3 Three Levels of Abstraction

3.1 Introduction

In order to understand the TR approach to implementing the relational model, it's necessary to be very clear over three distinct levels of the system, which I'll refer to as *the three levels of abstraction* (since each level is an abstraction of the one below, loosely speaking). The three levels, or layers, are:

1. The relational (or user) level
2. The file level
3. The TR level

They're illustrated in Fig. 3.1. In a nutshell:

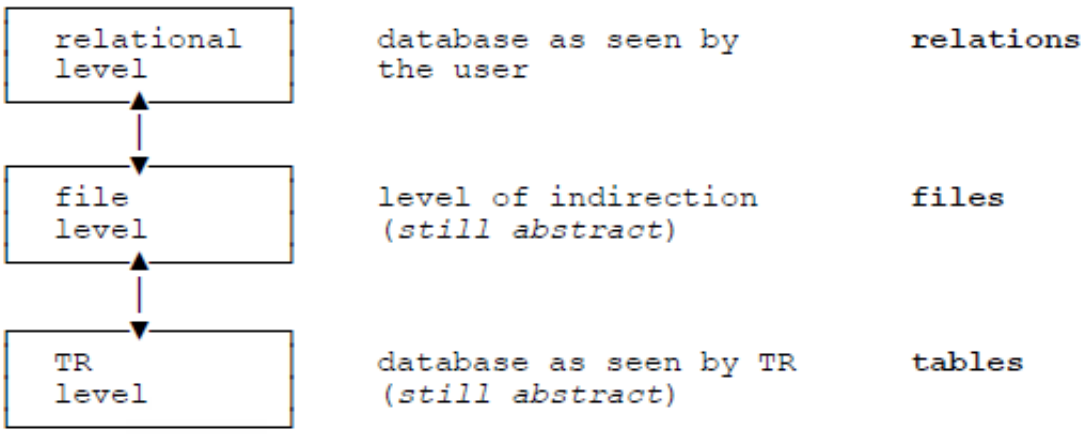

Fig. 3.1: The three levels of abstraction

- Level 1, which corresponds to the database as seen by the user, is the relational level. At this level, the data is perceived as **relations,** including, perhaps, the suppliers relation S discussed in Section 2.1 (and illustrated in Fig. 2.1) in the previous chapter.

- Level 3 is the fundamental TR implementation level. At this level, data is represented by means of a variety of internal structures called **tables**. *Please note immediately that those TR tables are NOT tables in the SQL sense and do NOT correspond directly to relations at the user level.*

- Level 2 is a level of indirection between the other two. Relations at the user or relational level are mapped to **files** at this level, and those files are then mapped to tables at the TR level. Of course, the mappings go both ways; that is, tables at the TR level map to files at the next level up, and those files then map to relations at the top level. *Note:* As I'm sure you know, *map* is a synonym for *transform* (and I'll be using the term in that sense throughout this book); thus, we're already beginning to touch on the TR transforms that were mentioned in Chapter 1. However, there's a great deal more to it, as we'll soon see.

Please now observe that each level has its own terminology: relational terms at the user level, file terms at the file level, and table terms at the TR level. Using different terms should, I hope, help you keep the three levels distinct and separate in your mind; for that reason, I plan to use the three sets of terms consistently and systematically throughout the rest of this book.

Having said that, I now need to say too that I'm well aware that some readers might object to my choice of terms—perhaps even find them confusing—for at least the following two reasons:

- First, the industry typically uses the terminology of tables, not relations, at the user level—almost exclusively so, in fact. But I've already explained some of my rationale for wanting to use relational terms at that level (see the previous chapter, Section 2.1), and I'm going to give some additional reasons in the next section.

- Second, the industry also typically tends to think of files as a fairly "physical" construct. In fact, I did the same thing myself in the previous chapter, somewhat, though I was careful in that chapter always to be quite clear that the files I was talking about were indeed *physically stored* files specifically. By contrast, the files I'll be talking about in the rest of the book are *not* physically stored; instead, they're an abstraction of what's physically stored, and hence a "logical" construct, not a physical one. (Though it wouldn't be wrong to think of them as "slightly more physical" than the user-level relations, if you like.)

If you still think my terms are confusing, then I'm sorry, but for better or worse they're the terms I'm going to use.

