CS411 Database Systems

Fall 2013, Prof. Chang

Department of Computer Science
University of Illinois at Urbana-Champaign

Final Examination

Dec 17, 2013
Time Limit: 180 minutes

- Print your name and NetID below. In addition, print your NetID in the upper right corner of every page.

  Name: ______________________ **NetID:** ______________________

- Including this cover page, this exam booklet contains **18** pages. Check if you have missing pages.
- The exam is closed book and closed notes. You are allowed to use scratch papers and calculators. No other electronic devices are permitted. Any form of cheating on the examination will result in a zero grade.
- Please write your solutions in the spaces provided on the exam. You may use the blank areas and backs of the exam pages for scratch work.
- Please make your answers clear and succinct; you will lose credit for verbose, convoluted, or confusing answers. *Simplicity does count!*
- Each problem has different weight, as listed below– So, plan your time accordingly. *You should look through the entire exam before getting started, to plan your strategy.*

Problem	1	2	3	4	5	6	7	Total
Points	36	14	8	8	10	10	14	100
Score								
Grader								

Problem 1 (*36 points*) Misc. Concepts

For each of the following statements:

- for true/false choices, indicate whether it is *TRUE* or *FALSE* by **circling** your choice, and provide an **explanation** to justify;
- for short answer questions, provide a brief **answer** with clear **explanation**.

You will get *2 points* for each correct answer with correct explanations, and ***no penalty* (of negative points) for wrong answers**.

(1) Answer: *True* *False*

A schema in BCNF has the advantage to speed up query processing.

⇒ *Explain*: False: BCNF decomposition can split tables, which require join, hence can be disadvantageous to query processing.

(2) Answer: *True* *False*

In SQL, for set operators (*e.g.*, INTERSECT), the default semantics is to eliminate duplicates– because, for such operations, the query processor needs to sort and compare tuples anyway.

⇒ *Explain*: True: In order to not eliminate duplicates need to specify "ALL". Most set operations can be efficiently done by sorting. Hence query processor sorts them before set operations.

(3) Answer: *True* *False*

Query processing can take the form of "multipass" algorithms– among them, two passes are usually enough, even for very large relations.

⇒ *Explain*: True: With 1 GB of memory, we can sort data sizes upto 4 TB using TPMMS

(4) Answer: *True* *False*

In query processing, sorting can help in many operations, such as *duplicate elimination* and *joins*.

⇒ *Explain*: True: Sorting helps to quickly identify the tuples required to perform join. It also help in duplicate elimination since duplicate tuples will be together after a sort.

(5) Failure recovery posts some restrictions to buffer/memory management. Consider *REDO logging*– give one such restriction.

⇒ *Answer*: REDO does not allow flush dirty data blogs before a transaction is committed.

(6) Answer: $\underline{True}$ $\underline{False}$

Edgar Codd envisioned the concept of transaction processing when he proposed the relational model.

⇒ *Explain*: False: Edgar Codd does not discuss transactions in the proposed relational model.

(7) Answer: $\underline{True}$ $\underline{False}$

Rule-based query optimization was popular in DBMS in the early days– *e.g.*, the early versions of Oracle used only heuristic rules to optimize queries.

⇒ *Explain*: True: Oracle only fully turned to cost-based optimization after 2000.

(8) Answer: $\underline{True}$ $\underline{False}$

In the ACID properties, I stands for *Idempotence*, which means a transaction can be executed multiple times without changing the result beyond the initial execution.

⇒ *Explain*: False: I stands for Isolation

(9) Answer: $\underline{True}$ $\underline{False}$

Secondary indexes are always dense.

⇒ *Explain*: True: Since a secondary index is not clustered, so each tuple needs its own address pointer, and thus the index must be dense.

(10) A B-tree organizes its blocks into a tree that is *balanced*. Why is *balanced* a good property to have?

⇒ *Answer*: A balanced tree has the same height (number of internal nodes) to reach a leaf node to get data, and thus guarantees that search time is uniform, consistent and predictable.

(11) Can you use *Map-Reduce* to process $R \bowtie_{R.a=S.b} S$? If so, explain how. If not, explain why.

⇒ *Answer*: Yes. The map function takes one tuple from table R or table S as input, and outputs a pair with $R.a$ or $S.b$ as the key and the tuple as the value; the reduce function takes a list of tuples (from R or S) which share the same $R.a$ and $S.b$ as input, and outputs the joined results.

