Question 1: Page Reference String (PRS) Analysis

(a) Sequentiality:

Given the following page reference string:

D G C D B C C C B C E B C D A G F

We yield the following table:

Sequential Page Reference String	Length of Page Reference String
D	1
G	1
C, D	2
B, C, C, C, B, C	2
E	1
B, C, D	3
A	1
G	1
F	1

Which yields the following results for Sequentiality:

S (1) = 6/9 = 0.67

S (2) = 8/9 = 0.89

S (3) = 9/9 = 1

Locality:

Based on question we made the following table:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T2	T2	T1	T1	T1	T1	T2	T1	T1	T1	T2	T1	T1	T1	T2	T2	T2
D	G	C	D	B	C	C	C	B	C	E	B	C	D	A	G	F
t-1	t-2	t-3	t-4	t-5	t-6	t-7	t-8	t-9	t-10	t-11	t-12	t-13	t-14	t-15	t-16	t-17

Given windows size of 3, we yield following tables to calculate locality for transaction 1 and 2:

Transaction 1:

t-i	T1: W (t-i, τ)	T1: W (t-i, τ)	AL: W (t-i, τ)
1		0	0
2		0	0
3	C	1	.33
4	C, D	2	.67
5	B, C, D	3	1
6	B, C, D	3	1
7	B, C, D	3	1
8	B, C	2	.67
9	B, C	2	.67
10	B, C	2	.67
11	B, C	2	.67
12	B, C	2	.67
13	B, C	2	.67
14	B, C, D	3	1
15	B, C, D	3	1
16	B, C, D	3	1
17	B, C, D	3	1

Average Locality = 12.02/17 = 0.71

Transaction 2:

t-i	T2: W (t-i, τ)	T2: W (t-i, τ)	AL: W (t-i, τ)
1	D	1	.33
2		2	.67
3	C	2	.67
4	C, D	2	.67
5	B, C, D	2	.67
6	B, C, D	2	.67
7	B, C, D	3	1
8	B, C	3	1
9	B, C	3	1
10	B, C	3	1
11	B, C	3	1
12	B, C	3	1
13	B, C	3	1
14	B, C, D	3	1
15	B, C, D	3	1
16	B, C, D	3	1
17	B, C, D	3	1

Average Locality = 14.68 / 17 = 0.86

Average locality for both transactions = (.71 + .86)/ 2 = .785

(b) Given stack depth transaction calculation of buffer:

(i) How large should the buffer be, if we want no more than 7.5% of cache-misses?

Solution:

Sum of all accesses = 521 + 413 + 311 + 265 + 143 + 121 + 89 + 56 + 37 + 17 = 1973

Cache miss = 7.5 %

Percentage of cache hit = 100 – 7.5 = 92.5%

No of cache hit = 1973 * 92.5% = 1825.025 (Approx. 1825)

So, we need 1825 cache hits

With buffer size 7, cumulative frequency is 1863. So, we need buffer size of 7

(ii) How many cache hits for buffer size 5 in percentage?

Solution:

Cache hits for buffer size 5 in percentage

(521 + 413 + 311 + 265 + 143) / 1973 = 83.78%

(iii) How many cache misses will be avoided, if buffer size increased from 5 to 7?

Solution:

Cache miss with buffer size 5 = 1973 – (521 + 413 + 311 + 265 + 143) = 1973 – 1653 = 320

Cache miss with buffer size 7 = 1973 – (521 + 413 + 311 + 265 + 143 + 121 + 89) = 1973 – 1863 = 110

No of cache misses that can be avoided = 320 – 110 = 210

Question 3: Performance in PostgreSQL and pgAdmin

a) Which query plan is generated for the following query? Please describe the plan as detailed as possible. HINT: If you hover your mouse over an element in the query plan, a tool tip, with additional information will be shown.

```
SELECT * FROM lineitem WHERE l_suppkey < 10
```

Solution:

Gather:

Estimated plan rows are 5993, but actual rows returned are 5244. With difference of 769, the estimated "plan rows" is not extremely bad.

Seq scan:

From node type is "Sequential Scan" we can say filter key "lineitem.l_suppkey" is not indexded. If the filter key was indexed, then it would run "Indexed Scan" instead.

