9 Merged Columns

9.1 Introduction

Now I want to turn to yet another very important refinement to the basic TR model, **merged columns**. In the previous chapter, I discussed condensed columns, which can be characterized as a way of sharing field values across records—but the records in question all had to come from the same file. Merged columns, by contrast, can be characterized as a way of sharing field values across records, where the records in question might or might not all come from the same file.[1] I'll consider two examples, the first involving just one file, the second involving two.

Note: Columns can be merged without necessarily having to be either sorted or condensed, but the general idea of merged columns makes much more sense if the columns in question are both. In what follows, I'll assume that merged columns are indeed always both, barring explicit statements to the contrary. In practice, in fact, it's hard to imagine a column being merged without being both sorted and condensed as well.

9.2 The Bill-of-Materials Example

Essentially, the basic idea underlying merged columns is that distinct fields at the file level can map to the same Field Values Table column at the TR level (just so long as the fields in question are of the same data type, of course). For example, consider the bill-of-materials relation MMQ depicted in Fig. 9.1. That relation is meant to be interpreted as follows: The indicated "major" part (MAJOR_P#) includes the indicated "minor" part (MINOR_P#) in the indicated quantity (QTY); that is, the minor part is a component of the major part, and it takes the specified quantity of the minor part to make the major part. For example, it takes four P6's (among other things) to make one P3. The attribute combination {MAJOR_P#,MINOR_P#} is the sole key; attributes MAJOR_P# and MINOR_P# are both defined on type P#, and attribute QTY is defined on type INTEGER.

MAJOR_P#	MINOR_P#	QTY
P1	P2	2
P1	P3	4
P1	P4	1
P2	P3	3
P2	P4	8
P2	P5	6
P3	P4	3
P3	P6	4
P5	P6	3

Fig. 9.1: The bill-of-materials relation MMQ

In what follows, I'll first consider what happens in this example without merged columns, and then take a look at how the situation changes if we apply the merged-column refinement. Fig. 9.2, then, shows a possible file corresponding to the relation of Fig. 9.1. Note that I've deliberately shuffled the record ordering around, purely to make later parts of the discussion a little more interesting. (If we stick to the "obvious" ordering as suggested by Fig. 9.1, it turns out that too many coincidences occur in, for example, the Record Reconstruction Table, coincidences that suggest the existence of certain intrinsic relationships that don't in fact exist.) Fig. 9.3 shows the corresponding *uncondensed* Field Values Table, and Fig. 9.4 shows a corresponding Record Reconstruction Table, based on the following permutations:

- MAJOR_P# - MINOR_P# - QTY : `7, 2, 4, 9, 3, 5, 1, 6, 8`
- MINOR_P# - MAJOR_P# - QTY : `7, 2, 9, 4, 3, 1, 5, 6, 8`
- QTY - MAJOR_P# - MINOR_P# : `4, 7, 9, 1, 8, 2, 6, 5, 3`

Note: In the first two permutations, attribute QTY is irrelevant, because the two leading attributes constitute a key. In the third permutation, the choice of MAJOR_P# - then - MINOR_P# over MINOR_P# - then - MAJOR_P# is arbitrary on my part. **Exercise 13:** Confirm for yourself that Figs. 9.3 and 9.4 are correct.