One final point: When I talk of three levels, or layers, of abstraction, I don't mean that each of those levels is physically materialized in any concrete sense—of course not. The relational level is only a way of looking at the file level, a way in which certain details are ignored (that's what "level of abstraction" *means*). Likewise, the file level in turn is only a way of looking at the TR level. Come to that, the TR level in turn is only a way of looking at the bits and bytes that are physically stored; that is, the TR level is itself—as already noted in Chapter 1, Section 1.2—still somewhat abstract. In a sense, the bits-and-bytes level is the *only* level that's physically materialized.[1]

3.2 The Relational Level

Since the focus of this book is on the use of TR technology to implement the relational model specifically, the topmost (user) level is relational by definition. In other words, the user sees the database as a set of **relations,** made up of **attributes** and **tuples** as explained in Chapter 2. For simplicity, I'm going to assume those relations are all **base** relations specifically (again, see Chapter 2); that is, I'll simply assume, barring explicit statements to the contrary, that any relation that's named and is included in the database is in fact a base relation specifically, and I won't usually bother to use the "base" qualifier.

Also, of course, the user at the relational level has available a set of **relational operators**—restrict, project, join, and so forth—for querying the relations in the database, as well as the usual INSERT, DELETE, and UPDATE operators for updating them. *Note:* If I wanted to be more precise here, I'd have to get into the important distinction between relation **values** and relation **variables**. Relational operators like join operate on relation *values,* while update operators like INSERT operate on relation *variables*. Informally, however, it's usual to call them all just relations, and—somewhat against my better judgment—I've decided to follow that common usage (for the most part) in the present book. For further discussion of such matters, see either reference [32] or reference [40].

Now, given the current state of the IT industry, the user level in a real database system will almost certainly be based on SQL, not on the relational model. As a consequence, users will typically tend to think, not in terms of relational concepts as such, but rather in terms of SQL analogs of those concepts. For example, there isn't any explicit project operator, as such, in SQL; instead, such an operation has to be formulated in terms of SQL's SELECT and FROM operators, and the user has to think in terms of those SQL operators, as in this example ("Project suppliers over supplier number and city name"):

```
SELECT S.S#, S.CITY
FROM S ;
```

Precisely because most of today's database systems are in fact SQL systems specifically, I'll show most of my examples in what follows in SQL, not in pure relational form. But I do still want to use the terms *relation, tuple,* and *attribute* at the user level (sometimes *user* relations, tuples, and attributes, for emphasis), instead of the more familiar SQL terms *table, row,* and *column,* and—as I promised I would, both in Chapter 2 and in the previous section—I'd like to give my reasons for adopting this perhaps rather purist or academic position. In essence, it seems to me that to use the SQL terms would lead to at least three problems:

- First of all, we're going to need to use the terminology of tables, rows, and columns—as we very often do when discussing software internals—at the implementation level (which is to say the TR level), and the TR and SQL constructs are, as already noted, completely different things. So there would be an obvious potential for confusion right away.

- Second, the SQL terminology tends to obscure the crucial distinction alluded to above between *relation values* and *relation variables*. (SQL doesn't clearly distinguish between these concepts at all, referring to them both simply as *tables,* a state of affairs that has demonstrably led to some confusion in the past.)

- Third, considered as possible user-level terms, *table, row,* and *column* are in fact actively misleading (indeed, I wish we'd never used them, not even in SQL), for at least the following reasons:

    - They lend weight to the "duplicate tuples" heresy. In fact, an SQL table can have duplicate rows, although as we know a relation can't have duplicate tuples. *Note*: The TR model itself doesn't care whether there are duplicates or not, and hence can support SQL's nonrelational tables as well as proper relations—but I don't propose to discuss that nonrelational support in any detail in this book.

    - They suggest there's a top-to-bottom ordering to the rows, though in fact there isn't.

    - They suggest there's a left-to-right ordering to the columns. (In fact there is, in SQL—another departure from the relational model, as noted in Chapter 1.)

    - They suggest that "row-and-column intersections" in those tables can be accessed via [*i,j*]-style subscripting, instead of associatively. That is, tables—but definitely not relations—are often thought of as being something like *arrays* (two-dimensional arrays, to be precise). *Note:* The term "associatively" refers to the fact that data at the relational level is accessed by value, not by address. For example, "Get tuples for suppliers in London" is a relational request, but "Get the first and fourth supplier tuples" isn't. Likewise, "Get status values from tuples for suppliers in Paris" is a relational request, but "Get values of the third attribute from tuples for suppliers in Paris" isn't.

