```
var databaseCategory = db.categories.findOne( { _id: "Databases" } );
db.categories.find( { left: { $gt: databaseCategory.left }, right: { $lt: databaseCategory.right } } )
```

The *Nested Sets* pattern provides a fast and efficient solution for finding subtrees but is inefficient for modifying the tree structure. As such, this pattern is best for static trees that do not change.

4.3.3 Model Specific Application Contexts

Model Data for Atomic Operations **(page 164)** Illustrates how embedding fields related to an atomic update within the same document ensures that the fields are in sync.

Model Data to Support Keyword Search **(page 165)** Describes one method for supporting keyword search by storing keywords in an array in the same document as the text field. Combined with a multi-key index, this pattern can support application's keyword search operations.

Model Monetary Data **(page 166)** Describes two methods to model monetary data in MongoDB.

Model Time Data **(page 168)** Describes how to deal with local time in MongoDB.

Model Data for Atomic Operations

Pattern

In MongoDB, write operations, e.g. `db.collection.update()`, `db.collection.findAndModify()`, `db.collection.remove()`, are atomic on the level of a single document. For fields that must be updated together, embedding the fields within the same document ensures that the fields can be updated atomically.

For example, consider a situation where you need to maintain information on books, including the number of copies available for checkout as well as the current checkout information.

The available copies of the book and the checkout information should be in sync. As such, embedding the `available` field and the `checkout` field within the same document ensures that you can update the two fields atomically.

```
{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly",
    available: 3,
    checkout: [ { by: "joe", date: ISODate("2012-10-15") } ]
}
```

Then to update with new checkout information, you can use the `db.collection.update()` method to atomically update both the `available` field and the `checkout` field:

```
db.books.update (
   { _id: 123456789, available: { $gt: 0 } },
   {
     $inc: { available: -1 },
     $push: { checkout: { by: "abc", date: new Date() } }
   }
)
```