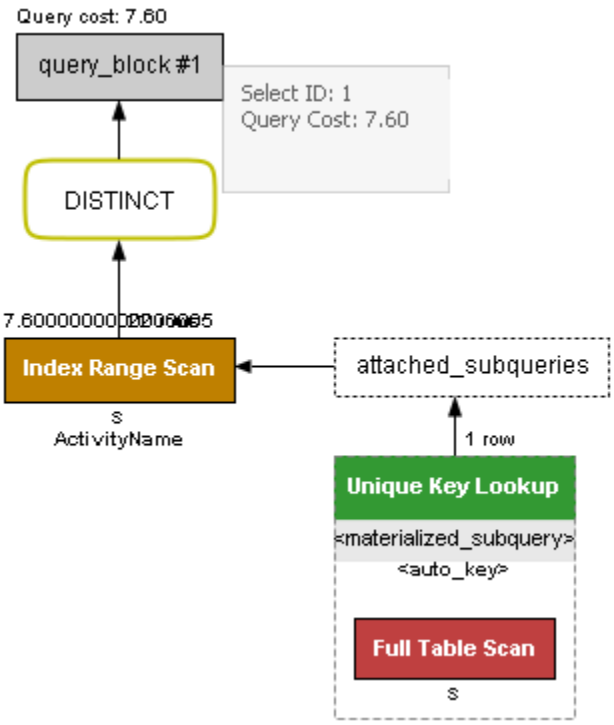
1.

Query 1: (Best Performance)

Execution Plan:



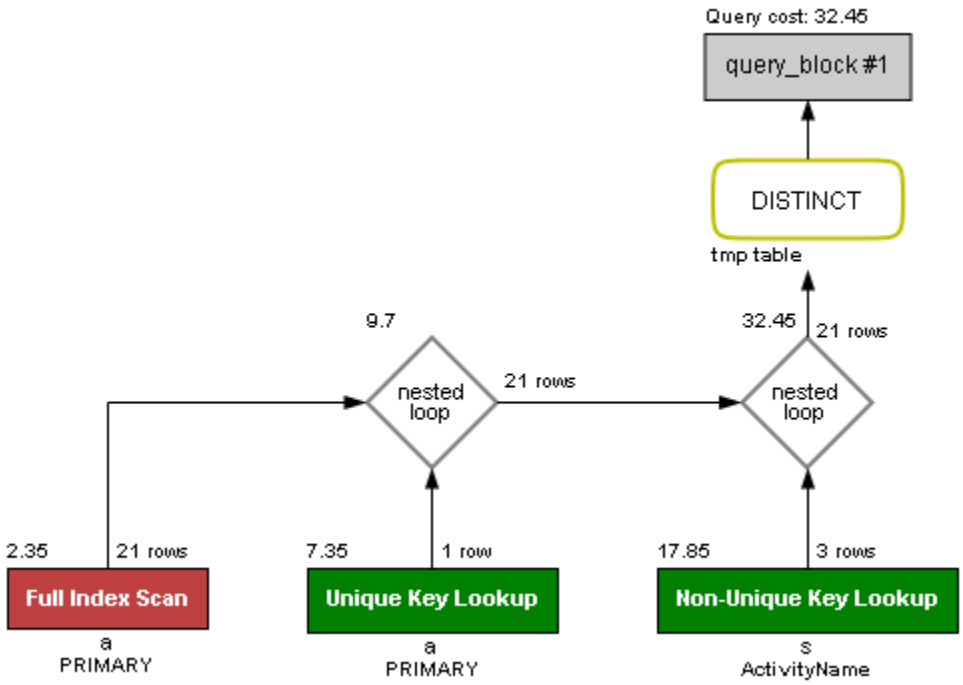
Explain Analyze:

```
-> Filter: <in_optimizer>(s.ActivityName,s.ActivityName in (select #2) is false)  (cost=1.1 rows=2) (actual time=0.121..0.193 rows=6 loops=1)
   -> Covering index skip scan for deduplication on s using ActivityName  (cost=1.1 rows=2) (actual time=0.0293..0.0752 rows=21 loops=1)
   -> Select #2 (subquery in condition; run only once)
      -> Filter: ((s.ActivityName = `<materialized_subquery>`.Activityname))  (cost=10.6..10.6 rows=1) (actual time=0.00461..0.00461 rows=0.682 loops=22)
```

This explain analyze seems to be optimized like the execution plan. It is denoted by “<in_optimizer>” and contains the index skip scan. It has no nested loops, the sub query is only executed once, and it is traversing through less relations than other queries. This is the most efficient of the 3 queries.

