1 Query Processing

1. Select Operation

   (a) $\rho_{(position='Manager')\wedge(city='London')\wedge(Staff.branchNo=Branch.branchNo)}(Staff \times Branch)$

```
1  for each tuple s in Staff
2    for each tuple b in Branch
3      check if s.position = 'Manager' and b.city = 'London' and
4        s.branchNo = b.branchNo
```

   (b) $\rho_{(position='Manager')\wedge(city='London')}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch)$

```
1  for each tuple s in Staff
2    for each tuple b in Branch
3      if s.branchNo = b.branchNo
4        put s × b into Temp
5
6  for each tuple t in Temp
7    check if t.position = 'London' and t.position = 'Manager'
```

   (c) $(\rho_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\rho_{city='London'}(Branch))$

```
1  for each tuple s in Staff
2    if s.position = 'Manager'
3      put s into TempS
4  for each tuple b in Branch
5    if b.position = 'London'
6      put b into TempB
7
8  for each tuple ts in TempS
9    for each tuple tb in TempB
10     check if ts.branchNo = tb.branchNo
```

2. Join Operation

   (a) Nested-loops/both are sorted/3 memory blocks
   method: sort merge join
   outer relation: r1

```
1  sort r1 on r1.C
2  sort r2 on r2.C
3  tr1 = first tuple in r1
4  tr2 = first tuple in r2
5
6  while tr1 != EOF and tr2 != EOF
7    if tr1.C < tr2.C
8      tr1 = next tuple in r1 after tr1
9    else if tr1.C > tr2.C
10     tr2 = next tuple in r2 after tr2
```

```
11    else
12      put tr1 × tr2 in the output relation
13
14      tmptr1 = next tuple in r1 after tr1
15      while tmptr1 != EOF and tmptr1.C = tr2.C
16        put tmptr1 × tr2 in the output relation
17        tmptr1 = next tuple in r1 after tr1
18
19      tmptr2 = next tuple in r2 after tr2
20      while tmptr2 != EOF and tr1.C = tmptr2.C
21        put tr1 × tmptr2 in the output relation
22        tmptr2 = next tuple in r2 after tr2
23
24      tr1 = next tuple in r1 after tr1
25      tr2 = next tuple in r2 after tr2
```

Blocks transferred: $b_{r1} + b_{r2} = 2000 + 3000 = 5000$
Seeks: $b_{r1} + b_{r2} = 2000 + 3000 = 5000$

(b) Nested-loops/none of the sorted/102 memory blocks
outer relation: r2

```
1  for each group of 100 tuples Gr2 in r2
2    for each tuple tr2 in Gr2
3      for each tuple tr1 in r1
4        check if tr1.C = tr2.C
```

Blocks transferred: $(n_{r2}/100) \times b_{r1} + b_{r2} = (30000/100) \times 2000 + 3000 = 603000$
Seeks: $2 \times (n_{r2}/100) = 2 \times (30000/100) = 600$

(c) Block Nested-loops/none of the sorted/102 memory blocks
outer relation: r1

```
1  for each group of 100 blocks Gr1 of r1
2    for each block Br2 of r2
3      for each tuple tr1 in Gr1
4        for each tuple tr2 in Br2
5          check if tr1.C = tr2.C
```

Blocks transferred: $(b_{r1}/100) \times b_{r2} + b_{r1} = (2000/100) \times 3000 + 2000 = 62000$
Seeks: $2 \times (b_{r1}/100) = 2 \times (2000/100) = 40$

2 2PL

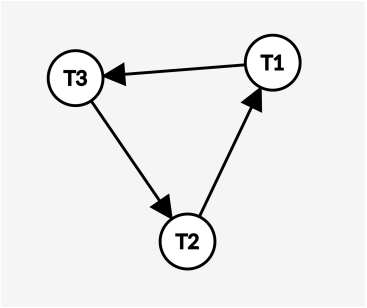
1. Minimum time: $12s$

	T1	T2	T3	T4
1	lock-S(B)	lock-S(B)		lock-S(C)
2	Read(B)	Read(B)		Read(C)
3		lock-S(A)		lock-S(A)
4		Read(A)		Read(A)
5		Unlock(B)		Unlock(C)
6	lock-X(C)	Unlock(A)		Unlock(A)
7	Write(C)		lock-X(A)	
8	Unlock(B)		Write(A)	
9	Unlock(C)		lock-X(B)	
10			Write(B)	
11			Unlock(A)	
12			Unlock(B)	

2. (a)

T1	T2	T3
lock-X(A)		
Read(A)		
		lock-X(C)
		Read(C)
	lock-X(B)	
	Read(B)	
$lock-S(C)$		
Read(C)		
	$lock-S(A)$	
	Read(A)	
		$lock-S(B)$
		Read(B)
Write(A)		
Unlock(A)		
Unlock(C)		
	Write(B)	
	Unlock(B)	
	Unlock(A)	
		Write(C)
		Unlock(C)
		Unlock(B)

Deadlock: T1 waits for T3 to unlock C, T2 waits for T1 to unlock A, and T3 waits for T2 to unlock B.



(b) wait-die protocol

T1	T2	T3
lock-X(A)		
Read(A)		
		lock-X(C)
		Read(C)
	lock-X(B)	
	Read(B)	
$lock-S(C)$		
$wait$		
Read(C)		
	$lock-S(A)$	
	$abort$	
		lock-S(B)
		Read(B)
Write(A)		
Unlock(A)		
Unlock(C)		
		Write(C)
		$Unlock(C)$
		Unlock(B)

(c) wound-wait protocol

T1	T2	T3
lock-X(A)		
Read(A)		
		lock-X(C)
		Read(C)
	lock-X(B)	
	Read(B)	
$lock-S(C)$		
$wound\ T3$		$abort$
Read(C)		
	$lock-S(A)$	
	$wait$	
	Read(A)	
Write(A)		
$Unlock(A)$		
Unlock(C)		
	Write(B)	
	Unlock(B)	
	Unlock(A)	

(d) timestamp-based protocol

T1	T2	T3
Read(A)		
		Read(C)
	Read(B)	
Read(C)		
	Read(A)	
		Read(B)
$Write(A)$		
$abort$		
	$Write(B)$	
	$abort$	
		Write(C)