14 Stars and Zigzags

14.1 Introduction

This is the last chapter in this part of the book. In it, I want to describe a rather different approach to the problem of implementing the TR model on disk: more specifically, to the problem of minimizing disk seeks. Note immediately, therefore, that the approach in question can be regarded in part as an alternative to file banding as discussed in Chapter 13—but only in part, because in fact file banding can be used in combination with the approach to be described, as we'll see in Section 14.4. Note too that, as with the discussion of file banding in Chapter 13, we're primarily concerned here with how to deal with the "large file" that remains after file factoring has been used to get all of the "small files" into memory. But first things first.

As we know, the basic problem with TR on the disk is that if we're not careful, the zigzags can splay out all over the disk. Well, if the splay problem is caused by the zigzags, then let's get rid of the zigzags! Recall from Chapter 5 (Section 5.8) that the linkage information that lets us reconstruct records doesn't have to be implemented as zigzags specifically—other possibilities exist, with (of course) different performance characteristics. The approach to be described in this chapter exploits this idea; essentially, what it does is replace the zigzags by a different kind of structure called a **star**.

Let me illustrate this idea right away. Fig. 14.1 shows the Field Values Table and corresponding Record Reconstruction Table from Figs. 13.2 and 13.3 in Chapter 13—except that, for pedagogic reasons, I've shown the Field Values Table in uncondensed form. Fig. 14.2 then highlights one particular zigzag from Fig. 14.1 (actually the one for part P7), and Fig. 14.3 shows what happens if we replace that zigzag by a star.

	1	2	3	4
	P#	PNAME	WEIGHT	CC#
1	P1	Bolt	12.0	cc1
2	P2	Cam	12.0	cc1
3	P3	Cog	14.0	cc1
4	P4	Hinge	15.0	cc1
5	P5	Nut	17.0	cc2
6	P6	Nut	17.0	cc3
7	P7	Screw	19.0	cc3
8	P8	Screw	19.0	cc4
9	P9	Wheel	20.0	cc5

	1	2	3	4
	P#	PNAME	WEIGHT	CC#
1	5	5	1	1
2	1	2	8	4
3	8	7	2	6
4	7	9	9	7
5	2	1	5	2
6	3	8	6	3
7	6	3	3	9
8	9	6	4	5
9	4	4	7	8

Fig. 14.1: Uncondensed Field Values Table and corresponding Record Reconstruction Table

