T–106.5221 Transaction Management in Databases

Exam, August 26th, 2011

Allowed materials: writing implements, calculator (not necessary).
Write the following clearly on top of every paper you submit: "T–106.5221, August 26th, 2011", your full name, student ID and study programme, and the **total number of papers** you submit.

1. (6p) Explain the following concepts briefly:
   (a) WAL (write-ahead logging) protocol
   (b) phantom phenomenon
   (c) "steal" buffering policy
   (d) latch-coupling
   (e) unrepeatable read
   (f) update-mode lock (U-lock)

2. (6p) We apply the key-range locking protocol. What locks are acquired by the transactions in the following histories and when are these locks released? Are the histories possible under the key-range locking protocol? What isolation anomalies (dirty writes, dirty reads, unrepeatable reads) do the histories contain? In all cases, the database is initially empty.
   (a) $B_1 I_1[5] B_2 I_2[7] C_2 B_3 I_3[3] C_3 I_1[9] C_1$
   (b) $B_1 I_1[5] C_1 B_2 R_2[5, > 2] B_3 I_3[3] C_3 I_2[7] C_2$
   (c) $B_1 I_1[5] C_1 B_2 I_2[9] D_2[5] B_3 I_3[5] C_3 C_2$

3. Relation $R(A, B, C, D)$ has a sparse index (a B$^+$-tree) on the pair of attributes AB (a unique key of R) and a dense static hash-table index on attribute C. (a–c) (3p) How can these index structures be utilized in the implementation of the following operations?
   (a) select D from R where $A = 12$ and $B < 8$;
   (b) select $*$ from R where $A \geq 59$ and $C \leq 400$;
   (c) select $*$ from R where $C = 35$;
   (d) (3p) Let us execute the following operation:
   insert into R values (15,20,35,40);
   Assume that the B$^+$-tree mentioned above has height 4, and all of its nodes are as full as possible before the insertion – thus, some nodes need to be split. What structure modifications are done to the tree, and in what order? Which pages of the tree are latched, and when is each latch released? The course discussed several possible answers to these questions; you can choose any one of the methods described on the course for your answer.

4.(a–c) (3p) The transaction T generated from the following program fragment is run concurrently with other transactions.
   set transaction isolation level L;
   begin transaction;
   select avg(V) into :a from r;
   select sum(V) into :s from r;
   select count($*$) into :c from r;
   insert into q values (:a, :s / :c);
   commit transaction.
   How can the result produced by transaction T vary when (a) L = "read committed", (b) L = "repeatable read", (c) L = "serializable"? We assume that all other transactions that possibly run concurrently with T have isolation level "read uncommitted", at least.
   (d) (2p) Assume that L = "serializable" and that all other concurrent transactions have the isolation level "cursor stability". Is it possible for transaction T to work more efficiently in this situation, when compared to the situation where L = "serializable" and all others have the isolation level "repeatable read"? Why or why not?
   (e) (1p) On which isolation levels can the Commit-LSN mechanism be used?

(problem 5 is on the next page)

5. (6p) The contents of the log on disk at the time of a system crash are the following:

101: $\langle$*begin-checkpoint*$\rangle$
102: $\langle$*transaction-table*, $\{\}\rangle$
103: $\langle$*page-table*, $\{\}\rangle$
104: $\langle$*end-checkpoint*$\rangle$
105: $\langle T_1, B\rangle$
106: $\langle T_1, I, p, 11, 5, 105\rangle$
107: $\langle T_2, B\rangle$
108: $\langle T_2, I, p, 15, 6, 107\rangle$
109: $\langle T_2, C\rangle$
110: $\langle T_3, B\rangle$
111: $\langle T_3, I, p, 16, 2, 110\rangle$
112: $\langle T_1, I, p, 19, 1, 106\rangle$
113: $\langle T_3, D, p, 15, 6, 111\rangle$
114: $\langle T_3, A\rangle$
115: $\langle T_3, D^{-1}, p, 15, 6, 111\rangle$

What actions are involved in restart recovery when using the ARIES algorithm? What log records are generated and when is the log forced to disk? Assume that Page-LSN = 108 in the disk version of p. We assume that all inverse operations in the undo pass can be performed physically.