آیدا جلالی 401170542 - یسنا نوشیروانی 401106674

پرسش ۱

```
EXPLAIN ANALYZE
SELECT f.feedback_id, f.text_comment, f.score
FROM feedbacks f
WHERE f.score BETWEEN 4 AND 5;
```

این query ، متن نظر و امتیاز را از جدول feedbacks می‌گیرد که در آن امتیاز بین ۴ و ۵ است.

طرح اجرا پیش از نمایه

	QUERY PLAN text
1	Seq Scan on feedbacks f (cost=0.00..4.35 rows=32 width=152) (actual time=0.011..0.022 rows=32 loops=1)
2	Filter: ((score >= 4) AND (score <= 5))
3	Rows Removed by Filter: 58
4	Planning Time: 1.755 ms
5	Execution Time: 0.033 ms

در ابتدا، بدون ایندکس، پایگاه داده برای انجام این پرس‌وجو تمام جدول را بررسی می‌کند (اسکن کامل جدول) که این کار زمان زیادی می‌برد و بین ۰.۰۱۱ تا ۰.۰۲۲ میلی‌ثانیه طول می‌کشد و زمان کلی اجرای پرس‌وجو ۰.۰۳۳ میلی‌ثانیه است. زمان برنامه‌ریزی هم ۱.۷۵۵ میلی‌ثانیه است.

```
CREATE INDEX idx_feedbacks_score ON feedbacks(score);
```

طرح اجرا پس از نمایه

	QUERY PLAN text
1	Bitmap Heap Scan on feedbacks f (cost=4.47..7.95 rows=32 width=152) (actual time=0.022..0.028 rows=32 loops=1)
2	Recheck Cond: ((score >= 4) AND (score <= 5))
3	Heap Blocks: exact=3
4	-> Bitmap Index Scan on idx_feedbacks_score (cost=0.00..4.46 rows=32 width=0) (actual time=0.012..0.013 rows=32 loops=1)
5	Index Cond: ((score >= 4) AND (score <= 5))
6	Planning Time: 0.096 ms
7	Execution Time: 0.042 ms

حالا اگر یک ایندکس روی ستون امتیاز بسازیم (با دستور CREATE INDEX idx_feedbacks_score ON feedbacks(score);)، برنامه اجرای پرس‌وجو تغییر می‌کند و از روش‌های سریع‌تری مثل "Bitmap Index Scan" و "Bitmap Heap Scan" استفاده می‌شود. این باعث می‌شود پایگاه داده خیلی سریع‌تر ردیف‌های مربوطه را پیدا کند و زمان برنامه‌ریزی به ۰.۰۹۶ میلی‌ثانیه کاهش پیدا کند. در نتیجه، زمان اجرای پرس‌وجو هم به ۰.۰۴۲ میلی‌ثانیه تغییر می‌کند.

پرسش ۲

```
EXPLAIN ANALYZE
SELECT mca.multiple_choice_answer_id, mca.selected_options, opt.text
FROM multiple_choice_answers mca
JOIN options opt ON opt.multiple_choice_question_id = mca.multiple_choice_question_id
WHERE opt.number IN (1, 2, 3);
```

این query داده‌هایی از جداول multiple_choice_answers و options می‌گیرد و آنها را بر اساس multiple_choice_question_id به هم متصل می‌کند. همچنین، فقط ردیف‌هایی از جدول options که ستون number آنها برابر با ۱، ۲ یا ۳ است، فیلتر می‌کند.

```
CREATE INDEX idx_options_number ON options (number);
CREATE INDEX idx_multiple_choice_answers_question_id ON multiple_choice_answers (multiple_choice_question_id);
```

برای بهینه‌سازی این پرس‌وجو، دو ایندکس ایجاد شده: یکی برای ستون number در جدول options و دیگری برای ستون multiple_choice_question_id در جدول multiple_choice_answers.

