Course	**Databases and Information Systems 2018**	
Exercise Sheet	**5**	
Points	–	
Release Date	**May 29 2018**	Due Date	**June 13 2018**

1 Logging and Recovery
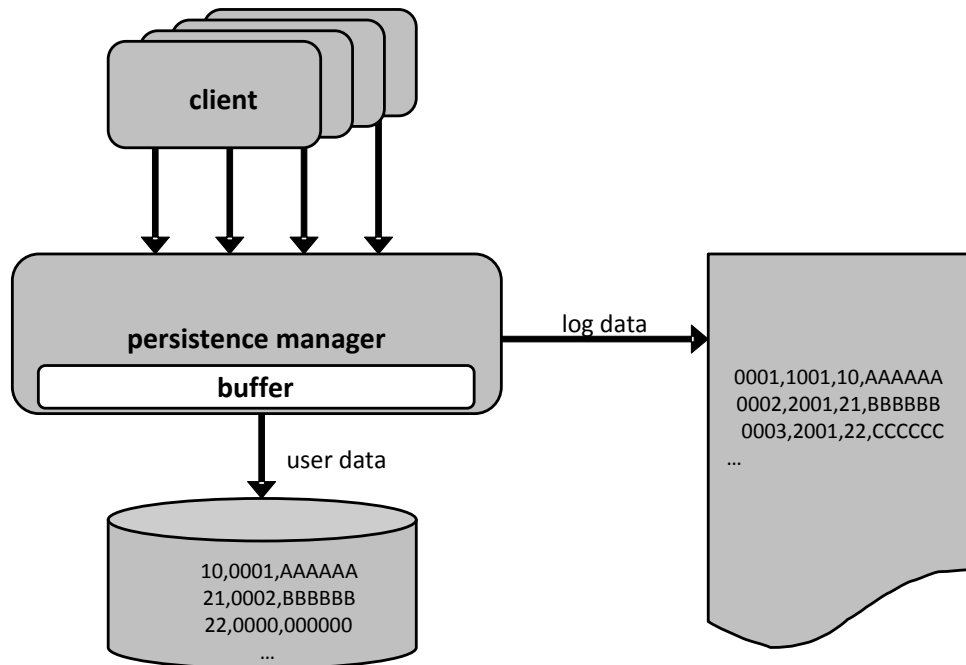
1.1 Persistence Administration

Realise the above architecture for a simplified persistence manager that allows concurrent requests from clients to user data, performs deferred writes for modified data and takes the necessary precautions for recovery after a system failure.

Besides the user data, the persistence manager also administers log data to keep track of modifications that have not yet been persisted. User data and log data have to be saved to a persistent storage. User data are stored with page ID and log sequence number (LSN); log data are stored with their LSN, transaction ID and page ID.

Initially, user data that have been modified by a write operation are stored to the internal buffer of the persistence manager. Already existing versions can be overwritten directly in the buffer without having been written to the persistent storage. If the buffer contains more than five datasets after a write operation, the datasets corresponding to a committed transaction are written directly to the persistent storage (non-atomic). Thus, outdated datasets can reside in the persistent storage and datasets of committed transactions that have not been written to the persistent storage can reside in the buffer, but "dirty" datasets of uncommitted transactions cannot exist in the persistent storage (noforce, nosteal). Log information have to be written to the persistent storage immediately and before the completion of a transaction, in particular, so that a recovery can be performed in the event of a system failure.

There's only one persistence manager that is accessed concurrently by several clients (remember the Singleton pattern and thread safety during concurrent access). The persistence manager should offer at least the following operations:

- `beginTransaction()`: starts a new transaction. The persistence manager creates a unique transaction ID and returns it to the client.

- `commit(int taid)`: commits the transaction specified by the given transaction ID.

- `write(int taid, int pageid, String data)`: writes the given data with the given page ID on behalf of the given transaction to the buffer. If the given page already exists, its content is replaced completely by the given data.

1.2 Clients

Implement a client class that can be started in several instances in parallel, so that all instances access the persistence manager concurrently.

The clients repeatedly execute transactions on the persistence manager according to the following scheme:

beginTransaction() write() write() ... commit()

For more realistic behaviour, every operation is followed by a brief pause. The number of writes in a transaction may vary.

In order to get along without locks, the clients do not access the same pages, i.e. Client 1 accesses pages 10..19, Client 2 accesses pages 20..29 and so forth. More than one page may be modified by one transaction. The user data may consist of simple strings.

1.3 Recovery Tools

Due to the noforce, nosteal, non-atomic implementation of the persistence manager, no undo recovery is required after a system failure, but a redo recovery is. Implement a recovery tool that performs the analysis and redo phase of the crash recovery as discussed in the lecture. First, the so-called winner transactions have to be determined from the log data. After that, the pending write operations have to be executed. Remember updating the LSNs in the user data.

	Course	**Databases and Information Systems 2018**		
	Exercise Sheet	**5**		
	Points	–		
	Release Date	**May 29 2018**	Due Date	**June 13 2018**

Note

- Persistence manager and clients can be implemented as components of the same Java application, so that clients can be started as single threads accessing the only persistence manager instance (Singleton pattern).

- Use the following procedure to save user and log data: store every page in a single (text) file that is accessed by the persistence manager via `FileReader` and `FileWriter`. The name of a file corresponds to the page ID/LSN and every file contains a single line of text with all the information, for example separated by commata.

- The buffer can be implemented with a `Hashtable`. Besides the user data, the correspondence between transactions and datasets has to be administered and an overview over ongoing and already completed transactions has to be maintained.

- Creating checkpoints is not necessary.

- The implementation of a graphical user interface is neither required nor desired.