Decision Management Community Challenge Nov 2017- Soldier Pay A solution using SQL

(Bob Moore, JETset Business Consulting, 28 Nov 2017)

1 Problem Statement (cut and pasted from the web site)

During different service periods a soldier may have the following characteristics:

Rank {private, corporal, sergeant, lieutenant, captain}
Profession {fighter, driver, cook, officer}
Service Type {active, reserve, retired}
Unit {HQ, paratroopers, marines, infantry}
Combat {yes, no}

Pay rate is determined by aggregating the amounts according to these rules:

Base rate is $1/hr.
Private $1/hr., corporal $2/hr., sergeant $3/hr., lieutenant $4/hr., captain $5/hr.
Fighter $2/hr., driver $1/hr., cook $1/hr., officer $3/hr.
Active $2/hr., Reserve $1/hr.
HQ $1/hr., others $2/hr.
Combat $5/hr., non-combat $0/hr.

Example:

Rank: Private 1/1/2015 -12/31/2015,
Profession: Fighter 1/1/2015-6/30/2015, Cook 7/1/2015-12/1/2015
Service Type: Active 1/1/2015-6/30/2015, Reserve 7/1/2015-12/1/2015
Unit: HQ 1/1/2015-12/31/2015
Combat: 4/1/2015-6/30/2015

On 6/1/2015 he was a private, a fighter, on active duty, at HQ, in combat, So, his pay rate was

1 (base) + 1 (private) +2 (fighter) +2 (active) +1 (HQ) +5 (combat) = $12/hr

The challenge:

assemble a single timeline for the soldier over a given service period that shows his/her hourly pay rate in any given time. Flag any conflicting dates (e.g. can't be a sergeant and a lieutenant at the same time).

Additional challenge:

What are all the different aggregated pay rates that apply and during which periods?

2 Initial Approach

Since I'm twiddling my thumbs a bit currently, I decided to have a bash at this challenge. I started off without much thought to what kind of technology to use but ended up thinking that a lot of it could be done just using relational tables and SQL, and so I thought I'd see how far I could get just using that. The answer is pretty much all the way. A fairly complete solution comprises only two lines of SQL (though one of these is a rather long line, and to fully round things out I ended up with about ten SQL queries overall)

There are those who will argue SQL is neither DMN nor a BRMS. But it is very powerful tool for extracting information, and for problems where you have a few simple rules and a mass of data, it can do a surprising amount of the heavy lifting for you. Indeed, for many decision tables where both inputs and outputs only comprise atomic data items, SQL is probably more powerful and flexible than something like FEEL.

My view when building decision systems has always been ‘the database is your friend’. Playing to the strengths of both your BRMS and your DBMS can make life a lot easier, a point which I hope this offering makes abundantly clear.

That said this solution is not without some limitation, so I’m planning on a second solution probably based on Drools to provide a more rounded effort

3 Problem Interpretation

As with any problem statement, there are one or two areas open to different interpretations. In this one, the ones I picked up on were:

- For a given characteristic, can a soldier exhibit different values for the same characteristic at the same time (eg can they simultaneously be a fighter and cook)? This is explicitly excluded for rank, but it makes little sense for most of the others too (you can’t be in the infantry and in the marines at the same time). For simplicity I’ve assumed that all the characteristics are single valued.
- A soldier can be ‘retired’, but what does that signify? Are we including army pensions here? I’ve assumed if you are retired you don’t get any money, so you can be ignored.
- What if any limitations are there on the start and end points of the periods given? In theory if you got promoted at mid-day, your hourly pay would change at that point. Again for simplicity, I’ve assumed the limit on granularity is the day (and that everything takes place in the same time zone!)

In addition, I’ve assumed pay rates do not change with time and that the time line covers the entire service period.

With a bit more work most of these assumptions can be relaxed.

4 Let’s Get Modelling

So, what is the decision we are modelling? In essence, we want to create for a ‘timeline’, I’m interpreting this as generating a report which says something like this:

```
From the 20017-01-01 hourly rate is  6 SVU[1]
From the 20017-02-01 hourly rate is 11 SVU
From the 20017-04-01 hourly rate is 12 SVU
From the 20017-06-01 hourly rate is 13 SVU
From the 20017-07-01 hourly rate is 14 SVU
From the 20017-09-01 hourly rate is 14 SVU
From the 20017-11-01 hourly rate is  9 SVU
```

[1] To internationalize the problem, I’ve opted a date format understood on both sides of the Atlantic and Jack Vance’s ‘Standard Value Unit’ as currency which nobody uses (yet).

There are obviously other ways of doing representing the outcome, but I think they are probably going to be equivalent. I've leapt ahead a little in presenting things in this form, which is roughly where we end up, so there are few points to note on this proposed format:

- We give start dates for an hourly rate, but end dates are implicitly set by a new start date
- The hourly rate is given, but not an explanation of its origin
- We may get a sequence of entries on the report where the rate does not change (e.g. after the 1st of July, there is an entry on the list for September, but the rate is still 14 SVU/hour)

Broadly these are down to limitations of SQL, however as we will see it is easy to add an 'explanation' of the hourly rates and the last point is simply down to the fact that while the overall rate may not change, the explanation for it can.