	1	2	3
	MAJOR_P#	MINOR_P#	QTY
1	P3	P4	3
2	P1	P3	4
3	P2	P4	8
4	P1	P4	1
5	P2	P5	6
6	P3	P6	4
7	P1	P2	2
8	P5	P6	3
9	P2	P3	3

Fig. 9.2: File corresponding to the bill-of-materials relation of Fig. 9.1

	1	2	3
	MAJOR_P#	MINOR_P#	QTY
1	P1	P2	1
2	P1	P3	2
3	P1	P3	3
4	P2	P4	3
5	P2	P4	3
6	P2	P4	4
7	P3	P5	4
8	P3	P6	6
9	P5	P6	8

Fig. 9.3: Uncondensed Field Values Table corresponding to the file of Fig. 9.2

	1	2	3
	MAJOR_P#	MINOR_P#	QTY
1	1	2	3
2	2	6	1
3	4	3	4
4	3	1	7
5	5	9	9
6	7	4	2
7	6	8	8
8	8	7	6
9	9	5	5

Fig. 9.4: Record Reconstruction Table corresponding to the file of Fig. 9.2

Fig. 9.5 now shows a condensed version of the Field Values Table from Fig. 9.3, and Fig. 9.6 shows a corresponding expanded Record Reconstruction Table. Again, I recommend strongly that you confirm for yourself that these tables are correct (**Exercise 14**). Perhaps I should remind you that:

	1	2	3
	MAJOR_P#	MINOR_P#	QTY
1	P1 [1:3]	P2 [1:1]	1 [1:1]
2	P2 [4:6]	P3 [2:3]	2 [2:2]
3	P3 [7:8]	P4 [4:6]	3 [3:5]
4	P5 [9:9]	P5 [7:7]	4 [6:7]
5		P6 [8:9]	6 [8:8]
6			8 [9:9]

Fig. 9.5: Condensed version of the Field Values Table from Fig. 9.3

	1	2	3
	MAJOR_P#	MINOR_P#	QTY
1	1▪1	1▪2	1▪3
2	1▪2	2▪6	2▪1
3	1▪4	2▪3	3▪4
4	2▪3	3▪1	3▪7
5	2▪5	3▪9	3▪9
6	2▪7	3▪4	4▪2
7	3▪6	4▪8	4▪8
8	3▪8	5▪7	5▪6
9	4▪9	5▪5	6▪5

Fig. 9.6: Expanded version of the Record Reconstruction Table from Fig. 9.4

- In the case of the Field Values Table (Fig. 9.5), the numbers in brackets represent *row ranges*. For example, the row range [*4*:*6*] in cell [*3,2*] indicates that the corresponding field value—namely, part number P4—appears in rows *4, 5,* and *6* of the corresponding uncondensed Field Values Table (all in column *2,* of course).

- In the case of the Record Reconstruction Table (Fig. 9.6), the two numbers in each cell are both pointers (row numbers); the first refers to a row of the Field Values Table, the second refers to a row of the Record Reconstruction Table itself. For example, cell [*7,2*] contains the entry *4*▪*8*. The *4* means the relevant field value—namely, part number P5—is to be found in cell [*4,2*] of the Field Values Table. The *8* means the next cell to be inspected in the Record Reconstruction Table is cell [*8,3*].

Now (at last) we can start our examination of merged columns. Going right back to the user-level relation MMQ, it's clear that attributes MAJOR_P# and MINOR_P# are of the same data type (they're both of type P#, in fact), and hence that fields MAJOR_P# and MINOR_P# of the corresponding file are of the same data type, too. They can therefore be mapped to the same column of the Field Values Table. Fig. 9.7 shows what happens. Note the following points:

	1	*2*
	MAJOR_P# + MINOR_P#	QTY
1	P1 [*1*:*3*] [:]	1 [*1*:*1*]
2	P2 [*4*:*6*] [*1*:*1*]	2 [*2*:*2*]
3	P3 [*7*:*8*] [*2*:*3*]	3 [*3*:*5*]
4	P4 [:] [*4*:*6*]	4 [*6*:*7*]
3	P5 [*9*:*9*] [*7*:*7*]	6 [*8*:*8*]
4	P6 [:] [*8*:*9*]	8 [*9*:*9*]

Fig. 9.7: Field Values Table of Fig. 9.5 after merging the first two columns

- Columns MAJOR_P# and MINOR_P# have been merged into a single column. In the figure, I've labeled the resulting column, not very elegantly, "MAJOR_P# + MINOR_P#."

