

Напредни бази на податоци  
Фаза 4- Индекси и оптимизација на прашалници  
Проект: Restaurant Management System

Александар Ангелевски	231219
Филип Хаџиевски	231034
Андреј Христовски	231162

**View1: Full Order Overview.**

1. Примарен филтер за погледот `vw_order_overview` ќе биде според примарниот клуч на келнерот, статус на нарачка (активна/завршена) а ќе се користи и временско пребарување (нарачки по датум).
2. Примарен случај на употреба ќе биде следење на тековните нарачки во реално време. За овој поглед важни се перформансите бидејќи се повикува често при секојдневното работење.

```
SELECT * FROM vw_order_overview  
WHERE DateCreated > NOW() - INTERVAL '300 days';
```

```
-> Seq Scan on "Order" o  
(cost=0.00..64009.00 rows=1555977 width=36) (actual  
time=0.010..925.627 rows=1557960 loops=1)
```

Execution Time: 2707.239 ms

77% од редиците опфатени нема потреба од индексирање.

3.

```
SELECT * FROM vw_order_overview WHERE OrderId = 111 0.303s
```

Нема потреба од оптимизација. Единствен full scan се прави на табелата Table и таа има само 150 редици.

```
INSERT INTO "Order" (WaiterId, Typeld, StatusId, TableNumber)¶ 0.199s
```

Insert нема потреба од оптимизација бидејќи веќе е доволно брза операција, додека пак update:

```
UPDATE "Order" SET StatusId = 2 WHERE WaiterId = 1 7.284s
```

Можеме да ставиме индекс на

```
CREATE INDEX idx_order_waiterid ON "Order" (WaiterId);
```

Text	Duration
UPDATE "Order" SET StatusId = 2 WHERE WaiterId = 1¶	0.954s
¶INSERT INTO "Order" (WaiterId, T SQL statement text/description r)	1.727s

Ова е значителна и прифатлива промена, но мора да имаме на ум дека индексот idx\_order\_waiterid го подобрува времето на SELECT прашалниците, меѓутоа UPDATE операциите стануваат побавни поради одржување на индексната структура. Ова е стандарден read/write trade-off кај индексирање.

Text	Duration
select * from vw_order_overview WHERE WaiterId = 4	0.095s

## View2: Order Items with Product and Price Info.

1. Примарен филтер за погледот vw\_order\_items\_details ќе биде според OrderId и Finished колоните.
2. Примарен случај на употреба ќе биде Користењето во кујната за следење на незавршени ставки, и на касата при приказ на сметка. Перформансите се критични бидејќи овој поглед се користи во реално време при секоја нарачка.
3. Пред индексирање време на извршување:

```
--2ro view¶SELECT * FROM vw_order_items_detail WHERE Order 0.031s  
--2ro view¶SELECT * FROM vw_order_items_detail WHERE Order 0.505s
```

Ова време е прифатливо нема потреба од индекси

4. За insert

```
INSERT INTO OrderItem (Quantity, OrderId, CreatedBy, MenuMem 0.056s
```

Нема потреба од оптимизација

За Update

```
UPDATE OrderItem SET Finished = TRUE WHERE OrderId = 1 0.029s
```

### View3: Invoice Summary.

1. Примарен филтер за погледот vw\_invoice\_summary ќе биде според InvoiceDate и WaiterId колоните.
2. Примарен случај на употреба ќе биде генерирање на финансиски извештаи и преглед на приходи. Перформансите се важни при периодични извештаи на крај на смена или ден.

```
Nested Loop Left Join (cost=1000.57..23232.32 rows=1 width=92) (actual time=0.684..73.804 rows=1 loops=1)
  Buffers: shared hit=14157
  -> Nested Loop (cost=1000.43..23224.07 rows=1 width=32) (actual time=0.671..73.790 rows=1 loops=1)
    Buffers: shared hit=14155
    -> Gather (cost=1000.00..23215.62 rows=1 width=28) (actual time=0.641..73.754 rows=1 loops=1)
      Workers Planned: 3
      Workers Launched: 3
      Buffers: shared hit=14151
      -> Parallel Seq Scan on invoice i (cost=0.00..22215.52 rows=1 width=28) (actual time=43.250..59.658)
        Filter: (orderid = 1)
        Rows Removed by Filter: 500000
        Buffers: shared hit=14151
        -> Index Scan using "Order_pkey" on "Order" o (cost=0.43..8.45 rows=1 width=8) (actual time=0.028..0.030)
          Index Cond: (id = 1)
          Buffers: shared hit=4
        -> Index Scan using employee_pkey on employee e (cost=0.14..8.16 rows=1 width=16) (actual time=0.005..0.006)
          Index Cond: (id = o.waiterid)
          Buffers: shared hit=2
```

3.

