Description
The objective of Homework 2 is to highlight: a) the possibility of configuring a main-memory database using Berkeley DB (BDB), b) role of primary and secondary databases, c) limitations/bugs/features of Berkeley DB. Towards this goal, this homework configures a main-memory instance of BDB with a database and a fixed amount of buffer-pool size (memory, $M$). Next, it inserts records onto it until it becomes full. The records are the same as those from Homework 1 consisting of four attributes: Id, Age, Salary, MemberName. The secondary database indexes the Age attribute using a B+-tree. The primary database indexes the Id attribute using a hash index structure.

Part 1:
Part 1 inserts records onto the database until it returns an error, indicating that memory is full. It iterates records to ensure that all the inserted records are there. If records are missing, it might be because your program did not initialize with appropriate flags (enable the database to be transactional so that it may recover from errors and does not become corrupted). Once BDB returns an error code that it cannot insert additional records, print how many records ($R$) and bytes ($B$) occupy the main memory database. Your program may report the memory utilization by computing $B/M$. Next, delete all those records with an even Id (multiple of two). Print how many records were deleted ($D$) and how many bytes were freed up ($F$). Next, insert back the even numbered records (those that were deleted) and measure the total number of records ($R$) and bytes ($B$) in memory. Repeat this process of deleting and inserting records ten times, reporting the utilization of memory after each iteration. You may switch between odd and even records in each iteration if you wish to do so. Analyze your implementation to answer the following: “Is BDB able to obtain the same memory utilization after each iteration of deleting and inserting records?”

To see the impact of memory utilization analyze the above question with variable sized records. Assume record sizes are dictated by their Id using the following function: $[1+(Id \mod 10)] \times 1024$. This produces ten possible record sizes: 1K, 2K, ..., 10K.

Part 2:
Once Part 1 of your program is functional, expand your code to partition the Employee records across 101 main-memory databases. Similar to Part 1, each main-memory database is configured with a B+-tree index on the Age attribute and a hash index on the Id attribute. However, this time, your program hash partitions the records across the databases by applying a mod function to the Id attribute of the Employee records ($Id \mod 101$). Once the main memory database is full, verify that all your inserted records do exist by retrieving them from all 101 databases. Next, delete half the records from one of the databases, say database 10. After verifying that the records have been deleted, answer the following questions. Are you able to insert new records into other databases? Are you able to insert records back into database 10? What if you deleted all the records from database 10: Are you able to insert records into other database? What if you deleted all the records from database 10 and closed it: Are you able to insert records into other databases?

Deliverables:
1. A brief report describing your observations from Part 1 and Part 2.
2. A zipped archive of your visual studio project(s) uploaded onto the blackboard system [http://den.usc.edu](http://den.usc.edu).

Due date: Feb 24, 2009.


Note: No late assignments are accepted.