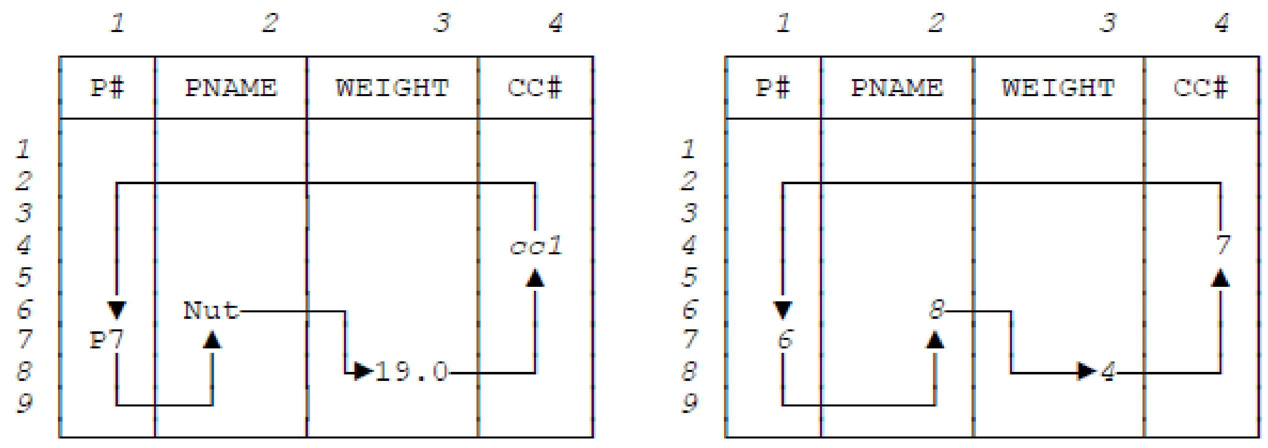


Fig. 14.2: Zigzag for part P7

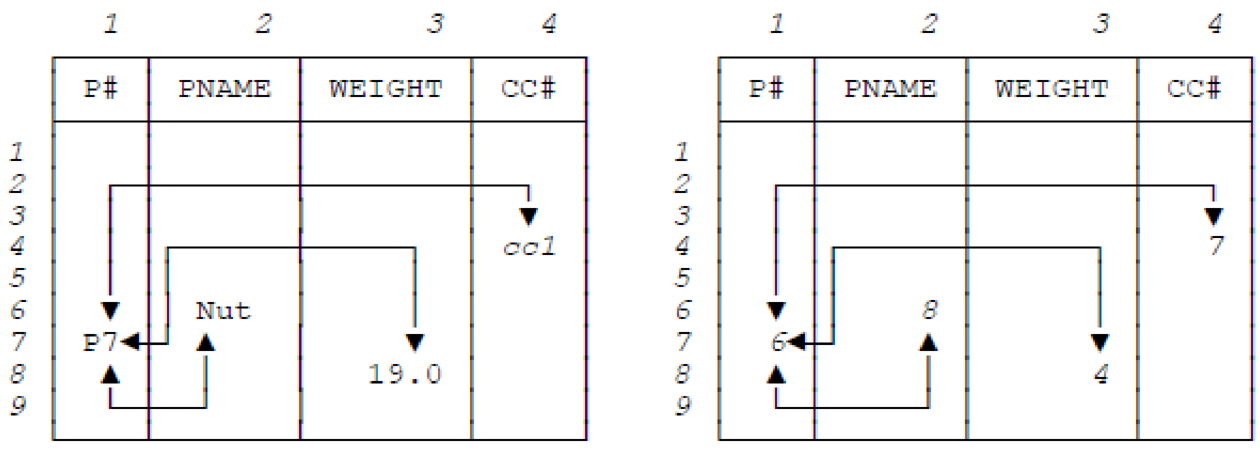


Fig. 14.3: Star for part P7 (with P# the core)

As you can see, where Fig. 14.2 has a *ring* of pointers (implemented within the Record Reconstruction Table and conceptually superimposed on the Field Values Table), Fig. 14.3 has a *star* of pointers instead. Cell [*7,1*], which corresponds to the P# value P7, serves as the center or **core** of that star. Three pointers emanate from that core and point to cells [*6,2*], [*8,3*], and [*4,4*], respectively; those cells correspond to the PNAME value Nut, the WEIGHT value 19.0, and the CC# value *cc1,* respectively. Those three pointers, which (as Fig. 14.3 indicates) are all two-way and can therefore be traversed in either direction, serve as the **spokes** or **rays** of the star.

Now, the star in the figure clearly does support reconstruction of the record for the part in question (part P7). To be specific:

a) If we start at the core, we can simply follow the three spoke pointers outward to obtain the other three field values.
b) If we start at any other point, we can follow the corresponding spoke pointer inward to the core and then proceed as under a) above—with the exception that, if we get to the core by following spoke pointer *sp* inward, then of course there's no need to follow that particular spoke *sp* outward again. *Note:* As a matter of fact, we *never* need to follow a spoke outward from the core within the Record Reconstruction Table as such; we only need to be able to go from the core outward to cells within the Field Values Table.

Now, you might have already realized that, for any given zigzag, there are several distinct but equivalent stars—it just depends on which field we choose as the core. I'll return to this point in Section 14.3. You might also have realized that the record reconstruction algorithm as just outlined displays asymmetric performance—access via the core field will be faster than access via any other field, because stars (unlike zigzags) are an inherently asymmetric structure—and I'll return to *this* point in Section 14.5.