طرح اجرا پیش از نمایه

QUERY PLAN text
Hash Join (cost=12.99..738.02 rows=39756 width=18) (actual time=0.289..14.148 rows=39756 loops=1)
Hash Cond: (mca.multiple_choice_question_id = opt.multiple_choice_question_id)
-> Seq Scan on multiple_choice_answers mca (cost=0.00..211.52 rows=13252 width=16) (actual time=0.022..1.548 rows=13252 loops=1)
-> Hash (cost=9.46..9.46 rows=282 width=10) (actual time=0.256..0.258 rows=282 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 21kB
-> Seq Scan on options opt (cost=0.00..9.46 rows=282 width=10) (actual time=0.019..0.177 rows=282 loops=1)
Filter: (number = ANY ('{1,2,3}'::integer[]))
Rows Removed by Filter: 188
Planning Time: 0.338 ms
Execution Time: 16.231 ms

طرح اجرا پس از نمایه

QUERY PLAN text
Nested Loop (cost=10.63..898.80 rows=39756 width=18) (actual time=0.046..5.692 rows=39756 loops=1)
-> Bitmap Heap Scan on options opt (cost=10.33..17.21 rows=282 width=10) (actual time=0.028..0.049 rows=282 loops=1)
Recheck Cond: (number = ANY ('{1,2,3}'::integer[]))
Heap Blocks: exact=3
-> Bitmap Index Scan on idx_options_number (cost=0.00..10.26 rows=282 width=0) (actual time=0.019..0.019 rows=282 loops=1)
Index Cond: (number = ANY ('{1,2,3}'::integer[]))
-> Memoize (cost=0.30..4.08 rows=141 width=16) (actual time=0.001..0.012 rows=141 loops=282)
Cache Key: opt.multiple_choice_question_id
Cache Mode: logical
Hits: 188 Misses: 94 Evictions: 0 Overflows: 0 Memory Usage: 653kB
-> Index Scan using idx_multiple_choice_answers_question_id on multiple_choice_answers mca (cost=0.29..4.07 rows=141 width=16) (actual time=0.001..0.017 rows=141 loops=94)
Index Cond: (multiple_choice_question_id = opt.multiple_choice_question_id)
Planning Time: 0.176 ms
Execution Time: 6.409 ms

با داشتن این ایندکس‌ها، پرس‌وجو خیلی سریع‌تر اجرا می‌شود. وقتی ایندکس‌ها وجود دارند، پرس‌وجو از روش‌های بهینه‌تری مثل Bitmap Index Scan و Index Scan استفاده می‌کند که زمان اجرای آن را به ۶.۴۰۹ میلی‌ثانیه کاهش می‌دهد. اما بدون ایندکس‌ها، پرس‌وجو از روش کندتری به نام Hash Join استفاده می‌کند و باید همه ردیف‌های جدول‌ها را یکی یکی بخواند که زمان اجرا را به ۱۶.۲۳۱ میلی‌ثانیه می‌رساند.

پرسش ۳

```sql
SELECT o.order_id, c.name AS customer_name, COUNT(fb.feedback_id) AS feedback_count
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN suggestions sg ON o.order_id = sg.order_id
JOIN feedbacks fb ON sg.suggestion_id = fb.suggestion_id
WHERE sg.status = 'completed'
GROUP BY o.order_id, c.name;
```

این query چهار جدول (orders، customers، suggestions و feedbacks) را به هم متصل می‌کند تا داده‌ها را جمع‌آوری کند. هدفش این است که تمام سفارش‌ها را همراه با اطلاعات مشتریان پیدا کند، فقط پیشنهادهایی با وضعیت "completed" را انتخاب کند، تعداد بازخوردهای مربوط به هر سفارش و مشتری را بشمارد و در نهایت نتایج را بر اساس order_id و customer_name گروه‌بندی کند.

```sql
CREATE INDEX idx_suggestions_status ON suggestions (status);
```

برای بهبود عملکرد، ایندکس idx_suggestions_status روی ستون status جدول suggestions ایجاد شده است.

