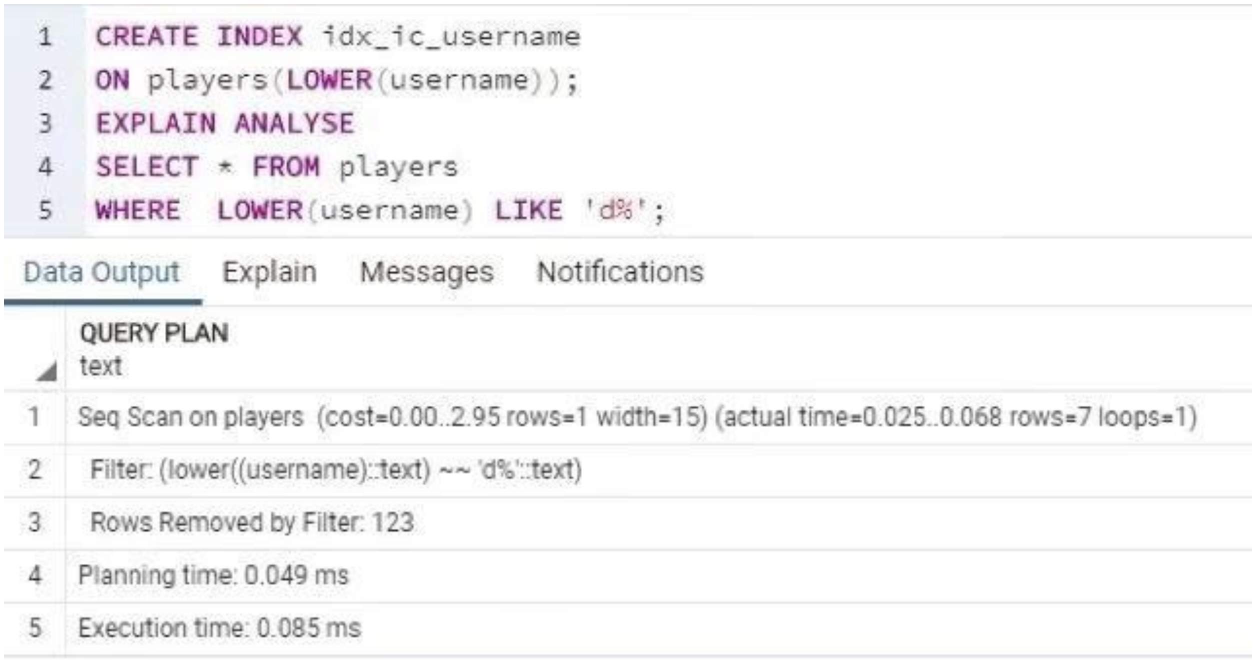
Q3.

1. Index On Expression (Functional Index): an index is defined on the result of a function applied to one or more columns of a single table. Functional indexes can be used to obtain fast access to data based on the result of function calls. Once you define an index expression, PostgreSQL will consider using that index when the expression that defines the index appears in the WHERE clause or in the ORDER BY clause of the SQL statement. In the query below we have a function LOWER() which motivates us to use Functional Index to handle the username in a case-insensitive manner.

Codes:

```
CREATE INDEX idx_ic_username

ON players(LOWER(username));

EXPLAIN ANALYSE

SELECT * FROM players

WHERE  LOWER(username) LIKE 'd%';

DROP INDEX idx_ic_username;
```

After adding index:

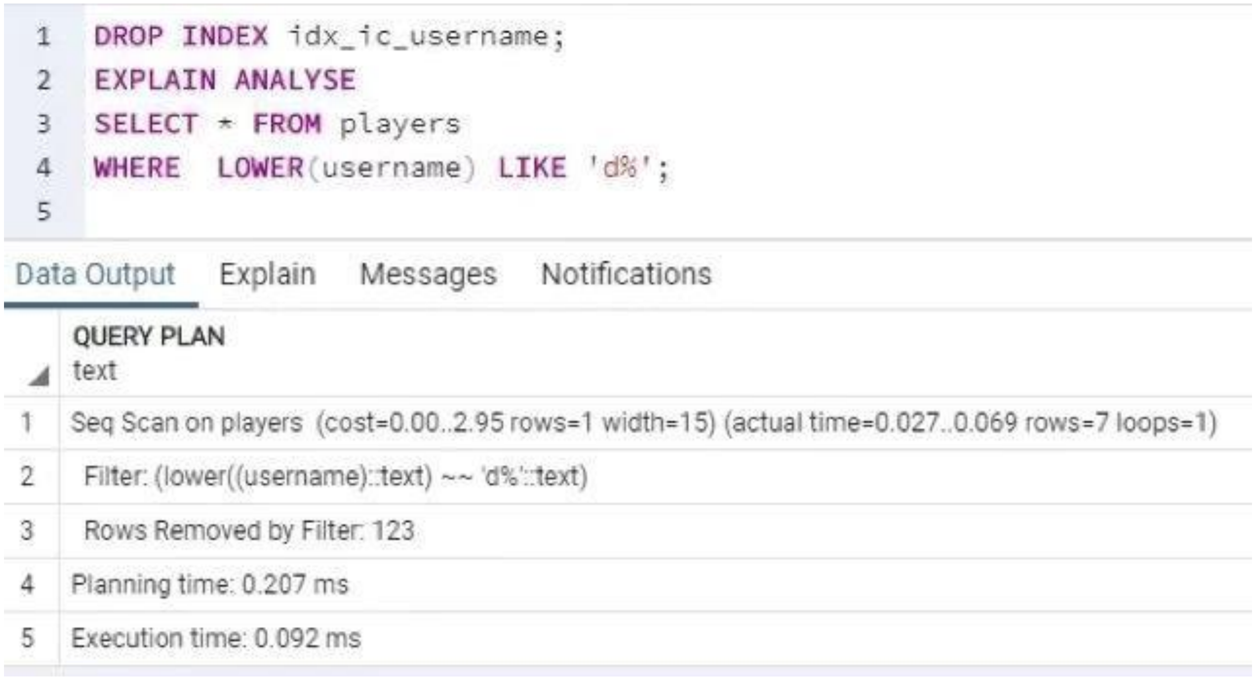


```
CREATE INDEX idx_ic_username
ON players(LOWER(username));
EXPLAIN ANALYSE
SELECT * FROM players
WHERE  LOWER(username) LIKE 'd%';
```

Data Output   Explain   Messages   Notifications

```
QUERY PLAN
text
1  Seq Scan on players  (cost=0.00..2.95 rows=1 width=15) (actual time=0.025..0.068 rows=7 loops=1)
2    Filter: (lower((username)::text) ~~ 'd%'::text)
3    Rows Removed by Filter: 123
4  Planning time: 0.049 ms
5  Execution time: 0.085 ms
```

Dropping index:



```
DROP INDEX idx_ic_username;
EXPLAIN ANALYSE
SELECT * FROM players
WHERE  LOWER(username) LIKE 'd%';
```

