Student Number: ________________

HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMP3311: Database Management Systems

Spring Semester, 2018

Final Examination

May 19th, 2018

4:30pm to 7:30pm

Name: ____________________ Student Number: ______________________

Email: ____________________

Instructions:

1. This is an open-note examination. You are allowed to bring and use the printed course slides from lectures, tutorials and labs in the examination. No electronic device is allowed throughout the exam.
2. This examination paper consists of **26** pages and **6** questions.
3. Please write your name, student ID and Email on this page.
4. For each subsequent page, please write your student ID at the top of the page in the space provided.
5. Please answer **all** the questions within the space provided on the examination paper. You may use the back of the pages for your rough work and afterwards drawn a diagonal line through it to show that it is not part of your answer.
6. Please read each question very carefully and answer the question clearly and to the point. Make sure that your answers are neatly written, readable and legible.
7. Leave all pages stapled together.
8. The examination period will last for **3 hours**.
9. Stop writing immediately when the time is up.

Questions	Marks	Scores
1	30	
2	16	
3	16	
4	10	
5	14	
6	14	
Total	100	

Student Number: __________________

Q1 Multiple-choice Questions (30 marks)

For each question, there is only one correct answer. Please put your answers in the following boxes. Only answers in the boxes will be considered and graded.

1	**C**	**6**	**C**	**11**	**D**	**16**	**A**
2	**C**	**7**	**A**	**12**	**A**	**17**	**A**
3	**A**	**8**	**B**	**13**	**B**	**18**	**A**
4	**C**	**9**	**A**	**14**	**D**	**19**	**B**
5	**B**	**10**	**B**	**15**	**B**	**20**	**C**

1. Which of the following statements about file organization and indexing are **true**?

   I. The cost of inserting or deleting of a record in a sorted file is better than that of a heap file.
   II. In a dense index, we should create index entries for all the distinct search-key values.
   III. Dynamic hashing is better than static hashing when the size of a database often changes drastically.

   A. I Only
   B. II Only
   C. II and III Only
   D. I, II and III

2. Consider the following relational schemas of a database where key attributes are underlined:
   - Course(courseID, department, instrID)
   - Instructor(instrID, office)
   - Student(studentID, major)
   - Enroll(studentID, courseID)

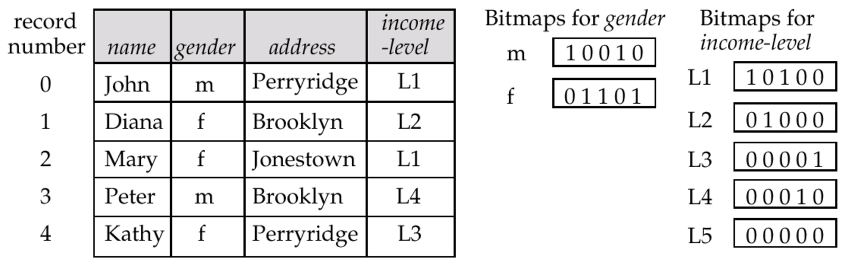
   Suppose there are four types of database queries commonly occurred:

   Q1: Given a courseID, find the department offering that course.
   Q2: List all studentIDs together with (if any) the departments they are taking their courses in.
   Q3: Given a studentID, find the IDs of all courses the student is enrolled in.
   Q4: List the offices of instructors teaching at least one course.

   Which of the following indices is **NOT** useful in speeding up execution of one or more of the above queries?

   A. Index on Instructor.instrID
   B. Index on Course.courseID
   C. Index on Course.department
   D. Index on Course.instrID

3. Consider the bitmap example shown in the following figure.



record number	*name*	*gender*	*address*	*income-level*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income-level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Which one is the correct bitmap operations for finding the records satisfying the conditions "gender = female" and "income level lower than L4 but higher than L1"? The income levels are set to be L5 > L4>…>L1?

I. (01000 OR 00001) AND 01101
II. (01000 AND 00001) AND NOT(10010)
III. (NOT (10100) OR NOT(00010)) AND 01101

A. I Only.
B. I and II Only.
C. I and III Only.
D. II and III Only.

4. Suppose we run an external sort merge on a file that consists of 108 pages using only 5 pages of main memory. Which of the following is **true** in the sorting process?

A. The number of runs in PASS 0 (i.e. the first pass) is 21.
B. The number of runs in PASS 1 (i.e. the second pass) is 5.
C. The total number of PASS in the process is 4.
D. The total number of page access in the process is 540.

5. Suppose we process $r \bowtie s$ by using the nested-loop join algorithm, where *r* has 100 pages and *s* has 200 pages. Each page can hold 10 records of *r* or *s*. Let *r* has 3 attributes (*A, B, C*) and *s* has 3 attributes (*A, D, E*). Each page can hold 10 records of *r* or *s*. What is the minimal number of memory pages if we want to complete the join with no more than 500 page I/O cost?

A. 3
B. 52
C. 300
D. 20000

6. Using the information of *r* and *s* in Question 5, we further assume both relations are already sorted on *A*. What is the best I/O cost estimation of running a sort-merge join algorithm for processing $r \bowtie s$?

A. 3
B. 52
C. 300
D. 20000

7. Using the information of *r* and *s* in Question 5, we further assume the join attribute is only *A* and a three level B+ tree index is available on *s.A*. What is the best I/O cost estimation of running an indexed nested-loop join algorithm for processing $r \bowtie s$?

   A. 4100
   B. 5100
   C. 8200
   D. 10000

8. Using the information of *r* and *s* in Question 5, we further assume the join attribute is only *A* and a hash function *h* on *A* can partition each relation into 10 buckets. Which of the following is **true** about running a hash-join algorithm for processing $r \bowtie s$?

   A. All buckets need only 10 pages for the partitioning.
   B. We only need 11 memory pages to perform the hash join.
   C. The hash function *h* is also used in the build and probe phase.
   D. If we want to apply hybrid hash-join, we should have at least 100 memory pages to hold *r*.

9. Consider a database consisting of the following three tables:
   - Customers (<u>cid</u>, cname, job, age, street, city)
   - Branches (<u>bid</u>, bname, city)
   - Accounts (<u>*cid, bid*</u>, balance)

   Which of the following statements is **false**?