- The merged column contains all of the field values—part numbers, to be specific—that previously appeared in either column MAJOR_P# or column MINOR_P# in the table before merging. Duplicates have been eliminated.[2]

- Each cell in the merged column thus contains a single part number, together with *two* row ranges: The first indicates which rows of the uncondensed Field Values Table (see Fig. 9.3) the corresponding part number appears in as a major part number; the second indicates which rows of that uncondensed Field Values Table the corresponding part number appears in as a minor part number.

- Note that those row ranges are basically the same as they were in the previous version of the Field Values Table, except for occasional appearances of the special **empty** row range "[:]." The empty range is used when the indicated part number doesn't appear at all in the corresponding column of the uncondensed Field Values Table; for example, P1 never appears as a minor part number.

  I remark in passing, without going into details, that certain further refinements can usefully be applied to the Field Values Table if empty ranges are either particularly common or particularly rare. The refinements in question have the effect of saving storage space and speeding up searches (in the "common" case), or simplifying the task of finding the entries with empty ranges (in the "rare" case). For more details, see reference [63].

- In the merged table, the merged column is column *1* and the QTY column is column *2* (after all, the table does now have just two columns, not three). Column *2,* the QTY column, is the same as it was in Fig. 9.5. *Note:* From this point forward, I'll use the term "merged table" to mean any Field Values Table that includes at least one merged column.

Fig. 9.8 shows the corresponding Record Reconstruction Table. Note the following points:

	1	2	3
	MAJOR_P#	MINOR_P#	QTY
1	1▪1	2▪2	1▪3
2	1▪2	3▪6	2▪1
3	1▪4	3▪3	3▪4
4	2▪3	4▪1	3▪7
5	2▪5	4▪9	3▪9
6	2▪7	4▪4	4▪2
7	3▪6	5▪8	4▪8
8	3▪8	6▪7	5▪6
9	5▪9	6▪5	6▪5

Fig. 9.8: Expanded Record Reconstruction Table corresponding to the merged Field Values Table of Fig. 9.7

- The Record Reconstruction Table still has three columns. However, columns *1* and *2* of that table now both correspond to column *1* (the merged column) of the Field Values Table; column *1* refers to the first row range in that merged column and column *2* to the second. Column *3* of the Record Reconstruction Table now refers to column *2* of the Field Values Table. These facts will obviously have to be taken into account when doing record or file reconstruction using the Record Reconstruction Table (see below).

- The algorithm for building the Record Reconstruction Table remains essentially unchanged. However, the table itself that results from executing that algorithm is *not* unchanged. To be specific, if you compare Figs. 9.8 and 9.6, you'll see that the last entry in column *1* and all of the entries in column *2* have changed, in that the first of the two row numbers—the one that refers to the Field Values Table—has increased by one in every case. This change is a result of the appearance of the aforementioned empty ranges in the merged Field Values Table.

Another strong recommendation (again you've probably already guessed this one): Try using the Record Reconstruction Table of Fig. 9.8, together with the Field Values Table of Fig. 9.7, to reconstruct a corresponding file. If you work down column *1* of the Record Reconstruction Table, you should wind up with a file that's a direct image of relation MMQ as shown in Fig. 9.1 (this is **Exercise 15**).

I'll finish up this section with a brief discussion of certain significant implications of the merged-columns idea. First, it obviously saves space. Suppose for the sake of the example that part numbers and quantities require four bytes each, while row numbers require two bytes each. Suppose too, realistically enough, that each row range is represented by a begin point only.[3] Then the unmerged Field Values Table of Fig. 9.5 would occupy a total of 90 bytes, while that of Fig. 9.7 would occupy a total of 78 bytes—a 13.3 percent reduction.