Data Output   Explain   Messages   Notifications

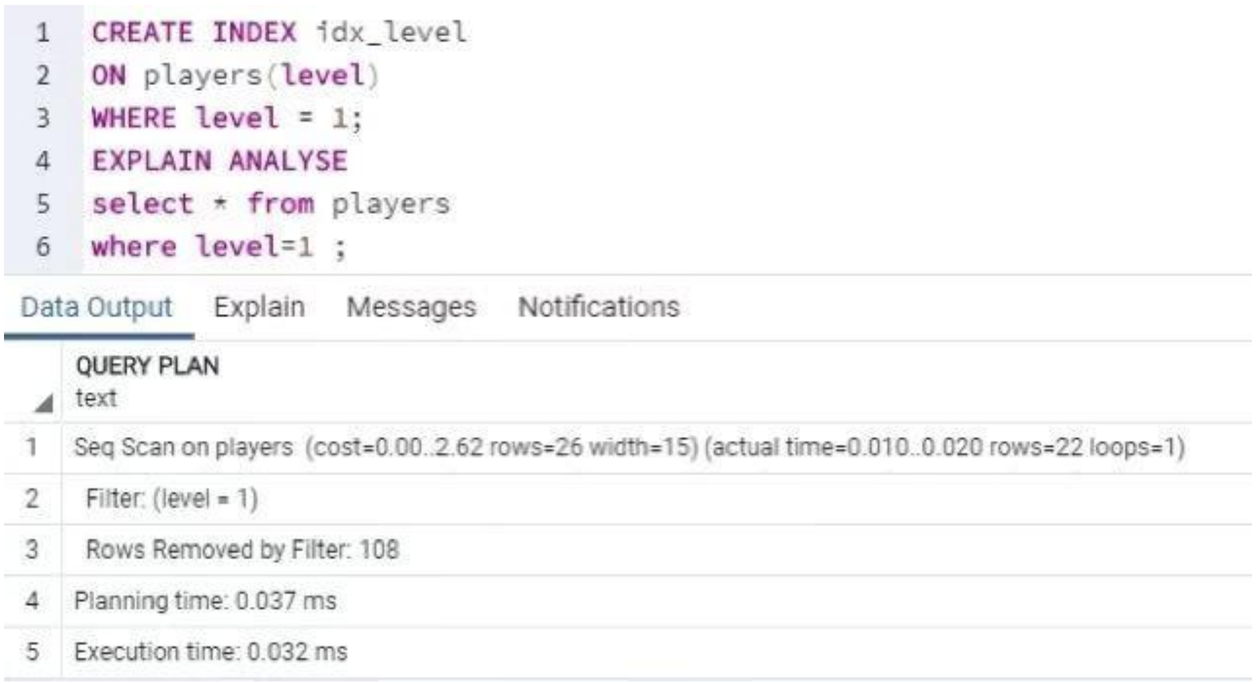
```
QUERY PLAN
text
1  Seq Scan on players  (cost=0.00..2.95 rows=1 width=15) (actual time=0.027..0.069 rows=7 loops=1)
2    Filter: (lower((username)::text) ~~ 'd%'::text)
3    Rows Removed by Filter: 123
4  Planning time: 0.207 ms
5  Execution time: 0.092 ms
```

2. PostgreSQL partial index allows us to specify the rows of a table that should be indexed. This partial index helps speed up the query while reducing the size of the index. The partial index is useful in case we have commonly used WHERE conditions which use constant values as follows(we typically are interested in

beginning players and often send some instructions to them, ‘WHERE level=1’ condition is used):

```
CREATE INDEX idx_level

ON players(level)

WHERE level = 1;

EXPLAIN ANALYSE

select * from players

where level=1 ;

DROP INDEX idx_level;
```

After adding index:

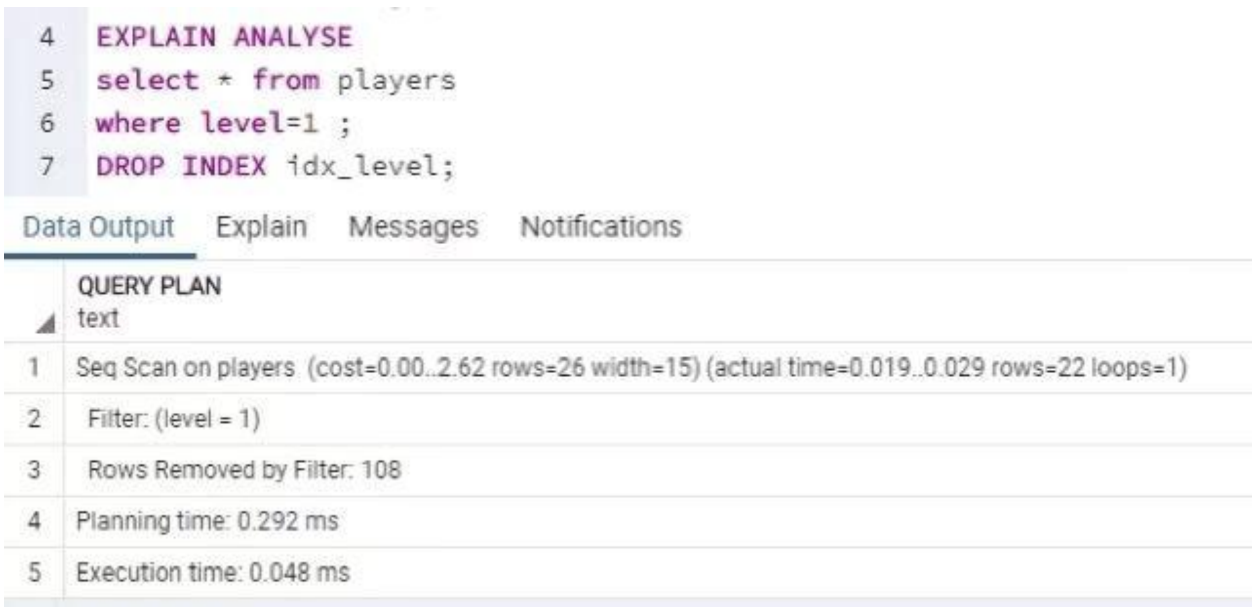


```
CREATE INDEX idx_level
ON players(level)
WHERE level = 1;
EXPLAIN ANALYSE
select * from players
where level=1 ;
```