طرح اجرا پیش از نمایه

QUERY PLAN text
HashAggregate (cost=217.21..218.14 rows=93 width=26) (actual time=1.128..1.154 rows=364 loops=1)
Group Key: o.order_id, c.name
Batches: 1 Memory Usage: 77kB
-> Nested Loop (cost=0.98..216.51 rows=93 width=22) (actual time=0.055..1.029 rows=364 loops=1)
-> Nested Loop (cost=0.71..188.66 rows=93 width=12) (actual time=0.050..0.717 rows=364 loops=1)
-> Merge Join (cost=0.43..98.76 rows=93 width=8) (actual time=0.043..0.310 rows=364 loops=1)
Merge Cond: (sg.suggestion_id = fb.suggestion_id)
-> Index Scan using suggestions_pkey on suggestions sg (cost=0.28..70.28 rows=408 width=8) (actual time=0.010..0.169 rows=408 loops=1)
Filter: (status = 'completed'::suggestion_status_t)
Rows Removed by Filter: 1192
-> Index Scan using feedbacks_suggestion_id_key on feedbacks fb (cost=0.15..25.62 rows=364 width=8) (actual time=0.006..0.038 rows=364 loops=1)
-> Index Scan using orders_pkey on orders o (cost=0.28..0.97 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=364)
Index Cond: (order_id = sg.order_id)
-> Index Scan using customers_pkey on customers c (cost=0.28..0.30 rows=1 width=18) (actual time=0.001..0.001 rows=1 loops=364)
Index Cond: (customer_id = o.customer_id)
Planning Time: 1.445 ms
Execution Time: 1.194 ms

قبل از ایجاد ایندکس، پایگاه داده از "Index Scan using suggestions_pkey" استفاده می‌کند و سپس فیلتر می‌کند که فقط ردیف‌های با وضعیت "completed" را نگه دارد. این کار باعث می‌شود که پایگاه داده ابتدا داده‌های زیادی را بخواند و بعد آنها را فیلتر کند، که کارآمد نیست. این باعث می‌شود که زمان برنامه‌ریزی به ۱.۴۴۵ میلی‌ثانیه و زمان اجرا به ۱.۱۹۴ میلی‌ثانیه برسد.

طرح اجرا پس از نمایه

QUERY PLAN text
HashAggregate (cost=177.66..178.59 rows=93 width=26) (actual time=1.190..1.217 rows=364 loops=1)
Group Key: o.order_id, c.name
Batches: 1 Memory Usage: 77kB
-> Nested Loop (cost=33.34..176.97 rows=93 width=22) (actual time=0.117..1.075 rows=364 loops=1)
-> Nested Loop (cost=33.07..149.11 rows=93 width=12) (actual time=0.112..0.749 rows=364 loops=1)
-> Hash Join (cost=32.79..59.21 rows=93 width=8) (actual time=0.105..0.228 rows=364 loops=1)
Hash Cond: (fb.suggestion_id = sg.suggestion_id)
-> Index Scan using feedbacks_suggestion_id_key on feedbacks fb (cost=0.15..25.62 rows=364 width=8) (actual time=0.008..0.070 rows=364 loops=1)
-> Hash (cost=27.54..27.54 rows=408 width=8) (actual time=0.091..0.091 rows=408 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 24kB
-> Bitmap Heap Scan on suggestions sg (cost=11.44..27.54 rows=408 width=8) (actual time=0.019..0.062 rows=408 loops=1)
Recheck Cond: (status = 'completed'::suggestion_status_t)
Heap Blocks: exact=11
-> Bitmap Index Scan on idx_suggestions_status (cost=0.00..11.34 rows=408 width=0) (actual time=0.009..0.009 rows=408 loops=1)
Index Cond: (status = 'completed'::suggestion_status_t)
-> Index Scan using orders_pkey on orders o (cost=0.28..0.97 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=364)
Index Cond: (order_id = sg.order_id)
-> Index Scan using customers_pkey on customers c (cost=0.28..0.30 rows=1 width=18) (actual time=0.001..0.001 rows=1 loops=364)
Index Cond: (customer_id = o.customer_id)
Planning Time: 0.328 ms
Execution Time: 1.283 ms

بعد از ایجاد ایندکس، پایگاه داده مستقیماً از "Bitmap Index Scan on idx_suggestions_status" استفاده می‌کند که سریع‌تر است و زمان برنامه‌ریزی را به ۰.۳۲۸ میلی‌ثانیه کاهش می‌دهد. با این حال، زمان اجرا کمی بیشتر می‌شود (۱.۲۸۳ میلی‌ثانیه) که ممکن است به دلیل فرآیند تبدیل داده‌های فیلتر شده به ردیف‌های واقعی باشد.