(12) For the following query over relations *Students*(sid, name, department), *Enrollment*(sid, cid, semester), and *Courses*(cid, title, instructor), design an efficient query plan for it– draw the query plan in a query execution tree, and explain why it would be efficient.

SELECT S.name FROM *Students* S, *Enrollment* E, *Courses* C
WHERE C.cid = "CS411" and E.semester = "Summer 2014"
and S.sid = E.sid and E.cid = C.cid

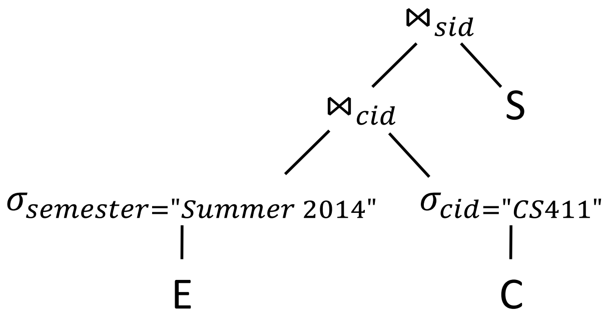
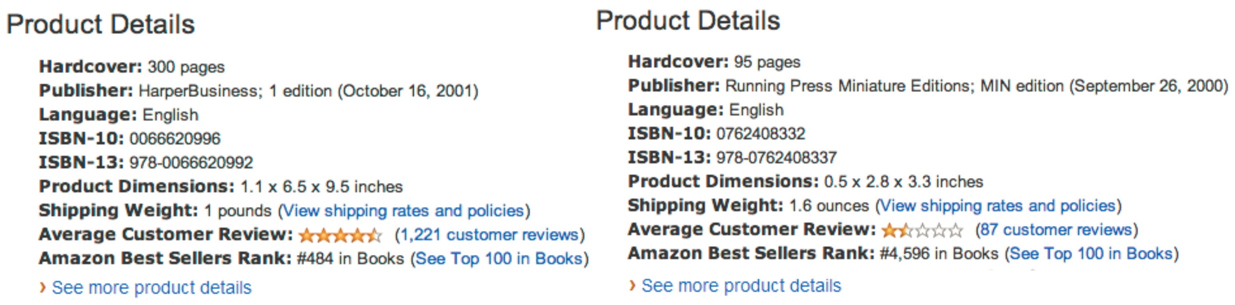


Figure 1: Query Execution Tree

⇒ *Answer*: Doing the tuple selection first can reduce the table size, which helps improve the efficiency of the join operation.

(13) Answer: *True* *False*

Consider a web page that displays book records like the following. To extract such records, it is suitable to use an HRLT wrapper technique.



Product Details

Hardcover: 300 pages
Publisher: HarperBusiness; 1 edition (October 16, 2001)
Language: English
ISBN-10: 0066620996
ISBN-13: 978-0066620992
Product Dimensions: 1.1 x 6.5 x 9.5 inches
Shipping Weight: 1 pounds (View shipping rates and policies)
Average Customer Review: ★★★★☆ (1,221 customer reviews)
Amazon Best Sellers Rank: #484 in Books (See Top 100 in Books)
› See more product details

Product Details

Hardcover: 95 pages
Publisher: Running Press Miniature Editions; MIN edition (September 26, 2000)
Language: English
ISBN-10: 0762408332
ISBN-13: 978-0762408337
Product Dimensions: 0.5 x 2.8 x 3.3 inches
Shipping Weight: 1.6 ounces (View shipping rates and policies)
Average Customer Review: ★★☆☆☆ (87 customer reviews)
Amazon Best Sellers Rank: #4,596 in Books (See Top 100 in Books)
› See more product details

Figure 2: Two *book* records from a web site.

⇒ *Explain*: True. HRLT wrapper could be applied because it contains a header part ("Product Details"), a tailer part ("See more product details") and a set of attribute-value pairs.

(14) Suppose that we use RoadRunner to induce rules for extracting the records in Fig. 2. Let's consider only these two records, and only the "Number of Pages" attribute (which is labeled

as "Hardcover" above). The HTML code for the first record is shown as follows. What do you think will be the pattern induced for this attribute?

```
... <li><b>Hardcover:</b><i>300</i> pages</li> ...
```

⇒ *Answer*: ... <li><b>Hardcover:</b><i>#PCDATA</i> pages</li> ...

