FINAL CSE 232, March 20, 2000
(open book)

(Please print your name):

FIRSTNAME: ___________ LASTNAME: ___________

TIME: You have 2 1/2 hours (150 minutes) to complete this exam.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max. Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>150</strong></td>
</tr>
</tbody>
</table>
Problem 1 (12P)
Indicate whether each of the following statements is true or false; explain briefly.

a) In query optimization, it is always better to perform projections as early as possible.

**False:** projecting early can save space. However, e.g., $\sigma_C(\pi_A(R))$ may be worse than a later projection if an index on $R$ is available (but not on $\pi_A(R)$).

b) The real world actions in a transaction should be executed before the transaction commits.

**False:** e.g., in an ATM transaction we want to dispense cash only after the transaction has committed: we cannot simply roll back the cash withdrawal.

c) Sequential disk IO is less expensive than random disk IO.

**True:** the head is already “on track”, i.e. at the right position for the next block-read.

d) Checkpoints can significantly reduce recovery time when using UNDO logging.

**False:** this statement holds for REDO logging. But UNDO logging just examines the non-committed transactions.

e) If a schedule is not conflict serializable then it will always violate some consistency constraint of the database.

**False:** e.g., there a schedules that are (view) serializable but not conflict serializable.

f) Consider two schedules with identical precedence graphs: $P(S_1) = P(S_2)$. Assume that $S_1$ is conflict serializable. Circle which of the following claims about $S_2$ are true:

(i) $S_2$ is a serial schedule.
(ii) $S_2$ is conflict equivalent to $S_1$.
(iii) $S_2$ is conflict serializable.

$S_1$ is conflict serializable, so there is a serial schedule $S$ to which it is conflict equivalent (by definition). We know that $P(S) = P(S_1)$. $S_2$ must be conflict serializable and conflict equivalent to $S_1$ since $P(S_1) = P(S_2)$. (Assume the contrary: then by Lemma s.154 their precedence graphs are different: contradiction.) So(ii) and (iii) hold.
Problem 2 (Recovery, 10P) Consider the transaction

\[
T \\
\begin{array}{c}
w(A) \\
w(B) \\
\text{COMMIT}
\end{array}
\]

Decide which of the following snapshots of the database are possible or impossible at any point during or after a transaction.¹
(Here, new(X) denotes the new value after w(X).)

<table>
<thead>
<tr>
<th></th>
<th>LOG</th>
<th>Database</th>
<th>UNDO LOGGING</th>
<th>REDO LOGGING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>possible</td>
<td>impossible</td>
</tr>
<tr>
<td>1.</td>
<td>&lt;T,w(A),...&gt;</td>
<td>new(A)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td>new(A)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.</td>
<td>&lt;T,w(A),...&gt;</td>
<td>new(A)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>&lt;T,w(B),...&gt;</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>&lt;T,COMMIT&gt;</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4.</td>
<td>&lt;T,w(A),...&gt;</td>
<td>new(A)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>&lt;T,w(B),...&gt;</td>
<td>new(B)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5.</td>
<td>&lt;T,w(A),...&gt;</td>
<td>new(A)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>&lt;T,w(B),...&gt;</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>&lt;T,COMMIT&gt;</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**UNDO logging** (i) writes ahead old values to the log, and (ii) writes a COMMIT only after all changes have been flushed to disk.

**REDO logging**: before any change is written to disk, all log values (= new values) have to be written to the log (including COMMIT)

¹Points = max{(correct – wrong), 0}.  

---

3
Problem 3 (Serializability, 10+8+10P) Let $S$ be the following schedule:

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r_1(A)$</td>
<td>$r_2(B)$</td>
</tr>
<tr>
<td></td>
<td>$A := 3 \times A$</td>
<td>$B := B + 5$</td>
</tr>
<tr>
<td></td>
<td>$w_1(A)$</td>
<td>$w_2(B)$</td>
</tr>
<tr>
<td></td>
<td>$r_1(B)$</td>
<td>$r_2(A)$</td>
</tr>
<tr>
<td></td>
<td>$B := B + 10$</td>
<td>$A := 4 \times A$</td>
</tr>
<tr>
<td></td>
<td>$w_1(B)$</td>
<td>$w_2(A)$</td>
</tr>
</tbody>
</table>

a) Instead of $S$, consider first the two serial schedules $S_1 = (T_1; T_2)$ and $S_2 = (T_2; T_1)$. What are the final values $A_f$ and $B_f$ after executing $S_1$ and $S_2$, respectively? (assume the initial values are $A_0, B_0$)

$A_f = 12 \times A_0, B_f = B_0 + 15$ for both serial schedules

b) Now consider the interleaved schedule $S$ above and briefly explain whether $S$ is ...

