1. [15 points] Locking.
   
   a. [5 points] Draw the compatibility matrix for the locks proposed in the “granularity of locking” paper.
      
      See the matrix in the paper.
      
   b. [5 points] Why is it useful for a DBMS to support locking at multiple granularities?
      
      The main idea here is that if you choose only a coarse granularity, you will get unnecessarily low concurrency for some transaction workloads, while if you choose only a fine granularity, you will get unnecessarily high overhead for other workloads. Supporting multiple granularities is an attempt to get the best of both.
      
   c. [5 points] State the requirements for obtaining locks in the presence of a hierarchy that is a DAG (for example, think of the case where one path through the hierarchy is DB → file → page → record, and another path is DB→index→record.)
      
      One important aspect was that writers have to lock all paths while readers only have to lock a single path. Another important issue has to do with implicit locks – for correctness, one has to consider implicit as well as explicit locks.

2. [5 points] Oracle Concurrency Control.

   In Oracle’s concurrency control scheme, readers never block. Given this, how is it possible for readers to see a consistent version of the data in the presence of other transactions updating the database?

   The key idea here is that Oracle uses multiversion CC – so that writers, instead of updating in place, create new versions while readers can read versions that were current as of their start time. The CC mechanism can determine which version a read-only transaction should read by checking the start time of that transaction.
3. [10 points] Degrees of consistency.
   
a. Give an example of an anomaly allowed at degree 2 but not degree 3.

   **Unrepeatable reads. For example,**
   
   \[ \begin{align*}
   T1: & \text{ update}(X) \\
   T1: & \text{commit} \\
   T2: & \text{read}(X) \\
   T3: & \text{update}(X) \\
   T3: & \text{commit} \\
   T2: & \text{read}(x)
   \end{align*} \]

   *Could read different values within T2.*

   b. Give an example of an anomaly allowed at degree 1 but not degree 2.

   **Dirty reads. For example,**
   
   \[ \begin{align*}
   T1: & \text{update}(X) \\
   T2: & \text{read}(X) \\
   T1: & \text{abort}
   \end{align*} \]

4. [10 points] What is the key insight into the B-link protocol that allows higher concurrency than traditional locking while still maintaining the correctness of the index?

   I was looking for an answer that said the problem arises when readers find that a key they are searching for does not belong on the node they are at (or in a subtree of that node.) By arranging things so that splits always cause keys to move to the right (if they move at all), and adding high-keys and right-links, readers can still find what they are looking for. Other possible answers focused on the fact that B+trees don’t have to be the result of any serial schedule of transactions, rather, they just have to be correct in that they can be used correctly to look up data values.

5. [10 points.] Consider the Kung and Robinson optimistic concurrency control algorithm. What is the benefit of the parallel validation algorithm over the serial validation algorithm? (Be more specific than just saying “the parallel algorithm allows something to happen in parallel.”)
The benefit is that it allows more schedules than are allowed by serial validation, so that higher concurrency is possible. It does this by allowing the overlap of validation and write phases (subject to additional restrictions on read and write sets.) In particular, for two transactions $T_i$ and $T_j$ such that $WS(T_i)$ intersect $RS(T_j) = \text{NULL}$ and $WS(T_i)$ intersect $WS(T_j) = \text{NULL}$ and $T_i$ completes its read phase before $T_j$ completes its read phase, the parallel validation may let $T_j$ commit in instances when the serial condition will not.

6. [15 points] ARIES. Give two conditions during the redo phase when the recovery manager can determine that it does not need to redo an update to a data page without even examining the page in question. For each, describe how the situation could arise.

The conditions are: (a) the page $\notin DPT$, and (b) the page’s recoveryLSN in $DPT > LSN_{\text{LogRec}}$. Condition (a) implies that the page was written out to disk before the most recent checkpoint but after the update being considered. Condition (b) means that it was written out to disk before the most recent checkpoint but after the update being considered, then read back in, and dirtied.

7. Transaction Management in R* Distributed Database Management System [15 points]

a. In the presumed commit protocol, in the abort case the subordinates force aborts and the coordinator waits for receiving “acks” from all subordinates before forgetting a transaction. Why is this necessary?

The subordinates have to force their aborts, otherwise they could crash after ack’ing the abort message from the coordinator, find no outcome for the transaction, and ask the coordinator (who will say “commit” in this case.) The coordinator waits for acks to be certain that the subordinates have all forced their aborts. The ack’s after forcing aborts constitute a guarantee from the subordinates that they will not ever ask the coordinator about this transaction.

b. By contrast, in the commit case the subordinates do not force commits and the coordinator does not wait for “acks” before forgetting the transaction. Why is this correct?

This goes to the heart of the “presumed commit” protocol. Since the subordinates are all going to commit, it is OK for them to ask the coordinator about the transaction even after the coordinator has forgotten it, because the coordinator will say “commit.”

8. Join Algorithms [10 points]
In the GRACE hash join algorithm, the join can be accomplished with one partitioning pass and one join pass through $R$ if $|R| < M^2$, where $M$ is the size of memory. Is the same statement true for Hybrid hash join? If so, say why; if not, explain why not and give an alternate bound that allows the join to be accomplished with one partitioning pass and one join pass through $R$.

The same is not true for Hybrid. The issue is that the first partition, say partition $R_0$, has to be memory resident during the partitioning phase, so if there are $k$ partitions, we need more than $k$ memory buffers. One way to get a bound is to note that $|R_0| = |R|/k$, so we require (a) $|R|/k + k < M$, and (b) $|R|/k < M$. Clearly condition (b) is redundant. From (a), rearranging a little, we get $|R| < k(M-k)$. The right side of this is maximized when $k = M/2$. So we get $|R| < M^2/4$, or $\sqrt{|R|} < M/2$.

9. ADTs to RDBMS [10 points]

What is the motivation for adding ADTs to RDBMSs? What is one difficulty in doing so?

Writing queries involving types not “natively” supported in the DBMS can be very awkward – one must sort of translate the type into basic SQL and operations on the type into SQL operations. It would be much nicer for query writers if they could define new types and use them in their queries. It might even be more efficient if the query evaluation mechanism in the DBMS could understand special semantics of these types. But that is one of the difficulties – the DBMS cannot understand the special properties in user’s methods implementing ADTs, and because of this it is hard to evaluate queries using these ADTs efficiently (both because of optimization difficulties, such as “how do you do selectivity estimation for user types”, and evaluation algorithms, for example, how do you get an RDBMS to do a spatial join if it doesn’t know about spatial types?)