(15) In a *Linear Chain CRF*, what variables would the value of y_i (*i.e.*, the label of x_i, where i is the position index of x_i in the input sequence) depend on?

⇒ *Answer*: In Linear Chain CRF, the value of a label not only depends on its word, but also its surrounding word labels. We accept both 1) x_i, x_{i-1}, x_{i+1}, y_{i-1}, y_{i+1} (corresponding to the graphical representation of Linear Chain CRF) and 2) x_i, y_{i-1}, y_{i+1} (corresponding to the factor design Linear Chain CRF) as correct answers.

(16) Answer: *True* *False*

In the redo scheme, when a data block contains data elements from different transactions, at a checkpoint, because these transactions may have different statuses (committed or active), they will impose contradictory requirements on how the block should be handled in terms of buffer management.

⇒ *Explain*: True. Committed transactions should be written to disk, and active transactions should be kept in the buffer.

(17) Answer: *True* *False*

NoSQL represents a category of non-relational databases which support very simple data operations over big data.

⇒ *Explain*: False. NoSQL represents a category of non-relational databases which aims at different balancing points between data size and data complexity.

(18) Answer: *True* *False*

As one important advantage of MongoDB, it supports much faster table joining operation compared with traditional RDBMS.

⇒ *Explain*: False. MongoDB does not officially support the joining operation.

Problem 2 (*14 points*) Relational Schema and Query Languages

(1) Consider the following database which stores information about constituencies, political parties, their members and which members compete in different constituencies.

1. Each political party has a unique id, a name and many members.
2. Each party also a head office(only the state is stored) and one member who is the party leader.
3. Each member has a unique member id, name, age, and the id of the party to which he belongs.
4. Each member belongs to exactly one party.
5. Each constituency has a unique cid.
6. Each constituency also has an associated state and city.
7. Each constituency has many members from different parties contesting with each other in it. But each member can only contest from one constituency.

Given the above requirements for the database design, draw the ER Diagram to represent this database. If you feel that you must make some assumptions, please state them clearly so that they are easily understood. Remember to indicate the key for each entity, as well as the multiplicity of each relationship (e.g. one-to-many) using the appropriate notation. (*3 points*)

Solution:

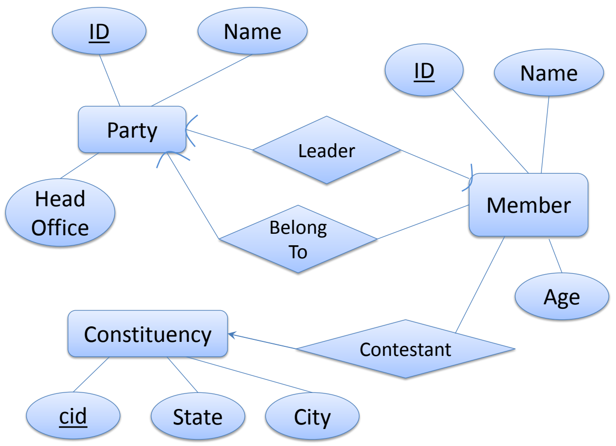


Figure 3: ER Diagram.

Consider a simplified schema design for the above scenario to answer the rest questions:

1. Party (PartyID, PartyName)
2. Member (MemberName, Age, PartyID)

(2) Answer the following queries using expressions of **relational algebra**:

(i) List all the parties that have a member named "John Smith" (can be different persons). (*2 points*)

Solution: $\pi_{\text{PartyName}}(\sigma_{\text{MemberName=``John Smith''}}(\text{Party} \bowtie \text{Member}))$

(ii) Give the name of the youngest person (assume that there is only ONE youngest person). (*3 points*)

Solution:
$\pi_{\text{MemberName}}(\text{Member}) - \pi_{n2}(\sigma_{a1<a2}(\rho_{\text{n1, a1, id1}}(\text{Member}) \times \rho_{\text{n2, a2, id2}}(\text{Member})))$

(3) Write **SQL** queries for each of the following:

(i) Find the names of all members from "XYZ" party who are younger than 40 years. (*2 points*)

Solution:

```
SELECT Member.MemberName
FROM Member, Party
WHERE Member.Age < 40
    AND Party.PartyName="XYZ"
    AND Member.PartyID = Party.PartyID
```