- **serializable** (i.e., is the outcome equivalent to some serial schedule)?
  - yes, since after executing $S$ we obtain the same $A_f$ and $B_f$

- **conflict serializable**
  - no, since the dependency graph $P(S) = \{T_1 \rightarrow T_2 \rightarrow T_1\}$ is cyclic

- **view serializable**
  - no, since the labeled dependency graph $LP(S)$ contains the cycle $T_1 \xrightarrow{A,0} T_2 \xrightarrow{B,0} T_1$ (cf. rule (3a) on slide 175)
c) Consider the schedule $S'$:

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1(A)$</td>
<td>$r_2(B)$</td>
</tr>
<tr>
<td>$A := 3 \times A$</td>
<td>$B := B + 5$</td>
</tr>
<tr>
<td>$w_2(B)$</td>
<td>$r_2(A)$</td>
</tr>
<tr>
<td>$A := 4 \times A$</td>
<td>$w_2(A)$</td>
</tr>
<tr>
<td>$w_1(A)$</td>
<td>$r_1(B)$</td>
</tr>
<tr>
<td>$B := B + 10$</td>
<td>$w_1(B)$</td>
</tr>
</tbody>
</table>

1) What are the final values for $A$ and $B$ (in terms of their initial values $A_0$ and $B_0$) after executing $S'$?²

$A_f = 3 \times A_0, B_f = B_0 + 15$ (note: the $w_2(A)$ is useless; $w_1(A)$ sees the old value of $A$, multiplied by 3.

2) Is $S'$ serializable/view serializable/conflict serializable?

According to (c1) the outcome is different from those of any serial schedule (a1), hence $S'$ is not serializable. It follows that it cannot be view or conflict serializable. There is also a cycle in LP($S'$): the “candidate serial schedule” is $T_0; T_2; T_1; T_f$ and looks “good” until we consider the dependency $w_b(A) \Rightarrow r_1(A)$: this means that any write of $T_2$ on $A$ must come after $T_1$ - cycle!

3) Under what condition on the initial values $A_0$ and $B_0$ will the outcome of $S'$ be equivalent to a serial schedule?

For $A_0 = 0$

²Remember that $T_1$ and $T_2$ actually work on in-memory copies of $A$ and $B$, so only after executing a $w(\ldots)$ do the corresponding values become visible to other transactions.
d) Consider the atomic actions

- $\text{INC}(X, k) := [r(X); \ X := X + k; \ w(X)]$ and
- $\text{MULT}(X, k) := [r(X); \ X := X \times k; \ w(X)]$.

### d1) Express $T_1$ and $T_2$ in terms of $\text{INC}(X, k)$ and $\text{MULT}(X, k)$.

$T_1 = \text{MULT}(A; 3); \ \text{INC}(B, 10)$

$T_2 = \text{INC}(B, 5); \ \text{MULT}(A, 4)$

### d2) For each of the following pairs of actions, decide whether or not they are in conflict; explain briefly.

- $r_i(X)$ and $r_j(X)$
  
  **no conflict: two reads**

- $r_i(X, k)$ and $\text{INC}_j(X, k')$
  
  **conflict: we cannot swap the actions since $T_i$ will see possibly see different values (before or after the inc)**

- $\text{INC}_i(X, k)$ and $\text{INC}_j(X, k')$
  
  **no conflict: as atomic actions, the two INC commute**

- $\text{INC}_i(X, k)$ and $\text{MULT}_j(X, k')$
  
  **conflict: different outcomes possible**

- $\text{MULT}_i(X, k)$ and $\text{MULT}_j(X, k')$
  
  **no conflict: like INC**

### d3) Give a conflict serializable (but non-serial) schedule based on $\text{INC}(X, k)$ and $\text{MULT}(X, k)$.

**Hint:** use (d2).

$\text{MULT}_1(A, 3); \ \text{INC}_2(B, 5); \ \text{INC}_1(B, 10); \ \text{MULT}_2(A, 4)$

**swapping the non-conflicting INC operations yields a serial schedule.**
Problem 4 (Serializability, 12P) Consider the following two transactions:

- $T_1$: $r_1(A); r_1(B);$ if $A > 0$ then $B := -B; w_1(B);$ 
- $T_2$: $r_2(B); r_2(A);$ if $B > 0$ then $A := -A; w_2(A);$ 

Assume the initial values are $A_0 = B_0 = 1,$ and that we have the following consistency requirement: $A + B \geq 0.$

a) Show that every serial execution maintains consistency.

