JAVA™ 2

SUN™ CERTIFIED PROGRAMMER & DEVELOPER

15

Database Issues

CERTIFICATION OBJECTIVE

- Understand Database Issues

CERTIFICATION OBJECTIVE

Understand Database Issues

Judge a man by his questions rather than his answers.

–Voltaire

A prudent question is one-half of wisdom.

–Francis Bacon

You're on your own for the other half.

–The Authors

As with the previous chapter, this chapter asks—you got it—questions. Some will seem obvious, some won't. But this is the area where your solution to the problem is going to have the greatest impact on your score. You're going to be asked to build a database. From scratch. And since there will be concurrent clients (or at least the *possibility* of concurrent clients), you'll have to be certain—dead certain—that you correctly manage record locking.

How you implement your searching, updating, and locking mechanism is entirely up to you. Again, there is definitely no One Right Answer for your solutions to these issues. But however you choose to do it, be certain that the logic is sound. For example, even if *you* never experience deadlock during testing, if there's even the slightest possibility (no matter how remote the chance) that it could happen, you could easily fail the exam even if nearly everything else in your application is perfect.

The two biggest issues are locking and searching, but locking is where the Big Money really is. We'll start with a brief overview of the key concepts, followed by yet another inspiring list of thought-provoking questions.

Building a Database

If you remember from Chapter 10, *you're* the one who has to build the database; the client's too cheap or neurotic to invest in a commercial database, even a free one. So what *is* a database? That depends on your assignment, but for the purposes of the exam, software-that-lets-you-access-a-set-of-records will do. You have some data, in some file format somewhere, with a known schema, and your job is to write an

application that allows that data to be searched and modified. You might also need to add and delete records.

So the concept is simple: the client makes a request, based on some search criteria, and your database returns a result. Sometimes the client might want to, say, book a Horse Cruise, in which case one or more records will have to be updated. And you might need to insert a new cruise or delete a cancelled cruise. Regardless of the actual scenario, the Really Big Issue is

How do I protect the data from concurrent access?

In other words, *how do I lock the records?*

exam Watch

Your locking design and implementation decisions (and execution) are the most important parts of your Developer assignment. Spend the greatest percentage of your time making sure you have a sound solution. Be sure you've met any requirements in your assignment document that pertain to locking and unlocking. If part of your assignment specification is vague or ambiguous, you need to make an interpretation (your best guess about what to do) and then document your assumption and strategy.

And remember, the clients could be either local or remote (in other words, on the same machine as the database or on a machine somewhere else on the network), so you'll have to think of issues related to *both* of those scenarios. Locking is crucial, but fortunately the Developer exam isn't asking you to implement a complete distributed transaction system using the two-phase commit protocol. In fact, this is much simpler than transactions, but it will require you to understand the fundamental issues surrounding concurrent access to data. Remember the bank account example from Chapter 9? The one where the husband and wife both shared the same account? If you're not absolutely clear about how to handle synchronization, then reread that chapter. In order to correctly implement your locking strategy, you're going to need a solid grasp on synchronization, `wait()`, `notify()`, and `notifyAll()`. So, ready for some questions? Once again, these are in no particular order.

Questions to Ask Yourself

We've split these into two categories, searching and locking. But there's a lot about searching that also falls into the category of GUI issues (Chapter 13). Specifically, you'll need to be certain that your end-users *know* how to build a search query, and that's discussed in Chapter 13.

Searching

- How *easy* is it for clients to perform a search? Assuming the GUI itself is user-friendly (and we have a lot to say about that in Chapter 13), what about the criteria?
- How are the search criteria items represented? A String? A CriteriaObject?
- How does a client know exactly what they can base a search on?
- Does your search support boolean matches? Does it need to?
- The database won't necessarily be indexed, so have you thought about other ways to make the search as efficient as possible?

exam Watch

Don't sacrifice clarity and simplicity for a small performance gain. If the performance gain is big, then redesign so that you can have a reasonably efficient algorithm that is also clear and maintainable.

- Have you documented your search algorithm?

exam Watch

If you find yourself writing a lot of documentation to explain your search algorithm, there's probably something wrong with the design.

- Is the documentation of your search algorithm easy to read and understand?
- When the client submits a search query, is a specific piece of the search criteria explicitly matched to a particular field? Or do you search all fields for each search?
- If you're using 1.4, have you investigated whether regular expressions would help?
- What happens if nothing matches the client's search criteria?
- Will it need to be an *exact* match?
- Could there be a scenario in which too many records match the search criteria?
- Have you considered bandwidth issues when designing and implementing the format of the search criteria requests and server results? Are you shipping things over the wire that are bigger than they need to be?
- Is your search capability flexible for the end-user?

- Is your search capability flexible for future changes to the program?
- How much code, if any, would have to change if the database schema changes? Have you isolated the places where changes can occur to avoid maintenance problems?
- Are you absolutely certain that you've met the searching requirements defined in your assignment specification? Go back and reread them. Slooooooooowly.

Locking

- Are you absolutely certain that your locking scheme works in all possible scenarios?
- Does your exam assignment specify a particular kind of locking with respect to reads and writes?
- What happens when a client attempts to get a record and the record is already locked? What does the client experience?

This is crucial. Think long and hard about what you want to happen.

- How will you keep track of which records are locked?
- How will you keep track of *who* locked each record? Do you need to know that?
- How will you uniquely identify clients in such a way that you *can* know which client locked which record? Is it the server's responsibility or the client's?
- Have you considered whether the ID of a thread is appropriate to uniquely identify a client?
- Have you considered whether a `Math.random()` number is appropriate to uniquely identify a client?
- If a client makes a request on a locked record, how will you verify that it's the same client who holds the lock?
- What happens if a client attempts to use a locked record when that client is *not* the client holding the lock?

- Is it possible to have a record locked for too long a time? How much time is *too long*?
- Is there anything you can or should do about the duration of a lock?
- What happens if a client goes down without releasing a lock?
- Does the server need a way to *know* a client went down? (As opposed to simply taking their sweet time or if they're on a painfully slow connection.)
- Is there any possibility of a deadlock? Where two or more clients are waiting for each other's locks?

exam Watch

Check for this more than you check for anything else.

- Are you correctly using `wait()`, `notify()`, and `notifyAll()`?
- Are you clear about the implications of `notify()` versus `notifyAll()`?

exam Watch

If not, go back and read Chapter 9.

- Are you relying on a nondeterministic thread mechanism such as priorities and/or yielding to guarantee your threads will behave properly?
- Are you synchronizing on the right objects?
- Are you *sure?*
- *Are you really really really sure?*
- Is everything that needs to be thread-safe, thread-safe?
- Have you made sure that things that don't need to be thread-safe, *aren't*? (You already know that synchronization carries a performance hit.)
- Have you selected appropriate data structures for implementing your lock scheme?
- Are you absolutely certain that you've met the locking requirements defined in your assignment specification?
- Would you like to revise your answers to the last two questions from Chapter 14?