Before getting down to trying to make decisions, we need to figure out what things we are trying to make decisions about. Let's build an object/data model. On the basis that we have a fully additive model, it looks like we have three basic structures:

- Soldiers – who have a number of characteristics
- Characteristics – which have a type (Rank, Profession, …) a value (Private, Fighter, …) and a period over which they apply (start and end dates)
- Pay Rates – which are associated with a specific value of a characteristic type

At this point it's worth thinking about how we might build a solution. Pragmatically, in any real-world solution all the information we've mentioned is going to be in a database, probably a relational database. So, what is that relational database going to look like?

While the army is probably interested in a host of things about the solider like their age their gender, their next of kin, none of this is relevant to the problem at hand. After a bit of thought we find we don't actually need a 'Soldiers' table. All we need something which tells us for each soldier, what is the setting (value) of each characteristic between certain times. Since 'characteristics' are time invariant (your rank may change but you always have a rank), we can't really call the table 'Characteristic', so I've used the term 'Engagement' to describe the idea that a particular characteristic takes a particular value over a particular time period for a particular soldier.

The 'Engagement' Table will look something like this:

soldier	characteristic	setting	start date	end date
----	----	----	----	----
alice	base	base	2017-01-01	2018-01-01
alice	rank	private	2017-01-01	2018-01-01
alice	profession	fighter	2017-01-01	2018-01-01
alice	service type	active	2017-01-01	2018-01-01
alice	unit	paratroopers	2017-01-01	2018-01-01
alice	combat	yes	2017-01-01	2018-01-01
bill	base	base	2017-01-01	2018-01-01
bill	rank	private	2017-01-01	2017-07-01

bill	rank	corporal	2017-07-01	2018-01-01
bill	profession	driver	2017-01-01	2017-06-01
bill	profession	fighter	2017-06-01	2018-01-01
bill	service type	reserve	2017-01-01	2017-04-01
bill	service type	active	2017-04-01	2018-01-01
bill	unit	marines	2017-01-01	2017-09-01
bill	unit	paratroopers	2017-09-01	2018-01-01
bill	combat	no	2017-01-01	2017-02-01
bill	combat	yes	2017-02-01	2017-11-01
bill	combat	no	2017-11-01	2018-01-01
----	----	----	----	----

Note in this approach we treat all characteristics in the same manner. The characteristic ‘rank’ is treated in exactly the same way as ‘profession’. This has a couple of consequences. Firstly, while the concept of a ‘characteristic’ is pervasive in the solution, the logic never mentions a characteristic by name. Following immediately from this we can see that adding (or removing or renaming) characteristics has no effect on the decisioning part of the solution – you just add/remove/update database records. Which is nice for extending the model.

Another point to notice is I’ve made a small alteration to the way engagements are defined. The engagement ends *before* the ‘end date’, not *on* the end date. So, an engagement applies for dates such that start date <= date < end date. This change is not needed for generating the time line, but it makes validation (see below) much simpler (it also helps were we relax the assumption that all characteristics are single valued).

It is also worth noting that the example data above is ‘correct’ – in that at any date in the service period, all the characteristics have one and only one value. Validating this turns out to be the real challenge in providing a comprehensive solution.

Finally, as anyone building database schema should know, we want to specify what the database key is. Clearly soldier and characteristic should form part of it, and since we might swap back and forth between values of a characteristic during service, at least one date must be involved. For validation purposes it makes most sense to say only one engagement for a given characteristic can start on a given day, so the key is the combination ‘soldier’ + ‘characteristic’ + ‘start date’ (you could use ‘end date’ instead or use both dates if you like – but the validation process if you only choose one).

In addition to the ‘Engagement’ table, we also need a ‘Payment’ table – which is rather simpler (and has the key ‘characteristic’ + ‘setting’):

characteristic	setting	hourly rate
base	base	1
rank	private	1
rank	corporal	2
rank	sergeant	3
rank	lieutenant	4
rank	captain	5
profession	fighter	2

profession	driver	1
profession	cook	1
profession	officer	3
service type	active	2
service type	reserve	1
unit	hq	1
unit	paratroopers	2
unit	marines	2
unit	infantry	2
combat	yes	5
combat	no	0

5 Building the timeline

Once we've got these tables in place and populated we can do some simple stuff with SQL. For example, we can ask, what are the contributions to the hourly pay rate of soldier 'Bill' on 4th July with a query like:

```
select soldier, e.characteristic, e.setting, hourly_rate
from engagement e, payment r
where soldier= 'bill' and start_date <= '2017-07-04' and end_date > '2017-07-04'
          and e.setting = r.setting and e.characteristic = r.characteristic
order by e.characteristic
```

If you're not familiar with SQL this may be a bit intimidating, but all it is saying is 'find the engagements for Bill, which are happening on the date of interest, and then look up the associated hourly rate for the given characteristic and characteristic setting. Note we use <= for start dates and > for end dates as described above. The order by is not needed but simply ensures the same output on different DBMS. The outcome is:

soldier	characteristic	setting	hourly_rate
bill	base	base	1
bill	combat	yes	5
bill	profession	fighter	2
bill	rank	private	1
bill	service type	active	2
bill	unit	marines	2

So, we can just add numbers up in the last column to work out that Bill's pay is 14 SVU per hour. We can do even better! SQL can do the sum for us:

```
select soldier, sum(hourly_rate) as "hourly rate"
from engagement e, payment r
where soldier= 'bill' and start_date <= '2017-07-04' and end_date > '2017-07-04'
          and e.setting = r.setting and e.characteristic = r.characteristic
group by soldier
```

Giving:

soldier	hourly rate
bill	14

So, give me any date, I can use this bit of SQL to work out a soldier's hourly pay. Are we done already?

