CHAPTER 12

Internal Query Representation

This chapter presents the first part of the advanced database technologies for the database-experiment project (DBXP). I begin by introducing the concept of the query tree structure, which is used for storing a query in memory. Next, I’ll present the query tree structure used for the project along with the first in a series of short projects for implementing the DBXP code. The chapter concludes with a set of exercises you can use to learn more about MySQL and query trees.

The Query Tree

A query tree is a tree structure that corresponds to a query, in which leaf nodes of the tree contain nodes that access a relation and internal nodes with zero, one, or more children. The internal nodes contain the relational operators. These operators include project (depicted as \(\pi\)), restrict (depicted as \(\sigma\)), and join (depicted as either \(\theta\) or \(\bowtie\)). The edges of a tree represent data flow from bottom to top—that is, from the leaves, which correspond to reading data in the database, to the root, which is the final operator producing the query results. Figure 12-1 depicts an example of a query tree.

```
SELECT name
FROM faculty f, classes c
WHERE f.id = c.fac_id AND 
f.department_id = 'CS' AND c.semester = 'F2001'
```

![Figure 12-1. An Example Query Tree](image)

Strangely, few texts give explanations for the choice of symbol. Traditionally, \(\theta\) represents a theta-join and \(\bowtie\) represents a natural join, but most texts interchange these concepts, resulting in all joins represented using one or the other symbol (and sometimes both).

Although similar drawings have appeared in several places in the literature, Figure 12-1 contains a subtle nuance of database theory that is often overlooked. Can you spot the often-misused trait? Hint: What is the domain of the semester attribute? Which rule has been violated by encoding data in a column?
An evaluation of the query tree consists of evaluating an internal node operation whenever its operands are available and passing the results from evaluating the operation up the tree to the parent node. The evaluation terminates when the root node is evaluated and replaced by the tuples that form the result of the query. The following sections present a variant of the query tree structure for use in storing representations of queries in memory. The advantages of using this mechanism versus a relational calculus internal representation are shown in Table 12-1.

### Table 12-1. Advantages of Using a Query Tree vs. Relational Calculus

<table>
<thead>
<tr>
<th>Operational Requirement</th>
<th>Query Tree</th>
<th>Relational Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can it be reduced?</td>
<td>Yes. It is possible to prune the query tree prior to evaluating query plans.</td>
<td>Only through application of algebraic operations.</td>
</tr>
<tr>
<td>Can it support execution?</td>
<td>Yes. The tree can be used to execute queries by passing data up the tree.</td>
<td>No. Requires translation to another form.</td>
</tr>
<tr>
<td>Can it support relational algebra expressions?</td>
<td>Yes. The tree lends itself well to relational algebra.</td>
<td>No. Requires conversion.</td>
</tr>
<tr>
<td>Can it be implemented in database systems?</td>
<td>Yes. Tree structures are a common data structure.</td>
<td>Only through designs that model the calculus.</td>
</tr>
<tr>
<td>Can it contain data?</td>
<td>Yes. The tree nodes can contain data, operations, and expressions.</td>
<td>No. Only the literals and variables that form the expression.</td>
</tr>
</tbody>
</table>

Clearly, the query-tree internal representation is superior to the more traditional mechanism employed in modern database systems. For example, the internal representation in MySQL is that of a set of classes and structures designed to contain the query and its elements for easy (fast) traversal. It organizes data for the optimization and execution.³

There are some disadvantages to the query-tree internal representation. Most optimizers are not designed to work within a tree structure. If you wanted to use the query tree with an optimizer, the optimizer would have to be altered. Similarly, query execution will be very different from most query-processing implementations. In this case, the query execution engine will be running from the tree rather than as a separate step. These disadvantages are addressed in later chapters as I explore an alternative optimizer and execution engine.

The DBXP query tree is a tree data structure that uses a node structure that contains all of the parameters necessary to represent these operations:

- **Restriction**: Allows you to include results that match an expression of the attributes.
- **Projection**: Provides the ability to select attributes to include in the result set.
- **Join**: Lets you combine two or more relations to form a composite set of attributes in the result set.
- **Sort (order by)**: Allows you to order the result set.
- **Distinct**: Provides the ability to reduce the result set to unique tuples.

**Note** Distinct is an operation that is added to accomplish a relational operation that isn’t supported by most SQL implementations and is not an inherent property of relational algebra.

³Some would say it shouldn’t have to, as the MySQL internal structure is used to organize the data for the optimizer. Query trees, on the other hand, are designed to be optimized and executed in place.