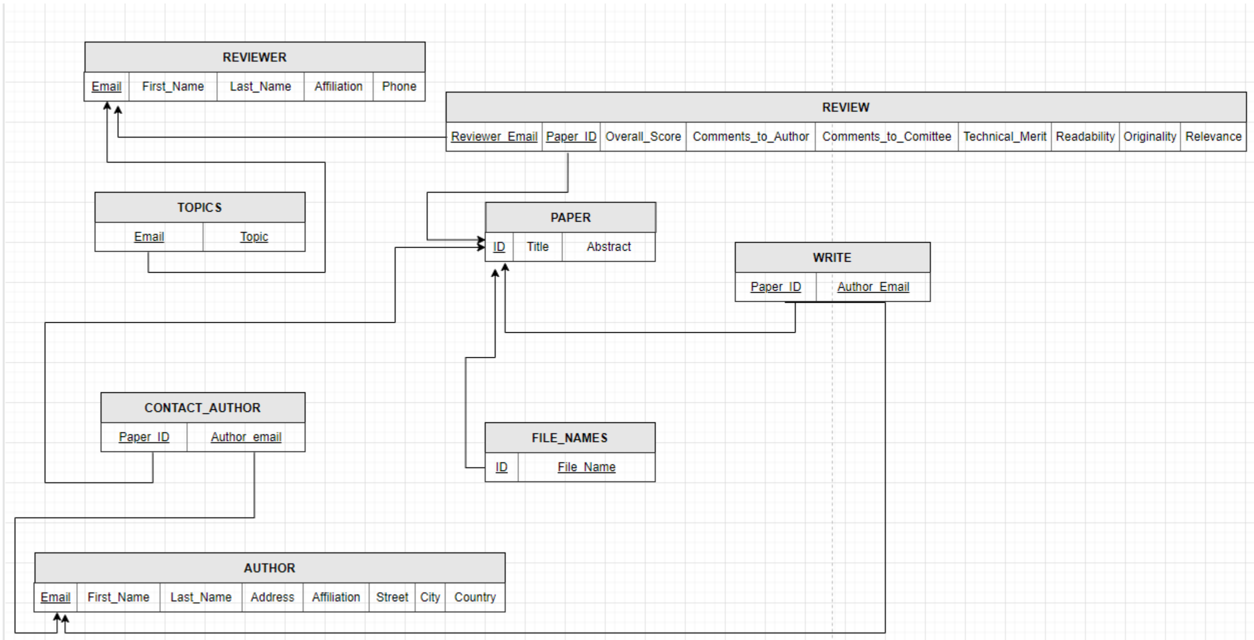
SOFTENG 351: Exam

July 4, 2020

Aiden Burgess
abur970 - 600280511

Question 1



Question 2

a)

i.

```sql
ALTER TABLE COURSE ALTER Department SET DEFAULT 'CS';
```

ii.

```sql
ALTER TABLE STUDENT ADD COLUMN Birth_Date DATE;
```

iii.

```sql
UPDATE GRADE_REPORT
SET Grade='B'
WHERE Student_number='17' AND Section_identifier='119';
```

iv.

```sql
SELECT Course_name, Course_number, Credit_hours, Semester, Year, Grade
FROM (GRADE_REPORT NATURAL JOIN SECTION NATURAL JOIN COURSE)
WHERE Student_number='8'
```

b)

i.

$ANDERSON\_SECTIONS \leftarrow \sigma_{Instructor='Anderson'}(SECTION)$

$NUM\_STUDENTS \leftarrow {}_{Section\_identifier}\Im_{COUNT\ Student\_number}(GRADE\_REPORT * ANDERSON\_SECTIONS)$

$COURSE\_INFO \leftarrow \pi_{Course\_name,\ Course\_number,\ Semester,\ Year}(COURSE * ANDERSON\_SECTIONS)$

$RESULT \leftarrow COURSE\_INFO * NUM\_STUDENTS$

ii.

Assume a student can not take two of the same course in the same year.

Assume that to satisfy the requirement a student needs to take all courses (non duplicate) available in the year '08', regardless of major.