Пред индексирање време на извршување:

```
SELECT * FROM vw_invoice_summary WHERE Invoiceld = 1 0.136s
```

```
select * from vw_invoice_summary where orderid=400 3.696s
```

4. Време за извршување без индекси

```
UPDATE Invoice SET TotalAmount = 1600.00 WHERE OrderId = 1 0.251s
INSERT INTO Invoice (OrderId, TotalAmount, TaxAmount) VALUES (1, 1600.00, 0.00) 0.338s
```

## 5. После индексирање времето на извршување

```
select * from vw_invoice_summary¶where orderid=400 0.25s
```

```
UPDATE Invoice SET TotalAmount = 1600.00 WHERE OrderId = 57 0.223s
```

```
INSERT INTO Invoice (OrderId, TotalAmount, TaxAmount)¶VALUES 0.031s
```

### View4: Active Menu with Items and Prices.

1. Примарен филтер по Active=True, секундарен е по MenuType
2. Се прикажува активното мени на клиентите и персоналот. Се вчитува честопати.

1	Nested Loop (cost=8.77..17.13 rows=1 width=231)
2	-> Nested Loop (cost=8.62..16.41 rows=1 width=177)
3	Join Filter: (mm.menuitemid = mi.id)
4	-> Nested Loop (cost=8.47..16.13 rows=1 width=185)
5	-> Hash Join (cost=8.33..15.83 rows=1 width=171)
6	Hash Cond: (m.typeid = mt.id)
7	-> Nested Loop (cost=0.15..7.53 rows=44 width=97)
8	-> Seq Scan on menumember mm (cost=0.00..1.88 rows=88 width=19)
9	-> Memoize (cost=0.15..0.72 rows=1 width=86)
10	Cache Key: mm.menuid
11	Cache Mode: logical
12	-> Index Scan using menu_pkey on menu m (cost=0.14..0.71 rows=1 width=86)
13	Index Cond: (id = mm.menuid)
14	Filter: active
15	-> Hash (cost=8.17..8.17 rows=1 width=82)
16	-> Index Scan using menutype_type_key on menutype mt (cost=0.15..8.17 rows=1 width=82)
17	Index Cond: ((type)::text = 'Lunch'::text)
18	-> Index Scan using product_pkey on product p (cost=0.14..0.30 rows=1 width=14)
19	Index Cond: (id = mm.menuitemid)
20	-> Index Only Scan using menuitem_pkey on menuitem mi (cost=0.14..0.28 rows=1 width=4)
21	Index Cond: (id = p.id)
22	-> Index Scan using memumbertype_pkey on memumbertype mmt (cost=0.15..0.71 rows=1 width=62)
23	Index Cond: (id = mm.typeid)

3.

```
SELECT * FROM vw_active_menu WHERE menutype = 'Lunch'¶ 0.088s
```

```
UPDATE Menu SET Active = FALSE WHERE Name = 'Summer Men 0.214s
```

```
INSERT INTO Menu (Name, Active, Typeld) VALUES ('Summer Me 0.744s
```

4. Оптимизација не би била потребна и нема да се подобри перформансот за голем % и затоа не додаваме индекс за оптимизација.

## View5: Table Availability.

1. **Примарен филтер по TableNumber или по Available**
2. Вработениот гледа кои маси се слободни, ова е мала табела така што прилично брзо ќе се изврши.

	Text	Duration
3.	<code>SELECT * FROM vw_table_availability WHERE Available = TRUE</code>	0.64s

4.	<code>UPDATE RestaurantTable SET Status = FALSE WHERE TableNumb</code>	1.916s
	<code>INSERT INTO RestaurantTable (Capacity, Status, TableTypeid) VAL</code>	1.347s

5. Нема потреба од индексирање сепак додавањето на индекси нема значително да ја зголеми брзината, затоа што табелата е мала, има само 150 редици.

## View6: Product Inventory.

1. **Примарен филтер по Product или по ProductType**
2. **Следење на инвентар и предупредување на ниска залиха**
3. **Време на извршување на операции**

--6то вју	<code>SELECT * FROM vw_inventory WHERE ProductType = 'I</code>	1.621s
-----------	--	--------

```
Nested Loop (cost=7.89..10.79 rows=1 width=150)
-> Hash Join (cost=7.74..10.14 rows=1 width=96)
    Hash Cond: (sp.productid = p.id)
    -> Seq Scan on storedproduct sp (cost=0.00..2.01 rows=101 width=8)
    -> Hash (cost=7.73..7.73 rows=1 width=96)
        -> Nested Loop (cost=0.16..7.73 rows=1 width=96)
            -> Seq Scan on product p (cost=0.00..2.01 rows=101 width=22)
            -> Memoize (cost=0.16..0.66 rows=1 width=82)
                Cache Key: p.typeid
                Cache Mode: logical
            -> Index Scan using producttype_pkey on producttype pt (cost=0.15..0.65 rows=1 width=8)
                Index Cond: (id = p.typeid)
                Filter: ((type)::text = 'Ingredient'::text)
        -> Index Scan using unit_pkey on unit u (cost=0.15..0.64 rows=1 width=62)
            Index Cond: (id = p.baseunitid)
```

```
UPDATE StoredProduct SET Quantity = 5 WHERE ProductId = 1 2.242s
INSERT INTO StoredProduct (Quantity, ProductId) VALUES (50, 1) 0.029s
```

