.cgi
RegisterProduct.html
ConsultProducts.cgi
ManageProducts.html
CustomerSelect.cgi
Orders.cgi
AddSupplier.cgi
RemoveSupplier.cgi
AddSupplier.html
ConsultSuppliers.cgi
ManageSuppliers.html
PlaceOrder.cgi
ConsultOrders.cgi
VerifyOrder.cgi
PayOrder.cgi
AddCustomer.cgi
RemoveCustomer.cgi
AddCustomer.html
ConsultCustomers.cgi
ManageCustomers.html