\[
\begin{align*}
T_1; T_2 & \text{ yields } A = 1, B = -1 \\
T_2; T_1 & \text{ yields } A = -1, B = 1 
\end{align*}
\]

b) Give a nonserializable schedule.

```
"squeeze in" $T_2$ just before the $w(B)$ in $T_1$; e.g.: $[^1r_1(A); r_1(B); \text{if } A > 0 \text{ then } B := -1;]^1$ $[^2r_2(B); r_2(A); \text{if } B > 0 \text{ then } A := -1; w_2(A);]^2$ $[^1w_1(B)]^1$

then $A = B = -1,$ so the result is different from any serial schedule (and violates the constraint)
```

c) Is there a serializable, non-serial schedule?

\[\text{No. Assume that any action of } T_2 \text{ occurs before the } w_1(B). \text{ Then } r_2(B) \text{ must occur before } w_1(B), \text{ hence will read the old value. So } A = -1 \text{ afterwards. In order to avoid the negation on } B, \text{ } w_2(A) \text{ must occur before } r_1(A); \text{ the only way this could be is a serial schedule - contradiction. Similarly (in fact the case is completely symmetric) we can show that no action of } T_1 \text{ can occur before } r_2(B) \text{ in any serializable schedule.}\]

d) Does (c) change if we swap the first two read actions of $T_2$?

\[\text{Yes, now there is a serializable schedule: e.g. start with } r_1(A); r_2(A); \ldots \text{ and continue with the rest of } T_1, \text{ then finally the rest of } T_2.\]
Problem 5 (2PL, 15P) Consider the following schedule $S$:

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r_1(A)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$w_1(A)$</td>
</tr>
<tr>
<td>$A$ := $2 \times A$</td>
<td></td>
<td>$w_2(A)$</td>
</tr>
<tr>
<td>$r_1(B)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B$ := $-B$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w_1(B)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$r_2(B)$</td>
</tr>
</tbody>
</table>

a) Extend $S$ to a 2PL schedule $S_{2PL}$, i.e., add lock and unlock actions.

```
ll(A); rl(A); A:=2*A; wl(A); ll(B); ul(A); l2(A); w2(A); rl(B); B:= B-1; wl(B);
ul(B); l2(B); r2(B); u2(B);
```

b) Add to $S$ a third transaction $T_3$ having a single action $w_3(A)$ immediately after $w_2(A)$. Can the resulting schedule $S'$ be extended to a 2PL schedule $S'_{2PL}$? Explain.

Any 2PL schedule for $T_1$, $T_2$ will place the locks similar as in (a):

- $T2$ must lock$(A)$....
- ... so $T1$ must unlock$(A)$ before
- ... so $T1$ must lock$(B)$ first

When can $T2$ unlock$(A)$?

- Only **after** lock$(B)$.
- But this must happen **after** $T1$ has unlocked$(B)$
- Thus $T3$ can not have a lock$(A)$ at that position!
Problem 6 (Deadlocks, 16P) Consider the following sequence of actions of transactions $T_1$, $T_2$, and $T_3$:

$$r_3 (B) \; r_1 (A) \; r_2 (C) \; w_1 (C) \; w_2 (B) \; w_3 (A)$$

The transactions $T_1$, $T_2$, and $T_3$ initially arrive at times 100, 200, and 300, respectively. We assume the following:

- A transaction requests the necessary lock (shared for read, exclusive for write) on a data item right before its action on that item is issued.
- If a transaction gets all requested locks, it instantaneously completes work, commits, and releases its locks.
- If a transaction dies or is wounded, it instantaneously gives up its locks, and restarts only after all current transactions commit or abort.
- When a lock is released, it is instantaneously given to any transaction waiting for it (in a first-come-first-served manner).

a) If the WAIT-DIE strategy\(^3\) is used to handle lock requests, in what order do the transactions finally commit?

| $T_1$ waits for $T_2$, then $T_2$ for $T_3$. The younger $T_3$ is rolled back when it tries to wait for $T_1$. Hence: $T_2; T_1; T_3$. |

b) Like (a) but for WOUND-WAIT\(^4\).

| The older $T_1$ wounds $T_2$ which is rolled back. Then the younger $T_3$ waits until $T_1$ commits: $T_1; T_3; T_2$. |

---

\(^3\)Older transactions can wait for younger ones, but younger ones are rolled back if they attempt to wait for older ones.