4. Овие операции се доволно брзи и нема потреба за оптимизација

#### View7: Today's reservations.

1. Примарен филтер Where "Date" = CURRENT ,а секундарен по TableNumber
2. На почетокот на секој работен ден, персоналот ги гледа резервациите за денес. Се извршува многу пати.
3. Времето на извршување на операциите се

```
SELECT * FROM vw_todays_reservations 0.537s
```

```
INSERT INTO Reservation (GuestName, GuestPhone, StartTime, E 1.778s
```

```
UPDATE Reservation SET GuestPhone = '071999888' WHERE "Da 1.135s
```

4. Брзината на извршување на операциите е доволно брза и нема да има значително убрзување на времето доколку пробаме да оптимизираме со индекси

#### View8: Revenue per Waiter. 8 drop

1. Примарен филтер нема WHERE – агрегат врз сите нарачки, се користи за извештаи по келнер
2. Вработен од повисоко ниво може да гледа колку приход остварил секој келнер.
3. Времето на извршување со операциите е:

```
--8мо вјуSELECT * FROM vw_revenue_per_waiter 2m 6s
```

```
INSERT INTO Invoice (OrderId, TotalAmount, TaxAmount) VALUES 0.235s
```

```
UPDATE Invoice SET TotalAmount = 2200.00 WHERE OrderId = 1 46s
```



4. Ова е аналитички поглед и ваквите статистики се предвидени за менаџерите да ги прават на подолг период и не ни е важна брзината на извршување на истиот и затоа не пробуваме да оптимизираме со индексирање.

#### View9: ProductUsage History.

1. Примарен филтер по ProductId, Timestamp и ChangeTypeId
2. Вработен ја следи историјата на употреба на залихи – кога, колку и зошто се смениле. Бара филтрирање по продукт и датум.
3. Пред оптимизација со индекси време на извршување:

```
UPDATE ProductUsageLog SET ChangeAmount = -10 WHERE Pro 0.633s  
INSERT INTO ProductUsageLog (ProductId, ChangeAmount, Input 1.642s
```

Иницијалното извршување на погледот е неприфатливо бавно. Анализата на execution планот открива дека причината е Parallel Sequential Scan на табелата ProductUsageLog без никаков индекс на колоната ProductId.

-> Parallel Seq Scan on productusagelog (cost=0.00..190625.71 rows=63854 width=36)

```
explain SELECT * FROM ProductUsageLog WHERE ProductId = 1 ORDER BY  
Timestamp DESC;
```

Плус тоа постои сортирање што претставува уште поголем трошок. Време >360 секунди

4. Додавање индекси

```
CREATE INDEX idx_productusage_product_time ON ProductUsageLog  
(ProductId, Timestamp DESC);  
CREATE INDEX idx_productusage_changetype ON ProductUsageLog  
(ChangeTypeId, Timestamp DESC);
```

5.Откако додадовме индекси оваа операција од невозможна за користење се претвори во поглед кој може да се користи во продукција

```
UPDATE ProductUsageLog SET ChangeAmount = -10 WHERE Pro 0.027s  
INSERT INTO ProductUsageLog (ProductId, ChangeAmount, Input 0.061s
```

Text	Duration
SELECT * FROM ProductUsageLog WHERE ProductId = 1 ORDER	0.872s

## View10: Active staff with roles.

1. Примарен филтер за погледот `vw_active_staff` ќе биде според `RoleType`(тип на вработен), а исто така ќе може да се пребарува и по `Id` на вработен.
2. Системот го користи за автентикација, доделување нарачки, приказ на персоналот. Се повикува многу пати.
3. Иницијално време за извршување на овој поглед изнесува **31ms**. Ова време е прифатливо па затоа нема да имаме потреба од индексирање. Причина за брзото извршување е малиот број на записи во табелата `Employee`(200 вработени)

<code>SELECT * FROM vw_active_staff WHERE Role = 'Waiter'</code>	0.031s
--	--------

<code>SELECT * FROM vw_active_staff</code>	0.156s
--	--------

<code>UPDATE Employee SET DateResignation = CURRENT_DATE WHERE</code>	0.057s
---	--------

<code>INSERT INTO Employee (FirstName, LastName, SSN, Sex, Passw</code>	0.076s
---	--------

4. Нема потреба да се преуреди и да се додаваат индекси за оптимизација на прашалникот, времињата се прифатливи.