$ALL\_COURSES \leftarrow \pi_{Course\_number}(\sigma_{Year='08'}(SECTION))$

$STUDENT\_COURSES \leftarrow \sigma_{Year='08'}(STUDENT * GRADE\_REPORT * SECTION)$

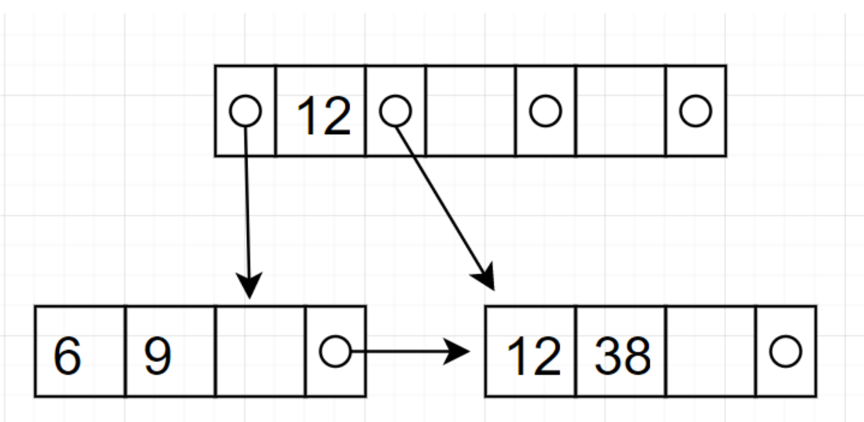
$NUM\_COURSES(Total\_courses) \leftarrow \Im_{COUNT\ Course\_number}(ALL\_COURSES)$

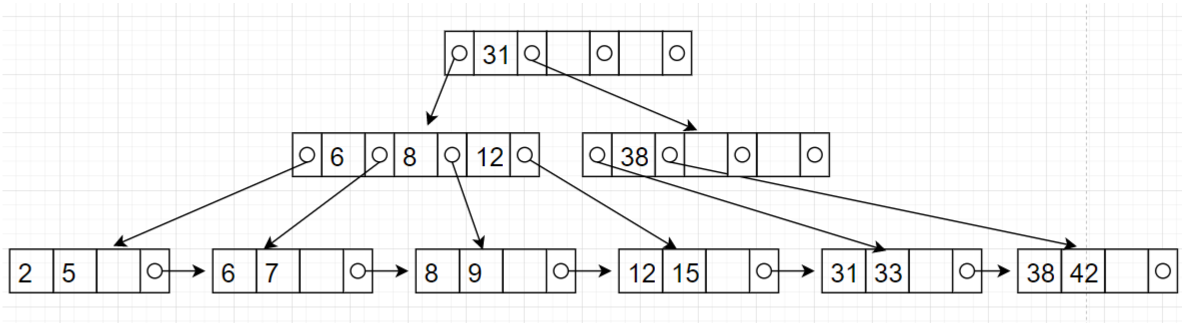
$STUDENT\_NUM\_COURSES(Student\_courses) \leftarrow_{Student\_number} \Im_{COUNT\ Course\_number}(STUDENT\_COURSES)$

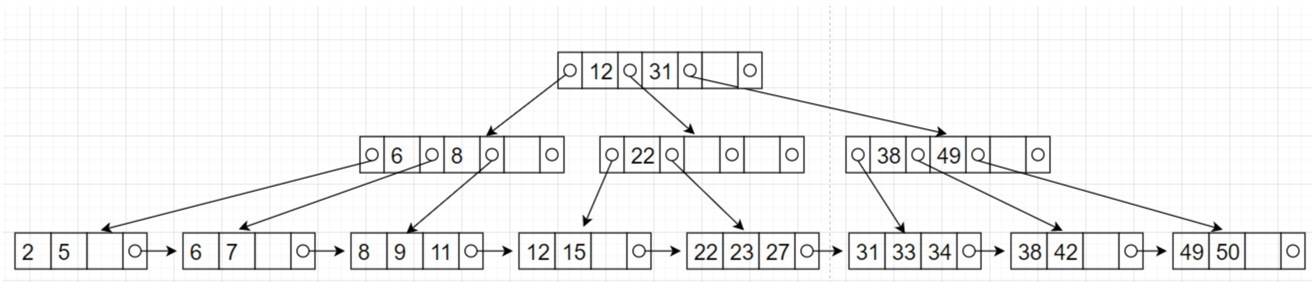
$RESULT(Name, Major) \leftarrow NUM\_COURSES \bowtie_{Total\_courses=Student\_courses} STUDENT\_NUM\_COURSES$