(ii) List the names of the parties whose members are ALL older than 40 years. (*4 points*)

Solution:

```
SELECT PartyName
FROM Party
WHERE NOT EXISTS
    (SELECT *
    FROM Member
    WHERE Member.PartyID = Party.PartyID AND Member.Age <= 40)
```

Problem 3 (*8 points*) XML/JSON and XQuery

Semi-structured data formats such as XML and JSON are frequently used for data exchange in Web services, *e.g.*, Facebook Graph Query. Let's consider the following users.xml file returned by the API of Facebook Graph Query:

```
<?xml version="1.0"?>
<user>
    <name> Jacob Smith </name>
    <user_id> smith1 </user_id>
    <birth_year> 1993 </birth_year>
    <university> UIUC </university>
    <friends>
        <friend_id> johnson10 </friend_id>
        <friend_id> davis2</friend_id>
    </friends>
</user>
<user>
    <name> Sophia Johnson </name>
    <user_id> johnson10 </user_id>
    <birth_year> 1990 </birth_year>
    <university> MIT </university>
    <friends>
        <friend_id> smith1 </friend_id>
    </friends>
</user>
```

(a) Translate the given XML data into the JSON format, which should represent the same information. (*2 points*)

Solution:

```
{
"users":[
    {
    "name": "Jacob Smith", "user_id":"smith1",
    "birth_year": 1993, "university":"UIUC",
    "friend_ids": ["johnson10", "davis2"]
    },
    {
    "name": "Sophia Johnson", "user_id":"johnson10",
    "birth_year": 1990, "university":"MIT",
    "friend_ids": ["smith1"]
    }
]
}
```

Answer the following queries using XQuery. Your answer should work for a larger XML file containing more persons, not just this one. Assume that the above XML data is already stored in the $users variable (by "let $users := doc('users.xml')").

(b) List all the university names, ordered by their number of students in increasing order. The result should take the following format:

```
<university>
    <name> UIUC </name>
    <student_num> 1 </student_num>
</university> ...
```

(*3 points*)

Solution:

```
for $u in $users/user
let $v := $u/university
group by $v
order by count($u)
return <university>
            <name> {$v} </name>
            <student_num> {count($u)} </student_num>
        </university>
```

(c) List all the pairs of *friends* who are in the same university and of the same age. The result should take the format of:

```
<friends>
    <name>Jacob Smith</name>
    <name>Daniel Davis</name>
</friends> ...
```

(*3 points*)

Solution:

```
for $u0 in $users/user
for $f in $u/friends/friend_id
for $u1 in $users/user
where $u0/university = $u1/university and $u0/birth_year = $u1/birth_year
        and $u1/user_id = $f and $u0/user_id < $u1/user_id
return <friends>
            <name> {$u0/name} </name>
            <name> {$u1/name} </name>
        </friends>
```

Problem 4 (*8 points*) Indexing

(a) Consider the following B+ Tree of order 4 (*i.e.*, n=4, each index can hold n keys and $n+1$ pointers), shown in Figure 4:

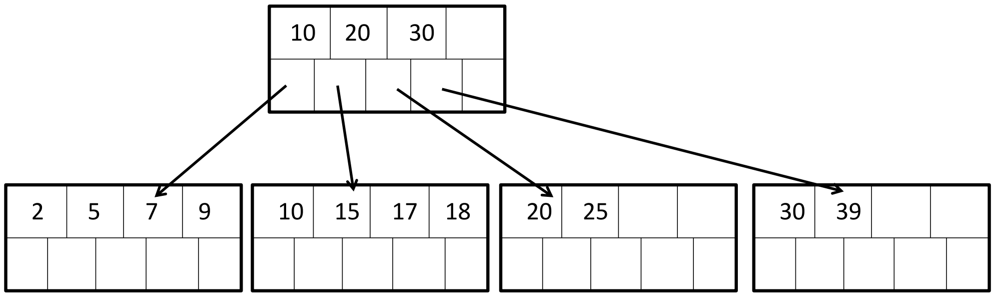


Figure 4: B+ Tree.

