This would generate the following intervals:

0000 hours to 2359 hours on 02/11/86
2100 hours to 2330 hours on 04/11/86
0900 hours to 1454 hours on 08/11/86
0900 hours to 1454 hours on 10/11/86
1200 hours to 1945 hours on 23/11/86
1200 hours to 1945 hours on 24/11/86
1200 hours to 1945 hours on 25/11/86
1200 hours to 1945 hours on 26/11/86

These examples show that, by using this structure, it is possible to respond to any user requirement for the processing of a chronologically ordered database.

3. Optimising search times.

It has been seen that a user request may include extensive time frame specifications against which a database is to be searched Where a database resides upon a fast, direct access medium, such as disc, the complexity of a time frame specification will not greatly detract from reasonable response times if the database is indexed by date or a binary search is carried out to provide start and end references. In these instances direct access can be made to the appropriate parts of the database as defined by each element of the time frame specification.

However, where such a database is not indexed or is stored upon a medium such as magnetic tape then the only way to comply with the time frame specification is to scan the data serially from the beginning . It is therefore vital, if search time is to be optimal, that the database is traversed only once.

The solution to ensuring an optimal search time is straight forward . It is sufficient to ensure that any given time frame specification, when expanded, is sorted by date and start time This will allow a database, which is itself chronologically ordered, to be traversed only once.

By sorting the expanded <TIME FRAME LIST>, prior to searching, we will not only provide the best possible response time but will also assist users by allowing mis97ordered (TIME FRAME>s to be specified . Furthermore, retrieved records will be encountered chronologically and so may be presented time ordered without further processing or sorting of the retrieval list.

For databases held on magnetic tape the time saved will be considerable where <TIME FRAME LIST>s are extensive.

Example:
Consider a chronologically order database residing on a single reel of magnetic tape

<TIME FRAME LIST>::= 23/O6/87{1200 1230},1,25/06/87
0l/06/87{1045 1700}
14/06/87

The expanded list would be:

from 1200 hours to 1230 hours on 23/06/87
from 1200 hours to 1230 hours on 24/06/87
from 1200 hours to 1230 hours on 25/06/87
from 1045 hours to 1700 hours on 01/06/87
from 0000 hours to 2359 hours on 14/06/87

If this expanded list of intervals was used without being sorted, by date/start time, then the tape would need to be rewound after processing the records for the 25th back to the beginning so that the file containing records for the 1st could be accessed . Sorting this list prior to the run would ensure that the tape would only need to be scanned in a forward motion only.

4. General method of application.

Consider of collection, D, of n records:

$D=\{R_1,R_2,\ldots,R_n\}$

held upon some suitable storage medium, where each R_i, for i=l,2,...,n ,contains a number of data fields against which a <SELECTION CRITERIA> may be applied . Let each R_i also contain two additional fields, $<d_i>$ and $<t_j>$, which identify the precise data and time that R_i is captured by a computer system.

With a suitable choice for the resolution of data/time stamping it is possible to provide uniqueness, irrespective of the processing power of a host computer, such that:

$R_i < R_i+1$, for i=1,2,...,n, where the symbol "<" is taken to mean chronologically before.

Now partition the set of records, D, into subsets, thus:

$D=\{F_1,F_2,\ldots,F_m\}$, where m=number of files covering the collection of records such that:

$F_i=\{R(<d_i>,<t_j>\}$, for i=1,2,...,m
and j=1,2,...,mi where m~ is the number of records in F_i.

Here, each F_i subset simply contains the collection of records which are date/time stamped for the same date . For convenience we shall refer to each sub97set, F_i, as a file of records for the same date.

Given the above database, D, it is required to perform a search given:

<TIME FRAME LIST>
<SELECTION CRITERIA>

The first task to be undertaken is to determine precisely the complete scope of the database search by expanding each <TIME FRAME> within the given <TIME FRAME LIST> Once this has been done it is necessary to sort the expanded list, chronologically, before using it to scan the database and applying the <SELECTION CRITERIA>

4.1. Expansion of the <TIME FRAME LIST>.

The following routine will expand a <TIME FRAME LIST> which complies with the syntax defined at section 2 and the subsequent derivatives at section 2.1.

Define the following variables:

let sd represent START DATE
ed END DATE
st START TIME
et END TIME
fr FREQUENCY
d GENERAL DATE VARIABLE

Further define a table, T, of elements, arbitrarily large enough to hold a set of time intervals generated from the expansion process, such that each entry of the table is defined with three fields - one date field, 'date', and two time fields, 'starttime' and 'endtime'. Each table entry may be access by a suitable index address, thus:

date[1], starttime[1], endtime[1], ...entry 1 values
date[2], starttime[2], endtime[2], ...entry 2 values

:
:
date[n], starttime[n] , endtime[n]. ...entry n values.

```
01: WHILE <TIME FRAME LIST> NOT empty DO
02:    initialise: sd=ed=fr=0; st=et=-1
03:    read date(sd)
04:    IF NOT end of line THEN
05:       read times(st,et)
06:       IF NOT end of line THEN
07:          read frequency(fr)
08:          IF NOT end of line THEN
09:             read date(ed)
10:             IF NOT end of line THEN
11:                read to end of current line
12:             ENDIF
13:          ENDIF
14:       ENDIF
15:    ENDIF
16:    IF st=-1 THEN
17:       st=0000
18:       et=2359
19:    ENDIF
20:    IF ed=0 THEN
21:       ed=sd + fr
22:    ENDIF
23:    IF fr=0 THEN
24:       fr=1
25:    ENDIF
26:    FOR d=sd STEP fr UNTIL ed DO
27:       insert d, st and et into next free entry of table T
28:    ENDDO
29: ENDDO
```

This routine does not attempt to perform any validation of the values read . Appropriate validation considerations would normally be applied between lines 22 and 23.

4.2. Sorting and Searching.

The result of applying the routine defined under 4.1. against a <TIME FRAME LIST> is to produce a table containing discrete dates and times, which may well not be in strict chronological order. Therefore, before the table can be used effectively, it must sorted . Any sort routine may be used to accomplish the ordering of the table, providing the entries are sorted using the date and start time fields. Note that the end time field is not used for the ordering process.

Once sorted it is possible to search out database, D, in an optimum time . The following routine provides a general method for carrying out a database search using the expanded <TIME FRAME LIST> produced by the expansion routine above.

Define the following variables:

let ne represent the number of entries in the table T,
e represent a general working variable,
r represent a general working variable

```
01: e=1
02: ne=number of entries in table T
03: WHILE e <= ne DO
```

```
04:   Open file containing records for date[e]
05:   IF file opened = TRUE THEN
06:      r=1
07:      read record (r)
08:      WHILE NOT end of file DO
09:         IF R(<date[e]>,<t_r>) >= starttime[e] AND R(<date[e]>,<t_r>) <= endtime[e] THEN
10:            Apply <selection criteria>
11:            IF match = TRUE THEN
12:               Add R(<date[e]>,<t_r>) to retrieval list
13:            ENDIF
14:         ENDIF
15:         r=r+1
16:      ENDDO
17:   ENDIF
18:   e+e+1
19:ENDDO
```

Ron Bentley.
Mid-1980's research