   A. $\pi_{cname,balance}(Customers \bowtie Accounts) = \pi_{cname,balance}\big((\pi_{cname}(Customers)) \bowtie (\pi_{balance}(Accounts))\big)$
   B. $\sigma_{cname="John" \wedge city="Chicago"}(Customers) = \sigma_{cname="John"}\big(\sigma_{city="Chicago"}Customers\big)$
   C. $\pi_{bname}\left(\pi_{bname,city}(Accounts)\right) = \pi_{bname}(Accounts)$
   D. $\sigma_{city="Milwaukee"}\left(\pi_{city,bid}(Branches) \bowtie \pi_{balance,bid}(Accounts)\right) = \pi_{city,bid,balance}(\sigma_{city="Milwaukee"}(Branches) \bowtie Accounts)$

10. Using the information in Question 9, we further assume that there are 25,000 records of Customers stored in 625 pages, *V*(job, Customers)=50, *V*(age, Customers)=10 and *V*(city, Customers) = 5. We also assume that he records are uniformly distributed and the attributes are independent with each other.

    SELECT cname
    FROM Customers
    WHERE job = “Banker” AND age = 30 AND city = “Boston”

    What is the expected number of records in the result of the above SQL query?

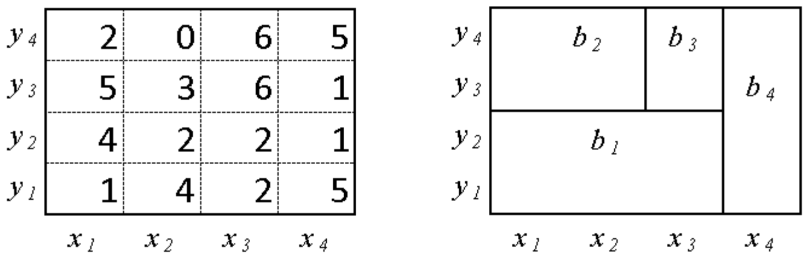
    A. 25
    B. 10
    C. 500
    D. 5000

11. Which of the following statements are **true** for relational query processing?

    I. In retrieving a relation, disk accesses are always based on pages or blocks but not on tuples.
    II. According to heuristic optimization, we should always push the selection and projection operations down as low as possible in a query plan tree.
    III. According to pipelining techniques, we should not use sort-merge join to obtain the intermediate join result in a query plan tree.

    A. I and II Only.
    B. I and III Only.
    C. II and III Only.
    D. All of them.

12. Consider the multi-dimensional histogram example in the following figure. The left diagram represents the attribute dependence between attributes X = {x_1, x_2, x_3, x_4} and Y = {y_1, y_2, y_3, y_4}, where X and Y use non-zero integer domains. The right diagram represents the collection of the records in a set of buckets B = {b_1, b_2, b_3, b_4}. The number of records in each bucket may be different, which is the information available for query size estimation. For example, b_1 has 15 records and b_2 has 10 records. Which of the following statements about the estimation of the number of records in the query window "where ($y_3 \leq y$ and $x = x_1$) or ($y \leq y_2$ and $x_3 \leq x$)" is **the best one**?



    A. 16
    B. 18
    C. 20
    D. 22

13. Consider the following schedules S1 and S2 with usual notation:

    S1: <R1(X), R3(Y), R2(Y), R3(X), R1(Z), R2(Z), W3(Y), W1(X), W2(Z), W1(Z)>
    S2: <R1(X), R3(Y), R3(X), R2(Y), R2(Z), W3(Y), W2(Z), R1(Z), W1(X), W1(Z)>

    Which one of the following statements about S1 and S2 is **true**?

    A. Only S1 is conflict-serializable.
    B. Only S2 is conflict-serializable.
    C. Both S1 and S2 are conflict-serializable.

D. Neither S1 nor S2 is conflict-serializable.

14. Which of the following scenarios may lead to an *irrecoverable* (i.e. not recoverable after failures) error in a database system?

    A. A transaction writes a data item after it is read by an uncommitted transaction.
    B. A transaction reads a data item after it is read by an uncommitted transaction.
    C. A transaction reads a data item after it is written by a committed transaction.
    D. A transaction reads a data item after it is written by an uncommitted transaction.

15. Which of the following statement(s) about locking protocols is/are correct?

    I. Time-stamp based protocol can ensure both conflict serializable and free from deadlock.
    II. Rigorous 2PL protocol can eliminate the deadlock problem.
    III. If a schedule is executed by strict 2PL protocol, it must be cascadeless.

    A. I and II only.
    B. I and III only.
    C. II and III only.
    D. All of them.

16. Assume a database system uses checkpoint protocol and immediate database modification log scheme. Consider the following set of operations in the log obtained when a crash happens:

```
<T3 start>
<T3, Y, 2, 3>
<T1 start>
<T3 commit>
<T1, Z, 5, 7>
<checkpoint>
<T2 start>
<T2, X, 1, 9>
<T2 commit>
<T4 start>
<T4, Z, 7, 2>
```

What are the contents of the undo list and the redo list if the system tries to recover the database?

A. Undo: T4, T1; Redo: T2
B. Undo: T4, T1; Redo: T2, T3
C. Undo: none; Redo: T2, T3, T4, T1
D. Undo: T4; Redo: T2

17. The result of the following Oracle SQL SELECT statement is ________ .

```
SELECT A, B
FROM R
ORDER BY A DESC, B;
```

Table R:

A	**B**
1	4
3	3
NULL	6
1	1

A.

A	B
	6
3	3
1	1
1	4

B.

A	B
	1
3	3
1	4
1	6

C.

A	B
3	3
1	1
1	4
	6

D.