Data Output | Explain | Messages | Notifications

```
QUERY PLAN
text
1  Seq Scan on players  (cost=0.00..2.62 rows=26 width=15) (actual time=0.010..0.020 rows=22 loops=1)
2    Filter: (level = 1)
3    Rows Removed by Filter: 108
4  Planning time: 0.037 ms
5  Execution time: 0.032 ms
```

Dropping index:



```
EXPLAIN ANALYSE
select * from players
where level=1 ;
DROP INDEX idx_level;
```

```
QUERY PLAN
text
1  Seq Scan on players  (cost=0.00..2.62 rows=26 width=15) (actual time=0.019..0.029 rows=22 loops=1)
2    Filter: (level = 1)
3    Rows Removed by Filter: 108
4  Planning time: 0.292 ms
5  Execution time: 0.048 ms
```

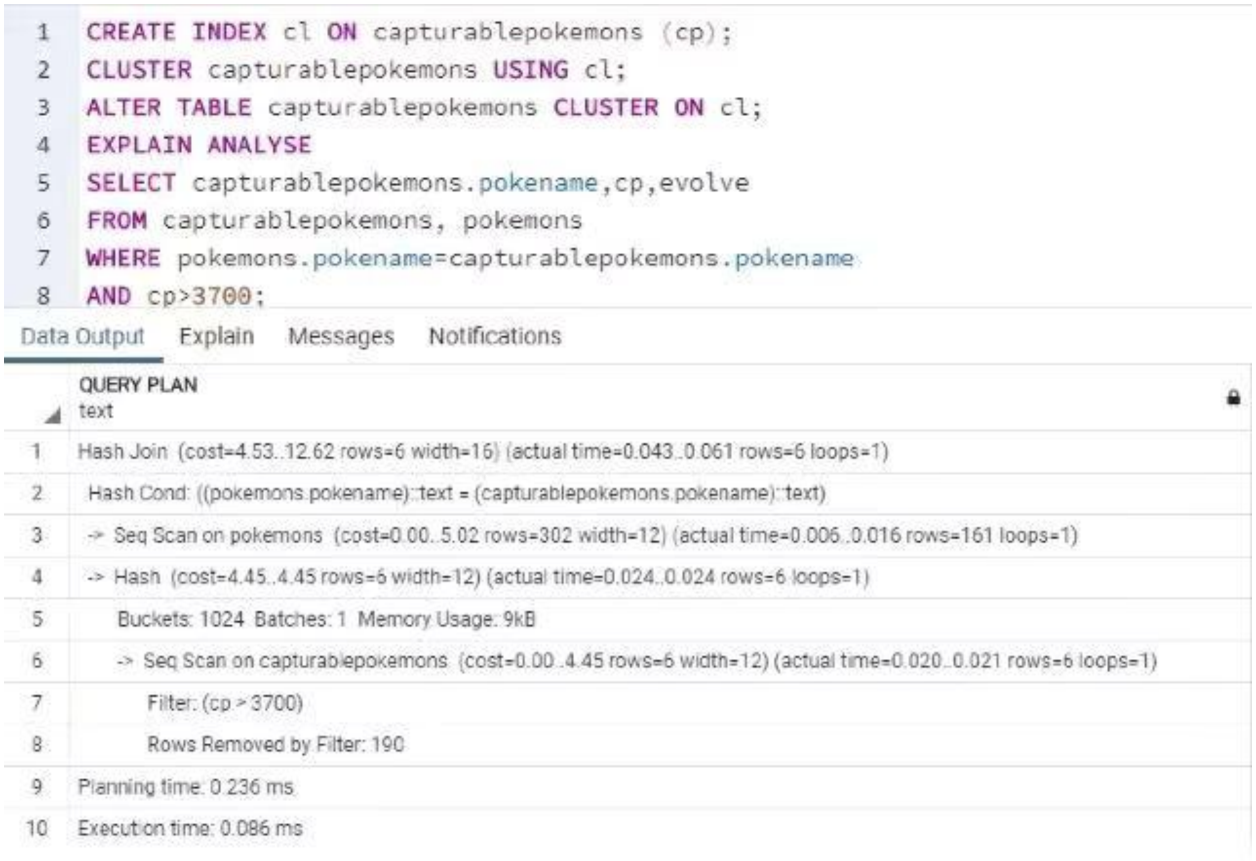
3.Cluster Index: telling the database to store the close values actually close to one another on the disk. They can uniquely identify the rows in the SQL table. Every table can have exactly one clustered index. Clustering helps by reducing page seeks. Once an index search is done and found, pulling out the data on the same page is vastly faster since once you find the start point all successive data nearby is easy picking. In this question we pick up the most powerful pokemons with the highest cp and seek to find the relationship with the cp and the evolving state.