Unfortunately, we're not. I may know Bill's hour rate of pay is 12 SVU on the 4th of April and 14 SVU on 4th of July, but when did it change? And has it been some other value in between (in fact, it was 13 SVU in June)?

It is clear we need to break the service period in smaller intervals, where in each of the intervals, none of the characteristics change (so the pay rate doesn't change either).

Suppose we start off with 'rank'. We split the overall service period into two intervals, one from 2017-01-01 to 2017-06-30 and one from 2017-07-01 to 2017-12-31. Next let us look at 'profession' There are two intervals again, but there is an overlap between the ones for 'rank'. Bill becomes a 'fighter' in June, before being promoted to corporal in July. So, we need to create another interval. This process begins to look a bit complicated. And we haven't looked at unit, combat status, or active status yet. Be assured it can get very complicated!

Take a step back though, and think about the problem, and a much simpler way of determining the intervals comes to light. Bill's hourly rate of pay changes through the year, but when and why does it change? It changes precisely when a new engagement starts (note that a new engagement starting implicitly means an old one ending). So, the intervals we are interested in are defined by the events of new engagements commencing, so to find the intervals we only need to find the events – the start dates of new engagements. A very simple bit of SQL:

```
select distinct start_date from engagement where soldier = 'bill' order by start_date
```

gives us the sequence of dates:

start_date
2017-01-01
2017-02-01
2017-04-01
2017-06-01
2017-07-01
2017-09-01
2017-11-01

(the 'distinct' qualification gets rid of duplicates as occur on 2017-01-01 for example)

Lo and behold we have all the different intervals we are looking for!

And there's more! By construction, we know that within an interval none of the characteristics change their setting, so the pay rates don't change, and in particular they will equal the pay rates on the start date of the interval. So, if we use the rather complicated SQL up above which calculates the hourly rate at a given date, using the start date of the interval as our given date we will get the hourly rate which holds throughout the interval.

So now we have one SQL statement which gives us the dates which define the timeline of a soldiers pay, and another SQL statement which gives us the hourly rate on a given date. If we can somehow put the two together we've got a solution.

And it proves quite easy. To do this let us first generalise the logic we used to find the 'interesting' dates which form our time line. Instead of looking for just the events which change Bill's hourly rate, we look at the events for all soldiers.

```
select distinct soldier, start_date from engagement
```

Then we wrap this as a view – a virtual table – which we call 'period'

```
create view period as select distinct soldier, start_date from engagement
```

Then we add a join to this view in our SQL that gets the hourly rates. This results in this slightly intimidating bit of SQL:

```
select p.soldier, p.start_date, sum(hourly_rate) as "hourly rate"
from engagement e, payment r, period p
where e.soldier = 'bill' and e.soldier = p.soldier
        and e.start_date <= p.start_date and e.end_date > p.start_date
        and e.setting = r.setting and e.characteristic = r.characteristic
group by p.soldier, p.start_date
order by p.soldier, p.start_date
```

It looks worse than it is, and I hope that having built up to this, those less familiar with SQL can see where it comes from. Basically, we are using the 'period' view to select which engagements in the 'engagement' table are of interest, where the clause:

```
e.soldier = 'bill' and e.soldier = p.soldier
```

makes sure we are only looking at information about the soldier of interest (Bill) and the clause:

```
e.start_date <= p.start_date and e.end_date > p.start_date
```

picks out all engagements are active when a new engagement starts (the event signifying a change in the way the monthly pay rate is made up). The clause:

```
e.setting = r.setting and e.characteristic = r.characteristic
```

simply matches up with the 'payment' table to enable us to extract the right rate. Finally the clause: `group by p.soldier, p.start_date` links to the `sum(hourly_rate)` part of the select, to ensure we have separate sums for each event date, rather than a sum over everything.

If we run this query we get:

soldier	start_date	hourly rate
bill	2017-01-01	6
bill	2017-02-01	11
bill	2017-04-01	12
bill	2017-06-01	13
bill	2017-07-01	14
bill	2017-09-01	14
bill	2017-11-01	9

One can reasonably argue that the query is a bit more complicated than necessary. Since we've specified the soldier of interest in the query, we don't really need it in the output, so we can omit p.soldier from the select the group by and the order by parts of the query On the other hand if we omit the e.soldier = 'bill' clause from the original query we can get the timelines for all the soldiers in the army at once (okay maybe that's not a good idea).

So, we've solved the problem (short of a bit of formatting) with just two SQL statements. One to create the view and one to do the query (we can't count the SQL to create and populate the 'engagement' and 'payment' tables as they should already be part of the database).