Question 3

a)

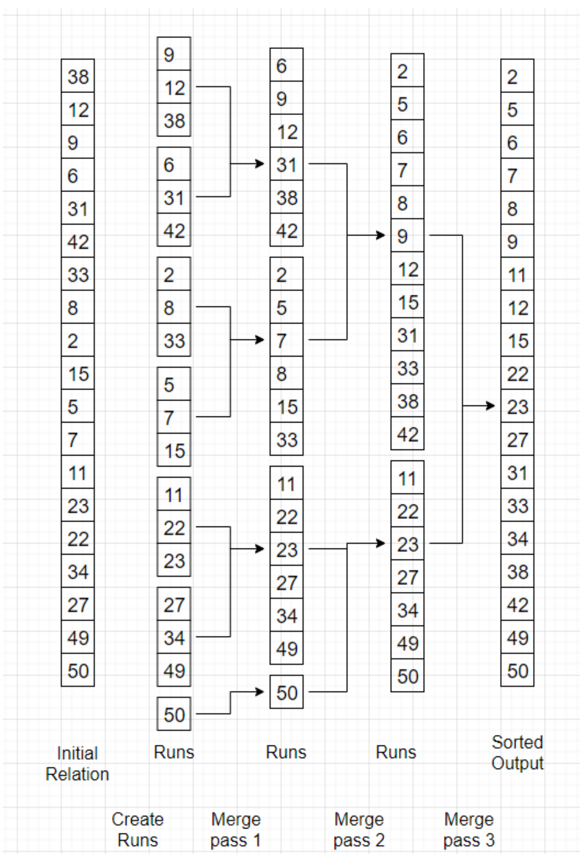



31
6
8
12
38
2
5
6
7
8
9
12
15
31
33
38
42


12
31
6
8
22
38
49
2
5
6
7
8
9
11
12
15
22
23
27
31
33
34
38
42
49
50

b)

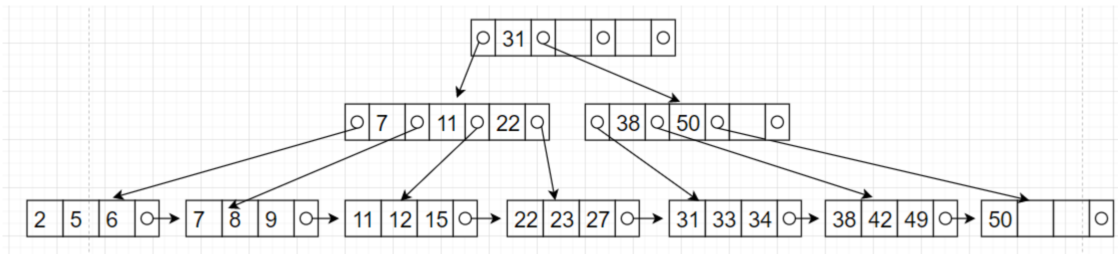
i.


Initial Relation
Runs
Runs
Runs
Sorted Output
Create Runs
Merge pass 1
Merge pass 2
Merge pass 3

Each pass takes $19 \times 2 = 38\, I/Os$

As there are three passes, the sorting takes $38 \times 3 = 114\, I/Os$ in total.

ii.



Question 4

Assume that each attribute of a "combined key" is a key as well.

Nested Loop Join then Nested Loop Join

Assume that joining the smallest relations first leads to lowest cost. Therefore, join y and z.

As z is smaller than y, it will be used in the outer loop.

$y \bowtie z = |z| \times \frac{n_y}{max(V(C,y), V(C,z),)} = 300 \times \frac{200}{100} = 300\, I/Os$

As y and z have a common key of A, the calculation of $n_{y \bowtie z}$ is relatively simple:

$|y \bowtie z| = min(|y|, |z|) = 300$

As there is no pipelining, the intermediate result must be written:

$Cost\, of\, writing = \frac{|y \bowtie z|}{bfr} = \frac{300}{2} = 150\, I/Os$

As $y \bowtie z$ is smaller than x, it will be used in the outer loop.