Show the resulting tree after inserting key 6 and deleting key 39 (only show one tree). (*4 points*)

Solution:

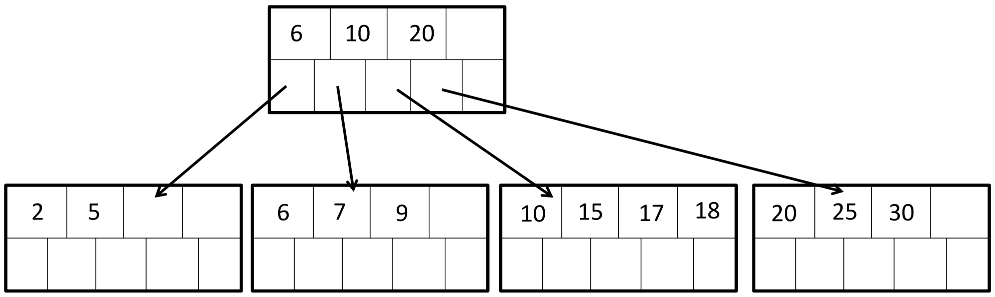


Figure 5: B+ Tree after Inserting 6 and Deleting 39.

(b) Consider you have been asked to index the following key values using an extensible hashing, in order: 34, 24, 9, 78, 44

The hash function $h(n)$ for key n is $h(n) = n \bmod 16$; *i.e.*, the hash function is the remainder after the key value is divided by 16. Thus, the hash value is a 4-bit value. Assume that each bucket can hold 2 data items.

You have performed this indexing correctly, and have come up with the following table, as shown in Figure 6:

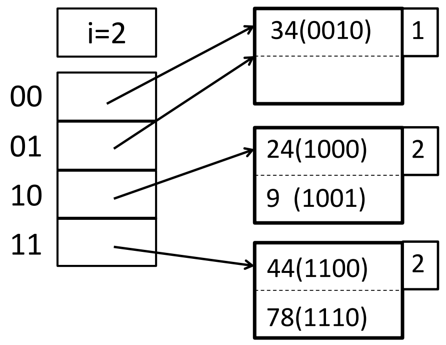


Figure 6: Extensible Hashing Table.

You are now asked to pick one of the following groups of key values to insert into the extensible table. You are interested in 1) which group will result in the largest size of i, 2) and which group will need the most number of data buckets. Give your answer and briefly explain it.

(A) 75(1011), 45(1101)

(B) 72(1000), 39(0111)

(C) 45(1101), 39(0111)

(*4 points*)

Solution:

Inserting 75 will need one more bucket, and change i to be 3;
Inserting 45 will need one more bucket, and change i to be 3;
Inserting 72 will need one more bucket, and change i to be 4;
Inserting 39 won't need any bucket, and won't change i.

Therefore, group A will need the most number of data buckets; Group B will result in the largest size of i.

Problem 5 (*10 points*) Query Processing

(1) Consider two relations, R and S, with $B(R) = B(S) = 10,000$, i.e., both the relations occupy $10,000$ blocks on disk. What is the minimum value of M, i.e., memory, required to compute $R \bowtie S$ using a **block based nested loop join** with no more than the following disk I/O's ?

(a) 35,000 blocks (*4 points*)

Solution:
B(S) + B(R) * B(S) / (M - 1) = 35,000
M - 1 = 10000 * 10000 / (35,000 - 10,000)
M -1 = 10000 * 10000 / 25000 = 100000 / 25
M = 4001

(b) 15,000 blocks (*3 points*)

Solution:
This is not possible since for the minimum IO required for the join is B(R) + B(S) = 20,000

(2) For the two relations mentioned in the earlier problem, i.e., R and S, what is the disk I/O required to perform a **partitioned hash join**? Assume you have sufficient memory.
(*3 points*)

Solution:

For partitioned hash join:
I/O = 3(B(S) + (B(R))
I/O = 3(10000 + 10000)
I/O = 60000

Problem 6 (*10 points*) Query Optimization - Dynamic Programming

Compute the optimal plan for R JOIN S JOIN T JOIN U using the technique of dynamic programming with the following assumptions about the number of tuples in each relations:

$$T(R) = 30$$
$$T(S) = 60$$
$$T(T) = 20$$
$$T(U) = 80$$

For the ease of computation, let's assume that size estimation of a join for two relations R1 and R2 is as follow: **T(R1 JOIN R2) = T(R1) * T(R2)**.

If a subplan is a single relation and does not involve any joins, the size of its intermediate result is zero. The cost of a join is estimated to be the cost of the subplans plus the size of the intermediate results. The cost of a scan is 0.