پرسش ۴

```
EXPLAIN ANALYZE
SELECT e.expert_id, e.name AS expert_name, COUNT(sg.suggestion_id) AS completed_suggestions, AVG(fb.score) AS average_score
FROM expert e
JOIN suggestions sg ON e.expert_id = sg.expert_id
JOIN feedbacks fb ON sg.suggestion_id = fb.suggestion_id
WHERE sg.status = 'completed'
GROUP BY e.expert_id, e.name
ORDER BY COUNT(sg.suggestion_id) DESC
LIMIT 5;
```

این پرس‌وجو برای پیدا کردن ۵ کارشناس برتر که بیشترین پیشنهادات "تکمیل‌شده" را دارند و میانگین امتیاز بازخورد آنهاست طراحی شده. این کار با اتصال جداول expert، suggestions و feedbacks انجام می‌شود. سپس فقط پیشنهادات "تکمیل‌شده" انتخاب می‌شود، داده‌ها بر اساس کارشناس گروه‌بندی می‌شود، نتایج بر اساس تعداد پیشنهادات تکمیل‌شده به ترتیب نزولی مرتب می‌شود و در نهایت نتایج به ۵ مورد اول محدود می‌شود.

طرح اجرا پیش از نمایه

QUERY PLAN text
Limit (cost=40.16..40.17 rows=5 width=560) (actual time=0.265..0.266 rows=5 loops=1)
-> Sort (cost=40.16..40.18 rows=10 width=560) (actual time=0.264..0.265 rows=5 loops=1)
Sort Key: (count(sg.suggestion_id)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> HashAggregate (cost=39.87..39.99 rows=10 width=560) (actual time=0.244..0.255 rows=40 loops=1)
Group Key: e.expert_id
Batches: 1 Memory Usage: 24kB
-> Hash Join (cost=18.69..39.17 rows=93 width=528) (actual time=0.096..0.199 rows=364 loops=1)
Hash Cond: (sg.expert_id = e.expert_id)
-> Hash Join (cost=17.47..37.68 rows=93 width=12) (actual time=0.071..0.137 rows=364 loops=1)
Hash Cond: (sg.suggestion_id = fb.suggestion_id)
-> Index Only Scan using idx_suggestions_status_expert on suggestions sg (cost=0.28..19.42 rows=408 width=8) (actual time=0.009..0.037 rows=408 loops=1)
Index Cond: (status = 'completed'::suggestion_status_t)
Heap Fetches: 0
-> Hash (cost=12.64..12.64 rows=364 width=8) (actual time=0.056..0.056 rows=364 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 23kB
-> Seq Scan on feedbacks fb (cost=0.00..12.64 rows=364 width=8) (actual time=0.004..0.032 rows=364 loops=1)
-> Hash (cost=1.10..1.10 rows=10 width=520) (actual time=0.018..0.018 rows=40 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 10kB
-> Seq Scan on expert e (cost=0.00..1.10 rows=10 width=520) (actual time=0.009..0.011 rows=40 loops=1)
Planning Time: 1.300 ms
Execution Time: 0.305 ms

قبل از اینکه ایندکس idx_suggestions_status_expert ایجاد شود، پایگاه داده ابتدا همه ردیف‌های جدول suggestions را می‌خواند و سپس فیلتر می‌کند تا فقط ردیف‌هایی با وضعیت "completed" را نگه دارد که این کار خیلی کارآمد نیست.

```
CREATE INDEX idx_suggestions_status_expert ON suggestions (status, expert_id, suggestion_id);
```