The next point is much more important. It has to do with join operations. Suppose we want to join relation MMQ to itself, matching minor part numbers in "the first copy" (as it were) of the relation with major part numbers in "the second copy." Such a join is very likely in practice, by the way; it's needed, for example, in computing the result of the well-known *part explosion* query "Get all components, at all levels, of some given part." Well, we can tell *in a single pass* over the merged

Field Values Table just which tuples join to which! For example, row *3* of that table (which contains the part number P3) shows a minor part number row range of [*2*:*3*] and a major one of [*7*:*8*]. It follows immediately that the second and third tuples in "the first copy" of relation MMQ both join to both the seventh and eighth tuples in "the second copy." And, of course, similar remarks apply to all of the other rows of that merged Field Values Table. In effect, therefore, we can do a sort/merge join without doing the sort *and without doing the merge, either!*[4]

Note: Lest I be accused of some hypocrisy, or at least inconsistency, in the way I've worded the previous paragraph, let me now try to state matters more precisely. Of course, there's no such thing as the "second" tuple, or the "third" tuple, or the "*i*th" tuple for any value of *i,* in any relation; the tuples of a relation aren't ordered. Thus, when I spoke of (for example) "the second tuple" of "the first copy" of relation MMQ, I was adopting a shorthand, and a pretty sloppy shorthand at that. What I really meant by such talk was as follows:

- Let *F1* be the reconstructed file obtained from the Field Values Table of Fig. 9.7 by processing column MAJOR_P# of the Record Reconstruction Table of Fig. 9.8 in top-to-bottom sequence. Then "the first copy" of relation MMQ is that file *F1,* and "the *i*th tuple" of that copy is that unique tuple of relation MMQ that corresponds to the *i*th record in *F1*.

- Likewise, let *F2* be the reconstructed file obtained from the Field Values Table of Fig. 9.7 by processing column MINOR_P# of the Record Reconstruction Table of Fig. 9.8 in top-to-bottom sequence. Then "the second copy" of relation MMQ is that file *F2,* and "the *i*th tuple" of that copy is that unique tuple of relation MMQ that corresponds to the *i*th record in *F2*.

The third and last point I want to mention is that merged columns can help improve update performance, especially for INSERT operations. Recall from Chapter 8 that condensed columns imply that such operations might be able to use field values that already exist, effectively sharing those values with other records. Well, the same is even more likely with merged columns, because the sharing can occur *across distinct fields*. By way of example, consider what happens if the user tries to insert an MMQ tuple with major part number P4, minor part number P6, and quantity 3.

9.3 A Foreign Key Example

For my second example, I want to return to the suppliers and shipments relations as discussed in earlier chapters. I've shown those two relations once again, side by side, in Fig. 9.9. Observe now that {S#} in the shipments relation SPJ is a **foreign key,** referencing the candidate key {S#} of the suppliers relation S (meaning that every value of {S#} in SPJ appears as a value of {S#} in S). Here's a slightly simplified definition of the concept:

- A *foreign key* is a subset of the attributes of some relation *R2* whose values are required to appear as values of some subset of the attributes of some relation *R1* (*R1* and *R2* not necessarily distinct). The attribute subset in question in relation *R1* must constitute a *candidate key* for that relation *R1*.

As I'm sure you know, joins over a foreign key and its corresponding candidate key are needed very frequently in relational systems.

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

S#	P#	J#	QTY
S1	P1	J1	200
S1	P3	J2	100
S2	P1	J1	200
S2	P1	J2	500
S2	P2	J2	500
S3	P1	J1	100
S3	P2	J2	500
S3	P3	J1	200
S3	P3	J2	200

Fig. 9.9: The suppliers and shipments relations S and SPJ

Let's assume the relations of Fig. 9.9 are mapped to files with field and record orderings that directly reflect those suggested by that figure. Then Figs. 9.10 and 9.11 show the corresponding Field Values Table and Record Reconstruction Table for suppliers, and Figs. 9.12 and 9.13 do the same for shipments. (**Exercise 16:** As usual, I recommend you check all of these tables carefully.) Note that I haven't condensed the supplier numbers column in Fig. 9.10, because each supplier is guaranteed to have a unique supplier number. By contrast, I *have* condensed the supplier names column, albeit to little effect.

