# Question 1 – How would you compare RR and CS

Repeatable Read is DB2’s highest level of isolation, and the closest equivalent to a serializable schedule (also avoids phantom problem, with an insert not being caught by the lock). Cursor Stability is a sort of Read committed isolation level. RR will be surer of correct results, but it will lock a lot more than CS and then force the DB to work less in parallel, making it slower than CS.

The currently commited semantics feature of CS makes it able to read from the currently commited result, instead of the read having to wait for an update lock to be release. So instead of waiting it can fetch the last commited, as if it didn’t know anything was updating the particular row. This is okay for most situations, and a lot faster in most scenarios. This has a slight performance penalty because of additional logging.

Almost always CS will be fine, unless you are working in a business where you have to be absolutely sure than everything is done atomically and serializable.

# Question 2 – Currently Commited semantics performance

With the currently commited semantics feature, the Cursor Stability isolation level no longer forces readers to wait on update locks, but can use ‘Snapshot isolation (oracle)’ to read the lastly commited data. From the readers point of view it looks like the update isn’t there, and we work on the data before (still don’t read dirty data). This is of course much faster for the reader because we don’t lock him.

There is a slight performance impact because the database has to provide the ability to return the last commited data (impacts the level of logging).

Another reason to use CC is also that it avoids lock timeouts and deadlocks

In the textbook, the reads under snapshot isolation don’t uptain a lock, but writes are ‘certified’ upon commit. But in the CC, readers just don’t have to wait for the writer to release the locks, but can return data based on the currently commited.

# Question 3

**How would an extra server impact the results?**

The problem in Assignment 2 wasn’t cpu performance but IO performance. So an extra server would only help if it ment that we provided an extra disc as well? But then we still would have two separate database-servers working along side, and would have to synchronize data somehow

**How would an extra disc impact the results?**

If we added an extra disc to the server, we could use a dedicated log-disc, which would ensure that the log was written sequentially (fastes normal-disc performance). As the slow part of the io’s actually is writing the log to disc and not the data (providing that the buffer is big enough), optimizing the log would lead to better overall performance.

# Question 4

1. **Whats the meaning of the given query?**

The query returns number, branch of only the rows which balance exactly matches the average balance of all accounts with the same branch.

No the distinct keyword is not needed as we select the primary key from the query (primary key is unique)

(and the distinct will slow the query up, as it could make the database sort the results to be sure that there really is no duplicates)

**C)**

First of, we could make the index covered, by adding the branch to the index.

Next, we are calculating the average for each row in the accounts table – which means that we are calculating it way to many times. It should optimally only be calculated once for every branch-type in the table. We could do this by calculating the average into a temporary table before doing the query. And then using the temporary table containing the averages in the main query.

Actually when doing the average query, an nonclustered index on the branch and balance, would work fine for this – As the query would be able to first find the correct branch, and then still use the index to find all the balances.

Also.. make sure that the indexes are online, because if they are offline and we inserted all the rows after creating the index. The index will be very inefficient if not rebuilt..

# Question 5

1. **Why is deletions in the person table slow, what is the fix?**

To delete a person the DB first has to find the person.. which is fast enough using the index.. but then it needs to uphold the foreign key references in Employee, on which the isn’t any index. So the DB has to scan employee for every person we wish to delete.

A Nonclustered index on person\_id and manager\_id would make this deletion faster. This should be two separate nonclustered indexes.

1. **Why is the query slow, how could we make it faster?**

Reasons for query being slow:

* No indexes on employee
* Person.name is not covered
* Having count(\*) >= 100 filters after selection.. could we do this earlier?

Optimize:

* Make the person index on id and name, so that its covered
* Create an index on employee manager\_id, person\_id

So that it can first use the manager\_id for the group, and then person\_id is covered.