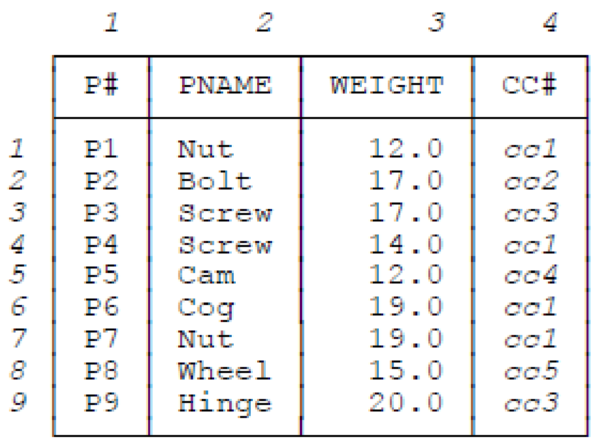
The structure of the chapter is as follows. Following this introductory section, Section 14.2 gives a simple example to illustrate the basic ideas behind star structures. Section 14.3 elaborates on and generalizes that example. Section 14.4 shows how the ideas from the first three sections work on the disk (those previous sections are principally concerned with a memory-based implementation only). Finally, Section 14.5 discusses the use of controlled redundancy in connection with star structures.

14.2 A Simple Example

As in the previous chapter, the basic problem we're trying to deal with is how to get the best possible performance out of the "large" Record Reconstruction Table in a disk-based system. So I'll base my discussions on the same running example as in that previous chapter; to be specific, I'll assume once again that we've factored the parts file into large and small files that look like this:

Large file	*Small file*
P#	CC#
PNAME	COLOR
WEIGHT	CITY
CC#	

However, we're interested here in the large file exclusively. Fig. 14.4 shows a sample value for that file (extracted from Fig. 13.1 in Chapter 13). And we've already seen a Field Values Table and a zigzag-based Record Reconstruction Table for that file in Fig. 14.1 above. *Note:* While the file shown in Fig. 14.4 is obviously not very large, let me remind you that we're really supposed to be dealing with files of millions or even billions of records, and the data in those files isn't supposed to display any "statistical clumpiness" at all.



	1	*2*	*3*	*4*
	P#	PNAME	WEIGHT	CC#
1	P1	Nut	12.0	*cc1*
2	P2	Bolt	17.0	*cc2*
3	P3	Screw	17.0	*cc3*
4	P4	Screw	14.0	*cc1*
5	P5	Cam	12.0	*cc4*
6	P6	Cog	19.0	*cc1*
7	P7	Nut	19.0	*cc1*
8	P8	Wheel	15.0	*cc5*
9	P9	Hinge	20.0	*cc3*

Fig. 14.4: Sample file

Now, despite the fact that we're really supposed to be talking about a disk implementation, it's convenient to pretend for the time being that everything's in memory, and I'll adopt that pretense until further notice. So how do we proceed? Well, since (as we've already seen) stars are asymmetric, the first thing we have to do is decide what the core's going to be; in other words, we first have to choose a **core field** (much as we had to choose a characteristic field in connection with with banding in the previous chapter).[1] Suppose we choose field P#. Then Fig. 14.5 shows a corresponding star-based Record Reconstruction Table for the file of Fig. 14.4. *Note:* From this point forward, for convenience, I'll abbreviate the term "star-based Record Reconstruction Table" to just **star table,** and similarly for **zigzag table**.

	1	2	3	4
	P# n-w-c	PNAME	WEIGHT	CC#
1	5-1-1	2	1	1
2	1-5-5	5	5	4
3	8-6-6	6	4	6
4	7-3-2	9	8	7
5	2-2-8	1	2	2
6	3-7-3	7	3	3
7	6-8-4	4	6	9
8	9-4-9	3	7	5
9	4-9-7	8	9	8

Fig. 14.5: Star-based Record Reconstruction Table for the file of Fig. 14.4 (with P# the core)

In order to explain the star table of Fig. 14.5, let's go back for a moment to the *zigzag* table of Fig. 14.1. Consider the zigzag for (say) part P7, which—as Fig. 14.2 shows graphically—looks like this:

```
[7,1], [6,2], [8,3], [4,4]
```