Codes:

```
CREATE INDEX cl ON capturablepokemons (cp);

CLUSTER capturablepokemons USING cl;

ALTER TABLE capturablepokemons CLUSTER ON cl;

EXPLAIN ANALYSE

SELECT capturablepokemons.pokename,cp,evolve

FROM capturablepokemons, pokemons
```

```
WHERE pokemons.pokename=capturablepokemons.pokename

AND cp>3700;

DROP INDEX IF EXISTS cl;
```

After adding index:

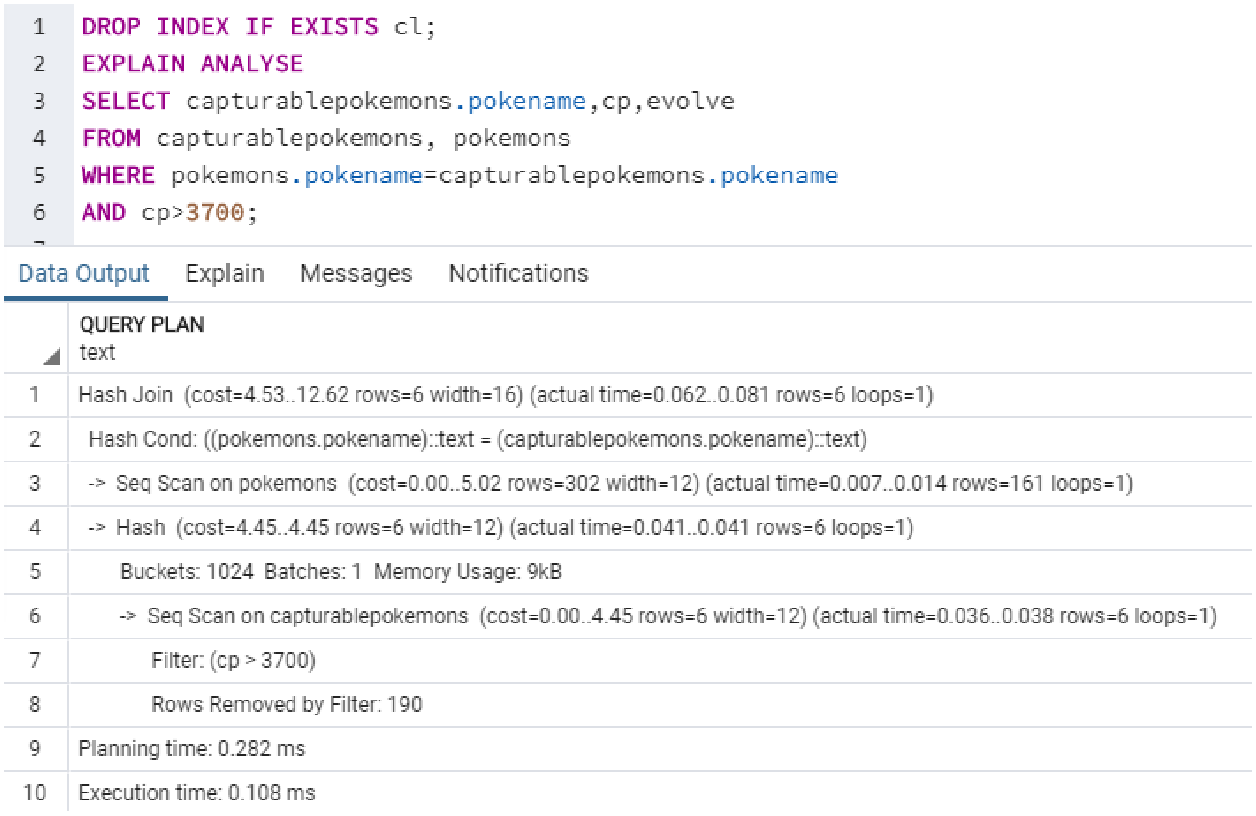


```
CREATE INDEX cl ON capturablepokemons (cp);
CLUSTER capturablepokemons USING cl;
ALTER TABLE capturablepokemons CLUSTER ON cl;
EXPLAIN ANALYSE
SELECT capturablepokemons.pokename,cp,evolve
FROM capturablepokemons, pokemons
WHERE pokemons.pokename=capturablepokemons.pokename
AND cp>3700;
```