\(^4\)Younger transactions are allowed to wait for older ones; but wounded and rolled back when older ones would have to wait for them.
Problem 7 (B+ Trees, 8P) Consider a relation $R$ with 4000 records. Suppose a B+ tree dense index is built on $R$’s key attribute. A block can either hold 10 records of relation $R$ or a B+ tree node with 20 keys and 21 pointers.

a) At least how many blocks are needed to hold the entire B+ tree index?

\[
\frac{4000 \text{ records}}{20} = 200 \text{ leaf nodes}, \text{ so } 10 \text{ level-2 nodes} + 1 \text{ root node will do: hence } 200 + 10 + 1 = 211
\]

b) We want to retrieve a tuple for a given key value. Assume we can keep 10 blocks in memory for holding some record blocks or index nodes. For minimizing the expected number of disk IOs, how would you make use of the 10 blocks? What is the expected number of disk IOs in that case?

E.g., keep root + 9 level-2 nodes in memory: \( \frac{10-9}{10} \times \text{level-2} + 1 \times \text{level-3 (leafs)} + 1 \times \text{(data block)} = 2.1 \)

More realistically, one block is reserved for read, and one for write: \( \frac{10-7}{10} \times \text{level-2} + 1 + 1 = 2.3 \)
Problem 8 (SQL, 15P) Consider the following relations:

Person(SSN, DeptId, Salary, ...), Department(DeptId, Name), Spouse(SSN1, SSN2).

The first relation contains data about persons: the SSN attribute is used to identify the person and DeptId identifies the department a person works for. A department has an id and a name. The Spouse relation is symmetric and contains all (current or former) spouses of a person.

Express in SQL the following queries:

a) “find all bachelors, i.e., persons who are not married to any other person”

```
SELECT SSN
FROM Person
WHERE SSN NOT IN
    (SELECT SSN FROM Spouse)
```

b) “find the names of departments and the average salary of persons working in the department”

```
SELECT Name, AVG(Salary)
FROM Department D, Person P
WHERE D.DeptId = P.DeptId
GROUP BY DeptId
```

c) “find the names of departments and the average salary of persons working in the department, for departments in which only bachelors work”

```
SELECT Name, AVG(Salary)
FROM Department D, Person P
WHERE D.DeptId = P.DeptId
AND D.DeptId NOT IN
    (SELECT DeptID
     FROM Department, Spouse, Person
     WHERE SSN = SSN1 AND Dept.DeptId = Person.DeptId)
GROUP BY DeptId
```
Consider the relations \( R(A, B, C) \) and \( S(C, D, E, F) \). Transform the following SQL to an equivalent algebra expression using only \( \sigma, \pi, \text{ and } \bowtie \) and in which selections and projections are applied as early as possible. (Hint: you may simplify the join conditions.)

\[
\text{SELECT } A, E \\
\text{FROM } R, S \\
\text{WHERE } R.A < 10 \text{ AND} \\
\hspace{1cm} R.C = S.C \text{ AND} \\
\hspace{2cm} R.C + S.D > 100
\]

Since \( R.C = S.C \) we can simplify the condition to \( R.C = S.C \text{ AND } S.C + S.D > 100 \text{ AND } R.A < 10 \). We eventually get: \( \pi_{A,E}(\sigma_{A<10}(R) \bowtie_{C} \sigma_{C+D>100}(S)) \). We can further apply \( \pi_{A,C} \) on \( R \) and \( \pi_{C,E} \) on \( S \).

Consider the relations \( R(A, B), S(A, B, C), \) and \( T(A, B, C) \). Which of the following expressions are equivalent to each other? Explain briefly.

a) \( \pi_{A,C}(\sigma_{B<10}(R) \bowtie (S - T)) \)

b) \( \pi_{A,C}(R \bowtie (\sigma_{B<10}(S) - \sigma_{B<10}(T))) \)

c) \( \pi_{A,C}(\pi_{A}(R) \bowtie (\sigma_{B<10}(S) - T)) \)

(a) and (b) are equivalent: push the \( \sigma \) up and over to the \( S - T \); this is correct since we also join on \( B \). In contrast, (c) is not equivalent to the others since the projection \( \pi_{A}(R) \) discards the formerly joined-upon attribute \( B \).
Problem 11 (IO Cost, Join Algorithms, 12P) Consider two relations R and S containing 6000 and 3000 tuples, respectively. Assume that 10 tuples fit into a block and that 101 blocks are available. We also assume that no block is needed for writing (say the output is consumed in a pipelined fashion). Determine the number of disk IOs for the following joins:

a) nested-loop join with clustered relations, block access

\[ \frac{3000}{100} \times (100 + 600) = 2,100 \]

b) merge join with clustered, ordered relations

\[ 600 + 300 = 900 \]

c) merge join with clustered, unordered relations

we have to add the cost of sorting which is 4 (or 3 if joining on the sorted runs) * 900

\[ = 4,500 \text{ (or 3,600)} \]

d) hash join (with ideal distribution across buckets).

\[ (2 \text{ (“bucketize”) + 1 (“join”)}) \times (600 + 300) = 2700 \]