Business
Strategic Marketing Management
BI
Financial Economics
International Business
Leadership & Organisational Psychology
Shipping Management
BI NORWEGIAN BUSINESS SCHOOL
EFMD EQUIS ACCREDITED

In a star analog of this zigzag, therefore, the core cell [*7,1*] will contain the pointer triple *6-8-4* (these are the outward portions of the spokes) and cells [*6,2*], [*8,3*], and [*4,4*] will each contain a *7* (these are the inward portions of the spokes). And similarly, of course, for all of the other stars in the table. *Note:* I've expanded the heading of column P# in Fig. 14.5 to show which pointers are which. To be specific, in the triple *n-w-c, n* is the PNAME (name) pointer, *w* is the WEIGHT pointer, and *c* is the CC# pointer.

Observe, incidentally, that it's a consequence of the way the star table in the example is defined that:

a) The first "subcolumn" within column P# of the star table—the one for PNAME, labeled *n* in the figure—is identical to column P# of the zigzag table (why, exactly?);
b) Column CC# of the star table (the last column) is identical to column CC# of the zigzag table (again, why exactly?).

Now let's consider how the star table of Fig. 14.5 might be used to implement queries. Consider the following simple example:

```
SELECT DISTINCT P.P#, P.WEIGHT
FROM P
ORDER BY P# ;
```

This query refers to relation P, but of course it can be implemented by accessing the large file only—we[2] don't need to touch the small file at all. (This is a generic observation, and I won't bother to repeat it in subsequent examples.) So what we have to do is this:

- Traverse column P# of the star table top to bottom to obtain the result in the desired ordering.
- The first cell encountered, cell [*1,1*], corresponds to cell [*1,1*] in the Field Values Table, which (as Fig. 14.1 shows) contains the part number P1.
- That same first cell in column P# of the star table contains the pointer triple *5-1-1*. The first pointer in this triple corresponds to a part name, the second to a weight, and the third to a CC# value. However, the query isn't interested in part names or CC# values, so we can go just to the WEIGHT cell in the Field Values Table—which is to say cell [*1,3*]—to obtain the desired weight value. The first result record has now been constructed.
- All other result records are constructed analogously.

Note: If the query had specified ORDER BY WEIGHT instead of ORDER BY P#, we would have accessed the star table by column WEIGHT instead of column P#. For each cell encountered, we would have followed the inward pointer to the corresponding core cell and then used that core cell to construct the corresponding result record as above.

One point that emerges right away from the foregoing is that stars might be better than zigzags for implementing projections. To be specific, if the file has *M* fields, then (in general) zigzags require *M* accesses to the Record Reconstruction Table for each result record no matter how many fields are requested, while stars require at most two (and often only one). This fact makes stars attractive, because it's quite rare in practice for a query to request *all* of the fields of the file (or all of the attributes of the relation, rather).

Here's another sample query:

```
SELECT DISTINCT P.P#
FROM P
WHERE P.WEIGHT = 19.0 ;
```

Here we do a binary search on column WEIGHT of the Field Values Table and determine that the records we want pass through cells [*7*,*3*] and [*8*,*3*] of the star table. Then we use the stars corresponding to those cells to construct the desired records.

Before closing this section, I should draw your attention to one more point: namely, that a star table will always be bigger than its zigzag analog. Again suppose the file has *M* fields. Then the zigzag table will have *M* pointers per record, but the star table will have 2(*M*-1)—so if *M* is large, the star table will be almost twice the size of the zigzag table. *Note:* I'm assuming here that *M* is greater than one. What happens if that assumption is invalid?

14.3 Elaborating on the Example

Now let's see what happens if we choose a field other than one corresponding to some key as the core field. Let's choose field WEIGHT. Fig. 14.6 shows what happens to the star for part P7 under this assumption; Fig. 14.7 shows the corresponding star table in its entirety. Observe that:

a) The first "subcolumn" within column WEIGHT of the star table of Fig. 14.7—the subcolumn for CC#, labeled *c* in the figure—is identical to column WEIGHT of the zigzag table of Fig. 14.1;
b) Column PNAME of the star table of Fig. 14.7 is identical to column PNAME of the zigzag table of Fig. 14.1.