طرح اجرا پس از نمایه

QUERY PLAN text
Limit (cost=84.27..84.28 rows=5 width=560) (actual time=1.848..1.851 rows=5 loops=1)
-> Sort (cost=84.27..84.29 rows=10 width=560) (actual time=1.846..1.849 rows=5 loops=1)
Sort Key: (count(sg.suggestion_id)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> HashAggregate (cost=83.97..84.10 rows=10 width=560) (actual time=1.750..1.811 rows=70 loops=1)
Group Key: e.expert_id
Batches: 1 Memory Usage: 40kB
-> Hash Join (cost=29.88..82.81 rows=155 width=528) (actual time=0.449..1.509 rows=611 loops=1)
Hash Cond: (sg.expert_id = e.expert_id)
-> Hash Join (cost=28.66..81.14 rows=155 width=12) (actual time=0.387..1.223 rows=611 loops=1)
Hash Cond: (sg.suggestion_id = fb.suggestion_id)
-> Seq Scan on suggestions sg (cost=0.00..50.73 rows=668 width=8) (actual time=0.010..0.569 rows=687 loops=1)
Filter: (status = 'completed'::suggestion_status_t)
Rows Removed by Filter: 2113
-> Hash (cost=21.07..21.07 rows=607 width=8) (actual time=0.370..0.370 rows=611 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 32kB
-> Seq Scan on feedbacks fb (cost=0.00..21.07 rows=607 width=8) (actual time=0.009..0.258 rows=611 loops=1)
-> Hash (cost=1.10..1.10 rows=10 width=520) (actual time=0.056..0.056 rows=70 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> Seq Scan on expert e (cost=0.00..1.10 rows=10 width=520) (actual time=0.022..0.034 rows=70 loops=1)
Planning Time: 0.698 ms
Execution Time: 1.919 ms

بعد از ایجاد ایندکس، پایگاه داده می‌تواند مستقیماً از ایندکس استفاده کند و فقط ردیف‌هایی که وضعیت "completed" دارند را پیدا کند که باعث بهبود زیادی در عملکرد می‌شود.

اگرچه زمان برنامه‌ریزی از ۰.۶۹۸ میلی‌ثانیه به ۱.۳۰۰ میلی‌ثانیه کمی بیشتر شده، اما زمان اجرا به طرز چشمگیری از ۱.۹۱۹ میلی‌ثانیه به ۰.۳۰۵ میلی‌ثانیه کاهش پیدا کرده که این نشان می‌دهد ایندکس مفیده بوده است.

پرسش ۵

```
EXPLAIN ANALYZE
SELECT MAX(suggested_price) as max_price FROM suggestions
WHERE expert_id = 10;
```

این پرسش بیشترین هزینه پیشنهادی توسط متخصص با شماره 10 را برمی‌گرداند.

طرح اجرا پیش از نمایه

```
                                             QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Aggregate  (cost=8.01..8.02 rows=1 width=32) (actual time=0.069..0.069 rows=1 loops=1)
   ->  Seq Scan on suggestions  (cost=0.00..8.00 rows=5 width=6) (actual time=0.026..0.055 rows=5 loops=1)
         Filter: (expert_id = 10)
         Rows Removed by Filter: 395
 Planning Time: 1.636 ms
 Execution Time: 0.132 ms
(6 rows)
```

همانطور که می‌بینیم برای پیدا کردن suggestion ها که به متخصص شماره 10 داده شده نیاز به اسکن ترتیبی جدول suggestions است. با اضافه کردن نمایه روی ستون expert_id و suggested_price می‌توانیم پرسش را سریع‌تر کنیم.

```
CREATE INDEX idx_suggestions_expertid ON suggestions(expert_id, suggested_price);
```

طرح اجرا پس از نمایه

```
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Result  (cost=3.49..3.50 rows=1 width=32) (actual time=0.065..0.066 rows=1 loops=1)
   InitPlan 1
     ->  Limit  (cost=0.27..3.49 rows=1 width=6) (actual time=0.057..0.057 rows=1 loops=1)
           ->  Index Only Scan Backward using idx_suggestions_expertid on suggestions  (cost=0.27..16.36 rows=5 width=6) (actual time=0.055..0.055 rows=1 loops=1)
                 Index Cond: (expert_id = 10)
                 Heap Fetches: 1
 Planning Time: 1.746 ms
 Execution Time: 0.102 ms
(8 rows)
```

سیستم از نمایه استفاده می‌کند تا سطرها با expert_id=10 را پیدا کند و یک backward scan انجام می‌دهد تا سطر با بیشترین هزینه را بیابد.