Query 2:

Execution Plan:



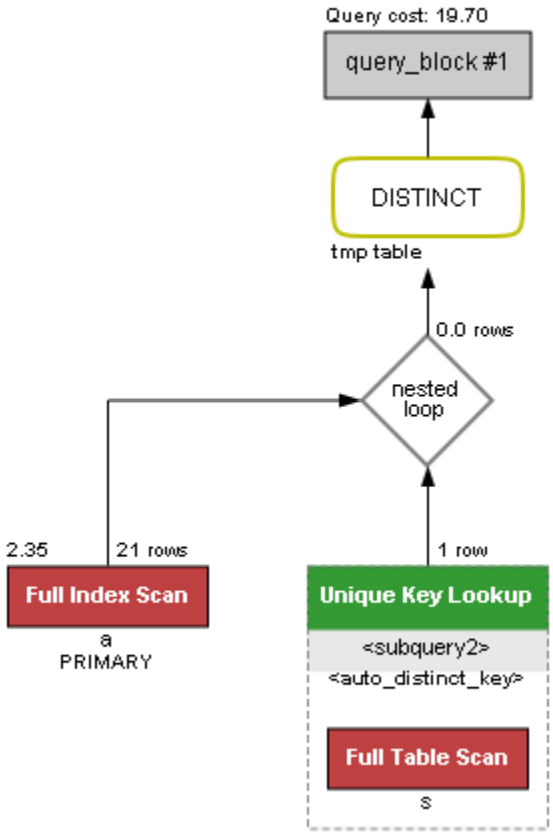
Explain Analyze:

```
-> Table scan on <temporary>  (cost=298..319 rows=1470) (actual time=0.354..0.355 rows=6 loops=1)
   -> Temporary table with deduplication  (cost=298..298 rows=1470) (actual time=0.353..0.353 rows=6 loops=1)
      -> Nested loop antijoin  (cost=151 rows=1470) (actual time=0.308..0.337 rows=6 loops=1)
         -> Covering index scan on a using PRIMARY  (cost=2.35 rows=21) (actual time=0.0387..0.0502 rows=21 loops=1)
```

Comparing the query 2 execution plan and the explain analyze, the execution plan seems to be optimized since it has a full index scan while the explain analyze has a table scan and temporary table with duplication which seems more computationally expensive. This is by far the least efficient of the 3 queries.

Query 3:

Execution Plan



Explain Analyze:

```
-> Table scan on <temporary>  (cost=298..319 rows=1470) (actual time=0.207..0.208 rows=6 loops=1)
  -> Temporary table with deduplication  (cost=298..298 rows=1470) (actual time=0.206..0.206 rows=6 loops=1)
    -> Nested loop antijoin  (cost=151 rows=1470) (actual time=0.165..0.19 rows=6 loops=1)
      -> Covering index scan on a using PRIMARY  (cost=2.35 rows=21) (actual time=0.0372..0.0448 rows=21 loops=1)
```

This execution plan had a full index scan and a full table scan but only a single nested loop. Both have an index scan using a primary key, only with a temporary table. The explain analyze scan I on the temporary table. This query does not have the best performance, but it falls in the middle of the first and second.

2.

Without Index

Execution Plan:

```
-- QUESTION 2
-- Activities in Massachussets parks with four players (With and without State index)
explain analyze
select distinct s.ActivityName
from SummerPlan as s
join Park as p on s.ParkName = p.ParkName
join Activity as a on s.ActivityName = a.ActivityName
Where p.State = "MA" and a.NumPlayers = 4;
```

Explain data not available for statement

Explain Analyze:

```
-> Table scan on <temporary>  (cost=4.36..4.36 rows=0.1) (actual time=0.233..0.234 rows=2 loops=1)
    -> Temporary table with deduplication  (cost=1.86..1.86 rows=0.1) (actual time=0.232..0.232 rows=2 loops=1)
        -> Nested loop inner join  (cost=1.85 rows=0.1) (actual time=0.0938..0.217 rows=2 loops=1)
            -> Nested loop inner join  (cost=1.5 rows=1) (actual time=0.0712..0.152 rows=27 loops=1)
```

With Index:

Execution Plan:

```
-- QUESTION 2
-- Activities in Massachussets parks with four players (With and without State index)
explain analyze
select distinct s.ActivityName
from SummerPlan as s
join Park as p on s.ParkName = p.ParkName
join Activity as a on s.ActivityName = a.ActivityName
Where p.State = "MA" and a.NumPlayers = 4;
```

Explain Analyze:

```
-> Table scan on <temporary>  (cost=4.36..4.36 rows=0.1) (actual time=0.235..0.235 rows=2 loops=1)
    -> Temporary table with deduplication  (cost=1.86..1.86 rows=0.1) (actual time=0.234..0.234 rows=2 loops=1)
        -> Nested loop inner join  (cost=1.85 rows=0.1) (actual time=0.12..0.217 rows=2 loops=1)
            -> Nested loop inner join  (cost=1.5 rows=1) (actual time=0.0969..0.156 rows=27 loops=1)
```

Between the two I see no performance benefits from the index in this instance since it may not be necessary to use. This is likely because our dataset is so small, and it will not be noticeable.

3. Explain Analyze:

- Activities while camping:

```
-> Filter: ((s.ParkName in ('Bear Brook','Pawtuckaway','Jigger Johnson')) and (s.ActivityName <> 'Camping'))  (cost=7.25 rows=29.7) (actual time=0.113..0.163 rows=28 loops=1)
    -> Table scan on s  (cost=7.25 rows=70) (actual time=0.108..0.13 rows=70 loops=1)
```

This table is efficient because it filters ParkName in the SummerPlan relation and only checks those matching rows. No nested loops. Query uses distinct to not return duplicates.

- Water Activities:

```
-> Index range scan on s using ActivityName over (ActivityName = 'Boogeyboarding') OR (ActivityName = 'Kayaking')
OR (3 more), with index condition: (s.ActivityName in ('Kayaking','Paddle
Boarding','Swimming','Surfing','Boogeyboarding'))  (cost=6.66 rows=12) (actual time=0.0406..0.0763 rows=12
loops=1)
```

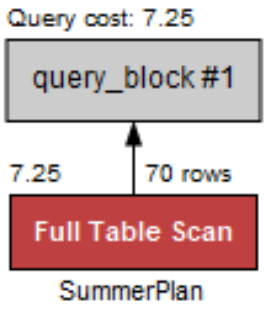
Has an index scan on ActivityName to only select matching rows. No nested loops. Query uses distinct to not return duplicates.

- Where to watch sports:

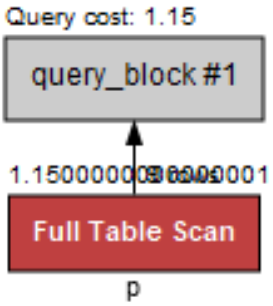
```
-> Index range scan on s using ParkName over (ParkName = 'Fenway Park') OR (ParkName = 'Gilette Stadium'), with
index condition: (s.ParkName in ('Gilette Stadium','Fenway Park'))  (cost=1.86 rows=3) (actual time=0.0335..0.0457
rows=3 loops=1)
```

Indexes on ParkName to make sure it matches the items in the tuple. No nested loops. Query uses distinct to not return duplicates.

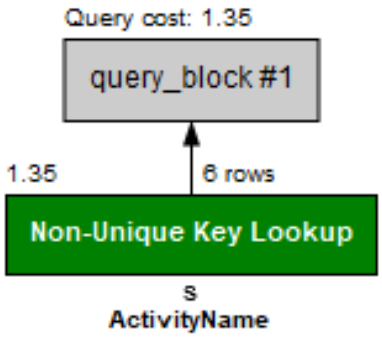
4.


Query cost: 7.25
query_block #1
7.25
70 rows
Full Table Scan
SummerPlan

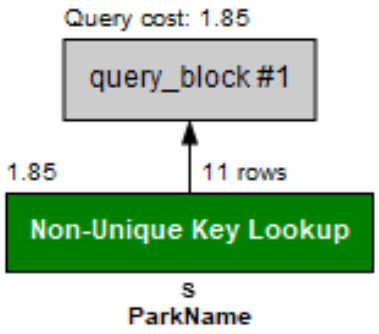
5.


Query cost: 1.15
query_block #1
Full Table Scan
p

6.


Query cost: 1.35
query_block #1
1.35
6 rows
Non-Unique Key Lookup
s
ActivityName

7.


Query cost: 1.85
query_block #1
1.85
11 rows
Non-Unique Key Lookup
s
ParkName