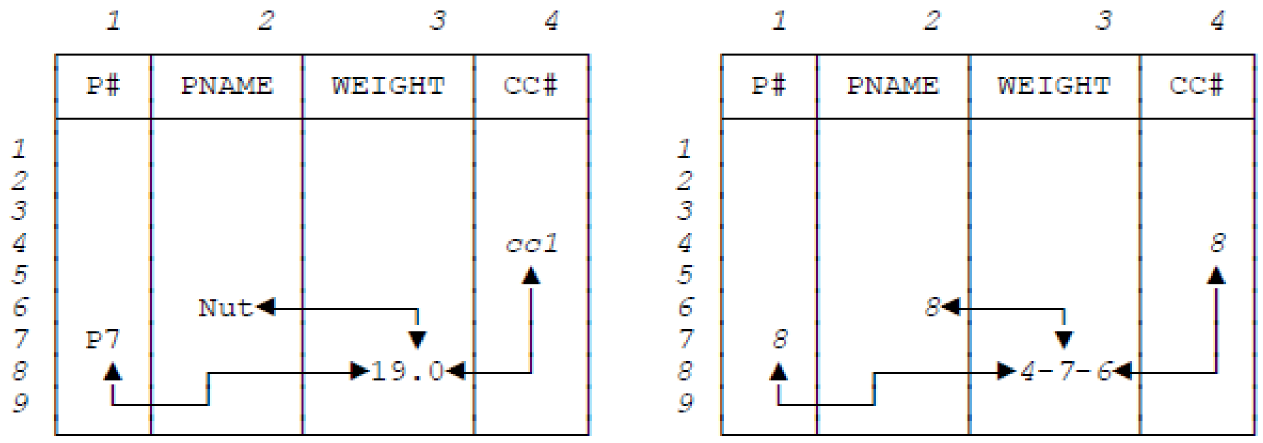
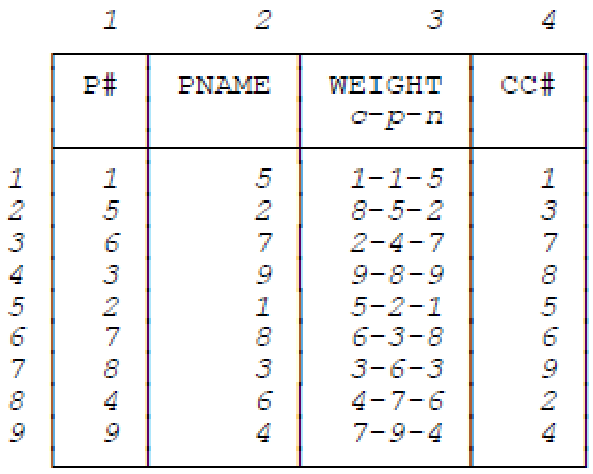


Fig. 14.6: Star for part P7 (with WEIGHT the core)



	1	2	3	4
	P#	PNAME	WEIGHT c-p-n	CC#
1	1	5	1-1-5	1
2	5	2	8-5-2	3
3	6	7	2-4-7	7
4	3	9	9-8-9	8
5	2	1	5-2-1	5
6	7	8	6-3-8	6
7	8	3	3-6-3	9
8	4	6	4-7-6	2
9	9	4	7-9-4	4

Fig. 14.7: Star table for the file of Fig. 14.4 (with WEIGHT the core)

Observe too that (to spell out the obvious) the star tables of Figs. 14.5 and 14.7 are different. Thus, while we might reasonably talk about "the" zigzag table for a given file, we can't sensibly talk about "the" star table for that same file; instead, we have to talk about the star table that corresponds to the given file *together with some given core field*.