    - Most significantly, they tend to obscure the important connections between the relational model and mathematics and logic. (Those connections are important because they're what make it possible to treat database management as a science; without them, the field becomes a mere ragbag of ad hoc tricks, techniques, and rules of thumb.)

This isn't an exhaustive list.[2]

3.3 The File Level

The first step, conceptually speaking, in mapping a given relation to an appropriate TR representation is to convert that relation into a **file,** with **records** corresponding to the tuples and **fields** corresponding to the attributes. For example, Fig. 3.2 shows a possible file corresponding—in a trivially obvious way—to the suppliers relation of Fig. 2.1 in Chapter 2.

field sequence: 1 2 3 4

record sequence:	S#	SNAME	STATUS	CITY
1	S1	Smith	20	London
2	S2	Jones	10	Paris
3	S3	Blake	30	Paris
4	S4	Clark	20	London
5	S5	Adams	30	Athens

Fig. 3.2: A file corresponding to the suppliers relation of Fig. 2.1

Within such a file, records do have a top-to-bottom ordering and fields do have a left-to-right ordering, as the record numbers and field numbers in the figure are meant to suggest.[3] However, the orderings in question are essentially arbitrary; thus, for example, the suppliers relation of Fig. 2.1 could map equally well to any of 2,880 different files (120 different orderings for the five records and 24 different orderings for the four fields). By way of illustration, Fig. 3.3 shows another possible file corresponding to the suppliers relation of Fig. 2.1.

field sequence: 1 2 3 4

record sequence:	STATUS	S#	CITY	SNAME
1	20	S4	London	Clark
2	30	S5	Athens	Adams
3	10	S2	Paris	Jones
4	20	S1	London	Smith
5	30	S3	Paris	Blake

Fig. 3.3: Another file corresponding to the suppliers relation of Fig. 2.1

Of course, those 2,880 different files are all equivalent to one another, in the sense that they all represent exactly the same information; in other words, they're all **information-equivalent**. It's sometimes convenient, therefore, to regard them, not so much as 2,880 distinct files as such, but rather as 2,880 different versions of "the same" file.[4] This perception will turn out to be important in the next chapter—also, especially, in Chapter 7.

Files, records, and fields (sometimes *user* files, records, and fields for emphasis, since in many respects the file level is still quite close to the user or relational level) can be operated upon by obvious counterparts to the operators available at the relational level. Also, reconstructing the corresponding relation from a given file (any version) is trivial: Just ignore the orderings.

Files such as those shown in Figs. 3.2 and 3.3 can now be represented by tables at the TR level and can be reconstructed from those TR tables. In fact (important!), many different versions of the same file can all be reconstructed from the same TR tables equally easily (using the term "versions" in the special sense explained above—that is, record and field orderings might be different, but content remains the same). We'll see how this works out in the next chapter.

One last point: I've called this level the *file* level and the next more specifically the *TR* level because most of the ingenuity, inventiveness, and novelty of the TR model is to be found at that next level. However, the file level too is part of the overall TR implementation approach, of course.

3.4 The TR Level

Files at the file level map to **tables** at the TR level, and those tables are made up of **rows** and **columns**. Like records and fields within files, rows in a TR table do have a top-to-bottom ordering and columns in such a table do have a left-to-right ordering. And, very importantly, a row-and-column intersection within such a table, which I'll refer to as a **cell,** can be addressed via [*i,j*]-style subscripting (where *i* is the row number and *j* is the column number); in other words, TR tables, unlike SQL tables, can legitimately, and usefully, be thought of as *two-dimensional arrays*. Cells in such a table or array contain **values**. What's more, those values can sometimes be composite; for example, a given cell might contain an ordered pair of pointer values, and an ordered pair of values can certainly be regarded as a value—a composite value—in its own right.

Now, the mapping of files to TR tables is quite a complex business, and I don't want to start getting into details of how it's done until the next chapter. Suffice it to say that it's nothing like the direct-image kind of mapping discussed in Chapters 1 and 2. In particular, rows in TR tables do *not* correspond in any one-to-one kind of way to records at the file level, nor a fortiori do they correspond in any one-to-one kind of way to tuples at the relational level. By way of illustration, Fig. 3.4 shows a TR table, the **Field Values Table,** corresponding to the file of Fig. 3.2. As I've already indicated, I don't want to get into details yet of just how that table is obtained from that file, but you might like to try to figure it out for yourself (it's not very difficult). All I want to do now is draw your attention to the fact that indeed, as claimed, the rows don't correspond in any obvious way to the records shown in Fig. 3.2.

