Multiple Answer Multiple Choice Question

1. A,B,D
2. A,B
3. A,B,D
4. A
5. C,E
6. B
7. A,B
8. D

TRUE/FALSE Multiple Answer Multiple Choice Question

9. A
10. B
11. B
12. A
13. B
14. A
15. B
16. B
17. B
18. A

SQL (can also be written as Rel algebra)

1.

```
SELECT sname, sphone FROM sitter
WHERE ssid IN (SELECT ssid FROM sitterCalendar WHERE avaliDate BETWEEN
‘2017-05-01’ AND ‘2017-08-31’)
```

2.

```
SELECT s.ssid, s.sname, s.sphone
FROM Sitter S, BabySitting B
WHERE S.ssid = B.ssid
        AND B.rating > 7
GROUP BY s.ssid, s.sname, s.sphone
HAVING COUNT(*) > 4
```

3.

```
SELECT psid
FROM Child
GROUP BY psid
HAVING COUNT(*) > 1
INTERSECT
SELECT psid
FROM Child
WHERE sgender = 'female'
```

OR

```
SELECT psid
FROM Child
WHERE psid IN ( SELECT psid FROM Child WHERE sgender = 'female' )
GROUP BY psid
HAVING COUNT(*) > 1
```

Query Evaluation

1. Steps
    1. First read the 20,000 pages of babySitting, use the selection on availdate and pipeline it to projection and store only the required fields (ssid, totalhrs, addnlChildren) of the selected records in the memory buffers. The width of the output tuple is 4+4+4 = 12, and can be done into 2 frames ⇒ temp1. The number of records in temp1 will be 1,000,000 x 5/10,000 = 500 . Total IO is 20,000 reads.
    2. Block Nested with temp1 as outer and sitter as the inner relation. Temp1 is in memory so join read cost = 20 (for sitter). Compute the amount earned for this babysitting reservation save only required fields (ssid, sname, amt), width = 4+20+8 = 32 … takes about 4 frames (no need to write can be held in memory). Total IO = 20 reads
    3. Do an in memory sorting based on ssid, sname. This is then grouped (in memory) - pipelined to sum - and pipeline to project - and send to client directly.

    Total IO is therefore only 20,000 + 20 = 20,020

2. Steps

    Create a clustered index on availDate of babySitting

    1. For selection operator on availDate use this index and , read 1 leaf page + 10 data pages , project and store only required fields in memory (ssid, totalhrs, addnlChildren) (2 frames) -> temp1 = 11 reads

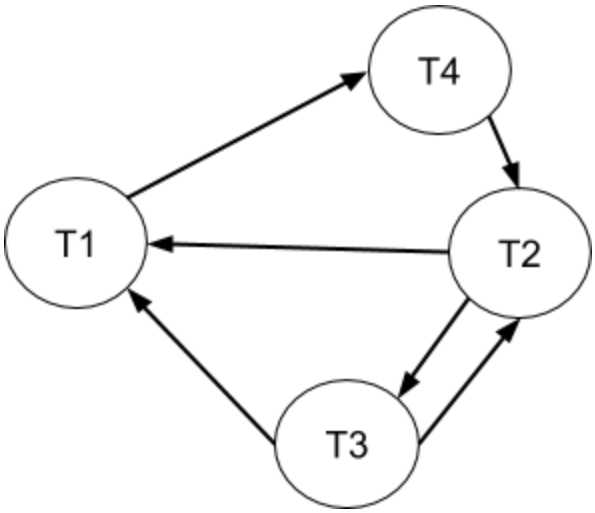
    Rest of the steps same as before - results in 11 + 20 = 31 I/O

3. N1 processes about 10,000 records (20% of the 50,000 qualified from babySitting), both entering and leaving. Incoming tuple size is either 108B or 104B, leaving tuple size is 24B

N2 processes about 10,000 records too (because only half of the parents will be needed), both entering and leaving. Incoming tuple size is either 60B or 56B and outgoing tuple size is 36B

Transactions

1. Dependency Graph



2. Execution sequence Transactions Lock Table

T1	T2	T3	T4	a	b	c
		s3(a),r3(a),u3(a)		s3,s3,-		
s1(a),r1(a),u1(a)				s1,s1,-		
			s4(c),r4(c),u4(c)			s4,s4,-
	s2(b),r2(b),u2(b)				s2,s2,-	
		x3(b)w3(b)			X3,X3	
		u3(b)c3				
	x2(b),w2(b)				X2,X2	
s1(b)r1(b)			s4(a),r4(a),u4(a)	s4,s4,-	X2,WL:S1	
			c4		X2,WL:S1	
	x2(c),w2(c)				X2,WL:S1	X2,X2
	u2(b,c),c2				S1	
r1(b),u1(b)					S1,-	
x1(a),w(a)				X1,X1		
u1(a),c1						

3. Anomalies

(A) Lost Update, yes w3(b) will be overwritten by w2(b)

(B) Dirty Read, no because X locks are held for write till a commit.

(C) Unrepeatable read, no, none of the transactions read the same item twice,otherwise it can happen if another transaction's writes the item it has already read and this transaction reads it again (after the write transaction has committed and released its locks).

Large Scale Data Processing and Trends in Databases

1.

```
femaleSitters = filter sitter by gender = 'female';
parentsBabySitting = filter babySitting by psid = 123456789;

jnd = join femaleSitters by ssid, parentsBabySitting by ssid;

grpd = group jnd by (ssid, name, phone);

smmd = foreach grpd generate($0), COUNT($1) as bookingCount;
fltrsmmd = filter smmd by bookingCount >= 10;

srtd = order fltrsmmd by name;
dump srtd;
```

2.

(A)

```
Map(referencingURL, referencedURL_List)
{
  For each referencedURL in referencedURL_List
      if(referencingURL != referencedURL)
            output(referencedURL,referencingURL);
}

Reduce(referencedURL, referencingURL_List)
{
  Qref = {}, counter=0;
  For each url in referencingURL_List
      if url not in Qref
            insert url into Qref
            Counter = counter + 1;
  output(referencedURL, counter);
}
```

(B)

```
Map:
For each tuple(referencedURL, referenceCount) of first MR job
      output (referencedURL, ('RefCounts', referenceCount)
For each tuple(URL, popValue) of popularity
      output (URL, ('popularity', popValue))
```

```
Reduce:
For each tuple (URL, tupleList)
      refTuple = NULL
      popTuple = NULL
      For each tuple t in tupleList
            If t.rel == 'RefCounts' refTuple = t;
            Else If t.rel == 'popularity' popTuple = t;
      If popTuple == NULL
            ouput(URL, refTuple.referenceCount)
      Else
            ouput(URL, refTuple.referenceCount * popTuple.popValue)
```

Graph databases

1. Draw graph database instance based on the following graph model

Nodes
(parent {name:"", sid:xx})
(babysitter {name:"", sid:xx, hourlyRate:yy})

Relationships
(parent) -[preferred_sitter {rating:x}]->(babysitter)

(parent1) -[knows]->(parent2)
(parent2) -[knows]->(parent1)

2. CQL

```
MATCH
(me:Parent{name:'Melanie'})-[r2:KNOWS*]->(n1:Parent)-[r:PREFERRED_SIT
TER]->(b:BabySitter) RETURN me, r2, n1, r, b
```

3. CQL

```
MATCH (p:Parent)-[r:PREFERRED_SITTER]->(b:BabySitter)
CREATE (p)-[r1:KNOWS]->(b), (b)-[r2:KNOWS]->(p)
```