Now I want to make another point. Suppose we use the star table of Fig. 14.7 to reconstruct the entire file; suppose for definiteness that we perform this process using column PNAME (that is, we traverse column PNAME of the star table top to bottom). Here's the reconstruction process spelled out in detail:

- From cell [*1*,*2*] of the star table, follow the spoke inward to the corresponding core cell [*5*,*3*].
- Go to the Field Values Table and extract the WEIGHT value in cell [*5*,*3*] (from Fig. 14.1, we see the value in question is 17.0).
- Cell [*5*,*3*] of the star table contains the pointers *5, 2,* and *1*. Go to the Field Values Table and extract the CC# value in cell [*5*,*4*], the P# value in cell [*2*,*1*], and the PNAME value in cell [*1*,*2*]. Those values are *cc2,* P2, and Bolt, respectively, and we have now constructed the first result record. *Note:* Alternatively, of course, we could have obtained the PNAME value (Bolt) in the first step by going straight to cell [*1*,*2*] of the Field Values Table (since we started out with cell [*1*,*2*] of the star table in the first place).

Performing this sequence of steps eight more times but starting successive iterations with cells [*2*,*2*], [*3*,*2*], ..., [*9*,*2*] of the star table (for the second, third, ..., ninth record in the overall file reconstruction process), we obtain the result shown in Fig. 14.8. That result, as you can see by inspection (or by comparison with Fig. 13.1 in Chapter 13), consists of the original nine part records ordered by weight within name (more precisely, ordered by P# within CC# within WEIGHT within PNAME). In other words, the star table of Fig. 14.7 is a "preferred" one in the sense of Chapter 7. (So too is the star table of Fig. 14.5, as a matter of fact.)

	1	*2*	*3*	*4*
	P#	PNAME	WEIGHT	CC#
1	P2	Bolt	17.0	*cc2*
2	P5	Cam	12.0	*cc4*
3	P6	Cog	19.0	*cc1*
4	P9	Hinge	20.0	*cc3*
5	P1	Nut	12.0	*cc1*
6	P7	Nut	19.0	*cc1*
7	P4	Screw	14.0	*cc1*
8	P3	Screw	17.0	*cc3*
9	P8	Wheel	15.0	*cc5*

Fig. 14.8: Part records ordered by WEIGHT within PNAME

One last point to close this section: Consider, by way of example, cell [*8*,*3*] of the star table in Fig. 14.7. That cell corresponds directly to cell [*8*,*3*] of the Field Values Table in Fig. 14.1, which contains the weight 19.0. That same cell [*8*,*3*] in the star table contains the pointers *4, 7,* and *6,* which take us to cells [*4*,*4*], [*7*,*1*], and [*6*,*2*] of the Field Values Table, and those cells in turn contain the CC# value *cc1,* the part number P7, and the name Nut, respectively. In a sense, therefore, that star table cell [*8*,*3*] can be thought of, all by itself, as a digitized version of the entire record for part P7[3]—its position within the star table effectively specifies one component of that record (the WEIGHT component), and its contents effectively specify the other three components (the CC#, P#, and PNAME components). *Note:* It's relevant to mention once again that—as we first saw in Chapter 5, Section 5.6—pointers to Field Values Table cells can usefully be thought of as *surrogates* for the field values contained within those cells.

14.4 What Happens on Disk

Now let's drop the pretense that everything's in memory and see how the ideas we've been discussing work out in a disk environment—by which I mean, primarily, an environment in which the star table is too big to be memory-resident. *Note*: It's easier to talk in terms of just one star table at a time; in what follows, therefore, I'll pretend there is indeed just one such table (barring explicit statements to the contrary), and I'll refer to it as "the" star table.

The first point is that, in the case of the core column in particular, we're probably going to want to extend the column-wise storage idea to store each subcolumn of that column as an independent column in its own right. The reason is that (as we saw in Section 14.2) we usually don't need to access all of those subcolumns in implementing any given query, and we certainly don't want to read anything into memory that we don't need, if we can help it. Fig. 14.9 shows the star table of Fig. 14.5—the one based on P# as the core field—with the core column divided up into separate columns in this way. *Note:* The term subcolumn, which I've now used several times, is (I hope) self-explanatory. From this point forward, I'll use the term core column to mean the combination of all pertinent subcolumns.