And if we want an explanation of where the hourly rate comes from we simply use a slightly modified version of the query without the 'sum' and 'group by' bits which includes the information about the characteristics involved:

```
select p.soldier, p.start_date, e.characteristic, e.setting, hourly_rate
from engagement e, payment r, period p
where e.soldier = 'bill' and e.soldier = p.soldier
    and e.start_date <= p.start_date and e.end_date > p.start_date
    and e.setting = r.setting and e.characteristic = r.characteristic
order by p.soldier, p.start_date, e.characteristic
```

Which gives us the full breakdown of the applicable rates per characteristic on the dates defining the timeline:

soldier	start_date	characteristic	setting	hourly_rate
bill	2017-01-01	base	base	1
bill	2017-01-01	combat	no	0
bill	2017-01-01	profession	driver	1
bill	2017-01-01	rank	private	1
bill	2017-01-01	service type	reserve	1
bill	2017-01-01	unit	marines	2
bill	2017-02-01	base	base	1
bill	2017-02-01	combat	yes	5
bill	2017-02-01	profession	driver	1
bill	2017-02-01	rank	private	1
bill	2017-02-01	service type	reserve	1
bill	2017-02-01	unit	marines	2
bill	2017-04-01	base	base	1

bill	2017-04-01	combat	yes	5
bill	2017-04-01	profession	driver	1
bill	2017-04-01	rank	private	1
bill	2017-04-01	service type	active	2
bill	2017-04-01	unit	marines	2
bill	2017-06-01	base	base	1
bill	2017-06-01	combat	yes	5
bill	2017-06-01	profession	fighter	2
bill	2017-06-01	rank	private	1
bill	2017-06-01	service type	active	2
bill	2017-06-01	unit	marines	2
bill	2017-07-01	base	base	1
bill	2017-07-01	combat	yes	5
bill	2017-07-01	profession	fighter	2
bill	2017-07-01	rank	corporal	2
bill	2017-07-01	service type	active	2
bill	2017-07-01	unit	marines	2
bill	2017-09-01	base	base	1
bill	2017-09-01	combat	yes	5
bill	2017-09-01	profession	fighter	2
bill	2017-09-01	rank	corporal	2
bill	2017-09-01	service type	active	2
bill	2017-09-01	unit	paratroopers	2
bill	2017-11-01	base	base	1
bill	2017-11-01	combat	no	0
bill	2017-11-01	profession	fighter	2
bill	2017-11-01	rank	corporal	2
bill	2017-11-01	service type	active	2
bill	2017-11-01	unit	paratroopers	2

6 <u>Validation</u>

The Elephant in the room is validation. A critical assumption is what we've done so far that the engagements for a specific characteristic do not have gaps or overlaps. In a well organised database, there should be constraints to ensure that adding, removing, or updating engagements meet this constraint, but sweeping it under the carpet like this seems to be dodging a key part of the challenge.

Let's bite the bullet, I didn't start off this bit of the work with much confidence, but in turns out we can do a fairly reasonable job. Before we start we must observe that SQL doesn't really do conditional actions, so we what are going to have to do is perform some queries and then 'interpret' the results. We can't get SQL to say: "if the result is like this then there is a problem, otherwise the data is valid". However, the interpretation is not complicated as we'll see – with one exception is basically a matter of saying does this query return any results (which means there is something wrong), so any front end – even the limited logic capabilities of a report generator should be able to cope.

Let's start of by adding add some invalid data to our engagement table:

soldier	characteristic	setting	start date	end date
----	----	----	----	----
carol	rank	private	2017-01-01	2017-06-01
carol	rank	corporal	2017-02-01	2017-06-01
carol	rank	sergeant	2017-06-01	2018-01-01
carol	service type	active	2017-01-01	2017-04-01
carol	service type	reserve	2017-05-01	2017-09-01
carol	service type	active	2017-08-01	2018-01-01
carol	unit	hq	2017-01-01	2017-03-01
carol	unit	marines	2017-04-01	2017-03-01
carol	unit	paratroopers	2017-04-02	2018-01-01
carol	combat	yes	2016-01-01	2017-07-01
carol	combat	no	2017-07-01	2018-01-01
----	----	----	----	----

A manual inspection reveals missing data, and gaps and overlaps. But how easy are they to find with SQL? First off let's check that the engagements make sense in respect of them ending after they start. All we need is a query like:

```
select characteristic, start_date, end_date
from engagement
where soldier = 'carol' and start_date >= end_date
```

(end dates must be strictly after start dates so we use >= not >). This gives the result:

characteristic	start_date	end_date
unit	2017-04-01	2017-03-01

Which tells us there is something wrong with carol's unit engagements.

Next let's check that we have an engagement for each characteristic. If we perform the query:

```
select distinct characteristic
from payment
where characteristic not in (select characteristic from engagement where soldier = 'carol')
```

we are asking how many characteristics are defined (in the payment table) for which the soldier (Carol) does not have an engagement. We get the result:

characteristic
profession

since we have no engagement record telling us what Carol's profession is

If we substitute Bill for Carol in either of these queries no records are returned, so we can conclude that we have (at least) one engagement for every characteristic, and each engagement has a valid duration – which is what we want. Note we could just 'count' have many results we get. If we get 0, we're okay, but then if we get more than 0 it would be nice to know which characteristic is causing the problem.