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

Fig. 9.10: Field Values Table for suppliers

	1	2	3	4
	S#	SNAME	STATUS	CITY
1	5	1▪5	1▪4	1▪5
2	4	2▪4	2▪2	2▪1
3	2	3▪3	2▪3	2▪4
4	3	4▪1	3▪5	3▪2
5	1	5▪2	3▪1	3▪3

Fig. 9.11: Record Reconstruction Table for suppliers

	1	2	3	4
	S#	P#	J#	QTY
1	S1 [1:2]	P1 [1:4]	J1 [1:4]	100 [1:2]
2	S2 [3:5]	P2 [5:6]	J2 [5:9]	200 [3:6]
3	S3 [6:9]	P3 [7:9]		500 [7:9]

Fig. 9.12: Field Values Table for shipments

	1	2	3	4
	S#	P#	J#	QTY
1	1▪2	1▪1	1▪2	1▪2
2	1▪8	1▪2	1▪3	1▪6
3	2▪3	1▪3	1▪4	2▪1
4	2▪4	1▪7	1▪5	2▪3
5	2▪5	2▪8	2▪1	2▪8
6	3▪1	2▪9	2▪6	2▪9
7	3▪6	3▪4	2▪7	3▪4
8	3▪7	3▪5	2▪8	3▪5
9	3▪9	3▪6	2▪9	3▪7

Fig. 9.13: Record Reconstruction Table for shipments

Now let's combine the Field Values Tables of Figs. 9.10 and 9.12, merging the two supplier number columns together (the merging is clearly legitimate, because a foreign key and its corresponding candidate key must necessarily be of the same data type).[5] Refer to Fig. 9.14. Note the following points:

	1	*2*	*3*	*4*	*5*	*6*	*7*
	S#	SNAME	STATUS	CITY	P#	J#	QTY
1	S1[*1:2*]	Adams[*1:1*]	10[*1:1*]	Athens[*1:1*]	P1[*1:4*]	J1[*1:4*]	100[*1:2*]
2	S2[*3:5*]	Blake[*2:2*]	20[*2:3*]	London[*2:3*]	P2[*5:6*]	J2[*5:9*]	200[*3:6*]
3	S3[*6:9*]	Clark[*3:3*]	30[*4:5*]	Paris [*4:5*]	P3[*7:9*]		500[*7:9*]
4	S4[:]	Jones[*4:4*]					
5	S5[:]	Smith[*5:5*]					

Fig. 9.14: Merged Field Values Table for suppliers and shipments

- The merged table has seven columns, not eight. Column *1* is the merged column.[6] Columns *2-4* correspond to columns *2-4* of the suppliers Field Values Table; columns 5-7 correspond to columns *2-4* of the shipments Field Values Table. These facts will have to be taken into account when doing record or file reconstruction for shipments, but have no analogous implications for suppliers.

- The row ranges shown in column *1* indicate which rows of the uncondensed Field Values Table for *shipments* the corresponding supplier number appears in—we obviously don't need any analogous row ranges for *suppliers* (why not?). Note that the supplier numbers S4 and S5 don't appear in the shipments relation at all, and therefore don't appear in the shipments Field Values Table either (hence the empty row ranges for those suppliers in the *merged* table of Fig. 9.14).

- The corresponding Record Reconstruction Tables remain unchanged and are as shown in Figs. 9.11 and 9.13, respectively—with the trivial exception that, strictly speaking, we ought to replace the column numbers *2, 3, 4* for the Record Reconstruction Table for shipments by the column numbers *5, 6, 7,* respectively.

It should be clear that the advantages of merging columns in this example are analogous to those that applied in the bill-of-materials example in the previous section. In particular, joining suppliers and shipments on supplier numbers—which is a foreign-key-to-corresponding-candidate-key join, of course, and thus likely to be much needed in practice—now has the potential to be extremely fast (see Chapter 10).