	1n	1w	1c	2	3	4
	core (P#)			PNAME	WEIGHT	CC#
	PNAME	WEIGHT	CC#			
1	5	1	1	2	1	1
2	1	5	5	5	5	4
3	8	6	6	6	4	6
4	7	3	2	9	8	7
5	2	2	8	1	2	2
6	3	7	3	7	3	3
7	6	8	4	4	6	9
8	9	4	9	3	7	5
9	4	9	7	8	9	8

Fig. 14.9: Star table of Fig. 14.5 with core column subdivided

The table of Fig. 14.9 will be stored on disk as six separate columns, and the implementation will thus be able to go directly to the start of any of those stored columns at any time. What's more, those six columns will also be stored in consecutive pages on the disk (the first column in one set of pages, the second in the immediately following set, and so on), with a view to minimizing seek activity and allowing successive columns to be streamed into memory at run time. (Of course, it's highly desirable for the pertinent core subcolumns to be in memory at run time—where by "pertinent" I mean the ones that are needed for any given query—even if the star table overall is too big to fit into memory in its entirety.)

So what's good about this arrangement from a performance point of view? Well, it certainly means that pointers in any given column of the star table will be physically contiguous on the disk, with the implication that traversing such a column will be fast. And since these remarks apply to subcolumns of the core column in particular, it follows that doing record or file reconstruction via the core column will *not* lead to the splay problem. What's more, so long as the core column is in memory, doing reconstruction via any other column will be efficient too; but if the core column is too big to fit into memory, then reconstruction via any other column still has the potential to be extremely inefficient. However, at least we don't have to deal with "splayed zigzags" as such.

So what can we do if the core column is too big to fit into memory? One approach would be to use **banding,** more or less as described in the previous chapter. Banding, as you'll recall, is a divide-and-conquer technique that works by dividing the file up into horizontal subfiles or bands such that (a) each band fits into memory and (b) no pointing occurs between bands. Refer to Chapter 13 for further discussion of this possibility. The section immediately following describes a different approach to the same problem.

14.5 Controlled Redundancy

We've seen that, in order to define a star table, the first thing we have to do is choose a core field.[4] Now, when we were discussing banding in Chapter 13, the choice of characteristic field had major performance implications, because that choice effectively dictated the stored sort order for the file. And the situation is similar (though not identical) with a star table and its core field; again our choice can have major performance implications. To be more specific, if we choose *C* as the core field, then queries that involve access to the file in sequence by values of *C* will be very efficient, but (as explained near the end of the previous section) queries that involve access in any other sequence might not be.

The obvious solution to this problem is to introduce some form of **controlled redundancy** once again. I discussed controlled redundancy at some length in Chapter 13 (Section 13.5); most of the points I made there apply here too, and I won't bother to repeat them all now. Let me just remind you yet again that the TR representation of a given data set typically requires only some 20 percent of the space required for a direct-image representation; as a result, we can afford to store the data up to five different ways without taking up any more space than a conventional system would need (and that's just for storing the raw data alone, in that conventional system).

So let's consider the question of redundancy in the context of star tables specifically. Clearly, the simplest thing to do is to store several different star tables for the same file, each one based on a different core field. For example, we could store both the star table of Fig. 14.5 (based on P#) and the star table of Fig. 14.7 (based on WEIGHT), and thereby achieve symmetric performance for access to parts based on P# and access to parts based on WEIGHT.

An alternative approach would be to store just one star table but to expand that table to include, in addition to what I'll now call the *primary* core column, a set of *secondary* core columns as well. Each such secondary core column will contain essentially the same information as the primary one, but sorted into a sequence that matches the sort sequence of some field that's not the (primary) core field.

To see how this idea works out in practice, let's work through an example. Consider column WEIGHT in the star table of Fig. 14.9. As you can see, that column contains the following pointers (row numbers) in top-to-bottom sequence:

```
1, 5, 4, 8, 2, 3, 6, 7, 9
```

This sequence is in fact the *permutation* that corresponds to the specification ORDER BY WEIGHT, CC#, P#, PNAME; it means, for example, that record *8* of the file—see Fig. 14.4—appears in position *4* in that ordering. It follows that if we were to process the core column of that same star table by taking the first cell first, the fifth cell second, the fourth cell third, and so on, we would reconstruct a version of the file that was in exactly that ordering.