پرسش ۶

```sql
EXPLAIN ANALYZE
SELECT
    ps.service_id,
    s.name AS service_name,
    ps.expert_id,
    e.name AS expert_name,
    ps.advertising_cost
FROM
    providing_services ps
JOIN
    expert e ON ps.expert_id = e.expert_id
JOIN
    service s ON ps.service_id = s.service_id
WHERE
    ps.advertising_cost = (
        SELECT
            MAX(ps_inner.advertising_cost)
        FROM
            providing_services ps_inner
    );
```

این پرسش اطلاعات متخصص و سرویس مربوط با تبلیغ با بیشترین هزینه را برمی‌گرداند.

طرح اجرا پیش از نمایه

```
                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=11.80..22.22 rows=1 width=34) (actual time=0.199..0.213 rows=1 loops=1)
   InitPlan 1
     ->  Aggregate  (cost=5.75..5.76 rows=1 width=32) (actual time=0.069..0.069 rows=1 loops=1)
           ->  Seq Scan on providing_services ps_inner  (cost=0.00..5.00 rows=300 width=6) (actual time=0.006..0.027 rows=300 loops=1)
   ->  Hash Join  (cost=5.76..8.15 rows=1 width=28) (actual time=0.185..0.198 rows=1 loops=1)
         Hash Cond: (e.expert_id = ps.expert_id)
         ->  Seq Scan on expert e  (cost=0.00..2.00 rows=100 width=18) (actual time=0.023..0.028 rows=100 loops=1)
         ->  Hash  (cost=5.75..5.75 rows=1 width=14) (actual time=0.139..0.139 rows=1 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on providing_services ps  (cost=0.00..5.75 rows=1 width=14) (actual time=0.100..0.130 rows=1 loops=1)
                     Filter: (advertising_cost = (InitPlan 1).col1)
                     Rows Removed by Filter: 299
   ->  Index Scan using service_pkey on service s  (cost=0.28..8.29 rows=1 width=10) (actual time=0.012..0.012 rows=1 loops=1)
         Index Cond: (service_id = ps.service_id)
 Planning Time: 4.305 ms
 Execution Time: 0.348 ms
(16 rows)
```

همانطور که می‌بینیم برای پیدا کردن بیشترین advertising_cost نیاز به جست و جوی ترتیبی داریم. با اضافه کردن نمایه روی این ستون می‌توان پرسش را سریع‌تر کرد.

```sql
CREATE INDEX idx_ps_ac ON providing_services(advertising_cost);
```

طرح اجرا پس از نمایه

```
                                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=6.26..16.68 rows=1 width=34) (actual time=0.171..0.186 rows=1 loops=1)
   InitPlan 2
     ->  Result  (cost=0.22..0.23 rows=1 width=32) (actual time=0.063..0.064 rows=1 loops=1)
           InitPlan 1
             ->  Limit  (cost=0.15..0.22 rows=1 width=6) (actual time=0.057..0.058 rows=1 loops=1)
                   ->  Index Only Scan Backward using idx_ps_ac on providing_services ps_inner  (cost=0.15..20.64 rows=300 width=6) (actual time=0.057..0.057 rows=1 loops=1)
                         Heap Fetches: 1
   ->  Hash Join  (cost=5.76..8.15 rows=1 width=28) (actual time=0.159..0.173 rows=1 loops=1)
         Hash Cond: (e.expert_id = ps.expert_id)
         ->  Seq Scan on expert e  (cost=0.00..2.00 rows=100 width=18) (actual time=0.011..0.016 rows=100 loops=1)
         ->  Hash  (cost=5.75..5.75 rows=1 width=14) (actual time=0.127..0.127 rows=1 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on providing_services ps  (cost=0.00..5.75 rows=1 width=14) (actual time=0.085..0.120 rows=1 loops=1)
                     Filter: (advertising_cost = (InitPlan 2).col1)
                     Rows Removed by Filter: 299
   ->  Index Scan using service_pkey on service s  (cost=0.28..8.29 rows=1 width=10) (actual time=0.011..0.011 rows=1 loops=1)
         Index Cond: (service_id = ps.service_id)
 Planning Time: 4.808 ms
 Execution Time: 0.288 ms
(19 rows)
```

حالا برای پیدا کردن بیشترین هزینه تبلیغات کافی‌ست به صورت برعکس روی index حرکت کنیم.