Although the filter has been used, it will scan through all the rows and omit those which do not pass the filter.

Actual startup time of sequential scan is 2.639 means initializing this step took 2.639 ms.

Actual total time spent is total 1341.201 means running this step took 1341.201 ms.

On the other hand, actual total time spent on whole query is 1353.668, which indicates high amount of time spent on sequential scan for the filter.

We can observe high time spent for sequential scan also from the estimated startup and total cost. Almost all the calculation cost is estimated to be done for sequential scan phase.

Estimated plan width 117 means each row returned is size of 117 bytes.

Total execution time of the query is 1354.921 ms.

b) How will the plan change, if you create the following index?

```
CREATE INDEX index1 ON lineitem(l_suppkey);
```

Solution:

After running the query in #b, we have created an INDEX for lineitem.l_suppkey field.

Not the query plan has changed from Sequential scan to Bitmap Index Scan, which is a drastic improvement.

From Actual Startup Time and Actual Total Time, we can observe both are same, maybe has ignorable difference. Thus, the Index Scan ran lot faster than Sequential scan.

From Startup Cost and Total Cost, we can say that the time improvement was also estimated by the query plan estimation. Total execution time of the query is now 130% better than the previous output.

c) By prefixing your query with EXPLAIN (ANALYZE, BUFFERS), PostgreSQL will explain give additional information on DB buffer usage. How does the buffer usage change if you run the following query three times? Give the output row of buffers each time.

EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM orders WHERE o_totalprice < 100

Solution:

Run 1: Buffers: shared read=26095

Run 2: Buffers: shared hit=96 read=25999

Run 3: Buffers: shared hit=192 read=25903

- "Buffers: shared read" is the number of blocks postgres read from the disk.
- Total 26,095 memory blocks were needed to read the whole output of the query from disk.
- "Buffers: shared hit" is the number of blocks retrieved from the postgres cache. With each query, postgres takes more and more data from and into the cache, and increase its own cache.
- Reading from cache is much faster, we can observe it from the better planning and execution time of the query after each run.
- After observing closely, we can see the size of "Buffers: shared hit" is increasing by the multiple of 96.
- After running 7 times, total 576 blocks were stored in postgres cache.

d) PostgreSQL ships with an extension which allows \prewarming" a table, which means filling the buffer with relevant pages. Execute the following queries and again give the Buffers ... row.

Explain what the output means and why there may still pages be read from the disk.

Solution:

```
Run 1:
["Gather  (cost=1000.00..34922.50 rows=150 width=107) (actual time=311.067..315.291 rows=0 loops=1)"
    "  Workers Planned: 2"
    "  Workers Launched: 2"
    "  Buffers: shared hit=10594 read=15501"
    "  ->  Parallel Seq Scan on orders  (cost=0.00..33907.50 rows=62 width=107) (actual time=300.791..300.791 rows=0 loops=3)"
    "        Filter: (o_totalprice < '100'::numeric)"
    "        Rows Removed by Filter: 500000"
    "        Buffers: shared hit=10594 read=15501"
    "Planning Time: 0.759 ms"
    "Execution Time: 315.389 ms"]
```

- With "prewarming" "orders" table, postgres instantiates it's shared buffer or OS cache or both with relevant pages from "orders" table.
- Thus "Buffers: shared hit" is much higher. Total 10,594 blocks were read from postgres cache.
- This query was run after running previous query 7 times. Thus, actual buffered pages were 10,594-576 = 10,018 after prewarming.

```
Run 2:
["Gather  (cost=1000.00..34922.50 rows=150 width=107) (actual time=305.266..310.124 rows=0 loops=1)"
"  Workers Planned: 2"
"  Workers Launched: 2"
"  Buffers: shared hit=15768 read=10327"
"  ->  Parallel Seq Scan on orders  (cost=0.00..33907.50 rows=62 width=107) (actual time=291.690..291.690 rows=0 loops=3)"
"        Filter: (o_totalprice < '100'::numeric)"
"        Rows Removed by Filter: 500000"
"        Buffers: shared hit=15768 read=10327"
"Planning Time: 1.284 ms"
"Execution Time: 310.587 ms"]
```

- Same query was run after a fresh restart of the system, here we get a Buffers: shared hit of 15,768.