A	B
3	1
1	3
1	4
	6

18. Consider the following tables **STUDENTS** and **DEPARTMENTS** in a database.

STUDENTS

Name	**Dept**
Cony	ACCT
Brown	ECON
Sally	NULL

DEPARTMENTS

Dept	**Head**

The result of the following SQL SELECT statement is ________ .

```
SELECT COUNT(*)
FROM STUDENTS, DEPARTMENTS;
```

A. 0
B. 1
C. 2
D. 3

19. Consider the following tables **STUDENTS** and **FRIENDS** in a database.

STUDENTS

Name	Dept	CGA
Cony	ACCT	3
Brown	ECON	2
Sally	CHEM	3
Jessica	ELEC	4
Moon	ELEC	2
Choco	CHEM	3

FRIENDS

Name1	Name2
Cony	Jessica
Brown	Choco
Sally	Moon
Jessica	Moon

We obtain three results (Result1 to Result3) from the following SQL statements:

Result1:
```
SELECT MAX(CGA) AS RESULT FROM STUDENTS
WHERE NAME IN
(
(SELECT NAME FROM STUDENTS)
MINUS
(SELECT NAME2 FROM FRIENDS WHERE NAME1 = 'CONY')
)
GROUP BY DEPT
HAVING MIN(CGA) >= 3;
```

Result2:
```
SELECT MAX(CGA) AS RESULT FROM STUDENTS
WHERE CGA < (SELECT MAX(CGA) FROM STUDENTS);
```

Result3:
```
SELECT AVG(CGA) AS RESULT FROM STUDENTS, FRIENDS
WHERE NAME = NAME1;
```

Which of the following statements about Result1, Result2 and Result3 is **true**?

A. The three results are all different.
B. The three results are all the same.
C. Only Result 2 and Result 3 are the same, but Result 1 is different from them.
D. None of the above.

20. For which of the following set of dependencies is relation R(*A*,*B*,*C*,*D*) in 3NF but *not* in BCNF?

A. $\{AB \longrightarrow CD, C \longrightarrow DA\}$
B. $\{AB \longrightarrow CD, A \longrightarrow C, D \longrightarrow B\}$
C. $\{AB \longrightarrow CD, C \longrightarrow A, D \longrightarrow B\}$
D. $\{A \longrightarrow BCD, B \longrightarrow CD, C \longrightarrow D\}$

Q2 Indexing and Hashing Techniques (16 marks)

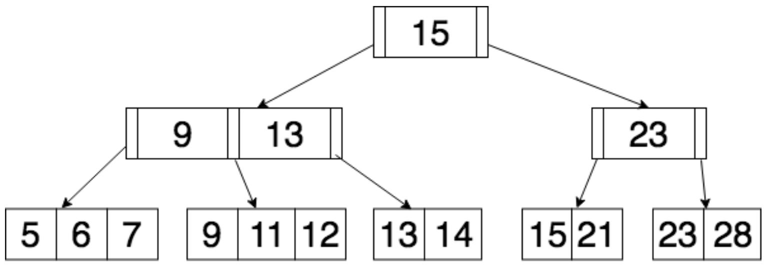
(a) (**8 marks**) You are given the following B+-tree with *fanout n*=4. The tree is based on the simplified notation used in the course.

i. Draw clearly the B+-tree for each of the following insertion operations: first insert 10 and then insert 8.

ii. Draw clearly the B+-trees for each of the following deletion operations: first delete 13 and then delete 15.

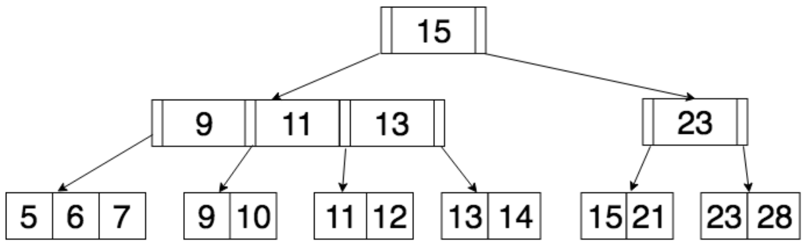
Both insertions and deletions are performed in the described orders. ***Both parts should begin with the initial B+-tree given below.***

(i) Initial B+-tree:



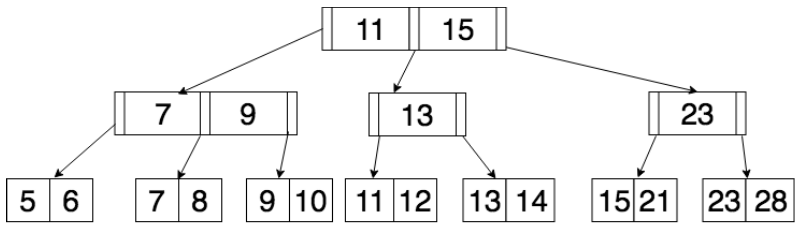
Insert 10:

Answer:

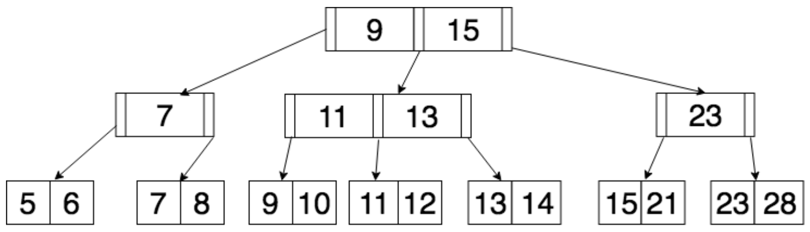


Insert 8:

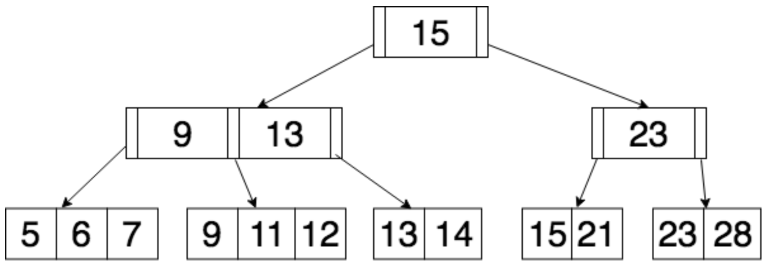
Answer:



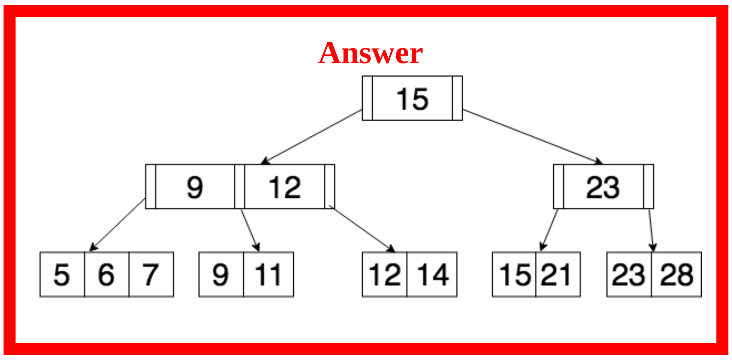
OR



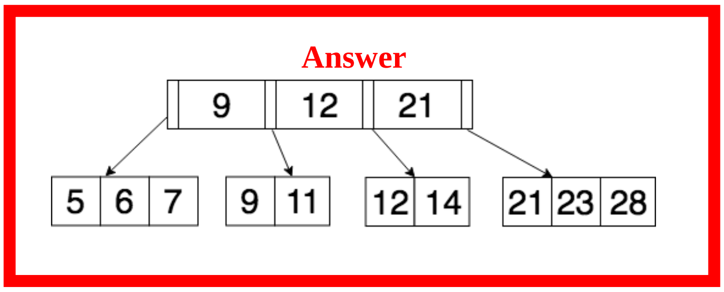
(ii) Initial B+-tree:



Delete 13:



Delete 15:



(b) **(8 marks)** You are given an initial hash structure with the following five search keys: 6, 9, 12, 13, 15 already inserted as below. The hash function is defined as **h(*x*)** = *x* mod 16 for a given key values *x*, and each bucket can hold only **two keys**.

i. Draw the extendable hash structure for each insertion of the following search keys: 5, 14 (first 5, then 14).

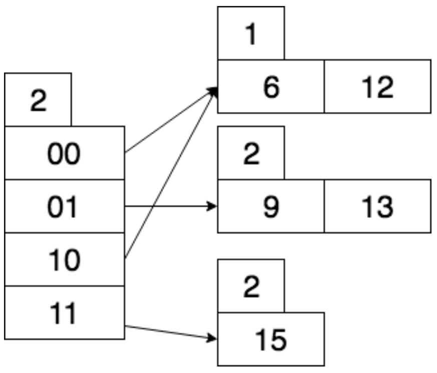
ii. Draw the extendable hash structure for each deletion of the search keys: 6, 15 (first 6, then 15). You should follow the convention used by lectures, in particular binary hash indices starting from **the least significant bit**.

Both insertions and deletions are performed in the described orders. ***Both parts should begin with the initial hash table structure.***

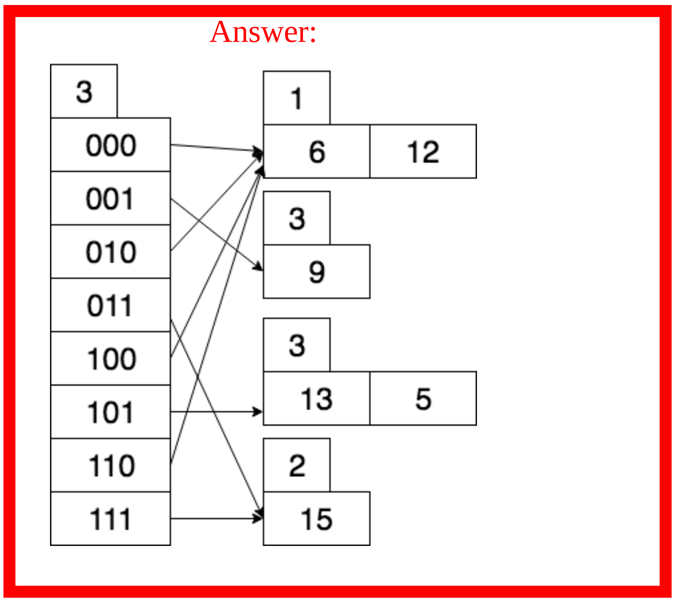
Hash values of the keys:

5	6	9	12	13	14	15
0101	0110	1001	1100	1101	1110	1111

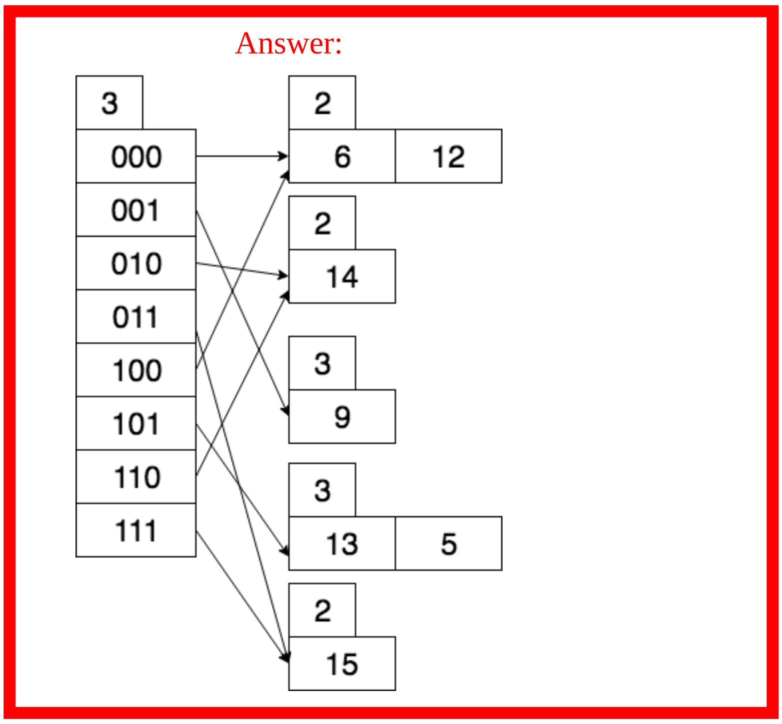
(i) Initial hash table structure:



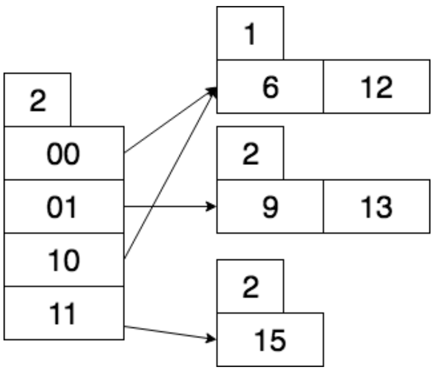
Insert 5:



Insert 14:

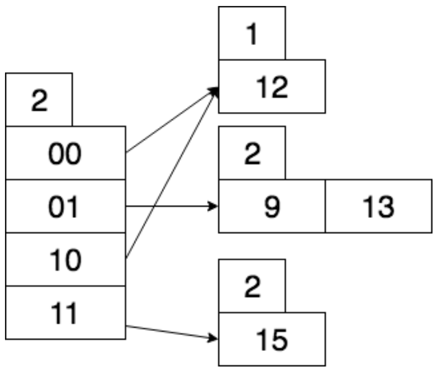


(ii) Initial hash table structure:



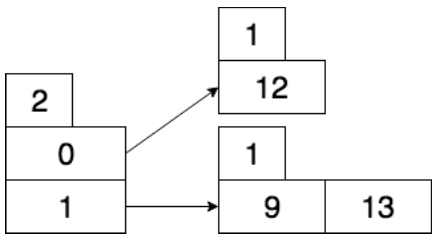
Delete 6:

Answer:



Delete 15:

Answer:



Q3 Join Operation (16 marks)

Consider the following two relational tables:

- Sailors (Sid, Name, Rating, Age)
- Reserves (*Sid*, Bid, Date)

Each attribute (and pointer whenever applicable) is 20 bytes. Each page is 1200 bytes. There are 15,000 sailors, 60,000 reservations and 100 pages memory.

Consider also the following query in SQL involving the join of Sailors and Reserves:

```
SELECT Name, Bid
FROM Sailors, Reserves
WHERE Sailors.Sid = Reserves.Sid;
```

For reference Only: Since there are 15K sailors and 60K reservations, a sailor has on the average 4 reservations (but it is possible that some sailors have more, while others have none)
Each Sailors record is 80 bytes and each Reserves record 60 bytes. There are 1200/80=15 sailors per page and 1200/60=20 reservations per page. Therefore, Sailors contains 15000/15=**1K** pages and Reserves 60000/20=**3K** pages.

(i) **(5 marks)** Fill in appropriate estimated values in the space:

- We need 1000 pages to store Sailors and __3000___ pages to store Reserves.
- On average there are __4___ reservations made by a sailor.
- The minimal memory space needed to run this join is _3__ pages.
- If we adopt block nested-loop join, the minimal memory space to achieve the most efficient join is __1002__ pages.
- The minimal cost using nested-loop join is __34000__ pages.

For reference Only: Case 1 needs to get the bucket page with index searching cost 1.5 and there is 1 bucket to hold 4 pointers that going to the data pages (on average 4 reservations per each sailor). Case 2 the same index searching cost and 1 data page access (on average 0.25 sailors per each reservation but each page access by a hash index should take at least 1 page).

(ii) **(5 marks)** Fill in appropriate estimated values in the space:

- Assume that we adopt indexed nested-loop join, and we have hash indexes on the join attribute of *both* relations. We also assume finding an entry in the hash index takes 1.5 page due to overflow buckets.
  Write down the key expression and the final answer to compute the total cost for the two cases:
  Case 1: Using Sailors as the external relation (i.e. hash indexing on Reserves.Sid)
  Expression: ______1000 + 15000*(1.5+1+4)__________ Total Cost:___ 98500____pages.
  Case 2: Using Reserves as the external relation (i.e. hash indexing on Sailors.Sid)
  Expression: ______3000+ 60000*(1.5+ 1)__________ Total Cost:___ 153000____pages.
- If we now use a clustered B+ index on Reserves.Sid and the number of the tree levels is 4, the total cost of adopting indexed nested-loop join is at least __76000__ pages.

(iii) **(6 marks)** Fill in appropriate estimated values in the space:

If we adopt *hybrid hash join* as follows:

Step1 We first partition Sailors to 50 buckets and remove unnecessary attributes of Sailors as many as possible. We then keep *as many as possible* the first *x* buckets in the memory and write other buckets back to the disk.

Step 2 Then we partition Reserves. Each record that belongs to the first *x* buckets is matched directly with the corresponding bucket of Sailors (which is still in memory). Before writing other Reserves buckets back to the disk, we first remove unnecessary attributes of Reserves as many as possible.

Step 3 Finally we read the remaining bucket of Sailors (as the built input) and match it against the corresponding bucket of Reserves (as the probe input).

- The bucket size for Sailor is 10 pages and the bucket size for Reserves is ___40__ pages.
- The value of *x* is __6__.
- After keeping the first *x* buckets in Step 1, we still have ___(100- 60 =)40____ pages in memory for partitioning Reserves in Step 2.
- The total cost of running the partition of Sailors in Step 1 is (1000+500-60 =)1440_pages.
- The total cost of running the partition of Reserves in Step 2 is (3000+2000-240 =)4760 pages.
- The total cost of running all the steps is __(1440+4760+2200 =) 8400__ pages.

For reference only: Only Sid and Name are kept in Sailors and so Sailors size is reduced to 500 pages when the partitioned relation is stored. So 1 bucket has size 500/50 = 10 and therefore 6 buckets of Sailors in the memory need 6x10=60 pages and the remaining 100-60=40 pages are sufficient to obtain 1 page of Reserves and allocate 34 bucket pages. Other 6 bucket pages are matched directly with the residing 6 Sailor buckets within the memory. Only Sid and Bid are kept in Reserves and so Reserves size is reduced to 2000 pages when the partitioned relation is stored. So 1 bucket has size 2000/50 = 40. The remaining 44 buckets of Reserves are stored back in disk.

Step1: 1000+500-60 = 1440
Step2: 3000+2000-240 =4760
Step3: 1440+4760+2200 =8400

Q4 Query Optimiazation (10 marks)

Consider the following relational schemas and the statistical data, where primary keys are underlined and foreign keys in italics.