Let me close this section by noting that foreign-key-to-corresponding-candidate-key joins are, by definition, many-to-one joins specifically, because a given tuple in the relation with the foreign key is guaranteed to join to exactly one tuple in the relation with the corresponding candidate key. (I discount the possibility that the foreign key might "be null" in some tuple, in which case it wouldn't join to any tuple at all. See the next chapter, Section 10.11.) By contrast, the join discussed in the previous section (a join of relation MMQ with itself) was a many-to-many join. And, while this latter example involved just a single relation, it should be clear that many-to-many joins between two distinct relations can also benefit from the merged-columns idea. It should also be clear that all of the concepts discussed in this chapter so far extend to three, four, ..., or any number of relations.

9.4 Another Kind of Merging

Toward the end of the previous chapter, I pointed out that column condensing was, among other things, a technique for saving storage space, and I took a brief look at certain other space-saving techniques that could be applied in the context of the TR model. Well, column merging too can be regarded among other things as a technique for saving storage space, and in the present section I'd like to take a quick look at a different kind of column merging that might also be used to save space.

The basic idea is that two distinct fields from the same file might map to a single combined column in the Field Values Table, even if they're of different data types. For example, consider the suppliers relation of Fig. 9.9 once again. Assume as before that the relation maps to a file with field and record orderings that directly reflect those suggested by that figure. Then, instead of mapping each field of that file to a Field Values Table column of its own as in Fig. 9.10, it would be possible to map—for example—the STATUS and CITY fields to a combined column, as shown in Fig. 9.15. Note the revised row ranges in particular.

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

Fig. 9.15: Field Values Table for suppliers with a combined STATUS / CITY column

Now, in this particular example, combining the STATUS and CITY columns in the Field Values Table as suggested in the figure probably doesn't save much space—at least, not in the Field Values Table, though it will certainly (and in fact more significantly) save space in the Record Reconstruction Table. But if there aren't very many distinct status values, or distinct city names, or (perhaps most important) distinct status-value / city-name combinations, then combining the columns has the potential to reduce space requirements significantly in the Field Values Table, too. However, there's a downside. Consider the problem of searching the Field Values Table for a particular status value or a particular city name. In the case of the status value, the search is no more difficult (and probably no more time-consuming) with the combined column than it was without it, because the combined column is still in status value order. But in the case of the city name, the search certainly is more difficult; in effect, separate searches will have to be done for each possible combination of a status value with the city name in question.

9.5 Concluding Remarks

In earlier chapters (Chapter 4 in particular), I noted that the TR model takes the concept of *data independence* much further than earlier systems did. Indeed, there's really no single thing, or combination of things, at all at the TR level that corresponds directly to a user-level tuple. From the discussions in this chapter, we can now see that there isn't necessarily any single thing or combination of things at the TR level that corresponds directly to a user-level relation, either—two or more user-level relations might all map to the same combination of constructs at the TR level. Analogous remarks apply to user-level attributes as well.

Endnotes

1. I also pointed out in the previous chapter that column condensing can be regarded, in part, as a kind of field-level compression. The same is true of column merging also.
2. In fact, the merged column contains the set theory union of the two original condensed columns, and we could thus reasonably call it (as reference [63] in fact does) a "union column." I prefer my term because it suggests, correctly, that there's some connection between such columns and the sort/merge approach to implementing join operations, as we'll see shortly.
3. Instead of empty ranges, we would then have adjacent entries in a column of the Field Values Table with the same begin point.
4. More accurately, the sort and the merge don't have to be done at run time; instead, they're done ahead of time when the Field Values and Record Reconstruction Tables are built (basically at load time).
5. I remark in passing that attributes STATUS and QTY are of the same data type, too (they're both of type INTEGER), and so we could have merged the STATUS and QTY columns as well if we had wanted.
6. I've labeled that column just "S#", but it's really "S# values from relation S or relation SPJ or both."