Next a slightly subtler point – but important in managing the intervals. We want to ensure that a soldier's service starts and ends in a coherent fashion, so all the initial engagements for each characteristic should start on the same date, and all the final engagements should end on the same date. Finding when earliest any engagement starts is pretty easy:

```
select min(start_date) from engagement where soldier = 'carol'
```

as is finding when the earliest of any engagement for a given characteristic:

```
select characteristic, min(start_date) as "start_date"
from engagement
where soldier = 'carol'
group by characteristic
```

The first of these gives a single result (2016-01-01) while the later gives:

characteristic	start_date
base	2017-01-01
combat	2016-01-01
rank	2017-01-01
service type	2017-01-01
unit	2017-01-01

And it is evident that the 'combat' engagement start date is before all the others. For the brave and bold we can actually bundle the two queries into one to give:

```
select distinct e1.characteristic from engagement e1
where (select min(start_date) from engagement where soldier = 'carol')
        < (select min(start_date) from engagement
                where characteristic = e1.characteristic and soldier = 'carol'
                group by characteristic)
order by characteristic
```

It looks horrible, but hopefully it is clear where it comes from looking at the earlier queries. It returns a list of characteristics where the first engagement for the characteristic starts later than the earliest engagement (which is implicitly the start of the service period). For Carol it gives the result:

characteristic
base
rank
service type
unit

Which tells us which characteristics are undefined at the start of the Carol's service. If the query returns no results (as it does for Bill), the first engagement for each characteristic all start on the same date. Swapping min for max, end_date for start_date and > for <, we get the (equivalently convoluted) SQL:

```
select distinct e1.characteristic from engagement e1
where (select max(end_date) from engagement where soldier = 'carol')
          > (select max(end_date) from engagement
                    where characteristic = e1.characteristic and soldier = 'carol'
                    group by characteristic)
order by characteristic
```

which tells us which characteristics (if any) are undefined at the end of Carol's service. The outcome here is:

characteristic
combat
rank
service type
unit

since the end point of her 'base' engagement is 2019-01-01.

Of course, the most challenging part of the validation is looking for overlaps and gaps in the engagements. Overlaps are actually easy to detect - you simply look for (distinct) engagements for the same characteristic where the start date of one is before the end date of the other. For gaps though we find at this point that SQL (or at least my somewhat limited knowledge of it) begins to flounder. However, we can come up with something which though not exactly elegant seems to meet the requirement.

The first step is to exclude a specific boundary case. Evidently if two engagements for the same characteristic have the same start date or the same end date they must overlap. The first possibility is excluded by our choice of the table key, so we only need to check the second (note if we had included both dates in the key we'd need to check the start date as well – one reason not to). The following query does the trick:

```
select e1.characteristic, e1.end_date
from engagement e1, engagement e2
where e1.soldier = 'carol' and e1.soldier = e2.soldier and e1.characteristic = e2.characteristic
          and e1.start_date < e2.start_date and e1.end_date = e2.end_date
```

It's may look a bit intimidating, but basically all this does is look for two distinct engagement (e1 & e2), where the soldier and characteristic match, and the first starts before the second (this is to ensure we only compare different engagements), but the end date is the same. If any are found obviously we have two engagements for the same characteristic with the same end date. The outcome for Carol is:

characteristic	end_date
rank	2017-06-01
unit	2017-03-01

(note the engagement for 'unit' was already flagged as bad, because it ends before it starts). For Bill, since the data for him is good, this query returns no results.

So, if our data has passed all the validations so far, we can assume that for any characteristic, no two engagements start on the same date and no two end on the same date. Now let’s think about how we can match up the end date of one engagement with the start date of another for the same characteristic. Since each start date is unique and each end date is unique, for any engagement, at most one other engagement can have a start date which matches the first engagement’s end date. If are no gaps or overlaps, the engagements for a characteristic form a sequence. The end date of one is the start date of the next, *except* in the case of the last engagement where the end date is the end of the soldier’s service.

From this analysis we deduce if there are no gaps and no overlaps, the number of engagements where we can match the end date with the start date of another is always exactly one less than the total number of engagements.

Now suppose there is a gap. Then the end date of the engagement before the gap does not match the start date of the engagement after the gap, so we cannot match the engagements and the count of matching engagements must be at least two less than the total number of engagements. The analysis is a bit more complicated in the case of an overlap, but the same result follows. Suppose there is an overlap, so the start date of one interval is after the start date of the next. We’ve already excluded the possibility that both have the same end date, so the end date at most one of the two engagements can match the start point of the next engagement, so again we have lost (at least) one engagement from our count of matching engagements, so our total of matching engagements must once again be less than the total we get if there are no overlaps.

So, we can now conclude that if we can count the number of matching engagements for a characteristic, and we find that the number is less than one minus the total number of engagements for the characteristic, there must be a gap or an overlap.