column sequence:		*1*	*2*	*3*	*4*
		S#	SNAME	STATUS	CITY
row sequence:	*1*	S1	Adams	10	Athens
	2	S2	Blake	20	London
	3	S3	Clark	20	London
	4	S4	Jones	30	Paris
	5	S5	Smith	30	Paris

Fig. 3.4: Field Values Table corresponding to the file of Fig. 3.2

In order to be able to reconstruct the file of Fig. 3.2 from the Field Values Table of Fig. 3.4, we need another table, the **Record Reconstruction Table**.[5] Again, I don't want to get into details yet of how the Record Reconstruction Table is obtained, nor how it's used in the reconstruction process; I'll just show, in Fig. 3.5, a possible Record Reconstruction Table corresponding to the file shown in Fig. 3.2 (and to the Field Values Table shown in Fig. 3.4)—and point out that the entries in the Record Reconstruction Table aren't supplier numbers or status values, etc., any longer (despite the column labels) but are **row numbers** instead. For further explanation, see the next chapter.

column sequence:		*1*	*2*	*3*	*4*
		S#	SNAME	STATUS	CITY
row sequence:	*1*	*5*	*5*	*4*	*5*
	2	*4*	*4*	*2*	*1*
	3	*2*	*3*	*3*	*4*
	4	*3*	*1*	*5*	*2*
	5	*1*	*2*	*1*	*3*

Fig. 3.5: Record Reconstruction Table corresponding to file of Fig. 3.2 (and Field Values Table of Fig. 3.4)

By the way, it's a little misleading to talk (as I've just been doing) in terms of *the* Field Values Table and *the* Record Reconstruction Table, because there'll probably be many such tables in any real implementation—one of each for each file at the file level, loosely speaking (but see Chapters 9 and 11-14 later). However, it's much easier to talk in terms of, for example, "*the* Field Values Table" instead of having to say something like "the particular Field Values Table that corresponds to the file of Fig. 3.2" every time we need to refer to such a thing. So I'll continue to talk this way for most of the rest of this book, and hope you won't find the practice confusing.

I'll be discussing what's involved in building and using the Field Values Table and the Record Reconstruction Table in the next few chapters. For now, let me close by stressing a point I've made a couple of times already: namely, that the TR level, though obviously at a much lower level of detail than the relational level, is nevertheless still abstract. In fact, TR is a *model* in the sense of Chapter 1, meaning it can be regarded as a layer of abstraction over something deeper down. In particular, the TR tables discussed above, and their associated operators, can be physically implemented in a variety of different ways, some of which I'll be talking about in later chapters. Very importantly, of course, they can be implemented in either main memory or secondary storage; indeed, they can be implemented on absolutely any hardware platform whatsoever, from a handheld or palmtop computer, to a laptop or desktop machine, to a mainframe, to a client/server or other distributed system, to the most massively parallel supercomputer. Now, this book is primarily concerned with the TR model as such, not so much with specific implementations of that model; however, Part III does specifically address the question of a disk-based implementation, since there are clearly special issues to be addressed in such an environment. By contrast, Part II doesn't assume any particular implementation environment at all (at least, not explicitly); however, you can think of it for the most part as implicitly assuming a main-memory environment, if you find it helpful to do so.

Endnotes

1. Well ... to be pedantic about it, those bits and bytes are an abstraction too, of course, and so on, all the way down to the level of electrons (and beyond!). But bits and bytes are physical enough for our purposes.
2. In particular, I'd like to point out that certain very important relational "tables"—namely, the ones that references [12] and [24] call TABLE_DEE and TABLE_DUM—don't have any "columns" anyway. (The analogy between relations and tables breaks down here.) Further discussion of this particular issue would take us much too far afield, however; if you're intrigued and want to know more, see either reference [32] or reference [40].
3. In practice, like the record numbers discussed in Chapter 2, those record and field numbers probably won't be simple sequential numbers as shown in the figure. The same is true for row and column numbers at the TR level (see the next section).
4. Incidentally, note that those different versions can't all be obtained by means of a simple ORDER BY.
5. I could logically have called this table the *File* Reconstruction Table, but I wanted to emphasize the point that it can be used to reconstruct individual records of the file as well as the file in its entirety. In fact, it can be used to reconstruct any subset of the records in that file, and any subset of the fields in those records, as we'll see in the course of the next few chapters.