Lecture 10

COMP207

Mistake in last weeks assessment

- Someone pointed out that I had forgotten to specify that after the crash in question 1, no changes were made before the server came back online, as is otherwise the case in all the similar questions.
- If you thought that it might have undone transaction 1, it would break durability. Therefore, if you answered either that it broke nothing or that it broke durability, you got points for that question.

FAQ to SQL assignment

- You may use any operation that works on CodeGrade, even if not covered in the course. Conversely, you may not use any operation that does not work on CodeGrade…
  - all operations covered in the course do work on CodeGrade
- If you have answered a question but it says that the view does not exist, it is mostly because there is an error in your SQL syntax of some kind
  - You can see the error if you click on > besides Setup and scroll down to Per Student Setup
- In regards to question 5, CustomerTrip contains a row for each customer trip. It states which customer is on that trip (in c_id), which stop they got on (from_stop_no) and which stop they got off (to_stop_no). The bus drives from stop 1, to stop 2, to stop 3, and so on (you may, if you wish to, use that it ends at stop 10, but it is really not useful)
  - If you want an actual hint: try to for a given bus id (say 1) and stop number (say 5), find the number of passengers on that bus at that stop. Then use the GROUP BY generalization trick

Checkpoints for logs

- Checkpoints are meant to speed up recovery using log files
- We saw two types:
  - Basic checkpoints
  - ARIES checkpoints

Creating basic checkpoints

- To create a basic checkpoint, the schedule must stop accepting new transactions until each running transaction has stopped (i.e. committed or rolled back)
  - You do not want to do that in most cases
- Ensure that all changes have been written to disk, all rolled back transactions rolled back and write <CHECKPOINT> in the log file
- Start accepting new transactions

Using basic checkpoints

- You never need to go before a checkpoint when you use basic checkpoints
  - All changes before that have been either undone or have been finished
- You do undo logging starting from the end of the file and up to the last basic checkpoint
- You do redo logging starting from the last basic checkpoint and until the end of the file
- I.e. in short: you can delete the part from the start of the log file and until the last basic checkpoint and everything just works

Creating ARIES checkpoints

- ARIES checkpoints require undo/redo logging
- Whenever you want to create an ARIES checkpoint:
- Write <CHECKPOINT($T_1, T_2, \ldots, T_n$)> where $T_1, T_2, \ldots, T_n$ are the transactions currently running
- You then write out all the buffers unto the harddisk
  - as they were at the time you started the checkpoint
- You write <END CHECKPOINT>

Using ARIES checkpoints

- Find the last <CHECKPOINT(T_1,T_2,…,T_n)> for some transactions T_1,T_2,…,T_n that has an <END CHECKPOINT>
- Redo each operation from there until the end of the file
  - I.e. for operation <T1,X,10,20>, set X to 20
  - You only need to do it for the transactions that commit but it does no harm to do it for all
- Also, keep track of which transactions did not commit
- Undo each operation from each transaction that did not commit (incl. before the checkpoint) from the end of the file to the transactions beginning
  - Note: This is backwards from the end
  - I.e. for operation <T1,X,10,20>, set X to 10

Schedule types

- We saw three types of schedules:
  - Recoverable
    - If a read operation reads something last written by another transaction then that transaction must commit before the transaction that has the read operation
  - Cascadeless
    - If a read operation reads something last written by another transaction then that transaction must commit before the read operation
  - Strict
    - If a read or write operation reads or writes something last written by another transaction then that transaction must commit before the read or write operation

Transaction isolation levels

- Transaction isolation level for short-hand notation
- Read uncommitted is always satisfied in this course (not always in MySQL though)
- Read committed is satisfied for a schedule if for each read, the last transaction that wrote to that variable before the read operation has committed
  - If the read and the write operation is from the same schedule that in itself does not break Read committed
  - I.e. read committed is satisfied by cascadeless schedules

Transaction levels continued

- Repeatable read is satisfied if read committed is AND whenever there are two reads on the same variable in the same transaction there is no write operation on that variable in-between
  - If the write operation is from the same schedule as the read operations it in itself does not break repeatedable read
- Serializable (as a transaction level) is satisfied if repeatable reads is AND the schedule is serializable (i.e. equivalent to a serial schedule)
  - If transaction level is not explicitly mentioned you get to ignore the repeatable reads part, because otherwise conflict serializable is not serializable

How do we ensure the transaction levels?

- Read Uncommitted
  - Trivial
- Read Committed
  - Cascadeless schedules
- Repeatable Read
  - ???
  - (real answer: we can use the timestamp-based schedules from next Tuesday)
- Serializable
  - Conflict serializable

Types of schedules

- We have seen the following kinds of schedules:
  - Serial $\subset$ conflict serializable $\subset$ serializable schedules
  - Serial $\subset$ strict $\subset$ cascadeless $\subset$ recoverable schedules

- Each pair of types of schedules: Either, they are on the same line and then have a specific subset relationship or they are not and there are examples (in the videos) of them not having a subset relationship in either direction
  - E.g. some strict schedules are not serializable and some serializable schedules are not strict