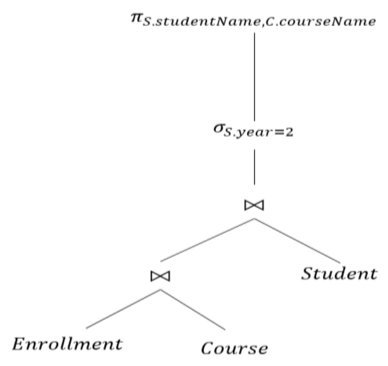
Student(<u>sid</u>, studentName, department, year)
Course(<u>courseCode</u>, courseName, description)
Enrollment(<u>*sid*,*courseCode*</u>)

- Number of tuples of Student: 15,000.
- Number of tuples of Course: 500.
- Number of tuples of Enrollment: 45,000.
- Number of tuples of Student fitting into a page: 30.
- Number of tuples of Course fitting into a page: 50.
- Number of tuples of Enrollment fitting into a page: 500.
- There are 5 different years for the year attribute of Student.

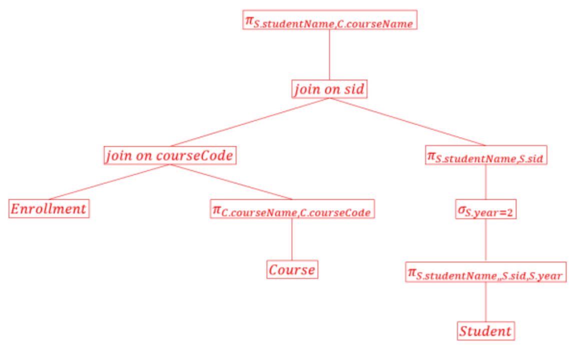
Consider the following SQL query:

```
SELECT studentName, courseName
FROM Student S, Course C, Enrollment E
WHERE E.sid = S.sid AND E.courseCode = C.courseCode AND S.year = 2
```

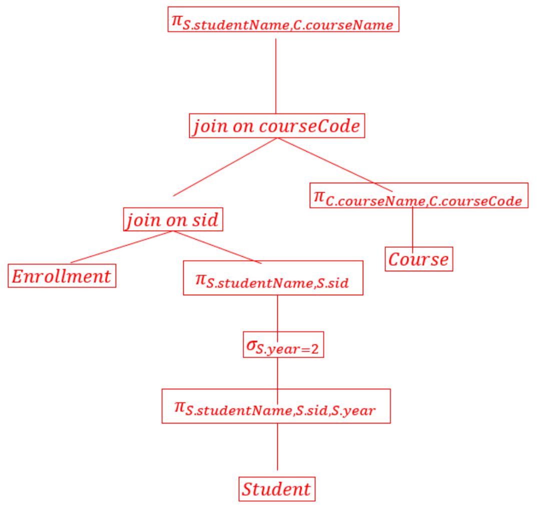
The following query tree plan is firstly generated for the execution of the above SQL query.



(a) **(5 marks)** The naïve query tree is clearly not the ideal plan. Use the heuristics discussed in the course to transform the query tree into a more efficient one as much as possible. (You should indicate clearly the content of the nodes but no algorithms or indexes of the operators are needed.)



$\pi_{S.studentName,C.courseName}$
join on sid
join on courseCode
$\pi_{S.studentName,S.sid}$
Enrollment
$\pi_{C.courseName,C.courseCode}$
$\sigma_{S.year=2}$
Course
$\pi_{S.studentName,,S.sid,S.year}$
Student



$\pi_{S.studentName,C.courseName}$
join on courseCode
join on sid
$\pi_{C.courseName,C.courseCode}$
Enrollment
$\pi_{S.studentName,S.sid}$
Course
$\sigma_{S.year=2}$
$\pi_{S.studentName,S.sid,S.year}$
Student

(b) **(5 marks)** Assume the schemas of four relations given as follows: R(A,B), S(B,C), T(C,D), U(D,A) and all relations have 3000 records having the same size. We want to find the join order of joining R, S, T and U, where only *natural join* is considered. Suppose our estimation is based on the following statistics, where *V*(*x*,*y*) be the number of distinct values that appear in the relation *y* for attribute *x*.

R(A,B)	S(B,C)	T(C,D)	U(D,A)
V(A,R)=200	*V(B,S)=200*	*V(C,T)=100*	*V(A,U)=100*
V(B,R)=600	*V(C,S)=1000*	*V(D,T)=300*	*V(D,U)=3000*

The following are the tables representing the progressive finding of best plan in dynamic programming. The cost here refers to the storage cost of writing in disk the intermediate result generated in a join combination. You are required to fill in the missing details (i.e. all empty cells in the tables).

i. Plan for grouping two relations for the join.

Combinations	R,S	R,T	R,U	S,T	S,U	T,U
Size (in recs)	15,000	9M	45,000	9,000	9M	3,000
Cost (in recs)	0	0	0	0	0	0
Best Plan (Trivial)	RJoinS	RJoinT	RJoinU	SJoinT	SJoinU	TJoinU

ii. Plan for grouping three relations for the join.

Combinations	R,S,T	R,S,U	R,T,U	S,T,U
Size (in recs)	45,000	225,000	45,000	9,000
Cost (in recs)	9,000	15,000	3,000	3,000
Best Plan	(SJoinT)JoinR	(RJoinS)JoinU	(TJoinU)JoinR	(TJoinU)JoinS

iii. What is the best plan for the order of joining the relations? Justify your answer based on Parts (b)(i) and (b)(ii). Fill in the following table and state your decision.

Grouping	Cost (in recs)
((SJoinT)JoinR)JoinU	54,000
((RJoinS)JoinU)JoinT	240,000
((TJoinU)JoinR)JoinS	48,000
((TJoinU)JoinS)JoinR	12,000 (Best Plan)
(RJoinS)Join(TJoinU)	18,000
(RJoinT)Join(SJoinU)	18M
(RJoinU)Join(SJoinT)	54,000

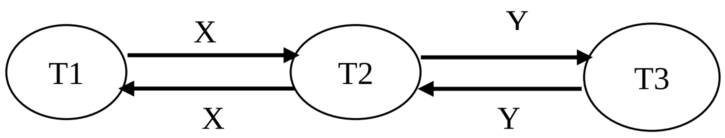
Q5 Transaction Management and Recovery Management (14 marks)

(a) **(3 marks)** Consider we have schedule S1 for transaction T1, T2 and T3:

S1: <R1(X), R2(X), W2(X), R3(Y), R2(Y), W3(Y), W2(Y), W1(X)>

Draw the precedence graph (with labeled arcs) and decide if the schedule is conflict-serializable. If it is conflict-serializable, give an equivalent serial transaction schedule. If not, explain the conflict in details (e.g. the transactions having conflicting pairs).