It’s quite easy to count the number of engagements for each characteristic:

```
select characteristic, count(start_date) as count
from engagement where soldier = 'carol'
group by characteristic
```

the clause: `group by e1.characteristic` links to the `count(e1.start_date)` to say we are counting per characteristic. This gives:

characteristic	count
base	1
combat	2
rank	3
service type	3
unit	3

Counting the number of ‘matching’ engagements for each characteristic is somewhat more challenging, but certainly possible, though the SQL is more convoluted. A query which does the trick is:

```
select e1.characteristic, count(e1.start_date) as count
from engagement e1, engagement e2
where e1.soldier = 'carol' and e1.soldier = e2.soldier and e1.characteristic = e2.characteristic
        and e1.start_date < e2.start_date and e1.end_date = e2.start_date
group by e1.characteristic
```

This is constructed in a similar way to the query we used to check for duplicate end dates. Again we look for two distinct engagement (e1 & e2), where the soldier and characteristic match, and the first starts before the second (which ensures they cannot be the same engagement) and again we group by characteristic. However now we want the end date of the first engagement to match the *start date* of the second (rather than the end date). The result this time is:

characteristic	count
combat	1
rank	2

There are a couple of frustrations I have here:

Firstly, if there are no matching engagements for a characteristic, instead of getting a result with a count of zero, I just don't get a result for that characteristic. In this example are no matches for the characteristics 'base', 'service type' or 'unit', so these are implicitly (rather than explicitly) zero engagements which have a matching successor.

Secondly, I don't know any way to compare the outcomes of the two queries directly in SQL. So, we need to summarise the outcome, and interpret it ourselves which is done in this table (not that it would be a major challenge to implement this logic in any simple front end):

characteristic	total count	matching count	difference
base	1	0 (inferred)	1 (correct)
combat	2	1	1 (correct)
rank	3	2	1 (correct – but…)
service type	3	0 (inferred)	3 (overlap or gap)
unit	3	0 (inferred)	3 (overlap or gap)

Note that the number of matching engagements for 'rank' is one less that the total number of engagements for 'rank', even though from the data, Carol is supposedly both a private and a corporal from Feb to May. However, this overlap already causes a validation exception because the end dates of the two engagements are identical. I got caught out by this special case when first building the query, which is when I realised we needed to have a validation check for engagements with matching end dates.

For comparison if we run the same queries for Bill we get:

characteristic	count	matching count	difference
base	1	0 (inferred)	1 (correct)
combat	3	2	1 (correct)
profession	2	1	1 (correct)
rank	2	1	1 (correct)
service type	2	1	1 (correct)

unit	2	1	1 (correct)

7 Conclusion

This seems a complete solution, though certainly not the one I had expected. Generating the timeline turned out to be astonishingly easy, though the validation is rather more clunky.

One can argue that the solution is a bit brittle. Suppose the problem becomes more complicated. Suppose you can be a ‘cook’ and a ‘fighter’ at the same time. Suppose (as obviously will happen) that hourly rates change over time. Well the simple response to this is *any* solution is going to need some modifications to accommodate these kind of changes. In fact, both of these modifications are probably pretty easy to manage in a pure SQL solution, since they really only need an extension to the idea of the ‘events’ which define the time line.

Obviously, all the manipulations done using the SQL can be done easily in a tool capable of reasoning over patterns (e.g. Drools, ODM, Blaze Advisor, or some functional language like Scala or Clojure), but to use them I am first going to have to import the data from a database, so if I can do it directly on the database why bother pulling the data in to manipulate it separately?

What would really start to cause problems if we start having dependencies between characteristics (an obvious one is that if your ‘rank’ is lieutenant, your profession is self-evidently ‘officer’). And the moment one introduces constraints in how you combine payments which moves away from a simple an additive model you are really going to need general-purpose reasoning capabilities than SQL provides. That said our SQL can probably still work out the timeline, even if it can’t work out the appropriate hourly rates within it.

So, considering extensions to the problem only reinforce the main point of I’m trying to make with this solution. You can get a lot of mileage out of the database in getting to an overall decision-making solution. Using the database to do as much work as possible on the more straightforward aspects of the problem, means you have more time to focus on the difficult (i.e. interesting) bits.

8 Appendix – RDBMS, Schema, Data and Queried

8.1 Which database to use

I initially built the solution using Apache Derby as my database, but re-ran everything against Oracle to verify there are no syntactic quirks in the SQL. Apart from some potential issues with the date format for the inserts all the SQL here should work from a command line tool like SQLPlus and its equivalents for DB2, SQL Server, mySQL etc.

On a technical point almost most of the queries used here are parameterised by the name of the soldier of interest. Once one fixes on a specific RDBMS tool, one can simply reduce the SQL to a stored procedure taking the name as a parameter which simplifies it all, but I’ve not gone down to this level as it gets very dependent on the database you are using.

8.2 Schema Definition

The database schema is as follows:

```
create table engagement
(
        soldier         varchar(12)     not null,
        characteristic  varchar(12)     not null,
        setting         varchar(12)     not null,
        start_date      date    not null,
        end_date        date    not null
);

create table payment
(
        characteristic  varchar(12)     not null,
        setting         varchar(12)     not null,
        hourly_rate     integer         not null
);

create view period as select distinct soldier, start_date from engagement;

alter table engagement
  add constraint engagement_pk primary key (soldier, characteristic, start_date);

alter table payment
  add constraint payment_pk primary key (characteristic, setting);

commit;
```

8.3 Test Data

To generate the data used in the description you can execute the following insert statements. Note that in general the default date format may not be correct. Specifically, for an Oracle database you will probably need to modify it with an alter session directive looking like this

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';
```

The insert statements:

```
-- engagement table data
insert into engagement values('alice', 'base', 'base', '2017-01-01', '2018-01-01');
insert into engagement values('alice', 'rank', 'private', '2017-01-01', '2018-01-01');
insert into engagement values('alice', 'profession', 'fighter', '2017-01-01', '2018-01-01');
insert into engagement values('alice', 'service type', 'active', '2017-01-01', '2018-01-01');
insert into engagement values('alice', 'unit', 'paratroopers', '2017-01-01', '2018-01-01');
insert into engagement values('alice', 'combat', 'yes', '2017-01-01', '2018-01-01');
insert into engagement values('bill', 'base', 'base', '2017-01-01', '2018-01-01');
insert into engagement values('bill', 'rank', 'private', '2017-01-01', '2017-07-01');
insert into engagement values('bill', 'rank', 'corporal', '2017-07-01', '2018-01-01');
insert into engagement values('bill', 'profession', 'driver', '2017-01-01', '2017-06-01');
insert into engagement values('bill', 'profession', 'fighter', '2017-06-01', '2018-01-01');
insert into engagement values('bill', 'service type', 'reserve', '2017-01-01', '2017-04-01');
insert into engagement values('bill', 'service type', 'active', '2017-04-01', '2018-01-01');
insert into engagement values('bill', 'unit', 'marines', '2017-01-01', '2017-09-01');
insert into engagement values('bill', 'unit', 'paratroopers', '2017-09-01', '2018-01-01');
insert into engagement values('bill', 'combat', 'no', '2017-01-01', '2017-02-01');
insert into engagement values('bill', 'combat', 'yes', '2017-02-01', '2017-11-01');
insert into engagement values('bill', 'combat', 'no', '2017-11-01', '2018-01-01');
insert into engagement values('carol', 'base', 'base', '2017-01-01', '2019-01-01');
insert into engagement values('carol', 'rank', 'private', '2017-01-01', '2017-06-01');
insert into engagement values('carol', 'rank', 'corporal', '2017-02-01', '2017-06-01');
insert into engagement values('carol', 'rank', 'sergeant', '2017-06-01', '2018-01-01');
insert into engagement values('carol', 'service type', 'active', '2017-01-01', '2017-04-01');
insert into engagement values('carol', 'service type', 'reserve', '2017-05-01', '2017-09-01');
insert into engagement values('carol', 'service type', 'active', '2017-08-01', '2018-01-01');
insert into engagement values('carol', 'unit', 'hq', '2017-01-01', '2017-03-01');
insert into engagement values('carol', 'unit', 'marines', '2017-04-01', '2017-03-01');
insert into engagement values('carol', 'unit', 'paratroopers', '2017-04-02', '2018-01-01');
insert into engagement values('carol', 'combat', 'yes', '2016-01-01', '2017-07-01');
insert into engagement values('carol', 'combat', 'no', '2017-07-01', '2018-01-01');

-- payment table data
insert into payment values('base', 'base', 1);
insert into payment values('base', 'retired', 0);
insert into payment values('rank', 'private', 1);
insert into payment values('rank', 'corporal', 2);
insert into payment values('rank', 'sergeant', 3);
insert into payment values('rank', 'lieutenant', 4);
insert into payment values('rank', 'captain', 5);
insert into payment values('rank', 'retired', 0);
insert into payment values('profession', 'fighter', 2);
insert into payment values('profession', 'driver', 1);
insert into payment values('profession', 'cook', 1);
insert into payment values('profession', 'officer', 3);
insert into payment values('profession', 'retired', 0);
insert into payment values('service type', 'active', 2);
insert into payment values('service type', 'reserve', 1);
insert into payment values('service type', 'retired', 0);
insert into payment values('unit', 'hq', 1);
insert into payment values('unit', 'paratroopers', 2);
insert into payment values('unit', 'marines', 2);
insert into payment values('unit', 'infantry', 2);
insert into payment values('unit', 'retired', 0);
insert into payment values('combat', 'yes', 5);
insert into payment values('combat', 'no', 0);
insert into payment values('combat', 'retired', 0);
commit;
```

8.4 The Queries

The queries are as follows:

```
-- EXAMPLE QUERIES LEADING TO A SOLUTION
-- the following query gives the engagements and associated pay rates for a particular soldier
-- (Bill) on the 4th of July 2017 (Note above comments on date formats)
select soldier, e.characteristic, e.setting, hourly_rate
from engagement e, payment r
where soldier= 'bill' and start_date <= '2017-07-04' and end_date > '2017-07-04'
and e.setting = r.setting and e.characteristic = r.characteristic order by e.characteristic;

-- the following query gives the total hourly rate for a particular soldier
-- (Bill) on the 4th of July 2017
select soldier, sum(hourly_rate) as "hourly rate"
from engagement e, payment r
where soldier= 'bill' and start_date <= '2017-07-04' and end_date > '2017-07-04'
and e.setting = r.setting and e.characteristic = r.characteristic
group by soldier;

-- the following query identifies all the engagements for a particular soldier
-- (Bill) during his service period. It uses the view 'period' rather than the
-- table engagement
select distinct start_date from period where soldier = 'bill' order by start_date;

-- ************************End of examples *******************************

-- QUERIES WHICH FORM THE SOLUTION TO GENERATING A TIMELINE

-- the following query produces a timeline of the hourly pay for a particular soldier
-- (Bill) during his service period.
select p.soldier, p.start_date, sum(hourly_rate) as "hourly rate"
from engagement e, payment r, period p
where e.soldier = 'bill' and e.soldier = p.soldier
and e.start_date <= p.start_date and e.end_date > p.start_date
and e.setting = r.setting and e.characteristic = r.characteristic
group by p.soldier, p.start_date order by p.soldier, p.start_date;