پرسش ۷

```sql
EXPLAIN ANALYZE
SELECT
    e.expert_id,
    e.name AS expert_name,
    ps.advertising_cost
FROM
    providing_services ps
JOIN
    expert e ON ps.expert_id = e.expert_id
WHERE
    ps.service_id = 322
ORDER BY
    ps.advertising_cost DESC;
```

این پرسش لیست متخصصانی که سرویس با شماره 322 را ارائه می‌کنند براساس هزینه تبلیغات مرتب می‌کند و نشان می‌دهد.

طرح اجرا پیش از نمایه

```
                                                      QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=16.10..16.11 rows=3 width=24) (actual time=0.112..0.113 rows=3 loops=1)
   Sort Key: ps.advertising_cost DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=11.54..16.07 rows=3 width=24) (actual time=0.067..0.087 rows=3 loops=1)
         Hash Cond: (e.expert_id = ps.expert_id)
         ->  Seq Scan on expert e  (cost=0.00..4.00 rows=200 width=18) (actual time=0.008..0.017 rows=200 loops=1)
         ->  Hash  (cost=11.50..11.50 rows=3 width=10) (actual time=0.047..0.047 rows=3 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on providing_services ps  (cost=0.00..11.50 rows=3 width=10) (actual time=0.009..0.040 rows=3 loops=1)
                     Filter: (service_id = 322)
                     Rows Removed by Filter: 597
 Planning Time: 1.734 ms
 Execution Time: 0.159 ms
(13 rows)
```

برای پیدا کردن جفت متخصص-سرویس‌هایی که سرویس 322 را ارائه می‌کنند به اسکن ترتیبی جدول providing_services نیاز است. با گذاشتن نمایه روی این ستون این نیاز را رفع می‌کنیم.

```sql
CREATE INDEX idx_ps_service_id ON providing_services(service_id);
```

طرح اجرا پس از نمایه

```
                                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=13.14..13.15 rows=3 width=24) (actual time=0.103..0.104 rows=3 loops=1)
   Sort Key: ps.advertising_cost DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=8.58..13.12 rows=3 width=24) (actual time=0.068..0.087 rows=3 loops=1)
         Hash Cond: (e.expert_id = ps.expert_id)
         ->  Seq Scan on expert e  (cost=0.00..4.00 rows=200 width=18) (actual time=0.009..0.018 rows=200 loops=1)
         ->  Hash  (cost=8.54..8.54 rows=3 width=10) (actual time=0.047..0.048 rows=3 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Bitmap Heap Scan on providing_services ps  (cost=4.30..8.54 rows=3 width=10) (actual time=0.038..0.040 rows=3 loops=1)
                     Recheck Cond: (service_id = 322)
                     Heap Blocks: exact=2
                     ->  Bitmap Index Scan on idx_ps_service_id  (cost=0.00..4.30 rows=3 width=0) (actual time=0.030..0.030 rows=3 loops=1)
                           Index Cond: (service_id = 322)
 Planning Time: 1.738 ms
 Execution Time: 0.152 ms
(15 rows)
```

حالا از نمایه به صورت bitmap استفاده می‌شود تا سطرهایی از جدول providing_services که مربوط به سرویس با شماره 322 هستند فیلتر شوند.

پرسش ۸

```sql
EXPLAIN ANALYZE
SELECT
    c.customer_id,
    c.name AS customer_name,
    SUM(s.suggested_price) AS total_spent
FROM
    customers c
JOIN
    orders o ON c.customer_id = o.customer_id
JOIN
    suggestions s ON o.order_id = s.order_id
WHERE
    s.status = 'completed'
GROUP BY
    c.customer_id, c.name
ORDER BY
    total_spent DESC;
```

این پرسش میزان پولی که هر مشتری صرف خدمات کرده است را برمی‌گرداند.