ANS: (1.5+1.5=3 points)



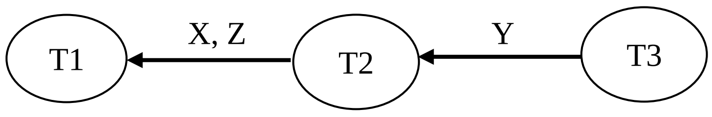
This is not conflict-serializable.
T1 and T2 have RW conflicts in both directions on X and T2 and T3 have RW conflicts on Y.
There are cycles (e.g T1 → T2 → T1) in the graph so it is not conflict-serializable.

(b) **(3 marks)** Consider we have schedule S2 for transaction T1, T2 and T3:

S2: <R1(X), R2(X), R3(Y), W1(X), R2(Z), R2(Y), W2(Y), W1(Z)>

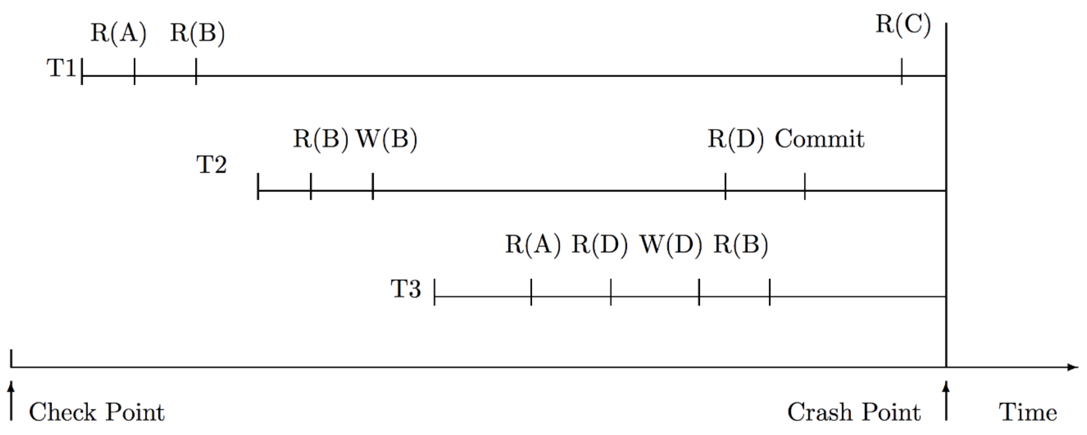
Draw the precedence graph (with labeled arcs) and decide if the schedule is conflict-serializable. If it is conflict-serializable, give an equivalent serial transaction schedule. If not, explain the conflict in details (e.g. the transactions and conflicting pairs).

ANS:(1.5+1.5=3points)



This is conflict-serializable since there are no cycles.
Equivalent serial schedule is <T3, T2, T1>.

(c) **(4 marks)** Consider the following log information that involves three transactions. Assume that the immediate update protocol with checkpoint is used for crash recovery.



Is this schedule having T1 to T3 recoverable or cascadeless? Explain why or why not in detail (e.g. stating which data items and transactions are problematic)?

The schedule is not recoverable because T2 reads data D whose value was written by T3, but T2 commits before T3 commits.

It is not cascadeless, since it is not recoverable. OR showing the cascading rollback of T2 and T3.

ANS: (2+1+1=4 points)

(d) **(4 marks)** Complete following schedule with **strict 2PL protocol**. Empty rows are for different choices of locking and unlocking instructions. You are required to use Lock-X(D) for exclusive locking on an item D and Lock-S(D) for shared locking on an item D. (Note that not all rows should be filled. You could leave some rows empty if appropriate.)

Strict 2PL		
T1	T2	T3
		Lock-X(C)
		Read(C)
		Write(C)
	Lock-S(A)	
	Read(A)	
Lock-X(B)		
Read(B)		
Write(B)		
		Commit
		Unlock(C)
Lock-S(C)		
Read(C)		
Unlock(C)		
Commit		
Unlock(B)		
	Lock-S(B)	
	Read(B)	
	Unlock(A)	
	Unlock(B)	
	Commit	

(4 points)
ANS are RED, remember to remove them.

Q6 SQL and PL/SQL (14 marks)

The following database contains three relational tables having attributes with usual interpretation.

- Student(<u>sid</u>, name(Not NULL), department, GPA, admission_date)
- Course(<u>cid</u>, department, instructor)
- Enrollment(*<u>sid</u>*, *<u>cid</u>*)

Where

- Primary keys are underlined and foreign keys are in italics.
- In **Enrollment** table foreign key sid refers to sid in **Student** table, and cid refers to cid in **Course** table.
- The datatype of sid , cid and GPA are **number(10)**; the datatype of admission_date is **date**, while other attributes in all tables are **varchar(100).**
- Different students could have the exact same name.
- Assume the Oracle environment is identical to the lab environment of this course.

(a) **(2 marks)** Write in one SQL statement that selects the names of students with a GPA strictly higher than the average GPA of his/her department and the student id (sid) of the student is an odd number.

You are only allowed to write ONE SQL statement. You are NOT allowed to create views, or use the keyword "with". Note that wrong or ambiguous syntax may lead to zero mark.

```
select name from Student
where mod(sid, 2) = 1
      and
      GPA > (select avggpa
                    from
                    (select avg(GPA) as avggpa , department
                     from Student group by department)temp
                     where Student.department = temp.department);
```

(b) (3 marks) Write in one SQL statement that returns the names of all students who have enrolled some courses from ‘ECE’ department and have enrolled all courses instructed by ‘Sheldon Copper’, and is admitted (admission_date) before December the 7th year 2017. The result should not contain duplicates but does not need to be ordered.