Data Output Explain Messages Notifications

```
QUERY PLAN
text
1  Hash Join  (cost=4.53..12.62 rows=6 width=16) (actual time=0.043..0.061 rows=6 loops=1)
2    Hash Cond: ((pokemons.pokename)::text = (capturablepokemons.pokename)::text)
3    -> Seq Scan on pokemons  (cost=0.00..5.02 rows=302 width=12) (actual time=0.006..0.016 rows=161 loops=1)
4    -> Hash  (cost=4.45..4.45 rows=6 width=12) (actual time=0.024..0.024 rows=6 loops=1)
5         Buckets: 1024  Batches: 1  Memory Usage: 9kB
6         -> Seq Scan on capturablepokemons  (cost=0.00..4.45 rows=6 width=12) (actual time=0.020..0.021 rows=6 loops=1)
7               Filter: (cp > 3700)
8               Rows Removed by Filter: 190
9  Planning time: 0.236 ms
10 Execution time: 0.086 ms
```

Dropping index:



```sql
DROP INDEX IF EXISTS cl;
EXPLAIN ANALYSE
SELECT capturablepokemons.pokename,cp,evolve
FROM capturablepokemons, pokemons
WHERE pokemons.pokename=capturablepokemons.pokename
AND cp>3700;
```

```
QUERY PLAN
text
1   Hash Join  (cost=4.53..12.62 rows=6 width=16) (actual time=0.062..0.081 rows=6 loops=1)
2     Hash Cond: ((pokemons.pokename)::text = (capturablepokemons.pokename)::text)
3     ->  Seq Scan on pokemons  (cost=0.00..5.02 rows=302 width=12) (actual time=0.007..0.014 rows=161 loops=1)
4     ->  Hash  (cost=4.45..4.45 rows=6 width=12) (actual time=0.041..0.041 rows=6 loops=1)
5           Buckets: 1024  Batches: 1  Memory Usage: 9kB
6           ->  Seq Scan on capturablepokemons  (cost=0.00..4.45 rows=6 width=12) (actual time=0.036..0.038 rows=6 loops=1)
7                 Filter: (cp > 3700)
8                 Rows Removed by Filter: 190
9   Planning time: 0.282 ms
10  Execution time: 0.108 ms
```