طرح اجرا پیش از نمایه

```
                                                          QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=55.57..55.67 rows=38 width=50) (actual time=0.389..0.390 rows=33 loops=1)
   Sort Key: (sum(s.suggested_price)) DESC
   Sort Method: quicksort  Memory: 26kB
   ->  HashAggregate  (cost=54.10..54.58 rows=38 width=50) (actual time=0.345..0.355 rows=33 loops=1)
         Group Key: c.customer_id
         Batches: 1  Memory Usage: 32kB
         ->  Nested Loop  (cost=8.62..53.91 rows=38 width=24) (actual time=0.069..0.313 rows=38 loops=1)
               ->  Hash Join  (cost=8.47..47.36 rows=38 width=10) (actual time=0.061..0.273 rows=38 loops=1)
                     Hash Cond: (o.order_id = s.order_id)
                     ->  Seq Scan on orders o  (cost=0.00..31.00 rows=2000 width=8) (actual time=0.009..0.114 rows=2000 loops=1)
                     ->  Hash  (cost=8.00..8.00 rows=38 width=10) (actual time=0.037..0.037 rows=38 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 10kB
                           ->  Seq Scan on suggestions s  (cost=0.00..8.00 rows=38 width=10) (actual time=0.009..0.030 rows=38 loops=1)
                                 Filter: (status = 'completed'::suggestion_status_t)
                                 Rows Removed by Filter: 362
               ->  Index Scan using customers_pkey on customers c  (cost=0.14..0.17 rows=1 width=18) (actual time=0.001..0.001 rows=1 loops=38)
                     Index Cond: (customer_id = o.customer_id)
 Planning Time: 2.999 ms
 Execution Time: 0.467 ms
(19 rows)
```

برای پیدا کردن پیشنهادهای کاری که کامل شده‌اند و هزینه آن‌ها توسط مشتری پرداخت شده به یک اسکن ترتیبی روی جدول suggestions نیاز است. با اضافه کردن نمایه روی ستون status می‌توان از این جست و جو پرهیز کرد.

```sql
CREATE INDEX idx_suggestions_status ON suggestions(status);
```

طرح اجرا پس از نمایه

```
CREATE INDEX
                                                                   QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=24.32..24.32 rows=2 width=552) (actual time=0.302..0.303 rows=33 loops=1)
   Sort Key: (sum(s.suggested_price)) DESC
   Sort Method: quicksort  Memory: 26kB
   ->  GroupAggregate  (cost=24.27..24.31 rows=2 width=552) (actual time=0.263..0.277 rows=33 loops=1)
         Group Key: c.customer_id
         ->  Sort  (cost=24.27..24.27 rows=2 width=536) (actual time=0.245..0.246 rows=38 loops=1)
               Sort Key: c.customer_id
               Sort Method: quicksort  Memory: 26kB
               ->  Nested Loop  (cost=4.59..24.26 rows=2 width=536) (actual time=0.077..0.228 rows=38 loops=1)
                     ->  Nested Loop  (cost=4.44..23.89 rows=2 width=20) (actual time=0.062..0.172 rows=38 loops=1)
                           ->  Bitmap Heap Scan on suggestions s  (cost=4.16..7.29 rows=2 width=20) (actual time=0.041..0.047 rows=38 loops=1)
                                 Recheck Cond: (status = 'completed'::suggestion_status_t)
                                 Heap Blocks: exact=3
                                 ->  Bitmap Index Scan on idx_suggestions_status  (cost=0.00..4.16 rows=2 width=0) (actual time=0.029..0.029 rows=38 loops=1)
                                       Index Cond: (status = 'completed'::suggestion_status_t)
                           ->  Index Scan using orders_pkey on orders o  (cost=0.28..8.30 rows=1 width=8) (actual time=0.003..0.003 rows=1 loops=38)
                                 Index Cond: (order_id = s.order_id)
                     ->  Index Scan using customers_pkey on customers c  (cost=0.14..0.18 rows=1 width=520) (actual time=0.001..0.001 rows=1 loops=38)
                           Index Cond: (customer_id = o.customer_id)
 Planning Time: 2.072 ms
 Execution Time: 0.392 ms
(21 rows)
```

سیستم ابتدا از نمایه ایجاد شده برای انتخاب suggestion هایی که کامل شده اند استفاده می‌کند و سپس باقی join ها را انجام می‌دهد.