You are only allowed to write ONE SQL statement. You are NOT allowed to create views, or use the keyword “with”. Note that wrong or ambiguous syntax may lead to zero mark.

```
select distinct name
from Student S, Course C, Enrollment E
where S.sid = E.sid and C.cid = E.cid and admission_date < to_date('07-DEC-17', 'DD-
MM-YY') and C.department = 'ECE' and NOT EXISTS
(
select cid from Course where instructor = 'Sheldon Copper'
MINUS
select E2.cid from Enrollment E2 where E2.sid = S.sid
);
```

Student Number: ___________________

(c) **(4 marks)** Complete the following PL/SQL script to check all the student ids (sid) of the students to make sure they are prime numbers, if a student id (sid) is not a *prime number* then:

- Throw a user defined EXCEPTION named 'non_prime_id_exception', and after catching the exception, do the following tasks:
    - i. Print out student id (sid) and name on the SQL/PLUS console.
    - ii. Insert student id (sid) into table **ErrorRecord**(<u>sid</u>), where sid is of number(10) data type, the primary key of the table and a foreign key with reference to Student.sid. (You need to create **ErrorRecord** table in the script, you may presume that **ErrorRecord** table does not exist in the database before you run your script.)

Otherwise (i.e. if sid is a prime number):

- Update his/her GPA by taking square root of the original GPA.

A *prime number* is a natural number greater than 1 that has no positive divisors other than 1 and itself. (Hint: Follow the instructions in the comments quoted by /* */.)

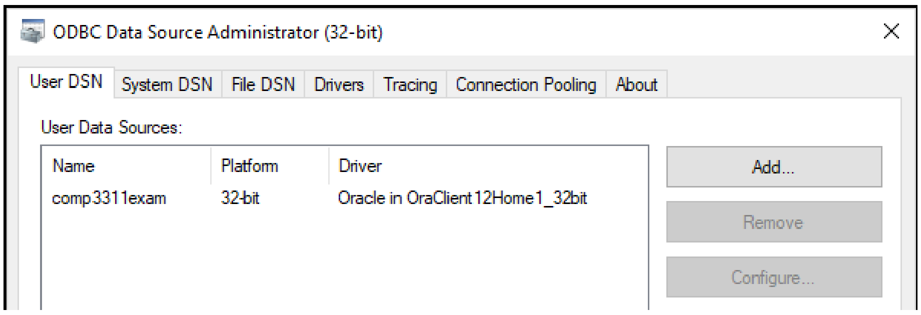
```
/*Create ErrorRecord table*/
CREATE table ErrorRecord(
   sid number(10) references Student(sid),
   primary key(sid));

DECLARE
         i number := 0;
         non_prime_id_exception EXCEPTION;
         CURSOR stu_cursor
/*Complete the cursor definition*/
  IS SELECT sid, GPA from Student;
BEGIN
FOR stu in stu_cursor
         LOOP
/*Initialization*/
                  i := 2;
                  BEGIN
/*Check if the student id (sid) is a prime number, throw an Exception if not.*/
                  LOOP
```

```
                    IF (mod(stu.sid, i)=0) THEN
                        RAISE non_prime_id_exception;
                    END IF;
                    i := i + 1;
                    EXIT WHEN i>power(stu.sid,0.5);
                END LOOP;
/*If the student id (sid) is a prime number, update GPA*/
                Update Student SET GPA=power(stu.GPA, 0.5) where sid = stu.sid;
        EXCEPTION
/*Handle the Exception*/
                    WHEN non_prime_id_exception THEN
                    DBMS_OUTPUT.PUT_LINE(TO_CHAR(stu.sid));
                    INSERT INTO ErrorRecord VALUES(stu.sid);
                END;
        END LOOP;
END;
/
```

(d) **(5 marks)** Complete the following C++ code to meet the following requirements:

- The ODBC driver is configured as follows, with TNS Service Name as "comp3311.cse.ust.hk" and you did not specify your user name for Oracle connection:



ODBC Data Source Administrator (32-bit)

User DSN | System DSN | File DSN | Drivers | Tracing | Connection Pooling | About

User Data Sources:

Name	Platform	Driver
comp3311exam	32-bit	Oracle in OraClient12Home1_32bit

- Connect to Oracle Database (Same environment as the 4210 Lab)
- Assume your oracle account user name is "COMP3311AC", password is "1234".
- Read a user input student id (sid) from the console, while running the program.
- Retrieve the corresponding GPA of the input sid.
- Display the retrieved GPA on the console in format like "GPA=3.5".

```
#include "stdafx.h"    #include <windows.h>      #include <sql.h>
#include <sqlext.h>     #include <sqltypes.h>       #include <iostream>
using namespace std;
void main()
{
        HENV   henv;
        HDBC   hdbc;
        HSTMT  hstmt;
        SQLINTEGER sqlcode, gpa, gpa_n, sid, sid_n;
        SQLAllocEnv(&henv);
        SQLAllocConnect(henv, &hdbc);
        cin >> sid;
```

```
/*Fill in the missing parameters*/
SQLConnectA(hdbc, (SQLCHAR*) "comp3311exam"  __________ ,

                SQL_NTS, (SQLCHAR*) "comp3311AC"_________,

                SQL_NTS, (SQLCHAR*) "1234" _______________,
                SQL_NTS);
```

```
/*Fill in the missing parameters*/
SQLAllocStmt(hdbc, &hstmt);
```

```
/*Fill in the missing parameters*/
SQLPrepareA(hstmt, (SQLCHAR*) "SELECT GPA FROM Student where sid=?",
SQL_NTS);
```

```
/*Fill in the missing parameters*/
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0,
0, &sid, 1, &sid_n);
```

```
/*Fill in the missing parameters*/
SQLExecute(hstmt);
```

```
/*Fill in the missing parameters*/
SQLBindCol(hstmt, 1, SQL_C_SLONG, &gpa, 1, &gpa_n);
```

```
/*Fetch the query result from database and display on the C++ console*/


while (TRUE) {
                RETCODE retcode = SQLFetch(hstmt);
                if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
                {
                        printf("GPA = %d\n", gpa);
                }
                else break;
        }
```

```
        SQLFreeStmt(hstmt,SQL_CLOSE);
        SQLDisconnect(hdbc);
        SQLFreeConnect(hdbc);
        SQLFreeEnv(henv);
}
```

- End of the Exam Paper-