$x \bowtie (y \bowtie z) = |y \bowtie z| \times \frac{n_x}{V(A,x)} = 300 \times \frac{250}{50} = 1500\, I/Os$

Total Cost $= 300 + 150 + 1500 = 1950\, I/Os$

Merge Join then Nested Loop Join

As x and y are both clustered on B, then

$x \bowtie y = n_x + n_y = 250 + 200 = 450\, I/Os$

As x and y have a common key of B, the calculation of $n_{x \bowtie y}$ is relatively simple:

$|x \bowtie y| = min(|x|, |y|) = 200$

As there is no pipelining, the intermediate result must be written:

$Cost\, of\, writing = \frac{|x \bowtie y|}{bfr} = \frac{200}{2} = 100\, I/Os$

$(x \bowtie y) \bowtie z = n_{x \bowtie y} + n_z = 200 + 400 = 600\, I/Os$

Total Cost $= 450 + 100 + 600 = 1150\, I/Os$

Therefore, the most optimal cost is by using merge join on x and z, then using nested loop join on the resultant relation with y.

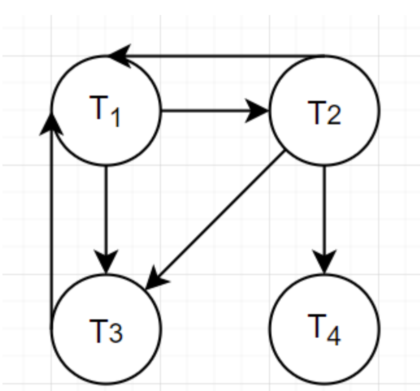
The cost is 1150 I/Os.

Question 5

a)

Deadlock occurs when a cycle appears in a wait-for graph. This means that a transaction, T_1 is waiting for a lock held by another transaction T_2, and that neither can be completed until the other has been completed. However, this is not possible using conservative 2PL because the earlier transaction T_1 which started earlier has already requested the locks needed, so it does not wait for a lock held by T_2. By induction, this process can be repeated until we reach the oldest transaction in the schedule. This transaction T_0 does not wait for any locks and therefore completes successfully and releases all locks held. This process is repeated until all transactions are completed. Therefore, a cycle is not possible, as there is an order to which locks are requested such that the oldest transaction does not wait for a lock held by any other.

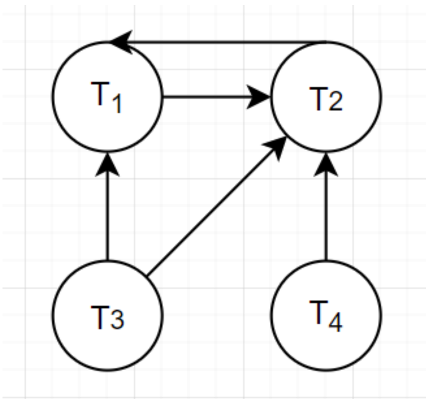
b)



As there is a cycle in the graph (e.g. T_1 and T_2, the schedule is not conflict-serializable.

c)

i.



There is a deadlock as a cycle exists between T_1 and T_2.

ii.

T_1 is waiting on T_2 for the lock on B.

T_2 is waiting on T_1 for the lock on A.

As T_2 has a higher TS, T_2 is aborted, and subsequently the lock on A is released. Immediately, T_1 grabs the lock exclusive lock on B and T_2 is waiting for T_1 for the lock on A and B. This is accepted because $TS(T_1) < TS(T_2)$.

This allows T_1 to finish its transaction and commit. Subsequently, T_2 can grab the locks for A and B and complete its transaction.

Therefore, the deadlock has been avoided.

Question 6

1

As the transaction has been committed and is included in a checkpoint, no further instructions are needed. No action

2

Redo all write_item operations of T_2 from the log after the checkpoint, in the order in which they were written into the log, using the REDO procedure.

3

Undo all the write_item operations of T_3 using the UNDO procedure before the checkpoint. The operations should be undone in the reverse of the order in which they were written to the log.

4

Redo all write_item operations of T_4 from the log after the checkpoint, in the order in which they were written into the log, using the REDO procedure.

5

Undo all the write_item operations of T_5 using the UNDO procedure after the checkpoint. The operations should be undone in the reverse of the order in which they were written to the log.