The trouble is, of course, that processing the core column in the way just indicated could lead to a lot of seek activity on the disk. So why not store another copy of that core column that's rearranged into exactly the sequence we want? The result might look like this:

```
5-1-1
2-2-8
7-3-2
9-4-9
1-5-5
8-6-6
3-7-3
6-8-4
4-9-7
```

If such a column is added—redundantly, of course—to the star table of Fig. 14.5, we'll be able to reconstruct the desired file from that table in the desired order without all of that seek activity on the disk, precisely because that new column will be stored as a separate column in its own right on the disk. (Well, actually it'll be stored as three separate columns, one for each subcolumn, but that detail need not concern us here.)

But wait a moment ... Recall that within a given triple of pointers *n-w-c* in the primary core column, *n* is the name pointer, *w* is the weight pointer, and *c* is the CC# pointer. Clearly there's no need to include the *w* pointers in the secondary core column, because the value of the *w* pointer in the *i*th cell will always simply be *i* (as you might have already noticed). Thus, the final version of the star table with both a primary core column and a secondary core column will look as shown in Fig. 14.10 (note the column labels *1n, 1w,* etc.).

	1n	1w	1c	2	3	3n	3c	4
	primary			PNAME	WEIGHT	secondary		CC#
	PNAME	WEIGHT	CC#			PNAME	CC#	
1	5	1	1	2	1	5	1	1
2	1	5	5	5	5	2	8	4
3	8	6	6	6	4	7	2	6
4	7	3	2	9	8	9	9	7
5	2	2	8	1	2	1	5	2
6	3	7	3	7	3	8	6	3
7	6	8	4	4	6	3	3	9
8	9	4	9	3	7	6	4	5
9	4	9	7	8	9	4	7	8

Fig. 14.10: Star table with primary (P#) and secondary (WEIGHT) cores

Now suppose we want to reconstruct the part record passing through some particular cell in the WEIGHT column of the star table of Fig. 14.10—let's say (arbitrarily) the fourth cell, which is still cell [*4*,*3*] according to the column labeling shown in the figure. The sequence of events is as follows.

- Go to cell [*4*,*3*] of the Field Values Table of Fig. 14.1 and extract the value stored there (weight 15.0).

- Cell [*4,3*] of the star table contains the row number *8*, so the part number of the record we want is in cell [*8,1*] of the Field Values Table. Go and extract it (part number P8).

- Within the secondary core column, go to the PNAME cell corresponding to WEIGHT cell [*4,3*]. That PNAME cell is cell [*4,3n*], and it contains the row number *9*. Go to cell [*9,2*] of the Field Values Table and extract the name (Wheel).

- Within the secondary core column, go to the CC# cell corresponding to WEIGHT cell [*4,3*]. That CC# cell is cell [*4,3c*], and it also contains the row number *9*. Go to cell [*9,4*] of the Field Values Table and extract the CC# value (*cc5*). Record reconstruction is now complete.

Here's the storage arithmetic again. Suppose once again that the file has *M* fields. Then:

a) A zigzag table will have *M* pointers per record.
b) A star table with a single core column will have 2(*M*-1) pointers per record.
c) A star table with a primary core column and *N* secondary core columns will have 2(*M*-1) + *N*(*M*-2) pointers per record.

Of course, Case b. is just that special case of Case c. in which *N* is zero.

Endnotes

1. In fact the core field is often referred to as a characteristic field. I'll stay with the term *core field* in this chapter.
2. As in Chapter 10, "we" here really means the DBMS.
3. *Chambers Twentieth Century Dictionary* defines *digitize* to mean "to put (data) into digital form for use in a digital computer."
4. It might be possible to automate that choice, but probably not if we introduce redundancy (which I'm about to do); in that case, human decisions are probably going to be needed.