Draw the table for dynamic programming, to show how to compute the optimal plan for all possible join orders allowing all trees. We have provided you the columns that you must fill in, and the values for the first two columns - you must fill in the rest of the table.

Solution:

Row	Subquery	Size	Lowest Cost	Plan
1	RS	1800	0	RS
2	RT	600	0	RT
3	RU	2400	0	RU
4	ST	1200	0	ST
5	SU	4800	0	SU
6	TU	1600	0	TU
7	RST	36000	600	(RT)S
8	RSU	144000	1800	(RS)U
9	RTU	48000	600	(RT)U
10	STU	96000	1200	(ST)U
11	RSTU	2880000	36000	(RST)U

Problem 7 (*14 points*) Failure Recovery

Consider the following log sequence, and use it to answer the following questions.

Log ID	Log
1	$\langle$START $T1\rangle$
2	$\langle T1, A, 10\rangle$
3	$\langle$START $T2\rangle$
4	$\langle T2, B, 5\rangle$
5	$\langle T1, C, 12\rangle$
6	$\langle T2, D, 8\rangle$
7	$\langle$COMMIT $T1\rangle$
8	$\langle$START $T3\rangle$
9	$\langle$START $T4\rangle$
10	$\langle T3, E, 7\rangle$
11	$\langle T4, A, 2\rangle$
12	$\langle T2, C, 5\rangle$
13	$\langle$ABORT $T3\rangle$
14	$\langle$COMMIT $T2\rangle$
15	$\langle T4, B, 11\rangle$
16	$\langle$START $T5\rangle$
17	$\langle$START $T6\rangle$
18	$\langle T5, D, 3\rangle$
19	$\langle$COMMIT $T5\rangle$
20	$\langle T6, E, 4\rangle$

Note: For the questions (a)-(c), assume the given log sequence is an *UNDO* log.

(a) Suppose we want to start nonquiescent checkpointing after LogID 7.

Between which two LogID lines would we start checkpointing, and what should the log entry (for checkingpointing) look like?

Then, between which two LogID lines would we stop checkpointing, and what should the log entry look like?

Solution: Between Log 7 and 8, $< STARTCKPT(T2) >$
Between Log 14 and 15, $< ENDCKPT >$

(b) Continue from (a). Suppose the system crashes after the last log entry shown above was written to disk. While reading the log backwards, till which Log ID will we have to read the log in this case? Explain. (*3 points*)

Solution:
Till Log ID 8 (after START CKPT). Explain: While reading backwards, we first see an END CKPT statement, and hence, we know that all incomplete transactions began after the previous START CKPT statement. Thus we have to read till that statement only, that is from Log ID 8.

(c) Continue from (a), (b). Show which transactions/actions (*e.g.*: $\langle T1,\ A,\ 15\rangle$) need to be undone and *in what order*. Identify all values of different variables would we need to write to the disk in order to recover the system.

Solution:
The undo transactions in sequence: $< T6, E, 4 >, < T4, B, 11 >, < T4, A, 2 >$
- $< T6, E, 4 >$: write $E = 4$ to the disk.
- $< T4, B, 11 >$: write $B = 11$ to the disk.
- $< T4, A, 2 >$: write $A = 2$ to the disk.

Note: For the questions (d)-(f), assume the given log sequence is a *REDO* log, and the questions refers to the log after question (a).

(e) Briefly explain the meaning of the record: $\langle T4,\ A,\ 2\rangle$ (which is LogID 11).

Solution:

Transaction T4 changed element A. A's new value is 2.

(f) Suppose the system crashes after the last log entry shown above was written to disk. While reading the log backwards, till which Log ID will we have to read the log in this case? Explain.

Solution:

Till Log ID 3. Explain: While reading backwards, we first see an END CKPT statement, and hence, we know that all incomplete transactions either began after the previous START CKPT statement, or are the transactions in the START CKPT statement. Thus we have to read till the earliest START statement for these transactions (i.e., Log ID 3).

(g) Continue from (f). Show which transactions/actions (*e.g.*: $\langle T1,\ A,\ 15\rangle$) need to be redone and *in what order*.

Solution:
The redo transactions in sequence: $< T2, B, 5 >, < T1, C, 12 >, < T2, D, 8 >, < T2, C, 5 >$ $, < T5, D, 3 >$