Sample projects for TOYDB:

Notes :
1) In each project, clearly define objectives, measurements to be made, inputs to be used. Each project below requires detailed planning on use of AM, PF layers, making changes in code to obtain statistics, defining buffering and replacement policies, etc. Your project submission should clearly state goals, detailed design, techniques to be implemented, and statistics to be collected. Generate input data size large enough to get meaningful results.
2) You can come up with your own project, but discuss it with me or any of the TAs.
3) Each group should work on a different project. If you repeat any project topic, marks will be deducted. We will use first-come-first-approved scheme.
4). Each group to have only 3 students. The groups have been made already and the list will be shared with you by Ms Priya. Please enter the title of your project in that list.
5) The sample list is only indicative. You have to expand the problem statement and then get into project work (analysis, implementation, etc).
6) Most of the following projects can be done in the context of flash storage, and you can appropriately extend the AM and PF layers for flash and do your experiments.

Project timelines;

1. Enter title of your project by 31st Octo morning, 8.00am latest
2. Submit Project design document which will contain : title, brief description of work planned, algorithm details, input design, overview of how you plan to use toyDB code, what will be obtained in the output (no coding is expected at this time) : Nov. 7, 2019, 8.00am latest
3. Code, final report giving details of implementation done, results obtained, explanation of the results, and conclusions : Nov. 14, 2019, 8.00am
4. Demos/quiz in the week of 18th Nov.

1. Provide page buffering for the PF layer using two page replacement strategies to be chosen for a file at the time of opening it : LRU and
MRU (most recently used - which may be profitably used for sequential file access). You also need to provide 'dirty' flags for updated pages
(and add a call to indicate that a page has been modified). Define buffer pool size as a parameter. The program should maintain statistics
on pages accessed (for read, write), logical and physical number of I/O's, etc.

2.To the paged file module, add disk storage simulator, which simulates pages for m disks (each with n cylinders, m tracks/cylinder, and k sectors per track), and also a specific disk technology (like RAID 0, 1 or 5). You may do mapping of page numbers to disk address. The simulation should provide for a suitable access arm movement strategy, a cache on the disk, and gathering of statistics that will allow us to evaluate performance.

3.Use PF layer to build a slotted page structure for storing a varying length record file. As a strategy to accommodate updates, you may want to ensure that each page has some p % free space at any time. Use it to store students data. Provide for insert, delete, update
of records, and for sequential reading. Collect some performance data like space utilization (after some number of updates).

4. The AM layer can be used to build index for a file. Compare the performance of this AM layer in

building an index by inserting the file records one-by-one. (when the input file is not sorted). Compare this performance with loading the file where the file is pre-sorted. In each case, provide for a large buffer. Also obtain buffer utilization factor (hit ratio). Also obtain B+ tree data like no of levels, no of nodes, space utilization.

5. Using AM layer, create a secondary index on a repeating (non-unique) attribute of a file (whose records are not in the key order) using two approaches : 1) using bucket of pointers, and 2) making keys unique by appending them with serial number or primary key of the records. Study space and performance characteristics of the two approaches for a selected set of queries. Use AM layer to create primary index but write B+ tree algorithms of your own to create the secondary index in (1) above.

6. Use the two layers to implement two external sort algorithms (one using the merge sort, and the other based on B+ tree scan), and compare performances for the sample data.

7.Implement extended hashing access method on top of the PF layer. Provide proper interface for your access method, and then try comparative performance
of this and the present AM layer in terms of storage and access costs for a given file and for a given set of queries.

8. Build an index on multiple attributes (take 2 attributes, say A and B) using the AM layer, by concatenating A with B values. For the same file, build2-dim GRID File on top of PF layer. You may fix the scales for A and B and build the GRID index. Get statistics during GRID file creation (no of pages in the index, no of pages in the file, IO counts, buffer size, hit ratio, etc). Compare performance of the two approaches for sample queries and sample data.

9) Implement construction of B+tree when the file is sorted by the key already, and where the tree is constructed from leaf level onwards, level by level. Use adequate buffers to get better performance (study for different size of buffers). Get B+ tree data, and compare performance with AM layer.

10) Use PF layer to simulate performance of disk-to-disk backup where the disks are within the same controller having cache of M pages. Assume the database to be on N1 disk spindles in RAID 01, and the backup to be on another set of disks but in RAID 0. Do as much sequential reading/writing as possible. Use as much parallelism in IO across the spindles as possible.

11) Use PF, AM layers for a database application of given size on disks in RAID5. Consider a set of query and update transactions (these can be generated as random set of operations on pages, e.g., R50, W72, R23, ... where Rn or Wn means read/write page n). Get performance of the DB for this workload when there are no failures. Now assume that while this workload is getting executed, we randomly get some data errors which need to be corrected using parity blocks. If the errors are about 2%, find impact on performance.

12) Use PF layer to simulate RAID01 disk system over n disks. Consider a workload W (consisting of a sequence of read/write operations) and get the DB performance. Next, assume that we need to take a snapshot of the database just before we start W. Use the copy-on-write technique for the snapshot. After the snapshot, we resume the workload W, but also initiate in parallel, the backup of the DB from the snapshot. Simulate this situation now, and get DB performance for the workload W when the backup is also going on.

13) Same as 12 above, except you use redirect-on-write technique for snapshot.

14) Use PF layer to simulate a large database stored on N disks in RAID 01. Assume that only a small percent m of the database is in active use ($m < 5\%$). Design workload and carry out measurements to simulate MAID technology and measure impact on performance, power savings, etc. We may have strategies to have 'hot' (active) data on one set of (usually) high rotation speed disks and passive or less active data migrating to slow disks. Plan such a technique and measure performance impact, power saving, etc.

15) Use PF layer to simulate a solid-state drive, and compare its performance for a set of random read-write operations with a normal disk.

(You may meet me for discussing the copy-on-write and redirect-on-write techniques mentioned in topics 12, 13 above)