B THE MINIBASE SOFTWARE

Practice is the best of all instructors.

—Publius Syrus, 42 B.C.

Minibase is a small relational DBMS, together with a suite of visualization tools, that has been developed for use with this book. While the book makes no direct reference to the software and can be used independently, Minibase offers instructors an opportunity to design a variety of hands-on assignments, with or without programming. To see an online description of the software, visit this URL:

`http://www.cs.wisc.edu/~dbbook/minibase.html`

The software is available freely through ftp. By registering themselves as users at the URL for the book, instructors can receive prompt notification of any major bug reports and fixes. Sample project assignments, which elaborate upon some of the briefly sketched ideas in the *project-based exercises* at the end of chapters, can be seen at

`http://www.cs.wisc.edu/~dbbook/minihwk.html`

Instructors should consider making small modifications to each assignment to discourage undesirable 'code reuse' by students; assignment handouts formatted using Latex are available by ftp. Instructors can also obtain solutions to these assignments by contacting the authors (`raghu@cs.wisc.edu`, `johannes@cs.cornell.edu`).

B.1 WHAT'S AVAILABLE

Minibase is intended to supplement the use of a commercial DBMS such as Oracle or Sybase in course projects, not to replace them. While a commercial DBMS is ideal for SQL assignments, it does not allow students to understand how the DBMS works. Minibase is intended to address the latter issue; the subset of SQL that it supports is intentionally kept small, and students should also be asked to use a commercial DBMS for writing SQL queries and programs. Minibase is provided on an as-is basis with no warranties or restrictions for educational or personal use. It includes the following:

- Code for a small single-user relational DBMS, including a parser and query optimizer for a subset of SQL, and components designed to be (re)written by students as project assignments: *heap files, buffer manager, B+ trees, sorting*, and *joins*.
- Graphical visualization tools to aid in students' exploration and understanding of the behavior of the *buffer management, B+ tree*, and *query optimization* components of the system. There is also a graphical tool to refine a relational database design using *normalization*.

B.2 OVERVIEW OF MINIBASE ASSIGNMENTS

Several assignments involving the use of Minibase are described below. Each of these has been tested in a course already, but the details of how Minibase is set up might vary at your school, so you may have to modify the assignments accordingly. If you plan to use these assignments, you are advised to download and try them at your site well in advance of handing them to students. We have done our best to test and document these assignments, and the Minibase software, but bugs undoubtedly persist. Please report bugs at this URL:

`http://www.cs.wisc.edu/~dbbook/minibase.comments.html`

I hope that users will contribute bug fixes, additional project assignments, and extensions to Minibase. These will be made publicly available through the Minibase site, together with pointers to the authors.

B.2.1 Overview of Programming Projects

In several assignments, students are asked to rewrite a component of Minibase. The book provides the necessary background for all of these assignments, and the assignment handout provides additional system-level details. The online HTML documentation provides an overview of the software, in particular the component interfaces, and can be downloaded and installed at each school that uses Minibase. The projects listed below should be assigned after covering the relevant material from the indicated chapter.

- **Buffer manager (Chapter 7):** Students are given code for the layer that manages space on disk and supports the concept of pages with page ids. They are asked to implement a buffer manager that brings requested pages into memory if they are not already there. One variation of this assignment could use different replacement policies. Students are asked to assume a single-user environment, with no concurrency control or recovery management.
- **HF page (Chapter 7):** Students must write code that manages records on a page using a slot-directory page format to keep track of records on a page. Possible

variants include fixed-length versus variable-length records and other ways to keep track of records on a page.

- **Heap files (Chapter 7):** Using the HF page and buffer manager code, students are asked to implement a layer that supports the abstraction of files of unordered pages, that is, heap files.
- **B+ trees (Chapter 9):** This is one of the more complex assignments. Students have to implement a page class that maintains records in sorted order within a page and implement the B+ tree index structure to impose a sort order across several leaf-level pages. Indexes store ⟨*key, record-pointer*⟩ pairs in leaf pages, and data records are stored separately (in heap files). Similar assignments can easily be created for Linear Hashing or Extendible Hashing index structures.
- **External sorting (Chapter 11):** Building upon the buffer manager and heap file layers, students are asked to implement external merge-sort. The emphasis is on minimizing I/O, rather than on the in-memory sort used to create sorted runs.
- **Sort-merge join (Chapter 12):** Building upon the code for external sorting, students are asked to implement the sort-merge join algorithm. This assignment can be easily modified to create assignments that involve other join algorithms.
- **Index nested-loop join (Chapter 12):** This assignment is similar to the sort-merge join assignment, but relies on B+ tree (or other indexing) code, instead of sorting code.

B.2.2 Overview of Nonprogramming Assignments

Four assignments that do not require students to write any code (other than SQL, in one assignment) are also available.

- **Optimizer exercises (Chapter 13):** The Minibase optimizer visualizer offers a flexible tool to explore how a typical relational query optimizer works. It accepts single-block SQL queries (including some queries that cannot be executed in Minibase, such as queries involving grouping and aggregate operators). Students can inspect and modify synthetic catalogs, add and drop indexes, enable or disable different join algorithms, enable or disable index-only evaluation strategies, and see the effect of such changes on the plan produced for a given query. All (sub)plans generated by an iterative System R style optimizer can be viewed, ordered by the iteration in which they are generated, and details on a given plan can be obtained readily. All interaction with the optimizer visualizer is through a GUI and requires no programming.

  The assignment introduces students to this tool and then requires them to answer questions involving specific catalogs, queries, and plans generated by controlling various parameters.

- **Buffer manager viewer (Chapter 12):** This viewer lets students visualize how pages are moved in and out of the buffer pool, their status (e.g., dirty bit, pin count) while in the pool, and some statistics (e.g., number of hits). The assignment requires students to generate traces by modifying some trace-generation code (provided) and to answer questions about these traces by using the visualizer to look at them. While this assignment can be used after covering Chapter 7, deferring it until after Chapter 12 enables students to examine traces that are representative of different relational operations.

- **B+ tree viewer (Chapter 9):** This viewer lets students see a B+ tree as it is modified through insert and delete statements. The assignment requires students to work with trace files, answer questions about them, and generate operation traces (i.e., a sequence of inserts and deletes) that create specified kinds of trees.

- **Normalization tool (Chapter 15):** The normalization viewer is a tool for normalizing relational tables. It supports the concept of a *refinement session*, in which a schema is decomposed repeatedly and the resulting decomposition tree is then saved. For a given schema, a user might consider several alternative decompositions (more precisely, decomposition trees), and each of these can be saved as a refinement session. Refinement sessions are a very flexible and convenient mechanism for trying out several alternative decomposition strategies. The normalization assignment introduces students to this tool and asks design-oriented questions involving the use of the tool.

Assignments that require students to evaluate various components can also be developed. For example, students can be asked to compare different join methods, different index methods, and different buffer management policies.

B.3 ACKNOWLEDGMENTS

The Minibase software was inpired by Minirel, a small relational DBMS developed by David DeWitt for instructional use. Minibase was developed by a large number of dedicated students over a long time, and the design was guided by Mike Carey and R. Ramakrishnan. See the online documentation for more on Minibase's history.