-- the following query produces a timeline showing way in which the hourly pay for
-- a particular soldier (Bill) is made up during his service period.
select p.soldier, p.start_date, e.characteristic, e.setting, hourly_rate
from engagement e, payment r, period p
where e.soldier = 'bill' and e.soldier = p.soldier
and e.start_date <= p.start_date and e.end_date > p.start_date
and e.setting = r.setting and e.characteristic = r.characteristic
order by p.soldier, p.start_date, e.characteristic;

-- ********************End of Timeline solution***************************

-- QUERIES DESIGNED TO VALIDATE THE CORRECTNESS OF THE INPUT DATA
-- ************************CHECKS FOR CAROL************************
-- CHECK 1: the following query checks engagements for a particular soldier
-- (Carol) start before they end
select characteristic, start_date, end_date from engagement
where soldier = 'carol' and start_date >= end_date;

-- CHECK 2: the following query checks engagements for a particular soldier
-- (Carol) to ensure all characteristics are defined
select distinct characteristic from payment
where characteristic not in (select characteristic from engagement where soldier = 'carol');
```

```
-- CHECK 3: the following query checks the initial engagements for a particular soldier
-- (Carol) all start at the same time
select distinct e1.characteristic from engagement e1
where (select min(start_date) from engagement where soldier = 'carol') <
 (select min(start_date) from engagement
   where characteristic = e1.characteristic and soldier = 'carol' group by characteristic)
order by characteristic;

-- CHECK 4: the following query checks the final engagements for a particular soldier
-- (Carol) all end at the same time
select distinct e1.characteristic from engagement e1
where (select max(end_date) from engagement where soldier = 'carol') >
 (select max(end_date) from engagement
   where characteristic = e1.characteristic and soldier = 'carol' group by characteristic)
order by characteristic;

-- CHECK 5: the following query checks no two engagements for a particular
-- characteristic and a particular soldier (Carol) end at the same time
select e1.characteristic, e1.end_date from engagement e1, engagement e2
where e1.soldier = 'carol' and e1.soldier = e2.soldier and e1.characteristic = e2.characteristic
and e1.start_date < e2.start_date and e1.end_date = e2.end_date;

-- CHECK 6 part 1:  the following query counts the number of engagements for a
-- particular characteristic and a particular soldier (Carol)
select characteristic, count(start_date) as count from engagement
where soldier = 'carol' group by characteristic;

-- CHECK 6 part 2: the following query counts the number of 'matched' engagements
-- for a particular characteristic and a particular soldier (Carol)
select e1.characteristic, count(e1.start_date) as count
from engagement e1, engagement e2
where e1.soldier = 'carol' and e1.soldier = e2.soldier and e1.characteristic = e2.characteristic
and e1.start_date < e2.start_date and e1.end_date = e2.start_date
group by e1.characteristic;
-- ********************End of Checks for Carol****************************

-- ************************CHECKS FOR BILL*************************
-- CHECK 1: the following query checks engagements for a particular soldier
-- (Bill) start before they end
select characteristic, start_date, end_date from engagement
where soldier = 'bill' and start_date >= end_date;

-- CHECK 2: the following query checks engagements for a particular soldier
-- (Bill) to ensure all characteristics are defined
select distinct characteristic from payment
where characteristic not in (select characteristic from engagement where soldier = 'bill');

-- CHECK 3: the following query checks the initial engagements for a particular soldier
-- (Bill) all start at the same time
select distinct e1.characteristic from engagement e1
where (select min(start_date) from engagement where soldier = 'bill') <
 (select min(start_date) from engagement
   where characteristic = e1.characteristic and soldier = 'bill' group by characteristic)
order by characteristic;
```

```
-- CHECK 4: the following query checks the final engagements for a particular soldier
-- (Bill) all end at the same time
select distinct e1.characteristic from engagement e1
where (select max(end_date) from engagement where soldier = 'bill') >
 (select max(end_date) from engagement
   where characteristic = e1.characteristic and soldier = 'bill' group by characteristic)
order by characteristic;

-- CHECK 5: the following query checks no two engagements for a particular
-- characteristic and a particular soldier (Bill) end at the same time
select e1.characteristic, e1.end_date from engagement e1, engagement e2
where e1.soldier = 'bill' and e1.soldier = e2.soldier and e1.characteristic = e2.characteristic
and e1.start_date < e2.start_date and e1.end_date = e2.end_date;

-- CHECK 6 part 1:  the following query counts the number of engagements for a
-- particular characteristic and a particular soldier (Bill)
select characteristic, count(start_date) as count from engagement
where soldier = 'bill' group by characteristic;

-- CHECK 6 part 2: the following query counts the number of 'matched' engagements
-- for a particular characteristic and a particular soldier (Bill)
select e1.characteristic, count(e1.start_date) as count
from engagement e1, engagement e2
where e1.soldier = 'bill' and e1.soldier = e2.soldier and e1.characteristic = e2.characteristic
and e1.start_date < e2.start_date and e1.end_date = e2.start_date
group by e1.characteristic;
-- ********************End of Checks for Bill ****************************
-- ********************End of Validation Logic****************************
```